Attack-Class-Based Analysis of Intrusion Detection Systems

Dominique Alessandri

May 2004

Ph.D. Thesis

University of Newcastle upon Tyne

School of Computing Science

Abstract

Designers of intrusion detection systems are often faced with the problem that their design fails to meet the specification because the actual implementation is not able to detect attacks as required. This work aims at addressing such shortcomings at an early stage of the design process. The proposed method provides guidance to intrusion detection systems designers by predicting whether or not a given design will be able to detect certain classes of attacks. Our method achieves this by introducing a classification of attacks and a description framework for intrusion detection systems. The attack classification and the description framework are defined at a common level of abstraction, and thereby form the basis for our analysis method, which determines the attack classes that a given intrusion detection system design can detect. Intrusion detection system designers can use these results to determine where the design meets the specification and where it does not. These insights facilitate a more systematic and effective design process because they can be gained at an early stage of the design process without the need of actually implementing the design. Finally, we show how our approach to intrusion detection system design analysis can be validated and how the analysis results can be used for further applications such as guiding the design of intrusion detection architectures that combine diverse intrusion detection systems.

Acknowledgements

First of all, I thank my wife, Sandra, for her tireless and invaluable help and encouragement and her patience. Without her continuous support, I would not have been able to write this thesis. Similarly I thank my family and all my friends for providing me with their tremendous support. Moreover I gratefully acknowledge the excellent scientific advice, assistance and motivation provided by my manager and mentor, Andreas Wespi, and my supervisor Richard (Dick) Snow. In the same context I also thank my initial supervisor, Brian Randell, my initial mentor, Marc Dacier, and my examiners, Roy Maxion and Robert Stroud. Last but not least, I thank my IBM friends and colleagues Magnus Almgren, Birgit Baum-Waidner, Charlotte Bolliger, Hervé Debar, Douglas Dykeman, Klaus Julisch, Raffael Marty, James Riordan, Christian Rohner, Morton Swimmer, Axel Tanner, Andreas Tscharner, Candid Wüest, Diego Zamboni, and all the people in the IBM Zurich Research Laboratory (ZRL) for their highly valuable and generous scientific and non-scientific support. I greatly enjoyed working at the ZRL and in particular in the GSAL team (Global Security Analysis Lab).

This research was conducted at the IBM Zurich Research Laboratory in collaboration with the Newcastle University upon Tyne. It contributed to and was partially funded by the MAFTIA (Malicious- and Accidental-Fault Tolerant Internet Applications) research project. MAFTIA was funded by the European Commission (EC) under contract IST 1999-11583.

Table of contents

Chapter 1 Introduction

In recent years, an increasing number of intrusion detection systems (IDSs) have become available [Sobire98]. This has been driven by numerous developments, including the growing e-business paradigm, the increasing interconnection of critical infrastructure elements, and the growing number of computer security incidents [CIN0799, Gross97, Howard97, Kumar95, LSMTTF98, Neuman98b, NeuPar89]. These incidents highlight the increasing need for organizations to protect their networks against adversaries [Sundar96]. The subject of protecting networks and making them secure and reliable has been addressed in many publications that have analyzed the problems and made pertinent recommendations [BeGlRa98, Neuman98]. Intrusion detection (ID) is widely regarded as being part of the solution for protecting today's networks.

IDSs are used to improve system security by detecting attacks and intrusions. However, although they have been under development for many years, installing and configuring them to provide the intended service still is a major undertaking and may involve an unacceptable effort. There are numerous reasons for these difficulties. The facts that

1. IDSs tend to generate excessive numbers (99% and more) of false alarms [Julisc00, Julisc01] and that
2. they often fail to meet their specification by not detecting attacks they should detect [LFGHKM00, LHFKD00]

are two of the most frequent and important weaknesses of IDSs. These weaknesses are in part caused by a problem inherent to intrusion detection: the difficulty of determining whether the *intent* of activities that IDSs observe is malicious or benign, i.e., whether or not an alarm should be generated. There exists no generally applicable technique to make such distinctions, not least because many attacks make legitimate use of features provided by the target system. Instead highly specific and, at times, ad-hoc techniques are required for rating activities as being either malicious or benign. As a consequence special care has to be exercised during the design process of IDSs, in particular when ad-hoc techniques are used. Such IDSs often suffer from the weaknesses described above.

1.1 Motivation

The design of an IDS is a challenging task because besides requirements such as the coverage of certain types of attacks, other limiting factors have also to be taken into account, such as the following:

- Feature extraction: The output of an IDS can only be as accurate as its input [MWSKHH90]. For detecting a given type of attack the IDS needs to be capable of making the appropriate observations, i.e., it needs access to data that is relevant for detecting the attack. Thus, the IDS designer needs to identify the data required and, most importantly, a data source that provides this data in an appropriate format. This is a non-trivial task because often the IDS designer is not

able to influence the properties of the data source and because different detection approaches may rely on different data sets for their operation. Moreover commonly available data sources may require significant pre-processing of the data they provide or do not provide all the data needed.

- Trade-off between costs and functionality: The use of a given data source and detection approach comes with a certain cost. Their use may, for instance, cause an unacceptable performance degradation, or require impractical modifications of the surveyed system. In order to limit such side-effects, IDSs often use entirely different or simplified solutions that may result in reduced performance of the IDS. In fact in many cases such alternative solutions yield IDSs that are not able to detect certain types of obfuscated, i.e., slightly modified, attacks [PtaNew98] or that generate numerous false alarms.
- Test data: Whenever an IDS is tested realistic input data is required [Maxion98, McHugh00, McHugh00b]. This is an issue because no two environments for which an IDS is to be designed are identical and because IDSs may make use of environment-dependent optimizations and ad-hoc solutions. Thus, there is no single general-purpose data set that can be used for testing.

The following example illustrates most of the issues described from the view of an IDS that monitors network traffic, i.e., a network-based IDS.

Example: *IP PDUs[1] may be fragmented by a router if they exceed the size that the link connecting the router to the destination network is able to transmit. Thus, the fragmentation of IP PDUs is a legitimate and necessary functionality provided by the IP protocol. However, this functionality can be used by adversaries to obfuscate attacks such as http-related attacks that operate at a higher level of the protocol stack. An IDS observing an IP PDU that has been fragmented has to reassemble its fragments before the content can be analyzed properly (feature extraction issue). Particularly in heavily loaded networks this may impose a significant load on the IDS, not least because fragments may be sent out of order. Because of this difficulty some IDS designers choose to not provide the functionality of reassembling IP fragments or at least give the user the option of disabling this functionality (cost—functionality trade-off). Moreover some IDSs generate alarms whenever a fragmented IP PDU is observed. However, because fragmented IP PDUs may occur in entirely legitimate traffic, such attack signatures generally result in excessive numbers of false alarms. As documented by Marty [Marty02, p. 66] some IDS designers address this issue by having their IDSs only generate alarms whenever the size of a IP PDU fragment is smaller than a chosen value. Such ad-hoc solutions may reduce the number false alarms, but do not eliminate them. Thus any such alarm has to be interpreted with care and, assuming it is not a false alarm, conveys only very little information about the actual attack (test data issue).*

[1] PDU: Protocol data unit; a packet

1.2 Goal

Recent evaluations [LFGHKM00, LHFKD00, Maxion98, Walder01a] of actual IDS implementations have clearly revealed that currently available IDSs suffer from various weaknesses such as the ones described thus far. This insight motivated this work, which aims to support the designers of IDSs in their task. The requirement is to provide a method and tool that predicts the potential of a given IDS design proposal to detect given types, i.e., classes, of attacks without the need to actually implement the proposed IDS design. Such an approach limits the effort required to a practical level because it saves us from having to conduct an impractical number of experiments. We therefore set the following goal for this work:

Goal: *Provide guidance to IDS designers by predicting the detection capabilities of intrusion detection systems.*

1.3 Approach

In order to achieve the above goal we propose an approach in which we claim it can be predicted whether or not an IDS design is able to detect a given class of attacks. As these predictions have to be made at the level of attack classes, we propose an approach that performs a combined analysis of descriptions of attack classes and IDS designs, i.e., without the need to evaluate IDS implementations. In the beginning the two tracks of analysis, i.e., the description of attack classes the description of IDS designs, are split to ensure their originality.

For the first track we introduce an *attack classification* that classifies attacks according to their externally observable characteristics, i.e., according to the attack characteristics that are observable by IDSs, humans etc. We determine attack characteristics using a generic system model that enables the identification of system components and the interaction among these components when the system is under attack. So, any two attacks belong to the same attack class if they share the same set of externally observable characteristics. Attack classes are therefore defined by a set of externally observable characteristics that is shared by all attacks belonging to a given class.

The identified attack classes are then described in terms of IDS characteristics an IDS must have in order to be capable of analyzing a given class of attacks, i.e., analyzing attacks belonging to any such class. Note also that these descriptions must reflect all conceivable approaches an IDS could take to analyze such attacks. The fact that attack classes are described using IDS characteristics as they are defined by the IDS description framework significantly simplifies the actual IDS analysis and ensures consistency across the entire work.

The second track describes IDSs using the IDS characteristics just mentioned. These characteristics are derived from IDS classifications and are used to describe the manner in which IDSs gather and analyze information for signs of attacks.

At the core of our approach we use a method that determines the manner in which an IDS analyzes a given class of attacks. As input to this analysis we use descriptions of IDSs and descriptions of attack classes. In a second step further analysis is performed and the actual output generated. The analysis results for each attack class may be different and consist of a set of *generalized alarms* that the analyzed IDS has the potential of generating. Generalized alarms are defined by the analysis techniques and the data an IDS uses to detect attacks that belong to the considered attack class, i.e., they reflect the semantics of the alarms that the actual implementation of the analyzed IDS design potentially generates. In order to broaden the scope of these results, the analysis step also includes the automated identification of classes of *attack variants*, i.e., attacks that have been obfuscated in some way, and the analysis of IDSs for these additional attack classes. Figure 1 provides an overview of the IDS analysis process. As part of this work, we have created a prototype implementation of this process that we named RIDAX (Rule-based Intrusion Detection system Analyzer and eXaminer).

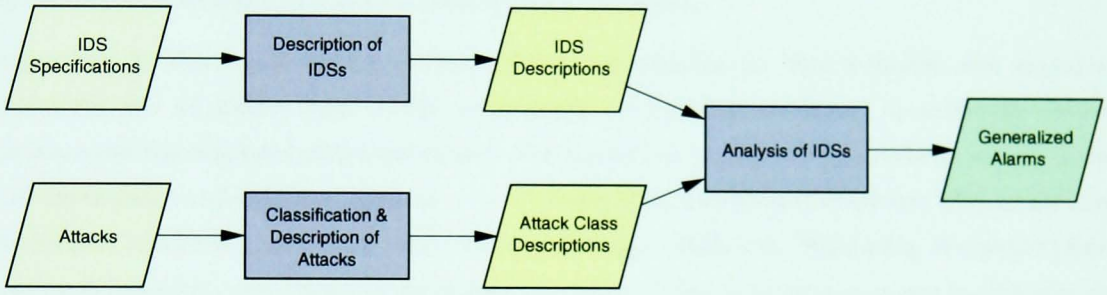


Figure 1—Overview of the IDS analysis process

The IDS analysis results are meant to feed into an iterative design process that aims at identifying an IDS design proposal that meets the IDS specification with regard to the classes of attacks the IDS has to be able to detect. However, it has to be emphasized that our approach uses descriptions of IDSs and classes of attacks, i.e., that it operates at a conceptual level. The use of such class-level results is advantageous for designers of IDSs, because they are not required to keep track of the numerous individual attacks that are newly discovered on a daily basis. Instead, the designers can focus on developing and optimizing the generic detection capabilities of IDSs, i.e., on their detection capabilities with regard to entire classes of attacks. However, the actual implementation of an IDS will only generate the results predicted by our approach if the IDS is configured accordingly, if all the attack signatures required are available. Also the IDS must not suffer from implementation flaws or use ad-hoc techniques that are not reflected by its description. In practice these limitations are merely of limited significance because it is in the interest of IDS implementers and users to make best use of the capabilities offered by IDSs. This means, for instance, that IDS users usually update attack signatures regularly—similar to the manner signatures of anti-virus systems are kept up-to-date.

The fact that under certain circumstances an IDS may behave differently than predicted also has implications on the validation of this work. We will discuss these difficulties and show that for practical reasons only limited validation can be provided. A limited validation can be achieved by providing evidence that for a representative number of attacks the analysis results produced by our approach reflect

the results that actual IDS implementations produce. We will outline such a validation approach by comparing a small number of results produced by our approach with the results IDS implementations produce.

1.4 Novelty of our approach

Numerous surveys and classifications of IDSs and attacks have been proposed. IDS surveys such as the one by Jackson [Jackso99] and those by many others [Amoro99, Axelss99b, EsSaPi95, Lunt88, MWSKHH90] generally focus on the detection capabilities of IDSs and less so on the IDS characteristics that are necessary to achieve these capabilities. IDS classifications such as the ones by ones Debar *et al.* [DeDaWe00, DeDaWe99] or the one by Axelsson [Axelss00] focus on the internal characteristics of IDSs. However, because they do not aim at the automated analysis of IDS detection capabilities, they operate at a level of abstraction that for our purposes is too coarse.

Approaches to IDS benchmarking evaluate IDS implementations for their behavior with regard to predefined sets of attacks. Most of the experiments are generally conducted in artificially created environments that simulate background activity. The best known work in this area is the so-called Lincoln Lab experiment conducted by Lippmann *et al.* [LFGHKM00, LHFKD00]. However, Maxion and Tan [Maxion98, MaxTan00] as well as many others [DCWMS99, GafUlv01, Walder01a, Walder01b] have also made important contributions to the evaluation of IDSs. It has to be mentioned that in particular the Lincoln Lab experiment has been criticized [McHugh00, McHugh00b] as having numerous shortcomings that in part also apply to the other approaches. From our perspective, the results provided by IDS evaluations are too low-level because the benchmarks are carried out using a selection of specific attacks rather than at the level of attack classes.

Note that IDS evaluations could be used to determine class-level results if one conducted a substantial number of tests to determine the IDS behavior for each equivalence class of attacks. However, such an approach requires a large number of tests that would exceed the number of tests involved in the Lincoln Lab experiments by far. Moreover benchmarks obviously require the IDSs to be implemented, which is a second requirement that we attempt to circumvent with our approach. Figure 2 provides a comparison of our approach (above the dashed line) to IDS benchmarking (below the dashed line).

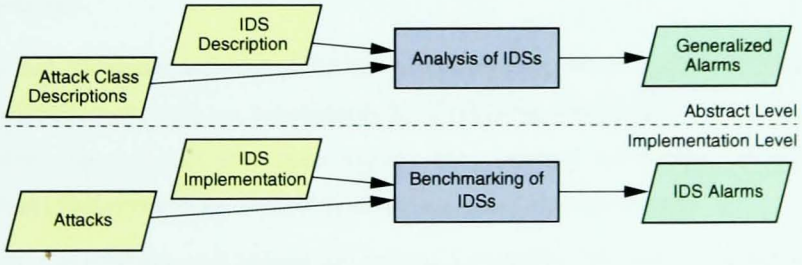


Figure 2—Comparison of our approach with IDS benchmarking

In the area of attack classifications there exists a substantial amount of prior work. Important contributions have been made by Neuman and Parker [Neuman95, NeuPar89], Kumar [Kumar95] and

many others [Cohen95, Howard97, LinJon97]. In many cases classifications of attacks are tightly coupled with the classification of vulnerabilities [Howard97, Krsul98]. When studying these classifications, one finds many of them to be quite different from each other because each of them pursues a different goal. Regarding the goal pursued by our approach, the attack characteristics that might potentially be observable by an IDS are the most important aspects to be taken into account.

Seeking a suitable classification, we found that most existing attack classifications were aimed at different goals or did not capture the attacks in appropriate detail and were therefore not well-suited for our purposes. Hence, we found it necessary to develop our own classification and description schemes for IDSs and attacks. However, in the following we also use, for instance, concepts developed by Debar *et al.* [DeDaWe00, DeDaWe99] as a basis for a part of our IDS description framework.

Finally note that it has been argued [McHugh00, McHugh00b] that one of the major weaknesses of IDS benchmarking approaches is the manner in which the attacks used for testing purposes are selected. For our approach this issue is only of limited importance as the analysis of IDSs is conducted at the level of attack classes rather than at the level of specific attacks. However, when it comes to providing relevant examples and a set of predefined attack class descriptions, we are in the advantageous situation of being able to make use of existing work to select attack classes that are known to be relevant. While conducting the work presented here, we have built and maintained IBM's security database (VulDa) [DacAle99], which we were able to use for categorizing attacks with regard to the observable aspects of attacks. We used the statistical results obtained from this categorization to select a number of relevant classes of attacks that we then used in the larger context of this work. The availability of such statistical data that reveals the most popular attack categories may also help IDS designers when specifying the list of attack classes that their IDSs should be able to detect.

1.5 Contributions

The method and tool presented in this work supports designers of IDSs by providing them with the capability of verifying whether their design meets the requirements before the actual system is implemented. Moreover the results provided by our approach may serve as the foundation for other ID-related research such as the assessment of IDSs and combinations thereof. In detail we make the following contributions:

1. A novel and systematic scheme to describe IDSs concisely: So far IDSs have been characterized and described based only on benchmarks [LFGHKM00, LHFKD00] and product descriptions [Jackso99]. Our scheme also goes further than existing taxonomies of IDSs [Axelss00, DeDaWe00, DeDaWe99] by describing IDSs in a much finer granularity.
2. A generic classification and description scheme for attacks: We created an attack classification and description scheme that is based on criteria that are directly relevant to the ways IDSs analyze their observations for signs of security threats. The attack and IDS description schemes make use of a common basis that ensures consistency and simplifies the analysis of IDS designs.

3. A novel approach to IDS analysis: Making use of the first two items, our approach [Alessa00] systematically predicts the attack classes that a given IDS design is able to detect and how these classes would be reported. The results obtained provide guidance to IDS designers by identifying strengths and weaknesses of IDS designs and, most importantly, by identifying where the design meets the specification and where it does not. Moreover we have outlined and successfully illustrated the manner in which our approach to IDS analysis can be validated—thereby implicitly also validating the above two points.

In addition we categorized attacks based on criteria that are relevant for their detection using IBM's security database VulDa [DacAle99]. The statistical results obtained served to identify representative attacks and attack classes for which we created descriptions in the course of this work. The statistics obtained support IDS designers in the specification of the attack classes that the IDS to be created needs to be able to detect. The categorization has become an integral part of VulDa, which is used on a daily basis by IBM's security professionals.

Finally note that the IDS analysis results obtained in the last item form the foundation for further ID research, such as for developing novel approaches to the assessment of IDSs and combinations thereof.

1.6 Outline

In the following chapters we first develop the necessary foundations for this work, and then explain and develop the approach introduced in Section 1.3 in detail. The foundations include a discussion of related work, and also introduce fundamental concepts and terminology, which is done in Chapter 2.

Then, in Chapter 3, we provide a more detailed overview of the entire approach that has been developed in this thesis.

In Chapter 4, we describe the attack categorization scheme that we apply to attacks taken from VulDa. VulDa is IBM's security database, which was developed by the author in parallel with the work described here (see Appendix A). The scheme provides a categorization of attacks that enables us to identify the most relevant attacks, which we then use to identify the attack classes used during the IDS analysis process. Furthermore we develop the generic concept of IDS scopes. These simplify the combination of our work items and support us in ensuring consistency.

In Chapter 5 we develop the IDS description framework. It makes use of the IDS scope concept developed in Chapter 4, which is why it is placed after the attack categorization chapter.

Chapter 6 defines the manner in which attack classes are to be described by means of IDS characteristics, and thereby combines the results obtained in Chapters 4 and 5.

The results of these chapters are then brought together in Chapter 7, where we develop the IDS analysis process, which resulted in the implementation of RIDAX. The latter is also described in this context. Finally, we outline and discuss an approach for validating IDS analysis results as, for instance, generated by RIDAX and provide a set of illustrative examples.

In Chapter 8 we provide an extensive example for a further application of the IDS analysis results as they are produced by RIDAX. The example explores a possibility of assessing individual IDSs and combinations thereof. In order to achieve this we first define suitable assessment metrics. Then we discuss results obtained by using an extended version of RIDAX to calculate these metrics. These results are based on the analysis of five configurations of three fundamentally different IDSs, and assess every possible combination of them. The results clearly show that not every combination of IDSs is beneficial and results in improved completeness and utility of the ID architecture design considered.

Chapter 9 concludes this work and provides a critical discussion as well as an outlook to future work that we envisage to pursue.

Figure 3 provides an overview of the chapters herein and indicates their main dependencies. The square boxes represent processes or tasks performed, while the rounded boxes represent concepts and results produced.

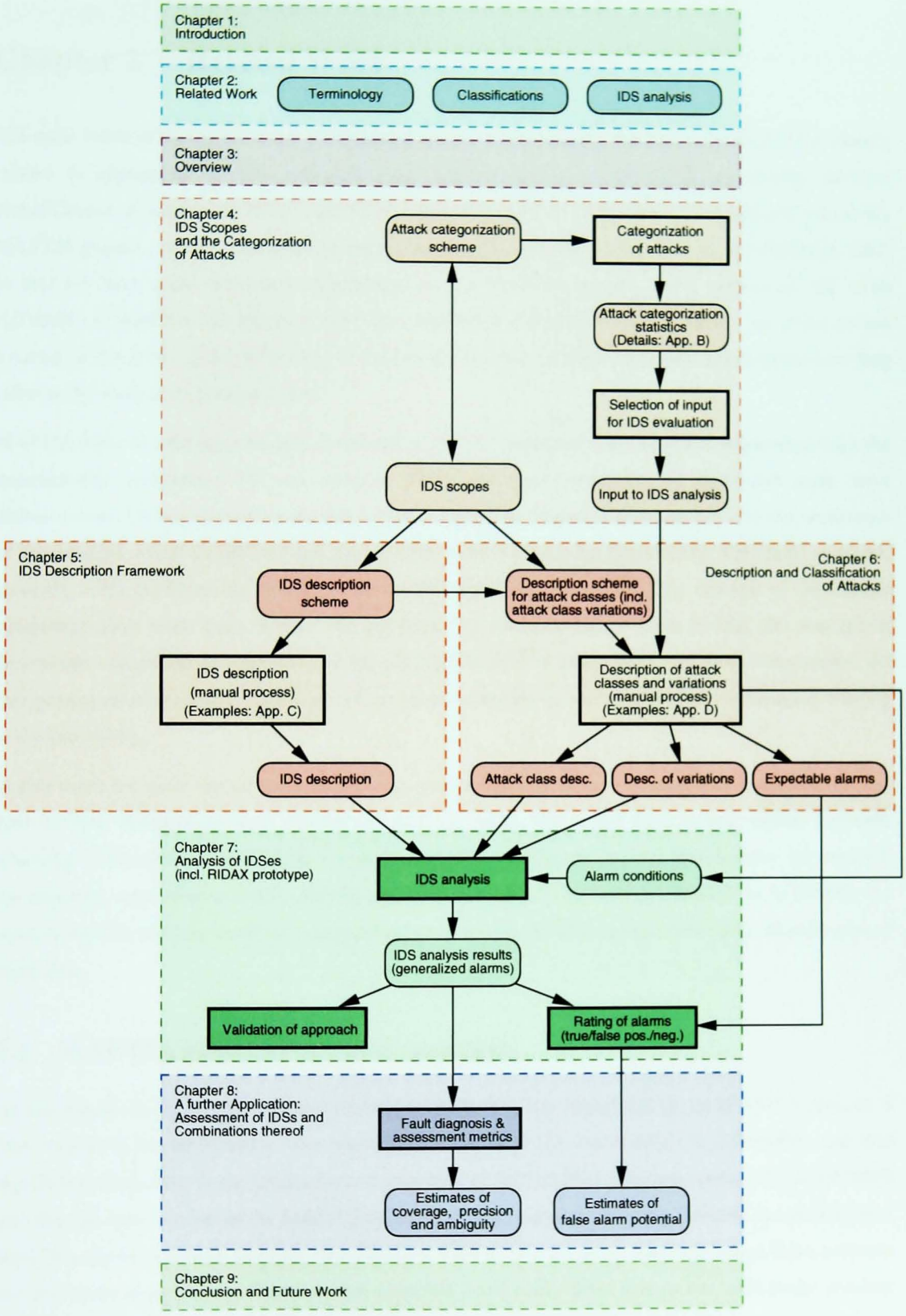


Figure 3—Overview of thesis chapters

Chapter 2 Related Work

The main focus of this work is the investigation of how IDSs analyze attacks. Accordingly it is closely related to approaches to IDS evaluation, but also to ID in general. This specifically includes classifications of attacks and IDSs. Last but not least, parts of this work relate in a significant way to the MAFTIA project, which makes use of concepts and terminology originating from the dependability field. In fact we have made important contributions to the MAFTIA project in the context of this work [D2Maf01, D3Maf01]. One aspect to which we contributed is the MAFTIA terminology, of which we use a subset in this work. In the following we provide an overview of these items and also explain how they relate to the work to be presented here.

MAFTIA aims at unifying concepts developed in the ID community and concepts originating from the dependability community. The two research fields, although overlapping in significant areas, have different roots. ID has its roots in the early seventies [Anders72] and gained impetus with the occurrence of the Internet worm [Spaffo88] and the seminal work by Denning [Dennin87]. The basic concepts, however, were discussed in the early eighties by Anderson [Anders80]. The concept of dependable computing dates back even further. As explained by Avizienis *et al.* [AvLaRa00], the concept of dependable computing first appeared in the 1830's. Because of their rather unreliable components, the first generation of electronic computers led to the development of new dependability techniques, starting in the late 1940's.

In this work we make use of the dependability concept known as *fault assumptions* [LaAvKo92]. The goal of fault assumptions is to identify all possible faults that might be activated within a system. Knowing all possible potential faults is a prerequisite for a systematic analysis to determine, for instance, the expected mean time to failure (MTTF) of a given system. This concept inspired us to identify the input to the IDS analysis based on a categorization of attacks, the goal being a systematic identification of input data.

2.1 MAFTIA terminology and concepts

As mentioned, this work uses the terminology as defined in the framework of the MAFTIA project. A first version of this terminology was introduced in the MAFTIA deliverable D1 [D1Maf00], and then significantly improved in the intermediate deliverable D2 [D2Maf01]. The deliverable D21 [D21Maf03] provides the final version of the MAFTIA terminology and concepts. Besides defining the terminology, MAFTIA applies concepts from the dependability field to ID. We have chosen to adopt these concepts and the terminology because they represent a suitable framework. Other approaches, such as the glossary initiated by NSA (National Security Agency) [NSA98], are not based on concepts as rigorous as those developed by MAFTIA, but merely try to consolidate the common use of terms. A glossary of terms defined in the context of MAFTIA that are relevant to this work can be found in Appendix E.

In the dependability field [LaAvKo92] the concepts of *fault, error* and *failure* play a central role. A fault constitutes the adjudged or hypothesized cause of an error. Applied to ID, attacks can be viewed as malicious interaction faults (see also Appendix E and D21 [D21Maf03]). An error represents the manifestation of a fault and is viewed as the part of the system-state liable to lead to failure. Finally, failure describes the event when the service delivered by a system deviates from fulfilling the system function. Accordingly a security failure represents the violation of the intended security policy, i.e., the system function in terms of confidentiality, integrity, availability or any other security-related requirement is not guaranteed anymore. Figure 4 illustrates the basic fault model. For a detailed discussion how this model can be applied to security and to ID in particular we refer to Section 3.3.2 of the MAFTIA deliverable D21 [D21Maf03].

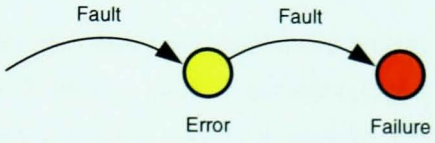


Figure 4—Basic fault model

In the remainder of this section we describe those concepts developed within MAFTIA that are relevant to this work, including the definition of *intrusion detection* that was made in terms of well-defined concepts originating from the dependability field. In the following several concepts taken from the MAFTIA deliverables D2 [D2Maf01], D3 [D3Maf01] and D21 [D21Maf03] are therefore repeated or summarized. Note, however, that this is only done for items to which the author made significant contributions.

2.1.1 Intrusion Detection

Considering the term *intrusion detection* from a linguistic viewpoint, it seems obvious that the goal is to detect intrusions. Unfortunately this turns out not to be quite correct once one takes a closer look at its current (rather unfocused) usage—the term *intrusion detection* is used as a *name* for a set of security-relevant practices and mechanisms.

In this work we use the terms intrusion detection and intrusion detection system as they are defined in the MAFTIA deliverable D2 [D2Maf01]. These definitions use the well-established dependability terminology:

- *intrusion detection:* the set of practices and mechanisms used towards detecting *errors* that may lead to *security failure*, and/or diagnosing *attacks*.
- *intrusion detection system:* an implementation of the practices and mechanisms of *intrusion detection*.

The error-detection portion of ID is the observation and analysis of the system aimed at detecting states that are error states as defined by the security policy. In practice this is often implemented by detecting symptoms or evidence of such error states and includes the detection of suspicious activities, vulnerability

scanning, and configuration checking. Additionally IDSs may perform a certain degree of fault diagnosis in which intrusions, vulnerabilities and/or attacks are analyzed and assessed further. However, most currently available IDSs do not include any fault-diagnosis mechanisms that go beyond what is required for detecting errors. When monitoring a system for suspicious activities the IDS has to analyze any activity observed to some degree. If the analyzed activity is found to be suspicious, additional analysis may be conducted. In this case we view the total of the two analysis steps as "fault diagnosis," because the diagnostic results provided along with the error report, i.e., the alarm, may contain information already determined in the first analysis phase.

2.1.2 CIDF intrusion detection model as viewed by MAFTIA

In Chapter 5 we shall develop our own IDS model. In fact this model is to be viewed as a simplified version of the CIDF ID model as it has been adopted and refined by the MAFTIA project [D2Maf01]. The following discussion summarizes the CIDF model, including the refinements done by MAFTIA.

MAFTIA presents a model of IDSs according to function, derived as a refinement of the Common Intrusion Detection Framework (CIDF) [CIDF98]. Wherever possible, the language of the CIDF is used, although some refinement has been necessary. MAFTIA also addresses issues of channels between components, which, however, are not a concern in the context of this work.

The CIDF classifies components of an IDS into four categories.

- An *e-box,* or event generator, is a component that gathers event information.
- An *a-box,* or analysis box, analyses event information toward detecting errors and diagnosing faults. The output of an analysis box may provide information to other analysis boxes.
- A *d-box,* or database, provides persistence for the IDSs. This facility will take on different forms depending upon use. It may be a complex relational database or a simple text file.
- An *r-box,* or response box, is the portion of the system that acts upon the results of analysis. According to [CIDF98], automated responses may include killing processes, resetting connections, or activating degraded service modes. In line with the discussion in Section 2.1.1, we do not consider the r-box to be part of ID *per se.* MAFTIA instead considers the r-boxes as part of the set of facilities providing error recovery, fault isolation, and system reconfiguration in a general intrusion-tolerance framework.

Figure 5 presents a refinement of the CIDF model that explicitly identifies sub-components of the e-box, and the fact that there may be multiple e-, a- and d-boxes.

Note that the decomposition may not correspond to particular physical boundaries. Vulnerabilities, and hence targets, exist at several different abstraction and implementation layers so that the model is to be applicable at several layers. The boundaries between components are determined by the level of abstraction with which we view the system: people, LANs, machines, processes, memory pages, etc.

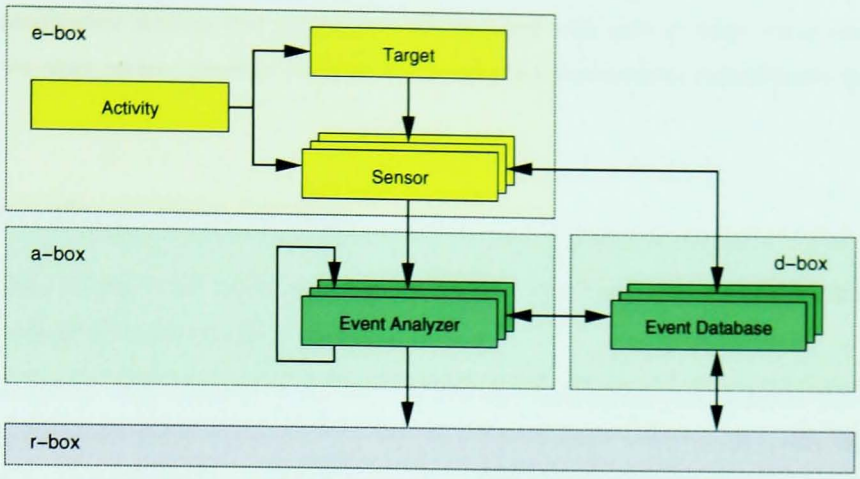


Figure 5—IDS components of the CIDF model

2.2 Classifications

Classifications and description schemes are an essential part of this work, as we shall develop a series of them in the context of this work. The following subsections aim at providing an overview of existing work, of which we shall adopt significant portions.

2.2.1 Classification requirements

An important characteristic of our approach to IDS analysis is the fact that it operates based on classes of attacks rather than on specific implementations of attacks. In order to ensure consistency and validity of the IDS analysis results, it is therefore vital that the classification scheme used for classifying attacks is sound and well-suited for its purpose. Hence, the requirements the attack classification scheme needs to fulfil have to be defined clearly.

Computer Science and most particularly ID has adopted the concept and mechanisms of classifications from other sciences such as Biology or Sociology. Based on a review of generically valid classification requirements [Bailey94, Marrad90] and requirements more specific to Computer Science [VeRaGl01, ZelWal97] and ID [Howard97, Krsul98, LinJon97], we have identified the following set of requirements that a classification scheme needs meet in order to be considered sound:

- Orthogonality: Any item can be put into only one class.
- Procedure: The classification procedure must be available, i.e., the classification must be reproducible.
- Observability: The classification procedure has to be based on observable, measurable features of the event.
- Hierarchy: Classes are hierarchical, subdivided into more specific classes.
- Consistency: Members of a given class must share common properties.

The attack classification scheme that we develop in this work will have to meet above requirements. Accordingly we shall review whether our attack classification meets these requirements (see Section 7.3.2).

2.2.2 Equivalence class testing

Equivalence class testing is an approach to system analysis that aims at systematically testing every conceivable mode of operation of a given system.

The term *equivalence class* stems from the field of discrete mathematics (see Sections 7.2 and 7.3 in [Biggs02]). Equivalence classes are non-empty and pair-wise *disjoint*. Their union covers the complete set of elements considered. Moreover all members of a given equivalence class are *related*. This means that if a is member of the equivalence class A and b is related to a, b is a member of A as well. It also means that neither a or b are members of the equivalence class B.

Equivalence class testing addresses the difficulty that in many cases it is not practical to test a system for every possible set of input parameters. Using this approach, one classifies all possible input parameters into equivalence classes—the goal being the identification of all conceivable modes of operation. The parameters assigned to a given class affect a given, previously identified mode of operation of the tested system. During the actual test runs one selects representative input parameters from each equivalence class. The number of parameters selected per class may vary. It is important that borders of equivalence classes are included in the tests if such borders can be identified. For instance, if a class is defined by the fact that a given numerical input parameter lies in a given range, the test should include values within the range but also at the limits of the range.

2.2.3 IDS classifications

The first proposals for IDS classifications date back to the late 1980s and were realized in the form of surveys such as the work by Lunt [Lunt88], Jackson [Jackso99], and others [Amoro99, Axelss99b, EsSaPi95, MWSKHH90].

Sound classifications and taxonomies of IDSs and ID-related technologies are a rather recent development, dating back only to 1999. The taxonomy proposed by Debar *et al.* [DeDaWe99] is probably one of the first real IDS taxonomies. Other taxonomies have since been published, such as the one proposed by Axelsson [Axelss00] and, more recently, the one proposed by Halme and Bauer [HalBau00].

Figure 6 shows the taxonomy proposed by Debar *et al.* [DeDaWe99]. This taxonomy was later extended and refined by Axelsson [Axelss00] and by Debar *et al.* themselves [DeDaWe00].

In the context of this work the most important elements of the taxonomy by Debar *et al.* are the *audit source location* and, especially, the *detection method*. The detection method is used to divide IDSs into so-called *behavior-based* and *knowledge-based* systems.

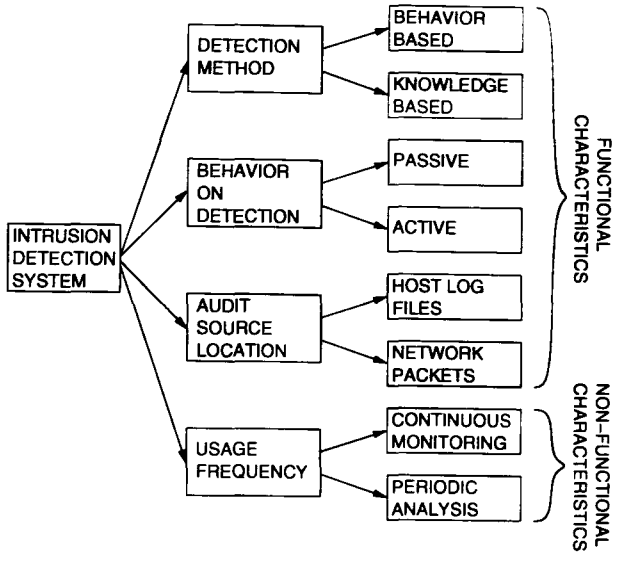


Figure 6—1999 IDS taxonomy by Debar *et al.*

Behavior-based systems, also called *anomaly-detection systems* [Mounji97], have no knowledge of specific attacks. However, they have been provided with knowledge of the behavior of the system being monitored during normal operation. Such knowledge has been acquired either by extensive training of the system [DeBeSi92, JLADGJ93] or by other more systematic approaches such as those implemented in the DaemonWatcher by Wespi *et al.* [WeDaDe00, WesDeb99]. Behavior-based systems have the important advantage that they require no database of attack signatures that needs to be kept up-to-date. The main drawback of behavior-based systems is that the alarms they generate are meaningless because generally they cannot provide any diagnostic information (fault diagnosis) such as the type of attack that was encountered. In other words, they can only signal that something unusual happened. A second drawback is the requirement that a database that describes the normal behavior of a system has to be built. Depending on the implementation of the IDS this may mean that for every single version of a product a separate signature set needs to be generated [WeDaDe00, WesDeb99] or that the IDS needs to be trained using data from the real operational environment [DeBeSi92, JLADGJ93]. When real operational data is used, one always runs the risk of including real attacks in the training data. As a consequence of this the IDS would not report attacks contained in the training data as being suspicious.

Knowledge-based systems, also called *misuse detection systems* [Mounji97], operate based on a database of known attack signatures. Whenever they encounter an activity matching a signature stored in the database, the corresponding alarm is generated. The advantage of such systems is that their alarms are meaningful, i.e., they contain diagnostic information about the cause of the alarm. On the other hand, their main drawback lies in the system component that enables the generation of meaningful alarms, i.e., the database. The database of attack signatures needs to be kept up-to-date, which is a tedious task because new vulnerabilities and attacks are discovered on a daily basis. However, most commercial systems available today, for instance NetRanger from Cisco [CiscoNR99] and RealSecure from ISS [ISSNet99], are knowledge-based systems.

Note that the revised IDS taxonomy by Debar *et al.* [DeDaWe00] as shown in Figure 7 takes into account the *detection paradigm* implemented by the IDS. If the detection paradigm of an IDS is *state-based*, the IDS tries to identify a given system state as being an error state or as being a failure state. *Transition-based* IDSs monitor a system for any state transition that represents an attack or an intrusion.

Moreover, Debar *et al.* redefine the *audit source location* by adding the categories application log files and IDS sensor alerts. This modification takes into account the differences in granularity of log data generated on a host[2]. Furthermore, the addition of IDS sensor alerts reflects the trend to hierarchical ID architectures, in which several IDSs send their alerts to a higher-level instance where the alerts are analyzed and possibly aggregated. The resulting alerts may then be sent to the next-higher instance or presented to the security officer.

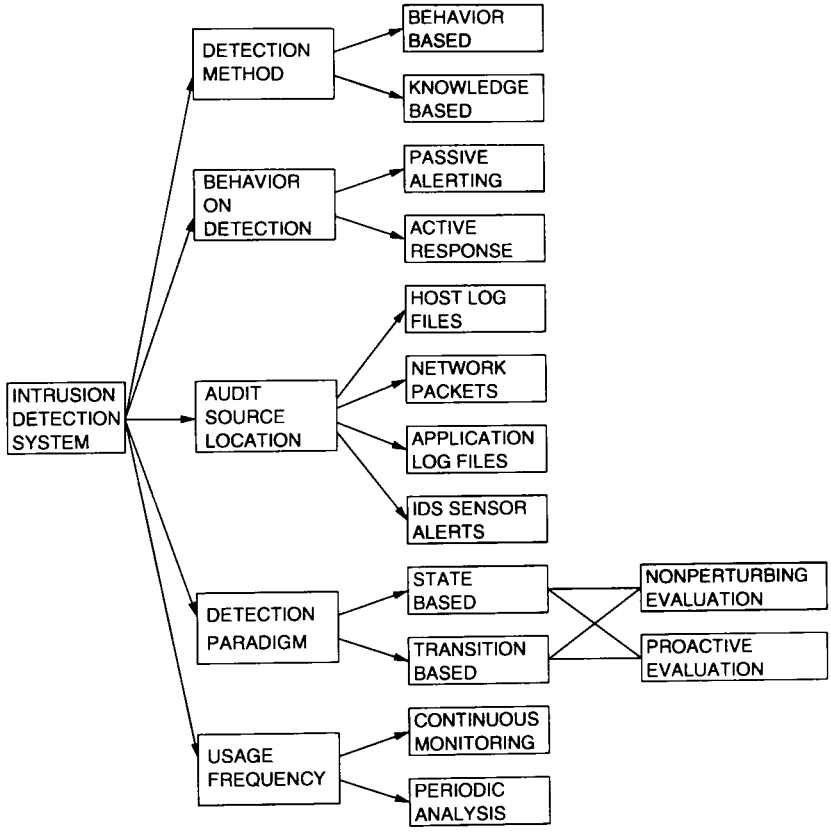


Figure 7—Revised IDS taxonomy by Debar et al.

Based on the work by Debar *et al.*, Axelsson [Axelss00] refined, using different terms, the detection method. Moreover he regrouped and extended the remaining categories into what he calls "a taxonomy of system characteristics."

[2] In Section 5.2.1.1 we classify information sources as well, but do so using a more generic approach.

As in Debar *et al.*, Axelsson classified a number of IDSs according to his taxonomy. Some of the systems appear in more than one category—raising the question whether his taxonomy is ambiguous. Axelsson gives a plausible explanation for this by stating [Axelss00, p. 7]:

> *...this is not because the classification is ambiguous but because the systems employ several different principles of detection.*

However, the presented schemes for classifying and describing IDSs represent merely informal schemes that are not suitable for the automated analysis of IDSs as envisaged in this work. They pursue different goals and, most importantly, do not enable IDS descriptions at the level of detail required for our approach. In Chapter 5 we develop a description scheme for IDSs that is partially based on above concepts and that enables the description of IDSs at sufficient level of detail.

2.2.4 Attack classifications

Attack taxonomies and the resulting classifications are of interest to us because we envisage using a classification of attacks to identify the input to the evaluation of IDSs. Computer security attacks and vulnerabilities have been classified in many ways; however, so far no commonly accepted reference classification exists.

As described in detail by Howard [Howard97] many attack classifications are based on empirical lists or simple lists of terms. The weakness of these classifications is that often the terms used to classify are not mutually exclusive and/or properties of vulnerabilities, and that properties of attacks are not clearly separated. An example of such a classification is the one proposed by Cohen [Cohen95].

One of the earliest works is the one by Neumann and Parker [NeuPar89]. There the authors classified data from about 3000 incidents, which they had collected over 20 years, according to nine different computer misuse techniques they had defined. These categories are, as the authors state themselves, not mutually exclusive. Based on [NeuPar89], Neumann proposed an extended scheme [Neuman95] where he also incorporates the vulnerability exploited and the impact of an attack.

Other approaches, such as the one proposed by Howard [Howard97], classify attacks according to several sets of categories concurrently. Lindqvist and Jonsson [LinJon97] proposed a similar approach by classifying attacks according to the two sets of categories "intrusion technique" and "intrusion result." In his Ph.D. thesis [Kumar95] Kumar introduces a classification based on attack signatures used within the IDS IDIOT [CDEKS96]. This classification is based on the type of observation required to be able to detect a given attack. As we shall discuss in Chapter 4, this work is related to our activity classification, which is based on activity properties observable by an IDS.

While considering these different approaches to the classification of attacks, we were able to identify the following classification categories:

- List of terms: a wide range of highly diverse terms. Examples: [Cohen95, IcSeVo95]

- Tools: type of tool used to execute an attack e.g., script, distributed tool etc. Example: [Howard97]
- Prerequisites: the prerequisites to be met before an attack can be staged successfully, e.g., access required, resources required, skills required etc. Examples: [CheBel94, JiSiIr00, Longst97, NeuPar89]
- Technique: the technique used to run a given attack, e.g., spoofing. Examples: [LinJon97, NeuPar89, Stalli95]
- Detection technique: the technique or type of signature required to detect a given attack. Examples: [Kumar95, KumSpa95]
- Impact: the immediate damage caused by a successful attack, i.e., an intrusion. Examples: [CheBel94, Howard97, JiSiIr00, LinJon97, NeuPar89, SinSig01]

Bear in mind that many attack classifications also include information on the vulnerabilities exploited, characteristics of the attacker, and his/her objectives etc. These inclusions let them become general classifications of security issues rather than attack classifications.

As mentioned in our approach to IDS evaluation, we use the statistical results of an attack classification to identify the classes of attacks that are actually of relevance. Clearly, for us to be able to derive a representative set of activities from such a classification, the classification needs to be centered on the attack aspects observable by IDSs. Also, we require the classification to be sound, i.e., to meet the requirements defined in Section 2.2.1.

Unfortunately most attack classifications do not meet these criteria (the same is true for many vulnerability classifications). This has been observed and extensively criticized earlier by several authors [Howard97, Krsul98, LinJon97]:

- Namely the categories used in attack classification are often not mutually exclusive. This is often due to a bad choice in the set of categories that form a category set.
- An attack may qualify for several—mutually exclusive—categories concurrently. In many cases this is not caused by a bad choice of the categories, but rather by the fact that attacks involving a sequence of steps are not atomic operations.
- An IDS can observe a given attack in many different ways, depending on the information source and detection techniques used.

The fact that we were not able to identify an attack classification that meets above requirements led us to develop a categorization scheme for attacks, described in Chapter 4.

2.2.5 Vulnerability classifications

Vulnerabilities are tightly linked to attacks. In fact, to successfully launch an attack, the corresponding exploitable vulnerability must be present in the system. This close relationship may cause confusion when

defining a classification. One example is the classification proposed by Howard in [Howard97], where he proposes a "computer and network attack taxonomy" that contains categories describing the vulnerability exploited. This is not to say that combining attack and vulnerability characteristics is not viable, but they should be clearly distinguished to avoid confusion.

Similar to attack classifications, the classes used for a vulnerability classification are determined by the goal pursued. For example, if the genesis of a vulnerability is of interest, classes describing the genesis of the fault will be introduced, as done in the classification proposed by Landwehr *et al.* [LBMW94]. Their work is based on hierarchical categories, i.e., a decision tree. However, as explained by Howard [Howard97], this classification is ambiguous because vulnerabilities may qualify for several categories concurrently, i.e., violate the orthogonality requirement.

In his Ph.D. thesis [Krsul98] Krsul discusses 17 different vulnerability classifications. We are not reproducing the entire discussion here, but instead provide an overview of the various classes chosen for these classifications:

- Genesis: The way the fault was introduced. Examples: [AsKrSp96, Aslam95, LBMW94, Longst97].
- Time: The point in time at which a fault was introduced, e.g., design phase, coding phase, maintenance etc. Examples: [Howard97, LBMW94].
- Cause: The cause for the introduction of a fault, e.g., wrong algorithm or parameter used etc. Examples: [Knuth89, Longst97].
- Removal: The steps to be taken to remove a given fault. Example: [DeMMat95].
- Type: The type of operation that is faulty, e.g., decision making, data handling etc. Examples: [BasPer84, KrSpTr98, OstWey84].
- Location: The location of the fault, e.g., the faulty object, protocol, device etc. Examples: [DLAR91, KrSpTr98, LBMW94, Tanenb87].
- Threat: The potential threat represented by a vulnerability. Examples: [KrSpTr98, Power96].

The threat category is closely related to the impact category introduced in Section 2.2.4, as are attacks and vulnerabilities in general. A given attack does not necessarily exploit a given vulnerability in the worst possible way, i.e., the threat represented by a given vulnerability is not necessarily fully exploited by an attack attempting to exploit that vulnerability.

Example: *A fully exploitable buffer overflow [Aleph96] vulnerability may be used to take over control of the system or merely to crash the system.*

2.2.5.1 Enumeration of vulnerabilities

A non-classificatory approach to deal with vulnerabilities is to enumerate them. Recent efforts to enumerate vulnerabilities are driven by the common need for unique identifiers for vulnerabilities when handling security incidents, reporting the finding of vulnerability on a given system, and also when

reporting the observation of an attack, i.e., when an IDS generates an alarm [DeHuDo00, WooErl01]. The latter, however, is less obvious as attacks may not always be mapped onto specific vulnerabilities and vice-versa.

Common Vulnerabilities and Exposures (CVE) [ManChr99] is a security-industry-wide effort coordinated by the MITRE Corporation [CVE99]. CVE is a dictionary that aims at facilitating the sharing of data across separate vulnerability databases and security tools. While CVE may make it easier to search for information in other databases, it should not be considered as a vulnerability database on its own merit.

Another well-known effort is the Bugtraq ID. Bugtraq IDs are assigned based on vulnerabilities as published on the security mailing list bugtraq, operated by the SecurityFocus [SecFoc] web site. CVE entries and Bugtraq ID database records both refer to their corresponding counterparts in the other database. However, neither CVE entries nor Bugtraq IDs—or any other identifiers—are assigned based on the same principles.

Example: *A design flaw recently discovered in the Microsoft IIS Webserver software enables a remote user to execute arbitrary commands on the machine running the webserver software. In this particular example CERT [CERT] released the advisory CA-2001.12 [CA1201]. The same vulnerability has been assigned the Bugtraq ID 2708 [SF2708] by SecurityFocus and the CVE candidate name CAN-2001-0333 [CVE033301] by the CVE editorial board. Once the review process of the CVE candidate entry is finalized, the name of the entry will be changed to CVE-2001-0333 provided that the entry is not rejected, which seems very unlikely given the severity of this case.*

2.3 Evaluation of intrusion detection systems

IDSs can be evaluated or simply compared in many ways—all pursuing different goals. So far we have been able to identify three different approaches:

1. Description-based comparison
2. Enumerative evaluation
3. Benchmarking

In the following, we discuss the three approaches listed above. Because the so-called Lincoln Lab experiment by Lippmann *et al.* [LFGHKM00, LHFKD00] had the most significant impact in the field with respect to IDS evaluation we shall dedicate Section 2.3.3.1 to their work and explain it in greater detail. Lastly we would like to mention the recent survey on ID by McHugh [McHugh01] in which he dedicates an entire chapter (Chapter 5) to the various approaches to IDS evaluation, including the initial version of the approach presented here [Alessa00].

2.3.1 Description-based comparison of intrusion detection systems

Summaries such as the one maintained by Sobirey [Sobire98] provide an overview of existing IDSs. However, they do not allow the comparison of IDSs based on a pre-defined set of criteria. In an extensive work Jackson [Jackso99] did exactly this. She described numerous IDSs by means of a predefined set of criteria that then allows their comparison. It is clear that this work does not take into account the quality of IDSs in any form as it is based on the description of IDSs only. Also, it provides limited information about the classes of attacks detected by the IDS only. Nevertheless this work provides a useful overview of the IDSs available and the technology used to perform ID.

As we use an IDS description framework, it is probably fair to consider our approach as a *description-based analysis of IDSs*.

2.3.2 Quantitative evaluation of intrusion detection systems

Another, more practical approach is to test IDSs for a predefined list of attacks: Here one systematically launches every attack on the list and records the alarms the IDSs being tested generate. Such enumerative testing can be based on pragmatically composed lists of attacks or on enumerations such as CVE or Bugtraq IDs. The test environment is generally kept simple and does not include additional elements to generate background activity artificially.

One example is the diploma thesis of Gigandet, the results of which were published in a research report [Gigand00]. In this report he describes the results he obtained by examining Symantec's host-based IDS product *Intruder Alert*[3] [InAlert01] for a large list of attacks.

2.3.3 Benchmarking of intrusion detection systems

Benchmarking of IDSs has become the most popular approach to IDS evaluation. In part this is certainly the effect of CERT's recommendation [ACFMPS99] to develop adequate ways to test IDSs. Benchmarking generally imposes a relatively complex testbed that permits the testing of IDSs under conditions closely matching those found in real environments. This means that, in contrast to the previous approach, a significant effort has to be made to generate background activity that is as realistic as possible. The background activity influences IDSs in two ways. Firstly, attack-similar background activity may cause IDSs to generate false alarms. Secondly, a high volume of background activity may reduce the detection rate of IDSs because they are overwhelmed by the amount of information to analyze. In mentioning these aspects, we have already identified one of the big issues with this type of approach: Because no two environments are identical, the evaluation results are of limited value only as soon as one starts considering IDSs in differing environments.

---

[3] Originally this product was developed by Axent. Axent was acquired by Symantec in 2000.

As mentioned, the most highly regarded work in this field is certainly the "1998 and 1999 DARPA off-line intrusion detection evaluation" [LFGHKM00, LHFKD00] performed at the Lincoln Lab by Lippmann *et al.* They used a complex testbed where IDSs were systematically exposed to malicious and benign activities. This work represents an important contribution to the field. It thereby builds the foundation for further work such as the work by Maxion and Tan [Maxion98, MaxTan00], the work by Durst *et al.* [DCWMS99], the work by Gaffney and Ulvila [GafUlv01] and last but not least the LARIAT (Lincoln Adaptable Real-time Information Assurance Testbed) project [HRLC01, RCFRLH01], which is the continuation of the Lincoln Lab evaluation.

It is worth mentioning that McHugh has criticized the Lincoln Lab evaluation repeatedly [McHugh00, McHugh00b]. His critique represents an important contribution as it not only identifies many issues that apply specifically to the Lincoln Lab work, but also—and even more importantly—issues that generally apply to IDS benchmarking approaches. It is therefore no surprise that many of the issues identified by McHugh also apply to most recent work such as that pursued by the NSS Group [Walder01a, Walder01b].

2.3.3.1 The Lincoln Lab evaluation

In their evaluations Lippmann *et al.* [LFGHKM00, LHFKD00] made a considerable effort to create a testbed that closely resembles a real environment—the fictional Eyreie Air Force base. Besides systems used for attacking other systems and systems serving as attack targets, the testbed also included a router to simulate an internal and an external network. In addition, hundreds of PCs and workstations were simulated on the internal network, and thousands of them on the external network.

This setting was then used to simulate five weeks of real operation of the site. During this period, all network traffic and host audit data (Solaris BSM[4] and Windows NT audit event logs) on the internal as well as on the external network was recorded. In addition, nightly filesystem dumps of the security-relevant files were made from the attack target machines. This setup resulted in a huge amount of information—mainly consisting of background activity. For instance, on average, every day 411 Mb of networking data was recorded.

During the five-week period, real attacks were run on the testbed—overlapping with the simulated background activity. The attacks were marked in the recorded data in order to enable the comparison of the alarms generated by the IDS tested and the attacks actually run.

Unfortunately Lippmann *et al.* provide no information about the criteria that were used to compose the set of attacks employed. In their 1999 evaluation, they used 56 different attacks that they classified according to the classification by Weber [Weber98]. This classification distinguishes five classes of attacks:

1. Probing attacks,

[4] BSM stands for Sun's *Basic Security Module,* which is used to generate level C2 audit logs on Solaris systems.

2. Denial-of-Service (DoS) attacks,
3. Remote to Local (R2L) attacks,
4. User to Root (U2R) attacks, and
5. Data attacks.

Once all this information was recorded, they split the data into two subsets. The first portion, covering three weeks, was made available to test and tune IDSs and included the markers to find attacks in the test data. The second portion, covering the final two weeks, did not include any markers where to find attacks in the test data. This second portion was then used to evaluate various IDSs.

The results then obtained illustrate the general weaknesses of current IDSs. As mainly knowledge-based IDSs were evaluated, it is clear that many attacks were missed because either the corresponding attack signature was not available or the attack was executed in a stealthy, i.e., obfuscated, fashion that prevented its detection.

Finally, they acknowledge the deficiency of their testbed with respect to false positives. They clearly state that the false-alarm rates need to be interpreted in the context of the testbed, and that the false-alarm rates may vary significantly depending on the environment in which the IDS is used.

As mentioned, McHugh published a critique [McHugh00, McHugh00b] of the Lincoln Lab evaluation. There he identifies a series of issues the evaluation suffers from. He points out that simulated background traffic seems to be too low. This has a negative impact on the false-alarm rates measured, which therefore also seem to be too low. Moreover, he criticizes the manner in which the attacks were selected and how the selected attacks were distributed over the test data—the different classes of attacks were not distributed in a realistic manner. In addition, he identifies a long list of further issues the approach suffers from. Last but not least, he criticizes the attack classification [Weber98] used because it is not useful in describing what an IDS might see.

McHugh identified the major deficiencies in a precise and well-founded manner. It is his critique that has significantly influenced the more recent work pursued at the Lincoln Lab. With LARIAT [HRLC01, RCFRLH01], the Lincoln Lab is developing a new benchmarking system that emphasizes the interactive generation of realistic attack sequences more strongly. Furthermore, they address many of the issues identified by McHugh.

However, one has to acknowledge the Lincoln Lab evaluation as an impressive and important work, especially because it has influenced a series of further undertakings.

One of them is the work of Maxion and Tan [Maxion98, MaxTan00] on the benchmarking of anomaly-based detection systems, i.e., behavior-based IDSs. In contrast to the Lincoln Lab evaluation, they also performed tests in a real-world environment, which yielded far more representative results with respect to false alarms generated by the system being evaluated.

Another recent undertaking is the work of Wan and Yang [WanYan01], who developed a software platform for the testing of network-based IDSs in real-world environments. Their approach uses the

traffic of real-world environments as background traffic, which is overlaid with the attacks used for evaluating IDSs.

Last but not least the work by Marty [Marty02] needs to be mentioned. The goal of this effort is not the benchmarking of IDSs but the in-depth analysis of how diverse IDSs react to attack obfuscation techniques and, most importantly, how the various IDSs report these obfuscated attacks. This work, which in part is motivated by the concepts developed in this thesis, is being conducted at the IBM Zurich Research Laboratory.

2.3.3.2 Benchmarking work pre-dating the Lincoln Lab evaluation

Prior to the work by Lippmann *et al.*, relatively little work was done in this field. The first notable work is that by Puketza *et al.* [PZCMO96]. In their work they made a considerable effort to evaluate the network-based IDS NSM [HDLMWW90] under stress conditions. By doing so, they were able to demonstrate the relation between the number of omitted PDUs (protocol data units) and the increasing load on the system hosting the IDS. In the continuation [PCOM97] of their work, they were able to demonstrate how attack obfuscation techniques can be used to prevent IDSs from detecting attacks or at least to reduce the risk of detection for adversaries.

Also prior to the Lincoln Lab effort, the IBM Zurich Research Laboratory developed an IDS testbed [DDWL98] that was used to evaluate IDSs for IBM internal purposes.

2.4 Discussion

When describing IDS benchmarking, especially as done at the Lincoln Lab, numerous issues have already been identified. Many of these issues are not specific to the Lincoln Lab work, but are valid for any form of IDS benchmarking, especially if performed in a testbed. However, this does not mean that these evaluation results are not useful. Clearly, any IDS benchmarking approach needs to be considered in the context of the goals pursued. The goal of the IDS benchmarking initiatives is, in most cases, to compare IDSs and to provide a foundation for deciding which IDS to choose to protect a given environment.

2.4.1 Issues in IDS benchmarking

Although a remarkable effort has been made to meet these goals, the results achieved so far are still imperfect. This is for several reasons—most of which have been described by McHugh [McHugh00, McHugh00b, McHugh01]. The most important issues are the following:

- The potential *threats* define the protection required. In order to judge the utility of an IDS the threats one wishes to address need to be defined such that one can clearly define the requirements one expects the IDS to meet.
  - In current benchmarking efforts the set of attacks used for benchmarking is selected in a relatively ad-hoc fashion. However, the choice of the *test data* is crucial to every form of

system testing. Many approaches to IDS evaluation lack a clear strategy in the selection of their test set or use a strategy that is flawed, e.g., [LFGHKM00, LHFKD00]. Our approach suffers far less from such insufficiencies because it is based on classes of attacks. The strategy used in this work to determine attack classes meets the equivalence class testing requirements presented in Section 2.2.2. It uses a categorization scheme that permits attacks and benign activities to be classified according to aspects that are observable by IDSs (see Chapter 4).

- Another aspect that requires increased attention are *attack-obfuscation techniques* or, in other words, *activity variations*. Although, for instance, Lippmann *et al.* acknowledge the importance of addressing this issue in their 1999 evaluation [LHFKD00] and have made a considerable effort to implement such variants [Das00, Kendall99], the issue generally is not addressed in a generic fashion. As we shall see in detail in Chapter 6, our approach addresses activity variants in a highly generic fashion that permits the concurrent application of multiple variations to one activity.
- Furthermore it seems advisable to weight the severity of threats, i.e., to weight the importance of the benchmarking results for any given attack. For instance, in an almost purely Windows-based environment, it seems obvious that one would give priority to the detection of Windows-related attacks. This does not mean that one would not also check for Unix-related attacks as they may provide additional indications about malicious activities in progress. However, one would certainly assign a lower weight to the benchmarking results for Unix-related threats than to those for Windows-related threats.

- The testbed *environment* biases the benchmarks. Whenever an IDS is connected to a real operational network or installed on a host, the IDS is exposed to a distribution of activities and activity variants that is specific to the environment considered. As a consequence benchmarks are mostly performed in an environment that is not representative of the target environment for which an IDS has to be selected.
  - False positives: The rate of false positives determined during a benchmark may not be representative of the rate of false positives one observes in the target environment. For instance, if a class of benign activities for which the evaluated IDS has the potential of generating false alarms is very frequent, the IDS is likely to generate many false positives. If, however, such activities are very infrequent or do not even occur in the evaluation environment the benchmarking results produced will not reflect this weakness of the IDS.
  - True positives: The frequency and type of attacks used in the evaluation environment directly influence the rate at which true positives are generated. The type of attacks used may non-uniformly emphasize strengths and weaknesses of IDSs.
  - Background activity: The amount and type of (benign) background activity may impose a significant load on IDSs, which may cause them to omit data, e.g., drop PDUs. The type of background activity used in the evaluation environment may impact IDSs in a non-uniform

fashion because they differ in the amount of resources spent on analyzing different types of activities. As a consequence evaluation results may emphasize non-relevant or suppress relevant strengths and weaknesses of IDSs because the artificial creation of background activity as observable in real-world environments is difficult. Moreover in real-world environments, background activity can change significantly over time—a fact nicely illustrated by the data-mining experiments described by Julisch [Julisc00, Julisc01].

- Topology: The modeling of environments is more complex than merely determining the frequency of activities and their variants, although already this may be a challenge. Especially when considering a highly distributed ID architecture, one has to take into account the topology of the system (i.e., the network) when modeling the environment, simply because not every IDS is exposed to same distribution of activities. This issue significantly increases the complexity of processing the alarms generated by multiple IDSs installed at different topological locations.

- The *utility* of the information provided by IDS alarms is not assessed. Current benchmarks do not really evaluate the expressiveness of IDS alarms. They only distinguish between true and false positives. The contribution that a given alarm may make to the interpretation of what the IDS has observed is not evaluated. Also, the method employed to decide whether a given alarm actually describes the detected attack correctly is not clearly defined. For instance, an http attack that is obfuscated by fragmented PDUs may be reported by an alarm that indicates the presence of fragmented PDUs. Such an alarm qualifies as a true positive because the fragmentation was reported correctly. However, the actual attack was not correctly identified, i.e., diagnosed[5]. Accordingly the results provided by IDS benchmarks that do not judge whether the semantics of alarms is sufficient to allow reasonable conclusions about the cause of the alarms have to be considered with care. An example for such an approach is the NSS work [Walder01a]. Also, current approaches do not clearly distinguish whether alarms contain diagnostic information about the impact of the attack, e.g., whether the attack was successful or not. This becomes especially relevant whenever one compares IDSs of different types, for instance, behavior- and knowledge-based systems.

- The *potential* for detecting unknown and non-tested attacks cannot be determined. The results of current benchmarks are not suitable to forecast an IDS's potential to detect attacks other than the ones it has been tested for. This obviously also includes attacks that are not yet known. This situation could certainly be improved by classifying attacks according to a classification that is centered around the observable aspects of attacks instead, for instance, around the impact of attacks. Nevertheless, the test result, i.e., a judgement as to whether an IDS has the potential for detecting a certain class of attacks, is always going to be biased by numerous external factors. For instance, it will depend on how the IDS has been configured or, even more importantly, on whether the signature for the attack considered has been defined for the IDS tested. It may well

be that the tested IDS has the potential for detecting a certain class of attacks. However, if the IDS does not provide a signature for the attack that is used to verify whether it is capable of detecting said class, the test result will be misleading.

2.4.2 Limitations of our approach

As explained in Chapter 1, the goals of current benchmarking approaches are different from those of our far more conceptual approach. The fact that our approach operates based on entire classes of attacks enables predictions of high generality about the behavior of IDSs. These predictions cover a significant number of actual attacks requiring a only comparatively limited analysis effort. Most importantly, these predictions can be made based on the design of an IDS, i.e., even before the IDS is actually implemented.

However, this does not mean that our approach does not have its own drawbacks. One might, for instance, argue that it should take into account the environment for which the analyzed IDS is envisaged. However, in view of the goal that we pursue with our approach, taking into account the environment is merely a desirable extension rather than a requirement. It should, however, be noted that an IDS design that our approach has determined to have the *potential* for detecting attacks of a certain class may turn out not to be able to do so in practice. This can happen if the IDS implementation

- uses highly IDS-implementation-specific heuristics,
- suffers from implementation flaws,
- is configured badly, or
- does not provide a sufficient set of signatures (assuming a knowledge-based IDS).

In other terms, our approach determines whether an IDS offers the potential to detect a given attack class. It does so also for different configurations of a given IDS. However, here one has to distinguish between configuration changes that impact the manner an IDS performs its analysis and changes that merely affect a given set of attacks. For instance, if one enables the reassembly of TCP streams for one installation of Snort [Roesch99] but not for a second installation that operates in parallel, significant differences can be expected. On the other hand, the fundamental properties of the IDS remain unchanged if one enables, disables or modifies the signature for detecting a given attack.

Moreover IDSs often use heuristics in order to rate observed activities or suffer from subtle implementation flaws that may increase the effort required for their evaluation significantly. Marty [Marty02] describes an IDS that was found to raise alarms whenever it observes IP fragments that are between 28 and 42 bytes in size (see also the example provided in the introduction of Chapter 1). Our approach operates at too high a level in order to reflect such highly implementation-specific aspects. However, benchmarking-based approaches do not analyze such subtleties either because the number of individual tests required would be impractically high.

---

[5] In fact, this is an example for the concurrent occurrence of a true positive and a false negative.

2.4.3 Summary

In this work we introduce a new description scheme for IDSs and a new classification scheme for attacks. These schemes are defined such that they enable predictions as to whether an IDS is capable of detecting a given class of attacks and to determine the manner in which any such finding will be reported. The existing schemes for IDS classification and attack classification either pursue an entirely different goal, are not of sufficient detail or both. Moreover our attack classification scheme is descriptive, whereas existing attack classifications merely associate attacks to predefined categories.

Our approach to IDS analysis is quite different from existing IDS benchmarking approaches. In Table 1 we have summarized the main differences between the Lincoln Lab evaluation, which is probably the most important work in this field so far, and our own approach.

Table 1—Comparison of the Lincoln Lab evaluation to our approach

Comparison of paradigms	Lincoln Lab evaluation	Our enumerative and description-based approach (rule-based evaluation)
Goal	Provide measurements to judge quality of, to compare and to support the selection of IDSs	Provide guidance to IDS designers by predicting the detection capabilities of IDSs
Realization	Evaluation *testbed; replay* of recorded *traffic*	Rule-based *Description* of the characteristics of IDSs and attack classes using Prolog rules
What is analyzed?	IDS *implementation* and *configuration*	*Potential* of IDS *designs*
Environment	A *chosen* testbed	*Independent* of environment
Input	Real *attacks* and *background activity* (e.g., traffic)	*Description* of classes of attack classes; the attack classes considered are determined by means of an extensive attack categorization
Input variation	*Known variants* of given attacks selected	Variants of attack classes are *generated systematically*
Results	List of *specific attacks* the IDS can detect; Number and percentage of detected attacks and false positives (ROC curves).	The set of attack classes an IDS design has the potential of detecting and the generalized alarms by which the attack classes are reported.
Limitations	Significant bias by test environment; enormous effort involved; only limited coverage of conceivable attacks.	Prediction made for IDS design may differ from behavior of the actual IDS implementation.

It seems inappropriate and impossible to establish a ranking of the existing approaches, including ours, in terms of one being superior to the others, because all approaches aim at different goals—in particular ours. Also, it is clear that our approach, being very conceptual, will only be able to provide results at a relatively high and conceptual level. However, this is perfectly in line with our goal of providing guidance to designers of IDSs by predicting the detection capabilities of their designs. For such guidance class-level results are required because the typical IDS design goal is to cover entire classes of attacks rather than specific attacks only.

In the preceding discussion on IDS evaluation, it has been observed that ID alarms are not of binary nature, i.e., they are not simply correct or incorrect, and that, as a consequence, it may not always be sensible to simply rate them as true or false positives. Naturally, this also applies to our approach, which is the reason why we introduced the concept of *generalized alarms*.

2.5 Conclusion

In this chapter we have introduced important concepts originating from the ID and dependability fields. In part this was done by summarizing concepts developed in the context of MAFTIA, to which the author has made significant contributions. As a result we were able to define the ID-related concepts used in this work in as consistent a manner as rarely done elsewhere.

Furthermore we have discussed various existing approaches to IDS, attack, and vulnerability classification, and found that none of them is directly usable for our approach to IDS analysis. However, to some extent, important contributions, such as the IDS classification by Debar *et al.* [DeDaWe00, DeDaWe99], will be adapted as we develop our IDS description scheme in Chapter 5. Finally we have discussed existing approaches to IDS evaluation and therein provided a comparison of our approach with the important Lincoln Lab evaluation by Lippmann *et al.* [LFGHKM00, LHFKD00]. However, as the two approaches rely on fundamentally different concepts and aim at different goals, they are not meaningfully comparable.

Chapter 3 Overview

In this chapter we provide a more detailed overview of our approach—thereby introducing the key concepts. We begin with an outline of the scheme that we employ for describing IDSs (Section 3.1). In Section 3.2 we introduce our classification of attacks that yields descriptions of attack classes. Section 3.3 describes the actual analysis method for IDSs as it was implemented in the RIDAX prototype (Section 3.4). Section 3.5 outlines a possible approach to validate the results produced by RIDAX and identifies the challenges involved. In Section 3.6 we discuss the results obtained and outline further applications of the approach presented, such as the one described in Chapter 8.

3.1 IDS description framework

In the following we introduce a description scheme for IDSs that enables a detailed but concise description of those IDS characteristics that are relevant to the detection of attacks. The scheme has been developed in a pragmatic manner based on the experiences gained by the development and deployment of IDSs and the in-depth analysis of attacks and vulnerabilities in the context of maintaining IBM's security database VulDa [DacAle99]. Further details on VulDa will be provided in Appendix A. Examples of IDS characteristics that our scheme captures are the data that is available for analysis, the extent to which protocols are interpreted and verified, and the analysis techniques that IDSs may use to determine whether or not an observed activity might represent an attack. IDSs may be rather complex systems, and, even more importantly, the characteristics of two IDSs may be fundamentally different. In order to be able to describe such a variety of complex systems we chose a scheme that clearly separates the various aspects of an IDS.

In a first step we chose a system model that divides an IDS into sensors and detectors. The sensor corresponds to the e-box known from the CIDF model [CIDF98] and retrieves raw data from an information source that it then passes on to the detector. In CIDF terminology, the detector can be viewed as the combination of an a- and a d-box. Figure 8 illustrates this simple model (see also Section 2.1.2).

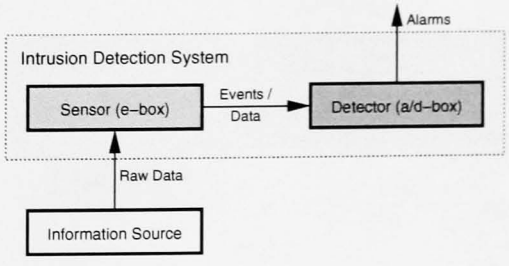


Figure 8—IDS model used for our description scheme

In a second step we introduce a scheme for representing IDS characteristics in a two-dimensional and hierarchical manner. Using such a scheme it becomes possible to specify very detailed and specific IDS characteristics using two comparatively simple parameters. One parameter denotes a generalized

characteristic such as the ability to apply regular expression matching on data. This first parameter, however, does not specify the scope within which this IDS characteristic is available, i.e., it does not determine the kind of data regular expression matching can be applied to. Instead, this is specified by the second parameter that determines the so-called *IDS scope*. The IDS scope specifies the scope of validity of generalized IDS characteristics in a hierarchical manner.

3.1.1 IDS scope

The IDS scope hierarchy is the result of an iterative process that includes not only this work on IDS analysis but also our work on attack categorization in the context of the VulDa maintenance [DacAle99]. The scheme is divided into the three top-level scopes "networking," "user" and "host." The IDS scopes "networking" and "host" are then divided into a number of lower-level scopes such as the IDS scope "application layer" or "process." Beneath these lower-level scopes further even more specific scopes can be defined. The "user" IDS scope refers to a human user and is not divided into lower-level scopes. Figure 9 provides an overview of the IDS scopes identified. In the outer right column it provides numerous examples of low-level IDS scopes that we have identified in the context of this work. Note that this scheme is not finalized but is meant to be extended as new technologies emerge.

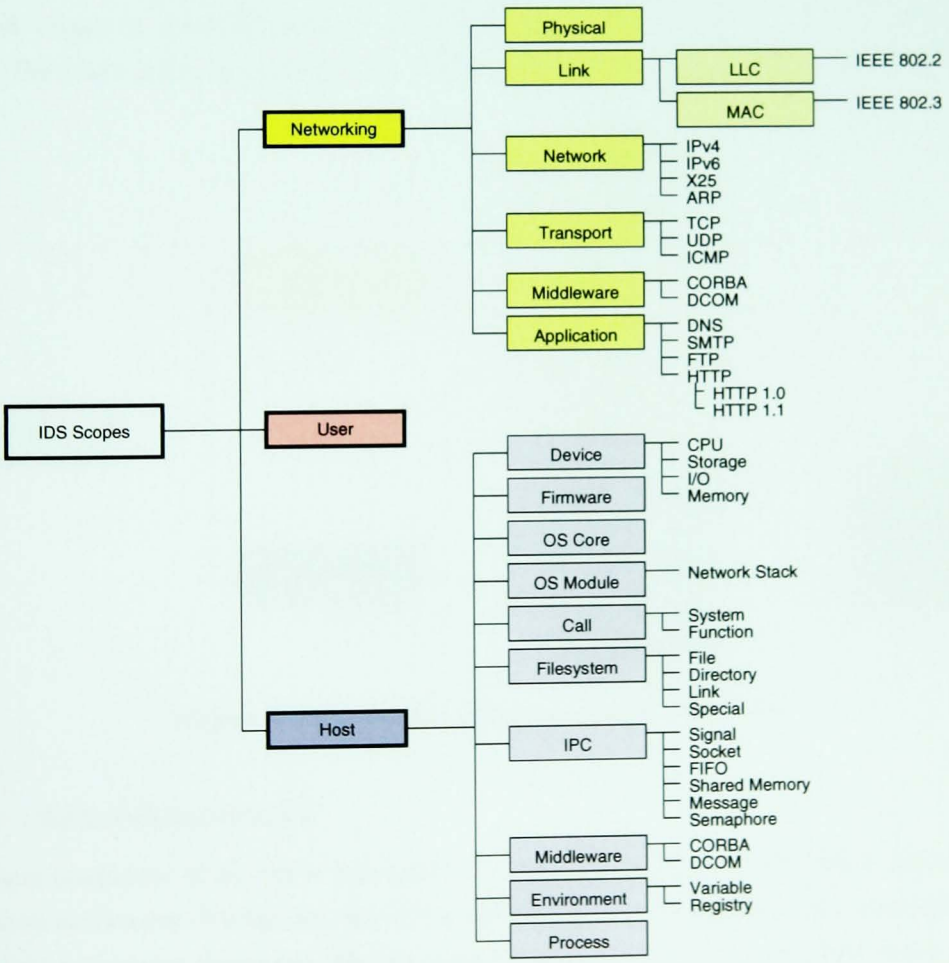


Figure 9—IDS scope tree with examples of low-level IDS scopes

We extended the IDS scope hierarchy shown in Figure 9 further by adding attributes to the IDS scopes if needed. These attributes express, for instance, whether a transport layer IDS scope is connection oriented or whether multiple transactions are supported within one session. Applied to http, the latter permits us to distinguish semantically between http versions 1.0 and 1.1 (unlike http version 1.0, http version 1.1 supports multiple subsequent http requests within one session).

IDS scopes play an important role in this work because we also use them for classifying attacks. Their use thereby supports us by ensuring consistency across all the models developed and proposed in this work. The concept of IDS scopes will be developed in Section 4.1. To a large degree it is based on insights gained during the operation of the VulDa database (Appendix A).

3.1.2 IDS characteristics

The IDS model we use in this work is illustrated in Figure 8. In our description scheme we therefore distinguish between sensor characteristics and detector characteristics. The majority of these characteristics are represented by a 2-tuple consisting of an IDS scope and a generalized IDS characteristic. In addition to the IDS-scope-dependent characteristics, we also use a small number of IDS-scope-independent characteristics for describing some generally valid properties of the components described. Figure 10 shows the hierarchy of IDS characteristics that are used in the IDS description scheme. The details of this scheme, which is outlined in the following, are developed in Chapter 5.

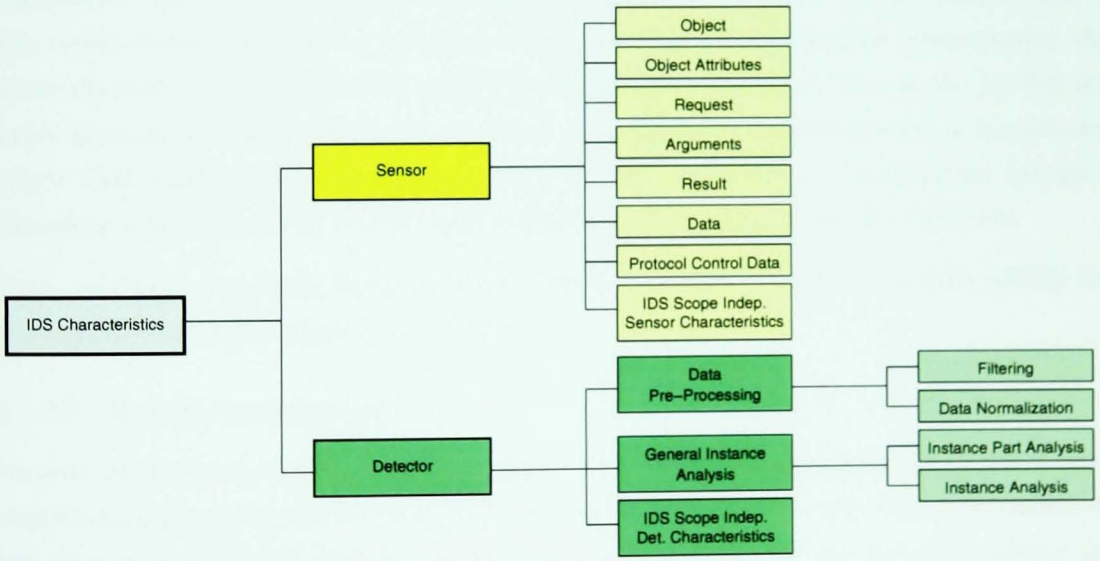


Figure 10—Top-levels of IDS characteristics hierarchy

3.1.2.1 Sensor characteristics

The sensor component of an IDS is responsible for retrieving data from a data source and making it available to the detector. We use very few IDS-scope-independent characteristics, the most important of which is the *information source type*. The information source type describes whether the data is captured in the form of raw data or in the form of log data as a process or the operating system creates it. This is

illustrated in Figure 11. Raw data sources are further separated into external and internal data sources. External sensors capture data before it actually reaches its destination, whereas internal sensors are implemented by a component that is embedded into the element surveyed.

The information source type is an important characteristics because it determines the extent to which the data provided needs to be pre-processed before the detector can analyze it. For instance, using a network packet sniffer provides data that requires substantial pre-processing before application layer data can be analyzed because the data has to traverse a number of network layers. Figure 11 provides an overview of the information source type hierarchy used along with some examples of actual sensor types.

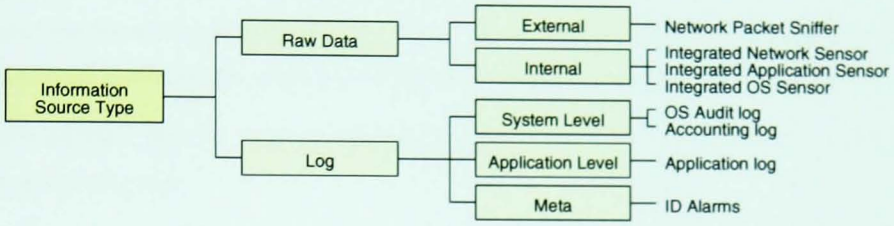


Figure 11—Information source types hierarchy including examples

The IDS-scope-dependent sensor characteristics are used to describe the data items that a sensor is providing to the detector. Each category of sensor characteristics shown in Figure 10 represents a number of actual sensor characteristics. The category "protocol control data," for instance, contains sensor characteristics such as "destination ID" and "destination name." The semantics of these characteristics is only clearly defined once they are combined with an IDS scope. Considering the above example, the sensor characteristics "destination ID" combined with the IDS scope "IPv4" denotes the fact that the sensor described is able to provide the IP address to the detector. Another example is the category "object" that contains the two characteristics "name" and "ID." Combined with the IDS scope "process," these characteristics denote the fact the sensor is able to capture the name and the ID of processes.

When describing a sensor, the proposed scheme enables us to systematically and concisely identify the data items a sensor is able to provide to a detector.

3.1.2.2 Detector characteristics

The description scheme for detectors is more complex than the one for sensors because their task is more complex and may vary far more than that of a sensor. For their description we also use a small number of IDS-scope-independent characteristics. The most important of these are the ones that reflect whether the detector operates in a knowledge- or behavior-based manner. These properties have been adopted directly from the IDS classification by Debar *et al.* [DeDaWe00, DeDaWe99].

One part of the IDS-scope-dependent characteristics describes the data pre-processing capabilities of the detector. Here it is described whether an IDS is able to filter observations based on data items such as the IP address and whether the detector is able to normalize data items. An example for data normalization is the decoding of an http URL that uses hexadecimal encoding of characters appearing in the URL as it is often used to obfuscate attacks [PtaNew98, RFP00].

The other portion of IDS-scope-dependent detector characteristics describes the actual analysis capabilities of a detector. We distinguish the two high-level categories "instance part analysis" and "instance analysis" (see Figure 5). The semantics of a so-called "instance" is defined when it is combined with an IDS scope. For example, considering the IDS scope "IPv4," an instance denotes a complete IP PDU and an instance part denotes a fragment of a fragmented IP PDU. Applying the IDS scope "process," an instance denotes a process and an instance part stands for a thread. Making this distinction between instances and instance parts is necessary because certain detectors are only able to analyze an observation when it is available in its complete form whereas others may also be able to analyze individual parts of it. Yet other detectors may be able to analyze instance parts only. However, apart from this difference the hierarchies beneath the two categories are identical and are composed of what is called the "analysis level" and "analysis techniques." The entire hierarchy is illustrated in Figure 12.

We use three different analysis levels to express the degree to which a detector attempts to analyze an instance or an instance part:

- Basic analysis: This level expresses the fact that a detector has a basic awareness of the instance or instance part, i.e., it is able to associate an observation to a specific IDS scope.
- Logic verification: The detector is able to verify whether the observation conforms to the specification. An example is the syntax verification of an http request.
- Semantic verification: The detector is able to determine the potential impact of its observation. An example is the detection of a syntactically correct http request that attempts to retrieve confidential information.

In order to achieve any of above analysis levels, detectors use analysis techniques such as regular expression matching, etc. Note, however, that for a given attack a detector is only able to achieve any of above analysis levels if all pre-conditions are met. This means that all the required data must be available and that the detector must be able to perform all the required pre-processing steps such as the normalization of URLs.

For our scheme we have identified two kinds of analysis techniques. The "general analysis techniques" denote techniques that can be applied to individual instances as well as across multiple instances and instance parts:

- Timing analysis: These characteristics reflect the detector's ability to draw conclusions from instance properties such as their duration or the time period of observation, e.g., specific hours.
- Data analysis: These characteristics describe a detector's ability to analyze data by applying techniques such as string matching or regular expression matching.

The "cross-instance analysis techniques" describe techniques that can only be applied across multiple instances or instance parts:

- Sequence analysis: This category covers the various techniques a detector can employ to discover sequences in its observations. It is, for instance, used to describe whether a detector is

able to recognize sequences at all, and if so, whether it is able to do so tolerating errors in the sequence.

- Statistical analysis: This category is used to describe the characteristics of statistical analysis methods that detectors may use. It is, for instance, used to describe whether a detector uses sliding windows, a decay function or both.

Figure 12 provides a simplified view of the hierarchy of characteristics that are used in the IDS description scheme for describing instance- and instance-part-related detector characteristics. The round-edged boxes represent the analysis techniques.

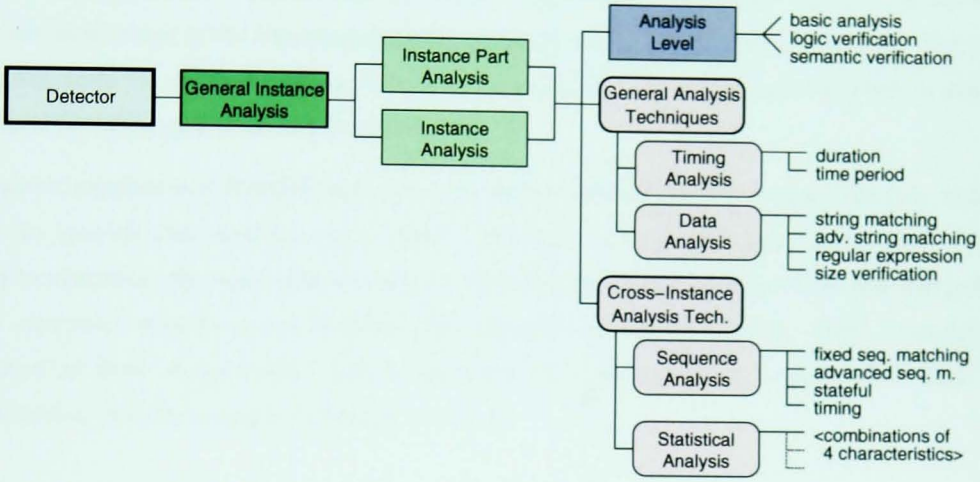


Figure 12—Simplified hierarchy of instance- and instance-part-related detector characteristics

3.1.3 Creation of IDS descriptions

The IDS description scheme introduced enables a detailed description of IDSs. In order to ensure that the resulting descriptions are not only concise but also repeatable, the creators of IDS descriptions have to obey the following rule:

Rule: *Every IDS characteristic must be expressed using the highest-level IDS scope applicable.*

This rule is necessary because it is possible to create *valid*, but possibly incomplete, IDS descriptions that do not use the highest-level IDS scope applicable for each IDS characteristic. It thereby prevents ambiguities such as the version-specific description of http protocol version-independent IDS characteristics. Such characteristics should be described at the IDS scope "http," but could as well be described twice: once for the IDS scope "http version 1.0" and once for "http version 1.1." Although such descriptions may be correct and complete at the time they are created, they may be rendered incomplete as the scheme evolves and new IDS scopes are added.

In the context of this work we have created descriptions for a number of IDSs. In the following we discuss an abbreviated example description of WebIDS [Almgre99] (for a complete description see

Appendix C.4). All the IDS characteristics mentioned are highlighted with double quotes. WebIDS is a comparatively simple IDS that is aimed at the detection of attacks targeted at webservers.

Example: *The sensor component of WebIDS, i.e., the parser that is used to parse the webserver access log file, retrieves all the information that is written to a CLF log file [Weinma98]. The information source type is therefore "application log." This means that the detector portion is provided with almost the entire information associated with a web request, including the URI (universal resource identifier), IP source address etc. The URI is represented as an "argument" sensor characteristic of the IDS scope "http." If basic http authentication is used, the log file even includes the user name, which is represented by the sensor characteristic "object name" for the IDS scope "user" (see Figure 10). Note, however, that it does not include any of the http headers or the data that is sent by an http POST request. This is data sent by the client, i.e., browser, on the lines after the actual request and includes information about the virtual host[6], cookies, data from submitted forms etc.*

The detector component of WebIDS uses several modules to analyze the http request. The most important one is the module that analyzes each "http" "instance" at the "semantic" level using "regular expression matching" for signs of known attacks. Moreover the detector contains a module that performs certain statistical analysis across multiple http requests, i.e., across multiple "http" instances. The description of these characteristics will be discussed at a later stage as further details need to be introduced first (see first example in Section 5.3.3.3.2).

3.2 Description and classification of attacks

Attacks can be classified according to numerous criteria—depending on purpose of the attack classification (see Section 2.2.4). The attack classification that we use in this work is based on descriptions of attacks and differentiates among classes of attacks based on attack properties that are relevant with regard to their detection by an IDS. The attack description scheme that we propose uses IDS characteristics to express the requirements an IDS has to meet in order to be able to detect a given attack. The resulting descriptions capture the nature of the attack but not all of its details. As a result, descriptions of different attacks may be identical—thereby identifying and defining an entire class of attacks. Hence, an attack class is defined by the set of all attacks having the same attack description, i.e., attack class description.

Example: *There exist many different URL-based buffer-overflow attacks [CA1301, CVE002100, CVE087499] against webservers. The description of any such attack will be identical even though the attacks may target different vulnerabilities that exist in different webserver products and therefore use entirely different overflow strings. However, from an IDS perspective the requirements for detecting any member of this class of attacks are the same.*

---

[6] A webserver may serve many different sites using the same physical host and server processes. The so-called http header field "Host," which is sent after the actual http request, determines the site to which the request is targeted.

Based on these considerations a two-step process for classifying attacks can be defined:

1. Description of attack: The attack to be classified is described using the scheme presented below.
2. Classification: If there exists a description of an attack class that is identical to the newly created attack description, the attack is a member of this previously described class. If no such attack class description exists, the newly created attack description identifies a new attack class.

The actual description of an attack, i.e., an attack class, is composed of *attack class description building blocks* that each describe a specific aspect of the attack. The re-use of already existing building blocks

- limits the effort required for describing an attack, and thereby also
- ensures consistency and maintainability among the descriptions of different attacks that belong to the same class.

New attack class description building blocks are created and added to the library of already existing building blocks whenever a new attack cannot be described using the already existing building blocks. As the number of readily available attack class building blocks increases, the number of additionally required building blocks that are required for each newly created attack class description decreases.

In the following sections we outline the components involved in this attack classification and description scheme. The details, however, are developed and described in Chapter 6.

3.2.1 Attack class description building blocks

Each attack class description building block only describes a very specific aspect of an attack class. It does so by specifying the characteristics that are required of an IDS in order to analyze the aspect described. It is, however, important that the description should cover all conceivable approaches an IDS could take for analyzing the attack. Practically, these descriptions have to be extended as new approaches for analyzing the described attack aspect are discovered. Therefore all existing attack class descriptions that make use of a given attack class description building block are extended implicitly whenever the building block is extended.

In general one can distinguish two categories of building blocks that formulate the requirements an IDS has to meet in order to

- take notice of the attack, i.e., to verify whether all required data is available, and requirements that need to be met in order to
- analyze the actual attack.

As a further measure to ensure consistency across attack class descriptions and to limit the effort of creating further attack class descriptions, the building blocks are formulated at the highest possible IDS scope (see above rule). When used for describing an attack class, the building blocks can also be used for describing the attack class at any IDS scope that is beneath the IDS scope of their specification. Moreover, the description of a building block may itself make use of other building blocks.

Example: *We have created a building block that specifies the requirements for analyzing the data of an application layer transaction. When creating an attack class description we can then specify whether the building block is to be used for http or smtp (mail) etc.*

This building block itself may make use of other building blocks. For instance, in the case where the information source is a network packet sniffer (see Figure 11), the building block may require the IDS to be able to provide the appropriate transport layer data. The building block that specifies the requirements for the transport layer data may then set a similar requirement for the data of the next lower-level protocol etc. However, the building block specifying the application layer requirements has to cover also other sensor types. For instance, in the case of an IDS that operates based on data retrieved from log files, the application-layer-specific requirements are different and do not include requirements for transport layer data.

As a further example we consider the description of an attack class that makes use of attack class description building blocks.

Example: *Consider the class of URL-based meta-character attacks. These attacks typically exploit shortcomings in the escaping of input parameters that are submitted to CGI scripts as part of the URI [CA0696, CA0797]. The attacks permit for instance the reading of private data. Using the available building blocks, this attack class can be described by only two building blocks:*

1. *Application layer control data for http: This building block is equivalent to the one described in above example. It verifies whether the required input data, i.e., the URI of the http request, is available.*
2. *Suspicious string in the http URI: This building block verifies whether the detector is capable of analyzing http request at the semantic level and whether it can do so using some form of string detecting, e.g., regular expression matching.*

3.2.2 Systematic creation of attack class variants

Earlier we mentioned that adversaries often attempt to obfuscate their attacks in order to evade detection. We take into account such obfuscation techniques by including hooks in the descriptions of attack class building blocks that allow us to analyze them under the assumption that the attack class has been obfuscated in a particular manner. Our scheme formulates so-called variations in a similar manner as the attack class building blocks by specifying the IDS characteristics that are required to deal with the obfuscation applied to the attack. In the context of this work we have created descriptions of seven variations that can be applied to attack class descriptions. It is even possible to apply several of them concurrently as long as they do not interfere by making use of the same system or protocol features in a different manner.

Note that a new attack class is created whenever a given variation or combination of variations is applied to an attack class. Each variation extends the description of the initial attack class, which causes the set of

IDS characteristics required for analyzing the initial and the new attack class to be different. Finally, note that multiple variations may be applied to an attack class concurrently.

Example: *A variation that we described is the IP fragmentation obfuscation technique introduced in the example of Section 1.1. This variation formulates the requirements an IDS has to meet in order to reassemble fragmented IP PDUs. The variation can be applied to all attack class building blocks that involve the processing of IP PDUs. For instance, if we analyze a network-based IDS that is not able to reassemble IP PDUs the analysis will reveal that this IDS is not capable of detecting attack classes that involve this obfuscation technique. Note that based on the information source type the scheme also recognizes that an IDS such as WebIDS is not affected by this variation because it does not have to process raw IP PDUs but instead retrieves its information from a log file.*

3.3 Putting it together: analyzing IDSs

In the previous two sections we have developed a scheme for describing IDSs and a scheme for classifying and describing attacks. In this section we show how IDS and attack class descriptions feed into an analysis method that determines the attack classes that a given IDSs has the potential of detecting. The IDS analysis method determines the type of alarms, i.e., generalized alarms, that an IDS potentially generates. Figure 13 illustrates this two-step process and shows a more detailed view of the IDS analysis part of Figure 1. Having described our approach in some detail, we will then provide an overview of the RIDAX prototype, which implements the approach. Finally we outline the issues involved in validating the results as produced by our RIDAX prototype. The details of the entire analysis process will be presented and developed in Chapter 7.

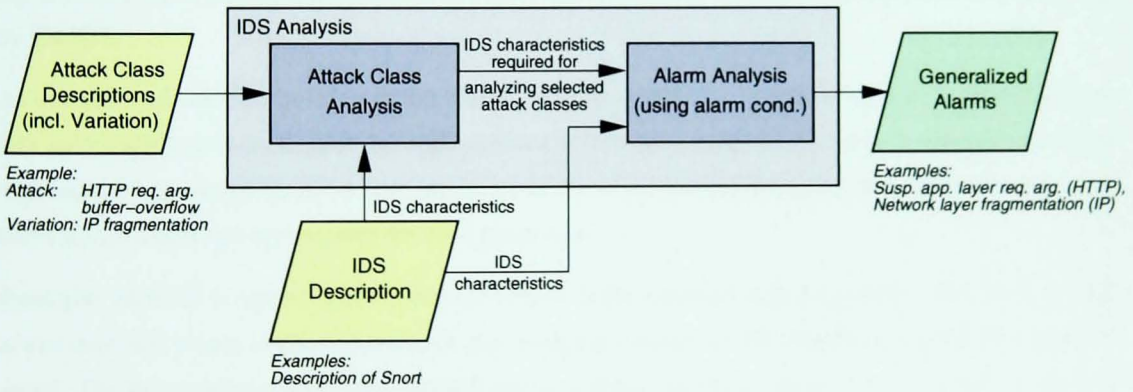


Figure 13—The two-step IDS analysis process (including examples)

3.3.1 Attack class analysis

Attack class descriptions may describe many different approaches how an IDS could analyze a given class of attacks (see Section 3.2.1). The attack class analysis step systematically explores all these possibilities while taking into account the characteristics of the IDS under evaluation. Note that at this stage the approach determines the manner in which the evaluated IDS processes, i.e., analyzes, attacks

from a given class. It is therefore not yet determined whether the IDS recognizes the fact that the analyzed class of activities actually represents a class of attacks. Neither does this step determine the alarms that the IDS is likely to generate. These are identified in the subsequent "alarm analysis" step. For a given attack class it is therefore determined whether or not the IDS is capable of analyzing the attack class considered. In some cases it may be found that the IDS offers multiple approaches for analyzing a given attack class. In such cases the analysis continues for each result independently with the alarm analysis step. In addition to the result whether the IDS considered is able to analyze a given attack class, the analysis also determines the approach the IDS uses to analyze the attack class by providing the set of IDS characteristics that were required of the IDS during the analysis. This set of IDS characteristics then serves as input to the alarm analysis.

3.3.2 Alarm analysis

The alarm analysis step operates based on the set of IDS characteristics that the IDS analyzed had to possess to analyze the attack class considered. As additional input this step makes use of the IDS description and of *alarm conditions*. Alarm conditions are an integral part of this analysis step and formulate the condition under which IDSs potentially generate alarms of a given type. These conditions are expressed in a manner that is equivalent to the manner in which the attack class building blocks are expressed, but are independent of attack class descriptions. The difference, however, is that they operate on the IDS characteristics that were required from the IDS for analyzing a given attack class rather than on the IDS description. In other words the second step verifies whether in the set of IDS characteristics obtained in the first step there exists a subset of IDS characteristics that matches any of the alarm conditions. If such a subset can be found, attacks belonging to the considered attack class are detectable by the IDS.

As our approach operates based on attack classes, it is impossible to determine the specific alarms that an IDS potentially generates. Instead the approach determines the generalized alarms, i.e., the type of alarms that might be generated by IDSs. The semantics of these generalized alarms is determined by the set of IDS characteristics that are required for their generation.

Example: *WebIDS is capable of detecting URI-based buffer-overflow attacks against webservers. For the actual detection it uses regular expressions for verifying whether a URI matches a signature of a known attack. The first analysis step reveals the IDS characteristics that were required from WebIDS to analyze attacks of this class. In the second analysis step it is verified whether in the set obtained from the first step there exists a subset of IDS characteristics that matches any of the alarm conditions. For this example this step reveals that WebIDS has the potential of generating a generalized alarm. This generalized alarm is generated mainly because of the use of the regular expression matching capability. It therefore reveals that WebIDS reports this class of buffer-overflow attacks by means of a generalized alarm indicating that a suspicious string in an URI has been observed. In fact both the alarms WebIDS generates for buffer-overflow attacks and the alarms it generates for meta-character-based attacks are represented by this very same generalized alarm, i.e., they are semantically identical.*

Under certain circumstances alarm conditions may require IDS characteristics that were not required for analyzing the attack class, but that are required for the IDS to be capable of generating the generalized alarm described by the alarm condition. For instance, if the semantics of a generalized alarm is such that it includes the source IP address in the alarm message, the alarm condition must formulate this requirement in terms of IDS characteristics.

Finally it should be emphasized that the alarm conditions are independent of specific IDS descriptions and attack class descriptions. They function solely based on IDS characteristics. The 14 alarm conditions we used in our experiments were created in the same context within which we created the attack class descriptions. For each attack class that we described, all conceivable generalized alarms that known IDSs might generate were identified and expressed in terms of alarm conditions. As a result of this approach, the alarm conditions have the inherent advantage that they also identify false alarms an IDS might generate.

3.4 RIDAX prototype

We have conducted a significant number of experiments using the RIDAX prototype, which has been implemented in Prolog [Diaz00] and makes use of a database [MySql] and some additional tools [Apache, phpAdm]. The IDS descriptions are stored in database tables that are accessed during the analysis process. The attack class descriptions and the attack class description building blocks are expressed in the form of prolog rules. The analysis process can thereby greatly benefit from the backtracking capabilities of Prolog in order to determine the manner in which an IDS analyzes a given attack class. The analysis results, i.e., the IDS characteristics used for analyzing a given attack class and the generalized alarms generated, are also stored in database tables.

In the context of the RIDAX development we have created descriptions for 27 attack classes and 23 attack class building blocks. After having applied up to two out of seven available variations concurrently to these attack classes, we obtained a total number of 498 distinct attack classes. The attack classes described were selected based on statistical data derived from VulDa [DacAle99] and were used to analyze different IDSs and different configurations thereof. These experiments are described in detail in Chapters 7 and 8.

3.5 Validation of the approach

RIDAX has been used to analyze five different configurations of four different IDSs. The results obtained enable us to outline the manner in which evidence can be provided that the predictions made by our IDS analysis approach are correct. However, a complete verification of the results produced by RIDAX or our approach in general is a major issue that exceeds the scope of what can be covered by this work:

- Environment: An environment comparable with that of the Lincoln Lab experiments would have to be created in order to test different types of IDSs in a heterogeneous environment.

- Attacks: Equivalence-class-testing requirements would need to be met (see Section 2.2.2). This means that for each attack class, including the ones determined when variations are applied to attack classes, a representative number of attacks is required for validation purposes. Assuming this number to be as small as five and considering the 498 attack classes to which the 27 attack classes descriptions expand when variations are applied, several thousand individual attacks would have to be implemented and exercised to validate the results RIDAX produced. Moreover these tests would need to be repeated for each IDS considered.

In other words, the same reasons that let us pursue the approach presented here render its validation a challenge. However, evidence can be provided that it is possible to predict the detection capabilities of IDSs and that these predictions can be made by means of a combined analysis of IDS descriptions and descriptions of attack classes. The validation of this claim can be outlined as follows:

1. First we have used an existing IDS model to develop an IDS description framework. This was accomplished in a systematic fashion by focusing on the identification of IDS characteristics that are relevant to the detection of attacks. The basis for this work was the analysis of attacks pursued in the context of the VulDa work [DacAle99] with respect to attack properties that are observable.
2. In the second step we used the resulting set of IDS characteristics for creating descriptions of attack classes. The descriptions are organized in a hierarchical fashion using attack class description building blocks and focus on observable aspects of attacks. Based on this description scheme we have defined a procedure that enables the unambiguous classification of attacks and that thereby ensures that there exists only one valid description per attack class.
3. Finally it should be verified whether the actual IDS analysis provides correct results. Such verification can be achieved by comparing the output produced by RIDAX and the output produced by IDS implementations for specific attack classes and attacks that are members of these classes. In the following we outline such a validation by means of some examples.

Example: *We consider two http-related attacks. One belongs to the class of meta-character attacks that we already introduced in Section 3.2.1. An example for a vulnerable CGI script is test.cgi [CA0696]. The other example belongs to the attack class that represents http-header-related buffer-overflow attacks as mentioned in Section 3.1.3. An example for such an attack exploits a vulnerability [CVE084800] in the IBM Websphere application server plugin for the apache webserver [Apache]. Although both attacks concern webservers, they are disparate as they target different server components and involve different types of vulnerabilities (that require disparate exploitation).*

In the following we consider the IDSs WebIDS [Almgre99] and Snort [Roesch99]. In the case of Snort we consider a simple configuration in which IP fragments are not processed. If in addition we consider the possibility that attacks may be obfuscated by means of IP fragmentation we obtain the following table:

Table 2—Alarms generated by WebIDS and Snort including generalized alarms (in brackets)

IDS / Attack	http Meta-character attack	Obfuscated http meta-character attack	http header buffer-overflow	Obfuscated http header buffer-overflow
WebIDS	test.cgi attack (susp. string in URI)	test.cgi attack (susp. string in URI)	Not detected (no generalized alarm)	Not detected (no generalized alarm)
Snort	test.cgi attack (suspicious string in URI)	Not detected (no generalized alarm)	Websphere http header buffer-overflow (suspicious string in http upstream data)	Not detected (no generalized alarm)

In brackets we provided the description of the generalized alarms that RIDAX generates for the respective attack classes. The alarms are of course not identical, but the generalized alarms represent higher-level descriptions of the alarms generated by the IDS implementations. Similar tables could be created for other IDSs and other attacks.

As mentioned in the introduction, there exist circumstances where the predictions made by RIDAX are not met by the actual IDS implementations. Such cases must be investigated carefully in order to determine whether it is the IDS implementation that does not behave as specified, whether a signature is missing etc., or whether it is our approach that produces wrong results. In the course of the experiments that we conducted with RIDAX, we did not observe any differences that indicated a failure of our approach. Differences were, for instance, observed in the treatment and reporting of fragmented IP PDUs (see the investigations by Marty [Marty02, p. 66] that were presented briefly in the example of Section 1.1). In this example the differences are caused by the ad-hoc manner in which different IDSs decide whether or not to report observed IP fragments as suspicious.

3.6 Discussion

The approach presented analyzes IDS designs at a conceptual level and determines their potential of detecting given classes of attacks. Its major advantages are the following:

1. Support of IDS designers: IDSs do not need to be implemented before the analysis can be made. The predictions made by our approach enable IDS designers to address weaknesses of IDSs at an early stage in the design process.
2. Class-level results: The results produced by our approach predict the behavior of IDSs with regard to entire, clearly defined, classes of attacks. This is not only important for the design of IDSs but also for the specification of the requirements that they have to meet. Class-based requirements are preferable to requirements that are based on the enumeration of individual attacks, because the latter would be outdated within days. New vulnerabilities and attacks are discovered on a daily basis [SecFoc].
3. Test environment: Our approach operates based on descriptions of IDSs and attack classes and does therefore not require the set-up of a test environment. As, for instance, observed by Lippmann *et al.* [LFGHKM00, LHFKD00] and others, such test environments may be of considerable complexity (see also Sections 2.3.3 and 2.4.1).

However, our approach also has some disadvantages:

1. Discrepancies between IDS design and its implementation: If the actual implementation does not obey the design and specification of the IDS, the behavior predicted by our approach may not correspond to the behavior of the IDS implementation.
2. Limitations of the IDS description framework: Our IDS description framework is highly flexible and extensible, and enables detailed and concise descriptions of IDSs. However, IDSs often make use of highly ad-hoc methods to decide whether an alarm should be raised. Even though our model might be expanded to represent such highly IDS-implementation-specific characteristics, their description remains difficult, not least because they are rarely documented.

Moreover note that an exhaustive validation of the approach is impractical. It is, however, possible to provide reasonable evidence that the results produced by our approach are correct by conducting a comparatively small number of experiments using existing IDS implementations and comparing the results obtained with the results predicted by our approach. The manner in which this can be achieved has been outlined in Section 3.5 and will be discussed in more detail in Chapter 7.

Finally, it should be noted that although the IDS description framework and attack classification are the result of a systematic analysis of IDSs and attacks, the schemes presented may not be able to represent future ID technologies. In such cases extensions of the schemes will be necessary, but simple to implement because the schemes themselves facilitate such extensions.

3.6.1 Facilitating a systematic IDS design process

Our IDS analysis approach supports the IDS design process by making predictions about the results that a given IDS design is able to provide. For an IDS designer this is helpful in determining whether the considered design meets the specification at an early stage of design process. Here our approach provides the advantage that the specification can be expressed in terms of clearly defined attack classes that the IDS has to be able to detect and, if required, how these attacks are to be reported. The approach even goes a step further by systematically analyzing the IDS design for large numbers of classes of obfuscated attacks—an analysis that would involve an enormous effort and costs when done with IDS implementations.

However, it has to be made clear that the task of making IDS design proposals remains the responsibility of the IDS designer, i.e., our approach is not capable of making IDS design proposals based on the specification of an IDS. It merely helps an IDS designer to verify whether the proposed design fulfills the specification. In practice this is likely to result in an iterative process in which an IDS design is extended by additional components until the entire design meets the requirements set by the specification. The repeated analysis of complete IDS designs would be necessary because individual IDS components might influence each other (see for instance the propagation of effects caused by variations as discussed in Section 3.2.2).

3.6.2 Generalizing from attack classes to activity classes

The attack classification and description scheme presented in this work can equally well be applied to classes of benign activities, i.e., activities in general. This is possible because the scheme describes observable aspects of attacks (and activities) and therefore does not take into account the *intent* of activities because IDSs are not able to observe or determine it. Figure 14 illustrates the proximity of attacks, i.e., malicious activities, and benign activities.

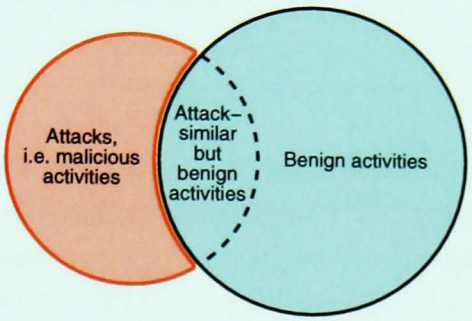


Figure 14—Venn diagram illustrating the proximity of attacks and attack-similar benign activities

The fact that it is possible to generalize from attacks to activities does not have an immediate impact on this work because our approach focuses on the analysis of IDSs with regard to classes of attacks. It is, however, worth mentioning the possibility of generalizing from attacks to activities because it might be relevant to further applications such as the assessment of IDS combinations as discussed in an example in Chapter 8. There we shall pay particular attention to classes of benign activities that may potentially cause the generation of false alarms, i.e., classes of benign activities that are "similar" to attacks when considered from the IDS perspective.

3.6.3 Assessment of IDS combinations

The results produced by a tool such as RIDAX may serve as the basis for a series of applications other than the design of IDSs. One example is the assessment of the combination of IDSs or, more precisely, the measurement of gains that are potentially achievable by combining IDSs. This can, for instance, be used to support the design process of an ID architecture. An ID architecture may consist of several diverse IDSs that distributed over the network surveyed. The alarms generated by these IDSs are collected and combined, i.e., correlated, in order to extract the maximum of information provided, while minimizing the total number of individual alarms (most importantly false alarms) that are presented to an human security officer.

For such an application metrics need to be defined that measure the quality of the information the IDSs provide. A property that one might measure in such a context is, for instance, the difficulty of discriminating between true and false alarms or the set of attack classes covered from a larger set of

attack classes. However, such an undertaking requires a number of extensions to the work as it has been presented here. For instance it is necessary to

- include classes of benign activity in addition to the attack classes in order to be able to investigate the problem of false positives in more detail, and
- investigate the mapping between attack classes and generalized alarms for combinations of IDSs.

In Chapter 8 we illustrate this possible application further by developing example metrics and methods that are suitable for the assessment of IDS combination.

3.7 Conclusion

In this chapter we have provided an overview of our approach to IDS analysis. In contrast to existing approaches, where IDSs are evaluated based on benchmarks using actual attack implementations, our approach operates at the conceptual level of attack classes and uses descriptions of IDSs rather the actual IDS implementations for its analysis. It thus produces IDS analysis results of high generality. This makes it a useful tool for the designers of IDSs because it enables them to verify that their design meets the specification before the system has actually been implemented, i.e., early in the design process. Moreover, the fact that our approach operates based on clearly defined classes of attacks enables a more rigorous design process. Firstly, it allows the clear specification of requirements that the envisaged IDS regarding detected attack classes. Secondly, our approach uses these very same classes for its analysis, thereby enabling a rigorous verification of whether a given IDS design meets its requirements and allowing the identification of weaknesses in the design. Finally, our approach provides a high-level description of the alarms that the analyzed IDSs are capable of generating. These may serve as the basis for further work, such as the assessment of ID architecture designs that require the combined processing of the alarms that are generated by diverse IDSs.

Chapter 4 Intrusion detection system scopes and the categorization of attacks

An important part of the analysis of IDSs is the identification of a representative input set. Ideally the input set ensures that all factors relevant for ID are taken into account and exercised during the IDS analysis process. For our approach to IDS analysis, the input set may be composed of activities and their variants. Each of them represents an entire class of malicious or benign activities. In this chapter we compose a selection of activity classes that ideally represent the most relevant activities. Note, however, that in the remainder of this work we focus on classes of malicious activities, i.e., on attack classes. It is only in Chapter 8 where we will make use of both the malicious as well as benign activity classes identified in the following. The activity classes are identified in a systematic fashion by first introducing the *IDS scope* concept. This concept is used not only as part of the multi-dimensional *activity categorization scheme*, which is developed next, but also as a common underlying concept to assure consistency across the entire work. The activity categorization scheme developed is the one that we used to categorize a large number (358) of attacks taken from VulDa. Based on the statistical results derived from this *attack categorization*, we subsequently compose a representative set of 48 malicious and benign activity classes that we use throughout this work. The identified attack categories are of higher generality than the derived attack classes, i.e., each attack category covers multiple attack classes.

Note that the concepts described in the following all resulted from an iterative process, which was pursued in the context of the VulDa development and maintenance described in Appendix A. By categorizing attacks, weaknesses of the scheme were identified and eliminated. This categorization effort therefore represents a pragmatic attempt to identify categories of activities suitable to analyze IDSs. It is, however, by no means to be seen as a general-purpose categorization or classification scheme for attacks. Moreover note that it was motivated and inspired by the concept of *fault assumptions* as introduced by Laprie *et al.* [LaAvKo92] and briefly discussed in Chapter 2. A key element of their approach is that the factors that are believed to be relevant to potential causes of faults are identified systematically. In a next step one then considers the impact these factors and combinations thereof have in terms of faults, which corresponds well to our approach that develops a scheme geared at categorizing attacks based on properties that are potentially observable by IDSs.

At first glance this categorization only provides us with the set of malicious activities to be used for the analysis of IDSs. To address this issue we make the assumption that attack-similar activities can be found in the "proximity" of attacks (see Figure 14). This assumption seems to be valid, as the observable properties of a benign activity need to be similar to those of an attack if the benign activity causes a false alarm. We accordingly use our activity categorization scheme to identify classes of attack-similar (benign) activities by systematically selecting attack categories for which we then seek classes of benign activities that would be categorized identically or at least similarly. This is possible because benign and malicious activities may be assigned to the same or at least to very similar activity categories. However,

benign activities are not necessarily attack-similar simply because they were assigned to the same category as a malicious one. This is where we have to rely on our experience gained from developing and maintaining VulDa as well as developing IDSs. Based on this experience it becomes possible to identify classes of benign activities that are similar to attacks and that have the potential of causing false alarms. We therefore use results obtained from the categorization of attacks as an underlying grid that imposes structure on our approach and on the selection of input data for the experiments described in Chapter 8. As the classes of attacks considered are identified in a systematic fashion, we consider the identification of classes of attack-similar activities to be reasonably systematic as well.

Example 1: *It was observed that an IDS capable of detecting TCP SYN flooding attacks [CA2196] on the network may generate a false alarm if a user, using http v1.0, visits a web-page that contains many images. Such a WWW access triggers the initialization of many TCP connection handshakes in a very short period of time, which may be confused with a TCP SYN flooding attack by an IDS.*

Example 2: *Another example involves SMTP (simple mail transfer protocol). If an IDS is not capable of tracking the state of an SMTP session, mail-message data might be confused with SMTP commands. For instance we have observed that IDSs confuse the words "DEBUG" and "WIZ" if contained in the body of mail message with (old) attacks against sendmail [CA1190] that misuse the SMTP DEBUG and WIZ commands.*

4.1 IDS scopes

The IDS scope concept is central to this work. Its development was initialized by our attack categorization effort. The motivation, however, was to identify a common underlying concept to assure consistency across the entire work. This primarily includes the goal of enabling the description and classification of various items at different levels of detail. Given this requirement, a hierarchical tree structure was chosen as the underlying component. This is also the reason that the IDS scopes should cover characteristics of greater generality that per se are not required for the categorization and classification of activities. The concurrent effort of describing IDSs, for instance, has also influenced the concept of IDS scopes. Hence, IDS scopes are a pragmatic and empiric concept that can be used for describing IDSs and activity classes.

4.1.1 IDS scope tree

The use of a hierarchical tree structure enables us to describe, for instance, IDS capabilities at a high degree of detail as well as at a high level of generality. This is required as IDSs may offer diagnosis capabilities applicable to any application layer protocol. Concurrently, they may also offer capabilities that are only applicable to a few specific protocols such as http.

Example: *Using IDS scopes one express the ability of an IDS to scan application layer data for suspicious strings (e.g., Snort [Roesch99]), as well as the fact that another IDS is only capable of performing this type of analysis on http requests (e.g., WebIDS [Almgre99]).*

The IDS scope tree described in the following has been derived from common system-partitioning and layering concepts such as the OSI model. The purpose of its sub-trees is to group more specific IDS scopes together. However, the tree was kept as simple as possible by focusing only on ID-relevant issues.

As illustrated in Figure 9, we have identified the following three top-level IDS scopes:

- Network: The *network* sub-tree corresponds to a simplified version of the OSI model. We decided not to take the presentation and session layers into account because IDSs generally treat them in the context of the application layer.
- Host: It is less straightforward to identify the IDS scopes on the *host* level because here no layering concept as it exists for the network stack exists. We therefore focus on objects that are of interest to ID and that are observable.
- User: Finally we have defined the top-level IDS scope *user*, which is not so much of relevance for the classification of activities but for the description of IDS capabilities. It will, for instance, enable us to express the ability of an IDS to relate its observations to a specific user, including the possibility that the user is using multiple sessions or even multiple user IDs concurrently.

Note that we split middleware into two IDS scopes, because it cannot be assigned to either only the host or network sub-tree. One middleware IDS scope is used to describe an IDS' capabilities with respect to activities observable on the network. The second covers middleware functionality used only on the host.

Figure 9 shows the IDS scope tree as it is used throughout this work. IDS scopes surrounded by boxes represent the higher-level IDS scopes, whereas those shown at the far right are lower-level IDS scopes that mostly represent implementations. In order to limit the complexity of the respective results we generally use the most high-level, but still applicable, IDS scopes. We do so for instance in the next chapter where we use the IDS scope tree to refine the semantics of IDS characteristics. However, the semantics of IDS characteristics that have been defined at a given IDS scope need to remain the same for any of the lower-level IDS scopes. For instance, if an IDS is said to offer a given capability at the application layer protocol scope, this must also be true for all lower-level IDS scopes such as http, SMTP and DNS[7].

In Appendix C.1 we provide more detailed definitions of all higher-level as well as of numerous lower-level IDS scopes used in this work and shown in Figure 9. Note, however, that the lower-level IDS scopes discussed in this work merely represent examples that illustrate the generic nature and extensibility of the proposed IDS scopes concept. For instance, protocols such as SNMP (Simple network management protocol) or wireless LAN protocols (e.g., IEEE 802.11b) are not explicitly included in the following discussions, but are covered implicitly by higher-level IDS scopes.

[7] DNS: Domain Name Service; the directory service used on the internet to translate host names into IP addresses.

4.1.2 IDS scope attributes

Considering an IDS scope such as the application layer one can easily identify lower-level IDS scopes such as http, DNS and FTP that differ significantly in the type of service they provide and, even more importantly in the way they function. For the analysis performed by IDSs the service offered is of relatively low importance. The way these application layer protocols function is by far more important. These functional differences need to be respected when describing IDSs as they may significantly influence the semantics of IDS characteristics used to describe the capabilities of an IDS. Therefore these functional differences also have to be taken into account in classifying activities. Because the IDS scope tree is not suitable to capture such differences we introduced so-called IDS scope attributes that enable us to characterize IDS scopes in more detail.

It is the goal of IDS scope attributes to allow the classification and description of items at a high level without having to descend to the level of specific protocols. It thereby becomes possible to express properties such as whether a specific application layer protocol can be run on top of a connectionless service, or whether it supports single or multiple *transactions* within one session. Note that for the purpose of this work we relax the formal definition of transactions as, for instance, known from database systems. We relax the definition to the extent that we consider an application layer protocol to support multiple transactions within a session if a protocol sequence can be repeated. This is the case if, for example, multiple mail messages can be sent within the same session, multiple documents can be transferred within one session etc. Examples: SMTP, http version 1.1, FTP. In describing IDSs, the IDS scopes enable us to create generic descriptions. We may, for instance, express that an IDS is capable of applying a simple pattern matching technique to application layer protocols operated on top of a connection-oriented service. In this way it is possible to describe IDSs in a highly generic fashion without having to describe the IDS at the level of specific protocols.

Examples: *When classifying an activity it is important to distinguish between protocols that use a single instance of a lower-layer service (e.g., http) and protocols that rely on multiple instances of lower layer services (e.g., FTP). This distinction is necessary because such differences may have a significant impact on the complexity of the analysis to be performed by an IDS. The service offered is of limited importance for the analysis performed by the IDS. Another, even simpler example is shown in Figure 15, which shows ICMP and UDP as being connectionless protocols, TCP as being connection-oriented, and UDP and TCP to be supporting an addressing scheme (i.e., ports).*

After having identified the IDS scope tree, we examined the higher-level IDS scopes in terms of functional properties that matter to ID. On the networking side, it is clear that we wish to distinguish whether a protocol is connection-oriented or based on datagrams, etc. On the host side, the attributes identified are of higher diversity and are, for instance, used in the context of inter-process communication. However, the precise definition of these attributes is important when implementing a tool such as RIDAX, but only of limited importance when explaining and developing concepts. Therefore we do not discuss all these attributes here but provide their definitions in Appendix C.2.

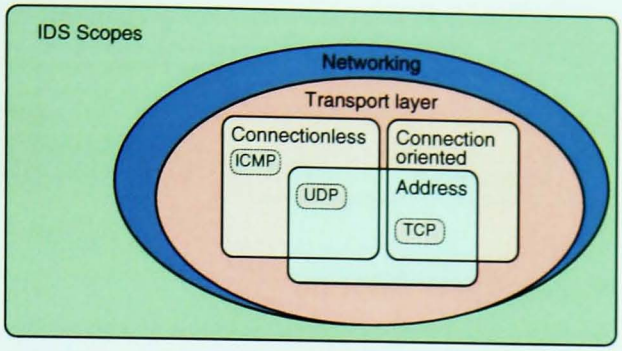


Figure 15—An example how IDS scope attributes can be used to refine transport layer IDS scopes

4.2 Activity categorization scheme for attack-like activities and attacks

Having introduced the IDS scopes concept, we may now develop our categorization scheme for activities. As underlying concept we first define a simple system model to describe the activity to be categorized. We do so by using existing concepts such as the IDS scopes just introduced and a simple object model. Note, however, that in order to simplify the attack categorization-task described in the following we do not consider the IDS scopes as a tree structure. Instead we use a selection of IDS scopes that can be found at a fixed level of the IDS scopes tree—generally at the level just above leaf entries such as specific protocols (see Figure 9).

4.2.1 System model for activity categorization scheme

In developing our categorization scheme for activities the goals of the categorization need to be kept in mind, i.e., the scheme should reflect activity characteristics relevant to the analysis performed by IDSs. Furthermore the choice of the categorization criteria has to be consistent with the goals of this categorization, and should not be too broad so that the effort remains of limited complexity, but still well focused. To achieve this we have to consider both IDSs and activities. This categorization scheme enables us to categorize activities based on criteria that determine an IDS's ability to detect a given activity, i.e., properties of the activity that are observable. This also means that we do not consider properties, such as the intent of the adversary, which are not observable, and would merely cause confusion.

Considering an activity one can distinguish two main types of characteristics. First there are the *static activity characteristics* that can be viewed as representing objects required to exist prior to occurrence of the activity. Second there are the *dynamic activity characteristics* that denote the transient appearance of the activity. Based on these two main types of activity characteristics we have identified a total of five sub-types—two sub-types of static activity characteristics and three sub-categories of dynamic activity characteristics. These static and dynamic activity characteristics were identified using a simple system model geared towards describing the observable aspects of activities as shown in Figure 16.

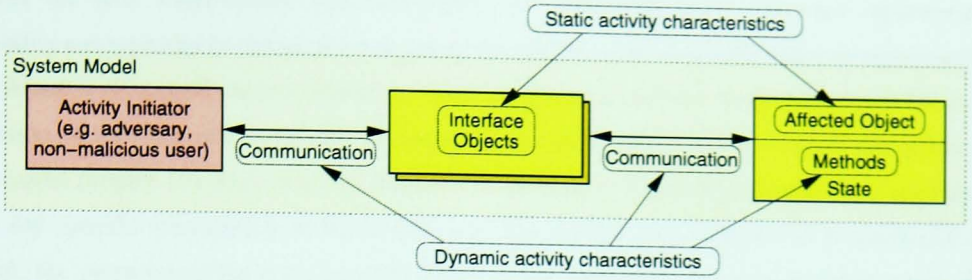


Figure 16—System model used to categorize activities

Looking at Figure 16 from the left to the right, we first find the *activity initiator*. The activity initiator is in most cases a human user who causes an activity to occur by his/her actions. These actions may be of malicious or non-malicious intent.

Once the activity initiator has initiated an activity, it is potentially observable at numerous *interface objects* until it finally reaches its destination, the *affected object*. It is not always possible to observe attacks at interface objects. For instance, it is not possible to detect attacks on the network if the data transferred is encrypted. However, it is often possible for an IDS to observe the signs of an activity on the affected object itself. These considerations lead to the introduction of the so-called *static activity characteristics* comprised of one or more interface objects and the affected object. Formally the activity initiator object should be considered as being part of the static activity characteristics. However, the activity initiator object is generally not directly observable by an IDS, which is why it is not included in the scheme developed in the following.

Another static element shown in Figure 16 that is not included in the static activity characteristics is the *internal state* of the affected object. It is not taken into account for the activity categorization scheme because this work focuses on real-time IDSs, i.e., on IDSs performing continuous monitoring, which generally are transition-based IDSs (see also Figure 7 and [DeDaWe00]). State-based IDSs in general perform only a periodic analysis of the system to monitor. This means that the state of a system i.e., an object, is inspected periodically for erroneous system states that indicate a fault.

Examples: *Typical examples are security scanners such as Nessus [Nessus00], ISS [ISSSca99], Satan etc., and system integrity checkers such as Tripwire [Tripw99]. The only state-based systems operating in real-time that we are aware of at the time of this writing are anti-virus systems. A well-known example of such a system is the Norton anti-virus product by Symantec [Symantec].*

In addition to the static activity characteristics, i.e., objects involved in an activity, we also consider dynamic activity characteristics, i.e., the transient appearance of activities. Here we have identified two main sets of characteristics that describe the dynamic portion of an activity. These describe the invocation of *methods* on the affected object, and the type of *communication* used. Furthermore, we have identified a set of additional *attributes* that refine the dynamic activity characteristics and are observable by an IDS. These attributes describe characteristics such as whether the input data provided is relevant to the attack.

Note that the term *vulnerability* was deliberately not mentioned here. The fault representing the vulnerability can typically be found either in one of the interface objects, in the affected object itself, or in the combination of several objects. However, when analyzing an activity for its aspects visible to IDSs, the location of the vulnerability is only of limited importance, which is why it does not appear in the system model (Figure 16). Moreover, the initiation of an activity threatening the security policy does not require any specific vulnerability to be present, e.g., the activity may represent an unsuccessful attack. However, the presence of the corresponding vulnerability determines whether an activity may lead to security policy violation, e.g., a successful attack or an intrusion.

Before further developing the notions of static and dynamic activity characteristics, note that our categorization scheme is not aimed at describing the path from the activity initiator to the affected object or the sequence of events involved. Such descriptions would exceed the goal of this categorization scheme by adding non-required information, thereby introducing unnecessary complexity. Instead we list the dynamic and static characteristics relevant to an activity, and define activity categories by the *set of activity characteristics* required to describe the activity. Other approaches to attack description mostly use languages that describe the intermediate stages of attacks. For our purposes these approaches are by far too expressive, and would therefore impose unnecessary complexity. Examples for such work are the STATL attack language by Eckmann *et al.* [EcViKe00] and the attack modeling work by Tidwell *et al.* [TLFH01]. STATL is part of the STAT tool suite [ViEcKe00], and describes the system states and state transitions that attacks drive a system into, respectively cause. Specific STATL constructs are required to develop the attack scenarios for the respective STATL IDS implementations[8]. These constructs depend on the attack, the operating system considered and the information source monitored (called "domain" by Eckmann *et al.*) Whereas STATL is designed for describing specific attacks such that the resulting descriptions can be used to actually detect them, the modeling work by Tidwell et al. operates at a higher level and is geared towards describing attack scenarios that may consist of several consecutive individual attacks.

4.2.2 Static activity characteristics

Although the location of the vulnerability is of limited importance to the detection process, it has a significant influence on the set of interface objects and the affected object needed to describe the corresponding attack. In fact, the vulnerability implicitly defines the avenues of possible attacks. Roughly speaking, such an avenue can be considered as the set of interface objects and the affected object describing the static characteristics of the corresponding attack, i.e., the activity. Furthermore the vulnerability implicitly defines the data sources i.e., once again, the interface objects and affected objects an IDS may monitor to detect a given attack.

---

[8] USTAT is the host-based implementation for Unix (Sun Solaris) systems [IlKePo95, PorKem92]; WinSTAT the respective implementation for Windows NT; and NetSTAT the implementation of a network-based IDS.

Figure 17 provides an overview of the activity categorization scheme to be developed in this and the following sections. The figure not only includes the static activity characteristics, but also the dynamic ones. The latter are discussed in the next section. The figure shows how we distinguish between static and dynamic activity characteristics, and how these are refined further into interface objects, affected objects etc.

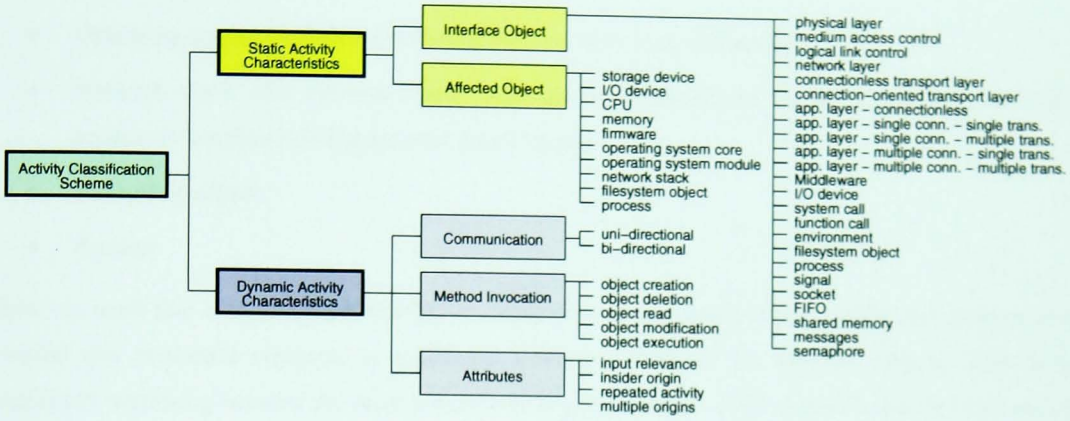


Figure 17—Overview of activity categorization scheme

Remember, this scheme is the result of an iterative process taking place in the context of the VulDa operation.

Next we describe the static portion of the activity categorization scheme as shown in Figure 17, starting with the affected object. The characteristics used to describe the affected object and the interface objects both represent a subset of the IDS scope concept introduced in Section 4.1.

4.2.2.1 Affected objects

It is the goal of a given attack to intrude a target object. More formally speaking, the goal of an attack is to change the state of the target object to an error state with respect to the security policy. However, as discussed in Section 2.1, a security policy violation does not necessarily involve malice. For this reason we introduce the term *affected object* instead of using a term such as attacked object or target object, which implies malicious intent. It is also worth recalling that the fault i.e., the vulnerability, that enables a successful attack need not necessarily be located in the affected object.

In the following we list the affected objects chosen from the IDS scopes described in Section 4.1. They have been identified by isolating the physical and logical ID-relevant components in a networked computing system. Naturally, we focus on components and finer grained sub-components known to be critical to security and ID in particular. We have done so by selecting IDS scopes at different levels in the IDS scope tree (see also Figure 9). Note that many of the scopes listed in the following are listed for the sake of completeness: They are listed because it is conceivable that they might play the role of an affected object for future attacks.

- Storage device

- I/O device
- CPU
- Memory
- Firmware
- Operating system core
- Operating system module—Excluding the networks stack OS module.
- Network stack—the network stack is commonly implemented as an OS module. We list it separately because it is a prominent attack target.
- Filesystem object
- Process

When we used this categorization scheme to categorize attacks taken from VulDa, the categorization revealed that processes represent a significant majority (76%) of the affected objects. This is not particularly surprising because the most noticed and reported attacks affect network services such as http, FTP, DNS etc., which are commonly implemented by (daemon) processes. For similar reasons it is also no surprise that filesystem objects are the second most-popular attack target (13%) (see Appendix B.3 for more detailed statistics). In a nutshell, it is certainly fair to state that the categorization did not reveal any surprises with respect to affected objects, but instead confirmed the impression one obtains when browsing, for instance, the CERT [CERT] advisories or the Bugtraq IDs [SecFoc]. It highlights, however, the most relevant attack categories in a well-founded manner.

4.2.2.2 Interface objects

Whenever an attack is launched, it targets what we just introduced as the affected object. As shown in Figure 16, in order to interact with this object, the activity initiator has to involve one or more *interface objects,* or in other words attack interfaces. This can naturally be extended to the more general case where no malice is involved. In this case we simply describe the interface objects that an activity involves. However, we are only interested in those interface objects that could, when monitored, enable an IDS to recognize evidence of the attack being categorized and that are technically observable. This allows us to limit the choice of interface objects to the proximity of the attacked object. It would, for instance, probably not be appropriate to consider the adversary's keyboard as a potential information source for the detection of a webserver attack because that keyboard is practically not relevant to the detection of evidence of this specific attack. The keyboard may, however, serve as meaningful information source for detecting other attacks.

As it is the case for the affected objects identified above, the list of interface objects was developed based on the notion of IDS scopes introduced in Section 4.1. In fact, most of the affected objects listed above may also serve as an interface object. However, there is an important number of (networking) objects that may be used as interfaces and that were not listed among the affected objects. A slight difference in the affected objects identified earlier is that we use IDS scopes attributes (see Section 4.1.2) to refine the

model. Moreover, note that we consider instances of networking IDS scopes rather than the corresponding communication layer abstractions, because IDSs operate on instances of communication layer protocols, e.g., PDUs. This enables us to consider these networking IDS scopes as objects, which ensures consistency among the (interface) objects identified. All this results in the following list of interface objects:

- Physical layer
- Medium access control
- Logical link control
- Network layer

In the case of the transport layer we distinguish, using IDS scope attributes, between the connection-oriented transport layer and the connectionless datagram service, as it may be a differentiator for the capability of an IDS to detect an ongoing attack.

- Connectionless transport layer
- Connection-oriented transport layer

In the case of application layer protocols, we differentiate even further. We distinguish among protocols based on connectionless and connection-oriented services, and consider the number of transactions (see Section 4.1.2) that can be executed in the context of a single session. We further distinguish protocols that use more than one lower level service concurrently. We distinguish these different ways of operation because they may be a differentiator for IDSs.

- Application layer based on a connectionless service
- Application layer based on a single connection, single transactions. Typical examples are http version 1.0 or the remote shell.
- Application layer based on a single connection, multiple transactions.

  Example: *A typical example is http version 1.1 that supports persistent connections. We have observed that IDS are incapable of recognizing http attacks when the first request of a persistent connection was non-malicious. Another example is the mail transfer protocol SMTP; here the situation is similar, i.e., we have found IDSs that were unable to recognize attacks when the first mail message transferred was non-malicious.*

- Application layer based on multiple connections, single transaction. We ignore this category in the following because we are not aware of any protocol that qualifies for this category.
- Application layer based on multiple connections, multiple transactions. A typical example is the file transfer protocol FTP. The analysis of such protocols is a nontrivial task for IDSs.

Before continuing with the identification of interface objects that are primarily host oriented, we consider the special case of middleware. When considering an activity that involves middleware, it is generally not possible to distinguish between host- and network-based use of middleware. Therefore we do not make

this distinction for our categorization scheme either. However, as we will see in the next chapter, it makes sense to distinguish the middleware-related characteristics of an IDS—separating the network and the host portion because this may have an influence on the degree to which an IDS is able to analyze such an activity.

- Middleware
- I/O device
- Operating system module: Some operating systems such as Linux or Solaris provide an interface that allows additional modules to be loaded dynamically. Such modules may represent an attack interface to the running kernel.
- System call
- Function call
- Environment
- Filesystem object: Filesystem objects typically serve as an indirect interface to processes and to the OS.
- Process: A process may be used as an interface to variety of other objects such as the filesystem.

Inter-process communication allows processes to communicate among each other. A number of different mechanisms have been developed over time:

- Signal
- Socket
- FIFO
- Shared memory
- Messages
- Semaphore

Considering the results of the attack categorization with respect to interface objects as they are presented in Appendix B.2, one might be surprised to find again that most frequently processes are the interface objects used to stage attacks. In fact, in 36% of all attacks categorized processes are used as an interface. However, taking a closer look at the categorization results, one can easily identify the total of the diverse application layers to be most prominent attack interface used. This corresponds to the observation made above that processes implementing network services are the most prominent attack targets.

4.2.3 Dynamic activity characteristics

The dynamic activity characteristics of this categorization scheme focus on observable and transient attack-relevant characteristics. These characteristics will, to a large extent, enable us to analyze the attack recognition and identification capabilities of IDSs. We do not model the impact of attacks, as we focus on

real-time and transition-based IDSs. In other words, we focus on observable evidence of attacks and not on the (possibly) resulting internal state change of the affected object.

In Section 4.2.1 we have already identified the three sets of dynamic activity characteristics that describe the interaction among objects: Namely inter-object *communication*, *method invocation*, and some additional *activity attributes*.

By separating the inter-object communication and the method-invocation characteristics it becomes possible to capture the differences between attacks staged over the network and attacks staged locally. However, this does not mean that network-related activities are only described by communication characteristics. In fact, in most cases activities have to be described by a mixture of network-related and host-related characteristics.

4.2.3.1 Communication

The communication characteristics we have identified are rather simple. This simple solution was possible because of the interface objects introduced earlier. These interface objects already capture a significant portion of communication-protocol-specific characteristics. In accordance with the separation of static and dynamic activity characteristics, this leaves us with the following two (network) communication-related, observable activity characteristics:

- Uni-directional: The communication flows in one direction only.
- Bi-directional: The communication flows between two peers, e.g., TCP connection but also UDP services such as DNS.

Note that an attack involving a bi-directional protocol such as TCP does not necessarily need to be bi-directional. In fact typical denial-of-service (DoS) attacks against a host's network stack, such as teardrop or land [CA2897], are often uni-directional only and consists of some malformed PDUs sent to the targeted host. In most cases the target host does not reply because the protocol used does not require it to do so or because the target has already become unresponsive, e.g., crashed.

As explained in detail in Appendix B.1 the bi-directional communication characteristic is the most frequent dynamic activity characteristics. This is no surprise in the light of the observations made with respect to the static activity characteristics, where we observed that attacks against network services are the most frequent ones.

4.2.3.2 Method invocation

The second set of dynamic activity characteristics is the set of methods invoked in the context of the affected object (see also Figure 16). The identification of these methods is relatively straight forward:

- Object creation: A new object is created. This generally occurs in the context of an existing object such as the filesystem within which a new file can be created.
- Object deletion: An object is deleted, e.g., deletion of a file.

- Object read: The internal state or part of an object's internal state is read, e.g., the memory of a process or the content of a file.
- Object modification: The internal state of an object is modified, e.g., the content of the password file is modified.
- Execution within object context: The observable behavior of an object is changed such that it threatens the security policy, e.g., the execution path of process is modified. The most typical examples are probably buffer overflow attacks [Aleph96, CA1395] and attacks involving special characters [CA0696, CA0797].

It is clear that the affected object, within whose context a given method is invoked, defines the semantics of these methods. Finally, note also that attacks usually involve the invocation of several methods concurrently.

Also for these activity characteristics the categorization of attacks did not reveal any surprise. Appendix B.1 shows that the "execution" method is the one most frequently invoked by attacks. This is clearly due to the highly popular buffer overflow attacks, but also to attacks that directly cause the execution of arbitrary commands on the target system.

4.2.3.3 Activity attributes

After having defined the dynamic activity characteristics describing the communication and the method invocation aspects, we have to admit that there are still ID relevant aspects of activities that have not yet been described. For instance it is not possible to distinguish between attacks where a given method is executed only once and attacks where the same method is executed repeatedly.

Example: *Categorizing attacks using the characteristics identified so far, it is not possible to distinguish between the (simple) creation of a file or a link, and the repeated creation of a file or link. The repeated creation of such filesystem objects is typical for attacks exploiting race conditions.*

In order to address these issues we have identified additional *activity attributes* that allow us to refine the description of attacks, i.e., activities:

- Input data relevance: The input provided to an object is relevant to the attack. This characteristic can be used to refine the description of buffer overflow attacks etc. It thereby addresses the fact that an IDS needs to be able to perform additional analysis on the data in order to recognize attacks in which input data is relevant.
- Repeated activity: Certain types of activities can only be clearly categorized as being malicious when they are observed repeatedly. Typical examples are scanning activities or the exploitation of race conditions.
- Internal origin: Some malicious activity may originate from inside the system to be protected. Typical examples are the hidden communication channels e.g., communication hidden in DNS traffic sent to the outside. Other examples include Trojan horses or the presence of an adversary among the employees of an organization.

- Multiple origins: In some cases attacks appear to have multiple sources. The recognition of this fact may be crucial to identify a given attack clearly. Examples are attacks such as Smurf [CA0198] or distributed DoS attacks such as Trinoo [CIN0799]. Such attacks send a large amount of PDUs with arbitrarily forged sender addresses to a target.

Note that we do not describe the dynamic activity characteristics of an activity merely by activity attributes. The activity attributes are only observable when combined with either communication or method invocation characteristics. In other words, they are merely a property of the activity in question.

Again considering the statistics resulting from the attack categorized (see also Appendix B.1), we can once more verify the common observation of attacks on the Internet. In more than 50% of all attacks categorized, the input data provided was directly relevant to the attack. Considering the resulting histogram shown in Figure 51 confirms the popularity of attacks against network services because a large portion of the attacks in which input data is relevant also involve some form of network communication. Considering the attribute "repeated activity," it is possible to verify the importance of race conditions and certain categories of DoS attacks.

4.2.4 Categorization examples

After having developed a categorization scheme for activities, there is clearly a need for illustration. In the following we provide four examples as they can be found in VulDa. At the end of this section we briefly discuss the question whether this activity categorization can be used to identify yet unknown attacks, and provide an additional example.

Example 1: *Weaknesses in the validation of the input provided over the CGI (common gateway interface) interface offered by webservers often lead to vulnerabilities. Classical examples are buffer overflow vulnerabilities such as the one present in Microsofts IIS webserver software [CA1301], which was exploited by the worm called "CodeRed" [CA1901, CA2301]. The corresponding attack can be categorized as follows:*

- *Affected object: process*
- *Interface objects: application layer (single connection, single and multiple transactions)*
- *Communication: bi-directional*
- *Method invoked: execution within object context*
- *Attributes: input data relevant*

Example 2: *Another similar vulnerability is the test-cgi vulnerability [CA0797] reported in 1997. This test script enables an adversary to read protected files from the webserver. So, the affected object is a filesystem object that is accessed by a process which itself is influenced by an application protocol request to the webserver. Last but not least, some special characters were used in the URL requested from the webserver. All in all this leads us to the following categorization:*

- *Affected object: filesystem object*

- *Interface objects: process, application layer (connection-based, single and multi transaction)*
- *Communication: bi-directional*
- *Method invoked: object read*
- *Attributes: input data relevant*

Example 3: *The following example affects a product called Pitbull LX, which hardens the Linux kernel to a degree that the classical root authority disappears. Pitbull LX is a product developed by Argus Systems*[9]*. In his Bugtraq posting [Postle01], Postle provides the following description:*

> *The vulnerability stems from Pitbull LX's failure to apply its enhanced security features to all kernel variables made available in /proc/sys/. Although the file-system will restrict access to the /proc/sys/ directory, these variables can be accessed through calls to sysctl() which only checks a process's standard unix credentials. Almost all variables are mode 644 or 444. So any user can read the kernel variables and a root user can modify many of them. A process with uid 0*[10]*, can thus bypass Pitbull and modify some very sensitive kernel data. (If that last statement makes you wonder what the problem is remember that "root means nothing on a Pitbull system".)*

Based on this description we categorize an attack exploiting this vulnerability as follows:

- *Affected object: OS core*
- *Interface objects: system call, filesystem object, process*
- *Communication: none*
- *Methods invoked: object modification*
- *Attributes: none*

4.2.5 Discovery of yet unknown attack categories

So far we provided examples on how to categorize known attacks. In the course of this categorization also the question was raised whether it is possible to discover attacks that are not yet know using the activity categorization scheme. We believe this to be possible, although nontrivial. We have not been able to identify such an attack, but we made the observation that one can *predict* categories of attacks that are yet to be discovered, at least to a limited degree.

Example 4: *In the course of maintaining VulDa, we began to wonder whether it is not possible that one can make use of the signal generated whenever TCP out-of-band traffic is received to attack a process. In fact, just recently, a theoretical attack against some FTP daemons has been discovered [Zalews01] in which a remote attacker is able to inject executable data over the network that can then be activated by*

[9] http://www.argus-systems.com

sending TCP out-of-band traffic. The TCP out-of-band traffic causes a signal to be sent to the process which then starts processing a signal handler. Unfortunately this signal handler contains calls to non-reentrant system calls, which may lead to the execution of an arbitrary command. While we do not know whether this attack has already been successfully run against a FTP daemon, it has been proven that the attack is theoretically possible—even though it is considered to be very difficult. This hypothetical attack would then be classified as follows:

- *Affected object: process*
- *Interface objects: transport layer (connection-oriented), signal, system call*
- *Communication: bi-directional*
- *Method invoked: execution within object context*
- *Attributes: input data relevant, repeated activity*

4.3 Selection of representative activity classes based on a categorization of attacks

After having developed the categorization scheme for activities, we can use this scheme to identify attack classes that are of interest to the analysis of IDSs. Concurrently the results of such a categorization allow us to assess the utility of the categorization scheme. The results of the latter compare well with the experience we gained while working in the security field and while populating VulDa.

At the time this categorization was made, approximately 800 attacks were described in the context of one of VulDa's vulnerability descriptions (see also Section A.3). In the time available, we succeeded in categorizing 358 of these attacks along the lines of the examples described above. The selection was done based on a randomly ordered list of vulnerability descriptions. We simply started at the top of this random list. Moreover note that the categorization scheme has become part of the maintenance process of VulDa. The categorization of attacks has thereby become an ongoing effort yielding increasingly representative results.

Because the categorization scheme permits the use of almost arbitrary combinations of activity characteristics to categorize one attack, the number of possible attack categories is enormous. The only restriction is that only one affected object may be identified per attack. Given the possibility to create quite fine-grained characterizations of attacks, it is no surprise that 203 categories were identified (for detailed results see Appendix B).

Figure 18 further suggests that our scheme has the correct granularity as no overly large number of attacks falls into the same category, and is not too fine-grained as there is no majority of attacks that form their own (single-member) categories. Figure 18 reveals that 41% of all attacks categorized define their

---

[10] In Unix systems the numerical user ID 0 (zero) corresponds to the "root" user, i.e., to the administrator account.

own category, i.e., belong to a category that has only one member. However, the figure also shows that an important percentage of attacks belongs to categories of attacks with several members. We have even been able to identify one category with 15 members, which thereby covers 4.2% of all attacks. Given this distribution, it is certainly not advisable to increase the granularity of the scheme as one might obtain results that are too detailed and therefore would no longer reveal the important attack categories. However, the granularity of the results suits our purposes well, and also corresponds to the observations made during the daily operation of VulDa.

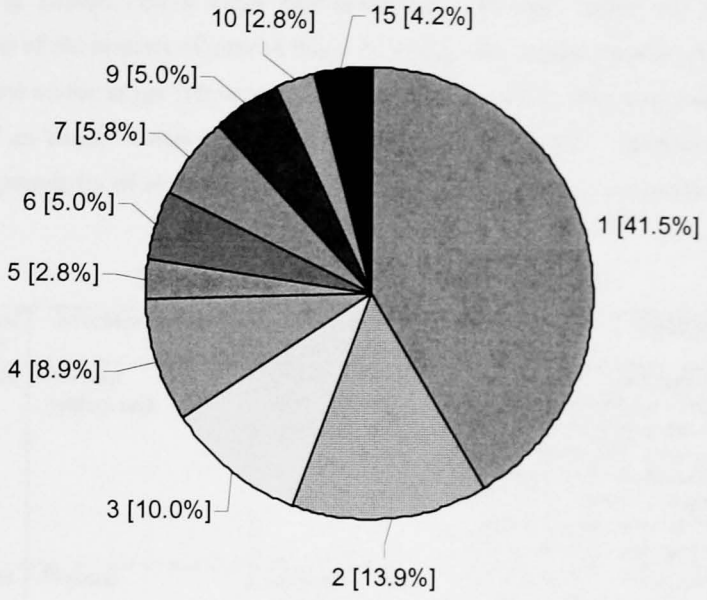


Figure 18—Distribution of attack category sizes

In a next step we have isolated the largest categories of attacks in order to use these attack categories to identify the activity classes that shall be used to analyze IDSs. It is clear that normally one would choose at least the ten most frequent categories of attacks for the analysis of IDSs. In fact, one should go quite a bit further than this. However, our choice focuses merely on attack categories relevant to the three IDSs (see Section 8.6.1) that we have chosen for our example application that assesses IDSs and combinations thereof. This means that we will not choose attack categories for analysis of which we know that none of the IDSs is designed for detecting them. The reason for this is not the fear of false results but merely to focus our efforts. Moreover, remember that the results provided in **Error! Not a valid bookmark self-reference.** reflect the importance of attack categories in terms of the number of attacks found in VulDa. The results thereby provide only an indirect and subjective view on the usage frequency of attacks. In other words, they only provide an indication of the popularity of an attack category revealed by the number of distinct attacks assigned to it, but no indication on the popularity of an attack category based on observations made on real networks.

Table 3 lists the ten largest categories of attacks, and provides a brief description of the type of attacks assigned to each of these categories.

It is clear that normally one would choose at least the ten most frequent categories of attacks for the analysis of IDSs. In fact, one should go quite a bit further than this. However, our choice focuses merely on attack categories relevant to the three IDSs (see Section 8.6.1) that we have chosen for our example application that assesses IDSs and combinations thereof. This means that we will not choose attack categories for analysis of which we know that none of the IDSs is designed for detecting them. The reason for this is not the fear of false results but merely to focus our efforts. Moreover, remember that the results provided in **Error! Not a valid bookmark self-reference.** reflect the importance of attack categories in terms of the number of attacks found in VulDa. The results thereby provide only an indirect and subjective view on the usage frequency of attacks. In other words, they only provide an indication of the popularity of an attack category revealed by the number of distinct attacks assigned to it, but no indication on the popularity of an attack category based on observations made on real networks.

Table 3—The ten largest attack categories

Nbr.	Cat. size	Affected object	Interface objects	Dynamic characteristics	Description
1	15	Process	- Process - System call	- Execution within object context - Input data relevant	This category covers local buffer overflow attacks and special-character attacks that result in observable change in the execution path. In most cases suid-root processes are targeted. The category almost exclusively covers attacks that involve command line arguments that are potentially observable at the system call used to launch the vulnerable program.
2	10	Process	- Process	- Execution within object context - Input data relevant	This category is almost identical to category 1. The only difference is that the attack is in most cases staged over the standard input instead of the command line arguments.
3	9	Process	- App. layer protocol (single connection, single and multiple transactions)	- Bi-directional communication - Execution within object context - Input data relevant	This category almost exclusively covers buffer overflow attacks against webservers. A smaller number of attacks involve special characters instead of buffer overflows, but also lead to noticeable changes in the execution path of the affected process. http v1.0 only supports a single request within one session, whereas http v1.1 supports multiple requests within one session. As such a difference between versions is not known for any other popular protocol, this category contains http related attacks only.
4	9	Process	- App. layer protocol (single connection, multiple transactions)	- Bi-directional communication - Execution within object context - Input data relevant	This category is almost identical to category 3. The only difference is that it affects other popular services such as FTP or SMTP.
5	7	Process	- App. layer protocol (single connection, single transaction)	- Bi-directional communication - Execution within object context - Input data relevant	This category is almost identical to the third category. The only difference is that it affects covers other, even simpler services such as http v1.0.
6	7	Process	- App. layer protocol (single connection, multiple transactions)	- Bi-directional communication - Input data relevant	This category mostly concerns authentication issues in services such as SMTP or IMAP.

Nbr.	Cat. size	Affected object	Interface objects	Dynamic characteristics	Description
7	7	Process	- App. layer protocol (no connection)	- Uni-directional communication - Input data relevant	This category covers classical DoS attacks against UDP-based services. In general simple 'crash-packets' are used to stop services such as Syslog or DNS.
8	6	Filesys. object	- Process	- Object read	This category generally covers local attacks in which the content of protected files is read without the permission to do so.
9	6	Process	- Environment	- Execution within object context - Input data relevant	Similar to attack category 1. The difference is that the data carrying the attack is passed to the process over the environment, e.g., environment variables.
10	6[11]	Process	- App. layer protocol (no connection)	- Bi-directional communication - Execution within object context - Input data relevant	This category again covers remote buffer overflow attacks. They affect services such as DNS.

Example: *Compared to buffer overflow attacks, only relatively few networking-related DoS attack scripts are published. However, while observing internet traffic, one finds DoS attacks to be quite prominent, although they do not appear in the categorization statistics shown here. Another, even more extreme example is port scanning, which can be observed very frequently, but where the attack itself is essentially always the same and is therefore covered by a single of VulDa's vulnerability descriptions.*

This said and considering It is clear that normally one would choose at least the ten most frequent categories of attacks for the analysis of IDSs. In fact, one should go quite a bit further than this. However, our choice focuses merely on attack categories relevant to the three IDSs (see Section 8.6.1) that we have chosen for our example application that assesses IDSs and combinations thereof. This means that we will not choose attack categories for analysis of which we know that none of the IDSs is designed for detecting them. The reason for this is not the fear of false results but merely to focus our efforts. Moreover, remember that the results provided in **Error! Not a valid bookmark self-reference.** reflect the importance of attack categories in terms of the number of attacks found in VulDa. The results thereby provide only an indirect and subjective view on the usage frequency of attacks. In other words, they only provide an indication of the popularity of an attack category revealed by the number of distinct attacks assigned to it, but no indication on the popularity of an attack category based on observations made on real networks.

Table 3, we choose attack categories 3-7 and 10, excluding categories 1, 2, 8 and 9, which are not relevant to the IDSs we selected for assessment. However, the activity classes mentioned and selected so far only concern application layer services. To demonstrate the flexibility of our approach we choose to include transport and network layer activities as well. These activities were selected among attacks known to be popular (see also the example above and the list of the ten most popular attacks maintained by SANS [SANS]). Snort [Roesch99], one of the IDSs chosen for assessment (see Section 8.6.1), has the potential of detecting such lower-layer attacks. Thus, it can be expected that RIDAX reveals that Snort

[11] The 10 largest attack classes include all classes with more than 5 members.

has the potential of detecting at least some of these lower-layer activities. This is discussed in greater detail in Section 8.6.

Based on the results obtained by categorizing attacks and based on the additional reasoning mentioned, we have identified a list of 48 activity classes, 21 benign and 27 malicious (see Table 4), that were used for our RIDAX experiments. In the first column of Table 4, we provide reference to the attack categories as identified in It is clear that normally one would choose at least the ten most frequent categories of attacks for the analysis of IDSs. In fact, one should go quite a bit further than this. However, our choice focuses merely on attack categories relevant to the three IDSs (see Section 8.6.1) that we have chosen for our example application that assesses IDSs and combinations thereof. This means that we will not choose attack categories for analysis of which we know that none of the IDSs is designed for detecting them. The reason for this is not the fear of false results but merely to focus our efforts. Moreover, remember that the results provided in **Error! Not a valid bookmark self-reference.** reflect the importance of attack categories in terms of the number of attacks found in VulDa. The results thereby provide only an indirect and subjective view on the usage frequency of attacks. In other words, they only provide an indication of the popularity of an attack category revealed by the number of distinct attacks assigned to it, but no indication on the popularity of an attack category based on observations made on real networks.

Table 3. In the second column we provide a rating of the activity category indicating whether the category is to be considered benign or malicious. Then, in the third column, we provide the IDS scope at which this activity category was specified. In column four it is then shown for which IDS scopes the IDSs are actually to be analyzed for. This parameter significantly influences the manner in which the activity category is refined to an activity class. After having provided the activity class numbering as used in RIDAX, we finally provide a short description of the activity classes in the last column.

Owing the flexibility of our activity categorization scheme, any of the activities identified in Table 4 also, at least partially, cover additional attack categories. For instance, consider an attack and one of the attack categories shown in It is clear that normally one would choose at least the ten most frequent categories of attacks for the analysis of IDSs. In fact, one should go quite a bit further than this. However, our choice focuses merely on attack categories relevant to the three IDSs (see Section 8.6.1) that we have chosen for our example application that assesses IDSs and combinations thereof. This means that we will not choose attack categories for analysis of which we know that none of the IDSs is designed for detecting them. The reason for this is not the fear of false results but merely to focus our efforts. Moreover, remember that the results provided in **Error! Not a valid bookmark self-reference.** reflect the importance of attack categories in terms of the number of attacks found in VulDa. The results thereby provide only an indirect and subjective view on the usage frequency of attacks. In other words, they only provide an indication of the popularity of an attack category revealed by the number of distinct attacks assigned to it, but no indication on the popularity of an attack category based on observations made on real networks.

Table 3. If the characterization of the attack requires a super-set of the activity characteristics used to describe the attack category, the attack would not belong to this category but to another, more specific,

category. This is the reason why the categorization resulted in the relative large number of 203 attack categories.

Table 4—Activities selected for analyzing IDSs

Attack categories addressed	Rating	IDS scope of activity category	IDS scope of activity class	Activity Nbr.	Description
3	Benign	Application layer (single connection, single and multiple transactions)	http	21	Class of http activities that use options strings that are so similar to attacks that even a sophisticated signature-based IDS can be confused.
				22	Class of http activities that use options strings that are so similar to attacks that IDS only using simplistic pattern-matching techniques may be confused.
	Malicious			1	Class of http-request line-based buffer overflow attacks that cause the execution path of the process offering the http service to divert.
				2	Class of http-request line-based special-character attacks that cause the execution path of the process offering the http service to divert.
				3	Class of http-request-options-based buffer overflow attacks that cause the execution path of the process offering the http service to divert.
				4	Class of http-request options-based special-character attacks that cause the execution path of the process offering the http service to divert.
6				5	Class of http-request options-based attacks that enable the adversary to read file or directory content that is protected otherwise.
			SMTP, http, FTP	6-8	Class of protocol statement-based (e.g., http-request line) attacks that enable the adversary to read a file or directory content that is protected otherwise. Usually an access control and/or authentication issue.

Attack categories addressed	Rating	IDS scope of activity category	IDS scope of activity class	Activity Nbr.	Description
3-7, 10	Benign	Application layer	SMTP, http, Syslog	27, 28, 35	Class of activities where strings that would be considered suspicious if used in the context of a protocol statement (e.g., http-request line) appear in the data portion of a protocol transaction. Although harmless, this may be confused with a real attack by IDSs using simplistic techniques to track protocol sessions.
			http, FTP, SMTP, Domain, Syslog	12-14, 29, 30	Ordinary http, FTP, SMTP, DNS and Syslog requests.
				15-17, 36, 37	Classes of activities that use request strings that are so similar to attacks that even a sophisticated signature-based IDS can be confused.
				18-20, 38, 39	Classes of activities that use request strings that are so similar to attacks that IDS only using simplistic pattern matching techniques may be confused.
N/a	Malicious			9-11, 31, 32	Flooding of the server with requests.
4, 5, 10			FTP, SMTP, Domain, Syslog	23, 24, 40, 41	Classes of protocol statement-based buffer overflow attacks that cause the execution path of the process offering the targeted service to divert.
4, 5			FTP, SMTP	25, 26	Classes of protocol statement-based special-character attacks that cause the execution path of the process offering the targeted service to divert.
7		App. layer (connection-less)	Domain, Syslog	33, 34	Classes of attacks that typically crash the process implementing the DNS service remotely. Example: SIG records BIND vulnerability [CA1499].
N/a	Benign	Transport layer (connection-based)	TCP	43	A single source is observed sending many requests to a server. There are settings in which the frequency of such requests may reach a level at which IDSs might suspect a DoS attack being in progress. Example: A firewall that masquerades many clients.
	Malicious			42	Typical SYN-flooding attacks. These attacks attempt to either temporarily or permanently overwhelm a server such that the services offered are no longer available.
		Transport layer	TCP, UDP	45-48	These attack classes cover normal and "slow" scanning of TCP and UDP ports.
		Network layer	IP	44	There are many DoS attacks against a variety of TCP/IP stacks that exploit vulnerabilities in the IP fragment reassembly code.

Example: *Consider attack category 7 as listed in It is clear that normally one would choose at least the ten most frequent categories of attacks for the analysis of IDSs. In fact, one should go quite a bit further than this. However, our choice focuses merely on attack categories relevant to the three IDSs (see Section 8.6.1) that we have chosen for our example application that assesses IDSs and combinations thereof. This means that we will not choose attack categories for analysis of which we know that none of the IDSs is designed for detecting them. The reason for this is not the fear of false results but merely to focus our efforts. Moreover, remember that the results provided in* ***Error! Not a valid bookmark self-reference.*** *reflect the importance of attack categories in terms of the number of attacks found in VulDa. The results thereby provide only an indirect and subjective view on the usage frequency of attacks. In other words, they only provide an indication of the popularity of an attack category revealed by the number of distinct attacks assigned to it, but no indication on the popularity of an attack category based on observations made on real networks.*

Table 3. This category describes uni-directional, connectionless application layer attacks in which the input data is relevant to the attack. If we query VulDa for attacks that show at least these characteristics we find not 7 but 10 attacks. This means that for this case there exist three additional attacks that are

closely related and that are at least partially addressed by activities 27-39 listed in Table 4. In fact, we find that the three additional attacks involve the additional activity characteristic "execution within object context." Again considering activities 27-39 we find that they cover many ID-relevant, i.e., observable, aspects of these attacks.

4.4 Discussion

The categorization of attacks according to our activity categorization scheme had the goal of identifying the attack categories most relevant to ID such that we could then determine the attack classes required for a systematic analysis of IDSs. As a consequence the categorization has to ensure that all aspects relevant to the analysis performed by IDSs are covered, thereby avoiding the irrelevant ones in order to limit complexity. As our categorization relies on a rigorous concept that combines the system model introduced in Section 4.2.1 with the richness of the data available in VulDa (see Appendix A), we are confident that we have indeed produced a highly viable categorization of attacks. However, statistics derived from the categorization of attacks taken from VulDa are not representative of the instances of attacks as they can be observed on a daily basis on the internet, for instance. The main reasons for this are the following:

1. One can observe remote attacks far more frequently than local ones because it is much simpler for adversaries to identify and attack potential targets on the network than it is to attack a vulnerable and attractive machine the adversary already has legitimate access to. It would be considered unwise of adversaries if they were to compromise systems they already have legitimate access to, as the likelihood of them being tracked down is simply to high.
2. There exists a large number of application layer protocols and therefore an even larger number of implementations of the corresponding services. This creates a large potential for vulnerabilities to be introduced and for the corresponding attacks to be published. On the lower-level network layers, one cannot find a comparable diversity of protocols and implementations. As a consequence fewer attacks affecting these lower layers are found and published. Therefore such lower-layer attacks do not show up in the top-ten of the attack categories, although they are very popular. In order to compensate for this, we have identified and described additional classes of attacks and activities targeting these lower layers.
3. The attacks found in VulDa are slightly biased towards attacks with significant impact, such as remote root-shell exploits etc. Accordingly some of the affected objects defined for the activity categorization were not used. The reason for this lies in the way VulDa was populated and in the overwhelming number of new vulnerabilities and attacks being discovered and published on a daily basis. This large number of new vulnerabilities forced us to focus on the most important, i.e., dangerous, vulnerabilities and attacks only. Therefore the fact that some of the affected objects were never used to categorize one of the 358 attacks clearly shows that they are unlikely to be used in high-profile attacks.

Finally, note that the activity categorization allows a more detailed characterization of attacks involving network communication. This is due to the fact that the network stack is fairly well structured by its layer scheme, which is not true for the host-level characteristics. This results in fewer, but larger attack categories that represent local exploits as shown in It is clear that normally one would choose at least the ten most frequent categories of attacks for the analysis of IDSs. In fact, one should go quite a bit further than this. However, our choice focuses merely on attack categories relevant to the three IDSs (see Section 8.6.1) that we have chosen for our example application that assesses IDSs and combinations thereof. This means that we will not choose attack categories for analysis of which we know that none of the IDSs is designed for detecting them. The reason for this is not the fear of false results but merely to focus our efforts. Moreover, remember that the results provided in **Error! Not a valid bookmark self-reference.** reflect the importance of attack categories in terms of the number of attacks found in VulDa. The results thereby provide only an indirect and subjective view on the usage frequency of attacks. In other words, they only provide an indication of the popularity of an attack category revealed by the number of distinct attacks assigned to it, but no indication on the popularity of an attack category based on observations made on real networks.

Table 3. Note also that based on the attack characteristics describing the network communication as either uni- or bi-directional, we were able to determine that 192, i.e., 54%, of the 358 attacks categorized represent remote attacks. Although the data has been identified as being slightly biased towards high-profile attacks, the categorization results represent well the common observation one makes when monitoring forums such as Bugtraq [SecFoc] that report or discuss new vulnerabilities and attacks.

Remember, the selection of activities, be they benign or malicious, that are to be used to analyze IDSs is one of the most important elements of any approach to IDS analysis. However, it is also one of the most difficult and controversial elements. The choice of activities not only significantly influences the analysis results, but also has to reflect the environment for which a suitable IDS is to be found. Because of these dependencies and because our approach operates at a rather conceptual level, we chose to develop a categorization of attacks and to use the categorization results to identify the activity classes to be used for our analysis approach. However, for the reasons given, it made no sense to rely solely on the categorization results. Additional facts such as the popularity of lower-layer network attacks, e.g., the so-called Teardrop and Land attacks [CA2897], also had to be taken into account. The reason these frequently observed attacks do not appear more prominently in our categorization is that only a comparatively small number of distinct implementations of such attacks exists. Moreover, our choice was influenced by the set of IDSs chosen for the experiment with RIDAX. These IDSs all differ significantly in the way they operate, but all of them address remote, i.e., networking-related, attacks only. Because of this we decided to limit our effort of creating activity descriptions for the RIDAX experiments to networking-related activities.

Finally one might ask whether 48 activity classes are sufficient to analyze IDSs. There are three aspects one should consider with respect this question:

1. Each of the 48 activity classes identified in Table 4 describes more than just single activity—they stand for entire classes of activities.
2. In the context of working with IDSs and creating the attack categorization, we have realized the importance of the fact that activities might be altered for obfuscation purposes [PtaNew98, RFP00]. A typical example is the fragmentation of PDUs at a lower networking layer, e.g., fragmentation of IP PDUs. In Chapter 6 we describe the variations that we considered in this work and how we applied them to activities to create activity class variants. However, although we do not explain the concept of varying activities here, note that using this concept, we derived almost 1000 activity class variants from the 48 activity classes identified here.
3. The RIDAX implementation represents a *prototype* that we use to demonstrate and validate our approach, which was achieved using the activities identified here.

4.5 Conclusion

In this chapter we have introduced the concept of *IDS scopes*, which forms a cornerstone of this work. Using this concept, we developed a categorization scheme for activities that was then used to categorize a large number of attacks—the goal being the identification of a representative input set to our IDS analysis approach. By doing so we illustrated the difficulties one generally faces when attempting to compose a set of activities to be used for the analysis of IDSs. Using the categorization results we finally identified 48 activity classes, each of them representing a complete class of activities, which we subsequently use in the RIDAX experiments described in Chapter 8: 21 of them represent benign and 27 represent malicious activity classes. Given the systematic fashion the activity categorization scheme was developed and based on further considerations made in this chapter, we are confident that the 48 activity classes identified are well suited to verifying and illustrating our approach by means of the RIDAX experiments.

Chapter 5 Intrusion detection system description framework

IDSs are complex systems as is their description. This was already apparent in Section 2.2.3, where we provided an overview of existing approaches to describe and classify IDSs. There is no generic classification or description scheme for IDSs as there is none for attacks. The goals of any such scheme will determine the criteria to be used. For the scheme introduced in this chapter, the goals are to describe the IDS capabilities that are relevant to the detection of attacks and to the generation of alarms. This includes the requirement that the description scheme developed in the following has to enable the creation of IDS descriptions that can be used as input to the RIDAX tool to analyze the IDS described. We achieve this by first introducing a system model that identifies the components an IDS consists of. The model represents a simplified version of the CIDF model [CIDF98], and consists of a *sensor* component, which is used to collect information, and a *detector* component, which performs the actual analysis. Subsequently we develop our description scheme for these components, and conclude the chapter by describing the database structure used by the RIDAX prototype (see Chapter 7). For a complete example of an IDS description, see Appendix C.4.

Owing to the generic and extensible concepts used, our scheme is suitable for the description of a large variety of IDSs: It does not describe the implementation details of the analysis steps employed by a given IDS, but rather describes the fact that the IDS uses analysis techniques of a given type to perform its analysis at a given level of abstraction. The underlying concepts of our scheme have been developed by combining our own experience with insights gained by IDS classifications and taxonomies such as those of Debar *et al.* [DeDaWe00, DeDaWe99] or that of Axelsson [Axelss00] (see also Section 2.2.3).

In view of the attack categorization scheme developed in the preceding chapter, it is clear that our IDS description and classification scheme has to be of greater detail than the existing ones. For instance, simply describing an IDS as either behavior-based or knowledge-based (see Figure 7, p. 16) does not enable us to draw precise conclusions about either the attack classes the IDS is able to detect or, for instance, the number of false positives it may potentially generate. Moreover, this issue cannot be resolved by considering further elements of existing classification schemes. As a consequence we propose a more specific scheme that uses *IDS characteristics* to describe IDSs. These IDS characteristics are defined by a two-tuple consisting of an *IDS description item* and an *IDS scope*. IDS description items denote generic, i.e., IDS scope-independent, properties of IDSs such as the capability of performing string-matching operations. In Appendix C.3 we provide an extensive list of examples of IDS characteristics and their definitions. Technically, i.e., in RIDAX, we use so-called *IDS description attributes* to represent IDS characteristics. Most of these attributes are booleans and simply represent whether a specific characteristic is present. However, some of them describe characteristics at pre-defined levels. For instance, for string-pattern recognition we distinguish between simple string matching and the more advanced regular expression matching. In practice this means that if an attribute represents the

characteristics "regular expression matching available," the lower-level characteristics such as simple string matching are considered to be present as well.

Example: *Consider the capability of Snort [Roesch99] to detect http buffer overflow attacks. We can easily describe the fact that the sensor portion of the IDS is capable of gathering application layer request arguments. This can be achieved using a single IDS description attribute. Using a second attribute, we can describe the fact that the detector is capable of searching for specific strings within application layer request arguments. Using these two attributes we are able to describe the core technique used by Snort to detect attacks such as buffer-overflow attacks, special-character attacks etc. However, it is clear that additional attributes are needed to describe the way Snort extracts the data from link layer PDUs etc.*

In order to enrich the expressiveness of our scheme further, we have introduced the concept of *instance analysis*. This concept encompasses the description of analysis techniques and extends them with the description of the level of abstraction at which the analysis is being performed. Moreover it specifies the analysis domain, i.e., the instance concerned, within which IDSs are capable of performing the analysis described.

5.1 A system model for IDSs

In our IDS model we split IDSs into a *sensor,* which gathers information from an information source, and a *detector,* which performs the analysis. IDSs may consist of several sensors and several detectors. For instance, an IDS may collect information from several daemon-specific log files that is then analyzed by a single detector. The model proposed represents a simplification of the CIDF model [CIDF98] that combines the so-called *a*- and *d*-boxes into a single element called *detector* (see also Section 2.1.2).

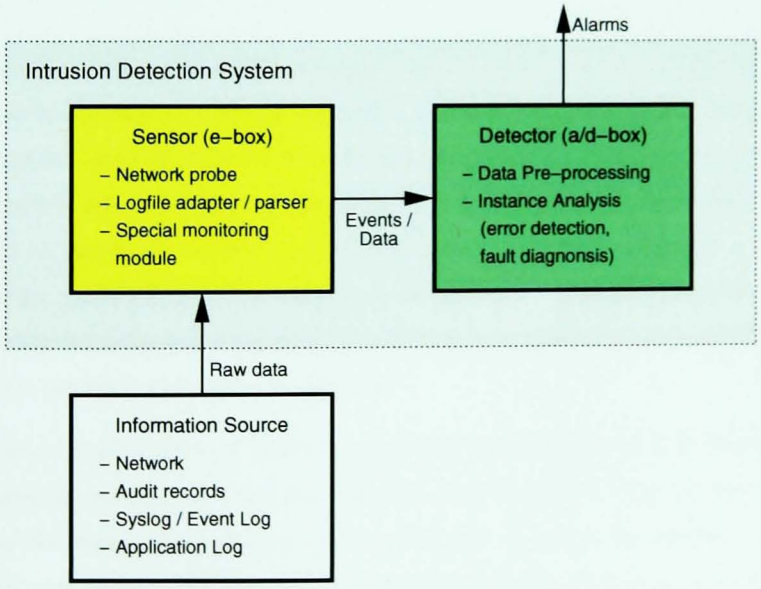


Figure 19—Intrusion detection system model

In the following we develop an IDS description scheme that is based on this simple model (see Figure 19). The model leads to the definition of two categories of characteristics—one for the description of sensors, the other for the description of detectors. In the following we address each of them in a separate section.

It is worth noting that in practice it is not always possible to draw a clear line between sensors and detectors, as shown in Figure 19. An example of such a system is the work described by Kerschbaum *et al.* in [KeSpZa00], where the IDS is embedded in the operating system. However, in general such systems still permit sensor and detector properties to be described separately.

5.2 Classification and description scheme for sensors

In our model, sensors are systems that transform the information provided by an information source into a form suitable for further analysis by the detector. To understand and describe the information the sensor passes to the detector, one has to take a closer look at sensor internals. In most cases (e.g., commercial products) this is not possible. Fortunately, most sensors are very simple and just provide some basic parsing of the data supplied by the information source only. In most cases such simple sensors can be accurately described by taking a close look at the documentation of and the information sources used by the IDS, and by investigating the diagnostic output the detector generates along with the alarms.

Figure 10, p. 32, provides an overview of the item categories that we use for describing IDSs. The figure includes the item sub-categories that we use to describe sensors and detectors. In the following subsections we develop all the IDS description items that we use to describe sensors. These items can be further distinguished as being either *IDS scope-independent* and *IDS scope-dependent*. In Figure 10, p. 32, all categories that are not explicitly marked as being IDS scope-independent are IDS scope-dependent.

One of the first observations made in this document was that the quality of today's IDSs is generally rather poor. In part this is caused by the sensor used, i.e., the information source, because in most cases the data transformation performed leads to loss of information. In other words, a sensor generally provides just a subset of the information available at the information source to the detector. Information may be suppressed on purpose because it is either believed not to be relevant for ID or because it is too costly to pass on the information and to analyze it. In addition information may be lost or damaged because of a failure in the information source, e.g., a misperception of a link layer PDU or because some system component is saturated and starts skipping data.

Example: *Consider a switched network that is monitored by a network-based IDS. This is generally done by configuring network switches such that they forward a copy of every PDU transferred to a specific monitoring port of the switch. If the overall traffic monitored surpasses the capacity of the monitoring port the switch will start dropping packets. Even if the monitoring port and the system hosting the IDS are well equipped (e.g., gigabit Ethernet), information might be dropped at the detector for similar reasons.*

Note that in this work we do not investigate the probabilistic aspects of whether a given activity will get processed. However, using the variations concept (see Section 3.2.2), we consider cases such as, for example, network PDUs that are not being received correctly by a network-based IDS.

5.2.1 IDS scope-independent sensor characteristics

The sensor characteristics defined in the following are independent of IDS scopes. This means they do not have to be combined with an IDS scope in order to define their semantics. Note that most of these characteristics do not influence the detection capabilities of IDSs, but instead influence the semantics of the diagnostic information generated by IDSs, i.e., they are merely relevant to the design of alarm-correlation systems. The only exception is the *information source type*, which may influence the analysis result significantly, i.e., influence the alarms that are generated.

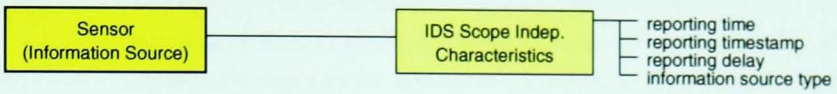


Figure 20—Overview of IDS scope-independent sensor characteristics

As Figure 20 shows, we have identified the following four IDS scope-independent sensor characteristics (the table headings describe the characteristic and the subsequent rows describe the permitted values):

Table 5—IDS scope-independent sensor characteristics

Reporting time	The reporting time denotes the point in time at which evidence of an activity is observed and reported to the detector. This property is not highly relevant to the detection process itself, but it may influence the timeliness of the IDS, i.e., in the context of this work this characteristic is not of high relevance. However, when developing alarm correlation systems for ID architectures the timeliness of the ID architecture components, namely IDSs, is relevant and might be taken into account.
Post-execution	*In the most common case a sensor will pass the data to the detector after the activity has been terminated i.e., post execution.*
During-execution	*In some rare cases, a sensor will report an activity after it has started but before it terminates, i.e., during execution.*
Pre-execution	*This last case does not describe IDSs as it analyzes activities before they actually happen. Such systems may deny an activity from being executed and therefore are merely policy-enforcement systems. For an example of such a system, see the work of Hutchison and Welz [HucWel00].*

Reporting timestamp	The reporting timestamp denotes the timestamp a sensor assigns to an observation when it is reported to the detector. Many ID sensors do not provide such a timestamp and leave it up to the detector to set a timestamp whenever it finds something worth reporting, i.e., whenever the IDS is issuing an alarm. Again, concerning the relevance of this characteristic to this work, the same as for the "Reporting time" characteristic applies.
None	*The sensor does not provide any timestamp information along with the reporting of an activity observed.*
Start of activity	*The timestamp provided by the sensor corresponds to the time at which an activity has started.*
End of activity	*The timestamp provided by the sensor corresponds to the end of an activity.*
Reporting delay	The delay between the point in time an activity is observed i.e., identified, and the point in time at which the activity is reported to the detector. Based on our experience with various sensors, we *arbitrarily* express that property in terms of the ranges given below. Concerning the relevance of this characteristic to this work, the same as for the "Reporting time" characteristic applies.
Less than 3 sec.	*It takes the sensor less than 3 sec to forward the data describing the activity observed to the detector.*
Less than 1 min.	
Less than 15 min.	
More than 15 min.	
Batch	*The sensor data is processed in batch mode. There is no fixed delay between the observation of an activity and the actual analysis of the activity data by the detector.*

5.2.1.1 Information source types

The information source type determines the view the IDS has of the system it monitors to a large extent. This fact is also taken into account in the IDS taxonomy by Debar *et al.* [DeDaWe00], where IDSs are classified based on what they call *audit source location* (see also Figure 7). However, knowing the location at which information is gathered is important and useful for the description of IDSs although it only *implicitly* describes the inherent properties of the information gathered from the respective sources.

Example: *Considering a network-based IDS gathering an URL from the network, it is impossible for the IDS to clearly determine how the webserver software will interpret the URL—especially if one additionally assumes the use of attack-obfuscation techniques. The situation is different if one considers a host-based IDS that analyzes the access logs as they are created by the webserver software. There the information is available in a form less prone to obfuscation and, more importantly, in the way the webserver has actually interpreted the respective request. In other words, in this example, the network-*

based IDS does not share the same view on the data as the monitored webserver, whereas a host-based IDS that analyzes the webserver logs shares the same view on the data. However, with respect to variations one has to note that the host-based IDS has only a limited view on that data.

Making such considerations, we have identified two main classes of information sources and a series of sub-classes. The two main classes are *raw data* sources and *log data* sources. The difference between the two is that raw data sources provide a nontransformed view of the data as it is part of an activity, whereas the log data sources represent a view on the manner in which data was interpreted by the monitored system, i.e., generally not on the data itself.

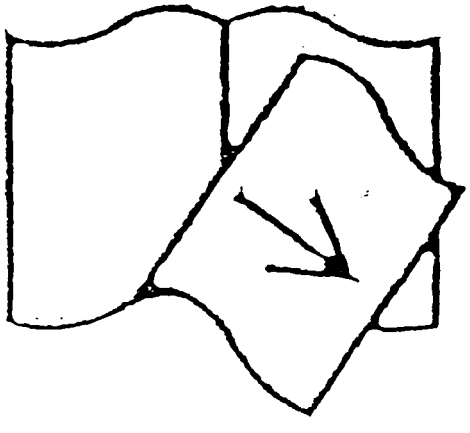
We further divide the two classes into a total of five sub-classes that mainly denote the location of the information source within the system monitored. The raw data class is sub-divided into an *external* and an *internal* class of information sources. External information sources provide a view of the raw data before it reaches its destination, i.e., the monitored system. This is where classical network-based IDSs on network sniffers fit in (examples: [CiscoNR99, ISSNet99, Paxson98, Paxson99, Roesch99]). Internal raw data information sources access the raw data at the system monitored and in the same IDS scope as the system monitored does. In this class mainly network-based IDSs (e.g., [ISSSer00]) that inspect the data on a host as it traverses the network stack have so far been implemented. However, it is conceivable that such sensors may also be implemented directly into applications or operating systems. Examples for such approaches are the work by Kerschbaum, Spafford and Zamboni [KeSpZa00, SpaZam00, Zambon01], who propose an IDS that is embedded into the operating system, and Almgren et al., who investigate a webserver sensor [AlmLin01] module that inspects transport layer data as it is received by a webserver daemon.

The second main class of information sources, the log data sources, can be divided into three sub-classes. Here we distinguish information sources provided by *operating systems* such as audit logs [LCRM98], or system accounting logs, sources provided by *applications,* and so-called *meta* information sources. Access logs are a typical example of application logs [Weinma98], as they are maintained by webservers. For the meta information source, the most typical examples are probably alarms as they are generated by other IDSs. These alarms generally include *interpretations* of observed activities. Such sources typically feed into alarm-correlation systems, and therefore are beyond the scope of this work.

Figure 11, p. 33, provides an overview of the information source type classification introduced and also includes the information source type examples mentioned above. In the table below we describe the information sources in more detail.

Pages Missing not Available

P78



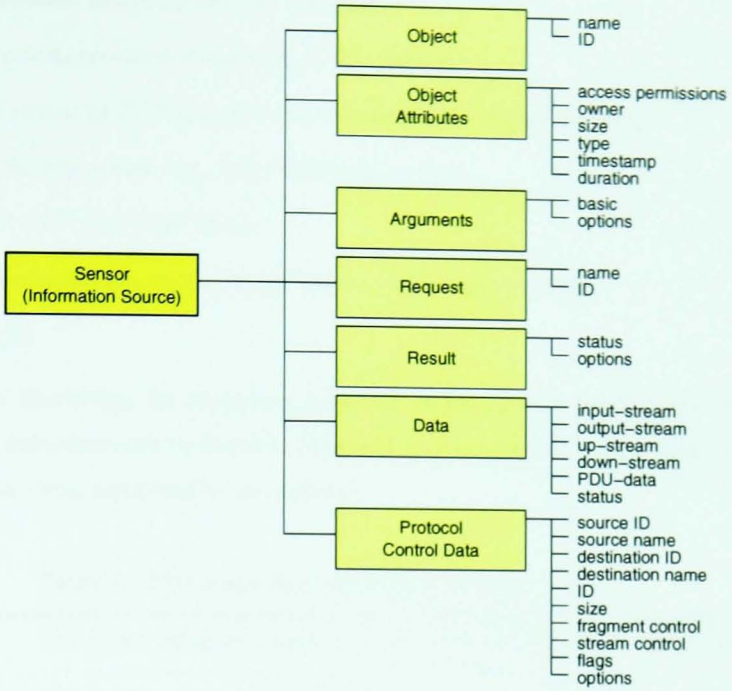


Figure 21—Overview of scope-dependent sensor items

Example: *In the following we provide brief definitions of sensor characteristics to illustrate how the sensor items shown in Figure 21 are combined with IDS scopes to define them.*

Sensor items \ Activity scopes	IP	TCP	UDP	HTTP	System call	Process	File
Object name						1	2
Basic arguments				3	4	5	
Optional arguments				6			
Up-stream data		7		8			
PDU-data	9						
status data						10	11

Figure 22—Examples of sensor attributes

The shaded fields in Figure 22 denote characteristics that are not defined because the combination of sensor item and IDS scope would not result in a meaningful characteristic. The numbered characteristics can be interpreted as follows:

1. *Name of the executable that was used to create the process.*
2. *Name of a file.*
3. *Arguments provided along with an http request*
4. *Arguments provided to a system call.*

5. *Arguments provided to a program, i.e., command.*
6. *http header options provided along with an http request.*
7. *Reassembled stream of TCP data as it can be found at the socket interface.*
8. *Data sent to the http server, e.g., http POST request data.*
9. *Payload of IP, TCP, and UDP PDUs.*
10. *Current state of a process, including the register and memory content.*
11. *Content of a file.*

This example aims at illustrating the pragmatic approach taken. Extensive examples of definitions of sensor characteristics definitions can be found in Appendix C.3.1. In the following tables we now provide definitions of all sensor items supported by our scheme.

Table 7—IDS scope-dependent sensor items—object

Object	The object category consists of two items (for examples, see Table 58):
Name	*The name of an object is generally human readable and, in most cases, uniquely identifies an object.*
ID	*An object identifier uniquely identifies an object by a numerical or possibly alphanumerical identifier.*

Table 8—IDS scope-dependent sensor items—object items

Object attribute	The object attribute category provides additional items required to describe an object. Note that the state of an object is captured in the data category. See also Table 59.
Type	*If an IDS is able to obtain the information on the type of an object, it can differentiate between similar objects in the same IDS scope, e.g., to differentiate among files, directories, links, etc.*
Access permissions	*Access permissions of a given object specify the objects, e.g., users etc., that are permitted to access the object.*
Owner	*The owner denotes the ownership of an object. This may include the notion of group ownership. Examples are Unix filesystem objects.*
Size	*The size of an object usually represents the storage or memory required to represent the object.*
Timestamp	*The timestamp item denotes timestamps such as creation time or login time.*
Duration	*The duration item denotes durations such as lifetime or the time consumed.*

Table 9—IDS scope-dependent sensor items—arguments

Argument	The argument category is used to represent arguments supplied to calls, process, requests, etc. We distinguish only two different properties (see also Table 60):
Basic	*The basic arguments represent the arguments directly associated with a request, call, etc.*
Options	*Optional arguments represent arguments that require the sensor to perform an additional effort to provide them.*

Table 10—IDS scope-dependent sensor items—request

Request	The request category is also very small. It is used to name calls, request etc. See also Table 61.
Name	*The name of the request.*
ID	*The ID of the request made.*

Table 11—IDS scope-dependent sensor items—protocol control data

Protocol control data	The protocol control data category is more complex than most of the other categories. It needs to capture the variety of protocols that have been defined. See also Table 62.
Source ID	*The source ID typically denotes the source address of the PDU considered.*
Source name	*Like the source ID, but a name is used instead of a numerical ID.*
Destination ID	*The definitions of the destination ID and the destination name is identical to the definition of the source ID and source name. The only difference is that they denote the receiver instead of the sender of a PDU.*
Destination name	*Like the destination ID, but a name is used instead of a numerical ID.*
ID	*The ID of a PDU helps a protocol to distinguish requests.*
Size	*The size field denotes the size of a PDU.*
Fragment control	*Fragment control information is used to reconstruct fragmented PDUs. The most typical example is probably the IP protocol.*
Flags	*Flags are used for a large range of functionalities.*
Options	*Protocols often offer a number of other fields and options that are not covered by the items listed above. We summarize these fields and options here.*

Table 12—IDS scope-dependent sensor items—data

Data	The data category is also quite complex. In this category we collect all properties that can be considered as data of any kind. As to be explored further in future work, one can conceive attack-obfuscation techniques that may render any kind of data related to an activity invisible to the detector. A typical example is the fragmentation of IP PDUs, which may make it impossible to analyze the payload of IP PDUs for IDSs that are not able to recompose IP PDUs. This effect can then be described by rendering the data portion related to the activity invisible to the detector. See also Table 63.
Input stream	*This item represents the stream of data fed into an object, e.g., a process.*
Output stream	*This item represents the stream of data generated by an object, e.g., a process.*
Up-stream	*This item represents the data that is sent to a server by a client. The most prominent examples are sockets, pipes, and application layer protocols such as http.*
Down-stream	*This item represents the data returned by a server to a client.*
PDU-data	*The item PDU-data represents the data portion of a PDU. This item applies for a wide variety of protocols such as UDP, TCP, ICMP, IP, MAC etc.*
Status data	*Status data represents the internal state of an object.*

Having provided the definitions of all sensor items, we are now able to describe sensors using sensor characteristics that are created by combining a sensor item and an IDS scope.

Example: *Consider the IDS WebIDS [Almgre99]. This IDS parses webserver logs in the common log format (CLF) [Weinma98]. Therefore its sensor is able to provide the following information to the detector (first listing the sensor item, then the IDS scope):*

- *Request name / http*
- *Basic arguments / http*
- *Source ID or source name / IP (depending on the server configuration)*
- *Object name / User (only if the server performs http authentication)*

5.3 Classification and description scheme for detectors

The detector is the IDS component that performs *error detection* and *fault diagnosis*. It does so based on the information provided by the ID sensor. The detector is generally the most complex component of the IDS. It is also the component that varies the most among the various approaches proposed and implemented by the ID community.

We have identified three categories of characteristics that distinguish the description of detectors. These sets, which have already been illustrated in Figure 10, p. 32, are called as follows:

- IDS scope independent characteristics: This category of IDS scope independent characteristics is very similar to that of the IDS sensors.
- Data pre-processing: The data pre-processing characteristics are IDS scope-dependent and define the operations a detector is able to perform on the data provided by the sensor before the data is analyzed for signs of errors.
- Instance analysis: The characteristics of the instance analysis category are also IDS scope-dependent, and describe the detector's capabilities to perform error detection and fault diagnosis.

In the following sections we develop these characteristics at the required level of detail. With respect to complexity, the first two sets of characteristics are similar to the sensor characteristics developed above. The third set is more complex and requires extensive descriptions. Numerous examples can be found in Appendices C.3.2 and C.3.3.

5.3.1 IDS scope-independent detector characteristics

The set of IDS scope-independent detector characteristics is similar to the set used to describe sensors. Namely, the alarm timestamp and the alarm delay characteristics shown in Figure 23 have very similar definitions to the corresponding sensor characteristics discussed in Table 5.

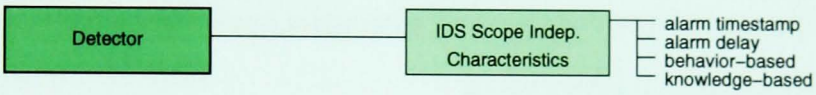


Figure 23—Overview of IDS scope-independent detector characteristics

However, the set of IDS scope-independent detector characteristics as defined in Table 13 also contains two characteristics that do not appear among the respective sensor characteristics. These characteristics are used to distinguish knowledge- and behavior-based detection methods as described by Debar *et al.* [DeDaWe99] (see also Section 2.2.3).

Table 13—IDS scope-independent detector characteristics

Alarm timestamp	The alarm timestamp denotes the point in time at which alarms are generated, with respect to the information reported by a sensor. In other words it tell us whether the alarm timestamp refers to the beginning or the end of a sequence of sensor reports that led to the generation of an alarm. The allowed values are identical to those of the *reporting timestamp* attribute discussed in Table 5.
Alarm delay	The alarm delay denotes the delay a detector adds between the reception of the last sensor report that leads to the generation of a given alarm and the actual creation of the alarm. The allowed values are identical to the ones of the *reporting delay* attribute discussed in Table 5.

Behavior-based	This boolean characteristic is used to denote the fact that the described detector applies a behavior-based detection method or at least uses a behavior-based component. See also Section 2.2.3.
Knowledge-based	This boolean characteristic is used to denote the fact that the described detector applies a knowledge-based detection method or at least uses a knowledge-based component. See also Section 2.2.3.

Note that it is conceivable that detectors combine knowledge- and behavior-based techniques. Because of this we have introduced two distinct characteristics to describe the detection method.

Example: *The IDS WebIDS [Almgre99] uses a database of signatures describing known attacks against webservers. It would therefore be described as being knowledge-based. DaemonWatcher [WeDaDe00], on the other hand, would have to be described as being behavior-based because it compares the observed system behavior with known models of normal behavior, i.e., it does not use attack signatures.*

5.3.2 Data pre-processing detector characteristics

In general the detector has to pre-process the data provided by the sensor because it may not yet be in a form suitable for further analysis, i.e., the data needs to be normalized. Also in some cases the detector suppresses data for various reasons such as preventing the system from being overwhelmed.

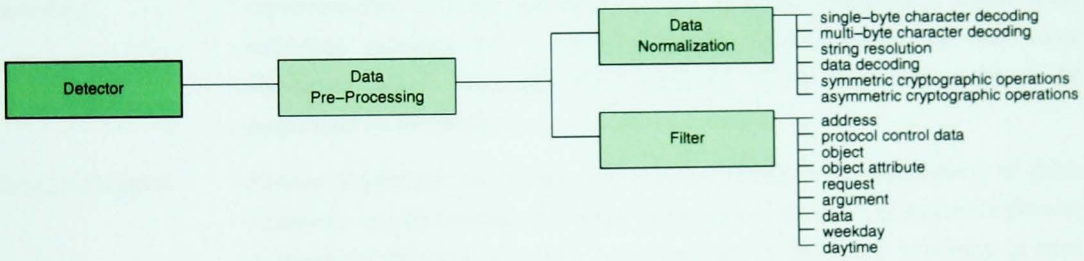


Figure 24—Overview of data pre-processing detector items

Figure 24 provides an overview of the data pre-processing items that can be described as follows:

Table 14—Data pre-processing detector items—data normalization

Data normalization	In particular application layer protocols often offer several different ways to express the same fact, i.e., it is often possible to formulate a sensor item using varying syntactical expressions that have the same semantic meaning. For examples see Table 64. Such a high degree of freedom in the representation of data enables adversaries to modify their attacks slightly such that it becomes significantly more difficult for IDSs to detect them. Sometimes this high degree of freedom actually enables the staging of an attack. For instance, this may be the case if the attacked object performs sanity checks on the data which would normally reject such suspicious data. However, if these sanity checks do not take into account the various data encoding techniques, the adversary might be able to stage an attack by encoding the malicious data—thereby bypassing these sanity checks. Generally speaking, data normalization is especially important for knowledge-based systems, because a detector's inability to normalize data may cause attack signatures to fail in matching suspicious data.
Single-byte character decoding	*Single-byte character decoding represents the ability to decode single bytes that have been encoded by some placeholder—typically their numerical ASCII value.*
Multi-byte character decoding	*Standards such as the UNI character encoding standard support the representation of large alphabets (more than 255 characters). Often those encoding schemes offer several possible representations for the same character, which increases the complexity of the decoding work to be performed by the detector.*
String resolution	*Escape sequences are frequently used to change the appearance of data. Examples are the quoting of strings, the use of a backslash character in front of a character that does not need to be escaped, or the use of shortcuts. If such techniques are used, an IDS needs to recognize them before performing any further analysis.*
Data decoding	*Data may be encoded in various ways and has to be decoded for meaningful analysis. Typical encoding techniques are the compression of data or the base64 encoding.*

Symmetric cryptographic operations / Asymmetric cryptographic operations	*We are not aware of IDSs that perform cryptographic transformations on the data they observed. However, it is conceivable that IDSs perform such transformations on the data they observe. In our case this applies to sensor items, as they were identified in Section 5.2.2. For instance, it is conceivable that a network-based IDS monitoring a webserver, that uses SSL (secure socket layer) to encrypt customer data and transactions, holds a copy of the webserver's private key. Knowing the webserver's private key enables the IDS to monitor encrypted https traffic. Note that we do not promote such solutions as they carry inherent weaknesses such as the cost in terms of processing, the problem of recovering from missing or corrupted PDUs, privacy issues, risks created by the fact that private keys are stored at multiple places etc.*

Table 15—Data pre-processing detector items—filtering

Filtering	Filtering may be an important measure to eliminate false positives or undesired alarms in general. A typical example is the filtering based on network addresses if a given host is known to cause many false alarms, even though the host itself is known to be harmless. For instance, this may be caused by a broken implementation of the host's TCP/IP stack, which generates many fragmented packets. Another cause might be a host that is used to scan the network for vulnerabilities and would therefore cause a flood of alarms to be generated each time a network scan is performed. Our model allows filters to be defined on every information category as defined in Section 5.2.2. In addition to those data categories, address data, weekday, and daytime have been added.
Object	*See Table 7.*
Object attributes	*See Table 8.*
Arguments	*See Table 9.*
Request	*See Table 10.*
Data	*See Table 12.*
Protocol control data	*Excluding address data. See Table 11.*
Address	*Address protocol control data is listed separately as it is one of the most important sensor items used in filtering rules.*
Weekday / Daytime	*We have extended the list by two notions of time period—weekday and daytime. These two time-period notions may be necessary for an IDS to eliminate alarms known to be caused by harmless, regularly occurring activities. An example are DNS zone-transfers that are scheduled on a regular basis.*

Example: *WebIDS [Almgre99] is an IDS capable of reversing the hexadecimal encoding allowed in URLs to represent non-printable characters. This corresponds to the data normalization capability of*

decoding single byte encoding for the http IDS scope. In addition the IDS supports filtering based on the URL, the request name and the source IP address. This results in the following list of data pre-processing attributes for WebIDS:

- *Single byte character decoding / http*
- *Argument-based filtering / http*
- *Request-based filtering / http*
- *Address-based filtering / IP*

5.3.3 Instance analysis detector characteristics

In this section we develop the part of the IDS description scheme used to describe the error detection and fault diagnosis capabilities of detectors. As expected, while developing the description scheme for sensors, it became apparent that different IDSs analyze the same activity from various different viewpoints. Accordingly this is also true for detectors. To address these differences, which in fact may influence the analysis performed by the detector significantly, we introduced the so-called notion of *instances,* which is then used to express the various detector characteristics. An instance represents the instantiation of an IDS scope, e.g., a process, an http request, etc. These instance-related characteristics are further divided into *analysis techniques* and *analysis levels*. Analysis techniques are used to describe the techniques, such as pattern matching, that detectors offer. Analysis levels describe the level of abstraction at which the analysis is performed.

However, before being able to describe the analysis techniques and levels, we have to fully develop the *instances concept*. If combined with an IDS scope, the instances concept can be used to identify the aspect of the IDS scope that is to be considered. An instance of http, for example, represents an http request. The result is as simple as it is pragmatic, but does not yet meet our requirements fully. IDSs often operate on parts of instances only or on multiple instances concurrently. We therefore generalized the instances concept by introducing the notions of *instance parts* and *instance groups* (see Figure 25). Using these additional notions the concept permits detector capabilities to be described, for example, with respect to single elements of an http request and http (v1.1) sessions that consists of several http requests.

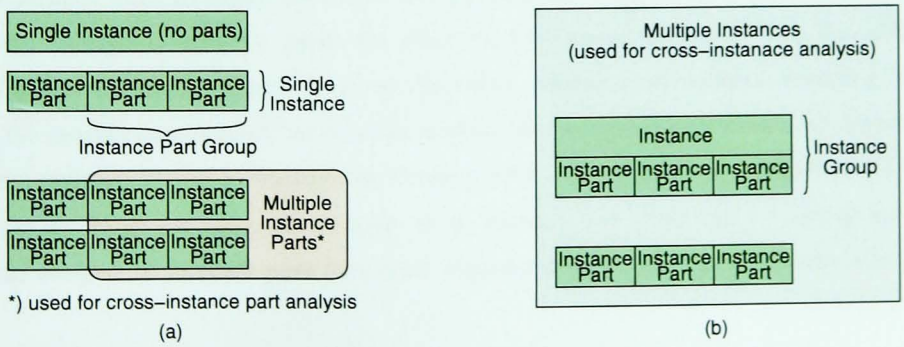


Figure 25—Concept of instances, instance parts and instance groups

Depending on the IDS scope of an instance, a detector may be analyzing *instances, instance parts,* or both (see Figure 25a). This can be illustrated further by considering an instance of the IDS scope IP. There it is intuitively clear that the instance is equivalent to an IP PDU. However, as mentioned earlier, an IP PDU can be split into so-called fragments. Using the more generic notion of instances, these fragments are equivalent to instance parts.

In addition these IP fragments are strongly interrelated because the receiver needs to be provided with the information required for recomposing the original IP PDU. The fact that a more or less strong dependency among instance parts and also among instances might exist leads to the introduction of *instance groups* and *instance part groups*. More generally, such groups consist of instances and instance parts that are—by definition—related at a higher abstraction level.

Even though instances and instance parts may be completely unrelated by design and therefore do not compose a group, they may nevertheless influence each other. In fact, many practical security problems arise because instances interact in a way they were not designed for or interact even though no interaction at all was foreseen. This also has to be addressed in this scheme. We do so by introducing so-called *cross-instance analysis,* and *cross-instance part analysis* reflecting the fact that multiple unrelated instances or instance parts are analyzed concurrently (see also Figure 25).

Example: *See Table 16, which shows the interpretation of the various instance-related terms.*

Table 16—Examples how to combine IDS scopes with instances etc.

IDS scopes / Instance notion	Processes	Application protocols	Link, network and transport layer (connectionless)	Link, network and transport layer (connection-oriented)
Instance part	Thread	Protocol statement or command	PDU fragment	Connection segment
Instance part group	Multiple threads of a process	Multiple statements of a transaction	Multiple fragments of a PDU	Multiple segments of a connection
Multiple instance parts	Unrelated threads	Seemingly unrelated statements	Unrelated fragments	Unrelated segments
Instance	Process	Transaction[12]	PDU	Connection
Instance group	Application / service	Session	N/A	N/A
Multiple instances	Unrelated processes	Unrelated transactions	Multiple PDUs	Multiple connections

Having developed the concept of instances, we now continue with developing the actual description scheme for detector analysis capabilities. The description scheme consists of two symmetric parts: one addresses the analysis of instance parts, the other the analysis of instances. This has already been introduced in Section 3.1.2.2. Figure 26 shows the entire scheme in more detail. Focusing on the part describing the instance part analysis, we consider instance part group analysis to be a sub-branch of cross-instance part analysis. We do so because the former can be considered to be a subset of the latter (see again Figure 26). More precisely, the analysis of an instance part group can be viewed as the cross-instance part analysis of instance parts that share common criteria. The same naturally also applies to

[12] We use the term transaction in the same generalized fashion as it was introduced in Section 4.1.2.

instances, and is illustrated in Figure 26, i.e., the two main branches are identical except for the fact that the upper one describes *instance part analysis* and the lower one describes *instance analysis*.

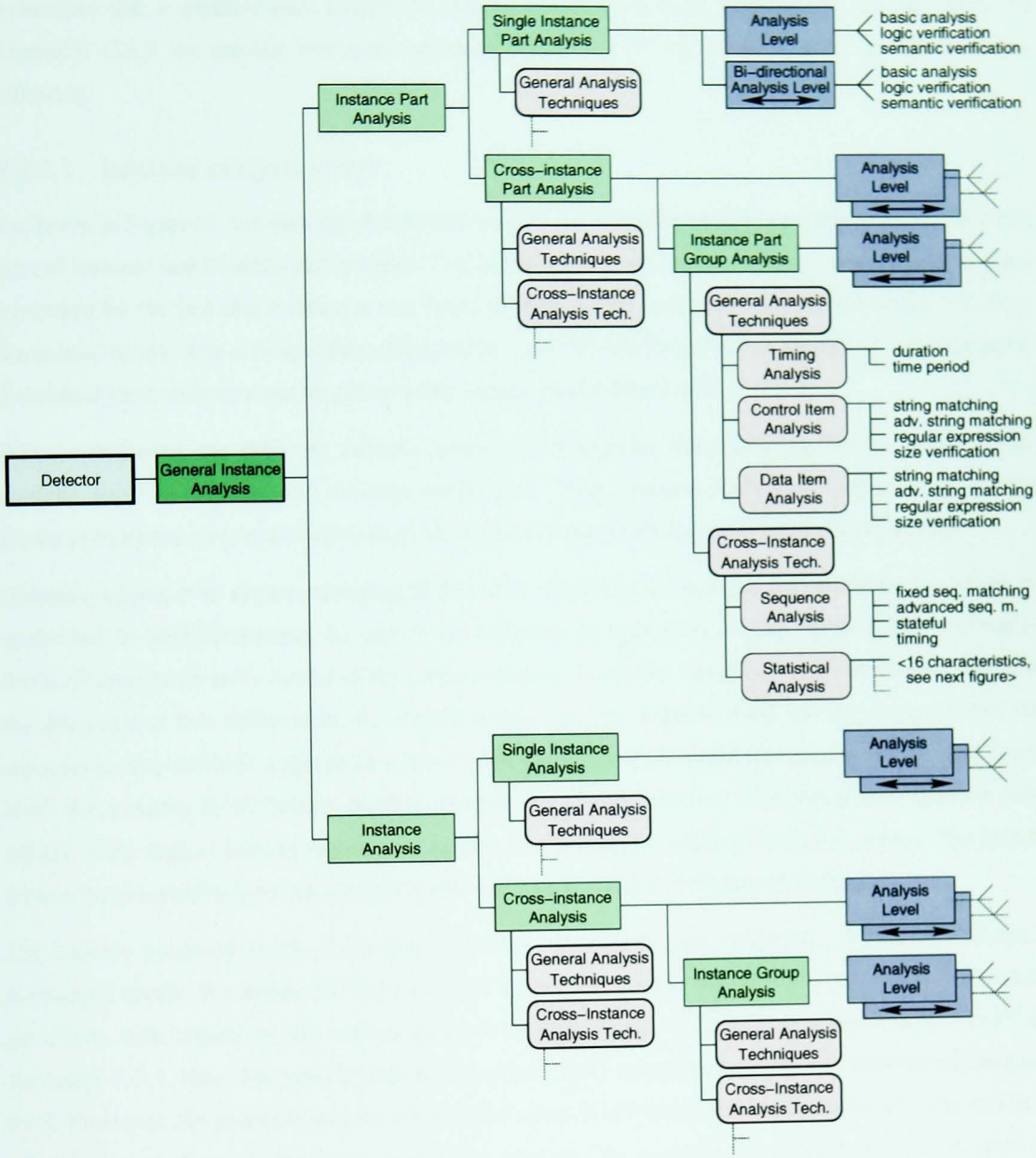


Figure 26—Description scheme for instance analysis

However, the scheme shown in Figure 26 consists of even more items that we develop in the following. We distinguish the following items:

- Instance analysis levels
- Generic analysis techniques
- Cross-instance analysis techniques

With the first item we aim at capturing the level of abstraction at which a detector is capable of analyzing instances and instance parts of a given IDS scope. With the remaining two items we describe the techniques that a detector may employ to achieve the analysis level identified by the first item. In Appendix C.3.3 we provide numerous example definitions for the characteristics developed in the following.

5.3.3.1 Instance analysis levels

As shown in Figure 26, we distinguish between *analysis levels* and *bi-directional analysis levels* for every type of instance and instance part analysis. The introduction of the notion bi-directional analysis levels is motivated by the fact that a detector may have to analyze protocols and other instances that are of bi-directional nature. The corresponding characteristics reflect whether an IDS is capable of associating bi-directional parts such as a system call or a http request and its return code.

When considering the different analysis levels, we distinguish three different levels, namely *basic analysis, logic verification,* and *semantic verification*. These analysis levels were inspired by Dobson's [Dobson89] abstraction levels and need to be interpreted separately for each instance analysis type.

Dobson's approach to system modeling [Dobson89] represents a systematic and hierarchical concept to model and to analyze systems. Its core is the modeling of systems at several hierarchically dependent levels of abstraction and a model of the communication among the various system components. Systems are described at five different levels of abstraction. The two highest levels are the *linguistic* and the *conceptual;* they describe a system in a non-formal but increasingly structured fashion. At the next lower level, the *semantic level*, Dobson starts to describe the system formally. This formal description is then refined in the *logical level* to explore the various viewpoints one might have on the system. The lowest level is the *descriptive level,* and deals with the technology used to implement the system.

The scheme proposed in the following represents an adapted and simplified version of Dobson's abstraction levels. We define the three analysis levels at a conceptual level and provide interpretation guidelines with respect to the various instance analysis types. Numerous examples can be found in Appendix C.3.3. Note that every higher-level analysis level comprises any possible lower-level analysis level. Moreover, for practical reasons, we consider every single instance that cannot be split into instance parts to consist of one single instance part. For example, we consider every single threaded process to consist of one thread.

Table 17—Instance and instance part analysis levels—basic analysis

Basic analysis	Basic analysis denotes the fact that a detector performs very low-level analysis such as simply recognizing the thing of interest, which can be an object, a request etc. The thing of interest is defined by the respective instance or instance part-analysis item sub-branch and the effective IDS scope.
Single instance and *instance part analysis*	*The detector identifies instances and instance parts, e.g., the IDS is able to distinguish protocol sequences or protocol statements. Based on this knowledge the IDS might apply further analysis, such as string matching, on the observed data. See also Table 65 and Table 66.*
Instance and *Instance part group analysis*	*The detector associates instances and instance parts as belonging to the same group. In the case of instance parts one can view it as the parts of an instance being associated by the detector. See also Table 67 and Table 69.*
Cross-instance and *cross-instance part analysis*	*The detector associates instances and instance parts that are formally unrelated. See also Table 68.*

Table 17 provides the definitions of the instance and instance part analysis level *basic analysis* with respect to the three different main types of instance and instance part analysis. We omit the repetition of the respective definitions for the bi-directional instance and instance part analysis levels because they strongly resemble to what is already defined in Table 17. They merely extend the corresponding definitions to the analysis of bi-directional instances and instance parts. We also do so in the definitions of the analysis levels *logic verification* and *semantic verification*, which will be discussed in the following.

Table 18—Instance and instance part analysis levels—logic verification

Logic verification	The analysis level logic verification denotes the fact that a given detector verifies the thing of interest at the logical level. Again, the thing of interest is defined by the respective instance or instance part analysis item sub-branch and the effective IDS scope. In most cases this is equivalent to syntax verification.
Instance and *Instance part analysis*	*The detector verifies the logical correctness of instances and instance parts. In most cases this is equivalent to syntax verification. In this context it is worth noting that in the domain of ID, instances need not to be complete or logically correct to be considered an instance. In fact many attacks are manifested by incomplete instances. See also Table 65 and Table 66.*
Instance and *instance part group analysis*	*The detector verifies the logical relation among instances and instance parts belonging to the same group. See also Table 67 and Table 69.*
Cross-instance and *cross-instance part analysis*	*As we are not aware that cases in which a logical dependency among instances and instance parts that by definition are unrelated exist, we do not further define this particular level.*

Again we omit the definition of the bi-directional counter parts. However, a simple illustrative example is TCP: there a detector is considered to be performing bi-directional logic verification if it verifies that the TCP-PDUs exchanged in both directions fulfill the protocol specification.

Table 19—Instance and instance part analysis levels—semantic verification

Semantic verification	If a detector performs its analysis at the semantic level it means that it verifies the semantic correctness and acceptability of the thing of interest. The thing of interest is defined by the respective instance or instance part analysis item sub-branch and the effective IDS scope. In most cases this is equivalent to the verification of security policy compliance. For example, the detection of the fact that a confidential document is being sent to some non-trusted party, using a perfectly valid mail transaction, i.e., mail message, falls into this category.
Instance and *Instance part analysis*	*The detector verifies the semantic correctness of the instances and instance parts. See also Table 65 and Table 66.*
Instance and *instance part group analysis*	*The detector verifies the semantic correctness of the relation among instance and instance part group members. See also Table 67 and Table 69.*
Cross-instance and *cross-instance part analysis*	*The detector verifies the semantic consistency and acceptability among instance parts and instances. See also Table 68.*

Bi-directional semantic verification can be illustrated by the example of a detector detecting the fact that a protected, non-public web page was revealed to the public. In order to do so, the detector has to recognize that an http request asking for a protected page is being fulfilled by the webserver.

Example: *To illustrate the description scheme for analysis levels just introduced, we consider an excerpt of the WebIDS [Almgre99] description. WebIDS mainly operates within the http IDS scope. Within this IDS scope it is able to verify the semantics of http request i.e., instances—even at the bi-directional analysis level. This means that it can not only detect suspicious requests (semantic instance verification), but also determine whether they were successful (bi-directional semantic instance verification). It does not perform semantic verification across requests, but is able to identify groups of requests i.e., instance groups, based on a common source IP address or user ID (basic analysis of instance groups).*

5.3.3.2 General analysis techniques

In addition to the instance analysis levels, Figure 26 also shows the instance analysis techniques. The corresponding characteristics describe detectors at a relatively high level by describing the resulting detector capabilities rather than their implementation. For instance, the stateful analysis of a sequence can be achieved using various techniques such as state machines or petri nets. (Although state machines and petri nets are not the same, they both represent a stateful technique to analyze sequences.)

The general analysis techniques apply to each of the six instance and instance part analysis types. We consider all these techniques separately, i.e., per analysis type. We do so because we need to reflect differences such as the fact that a detector is capable of performing string matching on IP fragments, i.e., instance parts, but not on groups of IP fragments or on completely recomposed IP PDUs.

As shown in Figure 26, we have identified three subsets of characteristics that describe techniques applicable to single instances and single instance parts:

- Control item analysis techniques
- Data item analysis techniques
- Timing analysis techniques

They describe the observable aspects of instances such as PDUs. We have, however, separated the pure data items from the control items to increase the level of detail of IDS descriptions. Consequently control item analysis techniques cover all the sensor items described in Section 5.2.2, but not the data items described in Table 12. These are covered by the data item analysis techniques instead. Note that not every technique represented by one of the items defined in the following is applicable for every type of instance analysis.

5.3.3.2.1 Timing analysis techniques

The set of characteristics denoting the timing analysis techniques is the smallest and can be represented by just two items:

Table 20—Instance and instance part timing analysis

Timing analysis techniques	
Time period	*The time period item denotes that the detector is able to verify whether the time period e.g., daytime, instance, instance part, instance group etc., observed is acceptable.*
Duration	*The duration item denotes that the detector is able to verify whether the time it took the monitored system to perform a task is acceptable.*

5.3.3.2.2 Control item analysis

Especially, but not solely, when verifying the logical and semantic correctness of instances, instance parts etc., control items need to be analyzed for their content. As illustrated in Figure 26, we have identified four additional items for expressing the corresponding detector characteristics. Three of them address the content and one the size of the control item.

Table 21—Instance and instance part control item analysis

Control item analysis	
String matching	*String matching allows a given sub-string to be identified within a string.*
Advanced string matching	*Advanced string matching additionally offers the possibility to go further than the identification of known strings by allowing case-insensitive matching and the use of "don't care" character placeholders.*
Regular expression matching	*Regular expressions generally allow a far more sophisticated specification of the matching conditions. Examples: Perl [Perl87] regular expressions.*
Size verification	*Size verification is a very simplistic check on the elements of a given instance. If the detector has a basic analysis of the instance in question only, size verification can be seen as a very limited syntax check. If the detector is able to verify the logical correctness of an instance, the size check may be applied in addition to identify suspiciously sized instance elements.*

Note that a detector may perform control item analysis without verifying the syntactical correctness of the instance. This actually is an important cause of false positives and false negatives. For example, IDSs that "blindly" apply string matching on protocol statements may generate erroneous reports of suspicious strings that are harmless or even normal in the context they appeared (this problem is similar to the one described in the example on p. 48). However, by applying string matching on protocol statements it is possible to perform some limited semantic verification, which is often used to identify undesired keywords in a flow of data.

Example: *Again considering WebIDS [Almgre99]. WebIDS is capable of applying regular expression matching on control items of the http protocol. This includes the URL, but excludes data such as data transfer in the context of an http POST request. Characteristics addressing the latter type of data are discussed next.*

5.3.3.2.3 Data item analysis

The set of items defined above in the context of control item analysis explicitly excludes the sensor (data) items introduced in Table 12. This is done because in real IDS implementations the data portion of an instance is often not as easily accessible as other sensor items related to an instance. An example is http, where the http request is often treated differently from the data associated with the request, i.e., the document served to the client or the data posted by the client (see also above example). As a consequence the capabilities of a detector with respect to data items may vary because of this.

We do not repeat the item definitions here because they are identical to those provided in Table 21.

5.3.3.3 Cross-instance analysis techniques

When considering multiple instances or instance parts concurrently, a detector can apply additional analysis techniques to those just identified. The two additional categories of analysis techniques that we were able to identify describe the detector's capabilities to verify the *sequence* of instances and instance parts and to analyze the *statistical* properties of instance and instance part sequences.

5.3.3.3.1 Sequence analysis techniques

The different types of sequence analysis that we identified are quite similar to those identified for the analysis of control items in Section 5.3.3.2.2. However their semantics differ to some extent as they address state transitions rather than string or information processing in general.

Table 22—Instance and instance part sequence analysis

Sequence analysis techniques	
Fixed sequence matching	*Fixed sequence matching allows a given sub-sequence to be identified within a sequence.*
Advanced sequence matching	*Advanced sequence matching also offers the possibility to go beyond a mere identification of known sequences by allowing the use of "don't care" placeholders and wildcards.*
Stateful sequence analysis	*In the case of stateful sequence analysis the detector analyzes a given sequence of instances or instance parts using a technique that keeps the state of past observations. Examples are state machines, petri nets etc.*

5.3.3.3.2 Statistical analysis

Statistical analysis of sequences, as implemented by IDSs such as Bro [Paxson98] or Snort's [Roesch99] port-scan preprocessor, is required to detect port scans, flooding attacks etc. The definition of these characteristics varies from those developed thus far. Instead of individual items we use four-tuples, each of which can be used to express a combination of detector characteristics. Each of the four item types represents a given characteristic that the statistical analysis performed by a detector may have.

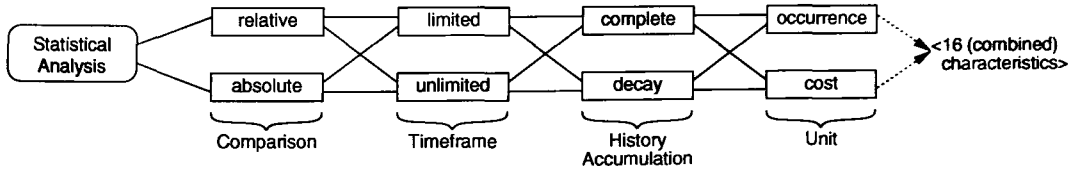


Figure 27—Overview of characteristics describing statistical analysis capabilities

As illustrated in Figure 27, the combination of these four characteristics results in a total of 16 combined characteristics that are used to describe statistical instance and instance part analysis techniques. Note that IDSs typically qualify for several of these 16 combined characteristics concurrently.

The following tables define the four characteristics, which have been identified by extending the work of others such as the Ph.D. thesis of Kumar [Kumar95], who has identified four different types of statistical measures. He, however, listed only four types of techniques found in the context of audit log analysis, i.e., these techniques were less general than those defined below.

Table 23—Statistical instance and instance part analysis—comparison

Comparison	Relative vs. absolute measure
Relative	*A detector is considered to be performing relative measurements of class instances and instance parts if it compares the measurements made for one class to measurements made for another class.*
Absolute	*A detector performing absolute measurements of instances and instance parts merely considers a class of instances without taking other classes into account.*

Table 24—Statistical instance and instance part analysis—timeframe

Timeframe	Limited vs. unlimited timeframe measurement
Limited	*The detector measures, i.e., counts, instances and instance parts with respect to a given limited timeframe. The resulting measurement is a frequency. This is commonly implemented using a sliding window.*
Unlimited	*The detector simply counts or accumulates instances, instance parts, or measurable properties of these. This means that the timeframe is unlimited.*

Table 25—Statistical instance and instance part analysis—history accumulation

History accumulation	Complete vs. decay
Complete	*The detector accumulates measurements made within the measurement timeframe without fading out older measurements.*
Decay	*The detector gradually decreases the weight of past measurements that were made within the measurement timeframe.*

Table 26—Statistical instance and instance part analysis—unit

Unit	Occurrence vs. cost
Occurrence	*The detector simply measures the fact that instances and instance parts occur.*
Cost	*The detector applies a cost function to the instances and instance parts observed. Typical examples are measurements of memory or CPU time consumed.*

Example 1: *We again use the now well-known WebIDS [Almgre99] to illustrate the description of statistical analysis capabilities. WebIDS may not calculate statistics on the requests made within a group of http requests, i.e., within an http v1.1 session, because the information source does not reveal this information, but may do so across all the http requests observed. It operates in limited and unlimited timeframes, i.e., with or without sliding windows, and it can do either complete or decaying history accumulation. Finally it only operates on the fact that an http request was made and not on the cost (e.g., number of served bytes) associated with a request. These characteristics can be expressed by the following attributes:*

- *Absolute, limited, complete, occurrence-based statistical analysis*
- *Absolute, limited, decaying, occurrence-based statistical analysis*
- *Absolute, unlimited, complete, occurrence-based statistical analysis*
- *Absolute, unlimited, decaying, occurrence-based statistical analysis*

Example 2: *Snort's [Roesch99] capabilities to statistical analysis can be described in an even simpler manner. The only such capability Snort v1.7 offers is implemented by its portscan detection module. This module operates at the transport layer and counts the number of TCP connection attempts to different ports made by a source within a given time frame. This also applies for UDP PDUs. As a result Snort's capabilities to statistical analysis at the transport layer can be expressed with a single attribute:*

- *Absolute, limited, complete, occurrence-based statistical analysis*

More recent versions of Snort provide increased capabilities for statistical analysis by including modules provided by external sources. An example is the SPADE/SPICE preprocessor provided by Silicon Defense [StHoMc00], which extends Snort with statistical anomaly detection capabilities.

5.4 Description of intrusion detection systems

In the sections above we have described an extensive and flexible description scheme for IDSs. In the context of this work we have created descriptions of the five IDS configurations. These IDSs and their selection are discussed in more detail in 8.6, where we use their descriptions as input to RIDAX. A complete IDS description example is provided in Appendix C.4. The IDS descriptions were stored in a database that was created using the database scheme described in following subsection. For the implementation of this IDS description storage facility we use a combination of open-source software:

- MySQL database [MySql].
- Apache webserver [Apache].
- PHP [PHP] scripting interpreter module.
- phpMyAdmin [phpAdm], a package of PHP scripts that allows a simple and efficient administration and population of the database.

MySQL is an open-source database that receives significant support in the Linux community and is easy to use and maintain. In addition interfaces to many applications and tools such as perl, Prolog, PHP, Apache etc. already exist. Lastly we needed a simple interface to maintain and populate the database which we implemented by using the phpMyAdmin package, which we operate on an Apache webserver. We have extended phpMyAdmin slightly to simplify the population of the database.

The RIDAX prototype directly accesses this database system to load IDS descriptions as well as to store the analysis results.

5.4.1 Database structure used to describe intrusion detection systems

In the context of the MAFTIA project [D3Maf01] and as a foundation for the RIDAX prototype we have developed and implemented a database structure that is capable of representing IDS descriptions created according to the scheme developed in this chapter. This also includes the specification of the sensors and detectors a given IDS consists of.

The database scheme consists of two groups of tables:

1. The first group of tables documents and reflects the description scheme developed in this chapter and the notion of IDS scopes including the IDS scope graph as introduced in Section 4.1.
2. The second group of tables is used to represent the IDS descriptions. Database consistency is ensured by referring to the first group of tables, i.e., using foreign keys.

The entity relation (ER) diagram shown in Figure 28 captures the first two groups of tables mentioned. The diagram itself is based on the notation by Elmasri and Navathe [ElmNav94], but was simplified to improve readability by suppressing the numerous ER-diagram attributes.

The high degree of symmetry in the entity relationship diagram is apparent. This is due to the fact that an IDS entity consists of either one or several *sensor* entities and one or several *detector* entities. The representation of the sensor and detector attributes is very similar. Both use the combination of the *IDS scope* entity and the sensor, or, respectively, the detector-specific item description entities to express their characteristics. In the case of the sensor entity the attribute description entity describes the various information items a sensor has the potential to provide (see also Section 5.2.2). The attribute description entity affiliated with the detector entity describes the capabilities an IDS has the potential to offer for the analysis of activities (see also Sections 5.3.2 and 5.3.3). As indicated in Figure 28 by double-lined borders, both attribute entities, i.e., sensor attributes and detector attributes, are so-called weak entities. A weak entity type is defined by the fact that its entries become unique only when also considering an externally supplied element. In this particular case all the relevant primary key information is supplied externally. In addition to the attribute entities, both the sensor and the detector entity are used to represent the generic characteristics as described in Sections 5.2.1 and 5.3.1.

The IDS scope entity reflects the concepts introduced in Section 4.1. It is not only used in the context of the representation of IDS component characteristics but also for the representation of the IDS scope

hierarchy, i.e., the IDS scope dependency graph. This is achieved by introducing a relation that defines one IDS scope to be an IDS sub-scope of another IDS (super-) scope.

Last but not least, the entity relationship diagram also requires documentary information to be included in the description of IDSs. Numerous description fields (suppressed in Figure 28 for ease of readability) and the introduction of the IDS vendor entity achieve this. The IDS vendor entity simply represents the fact that a vendor may offer more than only one IDS.

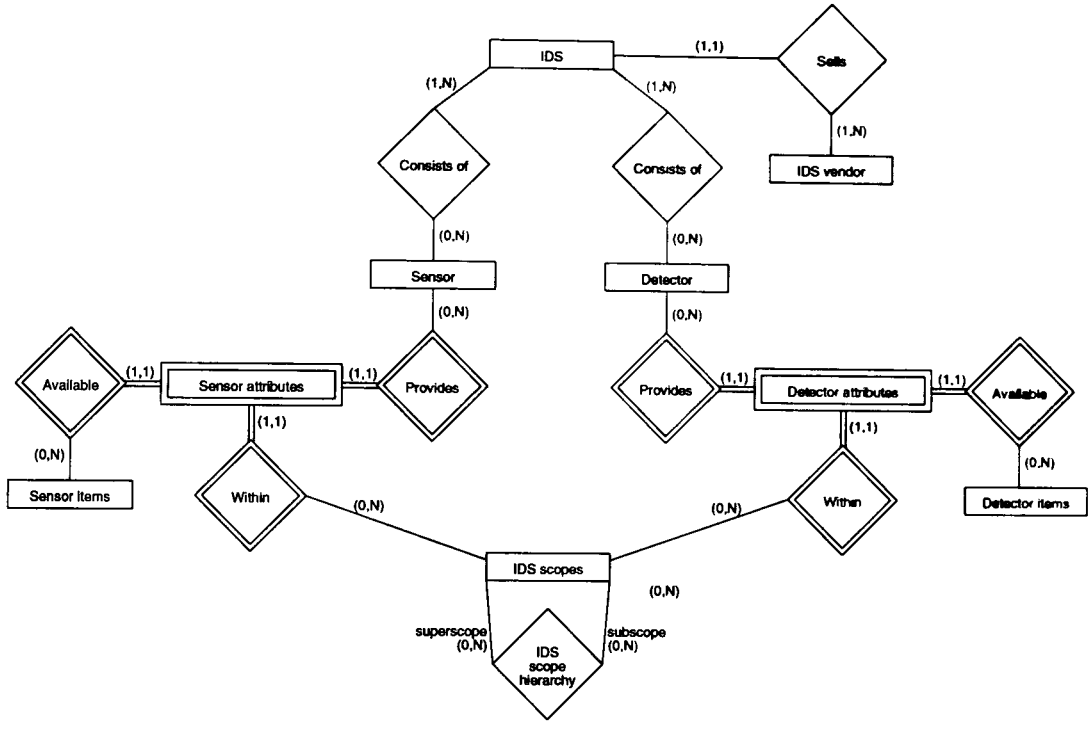


Figure 28—Entity relationship diagram of the database used to store IDS descriptions

5.5 Discussion

The IDS description scheme developed in this chapter represents an empirical, but highly systematic, approach to the description of IDSs. It is certainly true that the resulting scheme is too complex for somebody seeking informal classifications such as the ones proposed by Debar *et al.* [DeDaWe00], Axelsson [Axelss00], and others. However, as the goal was not the creation of an informal classification, but of a description scheme suitable for our pragmatic automated approach to IDS analysis, this issue is not relevant.

Given the hierarchical nature of the IDS scopes introduced in Section 4.1, one might argue that the description scheme as it has been developed thus far carries the danger of being ambiguous, i.e., that descriptions are not repeatable in exactly the same manner. The hierarchical nature of the IDS scopes allows an IDS to be described either at a more specific or at a more generic level at which both descriptions would be equally valid. This issue can be resolved by imposing a policy that defines the level

of detail at which IDSs shall be described (see also Section 3.1.3). One could, for example, require that IDSs are to be described as generically as possible or as detailed as possible. However, this issue becomes even more complicated because often detailed information about the capabilities of an (commercial) IDS is not publicly available. In such cases one has to estimate the capabilities of an IDS by observing its behavior when confronted with particular attacks and by investigating the type of alarms a given IDS is able to generate. This issue motivated the development of the Thor IDS analysis framework [Marty02]. However, in most cases, one can also obtain significant information by examining the information sources used by the IDS more closely. In doing so it is often possible to deduce facts such as the way IP fragments or TCP streams are being processed by an IDS. Finally note that because of how activities are described, the impact of differing IDS descriptions on our analysis approach, including RIDAX, is limited as long as the semantics of the descriptions does not differ significantly. As we shall see in the next chapter, one of the key design factors for attack class descriptions is that the descriptions have to operate based on the semantics of IDS characteristics and not based on specific IDS attributes that were used to express them.

Another point worth mentioning is the fact that the distinction between behavior-based and knowledge-based IDSs becomes less obvious. This is due to the fact that most attributes used to describe IDSs can be used to describe both of these IDS types. We have compensated for this drawback by introducing separate attributes (see Section 5.3.1) that allow us to describe the detection method employed by the IDS. It is interesting to note that these attributes primarily influence the types of the generalized alarms that are generated for a given attack class and far less the attack classes that are identified as being worth reporting.

Our scheme does not explicitly distinguish between IDS characteristics relevant to *fault diagnosis* and those relevant to *error detection* because in ID it is often not possible to distinguish them clearly (see also Section 2.1.1). This particularly applies to the more popular knowledge-based IDSs because they attempt to identify signs of known attacks and thereby often already perform some limited form of fault diagnosis. Once these systems recognize a known attack, they thereby detect the *error* and at the same time can already provide an indication about the cause of this error (i.e., fault diagnosis). The latter may happen in the form of an alarm identifier. In the case of behavior-based systems the ability to perform fault diagnosis is more limited because these systems generate more or less meaningless alarms once they detect that the system is no longer in an acceptable state. In other words, behavior-based systems detect errors that may lead to security failure and security failures implicitly.

5.6 Conclusion

The goal of our IDS description scheme is to enable the creation of IDS descriptions that are suitable for the analysis of IDSs as performed by RIDAX (see Chapter 7). As a result we have developed a scheme that achieves this and that thereby forms one of the foundations of this work. One might argue that the scheme is quite complex. It is certainly more complex than existing IDS classifications, but these are not meant to be used for the automated analysis of IDSs. Considering the goal pursued, our IDS description

scheme yields highly concise descriptions of IDSs that are modestly small (for an example see Appendix C.4). This is partially due to the fact that most IDSs offer only fairly limited capabilities. However, the use of (hierarchical) IDS scopes has an even more important impact on the size of IDS descriptions (see Section 4.1).

Moreover this scheme provides us with the building blocks for the description and classification of attacks. As explained in the subsequent chapter, attack classes are described by the set of IDS characteristics required for their analysis.

Chapter 6 Description and classification of attacks

For the analysis of a given activity, an IDS first of all has to collect a sufficient amount of data associated with the activity, and then analyze it. For this, the IDS needs certain capabilities. As outlined in Section 3.2, our approach describes attack classes, i.e., classes of malicious activities, by formulating these capability-requirements in terms of IDS characteristics that describe the capabilities of IDSs (see Chapter 5). More precisely we describe attack classes by means of the characteristics required of an IDS for analyzing a given class of attacks.

For the classification of attacks we use the same method. As explained in Section 3.2 we classify attacks by first describing them using IDS characteristics that are required for their analysis. The resulting description of the attack will be of reasonable detail, but not of sufficient detail such that it would be possible to implement the attack merely based on its description. Thus it is conceivable that several attacks can be described by the same description. In other words, such a description describes an entire class of attacks. Hence in the second step of the attack classification process, it is verified whether an attack class with an identical description has already been identified. If such an attack class exists, the attack to be classified can be associated to this class. In the other case, if no such class exists, the attack belongs to an attack class that has not yet been identified and the attack's description can be established as the description of the newly identified attack class.

Although the creation of attack class descriptions may seem relatively simple for a particular IDS and a given class of attacks, it becomes significantly more complex for a larger number of IDSs, attack classes and attack class variants. Each IDS may analyze attacks in a completely different manner. Moreover, depending on its capabilities, an IDS may choose from several different means to analyze a given attack. We address this issue by using the rule-based language Prolog [Diaz00] for describing attack classes. Its backtracking capability [CloMel94] enables us to explore the various ways in which IDSs may analyze classes of attacks efficiently and independently of specific IDSs.

However, although the problem can be addressed using a rule-based approach, special care has to be exercised to define a scheme that permits efficient, concise and repeatable descriptions of attack classes. In order to avoid the creation of large and repetitive descriptions of individual attack classes or even attack class variants, and to ensure consistency across descriptions of attacks that belong to the same class, we use a modular approach using several components:

1. Attack class description building blocks: Many attack classes share important characteristics, for example, they represent an application layer request or a network layer PDU. The building blocks express such shared characteristics by describing a particular aspect of entire classes of attacks[13] as identified in Chapter 4. These descriptions rely on the hierarchical nature of IDS

[13] In fact attack class description building blocks typically describe properties of entire super-classes, i.e., of multiple classes, of attacks such as "buffer overflow attack against a process."

scopes by describing attack class components at the highest-level IDS scope possible. Hence, the descriptions are made such that they can be instantiated at any lower-level IDS scopes, such as "IP" during the analysis process.

2. Attack class descriptions: An attack class description is a rule that describes classes of attacks as identified in Section 4.3. However, in practice these descriptions primarily specify the IDS scopes and the combination of attack class description building blocks relevant to the attack. The IDS analysis process will then analyze these attack class description building blocks for the IDS scopes specified by the attack class description.

3. Attack class variations: Attack class variations[14] are rules that express the additional IDS characteristics required to analyze classes of attacks, assuming that the latter have been altered slightly. Attack class variants can be defined by the name of the attack class and the list of variations included during the evaluation of the corresponding attack class description. The set of variations applicable to an attack class is identified based on the list of IDS scopes relevant to the attack class. In practice, variations are hooked into the applicable attack class description building blocks. For instance, the variation describing the fact that a network layer PDU has been fragmented is included during the evaluation of the attack class description building block describing the fact that a network layer PDU is involved.

4. Expectable alarms: Expectable alarms are *generalized alarms* that IDSs can be "expected" to, i.e., might, generate when observing a given attack. In this approach every attack class variant is associated to a list of expectable alarms, i.e., expectable *generalized alarms*, for two reasons. First, for a given attack, IDSs may generate more than just one correct alarm. Second, every IDS analyzes attacks differently, which means that multiple, semantically different alarms have to be accepted as true positives. For benign activities, the list of expectable alarms would of course be empty. Moreover, we prefer to associate expectable alarms with attack class description building blocks and variations rather than with individual attack class descriptions as this significantly simplifies the creation of attack class descriptions.

The following sections describe these components in greater detail, explaining their role in the IDS analysis process. In Appendix D, we provide BNF definitions for attack class descriptions and examples of attack class descriptions, attack class description building blocks, and attack class variations.

6.1 Attack class description building blocks

We introduce the concept of attack class description building blocks in order to simplify the task of creating attack class descriptions. In the context of this work, 23 attack class description building blocks

[14] For ease of readability, we hereafter refer to "variations" instead of "attack class variations."

were created. These were identified and described in the context of describing the 27 attack classes[15] identified in Section 4.3 (Table 4, p. 67).

Attack class description building blocks describe super-classes of attacks. i.e., they describe attack characteristics that are shared by multiple attacks classes. The attack class description building block describing argument-based buffer overflows is a good illustration of this. It was defined such that it can be used not only in the context of a specific IDS scope such as http but in an entire range of additional scopes such as the more generic application layer scope, the process, or the call IDS scope. Thus attack class description building blocks are highly reusable components for the description of attacks and attack classes. The relevant building blocks can be identified based on our attack categorization (see Chapter 4) by considering super-classes of attacks. Such super-classes unify multiple classes by permitting multiple values for classification attributes or by suppressing an attribute completely.

6.1.1 Instantiation of attack class description building blocks

Attack class description building blocks are highly generic building blocks that are specified at a high-level IDS scope. As such they are too generic when it comes to describing a specific class of attacks such as an http argument buffer overflow. This led to the concept of "instantiating" attack class description building blocks to a lower-level IDS scope when used for the description of an attack class. This becomes possible by using two IDS scopes to specify the validity of a building block. The first is the high-level IDS scope at which the attack class description building block was specified. This scope is fixed by the description of the building block. The second is what we call the *effective IDS scope* and represents a parameter that can be set whenever the building block is used to describe an attack class. Note, the instantiation of attack class description building blocks is not to be confused with the instantiation of classes as known from object-oriented technologies. The instantiation of an attack class description building block merely transforms a generic description into a more specific description.

Example: *Consider an attack class description building block that describes argument-based buffer overflow attacks at the application layer and the process IDS scope. By combining this attack class description building block and another attack class description building block that describes basic application layer requests, it becomes possible to describe an http request argument-based buffer overflow attack. To achieve this, we simply have to instantiate the two attack class description building blocks at the effective scope http.*

A practical example of how attack class description building blocks are instantiated is given in Section 6.2.1.

---

[15] At a later stage of this work, we used this method to describe also classes of benign activities. In total we created 48 descriptions of malicious and benign activity classes.

6.1.2 Analyzing multiple IDS types

Because attack class description building blocks have to address all the IDS types that have the potential of analyzing them, we formulate them in such a way that they permit the analysis of IDSs that apply different analysis techniques and monitor different types of data sources. For instance, there is a significant difference in the way the analysis is done by IDSs such as Snort, WebIDS or DaemonWatcher. Firstly, the information source (see Section 5.2.1.1) that is monitored differs in each case. Secondly, also the way the attack class is analyzed for suspicious elements differs significantly. WebIDS and Snort are similar in that they look for the presence of strings known to be suspicious. DaemonWatcher uses a completely different approach that analyzes the sequence of system calls of the monitored process for deviations from normal behavior. All these different paths of analysis need to be taken into account in the description of attack classes and attack class description building blocks.

Example: *Consider the attack class description building block describing a basic application layer request such that one can determine whether the IDS analyzed is capable of recognizing the fact that such a request was made. In the case where the information source used is raw network data, i.e., a sniffer as used by Snort, the IDS has to be capable of extracting the data first from link layer PDUs, and then from network layer PDUs etc. Once it is clear that the data required is available the analysis can be continued. It has to be verified whether the detector is capable of employing the required analysis techniques, and whether the analysis is conducted at an appropriate analysis level. For another class of IDSs such as WebIDS, the analysis part remains the same, but the verification of whether the data required for the analysis is actually available is different—again determined by the information source type. Finally, for the class of IDSs that monitors system audit logs, the requirements for the sensor and the detector are different again.*

In summary, the information provided by the sensor determines to a large degree the manner in which the detector analyzes a given class of attacks. However, the availability or absence of any IDS characteristic may enable or disable a given approach to analyzing a given class of attacks. It is also conceivable that an IDS may analyze an attack class using two or more different approaches, hence causing the analysis procedure to produce multiple results for the very same attack class. In the following, we assume that the IDS will always choose the most effective approach.

6.1.3 Dependencies among attack description building blocks

One could argue that in the ideal case attack class description building blocks should be independent of each other, because they describe independent analysis steps required from IDSs. For example, when designing a system communicating over an application layer protocol, one considers this application layer protocol to be independent of PDU fragmentation taking place at the networking layer. Normally this is certainly the case, but in the case of ID the layer and system boundaries are often not respected as precisely as one would wish. In fact many attacks involve such non specification-conformant elements

that violate the concepts of layer and system boundaries (e.g., data suddenly being executed in the case of buffer overflow attacks).

Example: *In the application layer buffer overflow example introduced in Section 6.1.1, it is conceivable that a network-based IDS is able to process normal network layer PDUs and to extract transport and application layer data as long as they are not fragmented. If they are fragmented, such an IDS might no longer be able to reconstruct them. In this particular case this would mean that although the IDS is capable of analyzing application layer request arguments, these arguments cannot be analyzed because the fragmentation either hides the data from analysis or causes the entire analysis process to fail.*

Although our attack class description building blocks can take such issues into account, they may lead to repetition and rather complex attack class description building block descriptions. To avoid unnecessary complexity of single attack class description building blocks and to guarantee consistency, we allow the inclusion of other attack class description building blocks within the description of any given attack class description building block.

Example: *The attack class description building block describing the basic analysis requirements for application layer requests when monitored by a network sensor requires that transport layer data be available. This is achieved by simply having the attack class description building block describing the application layer request require the attack class description building block that describes basic transport layer analysis. The latter attack class description building block then in turn requires basic network layer analysis etc.*

Thus attack class description building block descriptions may include any other attack class description building block, which is an ideal mechanism to describe dependencies such as they occur among the various network stack layers.

6.1.4 Example of an attack class description building block

Numerous examples of attack class description building blocks, attack class descriptions and attack class variations are given in Appendix D. The used prolog rules, variables and atoms are also described in this appendix. However, in order to illustrate the manner in which attack class description building blocks are constructed the following example is provided ("adbb" and "ADBB" are standing for "attack description building block"):

```
/* ----------------------------------------------------------------------
 * adbb/argBOF - buffer overflow attacks using arguments
 * i.e., the request line of protocol session or the arguments
 * of a function call etc. pattern matching based detection.
 */

adbb(basic, ADBB, ADBBScope, IDS, Detector, EffectiveScope, ScopeList,
    DIAGIN, DIAGOUT, Variations, Variations, notBlk) :-
        /* The detailed meaning of the argument list of this rule is
         * explained in Appendix D.2. This attack class description building
           block does not itself apply variations. */
```

```
    /* PART 1: Naming and IDS scope selection */
    /* The name of the attack class description building block: argument
       buffer-overflow */
ADBB=argBOF,
    /* The top-level scopes at which this attack class description
       building block can be evaluated: application layer and calls
       (see Figure 9). */
(ADBBScope=app_l; ADBBScope=call),

    /* Verify whether the scope list ScopeList contains a scope
       that is covered by the top-level scope. Moreover determine
       and/or verify the effective IDS scope for which the
       attack class description building block is to be evaluated
       (see Figure 9). */
selectSubScp(ADBBScope, EffectiveScope, ScopeList),

    /* PART 2: Information source requirements */
    /* Require the sensor to be able to provide the information needed
       for analysis: The request arguments (see Figure 21 and
       Table 9). */
reqSensorAttrib(IDS, _, EffectiveScope, args, basic, DIAGIN, DIAG1),

    /* PART 3: Detector requirements */
    /* Require the detector to be able to verify the logical correctness
       of the arguments (see Figure 26 and Table 18). */
reqDetAttrib(Detector, EffectiveScope, si_ip, logic, DIAG1, DIAG2),
    /* Require the detector to be able to either verify the string size
       or to apply string matching to the arguments (see Figure 26
       and Table 21). */
(reqDetAttrib(Detector, EffectiveScope, si_ip_info, size,
    DIAG2, DIAGOUT);
 reqDetAttrib(Detector, EffectiveScope, si_ip_info, string,
    DIAG2, DIAGOUT)), !.
```

The descriptions of attack class description building blocks, attack classes and variations generally consists of three parts as shown in above example. In the first part the name is defined and the effective IDS scope at which the attack class description building block is to be evaluated is selected or, if already given, verified. Then, in the second part, the information items that are required for analyzing the described attack class description building block are specified. In the last part the actual analysis requirements are specified. In Sections 6.2.1 and 6.3.4 similar examples will be given for attack classes and variations.

6.2 Attack class descriptions

So far we stated that attack class descriptions are primarily composed of attack class description building blocks. However, a complete description requires additional description elements, which leads to the following list of elements that activity descriptions may be composed of:

1. Attack class description building blocks as described above.
2. IDS characteristics required in addition to the ones formulated by attack class description building blocks.
3. The list of IDS scopes that are to be used for the instantiation of attack class description building blocks.

The second element addresses aspects of attack classes that are highly specific to the class described, i.e., that are too specific to be described by a dedicated attack class description building block. However, we

generally avoid their inclusions into attack class descriptions because they may make it more difficult to consistently repeat the attack description process.

The third element has multiple functions. First it specifies the IDS scope to which the attack class description building blocks have to be instantiated. Its second function, which will be explained in Section 6.3, concerns the selection of variations to be applied to the attack class.

Example: *The description of a buffer overflow attack may specify that it has to be considered at the IDS scope http and that TCP and IP have to be considered as the underlying protocols.*

6.2.1 Example of an attack class description

Most activity descriptions are fairly simple—provided appropriate attack class description building blocks have been identified and described in the first place. The following code excerpt is a slightly simplified example showing the rule used to describe an http buffer overflow attack using Prolog.

```
/* ------------------------------------------------------------------
 * attack/11/argBOF - httpd argument buffer overflow attack
 * ActNbr: 1
 */

attack(basic, Attack, EffectiveScp, IDS, Detector, ScpList, MaxVars,
        DiagIn, DiagOut, VarsToApply, BlockFlag) :-

        /* PART 1: Naming and IDS scope selection */
    Attack=argBOF[1], EffectiveScp=http[2],
        /* define the IDS scopes involved in this attack class */
    ScpList=[EffectiveScp, tcp, ipv4, ieee_802_3, sys_call][3],

        /* PART 2: Variations */
        /* select the variations to consider */
    selectVars[4](MaxVars, VarsToApply, ScpList, DiagIn, DIAG2),

        /* PART 3: Description of the actual attack class */
        /* This attack is in fact a layer 7 request */
    adbb[5](eval, reqCtlUp[6], app_l[7], IDS, Detector, EffectiveScp, ScpList,
        DIAG2, DIAG4, VarsToApply, VarsNotYetApplied, BLK1[8]),
        /* Verify whether the IDS was able to analyze the attack class
       description building block
         * reqCtlUp/app_l without being blocked due the lack of
         * characteristics. */
    relBlkFlag(BLK1, BlockFlag)[9],
        /* now check for the attack class specific things, i.e., the BOF */
    adbb(eval, argBOF[10], app_l, IDS, Detector, EffectiveScp, ScpList,
        DIAG4, DiagOut[11], VarsNotYetApplied, _, BlockFlag).
```

The rules and predicates used to describe this attack class are difficult to understand unless one is familiar with Prolog and the context. Being a rule-based language, Prolog will start at the top by trying to satisfy this rule by evaluating each predicate. These predicates may fail or succeed. In the case of failure Prolog performs backtracking [CloMel94] until an alternative solution is found. If no such solution is found, the entire rule fails. In this particular case backtracking is equivalent to the search of an alternative way of analyzing the attack class described.

The above example shows that attack class descriptions are composed of three parts. In the first part the attack class is identified and the list of relevant IDS scopes is defined. In the second part the set of

variations to apply is selected. Then, in the third part, the actual attack class is described. This description has to follow the path that information takes when being analyzed by an IDS, i.e., it starts with the prerequisites for any form of more advanced analysis (see also Section 6.1.3). In this example we first verify whether the IDS is actually aware of application layer, i.e., http, requests. Then, in a final step, the IDS's approach of detecting http buffer-overflows is analyzed.

In order to complete the description of this example, we have added superscript numbers in the example that denote Prolog clauses that represent important attack class properties such as the name of the attack class (for further details see Appendix D):

1. The atom `argBOF` denotes the generic name of the attack class. However, for a complete identification of the attack class also the following atom must be taken into account.
2. The atom `http` denotes the so-called effective scope, i.e., the primary IDS scope of the attack class. It is the second element that is used to name the attack class.
3. The list assigned to the variable `ScpList` represents the list of IDS scopes for which attack class description building blocks are to be evaluated (see also Section 4.2.1). For instance this list includes the IDS scope `ipv4` that triggers the evaluation of network layer attack class description building blocks for the IDS scope IP version 4.
4. The rule `selectVars` selects the set of variations to be applied to the attack class. This is done primarily based on the IDS scope list `ScpList`. This rule relies on backtracking and will provide a list of variations to be applied in the variable `VarsToApply`. This list will always have no more entries than specified in `MaxVars`.
5. The `adbb` rules are used describe attack class description building blocks (see Section 6.1).
6. This and the following atom specify the attack class description building block to be considered. In this case the atom `reqCtlUp` stands for the control information of a request sent to a server (i.e., no data included).
7. The atom `app_1` represent the second part of the attack class description building block name, and initiates the selection of the attack class description building block specified for the application layer.
8. After the evaluation of the `adbb` rule the `BLK1` variable is either set to `notBlk`, `adbbBlk` or `varBlk`. If the variable is not equal to `notBlk`, this means that the IDS was found to be able to analyze the attack class description building block in the selected analysis path. As shown in this example, in many cases the analysis of the attack class is not aborted because of an attack class description building block that could not be analyzed. A "blocked" attack class description building block simply means that there was an IDS capability missing that prevents the IDS from performing further analysis of the attack class on the path considered. The analysis performed so far may already be sufficient to raise certain alarms, e.g., that alarm that fragmented PDUs have been observed. The analysis that has been performed so far is recorded in the variables `DIAG2`, `DIAG4` etc. The final recordings are returned in `DiagOut`.

9. The `relBlkFlag` rule is a simple helper rule that is required to enable Prolog to continue the evaluation of the attack class if several attack class description building blocks or variations are to be evaluated in sequence. If `BLK1` has the value `notBlk`, `BlockFlag` will be an uninstantiated variable. Otherwise `BlockFlag` will be made equal to `BLK1`. At the end of the attack class description the variable `BlockFlag` carries the information whether any of the attack class elements, i.e., attack class description building blocks or variations, caused the IDS to stop further analysis of the attack class.
10. The atom `argBOF` and the following atom `app_1` specify that the attack class description building block describing argument-based buffer overflows at the application layer is to be included. The attack class description building block will be evaluated at the scope `http` that is specified by the variable `EffectiveScp`.
11. The variable `DiagOut` contains the diagnostic information needed for further analysis. This includes information such as the sensor and detector attributes required during the analysis, which then enable the identification of alarms that the IDS has the potential to generate.

Note that we only provide an example of an attack class description here. We do not include examples of attack class description building blocks, simply because in general their descriptions are large and complex, and would require far more detailed explanations than seem appropriate in this context. Instead, some examples have been included in Appendix D.

Moreover note that for the description of benign activities we may require certain IDS characteristics *not* to be available. This is necessary if we describe an attack-similar benign activity, which might be confused as a malicious one if the IDS is using simple analysis techniques, but that would not be confused if more advanced techniques were used (see also Appendix D). As a consequence the attack class will only appear as having been analyzed as an attack if the IDS was not sufficiently sophisticated.

Example: *Considering the SMTP (simple mail transfer protocol) example provided in Chapter 4, p. 48. There it is shown that IDSs, which are not capable of tracking the state of SMTP sessions, but just use a pattern matching technique instead, may confuse mail message data with SMTP commands. In this case we express this fact by letting the analysis of the attack class only succeed in its incomplete manner if the IDS does not implement the more advanced technique.*

6.3 Using attack class variations to create classes of attack variants

Attacks may be obfuscated in an attempt to elude IDSs [Horizo98, JiSilr00, PtaNew98, RFP00, SasBee00, Stewar99]. There exist tools such as Whisker [RFP00], Fragroute [Song02] and its predecessor Fragrouter [Song99] that implement such obfuscation techniques. Here we show how such obfuscating transformations of attack classes can be described by means of attack class variations. These are independent of, i.e., not associated to, particular attack classes. We also provide some background information and explain the difficulties of analyzing altered attack classes, i.e., attack class variants. In a next step we explain how these attack class variants can be created by applying variations to attack

classes. Finally we discuss the impact variations have on the analysis of an IDS when applied to attack classes.

Primarily knowledge-based IDSs that use some form of attack signature to identify known attacks are susceptible to obfuscation techniques. Generally the goal of varying attacks is to change the appearance of the attack such that existing attack signatures will no longer match. In real-world environments these techniques unfortunately proved to be fairly effective [Marty02]. To make matters worse, most variations can be combined with each other—especially if two variations affect different layers or sub-systems. To render IDS analysis even more complicated, variations may impact the analysis performed by IDSs in numerous ways.

However, one might argue that the problem of recognizing obfuscated attacks should not be difficult because the attack target is after all capable of reversing the transformation applied to the attack. Also, one might argue that unnecessary attack transformations are suspicious by definition. Unfortunately neither argument is true (see also Section 3.2.2) because

1. most techniques used to transform an attack also occur in the context of benign activities,
2. normalization is costly (see also Section 5.3.2), and
3. in some cases normalization is not possible—at least not in a non-ambiguous manner.

Most attack transformations used to obfuscate attacks are completely valid with respect to protocol specifications etc., and are frequently used in the context of benign activities such as the hexadecimal encoding of special characters in URLs.

Example: *Consider the fragmentation of IP PDUs. PDUs may be fragmented in an attempt to partition the data carrying the attack, but also because some entity in the network supports only a small MTU (Maximum Transfer Unit) size.*

The normalization of data as done by the attack target generally is a rather costly task. If done by the IDS it often significantly impacts the performance of the IDS and possibly the performance of the system being monitored. Good examples are the reassembly of fragmented IP PDUs or TCP streams.

For illustration purposes we cite the Snort documentation [Roesch99]. In the file `snort-lisapaper.txt` the following is stated:

> *... A Snort rule that has been tuned too tightly to key on a specific area of a packet's payload may overlook the real exploit that has been shifted to a different area within the packet. On the other hand, web CGI probes and attacks generally all take place at the beginning of the packet within the first thirty to fifty bytes. This can be a great place to optimize Snort content searching.*

This describes a way to optimize the performance of Snort. This optimization represents a restriction that might be used to elude Snort by obfuscating an http attack by artifically enlarging the http request line.

Another issue is that the normalization is not always non-ambiguous because multiple valid interpretations of the transformed attack class are possible, i.e., in some cases one can no longer call re-transformation "normalization". This increases the complexity of the analysis to be done by the IDS as it might have to investigate multiple re-transformations of an attack.

Example: *Consider overlapping IP fragments in which the overlapping data differs. Here it is not clear how the target will reassemble such fragments, i.e., it is not clear whether the target will give preference to the data contained in the first fragment or to the data of the last fragment. In fact, such differences exist between Linux and Windows operating systems. It is certainly true that this might be unusual for benign activities, but on the other hand, it is not necessarily an indication of malicious activity because a faulty network stack also may generate such PDUs.*

To make matters worse, most variations can be combined with each other—especially if two variations affect different layers or sub-systems. For instance it becomes possible to combine network layer [Horizo98] and application layer variations [RFP00].

Before continuing with a more detailed description of the principles used to describe variations, it is worth mentioning that we have selected and described seven attack class variations in RIDAX. These range from simply corrupted link layer PDUs and fragmented network layer PDUs to encoding variations at the application layer.

Generally, descriptions of variations are formulated in a fashion highly similar to that used for attack classes and attack class description building blocks. However, besides describing the IDS characteristics required for their analysis, they also have to support the following additional functions:

1. Define the IDS scope they are applicable to.
2. Represent the differing impact that variations may have on the analysis.
3. Define the variations and attack class description building blocks they may be combined with.

Last but not least, the variations have to be easily applicable to attack class descriptions and attack class description building blocks.

6.3.1 IDS scope of attack class variations

Similar to attack class description building blocks, variations are defined at a high-level IDS scope, possibly including IDS scope attributes, and then instantiated to the more specific IDS scope that is defined by the attack class description.

Example: *The variation used to investigate the behavior of an IDS with respect to the presence of fragmented IP PDUs is described for the network layer IDS scope, additionally requiring the "fragmentation" attribute. The latter is required in order to prevent the variation from being applied to an attack class involving a network layer protocol that does not support fragmentation. During the analysis of an attack class variant that includes this variation, the latter is, for example, instantiated for the IDS scope IP.*

As shown in Section 6.2.1, the list of IDS scopes relevant to an attack class is defined within the attack class description. It serves to select the variations that are applicable to an attack class (see Section 6.2.1, items 3 and 4). The attack class example in Section 6.2.1 illustrates the stage at which the variations to be considered are selected. However, the example does not show how the variations are applied. It is clear that variations need to be applied at that exact stage in the evaluation of the attack class description. For instance, the variation describing IP fragmentation should be considered before the IDS is analyzed for any of its higher-level capabilities such as the analysis of TCP streams. The solution to this issue is relatively straightforward because we can take advantage of the fact that we use attack class description building blocks to describe basic attack class elements: We simply include the evaluation of the corresponding variations in these attack class description building blocks. As a result the selected variations are automatically included at the appropriate stage of the analysis process, because attack class description building blocks such as application layer requests include further lower-level attack class elements such as TCP connections.

6.3.2 Impact of attack variations

Referring to the preceding section, taking variations into account at the appropriate analysis stage has also the welcome advantage that their impact can be modeled in a consistent manner. They may impact the evaluation of an attack class variant as follows:

1. Fatal impact: Any further evaluation of the attack class is blocked. In our description scheme this is reflected by setting a flag (see Section 6.2.1, item 11). This may result in a false negative.
2. Limited impact: Control items are suppressed or analysis techniques are rendered useless, but the evaluation itself is not blocked immediately. In our description scheme this is reflected by marking the affected IDS characteristics as "unavailable." As a consequence the evaluation may fail at a later stage and possibly result in a false negative, if one of the affected IDS characteristics is required for further evaluation of the attack class. However, it may turn out that the IDS has the potential of generating an alarm reporting the presence of a varied attack class, i.e., the possibility that an obfuscated attack has been staged. Note that in this case the stage of the evaluation at which the variation is considered is vital, simply because it does not make sense to suppress IDS characteristics after they have been used already.
3. No impact: The analysis is not affected because the attack class variant is normalized by some intermediate system before the IDS analyzes it. This is typically the case for IDSs that do not operate on raw data but use some form of log data (see Section 5.2.1.1). Generally this case does not need to be taken into account explicitly in the description of variations, as it is implicitly covered by the fact that variations are included in the appropriate attack class description building blocks.

Example 1: *Consider a network-based IDS. The fragmentation of network layer PDUs may inhibit any further analysis of the activity by a network-based IDS if the latter is not capable of recomposing*

fragmented PDUs. However, the IDS may be able to generate an alarm reporting the presence of fragmented PDUs, which is of limited use only. Fragmented PDUs do not represent a threat. Moreover, they are found quite frequently in networking environments where different types of networking technology are being used concurrently. In other words the alarm reporting the presence of fragmented PDUs is only useful when combined with an alarm reporting some actual threat. In this case the combination of the two alarms indicates that an adversary tries to stage an attack in an obfuscated manner, which itself may be an indication for the high severity of the attack observed.

Example 2: *Consider an http attack involving some insecure CGI script. If an adversary stages an URL-based attack against such a CGI script, the attack may be recognized fairly easily by some means of string matching performed on the URL of the http request. However, the adversary may attempt to hide his/her attack by obfuscating the URL by using the http protocol feature that allows every character in the URL to be replaced by its hexadecimal representation. This is a technique that has been implemented in the Whisker tool [RFP00]. If the IDS is not capable of reversing the hexadecimal encoding, simple string-matching algorithms will fail to discover the attack signature in the request submitted by the adversary. In this case, however, the impact of the variation is limited because it does not prevent the IDS from spending resources in analyzing the request. However, the variation renders its string matching-technique useless.*

Example 3: *Consider WebIDS [Almgre99] and assume that it is not capable of normalizing hexadecimally encoded URLs. The IDS would still be able to detect any known attack as long as the URLs carrying the attacks are not encoded. If we would apply the variation describing the fact that URLs are encoded to an attack, the IDS would be identified as not being able to reverse the encoding. In the case of encoded URLs, the URL-related pattern-recognition capabilities of the IDS become useless. We model this situation by having the variation marking the pattern-recognition characteristics as "unavailable."*

6.3.3 Combining attack variations

IDSs need to be analyzed for attack class variants including more than just one variation. This is necessary because nothing but technical limits prevents an adversary from obfuscating attacks in two or even more different ways concurrently to render their detection even more difficult (it is for instance possible to combine network layer variations [Horizo98] and application layer variations [RFP00]). In our scheme, variations are described independently of attack classes, so it is quite straightforward to apply multiple variations concurrently to an attack class. However, doing so will increase the number of attack class variants to be analyzed significantly. Moreover, some obfuscation techniques cannot be applied concurrently, for instance, because they abuse the same protocol features but in different ways.

Example: *As a generic example of contradictory variations one can consider two variations of which one represents the fact that PDU fragments are extremely large, whereas the other represents the fact that PDU fragments are extremely small.*

As a consequence, the description of a variation needs to provide information about the variations it may be combined with. This can be achieved using a relatively simple concept. We include a so-called *attack class variation index* with the description of every variation. Then, when selecting variations (see Section 6.2.1, item 4), we simply obey the rule that every variation selected must have a variation index not lower than and not equal to the index of all the variations already selected. If we now assign the same variation index to contradicting variations, we thereby ensure that they will never be applied to an attack class concurrently. Although this simple method has limited flexibility, it is sufficient for our purpose and also ensures that each possible combination of variations is considered only once.

Finally, note that any variation having a negative impact on the analysis of an attack class might also have a negative impact on the IDS's ability to deal with additional variations applied to the analyzed attack class. This is by no means different from the impact a variation may have on the analysis of any other attack class element. It, however, represents an additional reason why the order in which attack class description building blocks and variations are analyzed has to follow the natural path in which they are analyzed in the real world as well.

6.3.4 Example of an attack variation

The following example of a variation illustrates the analysis that an IDS has to perform when recomposing network layer fragments (typically IP fragments). Further details can be found in Appendix D.

```
/* ----------------------------------------------------------------------
 * variation/11/l3frgNotFirst - fragmentation on layer 3
 * This variation makes the assumption that the suspicious data is
 * distributed over multiple fragments, and that the IDS needs to be
 * able to recompose these fragments in order to recognize the attack.
 */

variation(blkChk, Variation, _, IDS, Detector, EffectiveScope, _,
        DIAGIN, DIAGOUT, _, _, notBlk) :-

        /* PART 1: Naming */
        /* the name of this variation */
    Variation=l3frgNotFirst,

        /* PART 2: Analysis capabilities required */
        /* We require the IDS to be able to recompose fragments */
        /* To do so the IDS needs to be able to deal with network
         * layer instances. In most cases this means IPv4 PDUs */
    reqIdeAttrib(Detector, EffectiveScope, si_i, basic, DIAGIN, DIAG1),
        /* In addition the IDS needs to be aware of instance parts,
         * i.e., IP fragments */
    reqIdeAttrib(Detector, EffectiveScope, si_ip, basic, DIAG1, DIAG2),
        /* Moreover the IDS needs to be able to associate instance
         * parts that belong to the same group, i.e., that belong to
         * the same IP PDU */
    reqIdeAttrib(Detector, EffectiveScope, ip_grp, logic, DIAG2, DIAG4),

        /* Part 3: Information items required */
        /* In addition certain layer 3 header information and data
         * is required */
    reqSensorAttrib(IDS, Detector, EffectiveScope, data, pdu, DIAG4, DIAG6),
    reqSensorAttrib(IDS, Detector, EffectiveScope, prot_ctl, frag_ctl,
        DIAG6, DIAGOUT), !.
```

The description of a variation as well as of attack class description building blocks and attack classes may consist of several rules. If the first one fails, i.e., the IDS is not able to analyze it properly, several alternatives may be considered by the Prolog engine. Each alternative rule describes another way (always expressed in terms of the IDS capabilities required) how IDSs may perform their analysis. If, however, no alternative is found, a last rule may handle this case by marking specific IDS capabilities as "unavailable" (for possible consequences see Section 6.3.2). Such a rule lets the evaluation continue and does not activate Prolog backtracking. Then, at a later stage, when the IDS would be required to perform a given analysis, this analysis may fail because the IDS is not anymore able to perform it owing to the suppressed capabilities. This can be used to model the loss of information that occurs if, for instance, an IDS is not able to deal with fragmented IP PDUs (see example 1 in Section 6.3.2). In this example case the data portion of the PDU will not be available to any consecutive attack class description building block. The purpose of letting the evaluation continue in such a case is to determine what the IDS is able to achieve without the missing information or capabilities. In many cases the IDS may still be able to generate alarms that indicate, for instance, the presence of fragmented IP PDUs. See Appendix D.4 for implementation details.

6.4 Expectable alarms

So far we have developed the description scheme for attack classes and variations, but have not yet explained how the results of their analysis can be used to verify whether the IDS analyzed meets the expectations, i.e., specification. To do this, it is necessary to know the generalized alarms one expects the IDS to generate for any attack class. These expectations are tightly coupled to attack classes, attack class description building blocks and variations, and will be described in the following.

However, note that the IDS analysis process itself will only be explained in the next chapter. This includes the specification of the so-called *alarm conditions,* which are an integral part of the IDS analysis procedure. They are used to judge whether an IDS design is capable of analyzing an attack class such that it can be considered capable of generating any given generalized alarm.

Let us therefore assume that an IDS design has been analyzed for an attack class and was found to have the potential of generating a set of generalized alarms. To judge whether these alarms are to be considered correct, i.e., true positives, we need to extend the description of attack classes by this additional information. However, this is not trivial, because

1. the application of variations to attack classes may change the set of expectable alarms, and
2. IDSs may report the same attack class using differing generalized alarms.

Moreover, for reasons of practicality in terms of development and maintenance, it seems desirable to couple the specification of the expectable alarms to attack class description components rather than to the top-level attack class descriptions. This is achieved by a simple solution that also resolves the issue that variations dynamically change the set of expectable alarms. In our solution we specify the expectable alarms per attack class description building block and variation. Only if inevitable, we also specify

expectable alarms for descriptions of attack classes. This may be required if activity descriptions themselves contain description components other than attack class description building blocks that may also cause the generation of legitimate alarms. Consecutively the specification of an expectable alarm is composed of the following four attributes:

1. Component type: Attack class description building block, attack class or variation.
2. Component name: Name of the attack class component. (For examples see Section 6.2.1, items 1, 6, and 8.)
3. Generic alarm name: Name of the alarm (this will be discussed in Section 7.1.2.1).
4. Alarm IDS scope: IDS scope at which the alarm condition is formulated.

When analyzing the set of generalized alarms that an IDS was found to have the potential to generate for a given attack class, we first compose the list of expectable alarms. We do so by unifying the lists of expectable alarms for every attack class component involved—including the alarms expectable owing to the presence of variations. It is clear that if analyzing a benign activity we only expect alarms indicating the presence of variations. In a next step we verify whether every attack class component for which at least one alarm is to be expected was reported. If, excluding variations, this is not the case, we consider an attack class as *not detected* (see also Table 27). Obviously we may also determine IDSs to generate alarms that are not among the ones on the list of expectable alarms. These alarms represent *false positives* (see Section 7.1.3).

Table 27—Rating of IDS analysis results based on lists of expectable alarms

Activity consists of (excluding activity variations)	None of the expectable alarms generated	At least one alarm, but not one per malicious activity component, is generated	For every attack class component at least one expectable alarm is generated
Benign activity components only	Benign (correct silence)	N/A	N/A
Attack class components	Not detected	Partially detected	Detected

The reason for considering an attack class component "detected" if at least one of the expectable alarms has been generated is that different types of IDSs report the same attack class or attack class component differently.

Example: *The following example is based on the argument-based buffer overflow example in Section 6.1.1. In this case an IDS may generate an alarm indicating the fact that a suspicious string was found. However it may also generate an alarm indicating that the execution path of the monitored process deviated from its normal path of execution. For this example the list of expectable alarms would contain both alarms in their generalized form. As explained, the attack would be considered "detected" as soon at least one of the expectable alarms is generated.*

Finally it is clear that we also use the list of expectable alarms to identify false positives and false negatives. If an alarm that is considered "expectable" is not generated, we consider this a false negative. However, as shown in Table 27 this does not necessarily mean that we consider an attack as "not

detected." If we find that an IDS has the potential to generate non-expected alarms, we rate these as false positives.

In the following chapter we describe the process of identifying and rating alarms. In Chapter 8 we will then show how these analysis results can be used to assess IDSs.

6.5 Discussion

Working with the attack classification and description scheme developed in this chapter has revealed that the scheme is well suited for constructing descriptions of attack classes in an efficient manner. While describing attack classes, we generally try to use attack class description building blocks rather than creating highly complex attack class descriptions. Whenever possible we try to create such building blocks that describe aspects of attack classes in a highly generic manner, which helps reduce the effort of creating new attack classes descriptions significantly over time because we can reuse an increasing number of attack class description building blocks.

In Section 2.2.1 we have identified a set of requirements that our attack classification would have to meet. Given the attack class description scheme that we have developed we are confident that all these requirements are met. The in-depth discussion of the manner in which this was verified will be provided in the next chapter, where also further validation issues will be discussed.

One concern one might have with respect to the way we describe attack classes is the fact that we describe a number of different alternatives of how we envision an IDS could analyze attacks. It seems that IDSs that were not taken into account during the creation of the attack class descriptions cannot be evaluated accurately. However, we believe this to be an issue of limited importance only. On the one hand the descriptions created, especially those of attack class description building blocks, are highly generic. Thanks to their generic nature and modularity, it is possible to cover a large number of ID approaches by describing only relatively few analysis approaches. On the other hand it is still true that certain IDSs may not be evaluated accurately using the existing descriptions. Here we have to emphasize that the existing attack class descriptions can be extended or modified to support the analysis of new types of IDSs with only little extra effort. This can be achieved by enriching existing attack class description building blocks with additional alternatives to analyze the attack class element they describe, thereby extending the range of IDS types covered fairly efficiently, without having to revisit every single attack class description.

The concept used to create attack class variants by means of attack class variations proved to be very effective in broadening the scope of our evaluation efforts. As we shall see in more detail in the remaining two chapters, the use of attack class variants increases the level of detail at which IDSs can be analyzed. Moreover, it permits the analysis of IDSs in closer to real-world situations than would otherwise be possible. However, the attack class variation concept has to be used with care to avoid causing misleading results by applying variations to attack classes where this would be inappropriate.

Example: *It does not make sense to apply the attack class variation describing the use of very small TCP segments when considering a TCP SYN-flooding attack [CA2196, SKKSSZ] because one would not be able to identify a meaningful data flow in the context of such an attack.*

Finally note that the scheme presented can equally well be used for describing of classes of benign activities. As explained in Section 3.6.2, this is possible because the intent (benign or malicious) of an activity cannot be observed by an IDS.

6.6 Conclusion

In this chapter we have developed a highly flexible, modular, and extensible description scheme for attack classes and attack class components that can even be extended towards the description of benign activity classes. The scheme, which is based on concepts developed in Chapters 4 and 5, supports the consideration of attack class variants, which proved to enrich the IDS analysis process significantly. It is clear that any such scheme will never cover all existing and upcoming IDS types upfront. In practice our scheme proved to be of high generality, because it is, for instance, capable of catering for IDSs as diverse as a host-based or a network-based systems using a single rule describing a buffer overflow attack against network services.

Chapter 7 Analysis of intrusion detection systems

In this chapter we develop what has been outlined in Section 3.3: A method that performs a combined analysis of IDS descriptions and attack class descriptions. Note that the analysis described in the following could be generalized from the analysis of attack classes to the analysis of activity classes that may be benign or malicious (see Section 3.6.2).

After having presented the method and the prototype implementation RIDAX, we finally explore possibilities of validating the results produced by this method and discuss the challenges associated with validation.

7.1 IDS analysis process

The actual IDS analysis process (see Figure 13) consists of two steps, which are repeated systematically for every individual IDS description and attack class variant. Each iteration analyzes a given IDS for a given attack class variant. In our RIDAX tool, this process is fully automated and takes advantage of Prolog's backtracking mechanism to systematically select IDSs and attack class variants for analysis. RIDAX even goes a step further and performs a rating of the generalized alarms in order to determine whether they can be considered a true or a false positive. The latter is of particular interest if our method is used for analyzing attack classes as well as classes of benign activity.

Figure 29 provides a more detailed overview of the data flow required for each iteration of the IDS analysis process as implemented by RIDAX. The process starts with the analysis of an attack class, i.e., attack class variant. This first analysis step provides a description of the capabilities that the IDS being evaluated has employed to analyze the attack class variant considered. These capabilities are described in terms of IDS characteristics. The process then continues with the alarm analysis step, which uses the IDS characteristics that were required to analyze the attack class variant considered as input. This alarm analysis step generates the list of generalized alarms that the IDS being evaluated has the potential of generating for the attack class variant considered. In the final step, these generalized alarms are rated whether they represent a true or a false positive. Moreover it is determined whether the IDS has the potential of suffering from false negatives, i.e., of not reporting attack classes as expected. This analysis also includes the rating of attack class variants, i.e., whether they have been detected, partially detected or not detected (see also Section 6.4).

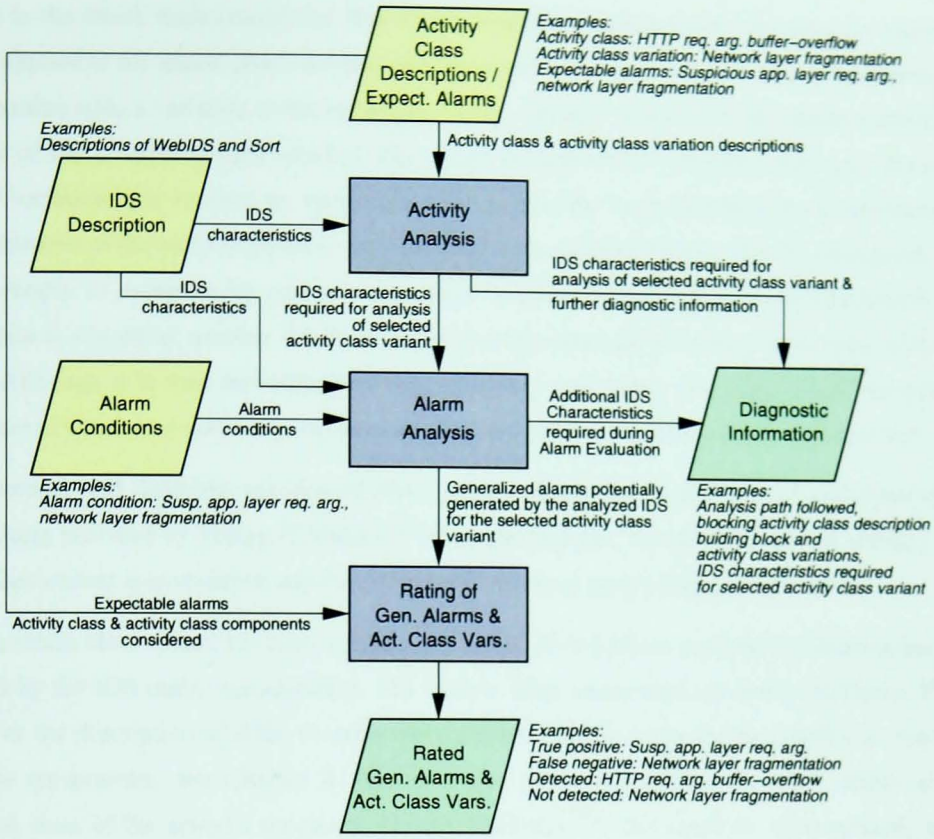


Figure 29—Overview of the data required in and generated by each iteration of the IDS analysis process, including examples

7.1.1 Attack class analysis

The first step of the IDS analysis process (shown in Figure 29) is the examination of the way that IDSs analyze attack classes and their variants. This requires the IDS descriptions and the attack class descriptions as input.

The analysis starts with the selection of the attack class variant for which the IDS is to be analyzed. This selection is made in a hierarchical manner, i.e., first the attack class to be analyzed is selected from the list of available attack classes. In the second phase, the set of variations to be applied to the attack class is selected. A new attack class is only selected for analysis if all applicable combinations of variations have been considered. Similarly, a new combination of variations is only selected if all possibilities of how the IDS considered may analyze the selected attack class variant have been examined. This means that the IDS analysis process might iterate multiple times for the same attack class variant—each time exploring a different analysis approach offered by the IDS (see also Section 6.1.2).

The selection of the set of variations that is to be applied concurrently to an attack class is made by means of a recursive algorithm that relies on the attack class variation index introduced in Section 6.3.3. The example attack class description provided in Section 6.2.1 illustrates how this selection is initiated by means of the `selectVars` statement (described by item 4, p. 109). Based on the list of IDS scopes

relevant to the attack class considered, this simple algorithm systematically searches for variations that may be applied to the attack class, and that may be combined with the set of variations already selected. The algorithm adds a variation to the set of previously selected variations if its variation index is higher than that of any of those already selected. The search for applicable variations stops once the number of selected variations has reached its maximum or all applicable variations have been considered. In the implementation of the analysis process, we have limited the number of concurrently considered variations to two, simply to constrain the number of possible combinations that have to be considered for every attack class to a practical number. However, this restriction does not influence the viability of the analysis results. Although it is true that variations may influence each other, our experiments did not reveal a single case in which two variations influenced a third any differently than a single variation did.

The systematic and complete selection of attack classes as well as the selection of variations rely on the backtracking provided by Prolog [CloMel94]. It thereby ensures, in a straightforward manner, that each attack class variant is considered, and that all possible analysis approaches are examined.

Once the attack class variant has been selected, the attack class analysis proceeds by determining how it is analyzed by the IDS under consideration. The various input items used are shown in Figure 30 in more detail. For the description of IDSs we refer the reader to Chapter 5; for the description of attack classes and their components, see Chapter 6. Owing to the rule-based manner in which attack classes are described, most of the analysis process is defined implicitly. At this stage we systematically search for ways to analyze the attack class variant considered using only the capabilities provided by the IDS under analysis. While doing so, we record all IDS characteristics employed for analyzing the attack class.

Example: *Consider an http argument buffer overflow attack being analyzed by a knowledge-based IDS such as WebIDS [Almgre99]. In this case one typically obtains an analysis result that shows that the IDS was analyzing the http instance at the semantic analysis level using a string-matching technique or possibly simply verifying the request length. In addition the result would reflect all sensor items required for the analysis.*

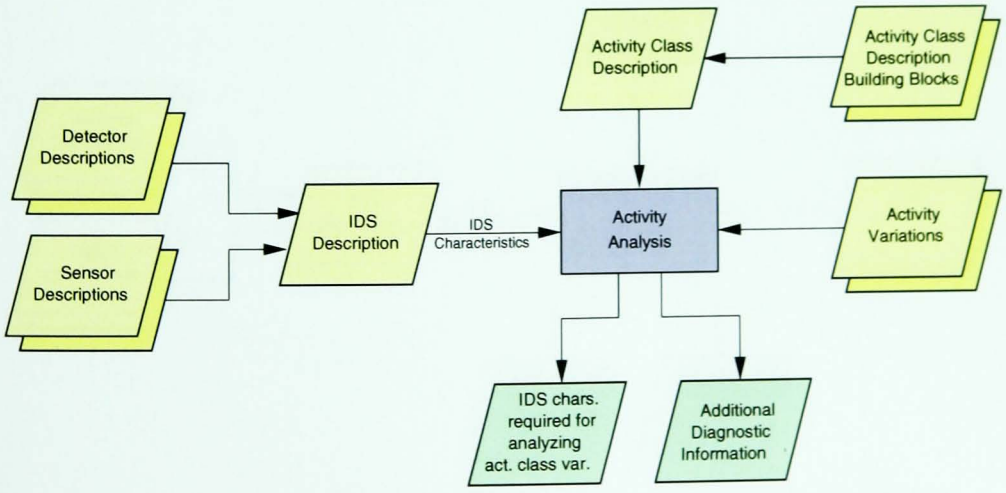


Figure 30—Input required for and output generated by the attack class analysis step

If the IDS considered is incapable of analyzing the attack class variant considered, the resulting set of IDS characteristics required for analyzing the attack class variant will lack the unavailable IDS characteristics. However, in general more than just the unavailable IDS characteristics will be missing in the result. Typically the IDS characteristics required for analyzing the attack class description building block or variation whose analysis failed, as well as the characteristics required for analyzing the omitted building blocks and variations will be missing. The analysis of further building blocks and variations is omitted if no alternative way of analyzing the attack class variant considered exists (see also Section 6.3.2). In the extreme case, in which the IDS considered is not even capable of observing the attack class variant, the result will be an empty set of IDS characteristics.

To facilitate debugging and further analysis, all attack class components involved and the fact whether the IDS was capable of analyzing them are recorded in addition to the IDS characteristics employed.

7.1.2 Alarm analysis

Once the examination of how an IDS analyzes a given attack class variant is completed, we continue with the second step of the IDS analysis process, which is also automated. Based on the IDS characteristics that were required to analyze the attack class variant selected, we determine the generalized alarms the IDS considered has the potential to generate. This is achieved by evaluating the attack class-independent *alarm conditions* with the set of IDS characteristics identified. The definition of the attack class variant and the identification of the generalized alarms that IDSs have the potential to generate are therefore *independent*. Note that for this to be true, the attack class descriptions should not include any indication on the generalized alarms the attack class described might be causing. This is achieved by not formulating any *a priori* expectations that might influence how attack class variants are being evaluated. This approach thus supports the identification of multiple generalized alarms that an IDS potentially generates for a single attack class variant.

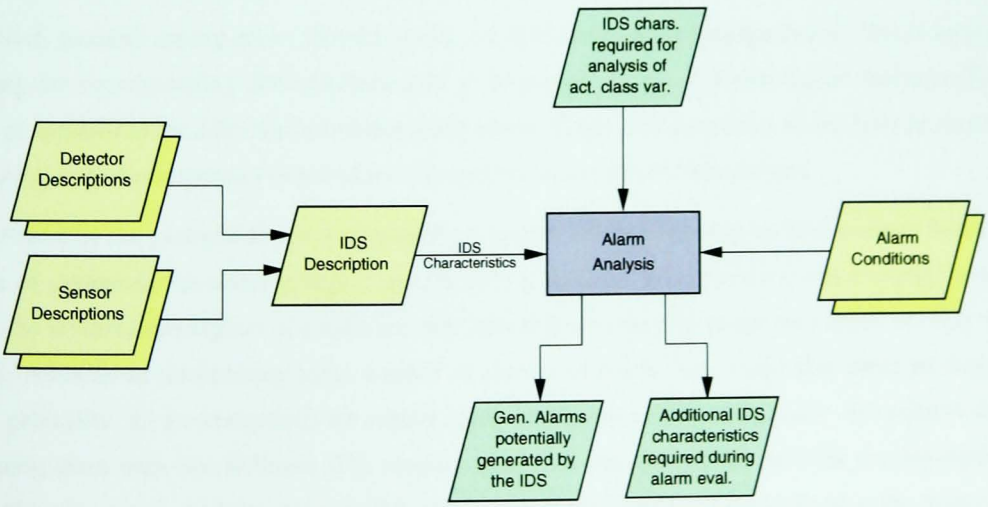


Figure 31—Input required for and output generated by the alarm analysis step

Figure 31 shows the input required for and the output produced by this step. As in the preceding step, the IDS descriptions are required. In addition, the alarm conditions and IDS characteristics that were required to analyze the attack class variant considered are required as input. On the output side, we find the set of generalized alarms that the IDS under considered was found to have the potential to generate. We also obtain the list of additional IDS characteristics that were required for the generation of the generalized alarms, in addition to some debugging information.

In the following we provide insights into the alarm conditions, explain how they were identified, and discuss the semantics of the alarms that they specify the conditions for.

7.1.2.1 Alarm conditions

It was mentioned that alarm conditions determine whether an IDS has the potential to generate a given generalized alarm based on the set of IDS characteristics that were required for the analysis of the attack class variant considered. However, when looking at the details, the situation is more complex. Beyond the IDS characteristics needed for analyzing the attack class variant, additional IDS characteristics may be required by the alarm condition. This may be necessary if one wishes to differentiate between knowledge- and behavior-based detection. In such cases it is important to differentiate clearly because the semantics of alarms may differ significantly.

Example: *Once more consider our http argument buffer overflow example. An IDS is found to have the potential of generating an alarm indicating a malformed http URL if the IDS characteristics that were required to analyze the attack class variant meet the requirements specified by the alarm condition. The latter requires, among other things, that the following IDS characteristics were required for the analysis of the attack class variant:*

- *Basic http arguments (see also Section 5.2.2)*
- *String matching for http information items (see also Section 5.3.3.2.2)*

In addition, possibly among other characteristics, the IDS has to be knowledge-based. This is verified by requiring the corresponding IDS characteristic to be present in the IDS description (see also Section 5.3.1), as opposed to the IDS characteristics listed above. These characteristics do not have to be present in the set of IDS characteristics required for the analysis of the attack class variant.

As illustrated by the example above, it is possible to specify the requirements an IDS needs to fulfill to be capable of generating the alarm at highly specific IDS scope, e.g., http. Such a practice would, however, lead to the repeated description of conditions that only differ in the IDS scope they apply to. This would not only result in an unnecessary large number of alarm conditions, but would also cause an important loss of generality. As a consequence we require alarm conditions to be more generic. We achieve this by associating them with two different IDS scopes—similar to the method we used for the description of attack class description building blocks. This means that alarm conditions are defined at the highest IDS scope level possible. Then, during the alarm analysis, we determine the effective IDS scope of the generalized alarm based on the IDS scopes of the IDS characteristics required for the analysis of the

attack class variant considered. The effective IDS scope of the generalized alarm may not be less specific than the IDS scope at which the alarm condition was specified. Technically, the identification of the effective IDS scope is achieved by taking advantage of Prolog's inference engine [Diaz00] (see also Appendix D.6.3).

Example: *Consider an alarm condition that could be triggered for our http argument buffer overflow example. Such an alarm condition might describe the fact that a suspicious argument was observed, and can be defined at the IDS scope "application layer." Then, during the evaluation of all alarm conditions, the alarm-analysis process would recognize the fact that the analyzed attack class variant has required http attributes and that http is an application layer protocol. If then the set of IDS characteristics used for the analysis of the attack class variant meets the conditions formulated by the alarm-condition example, we consider the alarm to have been generated. This can then be interpreted as the fact that the IDS analyzed has the potential of generating an alarm indicating the observation of a suspicious http argument (i.e., URL) for the attack class variant considered.*

Remember, the important property is that alarm conditions are independent of both attack class and IDS descriptions. However, how does one know which alarm conditions to create if they are independent of attack class and IDS descriptions? The 19 alarm conditions that we created in the context of the RIDAX prototype implementation were identified by searching for attack super-classes in the attack categorization described in Section 4.2. We focused our effort on attack classes and variations for which we actually created descriptions (see Section 4.3). These alarm conditions are of high generality as they are defined at a high-level IDS scope. In Appendix D we provide examples and a semi-formal specification.

7.1.2.2 Semantics of generalized alarms

When discussing alarm semantics one has to distinguish clearly between the generalized alarms generated in the course of our IDS analysis effort and alarms generated by IDS implementations. The alarms generated by IDS implementations denote the observation of a particular suspicious activity, whereas the generalized alarms generated in the context of our approach denote the potential of the IDSs analyzed to generate alarms indicating complete classes of suspicious activity. This means that the generalized alarms do not provide an indication of whether the signature database of a knowledge-based IDS actually contains signatures for specific attacks, but may provide us with information that is of great value for subsequent alarm-processing algorithms. These conditions may then be used to determine whether the potentially generated alarms carry any additional diagnostic information. In RIDAX we took advantage of this possibility by including a flag that indicates whether the IDS analyzed is capable of reporting the *success state* of the supposedly observed attack.

Example: *Referring to above example, and considering an IDS that "simply" checks for the presence of a string. Such an IDS is not capable of providing the information whether the supposedly identified attack was successful as long as the reaction of the attacked process is not taken into account. This typically is*

difficult for network-based IDSs but less so for host-based systems because for the latter it is generally easier to get hold of the necessary information.

This also means that in general the semantics of (seemingly) identical alarms that any two IDSs may generate differ. This is true for alarms generated by IDS implementations as well as for those generated in the context of our approach to IDS analysis. As a consequence we consider alarms generated by any two differing IDSs to be semantically different.

We fully take advantage of all the information that alarm conditions may provide us with by using the following 5-tuple of alarm properties to define and distinguish generalized alarms:

1. IDS identifier (e.g., "Snort, v1.7, light-weight configuration")
2. Generic alarm name (e.g., "suspicious argument string")
3. IDS scope of alarm definition (e.g., "application layer")
4. Effective IDS scope of alarm generated (e.g., "http")
5. Availability of attack success-state ("true" or "false")

The use of this 5-tuple also supports us in differentiating alarm classes based, for instance, on the detection method used by the IDSs. Differentiating generalized alarms that behavior- and knowledge-based IDSs generate is necessary because they differ significantly in their expressiveness and semantics. This difference is also reflected by the corresponding alarm conditions. Knowledge-based IDSs include a limited description of the attack identified in the alarm. They may also include additional diagnostic information such as IP addresses. Such alarms generally refer to identifiers such as CVE (see Section 2.2.5.1). In the case of behavior-based IDS the situation is different. These IDSs commonly only express the fact that suspicious activity was observed by signaling the fact that the system monitored deviates from its normal behavior. Their alarm identifier may for instance express the fact that a strange, abnormal sequence of instances was identified. However, such an identifier reveals no concrete information about the cause of the non-acceptable sequence.

Example: *Considering our http buffer overflow example. Assuming that IDSs such as Snort, WebIDS or DaemonWatcher are found to have the potential of detecting this attack, the corresponding alarms could look as follows:*

Table 28—Example of how various IDSs report a buffer overflow attack

Alarm property / IDS	Snort	WebIDS	DaemonWatcher
IDS identifier	Snort, v1.7, light-weight configuration	WebIDS	DaemonWatcher for httpd
Generic alarm name	Suspicious argument string	Suspicious argument string	Unknown execution path
IDS scope of alarm definition	Application layer	Application layer	Call
Effective IDS scope of generated alarm	http	http	System call
Availability of attack success state	False	True	True

As a possible application we will show in Chapter 8 how one can benefit by taking into account such semantic differences of alarms when combining alarms generated by diverse IDSs. However, note that there is no one-to-one mapping between generalized alarms and naming schemes for attacks or vulnerabilities, such as CVE. What we have instead is a many-to-many mapping between classes of attacks and real attacks. After all, one can view the generalized alarms defined by the above 5-tuple as an alarm classification scheme that could be applied to alarms generated by IDS implementations to simplify their further processing.

7.1.3 Rating of generalized alarms and attack classes

As an additional step, the RIDAX implementation performs a rating of the generalized alarms that the IDSs considered were found to have the potential of generating. This is done based on the expectable alarms for the attack class components involved (see also Section 6.4). These are used to verify whether every attack class component for which at least one expectable alarm was specified generated at least one of the expectable alarms. If this is not the case, the attack class component concerned is rated as "not detected."

The result of this step is three-fold. First, generalized alarms are rated to be true positives, false positives, or false negatives:

- True positives: The (generated) generalized alarm is listed in the list of expectable alarms of any of the attack class components involved in the attack class variant analyzed.
- False positives: The (generated) generalized alarm is *not* listed in the list of expectable alarms of any of the attack class components involved in the attack class variant analyzed.
- False negatives: An attack class component with a nonempty list of expectable alarms exists that was not reported by any of the expectable alarms.

Concurrently also the attack class components are rated:

- Benign: There are no expectable alarms specified for the attack class component (the component might in fact be benign).
- Detected: At least one of the expectable alarms was generated.
- Not detected: None of the expectable alarms was generated.

Based on these ratings for attack class components we then rate the complete attack class variant as explained in Section 6.4 (see Table 27, p. 117). Note that independently of how an attack class variant is rated, false positives may be present. In fact, false positives may obfuscate the result such that it becomes even more challenging to draw precise conclusions based only on the set of alarms generated for attack class variants.

Example: *Consider again the http argument buffer overflow attack (see also the example provided in Section 6.4). Moreover assume that the alarm analysis reveals that the IDS has the potential of reporting the attack class description building block describing the actual argument buffer overflow by means of an*

alarm. We would rate this building block as "detected" if the alarm indicates the observation of a suspicious string in the request. The same is true for any other alarm, such as an alarm reporting the observation of a suspicious execution path, that is "expectable" for this attack class description building block. Accordingly any alarm that appears on the list of expectable alarms is rated as a true positive. Assuming that the buffer overflow building block is the only malicious one referenced in the attack class description of the attack, we would consequently also rate the entire attack as "detected." If, however, the same attack was reported only by a non-expectable alarm, we would rate the attack as "not detected," and the alarm would be rated as a false positive, i.e., the attack was detected but incorrectly diagnosed.

Note that variations have a different impact on the rating of an attack class variant than other attack class components. We rate the alarms generated or expected because of variations in the same way as any other attack class component. Also, we rate attack class components describing variations as "detected," "not detected," or "benign." However, when rating attack class variants, we do not take the rating of variations into account because variations do not impact the core of an attack class but merely alter its appearance.

Example: *It would not seem reasonable to rate an otherwise correctly reported attack variant as only "partially detected" merely because the IDS analyzed did not report the fact that IP PDUs carrying the attack were fragmented.*

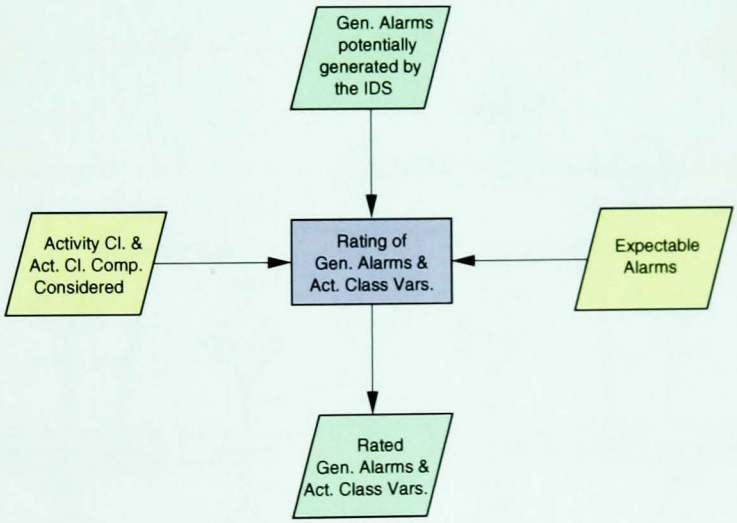


Figure 32—Input required for and output generated by the alarm and attack class rating step

7.2 Implementation: RIDAX, a tool for analyzing IDSs

We have now developed all the concepts that we require for analyzing IDSs at a conceptual level. In this section we provide an overview of RIDAX, which represents a prototype that implements all of these concepts.

As explained, we chose to describe attack classes, attack class components, and alarm conditions in a rule-based fashion because it seemed especially well suited for identifying generalized alarms that an IDS potentially generates (see Section 7.1). This solution is appropriate for creating and combining the generic and high-level descriptions of attack class components introduced in Chapter 6. For the implementation of RIDAX we chose the GNU implementation of *Prolog* by Diaz [Diaz00], as it is a rule-based language that meets our requirements, and in particular includes database connectivity. The database connectivity is implemented by integrating Prolog with the MySQL database [MySql] using their respective C interfaces. MySQL is already used to store IDS descriptions (see Section 5.4.1).

In the following subsection we describe the database structure used to store the analysis results. Subsequently we describe the most important phases of the analysis performed by RIDAX.

7.2.1 Database structure

During the analysis process RIDAX stores all the results generated into a database for later analysis. This database has close relations to the database used to store IDS descriptions (see Section 5.4.1). Figure 33 shows a simplified entity relationship diagram of this database, in which all the entities taken from the IDS description database are shaded. These entities are not specific to the analysis result database but solely illustrate the relations to the database developed in Section 5.4.1.

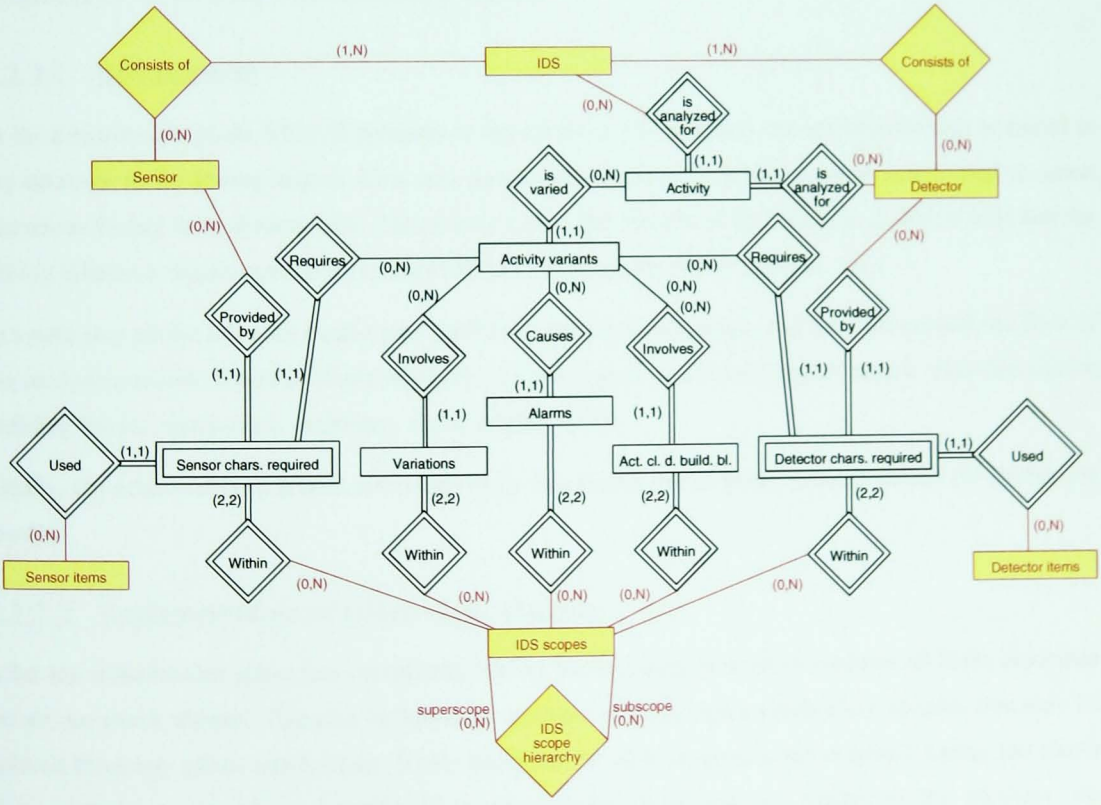


Figure 33—Entity relationship diagram of the database used to store analysis results

The most important entity is the *attack class variants* entity. This entity links all the results that have been collected during the analysis of an *attack class* for a given combination of *variations*. The *attack class variants* entity has relations to numerous entities representing the data collected during the analysis process. Most importantly it links generalized alarms, including false negatives, and attack class variants. In addition, four further entities are linked that are mainly used for development purposes. However, they may also be used to determine the additional diagnostic information an IDS might be providing along with the alarms it generates. These entities represent the IDS characteristics that were required for the analysis of attack class variants. Based on this data, it becomes possible to determine the additional information that an IDS provides along with an alarm. Furthermore, the *variations* and the *attack class description building blocks* entities are used to obtain some insight into the analysis process. Last but not least, the *attack class* entity links the analysis results of the attack classes (including attack class variants) with the IDS that has been analyzed. It thus enables us to obtain an overview of the analysis results on a per-attack-class basis across the various attack class variants analyzed.

7.2.2 Analysis steps

In the following we provide an overview of the IDS analysis steps as they are implemented in Prolog—without going into details of the Prolog code. However, note that much of RIDAX relies on the integration of GNU Prolog with the MySQL database.

7.2.2.1 Initialization

In the initialization phase RIDAX connects to the database and transfers the information that is stored in the database to the Prolog engine. This data primarily consists of IDS descriptions and is used to assert numerous Prolog facts dynamically. These facts reflect the content of the database in such a way that the Prolog inference engine can use it (i.e., the IDS descriptions, the IDS scope tree, etc.).

In a next step all the rules are loaded into the Prolog environment. Some of these rules control the flow of the analysis process, whereas others represent the descriptions of attack classes, attack class description building blocks, attack class variations, alarm conditions etc.

Finally, the actual analysis phases are prepared by initializing the database tables required for storing the results.

7.2.2.2 Implementation of attack class analysis

After the initialization phase has completed, Prolog backtracking is used to evaluate all IDSs described for all the attack classes. This also includes the analysis of all possible attack class variants that may be derived from any given attack class. While doing so, all IDS characteristics required during the attack class analysis are stored as dynamically asserted Prolog facts and are written to the database. As explained in Section 6.3, variations may suppress sensor items or may render the analysis capabilities of detectors useless for further analysis; this is also taken into account by asserting dynamic facts.

7.2.2.3 Implementation of alarm analysis and rating

After the Prolog rule representing an attack class description has reached its goal, all the alarm descriptions represented by Prolog rules need to be evaluated as well. This is where the advantages of the Prolog inference engine become very helpful. The alarm conditions are evaluated based on all the facts that were asserted dynamically during the attack class analysis phase. Any alarm rule that succeeds represents the creation of a generalized alarm, which is rated whether it is a false or a true positive before it is stored in the database. Concurrently all the missing generalized alarms are identified, which are then stored as false negatives into the database.

After all the alarm rules have been evaluated, all dynamically asserted facts are retracted, i.e., discarded, and the process continues by evaluating the next attack class variant. This process is repeated for every attack class variant. In other words, every attack class is analyzed several times, once for every allowable combination of variations.

7.3 Validation

In this and the preceding chapters we have gradually worked towards a solution of what was claimed in Section 1.3. There it was claimed that an approach exists that is able to predict the classes of attacks that a given IDS design is able to detect. Moreover it was claimed that such an approach can be implemented based on the combined analysis of IDS descriptions and descriptions of attack classes—thereby avoiding the need of conducting experiments with specific attacks and actual IDS implementations.

In Section 3.5 we highlighted the difficulties involved in validating our approach to IDS analysis and outlined the process that allows us to provide evidence that the results produced are correct. However, before we present the necessary steps to provide this evidence, we discuss the challenges involved in validating the IDS analysis results.

7.3.1 Validation challenges

One of the requirements for our approach was that the necessity to conduct experiments with actual IDS implementations has to be avoided. The reasons for this requirement were two-fold. On the one hand our approach should assist IDS designers early in the design phase of IDSs in order to improve the efficiency of the design process, i.e., before the IDS design is implemented. On the other hand the approach aims at providing results of high generality involving only a limited effort. In other words, the results have to be provided at the level of attack classes. We chose a description-based approach because this enables us to meet the above requirements and to limit the effort involved.

We face a number of issues when it comes to validating whether the predictions made by our approach to IDS analysis are correct and correspond to the actual behavior of IDS implementations. A systematic and complete validation would require that the class-level predictions made by our approach are compared with the behavior of actual IDS implementations. Such an undertaking would represent an enormous

challenge and in fact precisely exemplifies the problem this work attempts to address. It would be required that one or several rather complex environments are built such that IDSs can be analyzed under different conditions. The complexity of such environments would be comparable to the one used in the Lincoln Lab experiment or might even exceed its complexity (see also Section 2.3.3.1). However, the most challenging aspect of any such validation undertaking would be the number and diversity of individual tests to be executed. Complete validation would require the pursuing of an equivalence class testing strategy as discussed in Section 2.2.2. Applied to our approach for each attack class, including those derived automatically by applying variations to already known classes (see Section 6.3), several individual tests would have to be executed for each IDS analyzed. As outlined in Section 3.2.2, a comparatively simple setup using 27 attack class descriptions to which variations are applied may yield 498 or even more distinct attack classes. The validation of the analysis results that our approach produces for these attack classes would therefore involve the execution of several thousands of tests for each IDS considered because the analysis results of each attack class need to be validated by multiple tests. Such an undertaking would significantly exceed the scope of this work and would also exceed the Lincoln Lab effort. In the following we will therefore outline an alternative solution that is able to provide evidence that the results produced by our approach are correct.

Another issue that needs to be taken into account is the fact that under certain circumstances the results predicted by our approach to IDS analysis may not correspond to the actual behavior of IDS implementations. There exist numerous conceivable reasons for such inconsistencies:

- Incorrect or insufficient IDS description: The description of an IDS does not reflect the characteristics of the IDS correctly. This may, for instance, mean that the IDS description does not describe the particular configuration of the IDS considered correctly. Moreover it might be the case that the IDS uses new techniques that are not yet covered by the IDS description scheme. In this case the IDS description scheme would have to be extended accordingly.
- Incomplete attack signature set: If the set of attack signatures of a knowledge-based IDS does not include the signature for a given attack, the IDS will not be able to detect the attack even though it might in theory be *capable* of detecting it, i.e., even though it offers all the analysis techniques required. In this case the inconsistency is caused by the fact that our approach operates at the level of attack classes and therefore does not take into account signatures for specific attacks.
- Incorrect attack class description: If the description of an attack class is incorrect, e.g., incomplete, the IDS analysis results are likely to be incorrect.
- Ad-hoc techniques and heuristics: If IDS implementations use ad-hoc techniques or heuristics that are tailored to the detection of a specific attack or to the avoidance of specific false positives, their actual behavior may differ from the prediction made by our approach. Consider, for instance, the example in the introduction of Chapter 1 (see also Section 2.4.2). There an IDS is described that will generate alarms for observed IP fragments whenever the fragments are smaller than a given, pre-configured size. The accuracy of such techniques is generally very

limited [Marty02, p. 66]. The difficulty with regard to IDS analysis is that such techniques may surpass the granularity at which our IDS description scheme operates.

- IDS implementation flaws: IDS implementations may suffer from implementation flaws, i.e., bugs, that let their characteristics divert from the design.
- Failure of our approach to IDS analysis: If none of above reasons could be identified as the cause for an inconsistency between the results that our approach produces and the manner an IDS behaves, then this would indicate a weakness in our approach. However, so far we were not able to prove that our approach always produces correct results by means of a complete validation for the reasons indicated above, but neither were we able to prove the contrary by encountering counter-examples among the numerous experiments that were conducted in the context of this work.

However, despite these difficulties, we found our approach to produce correct and, most importantly, useful results. This fact is emphasized in the next section where meaningful evidence that our approach to IDS analysis provides correct results is provided.

7.3.2 Validation procedure

Above we explained the difficulties of complete validation using equivalence class testing. As an alternative we present a process in the following that is suitable for providing evidence that the IDS analysis results produced by our approach are correct.

As a first step we discuss the validity of the IDS description framework and the attack classification and description scheme. They represent the input to our IDS analysis approach (see Figure 1).

The IDS description framework represents an empirically developed scheme that describes IDSs by systematically identifying system characteristics that are relevant to the detection of attacks. The scheme has been developed based on a simplified CIDF IDS model (see Figure 8) and combines concepts from IDS taxonomies such as the one by Debar *et al.* [DeDaWe00], insights gained in the context of the VulDa work [DacAle99], and experiences made while using and developing IDSs. The scheme rigorously separates analytic functions, i.e., techniques, provided by IDSs from the domain to which these functions can be applied. The resulting scheme is extensible and enables the unambiguous and concise description of IDSs and IDS designs with respect to their attack detection capabilities.

The attack classification and description scheme uses the notion of IDS characteristics as defined by the IDS description framework for expressing the requirements IDSs have to meet in order to be able to detect a given class of attacks. The proposed attack classification scheme meets all the requirements (see Section 2.2.1) that a sound classification has to meet:

- Orthogonality: For any attack there exists one description only. Thus any described attack can either be associated with an already identified class or identifies a new attack class.

- Procedure: The classification procedure for attacks was defined in Section 3.2. It consists of a first step, in which a description of the considered attack is created, and a second step, in which the resulting description is compared with other already identified descriptions.
- Observability: The description of attacks is based on clearly defined IDS characteristics that are used to describe the observable aspects of attacks. The resulting attack descriptions can be compared with each other.
- Hierarchy: It is possible to create higher-level descriptions of attack classes by making use of the hierarchical nature of IDS scopes. However, this possibility was never used in this work.
- Consistency: Each attack that belongs to a given class is described by an attack description that is identical to the description of all attacks that belong the same class, i.e., attacks that belong to the same attack class share the same description as a common property.

Assuming that the IDS description framework and the attack classification are viable, we can now investigate the validity of the results provided by actual IDS analysis procedure. Here it needs to be shown that the predictions made for IDS designs correctly reflect the actual behavior of the corresponding IDS implementations. A complete validation of the correctness of the predictions made could be achieved by means of equivalence class testing (see Section 2.2.2). As explained in Section 7.3.1, in this particular case such an approach involves significant challenges that are difficult to address. It is, however, possible to provide evidence that the results provided are correct. First, a diverse set of attack classes, attacks that belong to these classes, and a set of existing IDS implementations have to be selected. Then it has to be shown that the IDS implementations report the selected attacks by means of alarms that correspond to the generalized alarms that were predicted by the combined analysis of the designs of the IDSs selected and the description of the attack classes considered (see also Figure 2). This last step requires the manual comparison of alarms generated by IDS implementations and generalized alarms predicted by our IDS analysis approach, because the alarms generated by IDS implementations do not explicitly convey the information conveyed by generalized alarms.

7.3.3 RIDAX example

The following discussion is based on the example that we introduced in Section 3.5. There we identified the alarms that the IDS implementations WebIDS and Snort generate for two specific http attacks. These results were then compared with the analysis results produced by RIDAX. In the following we provide some further background information related to this example and in particular to the information provided in Table 2.

In their second column Table 29 and Table 30 provide the raw alarm messages as they are generated by WebIDS and Snort for the attacks indicated in the first column. In the third column we provide the predictions as they were made by RIDAX for the attack classes to which the attacks considered belong. The obfuscated variants of the attacks considered are identical to the ones discussed in Section 3.5 and represent the use of IP fragmentation.

We can verify whether the alarm messages and the predicted generalized alarms correspond to each other by taking a closer look at the manner these two IDSs detect these attacks. WebIDS (see Table 29) detects the test-cgi attack by applying a pattern-matching algorithm to the CLF [Weinma98] log file entries written by the webserver software. Among other information CLF log files provide the URI of the requests that the server receives. Given the fact that the test-cgi attack involves a specific URI and that our IDS description scheme represents URIs as a "request argument" of the IDS scope "http," the prediction made by RIDAX is correct. For the second, http-header-based attack, RIDAX has correctly predicted that WebIDS is not capable of the detecting the attack because http header data is not written to CLF log files, i.e., the attack is never visible to WebIDS. Concerning the attack obfuscation technique considered, WebIDS proved to be immune against IP fragmentation tactics. This is not surprising because WebIDS is not analyzing network data.

Table 29—Alarms generated by and generalized alarms predicted for WebIDS

WebIDS	Syslog alarm message	Generalized alarm
http meta-character attack[16] [CA0696, CVE007099]	`May 1 16:54:45 loghost webids[14080]: 10239028_7 0x3e3d43d3 pattern(cgi) 10.4.2.116 /test-cgi 200 10.4.2.116 - - [01/May/2002:16:54:45 +0200] "GET /cgi-bin/test-cgi?/* HTTP/1.0" 200 - CVE-1999-0070 CVE`	• Alarm type: suspicious argument string • IDS scope of generalized alarm: http • IDS scope of alarm definition: Application layer
Obfuscated http meta-character attack	Detected; same as above	Detected; same as above
http header buffer-overflow[17] [CVE084800]	Not detected	Not detected
Obfuscated http header buffer-overflow	Not detected	Not detected

In the case of Snort the results are slightly different (see Table 30). First of all we consider a simple configuration of Snort that does not support the re-assembly of fragmented IP traffic. It is therefore no surprise that the Snort implementation does not detect the obfuscated attacks. The RIDAX analysis results predict this behavior correctly by generating no generalized alarms. During the RIDAX analysis this attack class variation imposes the requirement that the IDS be able to deal with IP fragments, i.e., instance part groups of the IDS scope IP, which the considered IDS configuration is not capable of doing. When it comes to the detection of the two non-obfuscated attacks Snort performs better than WebIDS does. It correctly detects both attacks as predicted by RIDAX. The detection techniques used by Snort are similar to the ones used by WebIDS. It is therefore no surprise that a test-cgi attack is reported by an alarm that corresponds to the same generalized alarm as was predicted for WebIDS. Considering the second attack, the data source used represents an important differentiator. As seen above, the fact that Snort operates based on network data makes it susceptible to certain obfuscation techniques, but provides the eminent advantage that it can observe the complete communication between the client and the (web-)server. It can

[16] A directory listing can for instance be retrieved by launching the following on a unix command line:

```
echo "GET /cgi-bin/test-cgi?/*" | nc vulnerable.server.com 80
```

therefore detect suspicious strings in http header data that do appear in CLF log files. Thus the RIDAX prediction for the http-header-based attack is correct because our approach describes http header data as "request options" of the IDS scope "http."

Table 30— Alarms generated by and generalized alarms predicted for Snort

Snort	Syslog alarm message	Generalized alarm
http meta-character attack [CA0696, CVE007099]	`May 1 11:06:54 snorthost snort: [1:835:5] WEB-CGI test-cgi access [Classification: Attempted Information Leak] [Priority: 2]: <eth0> {TCP} 10.4.2.116:2631 -> 10.4.2.111:80`	• Alarm type: suspicious argument string • IDS scope of generalized alarm: http • IDS scope of alarm definition: Application layer
Obfuscated http meta-character attack	Not detected	Not detected
http header buffer-overflow [CVE084800]	`May 1 11:10:56 snorthost snort: [1:9999`[18]`:1] WEB-MISC Host-header overflow [Classification: Attempted Denial of Service] [Priority: 2]: <eth0> {TCP} 10.4.2.116:2646 -> 10.4.2.111:80`	• Alarm type: suspicious options string • IDS scope of generalized alarm: http • IDS scope of alarm definition: Application layer
Obfuscated http header buffer-overflow	Not detected	Not detected

These examples, although comparatively simple, provide us with some evidence that the predictions made by RIDAX are correct. Moreover they illustrate the process that would need to be replicated for a large number of individual attacks in order to provide better evidence or even complete validation.

7.4 Discussion

In this chapter we have shown how IDS and attack class descriptions can be combined to predict detection capabilities of IDS designs. Our considerations also included the RIDAX prototype that actually implements the proposed approach to IDS analysis and an outline of how the results produced by RIDAX could be validated. By presenting this outline we provided some evidence that the predictions made by RIDAX are correct. This evidence could be improved by providing further examples. However, complete validation of whether the RIDAX predictions are correct represents a major challenge and would require thousands of further experiments. An undertaking of this kind would need to be addressed by using tools such as LARIAT [HRLC01, RCFRLH01] or Thor [Marty02] (see also Section 2.3.3). Such tools facilitate the automated analysis of IDS implementations for larger numbers of attacks, and in the case of Thor even provide the possibility to do this for systematically varied, i.e., obfuscated, attacks. However, it should be noted that the effort required would exceed the possibilities such tools currently provide.

---

[17] This attack can be demonstrated using the following unix command line:

```
perl -e 'print "GET /servlet http/1.0\nHost: " . "x" x 1092 . "\n"' | \
nc target.server.com 80
```

[18] We had to create our own signature for this attack because an appropriate signature is not provided by snort's default signature set.

In Chapter 8 we will provide further RIDAX prediction examples and will moreover show how our approach to IDS analysis can be extended towards the prediction of false alarms.

7.5 Conclusion

In this work we present an approach that performs a combined analysis of IDS descriptions and descriptions of attack classes in order to predict detection capabilities of IDS designs. In this chapter we have developed the method required to perform this combined analysis and presented the RIDAX prototype that we have implemented. Validation of the predictions made is a challenging task, but the considerations made in this work so far and, in particular, while outlining such a validation undertaking in this chapter make us confident that the results provided by RIDAX are correct. Our IDS analysis method can therefore be viewed as a suitable utility for IDS designers because it operates at a conceptual level, i.e., it does not involve actual attacks or IDS implementations. A tool such as RIDAX can therefore assist designers early in the design phase of IDSs by predicting where the design meets the specification and where it fails to do so.

Chapter 8 A further application: The assessment of IDSs and combinations thereof

In the preceding chapters we have developed schemes for describing IDSs and attack classes, and have presented an approach to IDS analysis that operates based on these descriptions. Our main goal is to provide guidance to IDS designers by predicting the classes of attacks a given IDS design has the potential of detecting. However, the insights that our approach to IDS analysis provides can also be used for other applications. In this chapter we will show how the RIDAX prototype can be extended to support the assessment of IDSs and arbitrary combinations thereof. The combination of IDSs is an important aspect of *ID architectures*. These may consist of diverse IDSs, i.e., IDSs that use different information sources and analysis techniques, that are being operated at different locations in the network. As a first step we will define assessment criteria and metrics according to which IDSs shall be assessed. Before developing a method for assessing IDS combinations, we show how existing assessment metrics, as developed in the information retrieval field and already used by IDS benchmarking approaches, can be used to assess individual IDSs. In the second step we propose an alarm-processing method that combines the analysis results of multiple IDSs in a manner that takes into account the semantics of the generalized alarms. From the results obtained with this method, we develop metrics that enable the assessment of arbitrary IDS combinations in terms of the completeness and utility of the information they have the potential of providing.

The goal of these metrics is to measure how an individual IDS or a combination of IDSs covers a given set of attacks, i.e., attack classes. Moreover they measure the quality (utility) of the coverage provided. This is achieved by making use of an information-theoretic approach to analyze the alarms sets that IDSs generate for given activity classes. This approach determines the suitability of the information provided by IDSs for fault diagnosis purposes, such as discriminating between true and false positives, or identifying the cause of a set of alarms. Once these metrics have been defined, we discuss the manner in which RIDAX had to be extended to calculate measurements according to these metrics.

Note that in order to achieve this, we will make use of the possibility of generalizing the analysis performed by our approach from classes of attacks to classes of activities (see Section 3.6.2). The following discussion will therefore be based on classes of activities and their description components rather than on attack classes as was the case in the preceding chapters. Hence, when applicable, we will use terms such as *activity class* and *activity class variant* instead of terms such as attack class or attack class variant. This is necessary because the metrics envisaged also assess the potential of IDSs to generate false positives. This requires the analysis of IDSs with regard to the manner in which they analyze benign activities that are similar to attacks (see Sections 3.6.2 and 4.3). Finally we will provide results that were obtained by assessing five IDS example configurations and their combinations. Note that the results provided merely serve illustrative purposes. The assessment method presented here operates based on descriptions of IDS designs and classes of attacks, and does not take into account characteristics specific

to the environment such as the type and amount of network traffic or the network topology. We assume a *normalized* environment in which every activity class variant occurs exactly once.

8.1 Related work

In the following we will propose a method for combining the information provided by diverse IDSs and define metrics that assess the information provided by the resulting combined system. However, before doing so, we shall discuss some related aspects. First we will consider the manner in which system reliability can be improved using concepts known from the dependability domain, and discuss the limitations of such an approach. Then we will present and discuss assessment metrics as they are used in the ID and information retrieval domains.

8.1.1 Dealing with false positives and negatives in the dependability context

In MAFTIA D2 [D2Maf01] it is shown how error detection and fault-diagnosis mechanisms are to be viewed in the context of intrusion detection. Unfortunately, IDSs may fail in various ways. Ideally, one should try to apply dependability concepts such as fault masking to handle such IDS failures. However, as also indicated in MAFTIA D3 [D3Maf01], it is not straightforward to apply these concepts to ID.

Whenever one deals with system components that are not sufficiently reliable to meet a given system's specification, a standard approach is to introduce redundant components into the system. Applied to ID, IDSs represent the components, and the system is the ID architecture. By letting the redundant components vote on the result, it is possible to increase the system's reliability. This has been explored extensively in the dependability field. However, it seems impossible to apply this concept to ID—one, but not the only, problem being that the failure rate of IDSs, e.g., the rate of false alarm, is too high [Axelss00, Julisc00, MCZH99, Schnei00]. This problem is illustrated in the work of Mathur *et al.* [MatAvi70]. They prove that one cannot increase a system's reliability simply by adding redundant components, e.g., in an N-modular redundancy (NMR) scheme, if the failure rate of the individual components exceeds 50%. Unfortunately, many IDSs generate much more than 50% false alarms.

In addition, the semantics of the alarms generated depends significantly on the capabilities of the IDS generating them. This influences the information and the trustworthiness of the information carried by these alarms, which in itself significantly increases the complexity of error detection and fault diagnosis when the alarms generated by multiple IDSs are being combined.

8.1.2 Assessment metrics

Whenever one seeks metrics for assessing systems, it is crucial that one first defines goals of the assessment. In some cases the choice of the assessment criteria seems to suggest itself. However, in most cases, several criteria are conceivable. Moreover their importance may vary significantly depending on

the usage envisaged of the systems—a fact that naturally also applies to the assessment of IDSs [Wilkis02].

8.1.2.1 Assessment metrics used in ID

The evaluation of IDSs with regard to the attacks they detect is a first, vital step in the context of IDS assessment. However, so far the interpretation of these evaluation results is an issue that has been addressed only to a limited degree.

Currently the alarms generated during the evaluation of IDSs are rated either as true or false positives. Moreover, false negatives, i.e., when an IDS fails to generate an alarm for an attack, are also recorded. Based on this information, one commonly defines metrics that measure the percentage of attacks detected or the rate at which false alarms are generated. In the Lincoln Lab evaluation, for instance, the false positive rate was measured in false alarms per day.

Often the results obtained are represented using receiver operating characteristics (ROC) curves. ROC curves have been already in use for many decades in the domain of signal processing. There they are used to judge the quality of a given receiver and to choose an operating point for it [PeBiFo54]. For a given receiver a ROC curve superimposes the probabilities for true and false positives for the value range of a selected receiver parameter. For a good receiver the curve will come very close to the point *(0,1)*, i.e., the point where each signal is detected correctly and no false positives are generated (see curve a) in Figure 34. A poor receiver will produce a curve that is close to the diagonal (see curve b) in Figure 34.

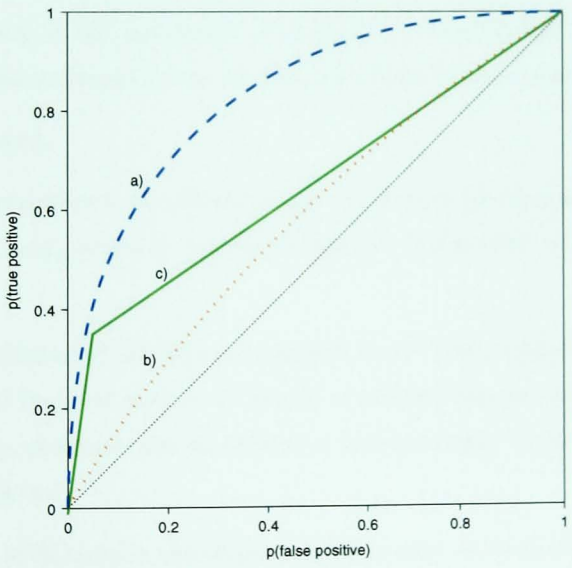


Figure 34—Example of ROC curves

ROC curves have been used to illustrate the IDS assessment results by Lippmann [LFGHKM00], Maxion and Tan [MaxTan00] and others. However, their application to ID is not without difficulties [McHugh00b] as illustrated by the following examples:

- The probability for a false alarm may be biased significantly by the environment in which the IDS has been tested—a fact that may influence the curve significantly.
- The number of available samples may be as low as one if there is no IDS parameter available that can be set to multiple values (this is a property inherent to our IDS description approach; see curve c) in Figure 34). As a consequence the resulting curves have to be interpreted with care.

Moreover, note that measuring and plotting the percentage of true and false positives does not take into account the *non-binary* nature of IDS alarms. Besides being a true or a false positive alarms convey information (implicitly and explicitly) that is rarely measured by IDS evaluation efforts. Also, the metrics used do not measure how well true and false positives can be discriminated or how well causes of alarms can be identified (see also Section 8.1.1).

8.1.2.2 Information retrieval

In our search for more generic metrics to assess IDSs, we have also looked at the field of information retrieval (IR), where such concepts have been investigated over the past years. Here people have defined the concepts of *precision* and *recall* [BaCoBe94, Lager96, Weiss97]. In his glossary [Weiss97] Weiss defines them as follows:

- *Recall*: Recall is the percentage of total relevant documents retrieved from all documents. Recall refers to how much information is retrieved by the search. Total recall would locate every document that matched the search criteria in a database.
- *Precision*: Precision is the percentage of documents retrieved that the searcher is actually interested in. Precision focuses on the relevant, most useful items retrieved in the search.

Weiss [Weiss97] further states:

> *Recall with high precision is the ultimate goal. The goal of information retrieval scientists is to provide the most precise or relevant documents in the midst of the recalled search results.*

If we now assume the distribution of activities to be known, *recall*, when applied to ID, can be considered as being the percentage of the total number of attacks considered that are reported by means of true positives. In a similar way, *precision* can be defined as the percentage of the total number of alarms generated that are true positives.

However, also in the case of IR metrics, one ought to exercise care. As recognized in Sections 8.1.1 and 8.1.2.1, a given attack may be reported by semantically diverse alarms depending on the IDS. Using the IR concepts one can only classify alarms into true and false positives and identify missed attacks as false negatives. Also one thereby does not take into account the utility of alarms for correctly distinguishing between true and false positives and identifying attacks.

8.1.3 Discussion

Current IDS assessment approaches only distinguish between true positives, false positives, and false negatives. As indicated, these metrics can be defined well using the notion of precision and recall, as described in Section 8.1.2.2. We will apply this concept in Section 8.2, where we define evaluation metrics for individual IDSs that do not assess the utility of the information provided by the alarms the IDSs generate. However, as with any metric that is based on the accounting of true and false positives, these metrics also have weaknesses. ID alarms are not of a binary nature, i.e., one can distinguish far more than just between correct and incorrect alarms. ID alarms implicitly and explicitly contain additional diagnostic information that is not assessed by any such metric. The diagnostic information may be key to further analysis as performed by alarm correlators within an ID architecture that combines multiple IDSs or the analysis performed by a human. Especially when one aims at identifying or diagnosing the activity that caused several IDSs to generate a set of alarms, the fact whether every single alarm is a false or a true positives becomes less important. When one analyzes alarms at this level, it is far more important to take into account the meaning of alarms as it is defined by alarm identifiers, etc., and as it is implicitly defined by the type of the generating IDS.

ROC curves are a popular means to represent IDS assessment results. However, we chose not to use them in this work because we would obtain curves defined by only a single point (see curve c) in Figure 34). Such curves are too imprecise for meaningful interpretation. Instead we chose to use two-dimensional plots and histograms to represent the results of our experiments.

8.2 Detection rate of IDSs

In this section we develop simple metrics that allow us to judge IDSs in a manner similar to existing approaches. We have extended RIDAX to automatically calculate these measurements on a per-IDS basis as part of the IDS analysis process. We measure the total number of true positives, false positives, and false negatives in a manner that we derived from the concepts of *precision* and *recall*, as used in the information retrieval field (see also Section 8.1.2.2). However, in doing so, there are some issues that we need to be aware of:

1. The semantics of generalized alarms generated differs significantly from that generated by IDS implementations. First, generalized alarms denote the potential of an IDS to generate a given class of alarms. Second, our approach may report, i.e., describe, activities using multiple generalized alarms. This can be caused by the combination of multiple activity, i.e., attack, class description components, but also by single components. It is clear that IDS implementations may also generate multiple alarms when reporting a single attack. Therefore comparing absolute alarm numbers may result in misleading measurements.
2. The number of false negatives determined by our approach cannot be compared with the number of false negatives determined while evaluating IDS implementations. This is due to the fact that we rate every generalized alarm that was expectable (see Section 6.4) but was not generated as a

false negative. However, as explained in Section 7.1.3, these false negatives only matter if no true positive reports the activity component in question. When judging *coverage* provided by an IDS, we focus on whether a malicious activity variant was rated "detected" or not (see Section 7.1.3), i.e., we define *coverage* as the percentage of all considered malicious activity variants that are detected.

3. The utility of the information carried by alarms generated is not measured.
4. The measurements are not made in a real environment, and encompass only one instance of every activity variant. However, as explained in the preceding section, other approaches suffer from this environment-related issue to an equal or greater extent. In spite of this, the results will provide a normalized assessment of the IDSs evaluated.

Most of these issues have already been mentioned in Section 2.4 (see Table 1) and in part provide the motivation for the considerations made in the next section. There we propose an assessment method that enables us to include the utility of the information provided by IDS alarms.

Being aware of above issues, we define the following metrics derived from the concepts of *precision* and *recall*:

- Recall r: the percentage of the total number of considered malicious activity class variants that was detected m_d. The total number of malicious activity class variants is determined by summarizing the number of detected malicious activity class variants m_d, the number of partially detected malicious activity class variants m_{pd} and the number of nondetected malicious activity class variants m_{nd}:

$$r = \frac{m_d}{m_d + m_{pd} + m_{nd}}.$$

  This measurement is normalized, and thus recall is equivalent to *coverage*. Applied to ID, coverage reflects the percentage of all considered attack classes that an IDS is capable of detecting, without taking into account the frequency of the occurrence of individual attacks.

- Precision p: the percentage of the total number of alarms generated that are true positives a_{tp}. The total number of alarms is composed of a_{tp} and the number of false positives a_{fp}:

$$p = \frac{a_{tp}}{a_{tp} + a_{fp}}.$$

  This metric enables us to assess the credibility of the alarms generated by an IDS.

Note that in above definitions we used both rated alarms (a_{tp} and a_{fp}) as well as rated activity class variants (m_d etc.). We chose to do so because the resulting definitions of precision and recall reflect the relevant information well. Moreover, the resulting definitions are to some extent comparable with the measurements determined in other approaches such as the Lincoln Lab experiment (see Section 2.3.3.1).

It would have been misleading to use the number of false negatives for the definition of recall, for instance, because the semantics of the false negatives as used in the context of our approach differs considerably from that used in other works.

In Section 8.6.2 we will provide concrete examples of precision and recall that were measured for a series of IDSs using the extended RIDAX prototype.

8.3 Fault diagnosis based on alarms generated by multiple IDSs

In the preceding section we defined metrics similar to the ones determined by other approaches. As explained, the resulting metrics neither take into account nor measure the utility of the information provided by alarms; they are not suitable for assessing combinations of individual IDSs. The latter, however, is an important foundations for the design of ID architectures. Bottom-up approaches that apply voting mechanisms such as the N-modular redundancy (NMR) scheme [MatAvi70] are not suitable for developing and building such supposedly highly complex architectures (see also Section 8.1.1). The realization of such an approach would be difficult because the semantics of all these alarms differs significantly depending on the type, configuration, and location of the IDS used. In addition given the fact that IDSs tend to generate far more than 50% false positives, an NMR scheme is likely to make matters even worse instead of improving them.

As a consequence we propose a method that attempts to take into account the semantics of alarms and, even more importantly, that of alarm sets. Later in this section this will enable us to develop metrics that reflect the usefulness of the information contained in alarms and alarm sets. For instance we consider alarms useful if they enable us to clearly rate an activity class variant as being malicious or to identify the activity class from which the activity class variant was derived.

8.3.1 Information provided by alarms

In Section 7.1.2.2 we introduced a 5-tuple of alarm properties to represent alarms. This representation enables us to distinguish between different generalized alarms semantically without having to describe their semantics explicitly. The latter would be difficult to do because the generating IDS determines much of the alarm's semantics, which in turn means that each of the influential IDS characteristics identified in Chapter 5 would have to taken into account accordingly. Finally the location of the IDS, also the environment, and possibly other factors influence the alarm semantics.

Our 5-tuples of alarm properties is sufficient to address all the *implicit* information conveyed by generalized alarms as they are used in the context of this work. They not only include sufficient information to distinguish alarms caused by different activity classes, but also provide information on the generating IDS. However, if we were to consider the IDS location and environment as well, the set of alarm properties might have to be extended.

In the following we wish to assess and exploit the complete information provided by generalized alarms. Hence, we searched for an analogy that would permit us to do so. We found that signals known from information and coding theory [CovTho91] provide a suitable analogy. Accordingly alarms can be viewed as being output signals that result from symbol transmissions over a channel. In this model an IDS corresponds to the channel which transforms symbols, i.e., activity variants, as they are transferred. Owing to the fact that in general information may be lost in transmission, it is not always possible to determine the initial symbol based on the output signal, i.e., the set of alarms an IDS generates. In the following we assess the completeness and utility of the alarms by measuring how well conclusions can be drawn on the respective activity class variants based on the set of alarms. This approach also enables the assessment of the potential gains one can achieve in terms of completeness and utility by combining individual IDSs, because such combinations can be viewed as the parallel use of multiple diverse channels.

Example: *Consider a network-based IDS such as Snort to represent a broad-band channel and a host-based IDS such as WebIDS a comparably narrow-band channel. This would mean that many signals, which use carrier frequencies that can be transmitted over the Snort-channel, could not be transmitted over the WebIDS channel, because the latter is not capable of transmitting signals at all the frequencies the former can. On the other hand, one can expect the signals that were successfully transmitted over the WebIDS channel to be of better quality than if they had been transmitted over the Snort channel. Reverting to ID, Snort is capable of analyzing SMTP (mail) messages, whereas WebIDS cannot. On the other hand, WebIDS is capable of taking into account the http server's request return code in order to determine the success state of a potential attack, whereas Snort cannot. Similar considerations can be made with respect to variations such as IP fragmentation. (See also Table 28).*

Note, in this model whether an alarm is rated a true or false positive is not relevant. An alarm, if the IDS actually generates one, is "just" to be seen as a transformed symbol that needs to be interpreted.

8.3.2 Fault diagnosis based on alarm sets

In this subsection we develop a method that applies the considerations made in the preceding section to the results obtained by analyzing IDSs as described in Section 7.1. The assessment method described is as well implemented by means of an extension to RIDAX, and is automatically executed as a continuation of the IDS analysis process. It is not related to the method described in Section 8.2.

Every alarm that an IDS generates indicates the observation of a possibly suspicious activity. With a probability specific to them, alarms thereby indicate an error that may lead to a security failure. However, the probabilistic aspects of the relation between activity variants and alarms are not explored in the following, because this would require an in-depth knowledge of the environment, which lies outside the scope of this thesis.

Here we view alarms and alarm sets as signals indicating a set of possible generating activities. Remember, whether an alarm is a true or a false positive is irrelevant for these considerations. In some

cases false positives may even support the identification of activities or their rating as being malicious or benign. In the following we therefore consider IDSs to be performing a simple uni-directional projection f^{v}_{IDS} that depends on the IDS being evaluated. Figure 35 illustrates this projection. Note that one of the alarm sets can denote the empty set.

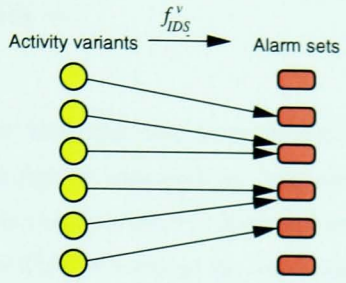


Figure 35—Projection of activity variants to alarm sets

In this projection several activity variants may cause the same alarm set R_s to be generated. By performing the IDS analysis described earlier, we obtain all these mappings between activity class variants and generalized alarms. If we do so for each activity class variant, it becomes possible to determine the sets of generalized alarms that uniquely identify a given activity class variant. However, in many cases this mapping is not unique.

In order to simplify the analysis we shall only identify the activity class but not the activity class variant derived from it. This simplification, which is illustrated in Figure 36, can be made without losing important information, because variations generally represent benign alterations of activity classes, i.e., activities.

Example: *Considering an attack that is staged over the network and targets an application layer service such as the webserver. When it comes to identifying the activity or to judging whether the observed activity is malicious or benign, the fact that the activity involved fragmented IP PDUs or minimum-sized TCP segments is of limited importance. This is not to say that this information might not be useful—especially when it comes to assessing the intentions of the adversary. However, when identifying and rating the activity, this additional information merely adds confusion.*

Figure 36c shows that it may not be possible to unambiguously identify the activity that caused a given alarm set. In the figure this is illustrated by alarm set R_3. However, knowing this relation may prove quite useful if, for instance, a_1 and a_2 are both benign, we can identify the alarm sets R_2, R_3 and R_4 as reports of benign activities that require no further attention. If both activities are considered malicious, R_3 at least informs us of a malicious activity in progress. Knowing the above mappings, we can even abridge the list of possible causes to a set of two activities. If, however, one of the activities is benign and the other malicious, any observation of R_3 is unfortunate, because it is impossible to judge whether the

detected activity is benign or malicious. This means that rating the cause, i.e., rating the activity, based on R_3 has become *ambiguous*.

When performing this analysis one might find that too many malicious activity variants

- are not detected at all,
- cannot be rated unambiguously, or
- cannot be identified.

Last but not least one might also find that too many benign activity variants cannot be rated unambiguously. These shortcomings can be addressed by increasing the information provided by the alarms sets used for analysis, which is exactly what was discussed in Section 8.3.1. In other words, using alarms generated by multiple diverse IDSs to compose the alarm sets will increase the information they provide. If we use the alarm representation as introduced in Section 7.1.2.2, the combination of alarms generated by diverse IDSs becomes straightforward and does not require any changes to our analysis method. Note, however, that combining IDSs may improve some of the issues mentioned, but may make others worse.

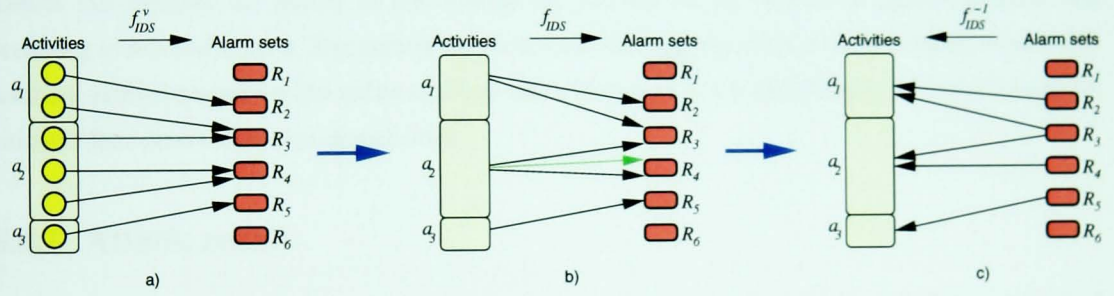


Figure 36—Projection of activities to alarm sets and vice-versa

In summary, this method of alarm-set analysis enables us to combine the alarms generated by multiple IDSs by identifying the set of activities that may cause the generation of a given alarm set. Hence an activity may be identified if the alarm set considered cannot be associated with any other activity. In a similar, but even simpler fashion, we can also attempt to rate the possible cause of an alarm set as being benign or malicious. If no clear rating can be made, the alarm set is considered ambiguous. In the examples provided in Section 8.6 we will apply these considerations to activity classes and generalized alarms.

8.4 Metrics for assessing individual IDSs and their combinations

In Section 8.2 we have defined the metrics *precision* and *recall* that enable a relatively simple but limited IDS assessment. These metrics cannot be easily applied to IDS combinations and do not assess the utility of the information provided. In this section we address this issue by proposing a set of metrics that

assesses the completeness and utility of arbitrary IDS combinations based on results obtained by analyzing alarm sets as described in the preceding section.

Before starting to define metrics, we need to clarify their goals. It is clear that in the most general case one seeks to measure and optimize *coverage* of the ID architecture envisaged (see Section 8.2). In this and the following context, it is important to ensure that each IDS is assessed for the same set of activity classes. When referring to the optimization of coverage, we should be aware that it often does not make sense to aim for total coverage. Often it suffices to optimize coverage in a given domain, i.e., just for a given set of IDS scopes, according to the security policy and the environment to be protected.

Example: *Consider a DMZ[19] of an e-business. There the majority of the activities encountered is most likely somehow related to web services. As a consequence one would concentrate on these services and pay less attention to others. The emphasis is naturally going to be different when envisaging the protection of an Intranet infrastructure.*

However, merely optimizing coverage will not result in a usable ID architecture. If the solution identified provides high coverage, but the alarms generated are too often false alarms and difficult to interpret, a solution may prove to be almost useless. To assess the utility of IDSs and their combinations, we propose metrics that measure the quality of the information provided by the sum of all generated alarm sets according to selected criteria. The criteria we choose are the ones identified in the preceding section. The resulting metrics summarize the utility of alarm sets with regard to the identification of activities and the rating of their causes as benign or malicious.

8.4.1 Attack recall

The so-called attack recall metric is highly similar to *recall* as defined in Section 7.1.3. Instead of considering only individual IDSs, we expand our considerations to a set of IDSs. One notable difference, however, is the fact that we do not distinguish between alarms generated because of variations and alarms with other causes. This is due to the manner in which alarms are analyzed (see above). In this analysis we do not distinguish between false positives and false negatives, which means that a malicious activity variant is considered *detected* if the IDS evaluated reacts with the generation of an alarm—independent of what the alarm is reporting. Note that it may not be possible to *identify* the activity based on the alarm set observed.

Furthermore, it is clear that one cannot simply add the numbers of detected and not detected malicious activity variants of all the IDSs considered because the domains they cover may overlap. In other words, m_d denotes the number of malicious activity variants that at least one of the IDSs involved was able to detect. In order not to complicate things further, let us consider activity variants that were detected only partially as "not detected." This results in the definition of m_{nd} as the number of malicious activity

[19] Demilitarized Zone: usually that part of the network that connects an organization's Intranet to the Internet.

variants that were either not detected or only partially detected by all of the IDSs. Finally we can define *attack recall* r_a as follows:

$$r_a = \frac{m_d}{m_d + m_{nd}}.$$

Again, in our case where only a single instance of every activity variant is considered, r_a provides a measurement for the coverage achieved by the combination of IDSs considered.

8.4.2 Attack identification recall

Attack identification recall is similar to the definition of attack recall. The only difference is that we count the number of malicious activity variants m_i for which it was possible to determine the activity that the activity variant was derived from. This results in the following definition of r_i:

$$r_i = \frac{m_i}{m_d + m_{nd}}.$$

It is clear that because $m_i \leq m_d$, r_i will never be larger than r_a, i.e., $r_i \leq r_a$.

Attack identification recall provides us with an absolute measure of the attacks that can be *identified* based on the information available. It thereby provides us with indication on the utility of the considered IDS or IDS combination with regard to diagnosis.

8.4.3 Attack identification precision

Attack identification recall provides a measurement in absolute terms. However, in most cases knowing the percentage of detected attacks that can be clearly identified is of higher interest. This is because it provides some measurement for the quality of the detection process. This relative metric p_i can be defined quite easily as follows:

$$p_i = \frac{m_i}{m_d}.$$

8.4.4 Rating ambiguity

In measuring recall it is important to verify whether the IDS or set of IDSs considered provide the required coverage. However, when it comes to assessing the usability of the system an important parameter is what we call the *rating ambiguity* a_r. This absolute metric measures the percentage of activity variants that cannot be rated unambiguously as benign or malicious. The number of ambiguously rated activity variants is composed of malicious (m_a) and benign (b_a) activity variants. m denotes the total number of malicious and b the total number of benign activity variants considered.

$$a_r = \frac{m_a + b_a}{m + b}.$$

The resulting measurement provides an indication of the operational effort that one has to spend because of false positives that cannot be clearly identified as being false positives. As explained earlier, we consider false positives as troublesome only if they cannot be recognized. If we observe a set of alarms that may only have benign causes we can simply discard them, i.e., they do not require any further treatment. Ambiguous alarm sets are troublesome because their generation may be caused by benign activity variants. This is especially annoying because in real-world environments benign activity variants may occur very frequently—far more frequently than the corresponding attacks. Thus one should focus on composing IDSs such that the overall rating ambiguity is as low as possible.

8.4.5 Rating precision

The metric *rating precision* p_r is closely related to the rating ambiguity metric. The two differences are that it is measured in relative terms instead of absolute ones, and that it assesses the activity variants that can be rated unambiguously. m_d denotes the malicious and b_d the benign activity variants that were detected or reported. m_{na} represents the malicious and b_{na} the benign activity variants that were unambiguously rated as benign or malicious.

$$p_r = \frac{m_{na} + b_{na}}{m_d + b_d}.$$

When measuring the rating precision we obtain an idea of the usability of the IDS combination considered with respect to the coverage it provides.

8.5 Extending RIDAX to include fault diagnosis and the calculation of metrics

Once every IDS has been analyzed for every activity class variant, the analysis process comes to its end. What we have obtained by now is a set of rather large database tables reflecting how the IDSs have analyzed the activity variants and the alarms that the IDSs were found to have the potential of generating. Based on this data, we can calculate the statistical data required for calculating per-IDS precision and recall as defined in Section 8.2.

In addition we further analyze the results obtained by identifying and analyzing alarm sets as described in Section 8.3. From this analysis we obtain, for every individual IDS but also for every possible combination of IDSs, measurements compiled using the metrics defined in Section 8.4.

8.6 RIDAX experiments

In above chapters and sections we developed methods for the analysis and assessment of IDSs. For illustration purposes we have extended RIDAX such that it is now able to calculate the IDS assessment metrics proposed. In the following we discuss experiments made using RIDAX. We do so by first providing brief descriptions of the IDSs analyzed in the course of these experiments. Then we discuss the attack detection rates, which are based on rated alarms, using the metrics described in Section 8.2. Next, we consider examples obtained while performing the far more advanced alarm-set-based fault diagnosis as developed in Section 8.3. We continue by considering measurements obtained by applying the metrics developed in Section 8.4 to the results of the alarm-set-based diagnosis. Whereas earlier attack detection rates are discussed primarily for the sake of comparison with other approaches, the measurements obtained using the alarm-set-based analysis enable us to assess the viability of IDS combinations in a manner not possible before.

Our experiments incorporate most of the concepts developed in the course of this work, including the descriptions of the 48 activity classes identified in Section 4.3 and seven variations (see also Section 6.3). Up to two of these seven variations were applied concurrently to each of the 48 activity classes. This resulted in a total of 928 activity class variants considered in our experiments. Of these 498 are considered malicious, and 430 benign. As explained in Section 6.3, any given variation can only be applied to activity classes that involve the IDS scope addressed by the variation. Therefore not every activity class leads to the same number of activity class variants.

Technically RIDAX is capable of applying any number of variations to any activity class concurrently. However, we limited this number to two because we felt it necessary to set an upper limit for practical reasons. This limit enables us to investigate the effects of multiple, concurrently applied variations, while providing us with a workable solution. The resulting number of 928 activity class variants is believed to represent a meaningful test set because they were generated in a systematic fashion, which assures consistent coverage of a significant portion of the most relevant attack classes.

8.6.1 IDSs assessed

For our experiments we have to select a small number of IDSs from a large list of candidates (see Sobirey's list [Sobire98]). This means that we need selection criteria that are well suited to our primary goal, which is the investigation of potentially achievable gains by combining diverse IDSs. As a result we selected three IDSs, of which we consider five different configurations, by following the following criteria:

1. Diversity: We require at least one knowledge-based and one behavior-based IDS, as well as at least one host-based and one network-based system.
2. Practicality: As these experiments are conducted for demonstration and validation purposes, it must be possible to describe the IDSs with only limited effort. This means that the internals of the IDSs need to be available, and, ideally, that they are already well known.

In our experiments we chose to focus on IDSs monitoring network services because various approaches to ID have been developed in this area (see also Section 4.3). This enables us to control the effort spent for the RIDAX prototype implementation as well as to address both network-based and host-based IDSs, which may either use knowledge- or behavior-based methods. We are aware that we thereby exclude classes of attacks such as those that describe attacks staged by local users (see also Section 4.3). Nevertheless we believe this choice to be viable because the goal of our experiments is merely to prove and illustrate the validity and flexibility of the concept and not so much the assessment of IDSs at large. Based on these requirements and considerations, we selected the three IDSs listed in the table below (see also Section 1.6). The last column defines the identifiers for the respective IDSs as used in the following discussion.

Table 31—IDSs analyzed and assessed using the RIDAX prototype

IDS	Detection method (see Section 5.3.1)	Information source used (see Section 5.2.1.1)	Configuration	ID
DaemonWatcher [WeDaDe00, WesDeb99]	Behavior-based	System level log (audit log)	http	DWH
			ftp	DWF
Snort [Roesch99]	Knowledge-based	External raw data (network PDUs)	Simple	SNS
			Full	SNF
WebIDS [Almgre99]	Knowledge-based	Application level log (httpd logs)	Normal	WI

We used a total of five different configurations of these three IDSs for our experiments: Two configurations of DaemonWatcher—one configured for the ftp and one for the http daemon; two configurations of Snort—one configuration using basic capabilities only and one with all additional modules enabled, and one of WebIDS.

8.6.1.1 DaemonWatcher for ftpd and httpd

DaemonWatcher [WeDaDe00, WesDeb99] by Wespi *et al.* is a behavior-based system that analyzes the audit records of processes as they are written by the OS. The system was developed at the IBM Zurich Research Laboratory, as was this work. For our experiments we consider two configurations of DaemonWatcher. A first configuration covers buffer overflow attacks against the ftp daemon, and the second covers the corresponding attacks against the http daemon.

DaemonWatcher matches the per-process sequences of system calls to system-call sub-sequences stored in a database. The latter represent known benign sequences that were isolated in a training phase. During this training phase, ideally, all possible execution paths of the executable to be protected are exercised and recorded. From these system-call traces, sub-sequences are isolated using a pattern-extraction algorithm. The resulting sub-sequences describe the complete system-call trace and thereby provide a sufficient description of the normal behavior of the executable to be monitored. As a result DaemonWatcher does not require signature updates for new attacks. However, its disadvantage is that its alarms do not identify the attack staged against the monitored process.

8.6.1.2 Snort

Snort [Roesch99] by Roesch is a network-based IDS that is freely available. Snort has become very popular, and is broadly supported by the open-source community. This support consists primarily of new signatures that are being made available on a daily basis and additional modules that extend Snort's detection capabilities. For our experiments we used two configurations of Snort v1.7. One configuration uses none of the resource-intensive extension such as the TCP stream re-assembly module. For the second configuration we consider all these additional features enabled.

Because Snort is a knowledge-based IDS, its signature database needs to be updated as new attacks become known. The signatures are primarily PDU or stream-oriented and support the verification of various protocol flags as well as string matching applied to the payload of the PDU.

8.6.1.3 WebIDS

WebIDS [Almgre99] by Almgren is a host-based, lightweight IDS tailored to the protection of webserver services (httpd) that was developed at IBM Zurich Research Laboratory. The IDS was implemented in the scripting language Perl [Perl87] and relies on its powerful regular expressions to recognize attacks. In addition the system builds up a list of suspicious hosts and is capable of statistical analysis towards the recognition of flooding attacks. It is clear that the signature database of WebIDS needs to be updated as new attacks become known.

8.6.2 Detection rates of individual IDSs

In Section 8.2 we defined the metrics precision and recall by deriving them from the corresponding definition in the information-retrieval field. The resulting measurements can only be compared in a limited fashion with the results produced by known benchmarking approaches such as the Lincoln Lab evaluation (see Section 2.3.3.1). A first important difference is the fact that we considered a "normalized" environment in which every activity variant occurs exactly once (see Chapter 7). A second difference is that the alarms and activity variants represent classes and therefore cannot be directly compared with their real-world counterparts.

The measurements produced with using RIDAX are shown in Table 32 and illustrated in Figure 37.

Table 32—Recall and precision of the IDSs assessed

IDS	Detected attack variants	Not detected attack variants	Recall (coverage)	True positives	False positives	Precision
DWF	42	456	**8.4%**	42	0	**100.0%**
DWH	112	386	**22.5%**	112	0	**100.0%**
SNF	72	426	**14.5%**	277	94	**74.7%**
SNS	44	454	**8.8%**	242	92	**72.5%**
WI	79	419	**15.9%**	164	15	**91.6%**

The first fact one notes is that DaemonWatcher for http (DWH) not only provides the highest recall but also 100% precision. This raises two questions: How does this come about? And, why are the results of

DaemonWatcher for ftp different? The main answer to this lies in the environment, i.e., the activities and activity variants used for this evaluation:

- We have defined nine malicious activities for http, but only four for ftp (see Section 4.3).
- We have defined one variation (hexadecimal encoding of the URL) that can only be applied to http-related activities. Such an additional variation can significantly increase the number of activity variants that can be derived from an activity.
- Because of the detection method used, DaemonWatcher is *not* susceptible to any of the variations considered, i.e., in the context of this work we were not able to identify a meaningful variation that could be used to elude DaemonWatcher.

The items identified illustrate that even our approach suffers some bias caused by the selection of the input data used for the evaluation. However, this impact is limited and well understood. On the other hand it seems fair to give more weight to http activities owing to the popularity of http (see also Section 4.3) and its nature, which permits variations that are not possible in other protocols. For future work it seems advisable to investigate approaches that would permit the results to be weighted on a per-activity and variation basis.

Moreover, DaemonWatcher has the inherent advantages that it does not have to repeat the error-prone network-stack processing and that it does not have to predict how the monitored application might interpret the observed activity.

Example: *A simple network layer variation that fragments IP PDUs will not prevent DaemonWatcher or WebIDS from detecting the actual attack because the host's network stack will reassemble the fragments before the data is passed on to the daemon process. For the same reason it is impossible for these IDSs to detect the variation itself.*

Note that the obtained measurements do not reflect whether the IDSs were able to provide any further information besides the fact that a number of attack variants was detected. Moreover, they do not reflect whether the success state of the attack variants considered is reported by the IDSs. DaemonWatcher, for instance, will only report attacks that were successful, whereas Snort generally cannot provide this information.

Considering Figure 37, we see that Snort has the lowest precision, followed by WebIDS and DaemonWatcher. These differences illustrate the increase in inherent difficulties the lower the level at which the IDS sensor operates is. IDSs that operate based on network PDUs have to make assumptions about how the PDUs will be treated by the target system. Based on these assumptions, they have to move up further levels of abstraction, trying to predict how the observed data PDUs or data sequences will be interpreted. This excessive crossing of abstraction levels typically results in relatively generic signatures that may not only be evaded, but may also cause false positives. Similar conclusions were drawn by Sekar *et al.* [SGVS99]. As a consequence their implementation of a network-based IDS only focuses on attacks up to the transport layer, and rarely considers application layer attacks.

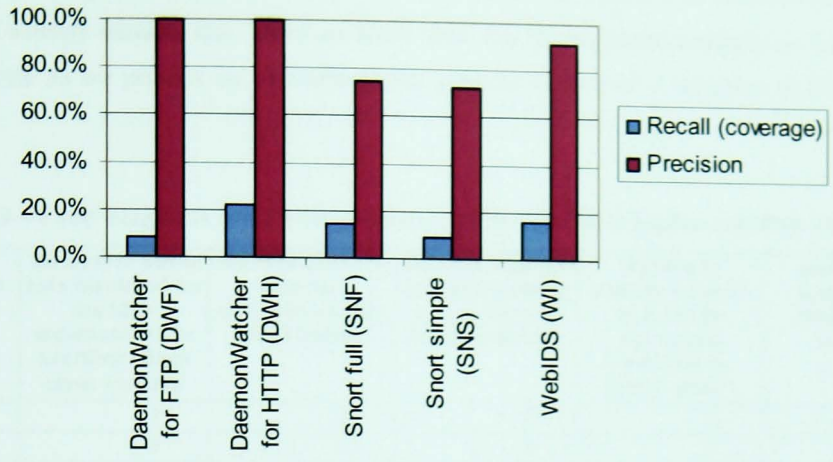


Figure 37—Chart representing precision and recall

8.6.3 Examples of alarm-set-based fault diagnosis

Especially for large-scale ID architectures, the meaningful processing of alarms is vital. One important factor is certainly the handling of the vast number of false positives one can expect. However, false positives are only part of the problem. With increasing size of the ID architecture, also the difficulty of interpreting the symptoms of a cause that may be malicious increases significantly. Worse, these two issues are closely related and therefore should not be separated. This resulted in the alarm-set-based approach to fault diagnosis developed in Section 8.3.

Depending on the number of IDSs combined, we were able to identify up to 72 unique alarm sets. However, when considering individual IDSs such as DaemonWatcher, this number may be as small as one. Each of the identified alarm sets may be caused by one or several activities. In the following, we discuss some examples of alarm sets as they were generated in the context of the RIDAX experiments.

We consider the http-argument buffer overflow attack we have already used several times as an example. This class of attacks was identified as activity 1 in Section 4.3, Table 4. In our experiments RIDAX was capable of considering 28 attack class variants of this attack class.

In DaemonWatcher (DWH) each of these 28 activity variants is being reported by an alarm indicating that the execution path of the process implementing the http services is diverting. However, as one might imagine, other activities can have the same effect and are reported by the same alarms. Based on DaemonWatcher alarms only, it is therefore not possible to clearly identify the activity, i.e., the cause of the alarm. Other IDSs, such as Snort or WebIDS, generate alarms indicating the observation of a suspicious or overly long argument string. However, these alarms do not permit the clear identification of the attack class either because they may also report other, possibly benign, activity classes. Moreover, in many cases, variations render the attack class invisible to these IDSs. In our experiments we found IDS combinations that would report activity variants of this activity with up to eleven different alarm sets.

However, it is not our goal to maximize this number. It should be the goal to find solutions that maximize the number of activity variants that, based on alarm sets, can be associated uniquely to the generating activity. In Table 33 we provide an overview of the relevant figures for a selection of IDSs and IDS combinations.

Table 33—Fault diagnosis results for the class of http argument buffer overflow attacks

IDS combination	Number of alarm sets identified for the http-argument buffer overflow attack class variants	Number of alarm sets that uniquely identify attack class 1	Number of alarm sets that identify only malicious activity classes	Number of activity variants that can be associated uniquely to attack class 1	Number of undetected attack class variants
DWH	1	0	1	0	0
SNF	6	0	0	0	18
WI	3	2	2	21	6
DWH, SNF	7	0	7	0	0
DWH, WI	4	2	4	21	0
SNF, WI	10	8	8	21	4
DWH, SNF, WI	11	8	11	21	0

In the above table we also included the number of undetected attack class variants and the number of alarm sets, i.e., sets of generalized alarms, caused only by attack classes. These numbers become important in the next section as we assess the utility and the coverage of IDS combinations. As explained in Section 8.3.2, an important factor is the number of alarm sets that may denote malicious as well as benign activity classes. Remember, we do not consider false positives to be problematic as long as they can be recognized as such. On the other hand, false positives are particularly annoying if the attack-similar but benign activity causing them occurs. It is difficult to eliminate them in a generic manner, because they may not be distinguishable from alarm sets reporting classes of malicious activity.

8.6.4 Measuring the results of alarm-set-based fault diagnosis

In this section we discuss the measurements made while performing fault diagnosis based on IDS analysis results. The fault diagnosis and measurements are made as introduced in Section 8.3. As opposed to the measurements determined in Section 8.6.3, those determined here provide us with indications of the utility of the information provided by alarms. The following discussion is primarily concerned with the metrics defined in Section 8.4. It not only considers the results provided by the five individual IDS configurations, but also those provided by all possible combinations thereof.

We start with a table that shows the measurements determined for all possible IDS combinations while performing fault diagnosis as described in Section 8.3. In contrast to the measurements discussed in Section 8.6.2, this analysis includes alarms reporting variations. In a next step we discuss bar charts and a Venn diagram illustrating the results. Then, in a third step, we discuss plots illustrating how different metrics may influence each other. Finally we discuss the impact and use of alarms reporting variations, and illustrate their impact by comparing fault-diagnosis results that include them with results where they are excluded.

The measurements shown in Table 34 are ordered by attack recall and by rating ambiguity. Intuitively one wishes to maximize attack recall, i.e., coverage, while minimizing ambiguity. As shown in Table 34 and illustrated in Figure 38, most IDS combinations result in a higher coverage than individual IDSs or smaller sets of combined IDSs do. However, in most cases the rating ambiguity seems to increase or at least to be relatively high. Considering Figure 38, we can identify Snort as having a negative impact on the rating ambiguity. On the other hand Snort generally increases attack recall quite significantly.

Table 34—Measurements resulting from alarm-set-based fault diagnosis (including variation alarms)

IDS combination	Attack recall (coverage)	Rating ambiguity	Rating precision	Attack identification recall	Attack identification precision
DWF, DWH, SNF, SNS, WI	55.6%	19.2%	57.6%	18.5%	33.2%
DWF, DWH, SNF, WI	55.6%	19.4%	57.1%	18.5%	33.2%
DWF, DWH, SNS, WI	50.0%	19.2%	54.4%	12.9%	25.7%
DWH, SNF, SNS, WI	49.2%	20.3%	51.5%	18.5%	37.6%
DWH, SNF, WI	49.2%	20.5%	51.0%	18.5%	37.6%
DWF, DWH, SNF, SNS	47.8%	16.6%	57.7%	5.6%	11.8%
DWF, DWH, SNF	47.8%	16.8%	57.1%	5.6%	11.8%
DWF, SNF, SNS, WI	46.4%	24.0%	40.4%	15.5%	33.3%
DWF, SNF, WI	46.4%	24.2%	39.8%	15.5%	33.3%
DWH, SNS, WI	43.6%	20.3%	47.5%	12.9%	29.5%
DWF, DWH, SNS	42.2%	16.6%	53.9%	0.0%	0.0%
DWF, DWH, WI	41.4%	7.4%	72.5%	12.9%	31.1%
DWH, SNF, SNS	41.4%	17.7%	50.6%	5.6%	13.6%
DWH, SNF	41.4%	17.9%	50.0%	5.6%	13.6%
DWF, SNS, WI	40.8%	24.0%	35.2%	9.8%	24.1%
SNF, SNS, WI	40.0%	25.1%	31.9%	15.5%	38.7%
SNF, WI	40.0%	25.3%	31.3%	15.5%	38.7%
DWH, SNS	35.7%	17.7%	45.7%	0.0%	0.0%
SNS, WI	34.3%	25.1%	25.3%	9.8%	28.7%
DWF, SNF, SNS	33.3%	20.9%	33.6%	5.6%	16.9%
DWF, SNF	33.3%	21.1%	32.9%	5.6%	16.9%
DWH, WI	32.9%	7.4%	67.0%	12.9%	39.0%
DWF, DWH	30.9%	0.0%	100.0%	0.0%	0.0%
DWF, WI	29.5%	10.9%	47.4%	9.8%	33.3%
DWF, SNS	27.7%	20.9%	26.0%	0.0%	0.0%
SNF, SNS	26.9%	22.0%	21.5%	5.6%	20.9%
SNF	26.9%	22.2%	20.8%	5.6%	20.9%
DWH	22.5%	0.0%	100.0%	0.0%	0.0%
SNS	21.3%	22.0%	11.3%	0.0%	0.0%
WI	21.1%	10.9%	32.7%	9.8%	46.7%
DWF	8.4%	0.0%	100.0%	0.0%	0.0%

Therefore the question arises whether one could combine Snort with other IDSs in such a way that the rating ambiguity decreases. Figure 38 clearly shows that this is possible. However, the improvement does not seem as significant as one might hope. Indeed, in many cases the situation becomes worse. This phenomenon is inherent in the combination of IDSs as illustrated by the following example.

Example: *Combining Snort (SNF) with WebIDS (WI) results in relatively high attack recall, but causes the rating ambiguity to increase above 25%. On the other hand combining Snort (SNF) with DaemonWatcher (DWF, DWH) improves the situation from 22.2% down to 16.8% rating ambiguity.*

Where do these differences come from? Snort and WebIDS partially cover the same attacks and by combining them one can improve coverage. However, because both systems use similar techniques for detecting an attack, both are susceptible to generating false alarms for similar activities. As a further consequence of using similar techniques, the semantics of their alarms are also similar[20]*. Because of this it becomes difficult for any fault-diagnosis algorithm to compensate the increased ambiguity caused by the increased number of benign activity variants that may cause alarms. The situation changes significantly when considering the combination of Snort and DaemonWatcher. These IDSs use completely different techniques and information sources for their analysis, and are therefore not susceptible to the same variations. Here it becomes not only possible to increase coverage, but also to compensate weaknesses of the respective IDSs by exploiting the semantic diversity of the alarms they generate.*

Similar observations can be made when considering attack identification recall. However, here the strengths and weaknesses are distributed differently among the IDSs. An IDS that distinguishes well between malicious and benign activity class variants is not necessarily as good when it comes to *identifying* the activity class causing a given alarm set. An extreme example of this is DaemonWatcher. This IDS distinguishes well between malicious and benign activities. However, as its alarms do not carry semantics beyond the fact that an unusual sequence of system calls was observed, it becomes impossible to determine the cause of such an observation. On the other hand, we are able to demonstrate that such systems may significantly increase the expressiveness, i.e., attack identification recall, of IDS combinations.

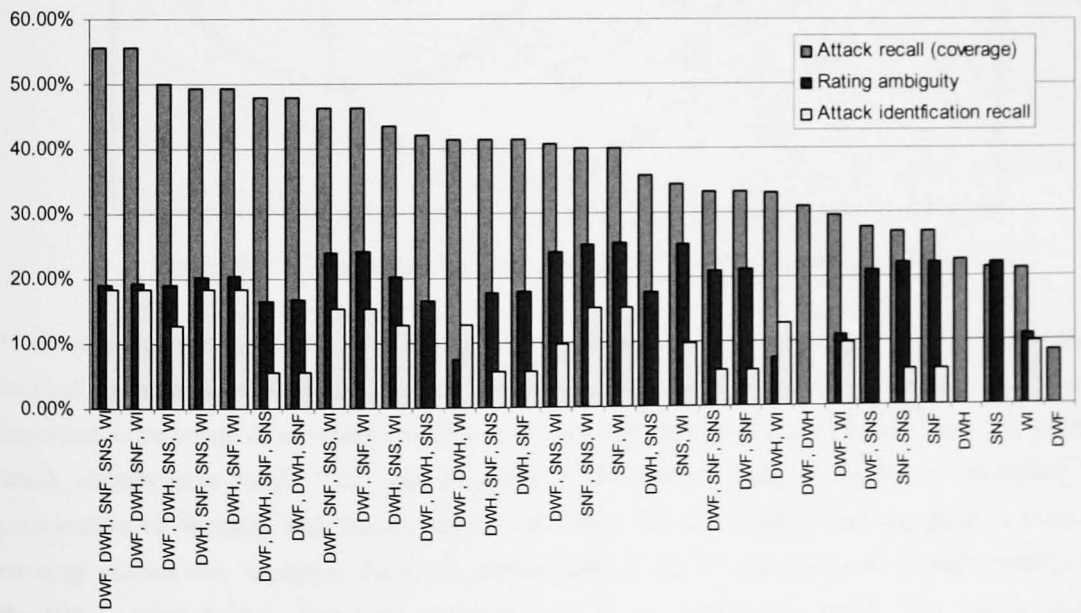


Figure 38—Attack recall, rating ambiguity and attack identification recall

[20] Using dependability terms, this indicates a high probability for the existence of common failure-modes.

Example: *Considering Figure 38, it is apparent that the combination of WebIDS (WI) and DaemonWatcher for http (DWH) results in an significant increase of the attack identification recall compared with any of the respective stand-alone configurations.*

Considering attack recall, i.e., coverage, as shown in Figure 38, we already noted that combining IDSs in most cases results in increased coverage. In some cases coverage increases significantly, and in others the increase is less important. The reason for this is quite simple: some IDSs overlap significantly in terms of coverage, and some do not overlap at all. Figure 39 illustrates how the five IDS configurations evaluated overlap.

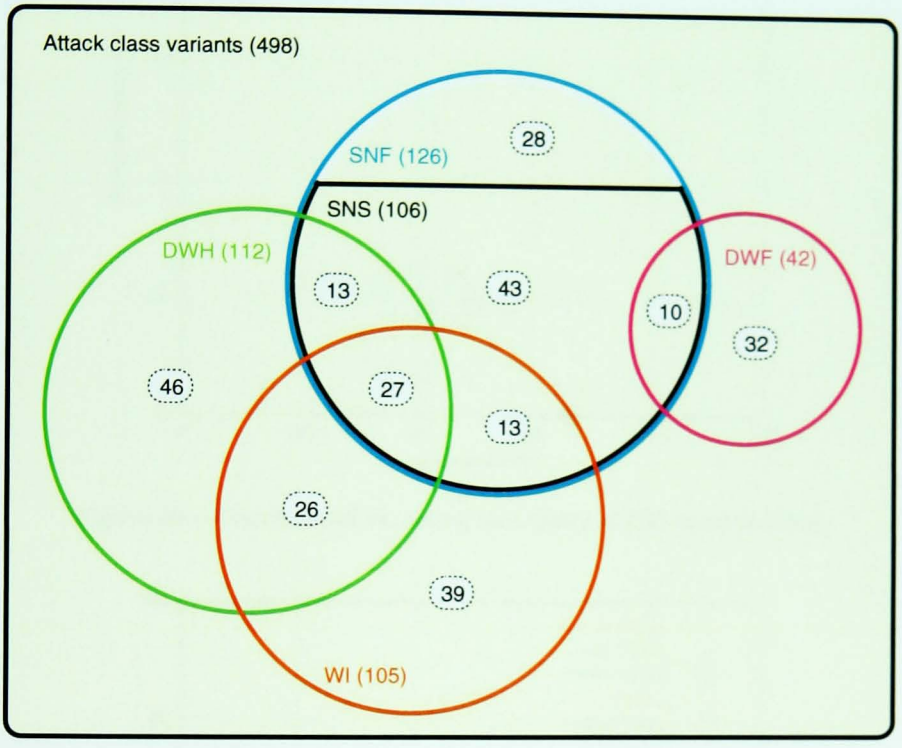


Figure 39—Venn-diagram showing coverage overlaps of evaluated IDSs

When seeking meaningful IDS combinations, it is important to know how the individual IDSs overlap in terms of coverage, in order to determine the coverage provided by a set of IDSs. However, it is even more important to be aware of coverage overlaps when seeking ways to reduce ambiguity and/or to increase attack identification recall. This was illustrated in the above example, where we considered the combination of WebIDS and DaemonWatcher for http. For this combination the gains in terms of coverage are not very important. However, the example clearly illustrates how the diverse semantics of the IDSs' alarms lead to improved ambiguity and attack identification recall. This can be further illustrated in plots such as those shown in Figure 40 and Figure 41. These plots reemphasize the fact that not every combination of IDSs leads to improved usability in terms of rating ambiguity or attack identification recall. In Figure 40, for instance, we can identify three bands of IDS combinations. A first band on the x-axis, a second around 10% rating ambiguity, and third around 20% rating ambiguity. Taking a closer look at the data provided in Table 34, we recognize the first band to consist of just the

two DaemonWatcher configurations and their combination. The band at the 10% level is defined by WebIDS, and represents the various combinations of WebIDS with DaemonWatcher configurations. The band at 20% is dominated by Snort. Every IDS combination at this level contains at least one of the two Snort configurations.

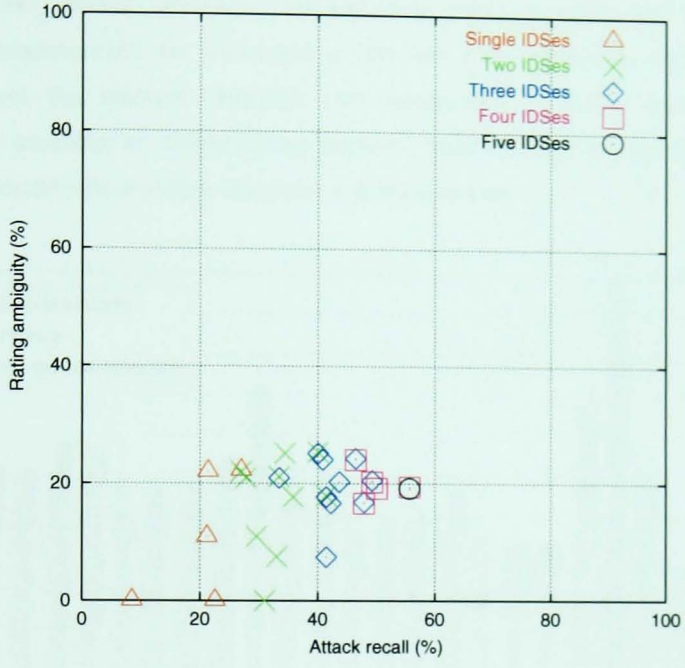


Figure 40—Attack recall vs. rating ambiguity of IDS combinations

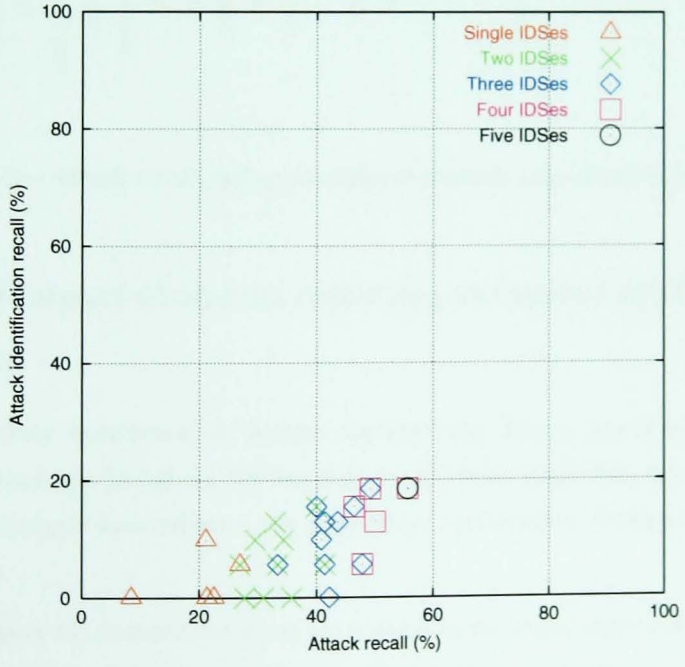


Figure 41—Attack recall vs. attack identification recall of IDS combinations

In combination with Table 34 these plots may also be used to investigate the impact of IDS crash failures. Assuming that one out of three combined IDSs fails, these results enable us to assess the degradation in terms of coverage to be expected etc.

So far we considered the *absolute* "recall" measurements only. To complete this discussion we compare attack recall with the *relative* "precision" measurements rating precision and attack identification precision. These measurements are illustrated in the bar chart shown in Figure 42. Also these measurements reflect the inherent strengths and weaknesses of IDSs, such as the fact that DaemonWatcher is excellent at distinguishing between malicious and benign activity variants, but completely fails to identify the activities that cause a given alarm set.

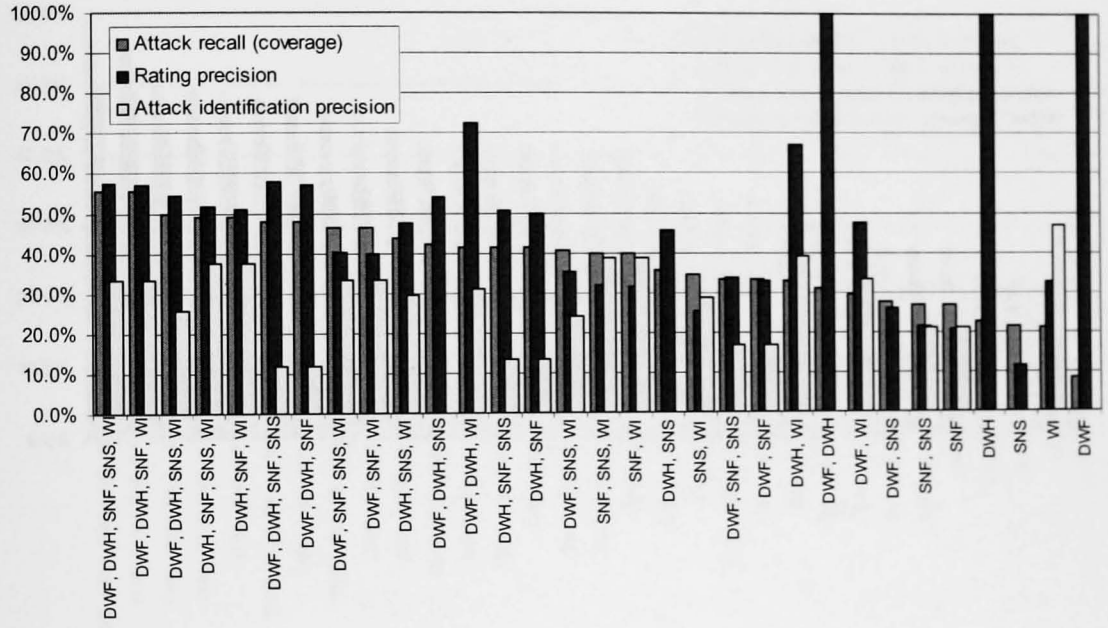


Figure 42—Attack recall, rating precision and attack identification precision

8.6.5 Use and impact of alarms reporting variations applied to activities

While performing these experiments it became apparent that alarms reporting the observation of variations are of limited use. In fact we felt that they would create misleading measurements for attack recall. To further investigate their influence, we repeated our experiments—disregarding alarms reporting variations.

Our analysis (see Figure 43) revealed that doing so caused both the attack recall and the rating ambiguity to decrease (measurements for rating and attack identification precision are affected even more significantly because they rely on relative metrics). The attack identification recall is not affected by this measure. At first glance the fact that the attack recall decreases appears as a disadvantage. However, if an

attack is reported only by means of an alarm indicating the presence of a variation, a suitable reaction to it is difficult. These alarms are highly ambiguous as, in practice, the majority of them report non-malicious activities. Second, assuming the presence of malicious activity, these alarms hardly provide any useful information about the attack observed. In Figure 43 the generally observable drop of the rating ambiguity nicely illustrates these facts. In most cases the drop in rating ambiguity is far more important than the loss of coverage in terms of attack recall.

As a result we concluded that alarms reporting the presence of variations should be used for adjusting the *severity* one associates with the potential cause of a given alarm set. The fact that an adversary obfuscates its attacks might provide an indication of the tools used and/or the adversary's skills.

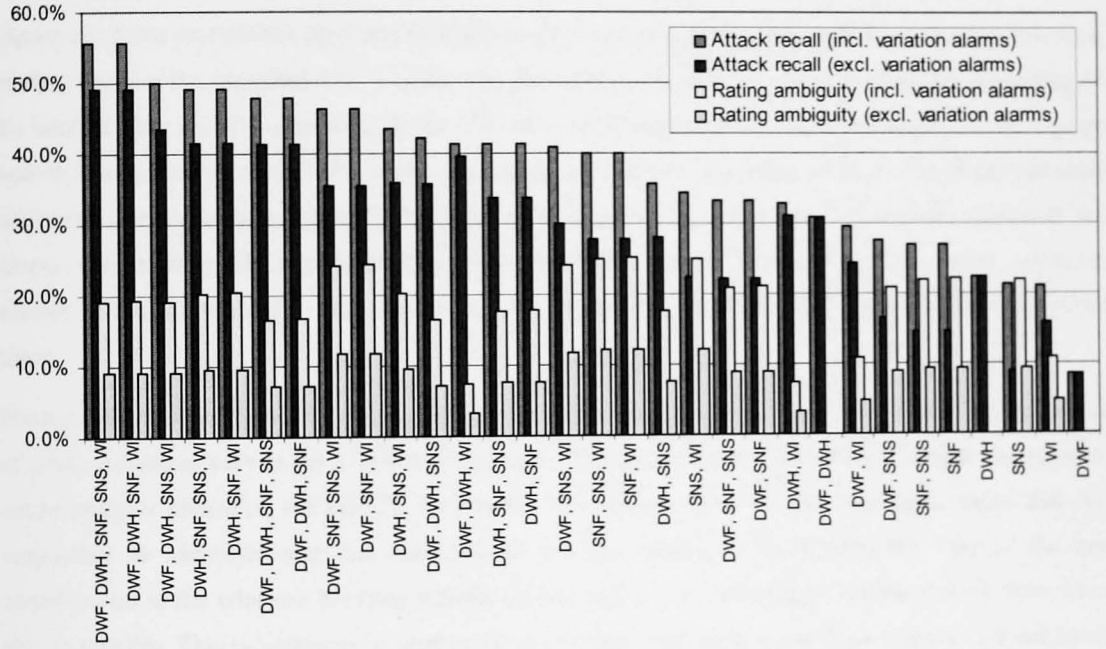


Figure 43—Attack recall and rating ambiguity including vs. excluding alarms reporting variations

8.7 Discussion

In this chapter we have used the results produced by our approach to IDS analysis for assessing IDS combinations at a conceptual level. The approach to IDS assessment presented cannot be compared with existing approaches directly simply because it operates at a different level of abstraction and also pursues different goals. Nevertheless we defined metrics that enable us to assess IDSs in a manner similar to what is done by other approaches (see Section 8.2). It is clear that measurements resulting from the use of these metrics will therefore suffer from the same deficiencies as the ones determined using other approaches such as the Lincoln Lab experiment (see Section 2.3.3.1) or the more commercially oriented work pursued by the NSS Group [Walder01a, Walder01b]. These deficiencies are primarily due to the fact that the environment significantly biases the measurements and that the environment by nature is neither identical at any two places nor stable over time (see also Section 2.4).

Therefore we focused on assessing the potential of fault diagnosis to develop metrics that provide indications on the utility of the information provided by individual IDSs and their combinations (see Section 8.3). One of the key differences to the metrics described in Section 8.2 is the fact that we consider false positives only as problematic if they cannot be eliminated based on known relations between activity classes and sets of generalized alarm they may cause IDSs to generate. As a result we developed metrics that enable us to assess factors such as the expectable coverage or utility of the information. The utility is assessed by measuring the extent to which malicious and benign activity class variants can be distinguished, and by measuring the share of attack classes that can be identified unambiguously.

The ability to assess the completeness and utility of the information provided by IDS combinations supports the development of ID architectures as it permits the comparison of architecture proposals.

However, if one extends the proposed methods to the real world, the analysis will become more involved, mainly because the identification of alarm sets becomes more complex. Also the fact that depending on its internal state and the environment, the IDS may not always generate the same alarm set for a given activity increases the complexity of this reasoning. In addition one needs to be aware of the semantic differences between the generalized alarms generated in the context of our IDS analysis approach and alarms generated by IDS implementations. The generalized alarms, similar to activity classes, represent classes of real alarms rather than alarms that report specific attacks as IDS implementations generate them.

Having taken note of these differences, it is hoped that these results will support the further enhancement of existing systems such as the Tivoli Risk Manager [TRM00] towards advanced fault diagnosis and *root-cause analysis* [Julisc01, KYYOS95, ParBus88]. A *root-cause* denotes the most basic cause that can reasonably be identified and that management has the control to fix [ParBus88]. One of the key contributions is the relations between activity classes and sets of generalized alarms that we have been able to identify. This promises to be applicable in a generic manner because these relations are not made at the level of specific real-world attacks or alarms, but rather at the level of attack classes and alarm classes.

8.7.1 ID architecture design process example

The information our approach provides may be used in the context of the design process of an ID architecture. For a given portion of the infrastructure considered a conceivable procedure could look as follows:

1. Attack classes: Based on the security policy one determines the attack classes that have to be covered (see Sections 2.2.2 and 3.6.1).
2. Identification of IDS combinations: From a given set of IDSs one determines the IDS combinations that provide the coverage required (see Section 8.4.1)
3. Selection of the IDS combination: Depending on the importance of the individual factors, one selects the most suitable combination of IDSs. Relevant factors are, for example, the following:

- The difficulty of discriminating true and false positives. A measure for this difficulty is the rating ambiguity defined in Section 8.4.4.
- The ability to identify the attack based on the alarms generated by the IDSs. This ability is measured by the attack identification recall defined in Section 8.4.2.
- System performance degradation due to failing IDSs. Whenever one component of a given combination of IDSs fails, the performance (see Section 8.4) of the IDS combination is degraded. The performance of the remaining IDSs corresponds to the performance of their combination. As the performance of both combinations is known (see Table 34), it is possible to investigate the impact of failing IDSs on the overall system.

In addition to this analysis one might take into account additional factors such as costs, quality of the IDS implementation, services provided by the IDS vendor, e.g., updating of signature databases, etc. As the requirements of organizations may differ significantly, many diverse ID architecture designs become conceivable.

8.7.2 Discussion of experiments

Roughly speaking we were able to identify three classes of results. First, we were able to produce precision and recall measurements for individual IDSs. Second, we were able to identify inter-class relationships between activity classes and sets of generalized alarm, which may serve as the foundation for future work on alarm processing towards root-cause analysis. Finally, we were able to exercise the metrics of attack recall, rating ambiguity etc., which we defined for our alarm set-based approach to fault diagnosis. These measurements enable an assessment of the potential viability of individual IDSs and, more importantly, of IDS combinations. All these results were obtained using RIDAX to analyze the 928 activity variants derived in an automated fashion from the 48 activities identified in Section 4.3. We created these variants by selecting up to two variations out of a set of seven variations described. Of the 928 resulting activity variants, 498 are to be considered malicious.

An important result is the inter-class relationships between activity classes and sets of generalized alarms that we were able to identify. Depending on the number of IDSs combined, we were able to identify up to 72 unique alarm sets. For individual IDSs, such as DaemonWatcher, this number is as small as one. Any given set of generalized alarms identifies one or several activity classes that may cause the generation of the alarm set considered. Recalling that generalized alarms represent classes of real-world alarms, our results describe relationships between the two classes. The knowledge of these inter-class relationships will support the development of future alarm-processing algorithms, such as will be required in large-scale ID architectures consisting of many highly diverse IDSs. The importance of this knowledge is likely to increase because existing solutions such as IBM's Tivoli Risk Manager [TRM00] gradually improve their alarm processing towards root-cause analysis. What has been developed in this chapter represents a proposal for an approach to develop this knowledge. The manner in which this knowledge can be used has been demonstrated in the context of the MAFTIA project [Alessa03a], where the alarms generated by diverse IDSs are combined based on knowledge obtain in this way.

In addition, in Section 8.6.2, we briefly considered detection rates of individual IDSs that were determined in ways similar to what is done in other approaches. We chose not to develop these measurements much further as they seem not well suited to our approach for three reasons:

1. The underlying metrics cannot be extended to incorporate results from multiple IDSs in a meaningful manner.
2. The measurements obtained do not permit judging the utility of the IDSs with respect to the semantics of the alarms they generate. This specifically includes the ability to distinguish false positives from true positives and the identification of the alarm cause, i.e., the activity.
3. As a consequence the metrics used only seem to provide meaningful information when applied to a real IDS implementation that is being evaluated in the environment for which it is envisaged. Because our approach operates at the more conceptual level of classes and does not include the modeling of an environment, the measurements obtained must be interpreted with care.

Therefore we focused on alarm-set-based fault diagnosis instead. Again, one needs to keep in mind that we do not consider a real-world environment, but rather a normalized environment, in which every activity variant is considered exactly once. Identifying the inter-class relationships does not require a model of a real environment because the relevant attack characteristics are not influenced by the environment. Also, for assessing the viability of IDS combinations, a normalized environment proved sufficient. While performing the alarm-set-based fault diagnosis and measuring its results, we were able to validate and quantify a number of common assumptions and to learn numerous lessons:

- Substantial gains in coverage, i.e., attack recall, result from the combination of IDSs with as distinct a coverage as possible (see also Figure 39).
- Improving the rating ambiguity by combining IDSs requires a significant overlap in coverage.
- The techniques used by the combined IDSs should be as diverse as possible.
- It is possible to slightly improve the rating ambiguity while combining IDSs to improve coverage, but this proved to be a difficult task requiring special care.
- Similar observations as made for rating ambiguity measurements apply to attack identification recall measurements.
- Alarms indicating the use of variation techniques increase both coverage and rating ambiguity. One should not consider attacks as detected if they are only reported by an alarm indicating the use of variation techniques. In other words, such alarms should not be included in attack recall calculations because in practice this leads to misleading measurements of coverage.
- It might be meaningful to use alarms that report the use of variation techniques to adjust the severity level associated with a finding presented to the human security officer on duty. Also such alarms may be used to facilitate the identification of the attack-tool used by the adversary.

- We were able to validate the common perception [SGVS99] that IDSs should not operate across too many levels of abstraction, e.g., protocol layers, while performing their analysis. Specifically, although generally considered very convenient to implement, systems applying any of kind string-pattern-matching algorithm to (external) raw data sources (see Section 5.2.1.1) are prone to false positives and obfuscation. Furthermore the severity of this issue clearly increases the more abstraction levels an IDS attempts to monitor. An example is the class of network-based systems, which in general are particularly susceptible to false positives due to string mismatches and obfuscation techniques.

In our experiments we always considered the total number of attack class variants identified—assuming a normalized environment. In future work one might wish to expand this by weighting the activity classes and variations according to the coverage required by the security policy and according to their importance in the respective environments. For instance one might wish to maximize coverage for http-related activities.

When taking into account the environment one should ideally use a model of the environment an individual IDS or the complete ID architecture is envisaged for. If this is not possible, one might consider the creation of multiple environment profiles, each describing a particular class of environments. Such profiles might include descriptions of typical commercial DMZs, Microsoft Windows-dominated intranets, Unix-based environments, etc. At the stage where one weights the assessment results, one might even choose to combine multiple environment descriptions if appropriate and necessary. Although this is more complex to implement, we believe that IDS benchmarking approaches as described in Section 2.3.3 should use multiple environment profiles rather than just a single one for their assessment. One might also want to assess the expressiveness of the alarms generated. In other words, one should not focus on the pure numbers of false and true positives but instead assess how well attacks can be *identified* and how well false and true positives can be *distinguished*.

Lastly, we note that our experiments consider worst-case scenarios only. This means that when considering an attack, we always assumed the attack to be successful. In future work, one might extend RIDAX towards considering successful and unsuccessful instances of each attack. This extension should result in more finely grained results for IDSs such as DaemonWatcher that are only able to detect successful attacks.

8.8 Conclusion

The IDS assessment method described here represents an application and extension of the approach to IDS analysis developed in this work. It provides a systematic way to assess individual IDSs and the benefits attainable thanks to the combination of diverse IDSs. However, one should bear in mind that the results obtained from such an analysis are based on descriptions of IDSs, and not on experiments with real systems, i.e., IDS implementations. The assessment results are based on the analysis for classes of activities rather than for specific activities, e.g., particular attacks. Thus the results are of rather limited use when assessing IDSs for particular attacks. Instead, the results can be used for the design and

development of ID architectures that combine multiple IDSs to improve completeness and utility. Moreover they may be used to derive high-level alarm-correlation rules for fault diagnosis and root-cause analysis purposes. Last but not least, we have illustrated a way in which the results provided by our approach to IDS analysis can be used for further investigations and analysis.

Chapter 9 Conclusions and future work

In this work we have presented a novel approach to IDS analysis that supports and simplifies the difficult task of designing and analyzing IDSs. The approach operates at a conceptual level and thereby facilitates the identification and analysis of strengths and weaknesses of IDS designs—even before they are actually implemented. Thus, the approach can be used to provide guidance to IDS designers at an early stage of the design process by predicting the set of attack classes that an IDS design will be able to detect when implemented. In addition our approach enables IDS designers to create a detailed specification for the IDS envisaged by precisely specifying the classes of attacks that the IDS has to be able to detect. The specification may even detail the manner in which the IDS has to report its findings.

In order to achieve the goal of improving the design process for IDSs, a number of well-structured concepts have been developed. These namely include highly generic and concise schemes for describing IDSs and classifying attacks. As illustrated by an example that assesses the information provided the combination of diverse IDSs, these schemes may serve as the basis for future conceptual work in the domain of ID. Similarly, this also applies to the analysis results produced by our IDS analysis approach.

9.1 Contributions

The main contribution of this work is a novel method of IDS analysis that provides guidance to IDS designers at early stage of the design process. It does so by predicting the attack classes that an implementation of the envisaged design will be able to detect and may even be used to express and verify an essential part of the IDS specification.

While working towards this goal, we designed, built, and continuously enhanced a highly structured security database (VulDa) [DacAle99]. We used the system to develop a scheme that categorizes attacks according to aspects of attack that are visible to IDSs. The categorization of over 350 attacks enabled us to identify categories of attacks that are relevant in practice. The VulDa system moreover evolved to a heavily used information system that provides access to security related information to hundreds of security professionals within IBM.

This attack categorization served as the basis for some the following concepts, methods and tools that we have developed in the context of this work:

- Description of IDSs: Based on the insights gained from creating the above attack categorization, we developed a highly flexible and generic but concise *description framework for IDSs* that uses multiple dimensions to describe IDS characteristics. The high generality and consistency are achieved thanks to the introduction of the so-called *IDS scopes* as one of the scheme's dimensions. It thereby enables a clear separation of ID techniques and the domain (IDS scope) to which these techniques can be applied.

- Classification and description of attacks: We have developed an attack classification that yields concise descriptions of attack classes. The resulting descriptions are expressed in terms of characteristics that are required of an IDS to analyze attack classes, i.e., attack classes are described at the same level of abstraction as IDSs are described. The scheme moreover supports the systematic identification and description of attack class *variants*. These variants reflect commonly used practices for obfuscating attacks in order to evade detection by IDSs. In addition to the analysis of IDSs, the set of identified attack classes and attack class variants can also be used for creating a part of the *IDS specification*. More specifically they can be used to set the requirements with regard to the attack classes an IDS has to be able to detect. Finally, it should be mentioned that the description scheme is of sufficient generality such that it can be used to describe not only attacks but also benign activities.
- Novel IDS analysis method: In order to predict the attack classes that IDSs, i.e., implementations of IDS designs, are able to detect we have developed a method that performs a combined analysis of IDS descriptions and attack class descriptions. It thereby determines the set of attack classes that IDSs have the potential of detecting. The results obtained provide guidance to the designers of IDSs. This guidance may be highly valuable because it can be provided at an early stage of the design process, i.e., before the IDS design has been implemented. Similarly, the method may also be used to determine the suitability of existing IDSs to detect newly discovered classes of attacks.
- RIDAX tool: We have created a prototype implementation of the entire approach to IDS analysis. The results of experimenting with RIDAX were used to outline a possible validation approach and thereby provide some evidence that the predictions made are actually accurate. Additionally, we have used the results to illustrate further potential applications for our approach by means of an example that shows how combinations of diverse IDSs can be assessed with respect to the usefulness of the information conveyed by the alarms that they may generate.

9.2 Discussion

In this thesis we have presented a novel approach for analyzing IDSs. It was the declared goal to support IDS designers early on in the design process by providing them with guidance in terms of the attack classes a given IDS design has the potential of detecting. The eminent advantage of this approach is that IDS designs can be analyzed before they have actually been implemented. This requires that the predictions made must be reliable. Although highly desirable, a rigorous validation of the results produced by our approach represents an undertaking that goes beyond what can be practically achieved. In order to provide some evidence of the accuracy of the predictions made by our approach, we have instead successfully illustrated and outlined a less involved validation approach that verifies the predictions made more selectively.

It may, however, still happen that the predictions made for an IDS design are not met by the IDS implementation. In this case the reasons for the discrepancy must be investigated. Our investigations

revealed that the predictions with regard to requirements for our description-based analysis approach were always consistent. The investigations, however, also revealed that in some cases the IDS implementation fails in meeting the requirements set by its specification and design owing to implementation flaws, i.e., bugs. Moreover there exist techniques used by IDS implementers the description of which would require an IDS description framework that supports even more detail than the one we have presented. As we have illustrated by means of the IP fragmentation example in Section 7.3.1, such techniques are typically ad-hoc and highly specific to a particular ID problem. Although the proposed IDS description framework could be extended to capture such IDS-specific peculiarities, we remained at the level of detail chosen. The attainable benefits from a systematic, i.e., uniform, extension of the description framework at such a level of detail would have been minor compared with the significant increase in complexity of the entire analysis process. Such an increase of complexity would have been undesirable as it was one of the goals to avoid unnecessary complexity, i.e., to find a balance between expressiveness and simplicity.

Many of the concepts developed in this work may be used as the foundation for future work in the domain of ID. In Chapter 8, for instance, we have described an approach that uses RIDAX results for analyzing the utility of arbitrary combinations of diverse IDSs. Note that further refinements would be necessary before such an approach can be used in practice, but it represents an illustrative example that highlights the possibilities of analyzing IDSs at a conceptual level.

9.3 Future directions

As we illustrated in Chapter 8, this work may be extended in a number of different directions. In addition to the IDS assessment described in Chapter 8, additional avenues of further research could be the following:

- Environment: Our description-based approach could be extended towards investigating the effects of environmental factors such as the traffic volume or the system load under which IDSs have to operate. If combinations of IDSs are to be analyzed it will most likely be necessary to take into account the network topology as well.
- False alarms: In the work as presented here we did not focus on the problem of IDSs generating excessive numbers of false alarms (except in the example presented in Chapter 8). We have, however, shown that the schemes presented can be used for analyzing the potential of IDSs to generate false alarms. This analysis could be extended further to provide even more information to IDS designers—supporting them in identifying potential causes for IDSs generating false alarms. To some limited degree, such analysis methods have been developed and explored in Chapter 8, but additional work will be necessary to develop these concepts further and to validate them.
- Analysis costs: Security always comes at a certain cost and so does ID. One aspect to be taken into account is the impact that IDSs may have on the systems they survey. This impact should remain below an "acceptable" limit. Another aspect is the possibility of the IDS being saturated

by the number of activities to analyze. As a consequence we suggest the introduction of per-IDS characteristic costs. These costs could be composed of cost items such as "CPU utilization," "memory utilization," and "storage utilization." Once RIDAX has completed the analysis of an activity class, one could then simply calculate the total costs caused by the IDS characteristics required for the analysis of activities that belong to that class.

- Worst-case considerations: RIDAX could be extended so that it performs its analysis for more than just worst-case scenarios. Currently the system assumes each attack to be successful. This is not always the case, and should be reflected in future extensions to RIDAX (see also Section 8.7.2).
- Alarm correlation: The results obtained from our approach to IDS analysis may be used to investigate and develop novel approaches to ID alarm correlation, i.e., techniques that aggregate and interpret the alarms that are generated by multiple, possibly diverse, IDSs. An approach of how this could be realized and used has already been demonstrated in the context of the MAFTIA project [Alessa03a, D10Maf02].

Last but not least, it is hoped that this work will help to promote efforts that go beyond the items just identified and thereby improve the utility of ID in general.

Appendix A VulDa, a database of collected attacks and vulnerabilities

VulDa is an actively maintained, searchable database of information relevant to computer security. Its particular focus is vulnerability information and contextual material. Being the only such service available within IBM, it represents one of the major contributions of this work. It has evolved to a highly regarded and used repository for security-related information within IBM. Moreover VulDa, which started its operation in 1996, was among the first to offer security-relevant information structured in a manner [DacAle99] as can be now found on public sites such as SecurityFocus [SecFoc] since 1999, or at NIST's ICAT metabase [ICAT] since 1998.

In addition of being important to IBM, VulDa significantly supported numerous concepts developed in this work by providing the necessary profound knowledge of practical security issues. Moreover we were able to use it for categorizing a large number of attacks, which yielded results that were highly valuable to the IDS analysis approach developed in this work.

Our VulDa effort, which lasted over a period of four years (1998-2001), included its design, population, and operation. In addition it included being the contact point for security- and operation-related issues of the entire VulDa user community of more than 500 registered IBM-internal users. It was this direct access to the internals of VulDa that enabled the iterative process that resulted in the categorization of more than 350 attacks, and the concept of IDS scopes, all of which we describe in detail in the next chapter. It is worth mentioning that VulDa now offers continuously updated attack categorization statistics to its users.

In the following we provide background information on the motivation for and the history of VulDa, followed by descriptions of its structure, operational processes, the concept of vulnerability descriptions, and user interfaces. Over time this database has evolved to a rather complex system whose detailed description lies beyond the scope of this document. We therefore focus on the core functionality of the database and on the data and functionality used in the context of the work described here.

Note that the following sections are not a prerequisite for understanding the remainder of this work, but they add valuable background information and describe a significant effort made in the context of this work.

A.1 Motivation and history

Back in 1996, members of the IBM Zurich Research Laboratory started, for internal purposes, to build up a structured repository of attacks. At that point it was sometimes still difficult to get hold of information about attacks because some of them were not disclosed to the public. Collecting information about newly discovered vulnerabilities and attacks was, however, a must for the ongoing research efforts in the ID research field. The team felt that a systematic and uniform description of the vulnerabilities and attacks

found was required to structure the collected data. This led to the introduction of a first version of the so-called *vulnerability description files*.

From 1998 onwards, an increasing number of security professionals were interested in accessing the database. This trend resulted in more that 500 registered accounts by the end of 2000. This increased interest led to various improvements of the system, which finally enabled the transfer of VulDa from IBM Research to IBM's Managed Security Services (MSS) organization.

A.2 VulDa structure

The structure of VulDa has evolved and has been improved significantly over time. However, the core of VulDa has remained the same [DacAle99]. The information available on VulDa is two-fold. First, VulDa contains a large repository of documents collected from sources known to provide security-relevant material, e.g., mailing lists such as Bugtraq [SecFoc], newsgroups, various web and ftp sites such as CERT/CC [CERT], SecurityFocus [SecFoc], SANS[21] [SANS] and NIAP[22] [NIAP97]. Second, so-called vulnerability descriptions provide highly structured information about vulnerabilities and their corresponding attacks. These vulnerability descriptions are tightly linked to the documents that are collected automatically by means of generic references.

The population of the database is automated to a high degree and is mostly achieved by unattended batch processes that actively gather (e.g., newsgroups, world wide web, ftp sites etc.), archive, convert and index data. In addition data is also gathered passively. For instance, VulDa maintains archives of more than 100 mailing lists.

Figure 44 shows the data flow within VulDa. For maintenance, control, state tracking, indexing, and searching numerous processes have been implemented that make use of freely available tools such as Perl [Perl87], GDBM (GNU database manager), or MySQL [MySql]. The database can be accessed from within IBM by means of a secured webserver that offers various ways for searching the various document categories.

As shown in Figure 44, much of the data is being collected from the Internet in an automated fashion. This data is then stored in a filesystem that in time has grown to over 1,500,000 documents. All newly collected data is then filtered according to rules, which we have defined based on experience. The goal of this filtering is to isolate important information on new attacks and vulnerabilities, which can then be used to create vulnerability descriptions.

We admit that it would have been preferable to store such a vast amount of information using a database server. However, the data was stored in filesystem for historical reasons, and the migration to a database

---

[21] SANS: "System Administration, Networking, and Security," an institute focusing on cooperative research and education; founded in 1989.

[22] NIAP: "National Information Assurance Partnership," part of the National (US) Institute of Standards and Technology (NIST).

server was not done owing to the limited (human) resources available. Instead the focus was the content of the database.

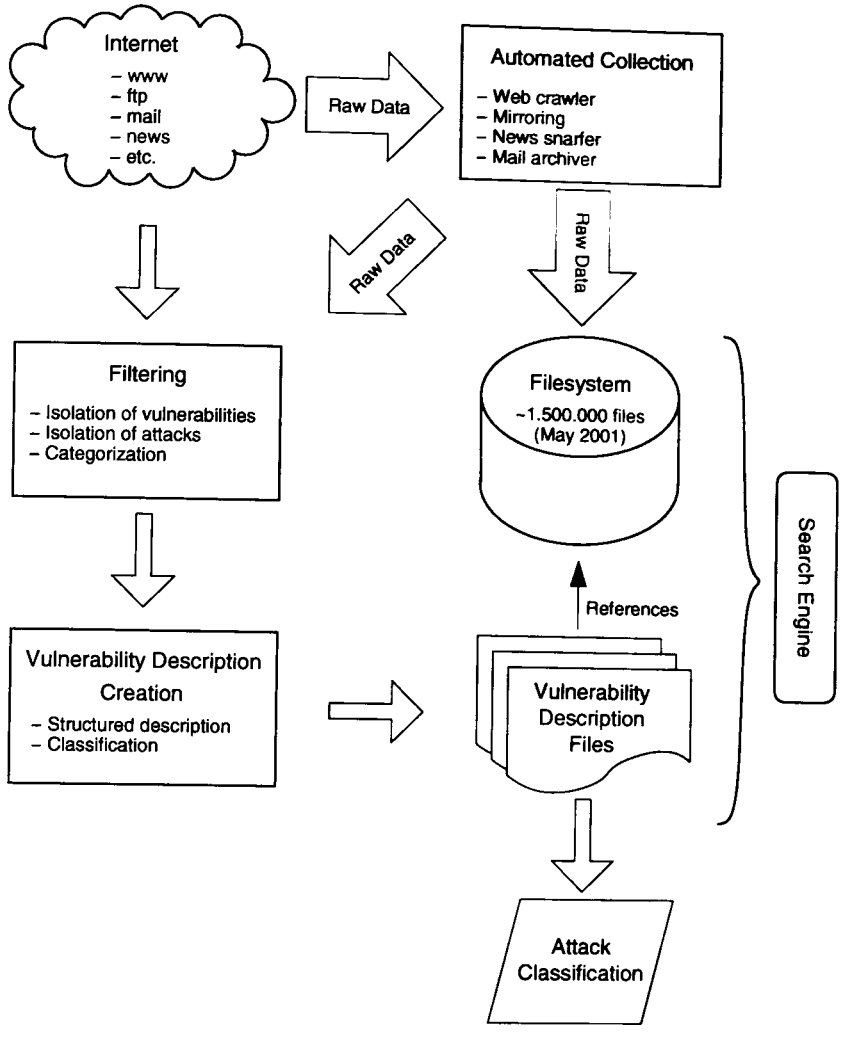


Figure 44—Data flow in VulDa

A.3 Vulnerability descriptions

The vulnerability descriptions represent an important asset of VulDa. They are generally used as the primary entry point for searches in VulDa. The creation of vulnerability descriptions requires both a good knowledge of networking and computing systems in general and an in-depth understanding of computer security, including the peculiarities of the security community. The latter is necessary because, for instance, often only wrong or incomplete information is made available, which has to be identified as such.

Vulnerability descriptions are stored in so-called vulnerability description files. These are composed of sections that contain attribute-value pairs, which are used to describe the properties of vulnerabilities and their corresponding attacks. Owing to these sections, VulDa's vulnerability descriptions are well

structured and thereby clearly separate vulnerabilities and attacks. Generally speaking the vulnerability descriptions are capable of representing a super-set of the information provided by other more recent efforts such as ICAT [ICAT] or the Bugtraq ID pursued by SecurityFocus [SecFoc].

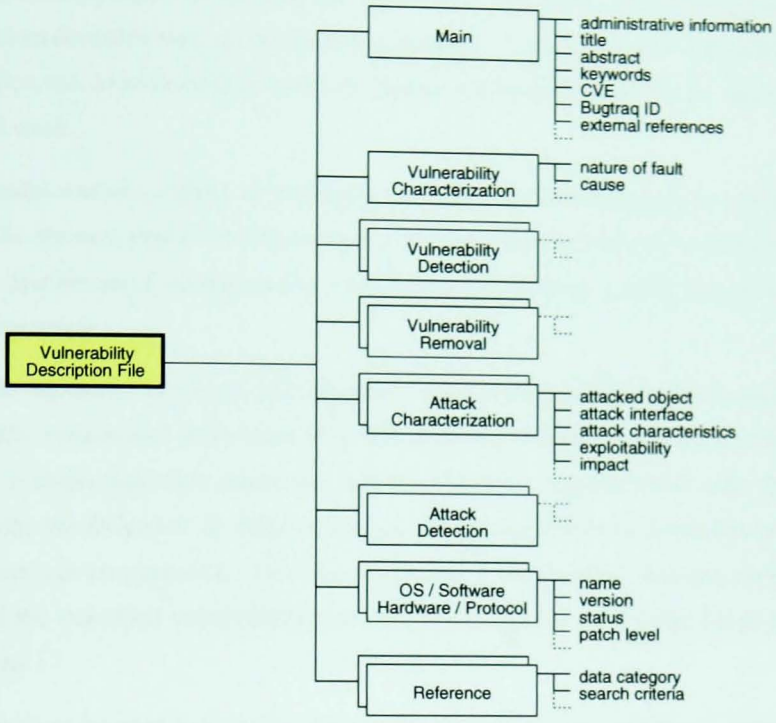


Figure 45—Overview of the vulnerability description structure

In the following we briefly discuss the purpose of the various sections shown in Figure 45. We only highlight details that are either believed to be of general interest or are used in the context of the work described here.

The *main* section of vulnerability description files contains generic information about the document. Besides information used for administrative purposes, it also contains the document title, an abstract, keywords, external references, and last but not least, CVE identifiers and Bugtraq IDs (see also section 2.2.5.1).

In the section *vulnerability characterization,* the vulnerability is classified according to numerous criteria such as the cause of the fault (e.g., design fault, implementation fault etc.). Furthermore, the vulnerability is classified according to well-known fault characteristics such as insufficient input validation, privilege abuse etc. (See also Section 2.2.5). Among other criteria this section also indicates the system component in which the fault is located.

The sections *vulnerability detection* and *vulnerability removal* will not be explained in detail here. In summary the vulnerability detection section includes information on testing tools, e.g., a scanning tool, and how these tools would report the vulnerability. The vulnerability removal section describes the various ways to remove the vulnerability, e.g., disabling of a service, reconfiguration, etc. Vulnerability

descriptions may include several of these sections because several different ways to detect a vulnerability or to remove a vulnerability might exist.

Each of the *attack characterization* sections characterizes one of the ways the previously described vulnerability can be exploited, i.e., the way the fault can be activated. The attributes of these sections describe attack characteristics such as the immediate impact of the attack, the exploitability (remote or local), prerequisites, etc. In addition these sections support attributes required for the attack categorization developed in this work.

The *attack detection* sections consist of properties that allow us specify how one can detect an attack using a given IDS. Several attack detection sections are supported because the vulnerability description may contain the description of several attacks, and because the various existing IDSs may detect attacks in various different ways.

The sections *OS, software, hardware* and *protocol* have an almost identical structure. Each section describes a specific version and patch-level of an OS, software, hardware, or protocol. In addition these sections contain a status field that marks the described as being vulnerable or safe. Thus it becomes possible to describe the difference in terms of version and patches between vulnerable and safe versions of a product or protocol very precisely. This also means that it is possible to describe the fact that a given patch introduced the described vulnerability and that the installation of another patch will remove the vulnerability again.

The *reference* sections are used to link the vulnerability description with additional information sources. It is possible to provide references to every document found in VulDa—including other vulnerability descriptions, exploits (attack scripts), advisories, RFCs etc. It is important to note, and completely natural, that in general information about a given vulnerability is discovered and published gradually. Furthermore it seems obvious that in practice the frequent updating of references in several hundred or thousand vulnerability descriptions is infeasible. This has led us to the introduction of so-called *dynamic references,* which proved to be very efficient both in terms of maintenance and usability. A dynamic reference consists of a search pattern that is resolved at runtime. We thereby make use of the possibility to search VulDa on a per-category basis, which allows us to refer to an entire series of subsequent documents, e.g., mailing list threads or a series of advisories, very efficiently and in an always up-to-date manner. Experience has shown that if the search patterns, i.e., the dynamic references, were formulated with sufficient care in the first place, the number of non-relevant document references generated can subsequently be kept to a negligible minimum. To further extend the expressiveness of the references generated it is possible to rank them.

A.4 User interfaces provided

Over the years the VulDa database has grown to a large repository of security-relevant documents that are organized by categories. The underlying technology extracts the text from all documents (including PostScript, PDF etc.) and creates a searchable index per category of documents. Based on these indices

VulDa offers a flexible keyword search facility that allows the user to search for documents by categories or to browse vulnerability descriptions according to various criteria such as the operating system affected.

A.4.1 Attack categorization

The attack categorization, which relies on dedicated attributes that we included in the attack characterization section of the vulnerability descriptions, is probably the most directly visible contribution that this work has made to VulDa. The statistical data derived from this categorization is used to select activity categories for the analysis of IDSs. The categorization scheme and the manner its results are used are explained at large in Chapter 4. Additional statistical data is provided in Appendix B.

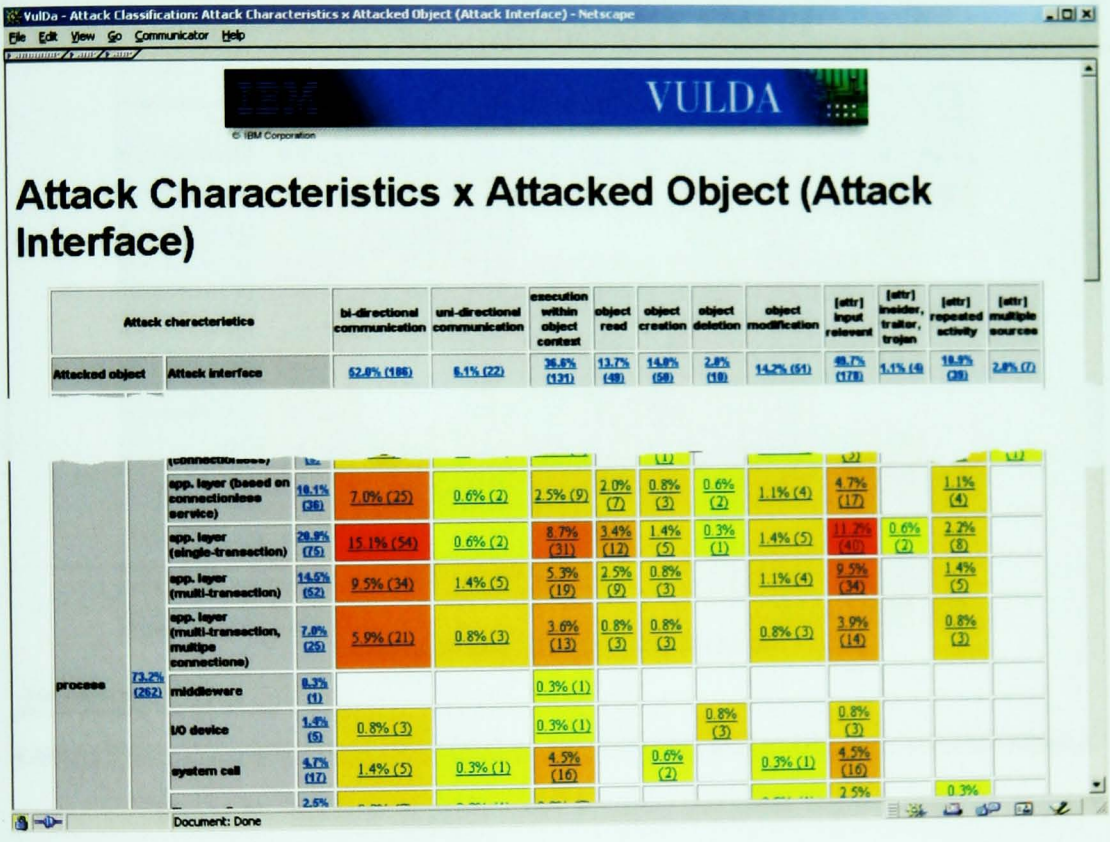


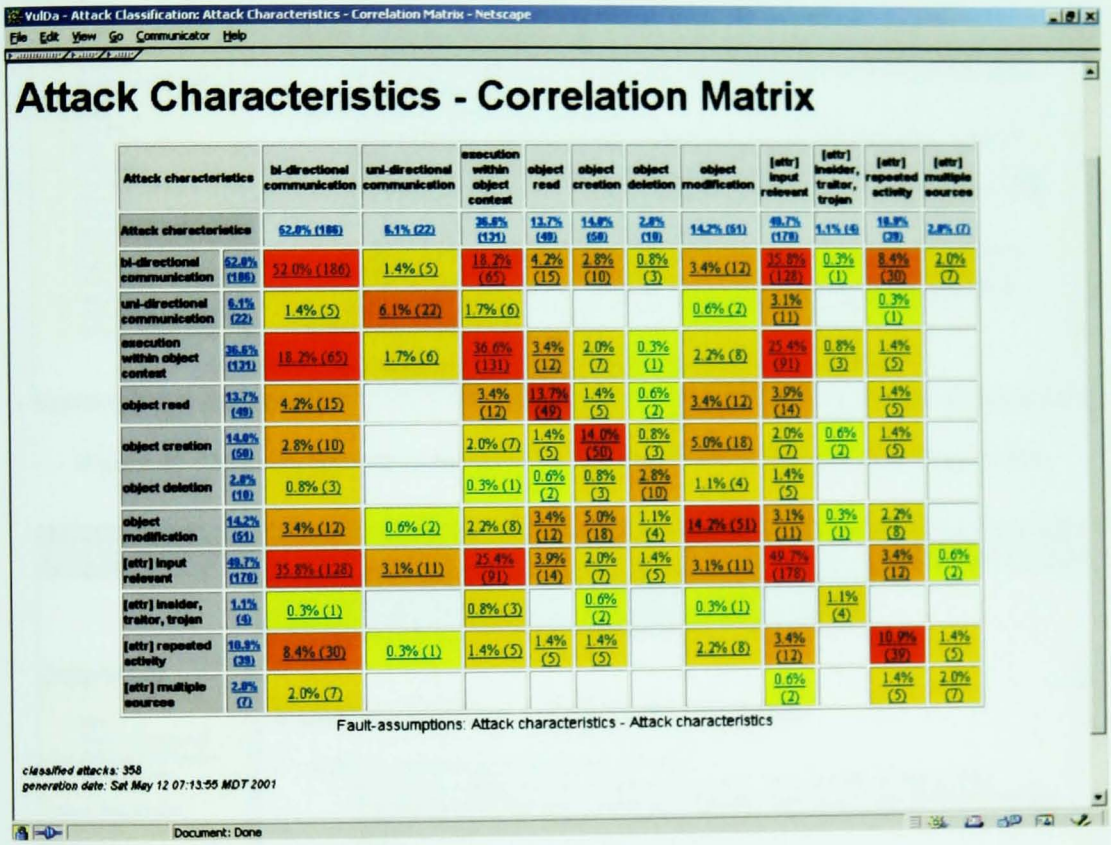
Figure 46—Statistics derived from attack categorization superposing attacked object, attack interface and attack characteristics

Besides the statistics discussed in Section 4.3, numerous HTML tables as shown in Figure 46 and Figure 47 are created, i.e., updated, on a daily basis. The tables not only provide an overview of the most frequent categories of attacks, but also allow us to obtain the list of vulnerabilities belonging to a given category by clicking on the corresponding table field. This functionality simplifies the task of describing attack classes that belong to a given category by providing us with attack examples.

Figure 46 shows a table that combines the static (rows of the table) and dynamic activity characteristics. Both are described in detail in Section 4.2. Note that every attack may qualify for several static and dynamic activity characteristics. As a consequence vulnerability descriptions may qualify for several

category fields in the table, which is a completely valid situation. (See also 2.2.1 and the citation of Axelsson [Axelss00], p. 17)

Figure 47 shows a different example of a table that is generated based on the categorized attacks. Here one can find the various ways the dynamic activity characteristics (labeled "attack characteristics" in the figure) are combined within one attack. Note that this table reflects only combinations of two characteristics per field. This means that attack categories that combine three or more attack characteristics cannot be identified using this table. It also means that attacks combining several characteristics appear more than once in this table.



Attack characteristics	bi-directional communication	uni-directional communication	execution within object context	object read	object creation	object deletion	object modification	[attr] input relevant	[attr] insider, traitor, trojan	[attr] repeated activity	[attr] multiple sources
Attack characteristics	52.8% (186)	6.1% (22)	36.8% (131)	13.7% (49)	14.0% (50)	2.8% (10)	14.2% (51)	49.7% (178)	1.1% (4)	10.9% (39)	2.8% (7)
bi-directional communication 52.8% (186)	52.8% (186)	1.4% (5)	18.2% (65)	4.2% (15)	2.8% (10)	0.8% (3)	3.4% (12)	35.8% (128)	0.3% (1)	8.4% (30)	2.0% (7)
uni-directional communication 6.1% (22)	1.4% (5)	6.1% (22)	1.7% (6)				0.6% (2)	3.1% (11)		0.3% (1)	
execution within object context 36.8% (131)	18.2% (65)	1.7% (6)	36.8% (131)	3.4% (12)	2.0% (7)	0.3% (1)	2.2% (8)	25.4% (91)	0.8% (3)	1.4% (5)	
object read 13.7% (49)	4.2% (15)		3.4% (12)	13.7% (49)	1.4% (5)	0.6% (2)	3.4% (12)	3.9% (14)		1.4% (5)	
object creation 14.0% (50)	2.8% (10)		2.0% (7)	1.4% (5)	14.0% (50)	0.8% (3)	5.0% (18)	2.0% (7)	0.6% (2)	1.4% (5)	
object deletion 2.8% (10)	0.8% (3)		0.3% (1)	0.6% (2)	0.8% (3)	2.8% (10)	1.1% (4)	1.4% (5)			
object modification 14.2% (51)	3.4% (12)	0.6% (2)	2.2% (8)	3.4% (12)	5.0% (18)	1.1% (4)	14.2% (51)	3.1% (11)	0.3% (1)	2.2% (8)	
[attr] input relevant 49.7% (178)	35.8% (128)	3.1% (11)	25.4% (91)	3.9% (14)	2.0% (7)	1.4% (5)	3.1% (11)	49.7% (178)		3.4% (12)	0.6% (2)
[attr] insider, traitor, trojan 1.1% (4)	0.3% (1)		0.8% (3)		0.6% (2)		0.3% (1)		1.1% (4)		
[attr] repeated activity 10.9% (39)	8.4% (30)	0.3% (1)	1.4% (5)	1.4% (5)	1.4% (5)		2.2% (8)	3.4% (12)		10.9% (39)	1.4% (5)
[attr] multiple sources 2.8% (7)	2.0% (7)							0.6% (2)		1.4% (5)	2.0% (7)

Figure 47—Statistics of concurrent occurrences of attack characteristics

A.4.2 Vulnerability browser

The vulnerability browser, also called "vulnerability overview," provides a browser-style interface that enables a user to search easily for categories of vulnerabilities and attacks. The interface is implemented based on VulDa's search engine. As shown in Figure 48 and Figure 49, the user may choose a category of vulnerability descriptions in the left column of the browser window and have them displayed in the main portion of the browser window.

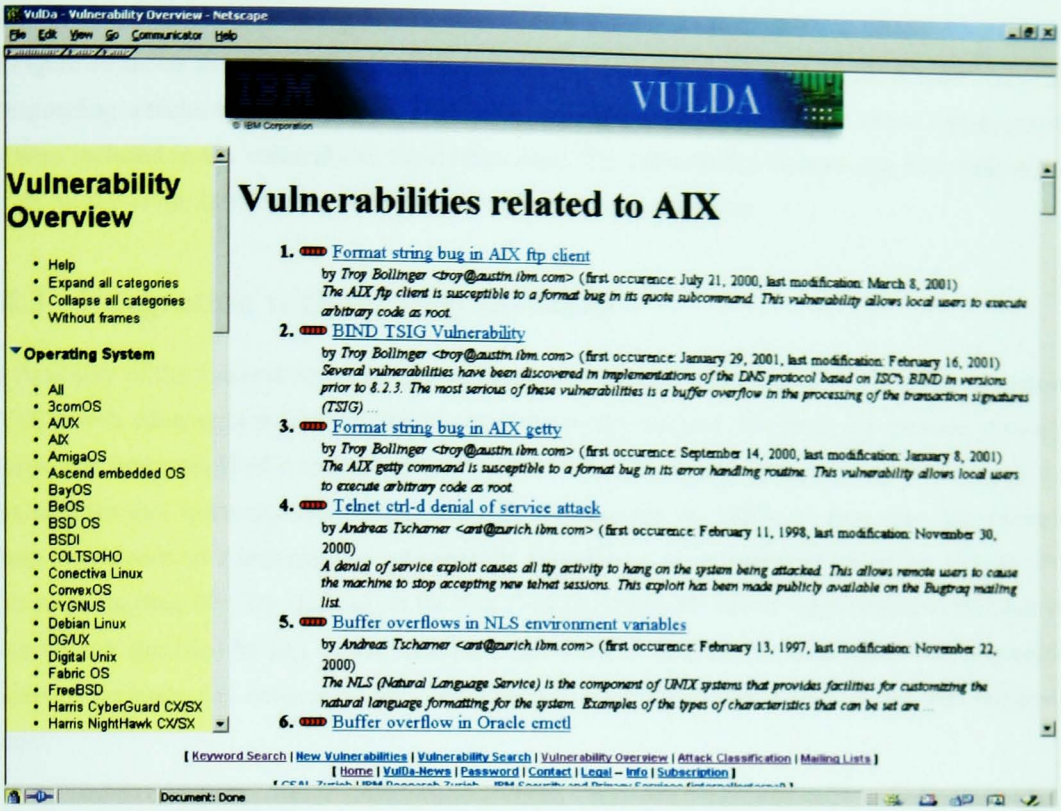


Figure 48—Vulnerability browser showing vulnerabilities of the AIX operating system

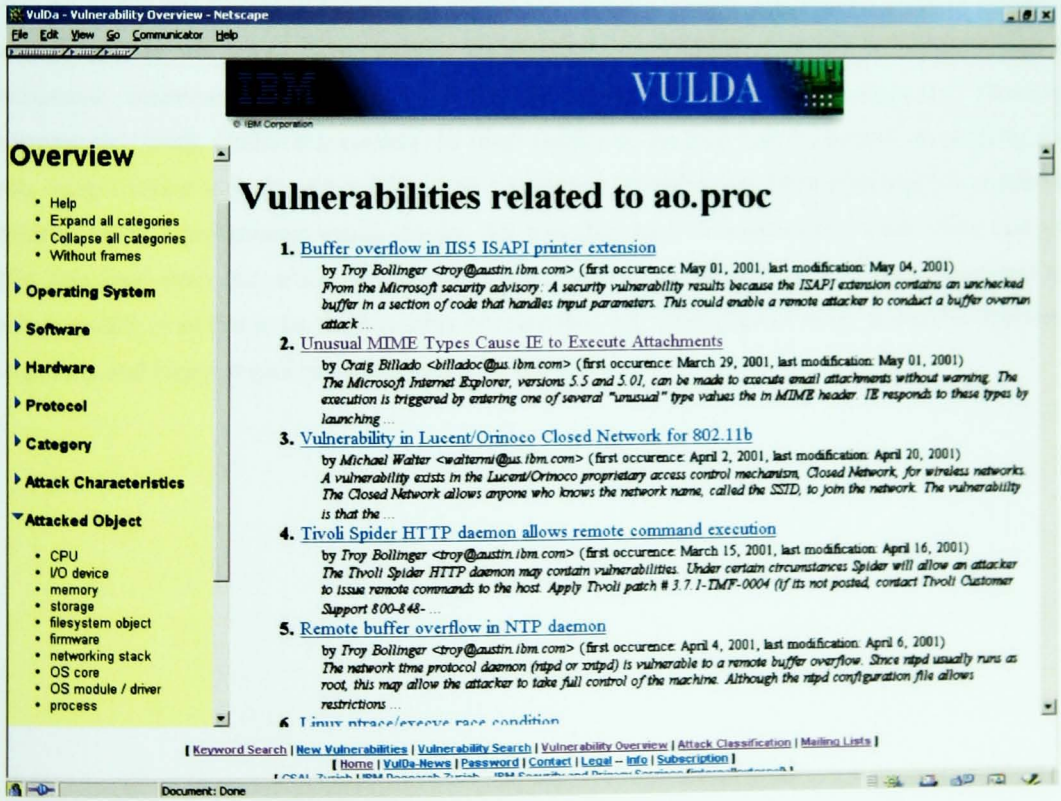


Figure 49—Vulnerability browser showing vulnerabilities in which at least one of the corresponding attacks affects a process

Figure 48 shows an example listing of all vulnerabilities known to affect IBM's AIX operating system, and Figure 49 shows an example listing of all vulnerabilities for which it is known that at least one of the corresponding attacks affects processes. The latter example exploits one of the attack categorization attributes included in the vulnerability description files. The vulnerability browser can be configured to operate on any of the attributes supported by the vulnerability descriptions.

A.4.3 Integration with security software

The flexibility of the vulnerability description files and the search engine enables the efficient integration of VulDa with other systems such as IBM's network security scanner. We have, for instance, created a package that integrates IBM's security scanner with VulDa. The scanner generates an HTML report that contains links that indirectly refer to vulnerability descriptions on VulDa. A link may, for example, contain a reference to a tool-specific vulnerability identifier or to vulnerability identifier such as CVE numbers or Bugtraq IDs. By clicking on the link, VulDa returns the list of vulnerabilities matching the search criteria specified by any of the identifiers mentioned. Using this level of indirection it becomes possible to maintain any external system and VulDa in a highly independent, but still well integrated manner.

A.5 Conclusion

Over time VulDa has proved to be a highly useful and valuable system with respect to many aspects of this work and to the entire IBM security community. Nevertheless, one has to mention that its development, maintenance, and population proved to be enormously time-consuming. However, concerning this work, VulDa represented the ideal facility to develop and implement the activity, i.e., attack, categorization scheme, and to educate us on security-related issues. Most importantly I enabled us to select a set of representative attack classes that was then used throughout this work. Note that also VulDa benefited from this work as the results have become an integral part of the system and are therefore readily available to its large number of more than 500 IBM-internal users. It thereby represents an important and highly regarded contribution of this work.

Appendix B Statistical results derived from the attack categorization

In Chapter 4 we have developed a classification scheme for activities, which is geared at the characterization of activities that are either malicious, i.e., attacks, or that are sufficiently attack-similar to be confused by an IDS with an attack. In Section 4.3 we have then explained how this classification scheme was used to classify 358 attacks taken from IBM's security database VulDa, which is described in Appendix A. Also in Section 4.3 we provide some more detailed insights gained from this classification effort.

As a general comment on the following figures, note that we provide histograms rather than pie charts for most of the data. This is done because the classification scheme permits attacks to be characterized by multiple activity interface and dynamic activity characteristics. Solely the affected objects are defined such that only one characteristics is permitted per attack classified. In this case the sum of all affected objects adds up to 100%, which is why we provide a pie chart in B.3.

B.1 Dynamic fault characteristics

As mentioned in Section 4.2.3, the dynamic activity characteristics allow the combination of several dynamic activity characteristics to describe an attack, i.e., we use the potential set created based on all the dynamic activity characteristics defined.

In Figure 50 we consider the communication model combined with the method invocation model only. By considering the communication characteristics (uni- and bi-directional) it is apparent that more than half of the attacks classified involve some means of communication.

Note that the histograms shown in Figure 50 and Figure 51 are simplifications of the underlying data. We have been able to identify 42 different combinations of dynamic activity characteristics in the data used for Figure 50, and 86 combinations in the data used for Figure 51. Figure 50 shows only 14 combinations of dynamic activity characteristics, whereas the remaining combinations are collected in the group called *other combinations*. Figure 51 shows 21 different combinations.

Figure 51 incorporates also the dynamic activity attributes (see Section 4.2.3.3). This enables us to identify some very frequent types of attacks such as buffer overflow or special-character attacks against server processes. These attacks typically involve *bi-directional communication* and *execution within object context*. In addition, the *input* provided is of high importance because it contains the buffer overflow data or the special characters. One can easily identify these attacks in the figure by considering the *input relevant, exec. within object context* portion of the second bar (49 attacks). In a similar way one can identify any kind of potential buffer overflow or special-character attack by considering the first bar (taking a look at the raw data one finds 99 attacks that combine the attack characteristics "exec. within

object context" and the attribute "input data relevant"). Additional combinations of characteristics exist that involve other characteristics in addition to those just mentioned. For instance, the special-character attack against a webserver that reveals the password file involves the *object read* characteristics in addition.

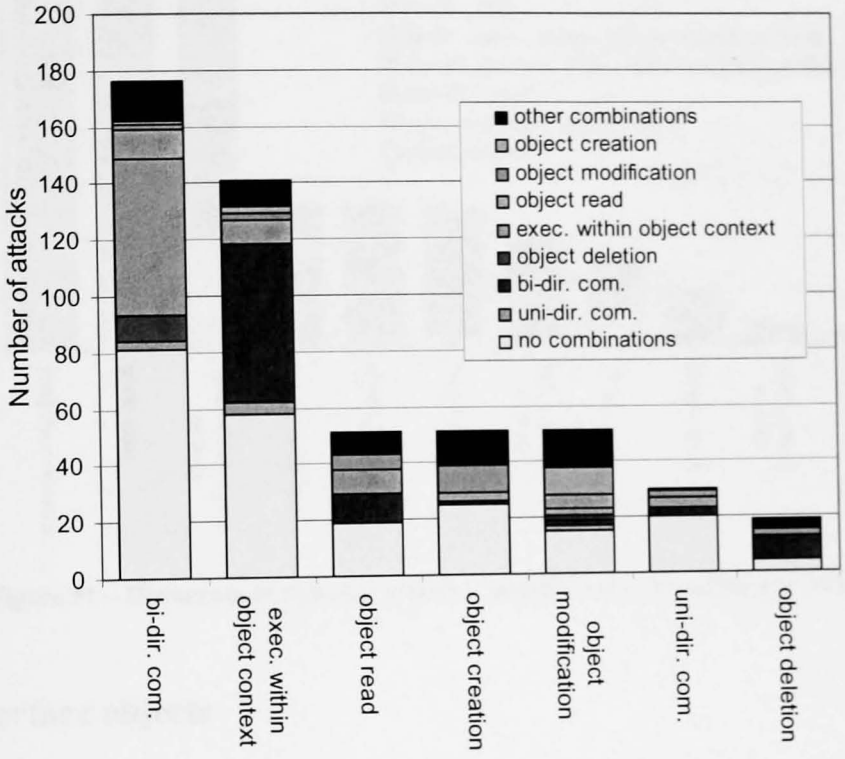


Figure 50—Histogram of dynamic activity characteristics, excluding attributes

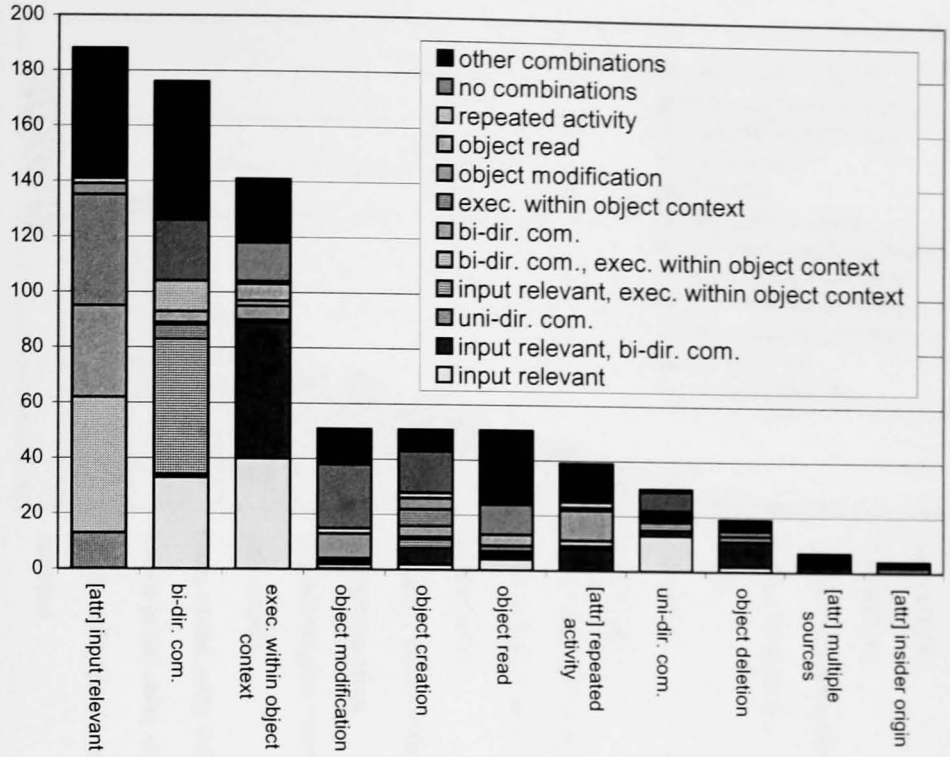


Figure 51—Histogram of dynamic activity characteristics, including attributes

B.2 Interface objects

The histogram shown in Figure 52 shows the distribution of interface objects used to stage attacks. The total size of the bars shows that *processes* and *application layer protocols* are the most frequently used interface objects. Taking a closer look at the respective bars one can further deduce that protocol layers and *processes* are rarely combined to attack another object. This is not surprising as it highlights the fact that remote attacks generally involve the protocol stack, whereas local attacks typically involve *processes* as attack interface. This observation allows us to distinguish clearly between locally and remotely executed attacks.

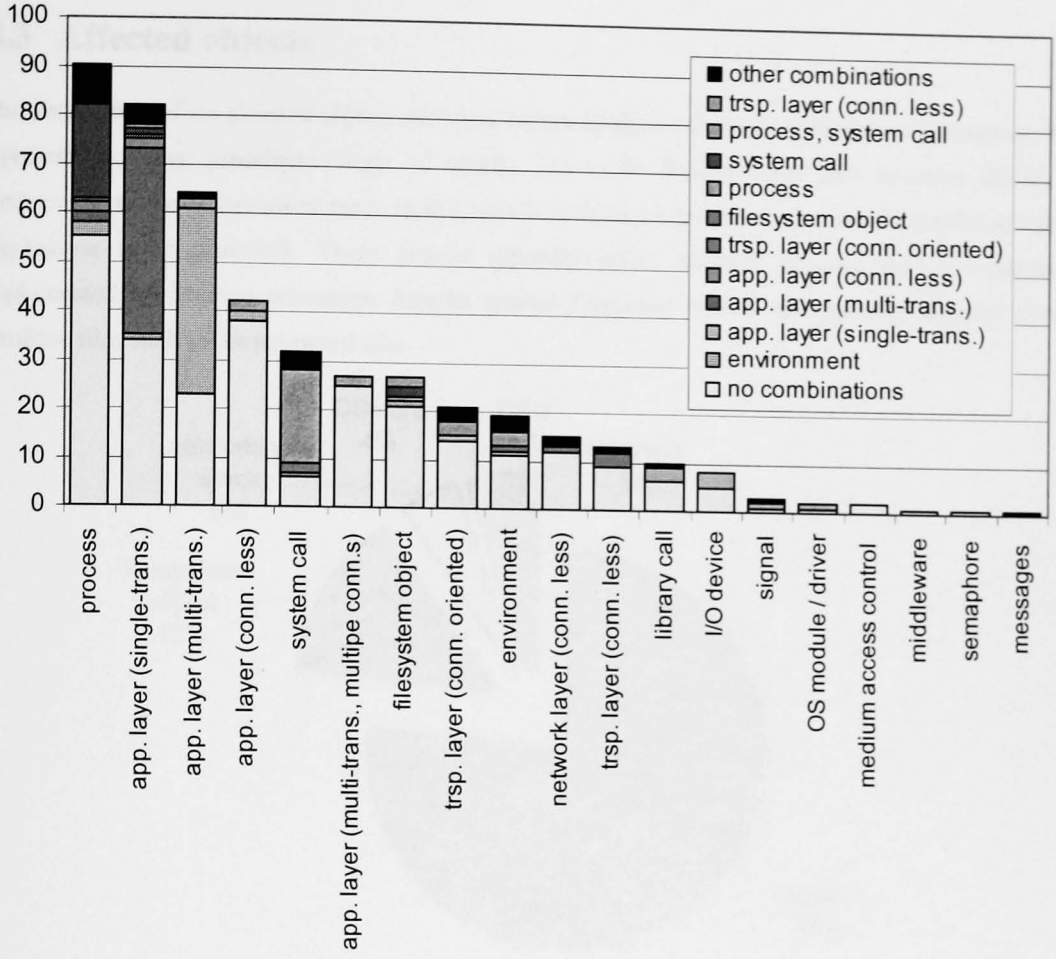


Figure 52—Histogram of interface objects

Moreover, it is apparent that the *filesystem objects* and the *environment* are not used in combination with communication protocol layers to attack other objects (see bars labeled *"environment"* and *"filesystem object"* in Figure 52). This is a reasonable result, because for remote attacks *filesystem objects* typically play the role of the affected objects, e.g., reading of the password file. On the other hand they may serve as interface objects for local attacks that target *processes* for instance. For the *environment,* the findings are similar. Although the *environment* was involved in some remote attacks involving the telnet protocol, remote attacks involving the environment typically are very rare, as also revealed by Figure 52.

B.3 Affected objects

The distribution of the affected objects shown in Figure 53 shows the clear dominance of *processes*. They represent the most prominent target of attacks. As to be demonstrated later in more detail, this corresponds to the observations made in B.2, which showed a large number of attacks involving various *application layer protocols*. These attacks typically affect network services that are commonly implemented by daemon *processes*. Attacks against *filesystem objects* are typically targeted towards sensitive files such as the password file.

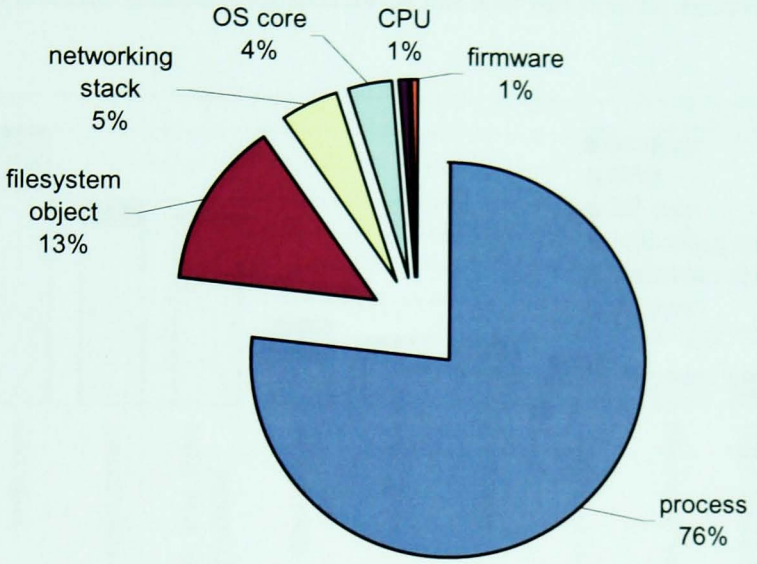


Figure 53—Distribution of affected objects

Note that we are not classifying the impact of attacks here. The impact of an attack is described by a system state corresponding to an error and/or failure state, which depends on the success of an attack. However, the success of an attack mostly depends on the presence of the corresponding vulnerability and its characteristics. For instance, an attack that is staged against a *process* or a *filesystem object* may result, in both cases, in a shell being provided to the attacker. Thus, the affected object can be a process or the filesystem respectively. However, the impact of the successful attack is merely determined by the vulnerability rather than by the attack.

B.4 Dynamic attack characteristics with affected objects

When considering the eight most frequent combinations of dynamic fault characteristics, we obtain a similar picture as in Figure 50. However, it is apparent that the combination of the activity characteristics *bi-directional communication* and *execution within object context* (second bar from the left in Figure 50) are even more frequent than the sole *execution within object context* (third bar from the left in Figure 50). This has already been observed in Section B.1, and demonstrates the importance of the attacks against (server) *processes*. This can also be verified byt the fact that almost all the attacks falling into to this category target *processes* (especially apparent for the first three bars from the left in Figure 50).

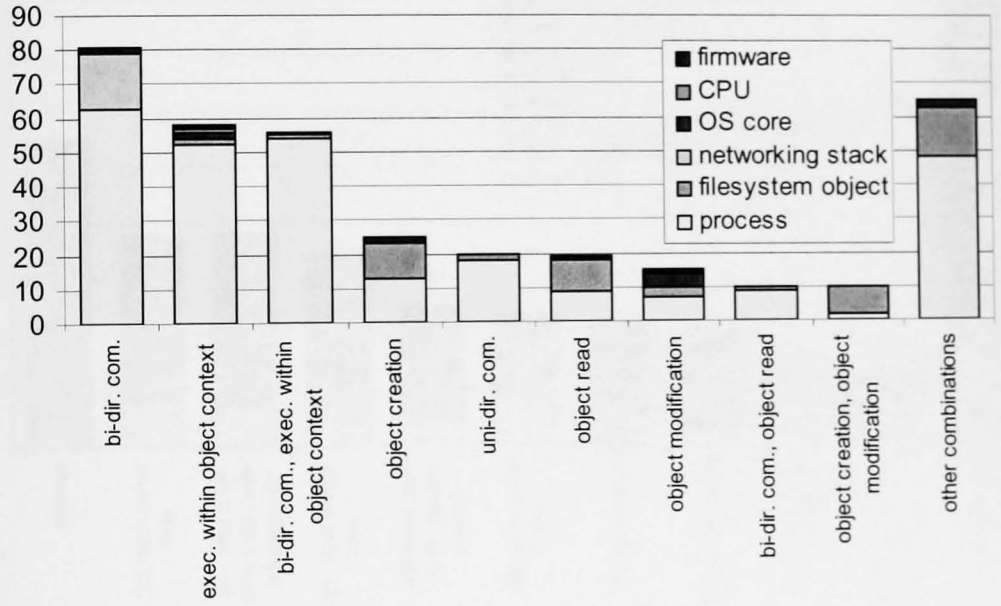


Figure 54—Histogram of dynamic activity characteristics with affected objects

Moreover we can identify the class of attacks that uses a communication mean to attack the *networking stack* of a system. Most of those attacks are denial-of-service (DoS) attacks, and attempt to crash the entire system by sending some malformed PDUs to the host. The malformed PDU may then force the *networking stack* into an undefined (error) state, a failure state. Finally we can identify an important class of attacks that affect the *filesystem* by means of *object creation, modification* or *reading*. Those attacks are typically staged on the local host.

B.5 Interface objects with dynamic attack characteristics

The distribution of the combinations of interface objects with dynamic activity characteristics (shown in Figure 55) is less bursty than the one found when considering the combination of affected objects and dynamic activity characteristics (see Section B.4). The resulting sub-classes are distributed more evenly. At a first glance the most important class by far is that consisting of *processes* (see first column of Figure 55). However, a detailed analysis shows that the classes involving *application layer protocols* sum up to an even more significant number.

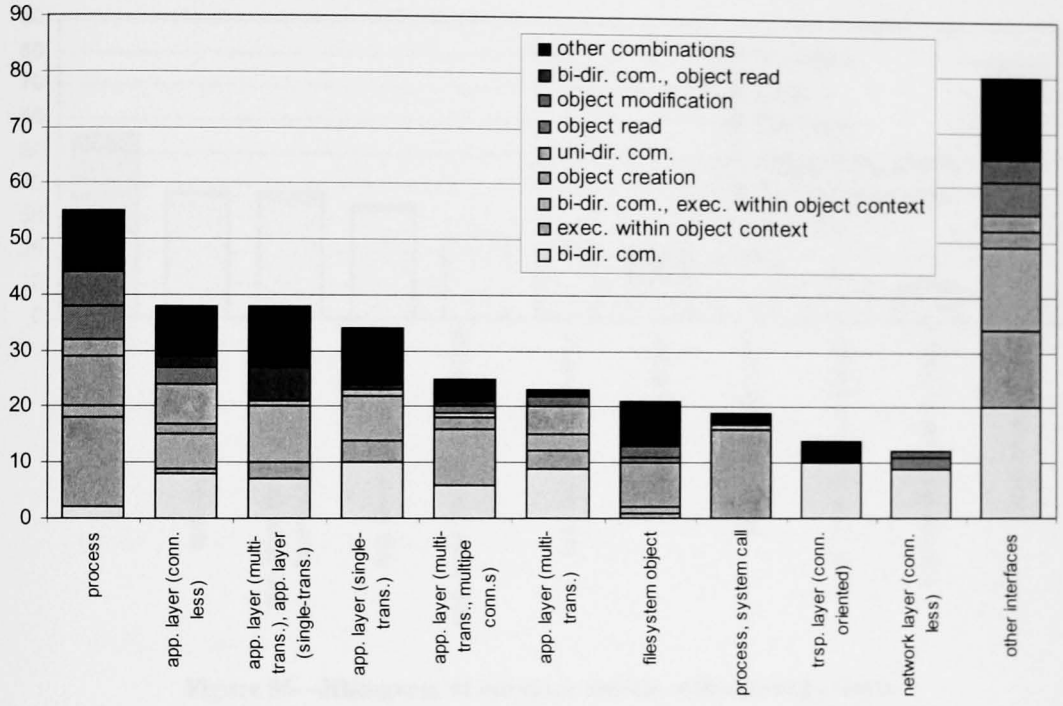


Figure 55—Histogram of interface objects with dynamic activity characteristics

Furthermore we can identify the class of remote buffer overflow and special-character attacks by considering the class of the combined dynamic activity characteristics *bi-directional communication* and *execution within object context* (appears in almost all columns).

B.6 Interface objects with affected objects

When considering the relation between affected objects and interface objects we can identify the interfaces most frequently used to attack a given object. Considering Figure 56, we recognize the dominance of *processes* as attack target objects and as attack interface objects. In addition we can identify *filesystem objects* to be attacked merely using a *process* as interface. This makes sense because *processes* are the primary objects making use of *filesystem objects*. However, even more frequently *processes* are attacked using some *application layer protocol* indicating attacks against daemon processes, i.e., services.

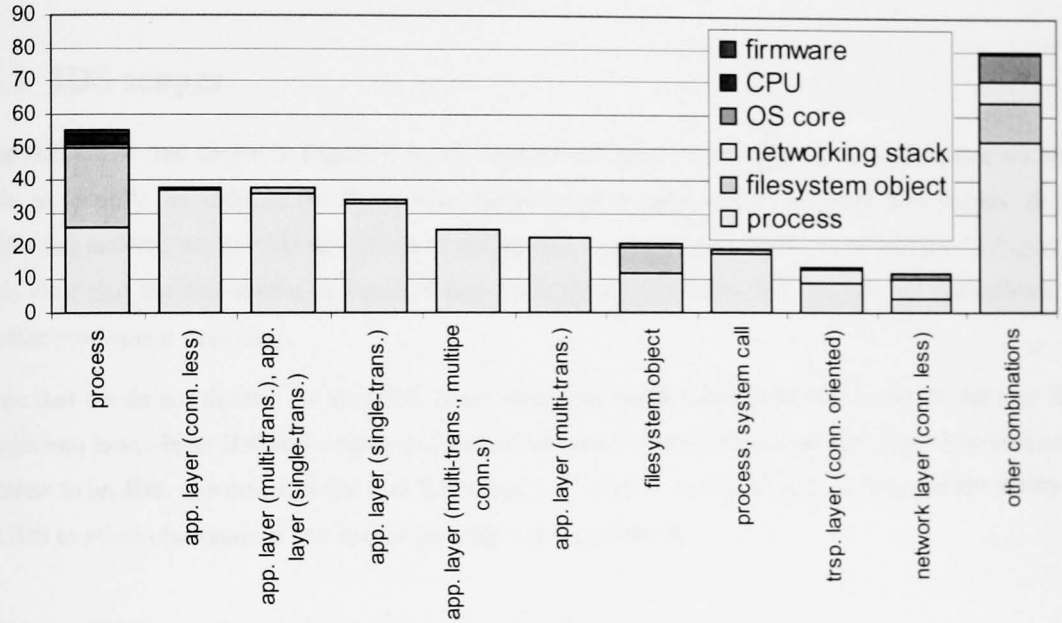


Figure 56—Histogram of interface objects with affected objects

In addition, one can observe that the *CPU* is attacked by *processes* only, which can be explained by the fact that *CPUs* are generally attacked by exercising some malicious, possibly invalid, command sequences. In most cases these attacks result in a denial of service by crashing the entire system. Another observation one can make is the fact that the *networking stack* is generally attacked by *transport* and *network layer protocols*. This is similar to the observation we made above when identifying the processes as a popular attack target, just at a lower level in terms of the protocol stack.

Appendix C Example IDS scopes and their use

In this appendix we provide additional information on IDS scopes and their use in the context of this work and specifically the RIDAX implementation. In a first section we provide definitions of the IDS scope tree elements introduced in Section 4.1.1. In the second section we provide the definitions for the IDS scope attributes used to describe functional properties of IDS scopes as introduced in Section 4.1.2. In the last section we provide examples on how the semantics of the various IDS description characteristics is defined using IDS scopes.

C.1 IDS scopes

The IDS scope tree shown in Figure 9, p. 31, contains all the relevant higher-level IDS scopes we were able to identify. In addition the figure also shows some examples of lower-level IDS scopes. In the following sections we provide definitions of all the IDS scopes as used in this work and shown Figure 9. It is clear that the tree shown in Figure 9 may be extended if needed, for example, by the addition of further protocols if necessary.

Note that we do not discuss the *user IDS scope* in greater detail because we do not divide the user IDS scope into lower-level IDS sub-scopes, as there are no user components that would seem to be of special interest to an IDS. Nevertheless the user IDS scope is needed as it may be used to describe the ability of an IDS to relate observations to a user or possibly a group of users.

C.1.1 IDS scopes related to networking

As a first step we consider the *networking IDS scopes* used to describe networking-related IDS capabilities. For further details we refer to Tanenbaum [Tanenb96]. Note that the protocols listed below are mostly examples illustrating the conceptual foundation of the IDS scope tree.

Table 35—Networking-related IDS scopes—physical layer

Physical layer	The physical layer of communication systems is not of importance for current IDSs. However, as technology evolves, this aspect may become relevant some day.

Table 36—Networking-related IDS scopes—link layer

Link layer	The link layer is split into the less known LLC layer (logical link control) and the MAC layer (medium access control).
LLC	*The logical link control layer used on today's mostly Ethernet-based networks is basically empty. In theory the LLC may offer the service of reliable communication, which is hardly used. The LLC has been defined in IEEE 802.2. See also Tanenbaum [Tanenb96].*
MAC	*The medium access control layer provides the addressing of entities on the LAN, and defines the way the medium is accessed e.g., Ethernet, which originally used a shared medium. This requires a special method to send data on the media such that possible collisions do not lead to the loss of data. The MAC has been defined in the IEEE 802.3 standard. See also Tanenbaum [Tanenb96].*

Table 37—Networking-related IDS scopes—network layer

Network layer	The network layer generally provides a routable addressing of entities, and may also offer reliable communication, i.e., a connection-based service. However the most commonly used implementation of the network layer used today is IPv4, which offers a datagram service only.
IPv4	*The internet protocol version 4 offers a routable datagram service that does not guarantee the delivery of a datagram nor the order of arrival of datagrams.*
IPv6	*This new, not yet widely deployed, version of the internet protocol provides a much wider address space, and offers improved support for the encryption of data, quality of service, mobility, dynamic routing etc.*
X25	*The X25 is a connection-oriented network layer service that was used by telecom operators for data services before the internet became as popular as it is today.*
ARP	*The address resolution protocol ARP resides at the lower network level, and provides the service of mapping MAC sub-layer addresses to network layer addresses.*

Table 38—Networking-related IDS scopes—transport layer

Transport layer	The transport layer is the layer where reliable communication is implemented in current networks. However, connectionless transport layer services also play an important role in today's networks. The transport layer is generally used to address a specific service on a given host.
TCP	*The transmission control protocol is the most commonly used connection-oriented protocol. The handling of TCP streams is an important but nontrivial issue for network-based IDSs.*
UDP	*The user datagram protocol is similar to TCP but does not provide bi-directional connection-based service. As this protocol is not connection-based it is far easier for IDSs to analyze.*
ICMP	*The internet control message protocol is also a connectionless protocol that was conceived to control the routing at the network layer and to offer simple service such as ping. However, the functionality offered by ICMP may be misused in various ways, which makes it an important protocol to be monitored by IDSs.*

Table 39—Networking-related IDS scopes—middleware

Middleware	The monitoring of middleware protocols is—if done at all—mostly implemented at the application layer level. However, we list it here for future developments.
CORBA	*CORBA is a middleware standard defined by the OMG (object management group) and is widely used in distributed systems.*
DCOM	*DCOM is Microsoft's answer to CORBA. DCOM includes technology such as Microsoft's Active-X controls etc.*

Table 40—Networking-related IDS scopes—application layer

Application layer	The monitoring of application layer protocols is a tedious task for IDSs because there are so many of them. In addition, several versions have been defined over the years for many of them e.g., http v0.9, http v1.0, http v1.1, POP v1, POP v2, POP v3 etc.
DNS	*The domain name service is the most commonly used name-resolution service. This protocol is mostly connectionless.*
SMTP	*The simple mail transfer protocol is the most commonly used protocol to transfer e-mail over the internet.*

FTP	*The file transfer protocol is used to transfer files over the internet. This protocol uses separate control and data connections which makes it a challenge for IDSs to analyze FTP traffic.*
http	*The hypertext transfer protocol is the protocol used by the world wide web (WWW). Several versions of this protocol have been defined and are still in use.*

C.1.2 Host-related IDS scopes

As already mentioned, it is not so straightforward to identify the IDS sub-scopes for the *host IDS scope*. The following IDS scopes have been defined by identifying system components that can be found in computing systems. The list focuses on security-relevant IDS scopes by refining areas of special interest to IDSs.

Table 41—Host-related IDS scopes—devices

Devices	Devices are rarely covered by today's IDSs. However, one might foresee ID work to be done in this area in the future. This area currently does not seem very promising but this may change as new paradigms evolve.
CPU	*The central processing unit is the target of some attacks that attempt to exploit faults present in the CPU's microcode.*
Storage	*Storage devices such as disks, tapes, CD-ROMs etc., are used for persistent storage of large volumes of data.*
I/O	*Input/output devices allow a system to communicate with the outside world. Examples: Network interface cards, serial/parallel line interfaces, keyboards, mouse etc.*
Memory	*The memory (RAM) is commonly used to store data and executable code.*

Table 42—Host-related IDS scopes—firmware

Firmware	The firmware is a low-level piece of code that runs beneath the operating system and that is responsible for managing the hardware of a system. It manages all the devices in a system e.g., power management and bootstrap, the operating system at power-up time. The firmware is not of importance to today's IDSs, but with the ongoing development of the technology [VMware00] that will enable the operation of several independent virtual machines on the same physical system this may change.

Table 43—Host-related IDS scopes—OS core

OS core	The operating system is used to control and to manage the system. Moreover, it provides various services.

Table 44—Host-related IDS scopes—OS modules

OS modules	Operating system modules are commonly used to extend the OS core with device drivers etc. Because the network stack is an OS module of high importance for ID we list it separately.
Network stack	*The network protocol stack is commonly implemented as an OS module for efficiency reasons. The reason why it is listed here separately is that it is a prominent target. If the network stack encounters a failure this often propagates to OS core and may therefore lead to the failure of the entire system. Also it is relatively easy to attack the network stack because it represents—by definition—one of the most important interfaces of the system to the outside world.*

Table 45—Host-related IDS scopes—calls

Calls	Calls represent an important change in the execution path of a process. One can distinguish calls that do not require a context switch from user space to kernel space and calls, i.e., system calls, that require such a switch.
System calls	*System calls are used by a process to interfere with the underlying OS. The sequence of system calls made by a process may be used by an IDS, such as the DaemonWatcher by Wespi et al.* [WeDaDe00, WesDeb99]*, to obtain an indication as to what a process actually does..*
Function calls	*The tracing of function calls is generally not easy because they occur in the user space and do not require an interaction with a central component such as the OS kernel. However, it is conceivable that calls to library functions are logged by the library involved and then used for later analysis by an IDS. An example of such an approach is the work by Kerschbaum, Spafford and Zamboni* [KeSpZa00, SpaZam00, SpaZam00b, Zambon01].

Table 46—Host-related IDS scopes—filesystem objects

Filesystem objects	The filesystem is a vital component of commonly used computing systems. It is used to store security-relevant data such as configuration files, passwords etc. The filesystem may also be used as a general address space. In Unix filesystems, for instance, one can find, in addition to ordinary files, directories and links, also the notion of special files such as named pipes, and sockets or device files.

Table 47—Host-related IDS scopes—IPC

IPC	Inter-process communication is a widely used method to interfere with running processes. It therefore represents a potential interface for an adversary to manipulate the behavior of a running process.
Signal	*Signals are a basic technique to inform processes about a given event. Signals may also force a process to terminate.*
Socket	*Sockets enable local clients to communicate with a local server process.*
FIFO	*FIFO stands for first in—first out. FIFOs are pipes that can be used to feed the output generated by one process to an input stream of another process.*
Shared memory	*Shared memory enables two processes to exchange data very efficiently directly over the memory.*
Messages	*Messages provide a mechanism that enables processes to exchange data in a well-structured way.*
Semaphore	*Semaphores are used for synchronization purposes e.g., to prevent the concurrent access to a resource.*

Table 48—Host-related IDS scopes—middleware

Middleware	As mentioned, certain components of commonly used middleware technology can only be monitored on the host level. This is why it is also listed here.

Table 49—Host-related IDS scopes—environment

Environment	The behavior of a process may be influenced by its environment, making the environment an important attack interface for adversaries.
Variable	*At process creation time, environment variables are copied from the parent process to the newly created child process.*
Registry	*The registry is specific to the family of the Windows operating systems. The registry represents a central repository of configuration information of the entire system. The modification of the registry can, sometimes, influence processes already running.*

Table 50—Host-related IDS scopes—process

Process	A process is the running instance of a program. Any application, tool, service etc., that is run on a system is generally reflected by one or several processes. This makes process the prime target and interface for attacks. Typical server processes are the http (world wide web) and sendmail (mail service) daemons. Typical applications are web browsers or text processors.

C.2 IDS scope attributes

IDS scope attributes are used to describe functional properties of IDS scopes. For the description we first list the high-level IDS scope to which the attributes described consecutively may be applied. The names of the IDS scope attributes are highlighted in italic font. We also provide examples wherever possible. Note that we again focus only on issues relevant to ID.

C.2.1 Networking-related IDS scope attributes

As before we start by discussing the networking scopes first:

Table 51—Networking-related IDS scope attributes—MAC layer

MAC	Medium access control layer, see Table 36.
Fragmentation	*Fragmentation at the MAC layer is generally not done in standard LAN environments. However, in other applications such as satellite communication, MAC layer PDUs may be fragmented.*

Table 52—Networking-related IDS scope attributes—network layer

Network layer	See Table 37.
Fragmentation	*Network layer protocols such as IP may offer the possibility to split PDUs into smaller pieces. This splitting is required whenever the underlying service has an MTU (maximum transmission unit) size that is smaller than the size of the network layer PDU to be transmitted. In order not to miss important data, network-based IDSs should recompose these fragments before they analyze the data. This is not very complicated, but costly in terms of CPU and memory required. Certain IDSs do not reassemble fragments for exactly those reasons. An adversary can fool those IDSs simply by fragmenting the data sent to the target. Examples: IPv4, IPv6 (see also* [Thomas96]*).*
Connection-oriented	*Connection-oriented network layer protocols are mainly used in the telecom world (circuit switching etc.). Examples: X.25, Frame Relay, ATM, DQDB.*
Connectionless	*Network layer protocols used in LAN environments are generally not connection-oriented. Examples: IPv4, IPv6.*

Table 53—Networking-related IDS scope attributes—transport layer

Transport layer	See also the example shown in Section 4.1.2 and Table 38.
Address	*Many transport layer protocols such as TCP or UDP provide the ability to address a service access point on the destination and source entity. In the case of TCP and UDP addressing is done by means of so-called port numbers.*
Connection-oriented	*The analysis of connection-oriented protocols imposes additional costs on the IDS monitoring the connection for suspicious traffic. Connection-oriented protocols may be tricked into splitting the data stream into arbitrary sequences. Such data-chopping can be used by an adversary to prevent the detection of their attacks by IDSs that are not sufficiently able to reconstruct data streams of connections. In general the IDS needs to keep track of the connection's state, data retransmissions etc. Example: TCP.*
Connectionless	*Connectionless protocols have the advantage that they do not impose the overhead of establishing a connection. This also reduces the burden for IDSs when monitoring such traffic. However, PDUs of connectionless protocols can be easily spoofed. Examples: UDP, ICMP.*
Fragmentation	*We are not aware of a connectionless transport layer protocol that would support fragmentation of data. However, as it is conceivable to be implemented, we mention it for the sake of completeness.*

Table 54—Networking-related IDS scope attributes—application layer

Application layer	See Table 40.
Connectionless	*Application layer protocols may be defined on top of a connection-oriented or a connectionless service. As mentioned, connectionless services may be subject to spoofing attacks. That attack naturally propagates to the overlaying application layer protocols. Examples: Domain (DNS), TFTP (trivial file transfer protocol), NTP (network time protocol), SNMP (simple network management protocol).*
Single connection	*Most application layer protocols that are based on a connection-oriented service only require a single connection. This means that an IDS has to monitor only one transport layer connection to analyze the application layer session. Examples: http, SMTP (simple mail transfer protocol), Telnet, SSH (secure shell).*
Multi connection	*Very few application layer protocols require several transport layer connections for their operation. However, the few that do are quite complex to analyze for IDS because it needs to keep track of all the transport layer connections and in addition it has to correlate the observations made across the various connections. Example: FTP (file transfer protocol).*

Multi transaction	*Particularly in the database area e.g., Oracle, DB2, MySQL etc., a clearly defined transaction concept exists. However, most application layer protocols do not have a clearly defined notion of transactions (see also Section 4.1.2).*
Single transaction	*Application protocols that do not support multiple transactions within a session are generally simpler to analyze for an IDS because it needs to keep limited protocol state information only. Examples: http version 0.9 and 1.0.*

C.2.2 Host-related IDS scope attributes

Within the host IDS scopes we have identified far fewer IDS scope attributes than for the networking IDS scopes:

Table 55—Host-related IDS scope attributes—environment

Environment	See Table 49.
Process creation	*Environment elements belonging to this functional scope influence a process at the time of its creation. Examples: Unix and Windows environment variables, Windows registry.*
Running process	*Environment elements may influence processes already running. Example: Windows Registry.*

Table 56—Host-related IDS scope attributes—IPC

IPC	Inter-process communication, see Table 47.
Synchronization	*The IPC object can be used for synchronization purposes. Examples: semaphore, signals, messages.*
Interrupts	*The IPC object causes interrupts of the normal program execution. Examples: FIFO, Signals, Sockets.*
Data exchange	*The IPC object is used to exchange data. Examples: FIFOs, sockets, shared memory, messages.*

Table 57—Host-related IDS scope attributes—filesystem object

Filesystem object	See Table 46.
Static	*The filesystem object is static. Examples: files, links, directories.*
Dynamic	*Filesystem objects are considered to be dynamic if they are used for communication purposes such as IPC or the interaction with devices. Examples: named pipes, sockets, device files.*

C.3 Definition and use of IDS characteristics with respect to IDS scopes

The examples provided in the following are given to illustrate the way the pairs of items and IDS scopes are interpreted. The following tables are therefore by no means meant to cover the entire IDS scope tree.

In some cases several semantic interpretations seem possible. In such cases we generally choose and document the one that seems most suitable for the analysis of IDSs and ID in general.

C.3.1 IDS sensor characteristics

Table 58—IDS sensor characteristics—Objects (see also Table 7)

IDS scope / items	User	Filesystem object	IPC	Device	Network middleware	Host middleware	Process	OS module	Calls	Environment
Name	Denotes the name of a user e.g., login name, real name etc.	Denotes the unique name of an object e.g., path and filename.	IPC objects may be identified by some unique name[23] e.g., signal name.	Denotes a unique device name.	The IDS sensor is able to gather the name of a middleware object from the network[24].	The IDS sensor is able to gather the name of a middleware object on the host.	Denotes the name of the executable used to create the process.	OS modules or drivers can be identified by a name e.g., Linux modules or Windows DLLs (dyn. loadable libraries).	System and function calls can be identified by a name.	An environment object e.g., an environment variable or a registry key may be identified by a unique name.
ID	Denotes the user ID. Example Unix user ID.	Denotes a file identifier e.g., Unix inode.	IPC objects may be identified by some unique ID e.g., signal number.	Denotes a unique device ID	Middleware object ID	Middleware object ID	The process ID identifies the running instance of a program.	N/A	System calls can be identified by an ID. However this generally does not apply to function calls.	N/A

Table 59—IDS sensor characteristics—Object attributes (see also Table 8)

IDS scope / items	User	Filesystem object	IPC	Device
Type	Indicates the role of a user, e.g., the fact that user is an administrative user. May be used to denote a user's role in role-based access control.	Differentiates files, links, directories etc.	Distinguish IPC object types	Differentiates the various types of I/O devices, storage devices etc.

[23] This is not be confused with IPC objects linked to a filesystem object as done on Unix systems.

[24] From a conceptual point of view, network and host middleware objects often do not exist. One usually refers to a middleware object and does not care about where it is located e.g., CORBA. However, when considering implementations of middleware solutions from the perspective of an IDS sensor this difference matters because the sensor may be monitoring object operations on a host or only on the network.

Table 60—IDS sensor characteristics—Arguments (see also Table 9)

IDS scope / items	Call	Application layer
Basic	The arguments provided to a call are available for analysis.	The arguments directly associated with a request are available e.g., the URL of an http request.
Options	N/A	Optional, possibly loosely associated request arguments are available for further analysis. A typical example are the http header fields that follow the http request statement.

Table 61—IDS sensor characteristics—Request (see also Table 10)

IDS scope / items	Call	Application layer	Transport layer
Name	The name of the function or system call made.	The name of the request made e.g., the name of an http request.	N/A
ID	The identifier of the call made.	The identifier of the type of request made.	The identifier of the protocol request, e.g., ICMP echo request.

Table 62—IDS sensor characteristics—Protocol control data (see also Table 11)

IDS scope / items	MAC layer	Network layer	Transport layer
Source / destination ID	There, the source / destination ID denotes the MAC address of the PDU sender / destination.	There, the source / destination ID denotes the network address of the sender / destination, e.g., IP address.	There, the source / destination ID generally denotes the transport layer source / destination address, e.g., TCP port number.
Source / destination name	N/A	There, the source / destination ID denotes the network name of a system, e.g., hostname as stored in DNS.	There, the source / destination name can often be associated with a service such as listed by the IANA well-known port numbers [IANAPN].
Options	N/A	Protocols such as IP offer the possibility to extend the header information with fields for source routing etc.	Protocols such as TCP use optional header fields to negotiate connection parameters.

Table 63—IDS sensor characteristics—Data (see also Table 12)

IDS scope / items	Filesystem object	Process	Device	TCP
Up- / down-stream	N/A	N/A	N/A	TCP provides a reliable bidirectional data stream service. If the IDS sensor collects TCP packets on the network, the IDS is generally not able to guarantee the data stream. If the IDS sensor obtains the TCP stream data from one of the connection end points, one can consider the IDS to be operating based on a TCP stream.
PDU data	N/A	N/A	N/A	If the sensor collects its data from the network it provides TCP PDUs only—and not as one could believe the reassembled TCP stream. On the other hand if the sensor collects its information on the host, TCP PDUs are not available.
Status data	In the filesystem IDS scope the status data represents the contents of a filesystem object e.g., the contents of a file.	The status data of a process can be seen as the memory image of the process.	In the device context the contents of physical memory or a storage device can be seen status data. This includes the status of device registers.	

C.3.2 Detector data pre-processing characteristics

Table 64—Detector characteristics—data normalization (see also Table 14)

IDS scope / items	http	FTP	SMTP	Calls	Filesystem object
Single-byte character decoding	In http it is possible to encode characters in the URL with their hexadecimal representation.	N/A	N/A	N/A	N/A
Multi-byte character decoding	It is possible to use UNI standard codes in URLs	N/A	N/A	N/A	N/A
String resolution	A typical escape sequence used in http URLs is '/.' (without quotes). Such an escape sequence does not change the document or script that is being accessed, but it may be able to obfuscate the attack from an IDS. Also it is possible to represent the host portion of an URL in various different ways.	FTP and many other protocol involving filenames are susceptible to the obfuscation of attacks similar to http.	E-mail addresses may be written in complicated obfuscated ways by adding quotes etc. Unfortunately this can also be done with strings containing attack data (e.g., he infamous pipe attack, CVE-1999-0203).	The arguments passed to a system or function call may be quoted in an unusual way.	Filenames may contain various types of escape sequences.
Data decoding	In http POST data is often encoded in base64.	N/A	Mail messages—especially the attachments—are often base64 encoded.	N/A	The content of files may be compressed.

C.3.3 Detector instance analysis characteristics

Table 65—Single instance part analysis

IDS scope / items	Call	Process	Transport / network / link layer	Application layer
Basic analysis	The detector is able to recognize calls.	The detector is able to identify a thread of a process. If a process consists of a single thread only, this falls also into this category. Basic instance part analysis for processes is for example required in combination with system call sequence analysis, where the detector needs some degree of process analysis to relate system calls to each other.	Connection segments and PDU fragments are recognized, i.e., the type of the protocol is recognized.	The detector is able to identify a protocol sequence or protocol statement e.g., SMTP, FTP etc. statements.
Logic verification	The arguments of system and function calls are verified to be syntactically correct (including data types).	N/A	The structure of the instance part i.e., PDU fragment or connection segment is verified.	The syntax of the protocol request is verified with respect to the protocol specification.
Semantic verification	The arguments of system and function calls are verified to be valid, e.g., the arguments are verified to be within a meaningful value range.	N/A	Unacceptable values, value combinations, or inconsistencies of header fields are recognized.	The detector verifies the plausibility and the policy compliance of the instance part.

Table 66—Single instance analysis

IDS scope / items	Call	Process	Transport / network layer	Application layer
Basic analysis	N/A	The detector is able to identify a process and its threads.	The detector is able to continuously identify the segments or fragments belonging to a connection or a PDU. This can be seen as a very simplistic reconstruction of the instance without any logical verification, e.g., reordering of fragments or segments.	The detector is able to identify a group of protocol statements, i.e., a transaction.
Logic verification	N/A	N/A	The detector is able to recompose the instance from its parts by following the protocol specification. Further the detector is able to recognize suspicious protocol sequences e.g., stealth TCP scanning [CIN0498].	The detector is able to verify the correctness of the protocol sequence.
Semantic verification	The detector is able to analyze the impact of a call, e.g., to verify whether its return values are acceptable with respect to the call arguments.	N/A	Inconsistent but logically correct parts are recognized. Overlapping fragments are recognized. Also connection segment retransmissions or overlapping fragments that no longer contain the same data are recognized.	In the case of http one expects the detector to be able to identify the fact that a protected document was revealed. In the case of SMTP one would expect the detector to recognize the fact that a confidential document is sent to a receiver outside of the organization.

Table 67—Instance group analysis

IDS scope / items	Call	Process	Application layer
Basic analysis	The detector is able to associate calls to a process or to a user. This may additionally require some degree of process analysis.	The detector is able to identify a process group.	The detector is able to recognize an application layer protocol session, e.g., http, SMTP, Oracle, DB2, MySQL etc. This may be realized based on a well-known port number or any other simple check.
Logic verification	The detector is able to verify the logical correctness of a sequence of calls made by a process or by a user.	N/A	The detector is able to analyze the instances, i.e., the transactions executed within a session, independently.
Semantic verification	The detector is able to verify the acceptability of a task represented by a sequence of calls made by a process or by a user.	N/A	If dependency among the instances of a group exists, the detector is able to verify its consistency and the acceptability of sequences.

Table 68—Cross-instance (multi-instance) analysis

IDS scope / items	Call	**Transport / *network* / *link* layer**	**Application layer**
Basic analysis	N/A	N/A	N/A
Logic verification	N/A	N/A	N/A
Semantic verification	The detector is able to verify the semantic correctness and acceptability of a sequence of calls (the calls may be made by several independent processes).	The detector is able to identify unacceptable sequences of instances.	If a dependency among instances exists, the detector is able to verify its consistency and the acceptability of sequences.

Table 69—Bi-directional instance part group analysis

IDS scope / items	**Connection-oriented transport / network / link layer**	**Application layer**
Basic analysis	The detector is able to associate PDUs flowing in both directions.	The detector is able to identify the server response.
Logic verification	The detector is able to verify that the PDUs flowing in both directions are consistent with respect to the protocol definition.	The detector is able to verify the logical, e.g., syntactical, correctness of the server response with respect to the request sent by the client.
Semantic verification	The detector is able to verify the acceptability of the bi-directional instance observed, e.g., it is able to detect the attempt of TCP connection hijacking.	The detector is able to verify that the server's response is acceptable with respect to the request sent by the client.

C.4 IDS description example

In the preceding sections we provided example definitions and interpretations of various IDS characteristics identified in Chapter 5. In this section we illustrate the underlying IDS description scheme by providing an example description of WebIDS [Almgre99]. Because WebIDS is a highly specialized host-based IDS that focuses on the monitoring of just one specific service, i.e., the http service, its description is comparatively short.

In the following we provide the description of WebIDS that was used for the experiments described in Section 8.6. The description is split into several tables describing the various characteristics identified in Chapter 5 (see for instance Figure 21 and Figure 26). The first two tables describe the sensor portion of WebIDS, the other four its detector portion:

Table 70—Generic sensor characteristics of WebIDS

Sensor item	Value / level	Comment (see also Section 5.2.1)
Reporting time	*Post-execution*	*The information is read as the webserver writes it to the log file.*
Reporting delay	*Less than 3 seconds*	*See above.*
Reporting timestamp	*End of activity*	*The sensor passes on the time information provided by the webserver.*
Information source type	*Application level log*	*The information is read from the webserver application log file.*

Table 71—IDS scope dependent sensor characteristics of WebIDS

Sensor item type	Sensor item	IDS scope	Comment (see also Section 5.2.2)
Argument	*Basic*	*http*	*The arguments of the http request line (URL, protocol version)*
Request	*Name*	*http*	*The http request type (GET, HEAD, PUT)*
Result	*Size*	*http*	*Number of bytes transferred*
Result	*Status*	*http*	*Status (success) code of the http request*
Object	*Name*	*User*	*The (http) user ID of the user (used by the authentication method provided by the http protocol)*

Protocol control data	*Source ID*	*Network layer*	*Typically the IP address of the client*
Protocol control data	*Destination ID*	*Transport layer*	*Typically the TCP port the request was sent to: the default is port 80 and is provided only implicitly*

Table 72—Generic detector characteristics of WebIDS

Detector item	Value / level	Comment (see also Section 5.3.1)
Alarm timestamp	*End of activity*	*The detector gets the information from sensor only once the request has completed.*
Alarm delay	*Less than 3 seconds*	*The (light-weight) detector processes the information almost in real-time.*
Knowledge-based	*True*	*The detector verifies each request for a list of signatures of known attacks.*
Behavior-based	*False*	*The detector does not use any behavior-based component.*

Table 73—Data pre-processing detector characteristics of WebIDS

Detector item type	Detector item(s)	IDS scope	Comment (see also Section 5.3.2)
Filter	*Arguments, request*	*http*	*The detector may filter information based all information available in the log file.*
Filter	*Source ID*	*Network layer*	*See above.*
Filter	*Object*	*User*	*See above.*
Data normalization	*Single character decoding*	*http*	*The detector is capable of decoding hexadecimal encoded characters used in the URL.*

Table 74—Analysis-level detector characteristics of WebIDS

Detector item type (analysis type)	Detector item (level)	IDS scope	Comment (see also Section 0)
Single instance, bi-directional instance, single instance part, bi-directional instance part group	*Semantic*	*http*	*The detector is capable of analyzing the semantics of the request (e.g., theft of password file). It may even investigate the success or non-success of an attack. Accordingly, these capabilities enable an appropriate analysis of individual parts of http requests.*
Cross-instance, bi-directional cross-instance	*Basic*	*http*	*The detector is able to identify simple relations between requests for statistical purposes (see also below).*

Table 75—Control item analysis detector characteristics of WebIDS

Detector item type (analysis type)	Detector item (level)	IDS scope	Comment (see also Section 5.3.3.2.2)
Single instance, single instance part	*Size*	*http*	*The detector is capable of verifying the size of http requests.*
Single instance, single instance part	*Regular expressions*	*http*	*The detector uses regular expressions to detect suspicious activities.*

To avoid repetition we omit the description of the statistical analysis capabilities of WebIDS here. It has already been provided in Example 1 of Section 5.3.3.3.2.

Appendix D Formal specifications and description examples

In this appendix we provide the semi-formal specification for the constructs that we use to describe attack classes. This includes brief descriptions of all constructs and their elements and some extensive examples.

The description scheme used in this work employs Prolog rules to express attack classes, attack class description building blocks, attack class variations, and alarm conditions. These rules are composed of a predefined set of elements, i.e., Prolog predicates. For the semi-formal specification of these constructs we rely on the BNF grammar for the Prolog core. Based on this grammar we specify the subset of predicates that may be used to create attack descriptions etc.

D.1 BNF specification of Prolog

In its core Prolog is relatively small programming language. However, it contains a large set of predefined predicates and supports many notational variations such as infix symbols. In the following BNF specification, which we derived from the book on logical programming by Slonneger and Kurtz [SloBar95a], only the core of the language is specified. We extended their specification towards covering the predicates we use to describe input to the IDS analysis process.

```
<program> ::= <clause list> <query> | <query> .
<clause list> ::= <clause> | <clause list> <clause> .
<clause> ::= <predicate> "." | <predicate> ":-" <predicate list> "." .
<predicate> ::= <atom> | <atom> "(" <term list> ")" .
<predicate list> ::= <predicate> | "(" <predicate list> ")" | "!" |
       <predicate list> "," <predicate list> |
       <predicate list> ";" <predicate list> .
<term list> ::= <term> | <term list> "," <term> .
<list> ::= "[" <term list> "]" | "[" <term list> "|" <term> "]" | "[]" .
<term> ::= <numeral> | <predicate> | <variable> | <list> .
<query> ::= "?-" <predicate list> "." .
<atom> ::= <small atom> | "'" <string> "'" .
<small atom> ::= <lowercase letter> | <small atom> <character> .
<variable> ::= <uppercase letter> | <variable> <character> .
<lowercase letter> ::= "a" | … | "z" .
<uppercase letter> ::= "A" | … | "Z" | "_" .
<numeral> ::= <digit> | <numeral> <digit>
<digit> ::= "0" | … | "9" .
<character> ::= <lowercase letter> | <uppercase letter> | <digit> |
       <special> .
<special> ::= "+" | "-" | "*" | "/" | "\" | "^" | "~" | ":" | "." | "?" |
       "@" | "#" | "$" | "&" .
<string> ::= <character> | <string> <character> .
```

In the subsequent sections we provide the specifications for the description of attack class description building blocks etc. These specifications are relatively simple, as we simply restrict the set of predicates that may be used to formulate clauses. Note, however, that in addition to the predicates to be defined in the following we always permit the use of built-in predicates such as `not(X)` or `equal(X,Y)` within right-hand predicates. The check for equality is generally used in its infix notation: `X = Y`. Moreover note that

in the following definitions terms starting with capital letters denote variables and terms starting with lower-case letters denote atoms with a fixed value. In addition we denote lists by adding the suffix `List` and use the Prolog term "_" as a generic placeholder for empty or non-relevant fields.

D.2 Attack class description building blocks

Attack class description building blocks were introduced in Section 6.1. We therefore restrict the following descriptions to a semi-formal specification and a brief description of the permitted predicates. For right-hand predicates we permit the use of one particular predicate only:

```
adbb(basic, ADBB, ADBBScp, IDS, Detector, EffScp, ScpList, DiagInList,
     DiagOutList, VarsInList, VarsOutList, BlkFlag)
```

The `adbb` atom is used to formulate rules that describe attack class description building blocks. It takes several arguments:

- `basic`: A variable used internally by RIDAX. For the description of attack class description building blocks this variable has to be substituted with the atom `basic` or `basic2`; however, the latter may only be used to define helper rules that are included by other `adbb(basic, ...)` clauses. The sole purpose of `adbb(basic2, ...)` clauses is the simplification of the description task.
- `ADBB`: Identifier of the attack class description building block.
- `ADBBScp`: (High-level) IDS scope for which this attack class description building block is defined.
- `IDS`: Identifier of the IDS analyzed.
- `Detector`: Identifier of the detector considered.
- `EffScp`: IDS scope for which the attack class description building block is to be evaluated (see Section 6.1.1); commonly referred to as "effective IDS scope."
- `ScpList`: List of IDS scopes that the attack class considered involves.
- `DiagInList, DiagOutList`: Lists that keep track of the analysis process and record all IDS attributes that were required to evaluate the attack class considered.
- `VarsInList, VarsOutList`: Lists that keep track of the variations already considered and of those that remain to be considered.
- `BlkFlag`: Variable indicating whether the evaluation of the attack class description building block has been blocked.

Most of these arguments are reused in the following. Moreover note that they have already been introduced in Section 6.2.1, where we provided a simplified example of an attack class description.

For the right-hand predicates we permit the use of a set of 13 predicates that can be used to describe attack class description building blocks (in the following we use "(...)" as a place holder for argument lists defined earlier):

```
adbb(eval, ...), adbb(basic2, ...)
```

In Section 6.1.3 we explained why and how attack class description building blocks can depend upon each other. We therefore permit the description of an attack class description building block to include other attack class description building blocks in its evaluation. See also the second example in Section 6.1.3.

```
chkDetAttrib(Detector, RequestedScp, ItemType, Item,
       DiagInList, DiagOutList)
```

This predicate verifies whether the detector description contains an attribute that enables the IDS to perform the analysis that is described by `ItemType` and `Item` at the IDS scope specified by `RequestedScp`. This rule, however, never fails. If the IDS is found capable of performing the analysis, this is recorded accordingly in `DiagOutList`, which serves as input to the alarm analysis process described in Section 7.1.2.

- `RequestedScp`: IDS scope of the requested IDS characteristics.
- `ItemType`, `Item`: Generic identifier of the requested IDS characteristics.

All other arguments have already been defined.

```
chkSensorAttrib(Ids, Sensor, RequestedScp, ItemType, Item,
       DiagInList, DiagOutList)
```

This predicate performs the same verification for sensors as `chkDetAttrib` does for detectors.

```
reqDetAttrib(...), reqSensorAttrib(...)
```

These predicates are similar to `chkDetAttrib` and `chkSensorAttrib`. The difference is that they fail if the requested characteristic is not available.

```
reqDetAttribAbsence(...), reqSensorAttribAbsence(...)
```

These predicates are similar to `reqDetAttrib` and `reqSensorAttrib`. The difference is that they fail if the requested characteristic is available and succeed if no characteristics specified by `ItemType`, `Item` and `RequestedScp` is available.

```
variation(exercise, _, _, Detector, EffScp, ScpList,
       DiagInList, DiagOutList, VarsInList, VarsOutList, BlkFlag)
```

This predicate exercises the variations listed in `VarsInList` that are applicable to the IDS scope specified by `EffScp`. The effects of the variation are recorded in `DiagOutList`. If the variation blocks any further analysis the `BlkFlag` variable is set accordingly.

```
relBlkFlag(BlkFlagIn, BlkFlagOut)
```

This predicate is used to resolve a RIDAX, i.e., Prolog, specific issue that requires the `BlkFlag` argument given to other predicates to be an uninstantiated variable. It is the first of a set of helper predicates described in the following.

```
selectAnyScp(SuperScp, SubScp, ScpList), selectSubScp(_), selectSuperScp(_)
```

These predicates traverse the IDS scope tree and return either a lower-level (`SubScp`) or a higher-level (`SuperScp`) IDS scope from the list of IDS scopes provided in `ScpList`. `selectSubScp`, for instance, searches `ScpList` for an IDS scope `SubScp` below `SuperScp`.

```
statCombinations(Comparison, Time, History, Unit, Item)
```

This helper predicate facilitates the specification of detector characteristics related to statistical analysis as described in Section 5.3.3.3.2.

D.3 Attack descriptions

The specification of the core attack classes, i.e., attack class descriptions, is similar to the one for attack class description building blocks, and is restricted to a single left-hand predicate as well:

```
attack(basic, Attack, EffScp, IDS, Detector, ScpList, MaxVars, DiagInList,
       DiagOutList, VarsList, BlkFlag)
```

However, there are some important differences in the purpose of attack class and attack class description building block descriptions, resulting in some differences in the argument list:

- `Attack`: Identifier of the attack class.
- `MaxVars`: Maximum number of variations that may be selected concurrently.
- `VarsList`: The set of variations selected.

Note that `EffScp`, `ScpList`, `VarsList` and `BlkFlag` represent output variables. In fact, the combination of `Attack`, `EffScp` and `VarsList` defines an variant, as introduced in Chapter 6.

The set of right-hand predicates that may be used to describe an attack class is almost identical to the one permitted for attack class description building blocks. The only difference is that attack class descriptions support an additional predicate that performs the selection of variations to be considered:

```
selectVars(MaxVars, VarsList, ScpList, DiagInList, DiagOutList)
```

Each attack class description should select the variations to be considered before describing the actual core of the attack class (see also the example description in Section 6.2.1). Moreover, note that creating complex attack class descriptions is discouraged. Instead, it is preferable to create attack class description building block descriptions that can then be used to compose multiple attack class descriptions.

D.4 Attack class variations

The specification of attack class variations is similar to that of the attack class description building blocks as well. However, the complete description of a variation requires two clauses, i.e., left-hand predicates. A first clause defines the existence of the variation:

```
variation(index, Var, VarScp, _, VarIndex, _, _, _, _, _, _, _)
```

Its primary goal is the definition of the variation index (`VarIndex`) and the IDS scope (`VarScp`) at which the variations is defined. See also Sections 6.3.1 and 6.3.3. Based on this clause the `selectVars` predicate selects the variations to be considered. Moreover, note that the two-tuple composed of the identifier `Var` and the IDS scope `VarScp` identifies variations. This clause does not support any right-hand predicates.

The second clause describes the actual variation and is defined by the following left-hand predicate:

```
variation(blkChk, Var, VarScp, IDS, Detector, EffScp, ScpList,
      DiagInList, DiagOutList, _, _, BlkFlag)
```

The set of right-hand predicates that may be used to express these clauses is again almost identical to that permitted for the description of attack class description building blocks. The only difference is that the description of a variation may in addition suppress IDS characteristics:

```
supDetAttrib(...), supSensorAttrib(...)
```

These predicates take the same arguments as `chkDetAttrib` and `chkSensorAttrib` defined in Appendix D.2. Once such a predicate has succeeded all IDS characteristics described by `ItemType` and `Item` are unavailable for further analysis at all IDS scopes of `RequestedScp` and below.

D.5 Alarm condition specification

The specification of alarm conditions introduced in Section 7.1.2.1 is also similar to that of attack class description building blocks. The clauses describing alarm conditions start with the following left-hand predicate:

```
alarmChk(basic, Alarm, AlarmScp, EffScp, SuccessState, IDS, Detector,
      _, _, DiagInList, DiagOutList)
```

In accordance to the clauses introduced, alarm conditions are identified by the arguments `Alarm` and `AlarmScp`. The (output) variable `EffScp` reflects the IDS scope for which the analysis IDS has the potential of generating alarms that belong to the alarm class identified by `Alarm` and `AlarmScp`. In addition, the variable `SuccessState` reflects whether the IDS analyzed is capable of analyzing the attack reported in terms of its success.

For the description of alarm conditions all except one of right-hand predicates used for describing attack class description building blocks are permitted. The only restriction is that we do not permit variations

(variation) to be included in alarm conditions. In addition, we include two predicates that permit the examination of the IDS characteristics that were required for the analysis of attack class variant considered:

```
reqDetAttribUsed (Detector, RequestedScp, EffScp, ItempType, Item,
        DiagInList, DiagOutList), reqSensorAttribUsed (_)
```

The predicates `reqDetAttribUsed` and `reqSensorAttribUsed` only succeed if the analysis of the attack class variant required that the IDS analyzed employs the IDS characteristics described by `ItemType`, `Item` and `RequestedScp`. If the predicate succeeds, `EffScp` reports the precise IDS scope at which the IDS made use of the IDS characteristics required.

D.6 Example descriptions

In this section we provide Prolog code examples to illustrate how the above specifications were used to describe attack classes etc. The examples are taken from the RIDAX prototype implementation.

To facilitate the reading of the examples we provide the definitions of the most basic abbreviations, i.e., atoms. All IDS characteristics used in the following examples have been defined in Chapter 5; example interpretations are provided in Appendix C.

Table 76—Abbreviations of detector item types

Basic detector item types	The detector item types (see Section 5.3) define item categories as illustrated for example in Figure 26. All item types related to instance analysis may be extended further by appending the extensions bidir (bi-directional analysis), data (analysis of data), info (analysis of control information), seq (analysis of sequences), stat (statistical analysis), and time (timing related analysis).
`si_i`	*Single instance analysis*
`si_ip`	*Single instance part analysis*
`cr_i`	*Cross-instance analysis*
`cr_ip`	*Cross-instance part analysis*
`i_grp`	*Instance group analysis*
`ip_grp`	*Instance part group analysis*
`method`	*Detection method*
`data_norm`	*Data normalization*
`filter`	*Filtering*
`alarm_delay`	*Delay of alarms*
`alarm_timestamp`	*Alarm timestamp*

Table 77—Abbreviations of sensor item types

Sensor item types	The sensor item types (see Section 5.2) define item categories as illustrated for example in Figure 21.
args	*Arguments*
data	*Data*
obj	*Object*
obj_attrib	*Object attributes*
prot_ctl	*Protocol control information*
report_delay	*Delay of report provided to detector*
report_time	*Reporting time*
report_timestmp	*Reporting timestamp provided*
req	*Request*
res	*Results*
sensor_type	*Sensor type; information source monitored*

D.6.1 Alarm class description building blocks examples

```
/* ----------------------------------------------------------------------
 * adbb/l7req - definition of the requirements to handle most
 * basic layer 7 activity. We start with description for a network-based
 * IDS.
 */

adbb(basic, AB, ABSCP, IDS, DET, EFFSCP, SCPLIST,
       DIAGIN, DIAGOUT, VARSIN, VARSOUT, BLKFLAG) :-
    (AB=reqDataUp; AB=reqDataDown; AB=reqDataBiDir), ABSCP=app_l,
       /* This only applies for external data sources */
    reqSensorAttrib(IDS, _, generic, sensor_type, data_external,
       DIAGIN, DIAG1),
       /* verify and get the scope */
    selectSubScp(ABSCP, EFFSCP, SCPLIST),
       /* blocking... */
    relBlkFlag(BLKFLAG, BLK1),
       /* check the lower layers */
    adbb(eval, AB, trsp_l, IDS, DET, _, SCPLIST,
       DIAG1, DIAG2, VARSIN, VARS1, BLK1),
       /* blocking... */
    relBlkFlag(BLK1, BLK2),
       /* check the required service access point of the
        * underlaying layer */
    adbb(basic2, AB, ABSCP, IDS, DET, EFFSCP, SCPLIST,
       DIAG2, DIAG3, VARS1, VARS2, BLK2),
       /* blocking... */
    relBlkFlag(BLK2, BLK3),
       /* now check the variations for this layer */
    selectAnyScp(ABSCP, VSCP, SCPLIST),
    variation(exercise, _, _, IDS, DET, VSCP, SCPLIST,
       DIAG3, DIAGOUT, VARS2, VARSOUT, BLK3),
    relBlkFlag(BLK3, BLKFLAG).
```

```
adbb(basic2, AB, ABSCP, IDS, _, EFFSCP, SCPLIST,
       DIAGIN, DIAGOUT, VARS, VARS, notBlk) :-
       /* here we verify the basic requirements for ABSCP scope awareness */
    (AB=reqDataUp; AB=reqDataDown; AB=reqDataBiDir), ABSCP=app_1,
       /* verify and get the scope */
    selectSubScp(ABSCP, EFFSCP, SCPLIST),
       /* require the layer 4 data to be available */
    reqSensorAttrib(IDS, _, EFFSCP, data, pdu, DIAGIN, DIAGOUT).

/* ---------------------------------------------------------------------
 * adbb/17req - alternatively we consider an IDS that is host based
 * and is providing access to tcp or udp data streams i.e., to the data
 * exchanged over sockets.
 */

adbb(basic, AB, ABSCP, IDS, _, EFFSCP, SCPLIST,
       DIAGIN, DIAGOUT, VARS, VARS, notBlk) :-
    AB=reqDataUp, ABSCP=app_1,
       /* This only applies for internal data sources */
    reqSensorAttrib(IDS, SENSOR, generic, sensor_type, data_internal,
       DIAGIN, DIAG1),
       /* verify and get the scope */
    selectSubScp(ABSCP, EFFSCP, SCPLIST),
       /* check the required service access point of the
         * underlaying layer */
    reqSensorAttrib(IDS, SENSOR, EFFSCP, data, up_stream, DIAG1, DIAGOUT).

adbb(basic, AB, ABSCP, IDS, _, EFFSCP, SCPLIST,
       DIAGIN, DIAGOUT, VARS, VARS, notBlk) :-
    AB=reqDataDown, ABSCP=app_1,
       /* This only applies for internal data sources */
    reqSensorAttrib(IDS, SENSOR, generic, sensor_type, data_internal,
       DIAGIN, DIAG1),
       /* verify and get the scope */
    selectSubScp(ABSCP, EFFSCP, SCPLIST),
       /* check the required service access point of the
         * underlaying layer */
    reqSensorAttrib(IDS, SENSOR, EFFSCP, data, down_stream, DIAG1, DIAGOUT).

adbb(basic, AB, ABSCP, IDS, _, EFFSCP, SCPLIST,
       DIAGIN, DIAGOUT, VARS, VARS, notBlk) :-
    AB=reqDataBiDir, ABSCP=app_1,
       /* This only applies for internal data sources */
    reqSensorAttrib(IDS, SENSOR, generic, sensor_type, data_internal,
       DIAGIN, DIAG1),
       /* verify and get the scope */
    selectSubScp(ABSCP, EFFSCP, SCPLIST),
       /* check the required service access point of the
         * underlaying layer */
    reqSensorAttrib(IDS, SENSOR, EFFSCP, data, up_stream, DIAG1, DIAG2),
    reqSensorAttrib(IDS, SENSOR, EFFSCP, data, down_stream, DIAG2, DIAGOUT).
/* we do not exercise any transport layer variations here because
 * they should have been removed by the networking stack already */
```

```
/* ------------------------------------------------------------------
 * adbb/argBOF/optBOF - a buffer overflow attack using arguments
 * i.e., the request line of protocol session or the arguments
 * of a function call etc.
 */

adbb(basic, AB, ABSCP, IDS, DET, _, SCPLIST,
       DIAGIN, DIAGOUT, VARS, VARS, notBlk) :-
       (AB=argBOF; AB=optBOF),
       (ABSCP=app_l; ABSCP=call),
           /* verify and get the scope */
       selectSubScp(call, EFFSCP, SCPLIST),
           /* information required */
       reqSensorAttrib(IDS, SENSOR, EFFSCP, req, _, DIAGIN, DIAG2),
       reqSensorAttrib(IDS, SENSOR, proc, obj, id, DIAG2, DIAG4),
           /* awareness level */
       reqDetAttrib(DET, proc, si_i, basic, DIAG4, DIAG6),
       reqDetAttrib(DET, EFFSCP, si_i, basic, DIAG6, DIAG8),
       reqDetAttrib(DET, EFFSCP, i_grp, logic, DIAG8, DIAG10),
           /* the techniques */
       reqDetAttrib(DET, EFFSCP, i_grp_seq, fixed, DIAG10, DIAGOUT).
```

D.6.2 Attack description example

See also the example of the http argument buffer overflow attack provided in Section 6.2.1.

```
/* ------------------------------------------------------------------
 * attack/suspArg - suspicious but benign argument string.
 * we verify the possibility of weak string signatures.
 * ActNbr: 18-20, 38, 39
 */
attack(basic, ACT, EFFSCP, IDS, DET, SCPLIST, MaxVars,
       DIAGIN, DIAGOUT, VARS, BLKFLAG) :-
       ACT=suspArg,
       (((EFFSCP=http; EFFSCP=smtp; EFFSCP=ftp),LSCP=tcp);
        ((EFFSCP=domain; EFFSCP=syslog),LSCP=udp)),
       SCPLIST=[EFFSCP,LSCP,ipv4,ieee_802_3],
       /* select the variations to consider */
    selectVars(MaxVars, VARS, SCPLIST, DIAGIN, DIAG2),
       /* This activity is in fact a layer 7 request */
    adbb(eval, reqCtlUp, app_l, IDS, DET, EFFSCP, SCPLIST,
       DIAG2, DIAG4, VARS, VARS2, BLK1),
    relBlkFlag(BLK1, BLKFLAG),
       /* now check for the attack specific things i.e., the BOF */
    adbb(eval, suspArg, app_l, IDS, DET, EFFSCP, SCPLIST,
       DIAG4, DIAGOUT, VARS2, _, BLKFLAG).
```

D.6.3 Attack variation examples

```
/* ------------------------------------------------------------------
 * variation/l3frgFirst - fragmentation on layer 3
 * This variation allows IDSs that are able to analyze only the first
 * fragment only to recognize an attack correctly - assuming that the
 * attack can be identified by looking at the first fragment.
 */
variation(index, l3frgFirst, net_l_frg, _, 3500, _, _, _, _, _, _, _).
variation(blkChk, VAR, _, _, DET, EFFSCP, _,
       DIAGIN, DIAGOUT, _, _, notBlk) :-
       VAR=l3frgFirst,
            /* simply check for fragment awareness
             * we assume that the interesting data is in the first
             * fragement. */
       chkDetAttrib(DET, EFFSCP, si_ip, basic, DIAGIN, DIAGOUT), ...
```

```
/* ------------------------------------------------------------------------
 * variation/l3frgNotFirst - fragmentation on layer 3
 * This variation makes the assumption that the suspicious data is
 * distributed over multiple fragments, and that the IDS needs to be
 * able recompose these fragments in order to recognize the attack.
 */
variation(index, l3frgNotFirst, net_l_frg, _, 3500, _, _, _, _, _, _, _).
variation(blkChk, VAR, _, IDS, DET, EFFSCP, _,
       DIAGIN, DIAGOUT, _, _, notBlk) :-
       VAR=l3frgNotFirst,
           /* We require the IDS to be able to recompose fragments */
       reqDetAttrib(DET, EFFSCP, si_i, basic, DIAGIN, DIAG1),
       reqDetAttrib(DET, EFFSCP, si_ip, basic, DIAG1, DIAG2),
       reqDetAttrib(DET, EFFSCP, ip_grp, logic, DIAG2, DIAG4),
           /* Layer 3 data and header */
       reqSensorAttrib(IDS, DET, EFFSCP, data, pdu, DIAG4, DIAG6),
       reqSensorAttrib(IDS, DET, EFFSCP, prot_ctl, frag_ctl, DIAG6,
           DIAGOUT), !.
```

D.6.4 Alarm condition examples

```
/* ------------------------------------------------------------------------
 * alarmChk/argStr - a suspicious argument string has been observed.
 * The IDS is capable of reporting the success-state of the suspected
 * attack.
 */
alarmChk(basic, ALR, ALRSCP, EFFSCP, true, IDS, DET, SCPLIST, _, DIAGIN,
       DIAGOUT) :-
    ALR=argStr, ALRSCP=app_l,
       /* simplification of rule: reuse the rule for detecting suspicious
       /* data without reporting of success-state */
    alarmChk(basic, ALR, ALRSCP, EFFSCP, false, IDS, DET, SCPLIST,
       _, DIAGIN, DIAG2),
       /* now verify whether the IDS is capable to verify the success */
    reqSensorAttrib(IDS, _, EFFSCP, res, status, DIAG2, DIAG4),
    (reqDetAttrib(DET, EFFSCP, si_i_bidir, semantic, DIAG4, DIAG6);
     reqDetAttrib(DET, EFFSCP, ip_grp_bidir, semantic, DIAG4, DIAG6)),
       /* require the data */
    adbb(eval, reqCtlDown, app_l, IDS, DET, EFFSCP, SCPLIST,
       DIAG6, DIAGOUT, [], _, notBlk).

/* ------------------------------------------------------------------------
 * alarmChk/argStr - a suspicious string has been observed.
 */
alarmChk(basic, ALR, ALRSCP, EFFSCP1, false, IDS, DET, SCPLIST, _, DIAGIN,
       DIAGOUT) :-
    (ALR=argStr;
     ALR=optStr),
    ALRSCP=app_l,
    selectSubScp(ALRSCP, EFFSCP1, SCPLIST),
    TSCP=trsp_l,
       /* Check the IDS capabilities that were required */
    (reqDetAttribUsed(DET, TSCP, EFFSCP, si_ip_data, string, DIAGIN, DIAG4);
     reqDetAttribUsed(DET, TSCP, EFFSCP, si_i_data, string, DIAGIN, DIAG4)),
       /* check the required the data */
    (reqSensorAttribUsed(IDS, _, TSCP,EFFSCP, data, pdu, DIAG4, DIAGOUT);
     reqSensorAttribUsed(IDS, _, TSCP,EFFSCP, data, up_stream, DIAG4,
       DIAGOUT)).
```

Appendix E Glossary

In the following we summarize the terms that were defined within the MAFTIA framework [D21Maf03] and that are relevant to this work and to ID. Note that we provide only the definition of terms relevant to this work and that many of these definitions are based on the concept of security policy as defined in MAFTIA D21 [D21Maf03], Section 3.1.

- *Activity:* event or a sequence of *events* within a given context.
- *Alarm (intrusion detection ~):* a report of an error that may lead to or has led to a security failure, optionally including diagnostic information about the cause of the error.
- *Alarm (false ~):* see *false positive.*
- *Attack (general sense):* a malicious interaction fault, through which an attacker aims to deliberately violate one or more security properties; an *intrusion* attempt; (human sense) a malicious human interaction fault whereby an attacker aims to deliberately violate one or more security properties; (technical sense) a malicious technical interaction fault aiming to exploit a vulnerability as a step towards achieving the final aim of the attacker.
- *Error:* part of the state of a system liable to lead to *failure* [LaAvKo92]; manifestation of a *fault* in a system [LaAvKo92].
- *Event:* a thing that happens or takes place [OMED92]; a change in *state.*
- *Failure:* event occurring when the delivered service deviates from fulfilling the *system function,* i.e., from what the system is intended for [Laprie98]; transition from *correct service* to *incorrect service* [LaAvKo92]; see also *security failure.*
- *Failure (security ~):* violation of a security property of the intended *security policy.*
- *False positive:* the *event* corresponding to the incorrect decision to rate an *activity* as being erroneous; also called a "false alarm" or "type II error."
- *False negative:* the *event* corresponding to the incorrect decision not to rate an *activity* as being erroneous; also called a "miss" or "type I error."
- *Fault:* the adjudged or hypothesized cause of an *error* [LaAvKo92]; *error* cause intended to be avoided or tolerated [LaAvKo92]; consequence for a *system* of the *failure* of another *system* that has interacted or is interacting with the *system* considered [LaAvKo92].
- *Intrusion:* a *malicious,* externally-induced fault resulting from an *attack* that has succeeded in exploiting a *vulnerability.*
- *Malicious:* intending or intended to do harm [OMED92].
- *Security policy:* description of 1) the security properties to be fulfilled by a computing system; 2) the rules according to which the system security state can evolve.
- *Service:* system behavior as perceived by a system user [LaAvKo92].

- *Service (correct ~): service* that fulfills the *system function* [Laprie98].
- *Service (incorrect ~): service* that does not fulfill the *system function* [Laprie98].
- *State (system ~):* a condition of being, with respect to a set of circumstances [LaAvKo92].
- *System:* entity having interacted, interacting or able to interact with other entities [LaAvKo92]; set of components bound together in order to interact [LaAvKo92].
- *System function:* that for which the *system* is intended [Laprie98].
- *True positive:* the *event* corresponding to the correct decision to rate an *activity* as being erroneous; also called a "hit."
- *True negative:* the *event* corresponding to the correct decision not to rate an *activity* as being erroneous.
- *User (system ~):* another *system* (physical, human) interacting with the *system* considered [LaAvKo92].
- *Vulnerability:* a fault created during development of the system, or during operation, that could be exploited to create an intrusion.

List of Tables

List of Figures

References

[ACFMPS99] Julia Allen, Alan Christie, William Fithen, John McHugh, Jed Pickel, and Ed Stoner, "State of the Practice of Intrusion Detection Technologies," Carnegie Mellon University, Pittsburgh, PA, Technical Report CMU/SEI-99-TR-028, http://www.sei.cmu.edu/pub/documents/99.reports/pdf/99tr028.pdf, 1999.

[Aleph96] Aleph One, "Smashing the stack for fun and profit," in *Phrack Magazine*, vol. 7. http://www.phrack.org/show.php?p=49&a=14, 1996.

[Alessa00] Dominique Alessandri, "Using Rule-Based Activity Descriptions to Evaluate Intrusion-Detection Systems," presented at Third International Workshop on Recent Advances in Intrusion Detection (RAID2000), Toulouse, France, published in LNCS, vol. 1907, http://link.springer.de/link/service/series/0558/bibs/1907/19070183.htm, 2000, pp. 183--96.

[Alessa03a] Dominique Alessandri, "Demonstration of an Intrusion-Tolerant Intrusion Detection System," http://www.newcastle.research.ec.org/maftia/meetings/plenary/papers/Demo-WP3.pdf, 2003.

[Almgre99] Magnus Almgren, "Design and Implementation of a Lightweight Tool for Detecting Web Server Attacks," M.S. Thesis, Uppsala: University of Uppsala, Sweden, Department of Scientific Computing, 1999, pp. 60.

[AlmLin01] Magnus Almgren and Ulf Lindqvist, "Interfacing Trusted Applications with Intrusion Detection Systems," presented at Fourth International Workshop on Recent Advances in Intrusion Detection (RAID2000), UC Davis, CA, http://www.raid-symposium.org/raid2001/program.html, 2001.

[Amoro99] E. G. Amoroso, *Intrusion Detection: An Introduction to Internet Surveillance, Correlation, Trace Back, Traps, and Response*, first ed. Sparta, New Jersey: Intrusion.Net Books, 1999, ISBN 0-9666700-7-8.

[Anders72] James P. Anderson, "Computer Security Technology Planning Study," Hanscom AFB, Bedford, Technical Report ESC-TR-73-51, http://seclab.cs.ucdavis.edu/projects/history/CD/ande72.pdf, 1972.

[Anders80] James P. Anderson, "Computer Security Threat Monitoring and Surveillance," James P. Anderson Co., Fort Washington, PA, April 1980.

[Apache] Apache Foundation, "Apache webserver software," http://www.apache.org/.

[AsKrSp96] Taimur Aslam, Ivan Krsul, and Eugene H. Spafford, "Use of A Taxonomy of Security Faults," Purdue University, COAST Laboratory, West Lafayette, IN, Tech. Report TR-96-051, 1996.

[Aslam95] Taimur Aslam, "A Taxonomy of Security Faults in the UNIX Operating System," M.S. Thesis, West Lafayette, IN: Purdue University, Computer Sciences Department, 1995, pp. 120.

[AvLaRa00] A. Avizienis, J.-C. Laprie, and B. Randell, "Fundamental Concepts of Dependability," presented at Third Information Survivability Workshop (ISW-2000), Boston, MA, http://www.cert.org/research/isw/isw2000/papers/56.pdf, 2000.

[Axelss00] Stefan Axelsson, "Intrusion Detection Systems: A Survey and Taxonomy," Chalmers University of Technology, Dept. of Computer Engineering, Göteborg, Sweden, Technical Report 99-15, http://www.ce.chalmers.se/staff/sax/taxonomy.ps, 2000.

[Axelss99b] Stefan Axelsson, "Research in Intrusion-Detection Systems: A Survey," Department of Computer Engineering, Chalmers University of Technology, Göteborg, Sweden, Technical Report 98-17, http://www.ce.chalmers.se/staff/sax/survey.ps, 1999.

[BaCoBe94] Brian T. Bartell, Garrison W. Cottrell, and Richard K. Belew, "Optimizing Parameters in a Ranked Retrieval System Using Multi-Query Relevance Feedback," presented at Third Annual Symposium on Document Analysis and Information Retrieval, Las Vegas, http://www-cse.ucsd.edu/users/rik/papers/dair94/inc.ps, 1994, pp. 8.

[Bailey94] Kenneth D. Bailey, *Typologies and Taxonomies: An Introduction to Classification Techniques*. Thousand Oaks, CA: Sage Publications, 1994, ISBN 0803952597.

[BasPer84] V. Basili and B. Perricone, "Software Errors and Complexity," *Communications of the ACM*, vol. 27, pp. 42--52, 1984.

[BeGlRa98] R. Benjamin, B. Gladman, and B. Randell, "Protecting IT Systems from Cyber Crime," Imperial College, London, UK, Technical Report 1998.

[Biggs02] Norman L. Biggs, *Discrete Mathematics*, Second ed. New York: Oxford University Press, 2002, ISBN 0-19-850717-8.

[CA0198] CERT Coordination Center, "'smurf' IP Denial-of-Service Attacks," CERT Coordination Center, Pittsburgh, PA, Advisory CA-98.01, ftp://ftp.cert.org/pub/cert_advisories/CA-98.01.smurf, 1998.

[CA0696] CERT Coordination Center, "Vulnerability in NCSA/Apache CGI example code," CERT Coordination Center, Pittsburgh, PA, Advisory CA-96.06, ftp://ftp.cert.org/pub/cert_advisories/CA-96.06.cgi_example_code, 1996.

[CA0797] CERT Coordination Center, "Vulnerability in the httpd nph-test-cgi script," CERT Coordination Center, Pittsburgh, PA, Advisory CA-97.07, ftp://ftp.cert.org/pub/cert_advisories/CA-97.07.nph-test-cgi_script, 1997.

[CA1190] CERT Coordination Center, "Security probes from Italy," CERT Coordination Center, Pittsburgh, PA, Advisory CA-90.11, http://www.cert.org/advisories/CA-1990-11.html, 1990.

[CA1201] CERT Coordination Center, "Superfluous Decoding Vulnerability in IIS," http://www.cert.org/advisories/CA-2001-12.html, 2001, last update: May 15, 2001.

[CA1301] CERT Coordination Center, "Buffer Overflow In IIS Indexing Service DLL," http://www.cert.org/advisories/CA-2001-13.html, 2001, last update: June 19, 2001.

[CA1395] CERT Coordination Center, "Syslog Vulnerability - A Workaround for Sendmail," CERT Coordination Center, Pittsburgh, PA, Advisory CA-95.13, 1995.

[CA1499] CERT Coordination Center, "Multiple Vulnerabilities in BIND," CERT Coordination Center, Pittsburgh, PA, Advisory CA-99.14, http://www.cert.org/advisories/CA-1999-14.html, 1999.

[CA1901] CERT Coordination Center, ""Code Red" Worm Exploiting Buffer Overflow in IIS Indexing Service DLL," http://www.cert.org/advisories/CA-2001-19.html, 2001, last update: August 23, 2001.

[CA2196] CERT Coordination Center, "TCP SYN Flooding," CERT Coordination Center, Pittsburgh, PA, Advisory CA-96.21, ftp://ftp.cert.org/pub/cert_advisories/CA-96.21.tcp_syn_flooding, 1996.

[CA2301] CERT Coordination Center, "Continuing Threat of the "Code Red" Worm," http://www.cert.org/advisories/CA-2001-23.html, 2001, last update: August 23, 2001.

[CA2897] CERT Coordination Center, "IP Denial-of-Service Attacks," CERT Coordination Center, Pittsburgh, PA, Advisory CA-97.28, ftp://ftp.cert.org/pub/cert_advisories/CA-97%3A28.Teardrop_Land, 1997.

[CDEKS96] Mark Crosbie, Bryn Dole, Todd Ellis, Ivan Krsul, and Eugene Spafford, "IDIOT - User Guide," Purdue University, COAST Laboratory, West Lafayette, IN, Tech. Report 1996.

[CERT] CERT Coordination Center, "Computer Emergency Response Team (CERT) / Coordination Center," http://www.cert.org/meet_cert/meetcertcc.html, http://www.cert.org/present/cert-overview-trends/, 1988.

[CheBel94] W. R. Cheswick and S. M. Bellovin, *Firewalls and Internet Security - Repelling the Wily Hacker*. Reading, MA: Addison-Wesley Publishing Company, 1994.

[CIDF98] Phil Porras, Dan Schnackenberg, Stuart Staniford-Chen, Maureen Stillman, and Felix Wu, "The Common Intrusion Detection Framework Architecture," http://www.isi.edu/~brian/cidf/drafts/architecture.txt, 1998.

[CIN0498] CERT Coordination Center, "CERT Incident Note IN-98-04 - Advanced Scanning," CERT Coordination Center, Pittsburgh, Incident Note IN-98-04, http://www.cert.org/incident_notes/IN-98-04.html, 1998.

[CIN0799] CERT Coordination Center, "CERT Incident Note IN-99-07 - Distributed Denial of Sevice Tools," CERT Coordination Center, Pittsburgh, Incident Note IN-99-07, http://www.cert.org/incident_notes/IN-99-07.html, 1999.

[CiscoNR99] Commercial Product, "NetRanger," Cisco Systems Inc., http://www.cisco.com/warp/public/cc/cisco/mkt/security/nranger/prodlit/netra_ds.htm, 1999.

[CloMel94] William F. Clocksin and Christopher S. Mellish, *Programming in Prolog*, Fourth ed. Berlin, Heidelberg: Springer-Verlag, 1994, ISBN 3-540-58350-5.

[Cohen95] F. B. Cohen, *Protection and Security on the Information Superhighway*. New York, NY: John Wiley & Sons Inc., 1995.

[CovTho91] Thomas M. Cover and Joy A. Thomas, *Elements of Information Theory*. New York, NY: John Wiley & Sons, Inc., 1991, ISBN 0-471-06259-6.

[CVE99] The MITRE Corporation, "Common Vulnerabilities and Exposures," http: /cve.mitre.org , 1999.

[CVE002100] CVE editorial board, "CAN-2000-0021: Lotus Domino HTTP buffer-overflow," http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2000-0021, 2000.

[CVE007099] CVE editorial board, "CVE-1999-0070: test-cgi webserver vulnerability," http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0070, 1999.

[CVE033301] CVE editorial board, "CVE-2001-0333: Directory traversal vulnerability in IIS," http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0333, 2001.

[CVE084800] CVE editorial board, "CVE-2000-0848: IBM WebSphere Application Server Plugin HTTP-header Vulnerability," http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0848, 2000.

[CVE087499] CVE editorial board, "CVE-1999-0874: Buffer overflow in IIS," http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0874, 1999.

[D1Maf00] MAFTIA Consortium, "Reference Model and Use Cases," C. Cachin, Ed., Malicious- and Accidental- Fault Tolerance for Internet Applications, MAFTIA project deliverable D1, 2000.

[D2Maf01] MAFTIA Consortium, "Architecture and revised model of MAFTIA," R. Stroud, Ed., Malicious- and Accidental- Fault Tolerance for Internet Applications, Newcastle upon Tyne, UK, MAFTIA project deliverable D2, 2001.

[D3Maf01] MAFTIA Consortium, "Towards a Taxonomy of Intrusion Detection Systems and Attacks," D. Alessandri, Ed., Malicious- and Accidental- Fault Tolerance for Internet Applications, Zurich, Switzerland, MAFTIA project deliverable D3, http://www.newcastle.research.ec.org/maftia/deliverables/D3.pdf, 2001.

[D10Maf02] MAFTIA Consortium, "Design of an Intrusion-Tolerant Intrusion Detection System," M. Dacier, Ed., Malicious- and Accidental- Fault Tolerance for Internet Applications, Zurich, Switzerland, MAFTIA project deliverable D10, http://www.newcastle.research.ec.org/maftia/deliverables/D10.pdf, 2002.

[D21Maf03] MAFTIA Consortium, "Conceptual Model and Architecture of MAFTIA," D. Powell and R. Stroud, Eds., Malicious- and Accidental- Fault Tolerance for Internet Applications; LAAS-CNRS, Toulouse and University of Newcastle upon Tyne, MAFTIA project deliverable D21, http://www.newcastle.research.ec.org/maftia/deliverables/D21.pdf, 2003.

[DacAle99] Marc Dacier and Dominique Alessandri, "VulDa: A Vulnerability Database," presented at 2nd Workshop on Research with Security Vulnerability Databases, Purdue University, IN, 1999.

[Das00] Kumar J. Das, "Attack Development for Intrusion Detection Evaluation," M.S. thesis: Massachusetts Institute of Technology, Electrical Engineering and Computer Science, http://www.cs.ucf.edu/~allen/security/papers/Das00.pdf, 2000, pp. 97.

[DCWMS99] Robert Durst, Terrence Champion, Brian Witten, Eric Miller, and Luigi Spanguolo, "Testing and Evaluating Computer Intrusion Detection Systems," *Comm. of ACM*, vol. 42, 1999.

[DDWL98] H. Debar, M. Dacier, A. Wespi, and S. Lampart, "An Experimentation Workbench For Intrusion Detection Systems," IBM Research Division, Zurich, Switzerland, Research Report RZ 2998, http://domino.watson.ibm.com/library/cyberdig.nsf/a3807c5b4823c53f85256561006324be/647aa2a69bcc8ff2852565e6004d3897?OpenDocument, 1998.

[DeBeSi92] Hervé Debar, Monique Becker, and Didier Siboni, "A neural network component for an intrusion detection system," presented at IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, CA, 1992, pp. 240--50.

[DeDaWe00] Hervé Debar, Marc Dacier, and Andreas Wespi, "A Revised Taxonomy for Intrusion-Detection Systems," presented at Annales des Télécommunications, vol. 55, 2000, pp. 361--78.

[DeDaWe99] Hervé Debar, Marc Dacier, and Andreas Wespi, "Towards a Taxonomy of Intrusion Detection Systems," *Computer Networks*, vol. 31, pp. 805--22, 1999.

[DeHuDo00] H. Debar, M.-Y. Huang, and D. J. Donahoo, "Intrusion Detection Exchange Format Data Model," http://www.ietf.org/internet-drafts/draft-ietf-idwg-data-model-03.txt, 2000, last update: June 15, 2000.

[DeMMat95] R. A. DeMillo and A. P. Mathur, "A Grammar Based Fault Classification Scheme and its Application to the Classification of the Errors of TEX," Purdue University, Software Engineering Research Center, West Lafayette, IN, Technical Report TR-165-P, 1995.

[Dennin87] Dorothy Denning, "An Intrusion-Detection Model," *IEEE Transactions on Software Engineering*, vol. 13, pp. 222--32, 1987.

[Diaz00] Daniel Diaz, "GNU Prolog - A Native Prolog Compiler with Constraint Solving over Finite Domains v1.2.1," http://www.gnu.org/software/prolog, 2000.

[DLAR91] P. Dasgupta, R. J. LeBlanc, M. Ahmad, and U. Ramachandran, "The Clouds Distributed Operating System," *IEEE Computer*, vol. 24, pp. 34--44, 1991.

[Dobson89] John Dobson, "Modeling real-world issues for dependable software," in *High-integrity Software*, C. T. Sennett, Ed. London: Pitman, 1989, pp. 274--316.

[EcViKe00] Steven T. Eckmann, Giovanni Vigna, and Richard A. Kemmerer, "STATL: An Attack Language for State-based Intrusion Detection," presented at ACM Workshop on Intrusion Detection Systems, Athens, Greece, http://www.cs.ucsb.edu/~rsg/pub/2000_eckmann_vigna_kemmerer_wids00.ps.gz, 2000, pp. 16.

[ElmNav94] Ramez Elmasri and Shamkant B. Navathe, *Fundamentals of Database Systems*, second ed. Redwood City: The Benjamin/Cummings Publishing Company, Inc., 1994, ISBN 0-8053-1753-8.

[EsSaPi95] M. Esmaili, R. Safavi-Naini, and J. Pieprzyk, "Computer Intrusion Detection: A Comparative Survey," Center for Computer Security Research, University of Wollongong, Wollongong, NSW, Australia, Technical Report 95-07/06, 1995.

[GafUlv01] John E. Gaffney and Jacob W. Ulvila, "Evaluation of Intrusion Detectors: A Decision Theory Approach," presented at 2001 IEEE Symposium on Security and Privacy, Oakland, CA, 2001.

[Gigand00] Christian Gigandet, "Integration of Host-based Intrusion Detection Systems into the Tivoli Enterprise Console," IBM Research Division, Zurich, Switzerland, Research Report RZ 3253, http://domino.watson.ibm.com/library/cyberdig.nsf/a3807c5b4823c53f85256561006324be/916dfa65ed8fec028525691900313b6e?OpenDocument, 2000.

[Gross97] Andrew H. Gross, "Analyzing Computer Intrusions," Ph.D. Thesis, San Diego, CA: University of California, San Diego Supercomputer Center, 1997, pp. 233.

[HalBau00] L. R. Halme and R.K. Bauer, "AINT Misbehaving: A Taxonomy of Anti-Intrusion Techniques," http://www.sans.org/newlook/resources/IDFAQ/aint.htm, 2000.

[HDLMWW90] L.T. Heberlein, G. Dias, K. Levitt, B. Mukherjee, J. Wook, and D. Wolber, "A Network Security Monitor," presented at IEEE Symposium on Research in Security and Privacy, Oakland, CA, 1990, pp. 296--304.

[Horizo98] Horizon, "Defeating Sniffers and Intrusion Detection Systems," in *Phrack Magazine*, vol. 8, http://www.phrack.org/show.php?p=54&a=10, 1998.

[Howard97] John D. Howard, "An Analysis Of Security Incidents On The Internet," Ph.D. Thesis, Pittsburgh, PA: Canegie Mellon University, Engineering and Public Policy, 1997, pp. 292.

[HRLC01] Joshua W. Haines, Lee M. Rossey, Richard P. Lippmann, and Robert K. Cunningham, "Extending the DARPA Off-Line Intrusion Detection Evaluations," presented at DARPA Information Survivability Conference & Exposition II (DISCEX '01), Anaheim, CA, vol. 1, 2001, pp. 35--45.

[HucWel00] Andrew Hutchison and Marc Welz, "IDS/A: An Interface between Intrusion Detection System and Application," presented at Recent Advances in Intrusion Detection, Third International Workshop, RAID2000, Toulouse, France, http://www.raid-symposium.org/raid2000/Materials/Abstracts/21/21.pdf, 2000, pp. 13.

[IANAPN] Internet Assinged Number Authority (IANA), "Port Numbers," http://www.iana.org/assignments/port-numbers, last update: April 23, 2002.

[ICAT] Peter Mell, Elizabeth Boteler, Derek Dye, Michael Reilly, and David Marks, "ICAT Metabase," http://icat.nist.gov/.

[IcSeVo95] D. Icove, K. Seger, and W. VonStorch, *Computer Crime: A Crimefighter's Handbook*. Sebastopol, CA: O'Reilly & Associates, Inc., 1995, ISBN 1-56592-086- 4.

[IlKePo95] Koral Ilgun, Richard A. Kemmerer, and Phillip A. Porras, "State Transition Analysis: A Rule-Based Intrusion Detection Approach," *IEEE Transactions on Software Engineering*, vol. 21, pp. 181--99, 1995.

[InAlert01] Symantec, "Symantec Intruder Alert v3.6," http://enterprisesecurity.symantec.com/products/products.cfm?ProductID=48&PID=8824150, 2001.

[ISSNet99] ISS, "RealSecure Network Sensor v3.0," Internet Security Systems Inc. (ISS), http://www.iss.net/securing_e-business/security_products/intrusion_detection/realsecure_networksensor . 1999.

[ISSSca99] Commercial Product, "Internet Scanner v5.8," Internet Security Systems Inc. (ISS), http://www.iss.net/, 1999.

[ISSSer00] ISS, "RealSecure Server Sensor," Internet Security Systems Inc. (ISS), http://www.iss.net/securing_e-business/security_products/intrusion_detection/realsecure_serversensor/, 2000.

[Jackso99] Kathleen Jackson, "Intrusion Detection System (IDS) Product Survey," Los Alamos National Laboratory, Los Alamos, NM, Technical Report LA-UR-99-3883, http://lib-www.lanl.gov/la-pubs/00416750.pdf, 1999.

[JiSiIr00] Jitsu-Disk, Simple Nomad, and Irib, "Project Area52: Delirium Tremens," in *Phrack Magazine*, vol. 10, http://www.phrack.org/show.php?p=56&a=6, 2000.

[JLADGJ93] R. Jagannathan, *et al.*, "System design document: Next-generation intrusion detection expert system (NIDES)," SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025, Tech. Report A007/A008/A009/A011/A012/A014, 1993.

[Julisc00] Klaus Julisch, "Dealing with False Positives in Intrusion Detection," presented at Recent Advances in Intrusion Detection, Third International Workshop, RAID2000, Toulouse, France, http://www.raid-symposium.org/raid2000/Materials/Abstracts/50/Julisch_foils_RAID2000.pdf, 2000.

[Julisc01] Klaus Julisch, "Mining Alarm Clusters to Improve Alarm Handling Efficiency," presented at 17th Annual Computer Security Applications Conference (ACSAC), New Orleans, LA, http://www.acsac.org/2001/abstracts/wed-1030-a-julisch.html, 2001.

[Kendall99] Kristopher Kendall, "A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems," M.S. Thesis: Massachusetts Institute of Technology, Electrical Engineering and Computer Science, http://www.cs.ucf.edu/~allen/security/papers/Kendall99.pdf, 1999, pp. 124.

[KeSpZa00] Florian Kerschbaum, Eugene H. Spafford, and Diego Zamboni, "Using embedded sensors for detecting network attacks," presented at First ACM Workshop on Intrusion Detection Systems, Athens, Greece, http://www.cerias.purdue.edu/homes/zamboni/pubs/wids2000.{pdf|ps}, 2000.

[Knuth89] D. E. Knuth, "The Errors of TEX," *Software - Practice and Experience*, vol. 19, pp. 607--85, 1989.

[KrSpTr98] Ivan V. Krsul, Eugene Spafford, and Mahesh Tripunitara, "Computer Vulnerability Analysis," COAST Laboratory, Purdue University, West Lafayette, IN, Technical Report 1998.

[Krsul98] Ivan V. Krsul, "Software Vulnerability Analysis," Ph.D. Thesis: Purdue University, Computer Sciences Department, 1998, pp. 171.

[Kumar95] Sandeep Kumar, "Classification and Detection of Computer Intrusions," Ph.D. Thesis, West Lafayette, IN: Purdue University, Computer Sciences Department, ftp://coast.cs.purdue.edu/pub/COAST/papers/kumar-intdet-phddiss.ps.Z, 1995.

[KumSpa95] Sandeep Kumar and Eugene Spafford, "A Taxonomy of Common Computer Security Vulnerabilities based on their Method of Detection," COAST Laboratory, Purdue University, West Lafayette, IN, Technical Report 1995.

[KYYOS95] S. Klinger, S. Yemini, Y. Yemini, D. Ohsie, and S. Stolfo, "A Coding Approach to Event Correlation," presented at Fourth IEEE/IFIP International Symposium on Integrated Network Management, Santa Barbara, CA, vol. 4, http://www.cs.columbia.edu/ids/research/keypapers/papers/eventcorrelation/isinm95.pdf, 1995, pp. 266--77.

[LaAvKo92] J. C. Laprie, A. Avizienis, and H. Kopetz (Eds.), *Dependability: Basic Concepts and Terminology*, vol. 5: Springer Verlag, 1992, ISBN 3-211-82296-8.

[Lager96] Mark Lager, "Spinning a Web Search," http://www.library.ucsb.edu/untangle/lager.html, 1996.

[Laprie98] J.-C. Laprie, *et al.*, "Dependability Handook," LAAS-CNRS, Report 98346, 1998.

[LBMW94] Carl E. Landwehr, Alan R. Bull, John P. McDermott, and William S. Choi, "A Taxonomy of Computer Program Security Flaws," Information Technology Division, Naval Research Laboratory, Washington, D.C., WA 20375-5337, 1994.

[LCRM98] Douglas J. Landoll, Diann A. Carpenter, Christopher J. Romeo, and Suzanne S. McMillion, "AIX Version 4.3.1 TCSEC Evaluated C2 Security," Arca Systems, TTAP Evaluation Facility, Final Report CSC-FER-98-004, http://www.radium.ncsc.mil/tpep/library/fers/CSC-FER-98-004.pdf, 1998.

[LFGHKM00] R. Lippmann, *et al.*, "Evaluating Intrusion Detection Systems: The 1998 DARPA Off-Line Intrusion Detection Evaluation," presented at DISCEX'00 - DARPA Information Survivability Conference & Exposition, Hilton Head, SC, vol. 2, 2000, pp. 12--26.

[LHFKD00] Richard Lippmann, Joshua W. Haines, David J. Fried, Jonathan Korba, and Kumar Das, "Analysis and Results of the 1999 DARPA Off-Line Intrusion Detection Evaluation," presented at Third Intl. Workshop on Recent Advances in Intrusion Detection (RAID2000), Toulouse, published in LNCS, vol. 1907, 2000, pp. 162--82.

[LinJon97] Ulf Lindqvist and Erland Jonsson, "How to Systematically Classify Computer Security Intrusions," presented at IEEE Symposium on Security & Privacy, Oakland, CA, http://www.ce.chalmers.se/staff/jonsson/publ97-.html, http://www.ce.chalmers.se/staff/ulfl/pubs/sp97ul.pdf, 1997, pp. 154--63.

[Longst97] T. Longstaff, "Update: CERT/CC Vulnerability Knowledgebase," presented at DARPA workshop, Savannah, GA, 1997.

[LSMTTF98] Peter A. Loscocco, Stephen D. Smalley, Patrick A. Muckelbauer, Ruth C. Taylor, S. Jeff Turner, and John F. Farrell, "The Inevitability of Failure: The Flawed Assumptions of Security in Modern Computing Environments," National Security Agency, 1998.

[Lunt88] T. F. Lunt, "Automated audit trail analysis and intrusion detection: A survey," presented at 11th National Computer Security Conference, Baltimore, MD, 1988, pp. 65--73.

[Lunt90a] Teresa F. Lunt, "IDES: An Intelligent System for Detecting Intruders," presented at Symposium of Computer Security, Threat and Countermeasures, Rome, Italy, 1990.

[ManChr99] David E. Mann and Steven M. Christey, "Towards a Common Enumeration of Vulnerabilities," presented at 2nd Workshop on Research with Security Vulnerability Databases, Purdue University, West Lafayette, IN, http://cve.mitre.org/docs/towards.ps, 1999.

[Marrad90] Alberto Marradi, "Classification, Typology, Taxonomy," *Quality & Quantity*, vol. XX, pp. 129-57, http://www.unibo.edu.ar/marradi/classqq.pdf, http://www.kluweronline.com/issn/0033-5177, 1990.

[Marty02] Raffael Marty, "Thor - A Tool to Test Intrusion Detection Systems by Variations of Attacks," Diploma Thesis, Zurich, Switzerland: Swiss Federal Institute of Technology (ETH), Institut für Technische Informatik und Kommunikationsnetze (TIK), http://www.raffy.ch/projects/ids/thor.pdf, 2002.

[MatAvi70] Francis Mathur and Algirdas Avizienis, "Reliability analysis and architecture of a hybrid-redundant digital system: Generaized triple modular redundancy with self repair," presented at AFIPS (American Federation for Information Processing), Atlantic City, NJ, 1970, pp. 375--83.

[Maxion98] Roy A. Maxion, "Measuring Intrusion-Detection Systems," presented at 1st International Workshop on Recent Advances in Intrusion Detection (RAID98), Louvain la Neuve, Belgium, http://www.raid-symposium.org/raid98/Prog_RAID98/Full_Papers/maxion.pdf, 1998.

[MaxTan00] Roy A. Maxion and Kymie M.C. Tan, "Benchmarking Anomaly-Based Detection Systems," presented at First International Concerence on Dependable Systems & Networks, New York, http://www-2.cs.cmu.edu/afs/cs.cmu.edu/user/maxion/www/pubs/maxiontan00.pdf, 2000, pp. 623--30.

[McHugh00] J. McHugh, "The Lincoln Laboratories Intrusion Detection System Evaluation: A Critique," presented at DISCEX'00 - DARPA Information Survivability Conference & Exposition, Hilton Head, SC, 2000.

[McHugh00b] John McHugh, "The 1998 Lincoln Laboratory IDS Evaluation: A Critique," presented at Third Intl. Workshop on Recent Advances in Intrusion Detection (RAID2000), Toulouse, published in LNCS, vol. 1907, 2000, pp. 143--61.

[McHugh01] John McHugh, "Intrusion and intrusion detection," *International Journal of Information Security*, vol. 1, pp. 14--35, http://link.springer.de/link/service/journals/10207/papers/1001001/10010014.pdf, 2001.

[MCZH99] St. Manganaris, M. Christensen, D. Zerkle, and K. Hermiz, " A Data Mining Analysis of RTID Alarms," presented at Second International Workshop on Recent Advances in Intrusion Detection (RAID'99), West Lafayette, IN, http://www.raid-symposium.org/raid99/PAPERS/Manganaris.pdf, 1999.

[Mounji97] Abdelaziz Mounji, "Languages and Tools for Rule-Based Distributed Intrusion Detection," Ph.D. Thesis: Facultés Universitaires Notre-Dame de la Paix Namur, Belgium, Computer Science Department, 1997.

[MWSKHH90] N. McAuliffe, D. Wolcott, L. Schaefer, N. Kelem, B. Hubbard, and T. Haley, "Is your computer being misused? A survey of current intrusion detection system technology," presented at Sixth Computer Security Applications Conference, 1990, pp. 260--72.

[MySql] MySQL AB, "MySQL Database," http://www.mysql.com/, 2000.

[Nessus00] Renaud Deraison, "Nessus," http://www.nessus.org/intro.html, 2000.

[Neuman95] Peter G. Neumann, *Computer-Related Risks*. Reading, MA: ACM Press and Addison-Wesley, 1995, ISBN 0-201-55805-X.

[Neuman98] Peter G. Neumann, "Practical Architectures for Survivable Systems and Networks," Computer Science Laboratory, SRI International, Menlo Park, CA, Technical Report http://www.csl.sri.com/~neumann/private/arldraft.{pdf|ps}, October 1998.

[Neuman98b] Peter G. Neumann, "Illustrative Risks to the Public in the Use of Computer Systems and Related Technology," Computer Science Laboratory, SRI International, Menlo Park, CA, Technical Report ftp://ftp.csl.sri.com/pub/users/neumann/illustrative.{pdf|ps}, October 1998.

[NeuPar89] Peter G. Neumann and Donn B. Parker, "A Summary of Computer Misuse Techniques," presented at 12th National Computer Security Conference, Baltimore, MD, 1989, pp. 396--407.

[NIAP97] National Institute of Standards and Technology (NIST) and National Security Agency (NSA), "NIAP - National Information Assurance Partnership," http://niap.nist.gov/, 1997.

[NSA98] National Security Agency (NSA), "NSA Glossary of Terms Used in Security and Intrusion Detection," http://www.sans.org/newlook/resources/glossary.htm, 1998.

[OMED92] *The Oxford Modern English Dictionary*: Oxford University Press, 1992.

[OstWey84] T. Ostrand and E. Weyuker, "Collecting and Categorizing Software Error Data in an industrial Environment," *The Journal of Systems and Software*, vol. 4, pp. 289--300, 1984.

[ParBus88] M. Paradies and D. Busch, "Root Cause Analysis at Savannah River Plant," presented at IEEE Conference on Human Factors and Power Plants, 1988, pp. 479--83.

[Paxson98] Vern Paxson, "Bro: A System for Detecting Network Intruders in Real-Time," presented at 7th USENIX Security Symposium, San Antonio, TX, http://www-nrg.ee.lbl.gov/nrg-papers.html, 1998.

[Paxson99] Vern Paxson, "Bro: A System for Detecting Network Intruders in Real-Time," *Computer Networks*, vol. 31, pp. 2435--63, 1999.

[PCOM97] N. Puketza, M. Chung, R.A. Olsson, and B. Mukherjee, "A software platform testing intrusion detection systems," *IEEE Software*, vol. 14, pp. 43--51, http://seclab.cs.ucdavis.edu/papers/pdfs/np-mc-97.pdf, 1997.

[PeBiFo54] W. Wesley Peterson, T. G. Birdsall, and W. C. Fox, "The theory of signal detectability," *IEEE Transactions on Information Theory*, vol. IT-4, pp. 171-212, 1954.

[Perl87] Perl Mongers, "Perl," http://www.perl.org/, 1987.

[PHP] PHP, "PHP - Hypertext preprocessor," http://www.php.net/, 2000.

[phpAdm] phpWizard, "phpMyAdmin - MySQL administration over the web," http://phpwizard.net/projects/phpMyAdmin/, 2000.

[PorKem92] Phillip A. Porras and Richard A. Kemmerer, "Penetration State Transition Analysis: A Rule-Based Intrusion Detection Approach," presented at Eight Computer Security Applications Conference, 1992, pp. 220--9.

[Postle01] Roland Postle, "Serious Pitbull LX Vulnerability," http://www.securityfocus.com/archive/1/172699, 2001.

[Power96] R. Power, "Current and Future Danger: A CSI Primer of Computer Crime & Information Warefare," CSI Bulletin 1996.

[PtaNew98] Thomas H. Ptacek and Timothy N. Newsham, "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection," Secure Networks Inc., 1998.

[PZCMO96] Nicholas J. Puketza, Kui Zhang, Mandy Chung, Biswanath Mukherjee, and Ronald A. Olsson, "A Methodology for Testing Intrusion Detection Systems," *IEEE Trans. On Software Engineering*, vol. 22, pp. 719--29, October 1996.

[RCFRLH01] Lee M. Rossey, Robert K. Cunningham, David J. Fried, Jesse C. Rabek, Richard P. Lippmann, and Joshua W. Haines, "LARIAT: Lincoln Adaptable Real-time Information Assurance Testbed," presented at Fourth International Workshop on Recent Advances in Intrusion Detection (RAID2001), UC Davis, CA, http://www.raid-symposium.org/raid2001/program.html, 2001.

[RFP00] Rain Forest Puppy, "A look at whisker's anti-IDS tactics - Just how bad can we ruin a good thing?," http://www.securityfocus.com/templates/forum_message.html?forum=2&head=670&id=670, 2000.

[Roesch99] Martin Roesch, "Snort: Lightweight Intrusion Detection for Networks," presented at LISA '99: 13th Systems Administration Conference, Seattle, WA, http://www.snort.org/, http://www.usenix.org/publications/library/proceedings/lisa99/full_papers/roesch/roesch.pdf, 1999, pp. 229--38.

[SANS] Consortium, "SANS (System Administration, Networking, and Security) Institute," http://www.sans.org/.

[SasBee00] sasha and beetle, "A Strict Anomaly Detection Model for IDS," in *Phrack Magazine*, vol. 10, http://www.phrack.org/show.php?p=56&a=11, 2000.

[Schnei00] Bruce Schneier, *Secret & Lies: Digital Security in a Networked World*, 1st ed. New York: John Wiley & Sons, inc., 2000, ISBN 0-471-25311-1.

[SecFoc] SecurityFocus Inc., "SecurityFocus," http://www.securityfocus.com. 1999.

[SF2708] SecurityFocus Inc., "MS IIS/PWS Escaped Characters Decoding Command Execution Vulnerability," http://www.securityfocus.com/bid/2708, 2001.

[SGVS99] R. Sekar, Y. Guang, S. Verma, and T. Shanbhag, "A High-Performance Network Intrusion Detection System," presented at ACM Conference on Computer and Communications Security, 1999.

[SinSig01] Thomas Singer and Rolf Sigg, "Smart Intrusion Detection Systems," Diploma Thesis, Zurich, Switzerland: Swiss Federal Institute of Technology (ETH), Institut für Technische Informatik und Kommunikationsnetze (TIK), 2001, pp. 113.

[SKKSSZ] Christoph L. Schuba, Ivan V. Krsul, Markus G. Kuhn, Eugene H. Spafford, Aurobindo Sundaram, and Diego Zamboni, "Analysis of Denial of Serice Attack on TCP," COAST Laboratory, Purdue University, West Lafayette, IN, Technical Report.

[SloBar95a] Ken Slonneger and Barry L. Kurtz, "Appendix A: Logic Programming with Prolog," in *Formal Syntax and Semantics of Programming Langauges: A Laboratory-Based Approach*. Reading, MA: Addison-Wesley, 1995, pp. 610.

[Sobire98] Michael Sobirey, "Michael Sobirey's Intrusion Detection Systems page," http://www-rnks.informatik.tu-cottbus.de/~sobirey/ids.html, November 1998, last update: June 15, 2000.

[Song02] Dug Song, "Fragroute," http://www.monkey.org/~dugsong/fragroute/, 2002.

[Song99] Dug Song, "Fragrouter - network intrusion detection evasion toolkit," Anzen Computing, Manual Page http://www.netflood.net/files/IDS/fragrouter.html, 1999.

[Spaffo88] E. H. Spafford, "The Internet Worm Program: An Analysis," Purdue University, Tech. Report NCSD-TR-823, 1988.

[SpaZam00] Eugene H. Spafford and Diego Zamboni, "Design and implementation issues for embedded sensors in intrusion detection," presented at Third International Workshop on Recent Advances in Intrusion Detection (RAID2000), Toulouse, France, http://www.cerias.purdue.edu/homes/zamboni/pubs/sensors-raid2000.{ps|pdf}, 2000.

[SpaZam00b] Eugene H. Spafford and Diego Zamboni, "Data collection mechanisms for intrusion detection systems," CERIAS, Purdue University, 1315 Recitation Building, West Lafayette, IN, Tech. Report 2000-08, http://www.cerias.purdue.edu/homes/zamboni/pubs/2000-08.{ps|pdf}, 2000.

[Stalli95] W. Stallings, *Network and Internetwork Security Principles and Practice*. Englewood Cliffs, NJ: Prentice Hall, 1995, ISBN 0-02-415483-0.

[Stewar99] Andrew J. Stewart, "Distributed Metastasis: A Computer Network Penetration Methodology," The packet Factory, http://magnificent.sk/d0x/distributed_metastasis.pdf, 1999.

[StHoMc00] Stuart Staniford-Chen, James A. Hoagland, and Joseph M. McAlerney, "Practical Automated Detection of Stealthy Portscans," presented at CCS IDS Workshop 2000, Athens, Greece, http://www.silicondefense.com/pptntext/spice-ccs2000.pdf, 2000.

[Sundar96] Aurobindo Sundaram, "An Introduction to Intrusion Detection," *ACM Crossroads Student Magazine*, pp. 10, http://www.acm.org/crossroads/xrds2-4/intrus.html, 1996.

[Symantec] Symantec Corporation, "Symantec Corporation," http://www.semantec.com, 2001.

[Tanenb87] A. S. Tanenbaum, *Operating Systems Design and Implementation*: Prentice Hall, 1987.

[Tanenb96] Andrew S. Tanenbaum, *Computer Networks*, 3rd ed: Prentice-Hall Inc., 1996, ISBN 0-13-394248-1.

[Thomas96] Stephen A. Thomas, *IPng and the TCP/IP protocols: implementing the next generation internet*, first ed. New York: John Wiley & Sons, Inc., 1996, ISBN 0-471-13088-5.

[TLFH01] T. Tidwell, R. Larson, K. Fitch, and J. Hale, "Modeling Internet Attacks," presented at IEEE Workshop on Information Assurance and Security, United States Military Academy, West Point, NY, 2001, pp. 54--9.

[Tripw99] Commercial Product, "Tripwire v1.2," Tripwire Security Systems Inc., http://www.tripwiresecurity.com/, 1999.

[TRM00] Tivoli Systems, "Tivoli SecureWay Risk Manager, User's Guide v3.7," IBM Corp., http://www.tivoli.com/products/index/secureway_risk_mgr/, 2000.

[VeRaGl01] Iris Vessey, V. Ramesh, and Robert L. Glass, "A Unified Classification System for Research in the Computing Disciplines," Indiana University, Technical Report TR107-1, http://www.bus.indiana.edu/ardennis/wp/tr107-1.doc, 2001.

[ViEcKe00] G. Vigna, S.T. Eckmann, and R.A. Kemmerer, "The STAT Tool Suite," presented at DISCEX'00 - DARPA Information Survivability Conference & Exposition, Hilton Head, SC, http://www.cs.ucsb.edu/~vigna/pub/vigna_eckmann_kemmerer_discex00.ps.gz, 2000.

[VMware00] Inc. VMware, "VMware Workstation v2.0," http://www.vmware.com/, 2000.

[Walder01a] The NSS Group, "Intrusion Detection Systems Group Test," B. Walder, Ed., Cambridgeshire, UK, http://www.nss.co.uk/ids/IDS%20Group%20Test%20Report%20Edition%202.pdf, 2001.

[Walder01b] The NSS Group, "Vulnerability Assessment Group Test," B. Walder, Ed., Cambridgeshire, UK, http://www.nss.co.uk/va/VA%20Group%20Test%20Report%20Edition%202.pdf, 2001.

[WanYan01] Tao Wan and Xue Dong Yang, "IntruDetector: A Software Platform for Testing Network Intrusion Detection Algorithms," presented at 17th Annual Computer Security Applications Conference (ACSAC), New Orleans, LA, http://www.acsac.org/2001/abstracts/wed-1030-a-wan.html, 2001.

[WDDN98] Andreas Wespi, Marc Dacier, Hervé Debar, and Mehdi M. Nassehi, "Audit Trail Pattern Analysis for Detecting Suspicious Process Behavior," presented at RAID 98, Workshop on Recent Advances in Intrusion Detection, Louvain-la-Neuve, Belgium, http://www.raid-symposium.org/raid98/Prog_RAID98/Table_of_content.html, 1998.

[Weber98] Daniel Weber, "A Taxonomy of Computer Intrusions," M.S. thesis, Cambridge, MA: Massachusetts Institute of Technology, 1998.

[WeDaDe00] Andreas Wespi, Marc Dacier, and Hervé Debar, "Intrusion Detection Using Variable-Length Audit Trail Patterns," presented at Third International Workshop on Recent Advances in Intrusion Detection (RAID2000), Toulouse, France, published in LNCS. vol. 1907, http://link.springer.de/link/service/series/0558/bibs/1907/19070110.htm, 2000, pp. 110--30.

[Weinma98] William E. Weinman, "About Web Server Logs: Common Log Format," http://www.weinman.com/wew/log-talk/clf.html, 1998.

[Weiss97] Scott Weiss, "Glossary for Information Retrieval," http://www.cs.jhu.edu/~weiss/glossary.html, 1997, last update: 21.1.1997.

[WesDeb99] Andreas Wespi and Hervé Debar, "Building an Intrusion-Detection System to Detect Suspicious Process Behavior," presented at RAID 99, Workshop on Recent Advances in Intrusion Detection, West Lafayette, IN, 1999.

[Wilkis02] Michael Wilkison, "Intrusion Detection FAQ: How to evaluating Network Intrusion Detection Systems?," http://www.sans.org/resources/idfaq/eval_ids.php, 2002.

[WooErl01] M. Wood and M. Erlinger, "Intrusion Detection Message Exchange Requirements," http://www.ietf.org/internet-drafts/draft-ietf-idwg-requirements-05.txt, 2001, last update: February 20, 2001.

[Zalews01] Michal Zalewski, "Delivering Signals for Fun and Profit - Understanding, exploiting and preventing signal-handling related vulnerabilities," http://www.securityfocus.com/archive/1/187124, 2001, last update: May 16, 2001.

[Zambon01] Diego Zamboni, "Using Internal Sensors for Computer Intrusion Detection," Ph.D. Thesis, Purdue, IN: Purdue University, Center for Education and Research in Information Assurance and Security, http://www.cerias.purdue.edu/homes/zamboni/pubs/thesis-techreport.pdf, 2001, pp. 169.

[ZelWal97] Marvin V. Zelkowitz and Dolores Wallace, "Experimental Validation in Software Engineering," presented at Empirical Assessment & Evaluation in Software Engineering, Keele University, Staffordshire, U.K., http://hissa.nist.gov/exper/ease.html, 1997.