Project IST-1999-11583

Malicious- and Accidental-Fault Tolerance for Internet Applications

Towards a Taxonomy of Intrusion Detection Systems and Attacks

MAFTIA deliverable D3

Version 1.01

September 6, 2001

Revisions

Rev.	**Date**	**Comment**
0.1		Outline of document
0.2	23.02.2001	Revised outline
0.9	15.06.2001	Major revision
0.9.1	26.06.2001	Minor corrections. Modified section 2.1
0.9.2	2.7.2001	Minor corrections
0.9.3	24.7.2001	Brought the document in sync with D2. Almost final.
0.9.4	1.8.2001	Added information source taxonomy
1.0	23.8.2001	Corrected the English and all the references corrupted by word
1.01	6.9.2001	Added reference to IBM Research Report RZ 3366

Editor

Dominique Alessandri

Contributors

Christian Cachin

Marc Dacier

Oliver Deak

Klaus Julisch

Brian Randell (University of Newcastle upon Tyne)

James Riordan

Andreas Tscharner

Andreas Wespi

Candid Wüest

Address

IBM Research
Zurich Research Laboratory
Säumerstrasse 4
CH-8803 Rüschlikon
Switzerland

Research Report RZ 3366, IBM Research, Zurich Research Laboratory

Table of contents

Chapter 1: Introduction

In recent years, an increasing number of intrusion-detection systems (IDSes) have become available [Sobire98]. This development has been driven, among other things, by the growing number of computer security incidents [CIN0799, Gross97, Howard97, Kumar95, LSMTTF98, Neuman98b, NeuPar89] which have highlighted the need for organizations to protect their networks against adversaries [Sundar96]. The issue of protecting networks and making them secure and reliable has been addressed in many publications, which 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. However, by generating false alarms or not recognizing attacks, IDSes may fail. This, together with the fact that today's networks are not only distributed but also highly heterogeneous, makes it desirable to deploy multiple instances of diverse IDSes in order to achieve adequate protection of such networks. Last but not least, an *ID architecture* embodying multiple IDSes has to achieve adequate compliance with an organization's security policy and should itself be tolerant to intrusions.

1.1 Motivation

This work pursues two goals. The first is the reduction in the number of alarms a human security officer has to handle. This goal is motivated by the fact that IDSes tend to generate numerous *alarms* (reports of suspicious activities) that need to be collected and analyzed. This is an issue because a substantial number of these alarms (up to 99% and more for some IDSes) are false alarms, and these IDSes may still miss real attacks [DCWMS99, LFGHKM00]. Experience has shown that the processing of IDS alarms becomes even more challenging when considering a large-scale deployment of IDSes.

The second goal is long-term and is to provide a framework that allows the efficient, reliable, and intrusion tolerant operation of a large-scale ID architecture as foreseen within the MAFTIA context. This document describes a foundation for the evaluation of IDSes in terms of their strengths and weaknesses. These evaluation results will allow us to validate and improve ID-architecture designs so that we can identify measures to process and interpret IDS alarms (fault diagnosis), maximize the coverage in terms of ID functionality, and eliminate common failure modes–thereby making the ID architecture tolerant to intrusions.

More specifically, the acquired knowledge will enable us to develop ID alarm correlation rules that allow us to recognize and eliminate many false alarms and to interpret the semantics of alarms. In this context the detailed knowledge of an IDS' strengths and weaknesses is highly relevant.

Like systems in general, IDSes can be evaluated in various ways, such as benchmarking or modeling. We feel that benchmarking real IDSes [DCWMS99, LFGHKM00, LHFKD00] is not generic and systematic enough for our evaluation needs. John McHugh has criticized the benchmarking approach for exactly these reasons [McHugh00, McHugh00b]. Because of these insufficiencies we are investigating another approach which consists of comparing and evaluating IDSes at the level of their specification rather than at the level of their implementation.

1.2 Approach

Our approach describes IDSes by formalizing their characteristics, we do not attempt to describe the implementation of the ID algorithms used. Instead our approach aims at describing the capabilities that result from the algorithms used. These resulting capabilities shall then be used to describe the criteria an IDS has to meet in order to detect a given attack, or more generally, to process activities observed. That is to say we are describing attacks and harmless activities in terms of the IDS characteristics required for their detection, which is what we described in [Alessa00].

One of the advantages of this approach is that it enables us to evaluate a given IDS for its ability to detect a given attack even in the case where the corresponding attack signature has not yet been written for the IDS considered. Furthermore this approach is more generic and requires a relatively limited effort compared to the modeling and description of IDS implementations.

This approach builds upon the novel concept of activities and the description of IDSes. Clearly, the choice of activities and scheme used to describe IDSes is crucial. In this document we describe the results achieved so far. These results build the foundation of our approach to IDS evaluation. They namely include a scheme to describe IDSes and a classification of attacks that allows us to identify a representative set of activities.

While doing the work described here, the experience and knowledge we have gained maintaining IBM's extensive vulnerability database VulDa for several years, has proven highly valuable. In addition we were able to use the database to validate results achieved and described here by a classification of attacks.

1.3 Contributions

Although the ID architecture just mentioned profits from concepts developed within the MAFTIA context, this work contributes to MAFTIA a taxonomy of detectors for errors that may lead to security failure and for security failures. A clear understanding of these detectors is important when building a dependable distributed system–especially when dealing with malicious faults.

In addition, the work presented here contributes several items to the field of ID that are likely to build the foundation for further work in the domain of IDS evaluation.

The first item, a systematic scheme to describe IDSes, is novel to the field of ID. Thus far IDSes have been characterized and described based only on benchmarks [LFGHKM00, LHFKD00] and product descriptions [Jackso99]. It also goes further than existing taxonomies of IDSes [Axelss00, DeDaWe00, DeDaWe99] by describing IDSes in a much finer granularity.

The second item is a generic taxonomy of activities based on criteria that are directly relevant to the ways IDSes analyze their observations for signs of security threats. This taxonomy is then used to create a classification of attacks. The resulting taxonomy and classification are fundamental to our approach and do not yet exist in a comparable form.

It might be worth mentioning that the items described above aim at building the foundation for further work in this field.

1.4 Outline

We start the description of our work by a discussion of related work and fields in Chapter 2. In the same chapter we also introduce the terminology used and discuss the motivation for the approach proposed in more detail.

In Chapter 3, we introduce and develop so-called activity assumptions that are defined by means of an activity taxonomy that is developed with the classification of activities relevant to security in mind. This enables us to systematically identify the various classes of security threatening or seemingly security threatening activities that can be used for the evaluation of IDSes. The taxonomy developed in this chapter is validated by a classification of attacks that are described in IBM's vulnerability database VulDa. The structure of VulDa is described in Appendix B, whereas the attack classification is extensively discussed in Appendix A.

In Chapter 4 we develop a description scheme for IDSes based on a taxonomy–focusing on capabilities IDSes have.

In Chapter 5 we draw the conclusions based on the work described and provide an outlook on future work–namely the evaluation of IDSes.

Chapter 2: MAFTIA and Related work

In line with the top-level MAFTIA goals, this work contributes to the 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 eighties and got real traction with the occurrence of the Internet worm [Spaffo88] and the seminal work by D. Denning [Dennin87]. The basic concepts were however discussed earlier e.g., Anderson [Anders80].

For the work described in this document the dependability concept known as fault-assumptions [LaAvKo92] is of high relevance. The goal of fault-assumptions is to identify all the possible faults that might be activated within a system. The knowledge of all the possible potential faults is a prerequisite to analyze the expected mean time to failure of a given system, for example.

The dependability field has also developed a concept that supports the modeling of systems [Dobson89] such that the system modeled can be evaluated in a systematic way. This is mainly based on the modeling of a system at several hierarchically dependent levels of abstraction and a model of the communication among the various components.

The combination of those concepts–fault assumptions and system modeling–provides us with a powerful technique that seems to be well suited to systematically evaluate IDSes.

2.1 MAFTIA Terminology

In the MAFTIA deliverable D1 [D1Maf00], Section 3.2, dependability concepts have been discussed with respect to malicious faults. These concepts have then been generalized in the MAFTIA deliverable D2 [D2Maf01] with respect to the concept of a security policy. These generalizations are required because in the security domain, or specifically the ID domain, a common, accepted, and unified terminology does not exist. In addition most existing work, such as the glossary initiated by the NSA (National Security Agency) [NSA98], is not based on concepts as rigorous as the ones used in the dependability field i.e., by MAFTIA.

In order to make this document self-contained, we are summarizing the definitions made in MAFTIA D2 [D2Maf01] and MAFTIA D1 [D1Maf00]. We do so by first introducing the concept of security policies, which then serves as a basis for definitions of terms such as *security failure*.

2.1.1 Security policy

The following section is taken largely from Section 3.1 of the MAFTIA deliverable D2 [D2Maf01].

Meaningful discussions on security related topics, often require reference to a *security policy*. Unfortunately, there are many different senses of the term applied to many different levels of abstraction.

In the highest-level sense, the high-level security policy describes the system, the properties it should have, and who (at least in title) is responsible for what; it includes the *security* portion of the *specification* of the systems *function*.

Systems are recursively composed of subsystems; a system's specification recursively determines specifications of the subsystems.

Eventually, through recursive refinement, specialization, and implementation, the systems *behavior* is determined. This process includes the creation of guidelines, processes, site descriptions, practices, specifications, mechanisms, implementations, and configurations.

At the lowest level, we can view the system as a (very large) finite state machine in which the black dot as shown in Figure 1 is the state disallowed by the security policy (i.e., the failure-state).

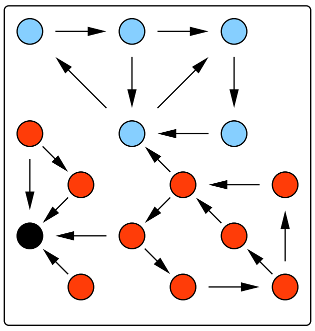


Figure 1 – Low level security policy

At this level, the security policy is the collection of rules according to which the system's security state should evolve (see also Figure 1). In standard cases where one talks of security policies, separate from functional properties, the *security state* can be viewed as a (very large) matrix of subject privileges on object operations. The security policy rules specify the legitimate evolutions of this state.

For our purposes, we will define a security policy as:

1. the *security properties* that are to be fulfilled by the system (high-level views);
2. the *rules* according to which the system security state may evolve (low-level views).

The security policy properties are defined in terms of the security attributes required for the services delivered to the various stakeholders of the system. A *security failure* occurs when a property of the intended[1] security policy is violated. Such a failure may occur in two ways:

1. the rules are incoherent, so a security property violation can occur even if the rules are respected (i.e., the case illustrated in Figures 5 and 3), or
2. the rules are broken (e.g., due to an intrusion or other fault) so the transitions between states are different to those of the state machine inferred from the rules.

In-between the high and low-level views, one can view the system with different levels of resolution. At a slightly higher level than the finite state machine view a system comprised of three different machines, one of which supports two security domains, might look like Figure 2.

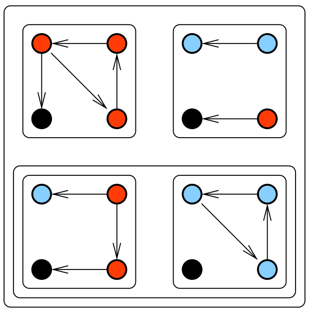


Figure 2 – System with a structured security policy

[1] We add the adjective *intended* to cater for the case where the security policy is incorrectly specified.

In this system, failures of a sub-component are errors for the whole.

For our purposes, it is important to think of the *security policy* as being the recursively structured complete collection of these views. Doing so corresponds naturally with the recursive structure of dependability practices and provides a clean framework in which to discuss security as a whole.

2.1.2 Generalized MAFTIA terminology

This and the following two sections represent a summary of the definitions developed in MAFTIA D1 [D1Maf00] that were then generalized in MAFTIA D2 [D2Maf01]. These generalizations are based on the concept of the security policies just introduced. This work has consecutively resulted in the following definitions:

- *Event*–something that happens or takes place [OMED92]; a change in *state*.
- *Activity*–event or a sequence of *events* within a given context.
- *Activity scope*–defines the context to which a given *activity* applies. Loosely speaking, one can compare activity scopes to the abstraction levels of security policies.
- *Failure*–when the delivered service deviates from fulfilling the intended function.
- *Failure (security ~)*–violation of a security property of the intended *security policy*.
- *Error*–that part of the system state that is liable to lead to failure.
- *Fault* – the adjudged or hypothesized cause of an error.
- *Security policy threatening activity*–activity that represents an external fault which causes an error that may lead to security failure.
- *Attack*–a malicious activity threatening the security policy. It is therefore a malicious external fault.
- *Intrusion*–a malicious *activity threatening the security policy* that leads to a *security failure* i.e., to a security policy violation. It is therefore a malicious external fault that leads to failure.
- *Vulnerability*–an accidental fault, or a malicious or non-malicious intentional fault in the requirements, the specification, the design or the configuration of the system, or in the way it is used. The presence of a vulnerability may enable an *error* to lead to *security failure*.
- *State (system ~)*–a condition of being, with respect to a set of circumstances [LaAvKo92].
- *Alarm*–report of an error that may lead to security failure, optionally including indications whether the error led to *security failure*. The report may include diagnostic information about the fault i.e., the security policy threatening activity that led to the generation of the report.
- *False negative–event* corresponding to the occurrence of an error that may lead to security failure or a *security failure* that is not detected as such. This means that no *alarm* is raised due to either a lack of *coverage* or to excessive latency–also called a *miss*.
- *False positive–event* corresponding to an *alarm* generated in the absence of an error that may lead to security failure or a *security failure* i.e., a false alarm.
- *True positive*–correct generation of an *alarm*. This means that an error that may lead to security failure or a *security failure* has been correctly detected, recognized, and reported.
- *True negative*–correct decision to not rate an *activity* as causing an error that may lead to security failure or a *security failure*.

Please note that a security policy often only defines *security failure* states *explicitly*. Often error states that may lead to security failure are defined *implicitly* only. The definition of what is considered as being an error that may lead to security failure depends on the way the enforcement of the security policy is verified. Loosely speaking, the definition of such errors depends on how sensitive one is with respect to the security policy and consequently how sensitive one is with respect to signs of activities threatening the security policy.

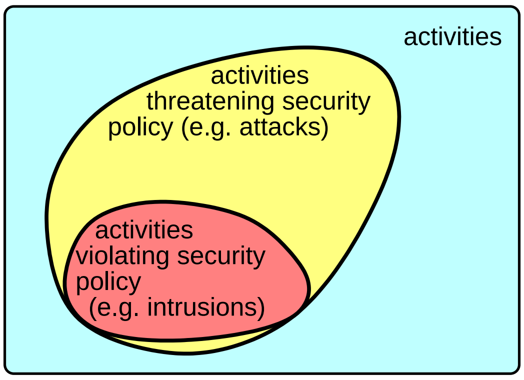

activities
activities threatening security policy (e.g. attacks)
activities violating security policy (e.g. intrusions)

Figure 3 – Activities vs. security policy

Figure 3 illustrates the relation of activities with respect to the security policy.

2.1.3 Intrusion Detection

When discussing ID and IDS one of the first and most important things one should mention is that the term *intrusion detection* is misleading in the sense of its common use.

Clearly the motivation for deploying IDSes is to detect evidence of maliciously intended activities i.e., attacks. However, it is not possible for an IDS to determine or judge the *intent* of the activities it is observing. Instead, IDSes are verifying whether the observed activities are threatening the security policy. In other terms IDSes are detecting errors that may lead to security failure, whereas the capabilities and the configuration of the IDS implicitly define these errors. Furthermore IDSes are performing fault diagnosis by providing information with respect to the activity that caused an error that may lead to security failure.

This discussion naturally leads to the following definitions:

- *intrusion detection*–concerns the set of practices and mechanisms used towards detecting errors that may lead to security failures and security failures (including anomaly and misuse detection) and diagnosing intrusions and attacks.
- *intrusion detection system*–is an implementation of the practices and mechanisms of intrusion detection.

Thus, we can divide the intrusion detection system along dependability lines into *error detection* and *fault diagnosis*.

2.1.3.1 Error Detection

While it is the aim of the intrusion detection system to discover the presence of intrusions (which by definition are malicious), the error detection portion of the facility is merely able to observe and analyze states.

The relevance of malice in error detection is simply in setting requirements. If we are able to dependably detect errors caused by malicious faults, then we are implicitly able to dependably detect errors caused by non-malicious faults. The converse is not true.

The nature of the fault causing the error is otherwise not relevant for three reasons:

1. The nature of the adjudged cause of error has no effect on the means used to detect the error.
2. Determination of whether the cause of error has a malicious nature, is not a computational matter (rather, it is a concern of psychology).
3. We would not want to squelch notification of a dangerous error state merely because the adjudged cause was not deemed malicious (thus, not an intrusion).

Hence, the error detection portion of intrusion detection is the observation and analysis of the system toward detecting states that are error-states as defined by the security policy.

Finally it is important to note that often while looking for 'a thing,' one looks for evidence, side-effects, precursors, conduits, and habitats of the thing. As such the definition would naturally include detection of suspicious activities, vulnerability scanning, and configuration-checking as belonging to error detection.

2.1.3.2 Fault Diagnosis

It should be noted that most currently available intrusion detection systems do not include any intrusion diagnosis mechanisms. The explicit recognition of the fact that misuses and anomalies are indeed errors that can be caused by any sort of fault, is an initial result of the MAFTIA project. Indeed, a good intrusion detection system *requires* such a fault diagnosis mechanism to minimize the rate of false alarms caused by errors due to other classes of faults (e.g., design faults in the reference for defining "misuse" or "anomalies," accidental interaction faults such as typing a password incorrectly etc.).

2.1.4 Failure model

This section roughly summarizes and generalizes the failure model as introduced in Section 4.2 of the MAFTIA deliverable D1 [D1Maf00]. This model was then generalized with respect to security policies in MAFTIA D2 [D2Maf01].

The goal of the failure model is to enable us to systematically analyze a given system for its *failure modes* i.e., all the possible ways a system can fail. This is achieved by making so-called *failure assumptions* for every system component [D1Maf00, D2Maf01, LaAvKo92, MeyPra87, PBSVW88, Powell95].

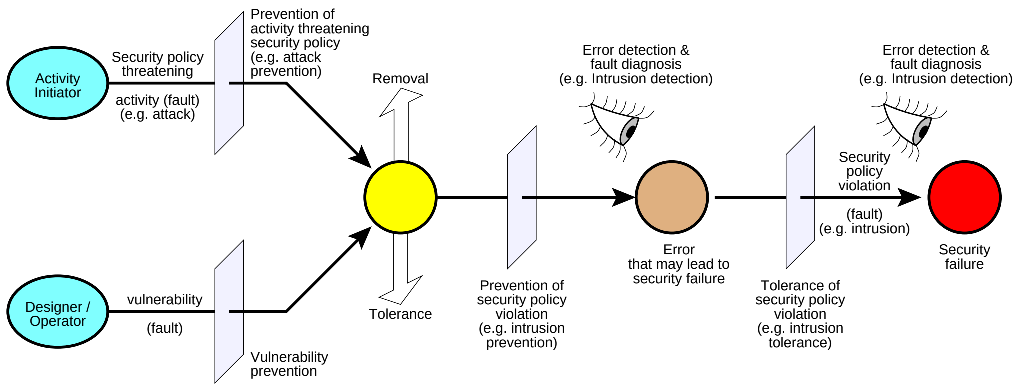

Activity Initiator
Security policy threatening
activity (fault) (e.g. attack)
Prevention of activity threatening security policy (e.g. attack prevention)
Removal
Tolerance
Error detection & fault diagnosis (e.g. Intrusion detection)
Error detection & fault diagnosis (e.g. Intrusion detection)
Designer / Operator
vulnerability
(fault)
Vulnerability prevention
Prevention of security policy violation (e.g. intrusion prevention)
Error that may lead to security failure
Tolerance of security policy violation (e.g. intrusion tolerance)
Security policy violation
(fault) (e.g. intrusion)
Security failure

Figure 4 – The generalized composite failure model of MAFTIA

Figure 4 shows the failure model as defined in MAFTIA D2 [D2Maf01]. This model reflects the fact that an attack can only be successful i.e., violate the security policy, if the corresponding vulnerability is present and if no additional precautions have been taken to prevent the security policy violation.

2.2 Taxonomies

Before starting the discussion on IDS attack and vulnerability taxonomies and classifications, we have to define the terms taxonomy and classification:

- *Classification*–systematic arrangement, in groups or categories, according to established criteria [Webster].
- *Taxonomy*–the study of the general principles of scientific classification [Webster].

We believe this definition to be necessary as the two terms are often confused–especially in existing attack and vulnerability taxonomies and classifications. An extensive discussion of taxonomies can be found in Krusl's Ph.D. thesis [Krsul98]. Krsul provides an extensive discussion of taxonomies and classifications in general and with respect to security. In particular, he explains their differences as follows,

> *…a taxonomy includes the theory of classification, including the procedures that must be followed to create classifications, and the procedures that must be followed to assign objects to classes. [Krsul98], p. 22.*

Besides having a specific goal in mind the creator of a classification needs to obey some basic criteria to ensure that the resulting classification is sound. However, the choice of these basic criteria is not unambiguous either. As an example, in [LinJon97] Lindqvist and Jonsson propose a different set of criteria than the one by Howard in [Howard97]. In summary one can say that at least the following criteria should be fulfilled to make a classification sound:

- Mutually exclusive–the categories used have to be mutually exclusive.
- Exhaustive–the classification categories need to cover the entire scope at which the classification is aimed.
- Repeatable–the classification criteria need to be clear and unambiguous such that the classification can be repeated.

These criteria of course imply that the terminology used needs to be clearly defined.

2.2.1 IDS taxonomies

Taxonomies of IDSes and ID related technologies are a rather recent development and date 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, one proposed by Halme and Bauer [HalBau00] are even more recent. However, the classification of IDSes has been common for a while already and has been realized in the form of surveys such as the work by Lunt [Lunt88], Jackson [Jackso99] and others [Amoro99, EsSaPi95, MWSKHH90].

Figure 5 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].

Within the context of this work the most important elements of the taxonomy by Debar et al. are the *audit source location* and even more the *detection method*. The detection method is used to distinguish IDSes into so-called *behavior-based* and *knowledge-based* systems.

Behavior-based systems, also called anomaly detection systems, have no knowledge of specific attacks. However, they have been provided with knowledge of the behavior of the monitored system 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 daemon-watcher by Wespi et al. [WeDaDe00, WesDeb99]. Behavior-based systems have the important advantage that they do not require a database of attack signatures that needs to be kept up-to-date. The 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.

Knowledge-based systems, also called misuse detection systems, operate based on a database of known attack signatures. Whenever they encounter an activity matching a signature that is stored in the database, the corresponding alarm is generated. The advantage of such systems is that their alarms are meaningful e.g., they contain diagnostic information about the cause of the alarm. On the other hand the main drawback of these systems 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, this is a tedious task because new vulnerabilities are discovered on a daily basis. However, most commercial systems used today e.g., [CiscoNR99, ISSNet99], are knowledge-based systems.

The distinction of the audit-source location is an important prerequisite because we shall develop a taxonomy and methodology that enables us to precisely describe the sensor portion of an IDS. As to be explained in more detail in Chapter 4, we describe the characteristics of an IDS with respect to a given scope. We thereby take into account the fact that an IDS may be powerful for a certain application and less powerful in another.

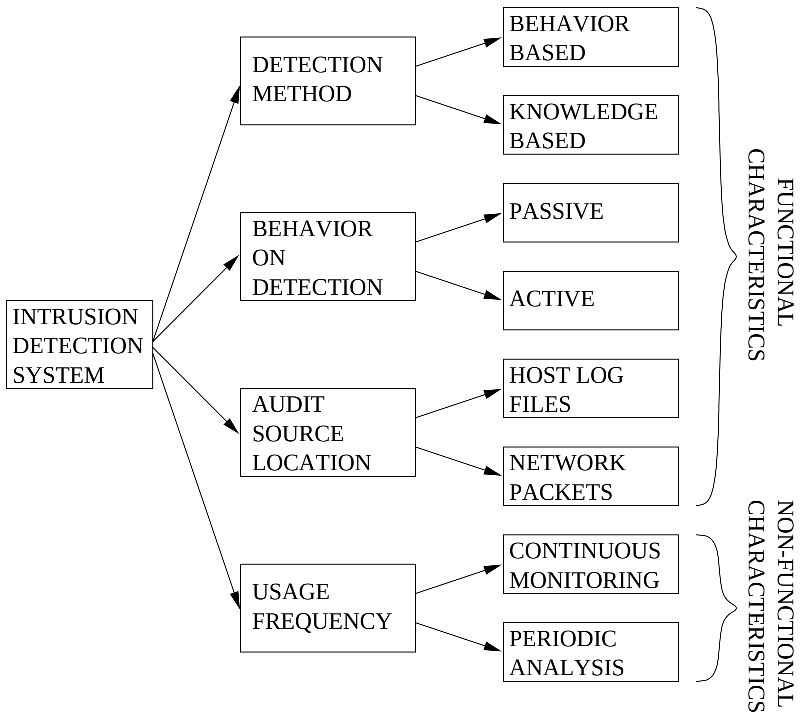


Figure 5 – IDS taxonomy by Debar et al.

It is worth mentioning that the revised IDS taxonomy by Debar et al. [DeDaWe00] as shown in Figure 6 takes into account the *detection paradigm* implemented by the IDS. If the detection paradigm of an IDS is *state based* the IDS is trying to recognize a given system state as being an error state or as being a failure state. *Transition based* IDSes monitor a system for any state transition that represents an attack or an intrusion.

Moreover Debar et al. are refining 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. Furthermore, the addition of IDS sensor alerts reflects the tendency of hierarchical ID architectures, where several IDSes send their alerts to a higher level instance where the alerts are analyzed and possibly aggregated. The resulting alerts are then sent to the next higher instance or are presented to the security officer.

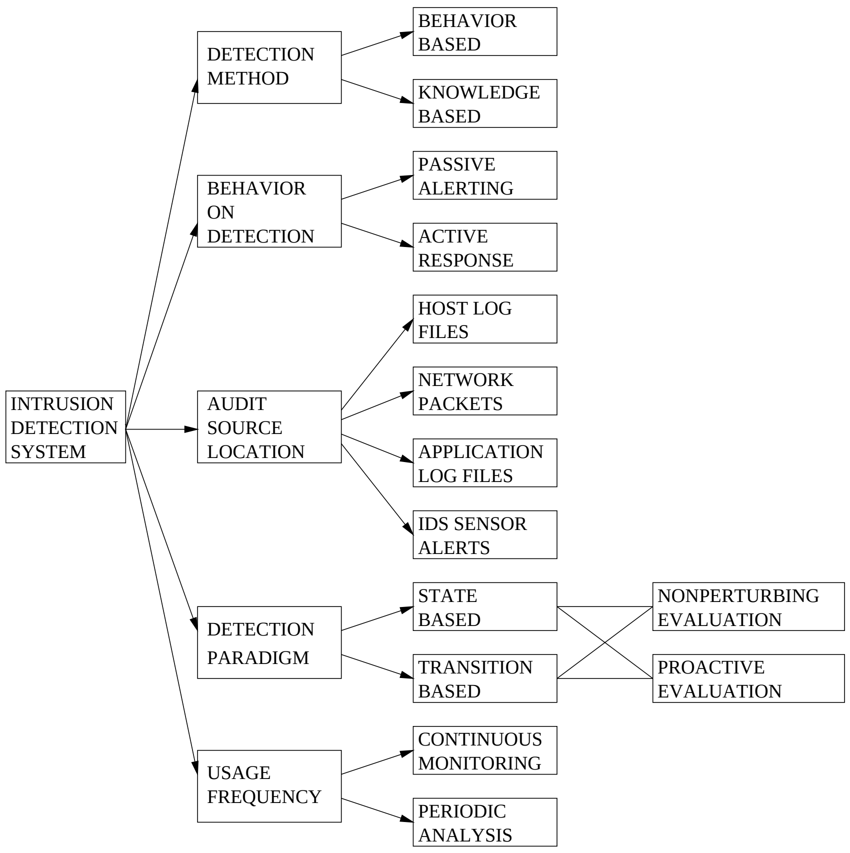


Figure 6 – Revised IDS taxonomy by Debar et al.

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.'

As in Debar et al., Axelsson classified a number of IDSes 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,

> *...this is not because the classification is ambiguous but because the systems employ several different principles of detection. [Axelss00], p. 7.*

This turns out to be an important statement–especially when looking into the detailed description of IDSes envisaged by this work. It shall soon become apparent that this statement also applies to the work presented here.

2.2.2 Attack taxonomies

As mentioned, attack taxonomies and the resulting classifications are of interest to us because we are considering using an attack classification as the foundation for the evaluation of IDSes. Computer security attacks and vulnerabilities have been classified in many ways however, no commonly accepted reference classification exists as yet.

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 properties of attacks are not clearly separated. An example of such classification is the one proposed by Cohen [Cohen95].

One of the earliest works is one by Neumann and Parker [NeuPar89]. In this work the authors classified data from about 3000 incidents which they had collected over 20 years according to nine different computer misuse techniques. Those categories are, as the authors state themselves, not mutually exclusive. Based on [NeuPar89], Neumann came up with 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 is explained later in this document, this is interesting for this work as it comes relatively close to the goal of our classification.

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

- List of terms–a wide range of 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. Example: [Kumar95, KumSpa95]
- Impact–the immediate damage caused by a successful attack i.e., an intrusion. Examples: [CheBel94, Howard97, JiSiIr00, LinJon97, NeuPar89, SinSig01]

It is worth noting 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.

2.2.3 Vulnerability taxonomies

As already indicated, attacks are tightly linked to vulnerabilities. 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 distinguished clearly 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. This is clear 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.

In his Ph.D. thesis Krsul [Krsul98] discusses seventeen different vulnerability classifications. We are not going to reproduce the whole discussion here, rather, we provide an overview of the various classes chosen for those classifications:

- Genesis–The way the fault was introduced. Examples: [AsKrSp96, Aslam95, LBMW94, Longst97].
- Time–When a fault was introduced e.g., design phase, coding phase, maintenance etc. Example: [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, OstWey].
- 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. Example: [KrSpTr98, Power96].

The threat category is highly related to the impact category introduced in Section 2.2.2, as are attacks and vulnerabilities in general. A given attack does not necessarily exploit a given vulnerability in the most malicious way i.e., the threat represented by a given vulnerability is not necessarily fully exploited by an attack attempting to exploit the vulnerability.

2.2.3.1 Enumeration of vulnerabilities

A non-classifying 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 is generating 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, CVE 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 which is operated by the SecurityFocus [SecFoc] web site. CVE entries and Bugtraq ID database records both refer to their respective counterparts.

However, CVE entries, Bugtraq Ids, nor 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 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 in this severe case.*

2.3 Evaluation of IDSes

It can be expected that IDSes will be evaluated and compared based on the set of CVE and Bugtraq IDs that they are able to deal with in the near future. The various enumeration efforts will ease the comparison of IDS, but are not able to take an IDS' quality with respect to failure i.e., false positives and false negatives, into account. An evaluation that is based on IDS features such as the extensive work presented by Jackson [Jackso99] suffers from the same problem, but at least provides good overview for a series of IDSes.

Another more pragmatic approach to evaluate IDSes is the benchmarking of IDSes as done by Lippmann et al. [LFGHKM00, LHFKD00] in the so-called Lincoln Lab Experiment and others [DCWMS99]. A critique of the Lincoln Lab Experiment by McHugh [McHugh00, McHugh00b] shows that the experiment fails to provide a picture of IDSes that is fine grained enough. This is mainly due to limits of testing and the differing signature-sets of IDSes. In fact an IDS that happens to detect most of the attacks tested by the Lincoln Lab Experiment but which otherwise has a small signature base may get a high rating. In contrast, another IDS having a larger and different signature-set and which may be better, from a technical point of view, could get a lower rating because attacks used in the Lincoln Lab Experiment are missing. These attacks are missing simply because the signatures required to detect these attacks are not contained in the signature-set. Among others, these insights led to the proposal that the IDS evaluation method should be developed based on this work.

Existing evaluation techniques are not well suited for the systematic IDS characterization required in the context of this work. As we have seen, existing approaches are either operating at a level too high or too low, do not take the issue of false positives into account at all [Jackso99], or are not generic and systematic enough [LFGHKM00, LHFKD00, McHugh00, McHugh00b]. In other terms existing efforts have not been designed to enable an analysis of IDSes as it is envisaged by the continuation of this work.

2.4 Discussion of and motivation for IDS evaluation

As indicated in Section 1.1, it is the goal of this work to build the theoretical foundation for large-scale intrusion detection architectures. Such architectures are needed as soon as intrusion detection is used to protect large intranets instead of small so-called Demilitarized Zones (DMZ). In that context, and in the near future, each desktop will eventually be instrumented with some host-based ID capability.

When considering such a scenario one can easily identify the following issues that are related to intrusion detection architectures:

- Low sensor quality–It is well-known that most IDSes generate an overwhelming number of false positives–sometimes 99% (or more) of all the alarms are false alarms [Axelss00, Julisc00, MCZH99, Schnei00].
- Sensor diversity–The semantic of alarms as generated by IDSes is sensor-dependent, i.e., it is defined by the generating IDS. Even though a common naming scheme for attacks is in the process of being established [CVE99, DeHuDo00, ManChr99] (see also Section 2.2.3.1) the conditions under which a given IDS is raising a given alarm are not defined. When considering behavior-based IDSes the issue becomes even more challenging as the alarms generated by such systems usually express the fact that an abnormal behavior has been observed–without providing any further information on the type of attack that caused the alarm to be generated.
- Alarm context data diversity–The additional data provided by an IDS along with an alarm vary depending on the type of IDS (e.g., the information source monitored by the IDS) and its implementation.
- IDS placement–In order to achieve the required intrusion detection coverage within a network numerous instances of different IDSes are required. The location of these

IDSes is critical and has to take into account the strengths and weaknesses of every IDS type deployed.

- Management–This type of intrusion detection architecture requires a flexible management infrastructure.

As mentioned in Section 1.1, this work focuses on the issues that concern the processing of alarms as generated by IDSes, in other words all issues listed above but the management one.

One can expect IDSes to mature, but we expect IDSes to continue generating a significant number of false positives. This issue is partially being addressed by the development of highly specialized sensors that focus on very specific problems only and are application specific, such as WebIDS [Almgre99]. However, the introduction of highly specialized IDSes increases the issues of sensor diversity, alarm context data diversity, and sensor placement even further. Moreover, the recognition and identification of complex attacks e.g., attacks involving several services concurrently, requires the deployment of several types of IDSes as well as the concurrent and uniform processing of all the alarms collected.

2.4.1 Fault-masking by means of redundancy and voting

When dealing with system components that are not reliable enough to meet a given system's specification, a standard approach is to introduce redundant components into the system. By letting the redundant components vote on the result it is possible to increase the system's reliability. This has been explored extensively by the dependability community from which MAFTIA is also profiting. However it seems impossible to apply this simple concept to intrusion detection architectures not only because the failure i.e., false-alarm rate, of IDSes is too high [Axelss99].

The problem with the false-alarm rate being too high, is illustrated in the work done by Mathur et al. [MatAvi70], where they prove that one cannot increase a system's reliability simply by adding redundant components, if the failure rate of the single components exceeds 50%.

However, this is not the main reason why the work by Mathur et al. is not applicable to ID. The main reasons why their results do not apply are the different assumptions about systems and system components. In their work they consider systems that are generating a single output for a single input. This does not apply to IDSes because for any given input they may remain mute, generate one alarm, or generate a whole series of alarms. Moreover the semantic of the output generated by the various IDSes varies depending on the type of IDS used, the way the IDS is configured, and the environment within which the IDS is operated.

In addition simple voting mechanisms are not generally applicable. In the case where the IDSes used are of different types, the semantics of the output generated is not uniform. The semantics of alarms may differ across IDSes and at the same time not every IDS generates the same set of alarms for a given attack.

2.4.2 Failure assumption coverage and correlation

As mentioned, it is our long-term goal to find practical solutions to the issues of designing and operating large-scale intrusion detection architectures. A central part of our solution is going to be based on the correlation of alarms, which shall enable us to interpret the semantics of alarms and more specifically groups of alarms. To achieve that goal, we have to develop a methodology that enables us to determine how the different types of IDSes have to be distributed in a given environment. This methodology must be such that the required coverage is guaranteed and moreover that the information provided by all those sensors contains sufficient diversity and redundancy to enable the elimination of false positives by means of correlation.

Considering these requirements, one can intuitively identify the IDSes as being the key elements that determine the way alarm correlation has to be done and the way the IDSes have to be distributed over a given network. We need to obtain a clear understanding of the characteristics of every IDS involved. When referring to IDS characteristics in this context, we are interested in the strengths and weakness of an IDS, or more specifically, in the reaction and the output generated by a given IDS when presented with a given input. Please note that the intent (malicious or non-malicious) of the input generator has no influence on the reaction of the IDS. The intent of an activity is not detectable, but needs to be interpreted in a larger context–most likely also taking other observations into account.

This work represents the foundation of the long-term goal just described. In this work it is shown how a set of assumptions for the evaluation of IDSes can be developed systematically. It is our belief that these assumptions can then be used to analyze the assumption coverage of a set of similar IDSes, as described by Powell [Powell95], in a more generic fashion.

2.4.3 Our approach to IDS evaluation

IDS characteristics can be determined by various means such as the evaluation of IDSes as described in Section 2.3. However, as also explained in Section 2.3, existing evaluation techniques are not well suited for the systematic IDS characterization required in the context of this work [Jackso99, LFGHKM00, LHFKD00, McHugh00, McHugh00b].

Therefore, we propose a different approach to IDS evaluation, which we believe to be sufficiently systematic and generic for our purposes. The proposed approach is three-fold [Alessa00]:

- We describe the capabilities of IDSes in a generic and extensible way.
- We systematically identify a representative input set that IDSes might be confronted with i.e., activities that might be considered as a threat to the security policy..
- We describe every potential input by the set of IDS capabilities required to enable an IDS to analyze and possibly recognize the considered input as an activity threatening the security policy e.g. an attack.

Our approach allows us to predict how a specific IDS would potentially behave when provided with a given input.

Example: *Let us consider the attacks attempting to compromise a system by exploiting a vulnerability present in earlier version of the test-cgi example script* [CA0696, CA0797]. *They can only be detected by an IDS which is capable of monitoring and recognizing http requests. Furthermore, the IDS must be able to apply some kind of pattern matching algorithm to analyze the stream of incoming http requests. This is a simplified example, but if the IDS meets the corresponding requirements it is considered capable to generate the corresponding alert.*

The three main points of this work are addressed separately in a chapter each. Chapter 3 deals with the issue of identifying a representative input set by applying a combination of the concepts of fault-assumptions originating from the dependable system design and core element of MAFTIA [D1Maf00, LaAvKo92]. Chapter 4 addresses IDS descriptions similar to a concept originally developed in Dobson’s work on system modeling [Dobson89].

Dobson’s concept is mainly based on the modeling of a system at several hierarchically dependent abstraction levels and a model of the communication among the various components. He is introducing a methodology that describes a system at five different levels of abstraction. The two highest levels are the *linguistic level* and the *conceptual level* and 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* in order 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. In addition Dobson develops the notion of frame for communication systems by introducing four models for various aspects of the system modeled. He introduces a model of system composition, a model of system behavior, a model of messages, and a model of communication. Each of those models ignores details irrelevant to its purpose.

As mentioned, we are identifying a representative input set based on a combination of the concept of fault-assumptions [LaAvKo92] and Dobson's approach to system modeling by means of frame specific system models. Intuitively the approach assures that all factors and combinations of factors that are influencing an IDS' ability to detect a given attack are taken into account. Such a high degree of coverage can be achieved by first modeling all the system components and functions relevant to the process of ID. In a next step we then systematically combine those models and apply them to a set of pre-defined scopes that we shall develop and define within the same context. This leads to a concept that we shall call *activity assumptions*. In this work the activity assumptions are validated and illustrated by means of an attack classification.

The second main topic addressed in this work is the description of the capabilities IDSes offer for the analysis of activities, including specific attacks. We describe these IDS capabilities by developing a taxonomy that characterizes IDSes by defining various so-called analysis levels and features that are inspired by Dobson's abstraction levels. These taxonomy elements are then superposed to the same set of pre-defined scopes used for the identification of a representative input set. The reuse of the pre-defined scopes for the description of IDSes helps us by assuring the consistency between the input set identification and the IDS description.

As mentioned, the activity assumptions and the IDS descriptions form the basis for the actual evaluation of IDSes. In a future step we shall describe so-called activities that have been identified by means of activity assumptions. Those activity descriptions shall be expressed by the combination of the capabilities required from the IDS to analyze the given activity. The analysis of such activity descriptions then enables us to determine whether an IDS is potentially able to detect a given activity and whether the IDS can be expected to generate any specific alarms.

2.4.4 Attack classifications

Most attack classifications that are based on taxonomies do not meet the criteria for taxonomies as summarized in Section 2.2 (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 caused by 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 rather by the fact that attacks involve 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 lack of attack classifications suitable for the evaluation of IDSes leads us to propose our own, described in the next chapter.

Chapter 3: Activity taxonomy

An important part of the evaluation of IDSes as introduced in 2.4.3 is the identification of a representative input set. The input set must ensure that all the factors relevant for ID are taken into account and exercised during the IDS evaluation process. We address this requirement by developing an *activity taxonomy*. This taxonomy builds the foundation for what we call *activity assumptions*. The concept of activity assumptions guarantees a systematic evaluation of IDSes. In this chapter we develop these activity assumptions by means of an *attack classification* i.e., the classification of malicious activities according to the activity taxonomy.

It is worth mentioning that the concept of activity assumptions is similar to the concept of the so-called fault assumptions as introduced by Laprie et al. [LaAvKo92]. This concept was briefly discussed in Section 2.4.2. A key element of this approach is that one identifies all the factors that are believed to be potential causes of faults. In a next step one then considers the impact those factors and the combination of those factors have in terms of faults.

Our proposal is similar in the sense that we identify all the factors believed to be relevant to the detection process performed by an IDS. This means in this case, that we are identifying all the factors relevant to the detection process of attacks. However, as mentioned in Section 2.4 attacks shall not be the sole input used to evaluate an IDS. As illustrated in Figure 7, we are merely interested in the concept of *activities* that are similar to attacks in the sense that they might cause an IDS to generate false alarms. We consider an *activity* to be an event or a sequence of events that describes an action within a given context.

Examples: *Within the context of http an activity represents an http request; within the context of IP an activity represents an IP Protocol Data Unit (PDU); within the context of a process many activities are conceivable. An example is the creation of a file.*

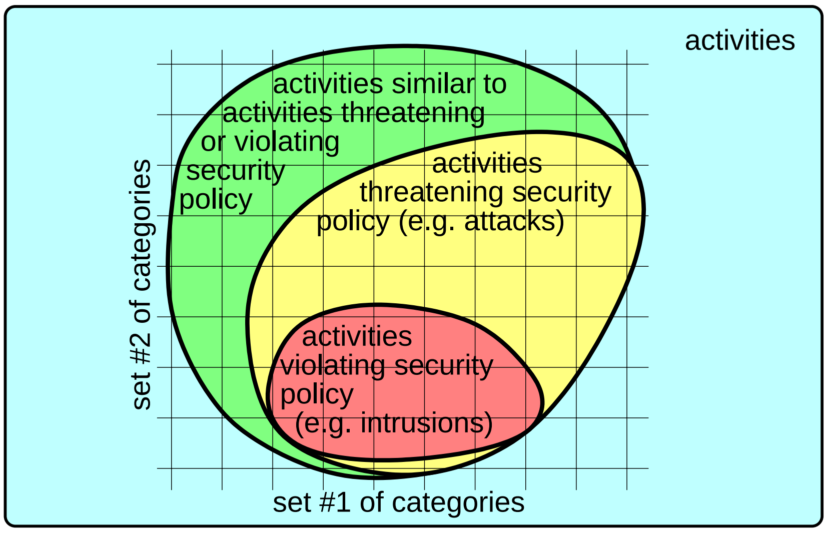

activities
activities similar to activities threatening or violating security policy
activities threatening security policy (e.g. attacks)
activities violating security policy (e.g. intrusions)
set #2 of categories
set #1 of categories

Figure 7 – Identification of activities relevant to the evaluation of IDSes

As illustrated by the grid shown in Figure 7, the goal is to identify an *activity taxonomy* that allows the description of activities based on criteria relevant to the detection process performed by IDSes. As further illustrated in Figure 7, the taxonomy covers not only activities that are threatening the security policy, but also activities in the neighborhood. These surrounding activities are relevant because they are similar to activities threatening or even violating the security policy. The similarity of these activities to undesired activities is manifested by the fact that they may cause IDSes to generate false positives. Please note that it is not possible to clearly identify those surrounding activities because, whether a given activity is causing a false alarm to be generated, depends on the IDS evaluated. However, it seems accurate to state that there is a high probability that the activities identified in the neighborhood of activities threatening or violating the security policy, might cause IDSes to generate false positives.

When identifying the taxonomy categories we have to keep the goals of the taxonomy in mind. This means that the taxonomy categories should reflect activity characteristics relevant to the analysis performed by IDSes. Furthermore the choice of the classification categories has to be consistent with the goals of this taxonomy and should not be too broad, so that the effort remains limited and well focused. In order to achieve this we have to consider IDSes on the one hand and activities on the other. This taxonomy will enable us to classify activities based on criteria that determine an IDS' 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 for example, which are not observable.

In order to achieve these goals we propose a model that takes the various observable aspects of an activity into account. The model proposed is shown in Figure 6.

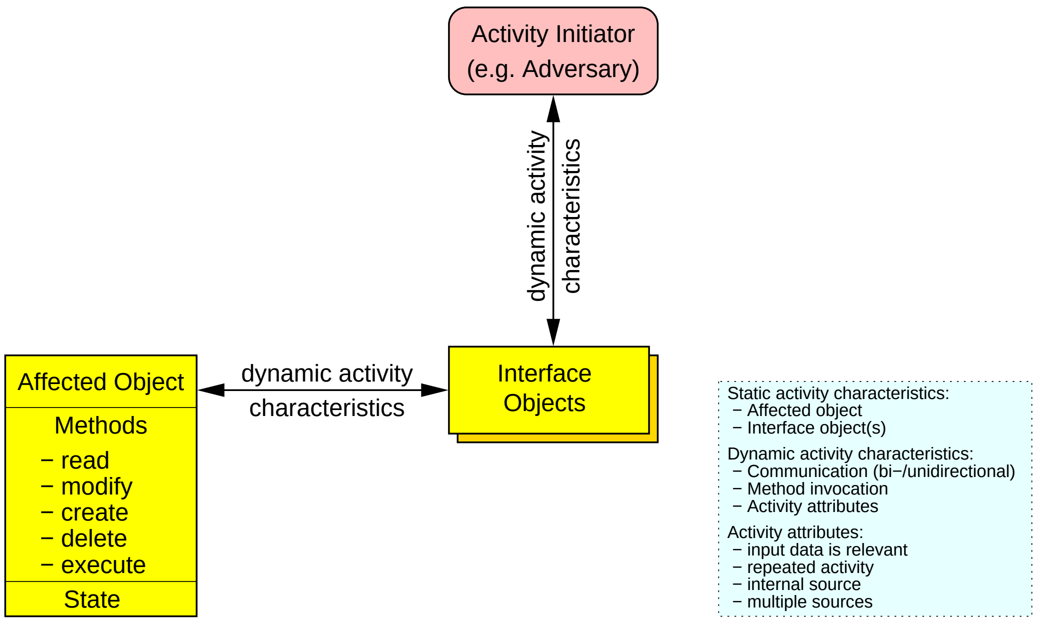


Figure 8 – System model used for activity taxonomy

Real-world activities tend to be quite complex and can be classified according to a large number of criteria. The model shown in Figure 6 permits simplification in the classification of activities down to a level at which one only deals with characteristics observable and relevant to ID.

The model introduced here consists of three types of objects, however, only two of them are relevant for the taxonomy. The *activity initiator* is in most cases an object that is not directly observable. The most typical examples of an activity initiator are a remote system or human being that may act with malicious intent, but does not have to.

We consider the activities initiated by the activity initiator to be potentially[2] observable at any *interface object* involving the activity while interfering with the *affected object*. It is also

[2] The observation of attacks at interface objects is not always possible. For instance, it is not possible to detect attacks on the network if the data transferred is encrypted.

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* that are comprised of one or more interface objects and the affected object. The activity initiator object should be formally considered as being part of the static activity characteristics. However, as mentioned the activity initiator object is generally not directly observable by an IDS, which is why it is not included in the taxonomy to be developed in the following.

Another static element shown in Figure 6 that is not included in the static activity characteristics, is the *internal state* of the affected object. The internal state is not taken into account for the activity taxonomy because this work focuses on real-time IDSes or in other words, IDSes performing continuous monitoring. These are generally transition-based IDSes (see also Figure 6 and [DeDaWe00]). State-based IDSes generally perform a periodic analysis of the system to monitor only. This means that the state of a system i.e., an object, is inspected periodically for erroneous system states that indicate a fault. Typical examples are security scanners such as Nessus [Nessus00], ISS [ISSSca99], Satan etc., and system integrity checkers such as tripwire [Tripw99]. In fact, at the time of writing this document we are not aware of a real-time IDS implementing a state-based detection paradigm.

So far the objects involved in an activity have been introduced and it has been mentioned that they interfere. This interference can be described by so-called *dynamic activity characteristics* that describe the way the objects interfere. We have identified two main sets of characteristics that describe the dynamic portion of an activity. There is on the one hand the invocation of *methods* on the affected object and on the other, the type of *communication* used. Besides these two sets of categories, we have identified a set of additional *attributes* that refine the dynamic activity characteristics and are observable by an IDS.

Please note that the term *vulnerability* was not mentioned here. The fault representing the vulnerability can typically be found in either one of the interface objects, in the affected object itself or in the combination of several objects. However, when analyzing an activity for its visibility of an IDS, the location of the vulnerability is of limited importance only, which is why it does not appear in the system model shown in Figure 6. Moreover, the initiation of an activity threatening the security policy does not require a specific vulnerability to be present, e.g. the activity may represent an unsuccessful attack. However, it is the presence of the corresponding vulnerability that determines whether an activity may lead to security policy violation, e.g. a successful attack or an intrusion.

Before further developing the mentioned notions of static and dynamic activity characteristics we introduce the so-called *activity scope*. The activity scopes identified in the following section will be the foundation for the identification of the interface objects and affected objects to be developed in this chapter. However, the activity scopes are not only relevant to this chapter. The hierarchical aspects of the activity scopes, in particular, shall become even more important in the taxonomy of IDSes developed in the follow-up chapter.

3.1 Activity scope

IDSes are complex systems and so are their components. An IDS may be able to apply a certain type of analysis to a given activity–but does that mean that the IDS is able to perform the very same analysis on all the different activities it is exposed to? Definitely not. Some of the activities may not even be visible to the IDS because the information source monitored simply does not provide the required information. Furthermore, many analysis techniques need to be adapted for every single object analyzed. For example an IDS that is able to analyze a HTTP session in a stateful fashion is not necessarily able to perform a stateful analysis for any other protocol session, such as SMTP sessions (mail). The IDS' ability to analyze an observation may even depend on more subtle differences such as the version of a given protocol. A good example for this is HTTP where the version 1.0 of this popular protocol does not support the processing of multiple request i.e., transactions over the same transport layer connection whereas version 1.1 of HTTP supports consecutive requests being served over the same connection.

These examples demonstrate the need for a taxonomy that describes IDSes down to a very low level of detail. Describing complex IDSes such as current network-based products at such a low level can easily be identified as being a very tedious task. Also, when going a bit further and considering the way we plan to describe activities–we foresee to describe them by the combination of IDS properties required for the detection of a given activity–this does not seem to be a practical solution. We would have to describe identical attacks multiple times–once for every subtle difference among similar activity scopes. This would mean that we need to create a separate activity description for every version of HTTP, for example.

We are addressing this need for high generality on one hand and high specificity on the other by introducing *activity scopes* and the so-called *activity scope tree*. The notion of activity scope defines the context within which IDS capabilities are available and to which a given activity applies.

The activity scope tree defines the dependencies among activity scopes such that we can clearly identify activity scopes as representing a subset of a higher level activity scope. The activity scope tree enables us to describe IDSes in a compact and flexible manner. For instance, this means that it allows us to express the fact that an IDS is able to perform string matching on any application layer protocol which is fairly generic. At the same time we can express the fact that the IDS is able to perform stateful analysis on HTTP version 1.1 only.

As we explain in the following sections we use the activity scope tree not only to classify protocols and system components under the umbrella of some more generic activity scope, we also introduce so-called *functional activity scopes* that are used to classify activity scopes reflecting their inherent functional differences. For instance, transport layer protocols can be distinguished into connection oriented and connection-less protocols. As we shall see, the notion of functional activity scopes enables a detailed description and evaluation of IDSes without the need of going down to the implementation level of highly specific activity scopes, e.g. protocols.

3.1.1 Generic activity scopes

An activity scope tree consists of generic activity scopes that build the foundation for more advanced things such as the mentioned functional activity scopes. Generic activity scopes have the purpose to group more specific activity scopes based on common concepts taken from the system design field.

The generic activity scopes described in the following were derived from common system partitioning and layering concepts such as the OSI model. However, the number of generic activity scopes was kept to a minimum by focusing on ID relevant issues.

Later we will describe the semantics of IDS characteristics with respect to those generic activity scopes. Because the semantics of IDS characteristics have to be consistent within a given generic activity scope, all the objects, protocols etc., covered by a generic activity scope have to conserve those semantics. However, the semantic of IDS characteristics can be defined with respect to any appropriate activity scope–provided the semantics of the given IDS characteristic has not yet been defined on any higher level i.e., a more generic activity scope.

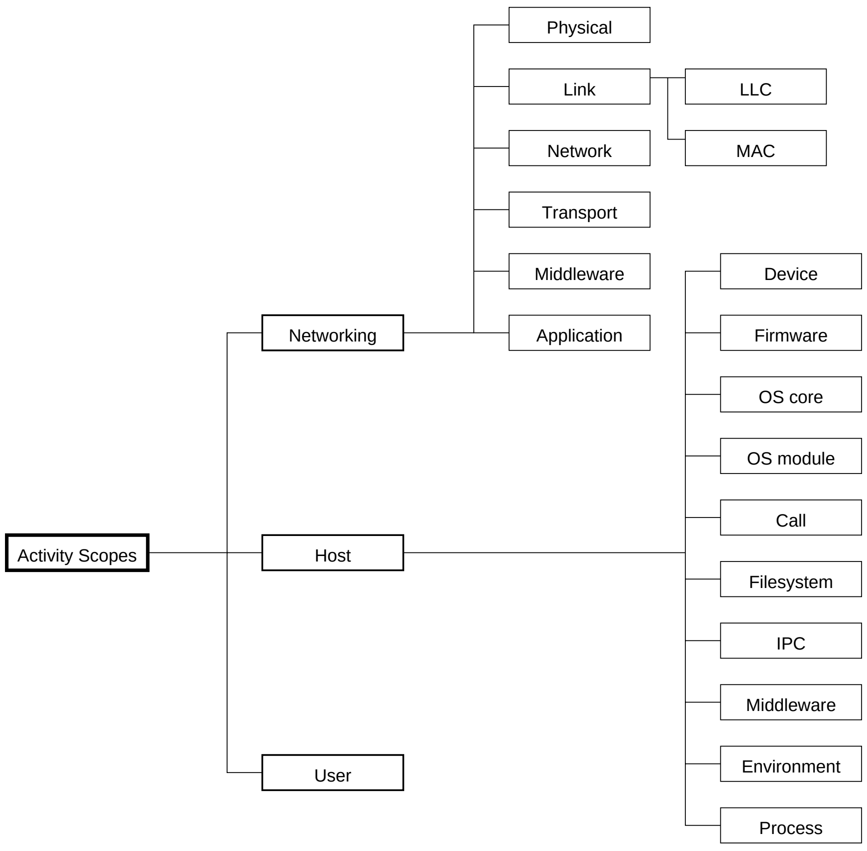


Figure 9 – Activity scope tree with generic activity scopes only

As one can see in Figure 9, we have identified three generic activity scopes that represent the first level of the tree: the user, the host, and the network.

The networking activity scope is further divided into generic activity sub-scopes that roughly correspond to the ISO network stack. We choose not to take the presentation and session layers into account because IDSes generally treat them within the context of the application layer.

The second level activity scopes, identified for the host activity scope, consist of activity scopes that are of interest for IDSes. However, it is less straightforward to identify the activity scopes on the host level because a layering concept as it exists for the network stack does not exist here. We therefore focused on objects that are of interest to IDSes and that are observable.

The middleware activity scope is split into a portion that can be used to describe an IDS' characteristics with respect to activities observable on the network and a second portion that covers middleware functionality used on the host only.

3.1.2 Specific activity scopes

The generic activity scope tree just developed can be extended with more specific activity scopes that, generally speaking, represent implementations of a generic activity scope. Examples of such implementations are protocols or the various types of filesystem objects. However, our goal is to describe IDSes and later activities using generic activity scopes wherever possible.

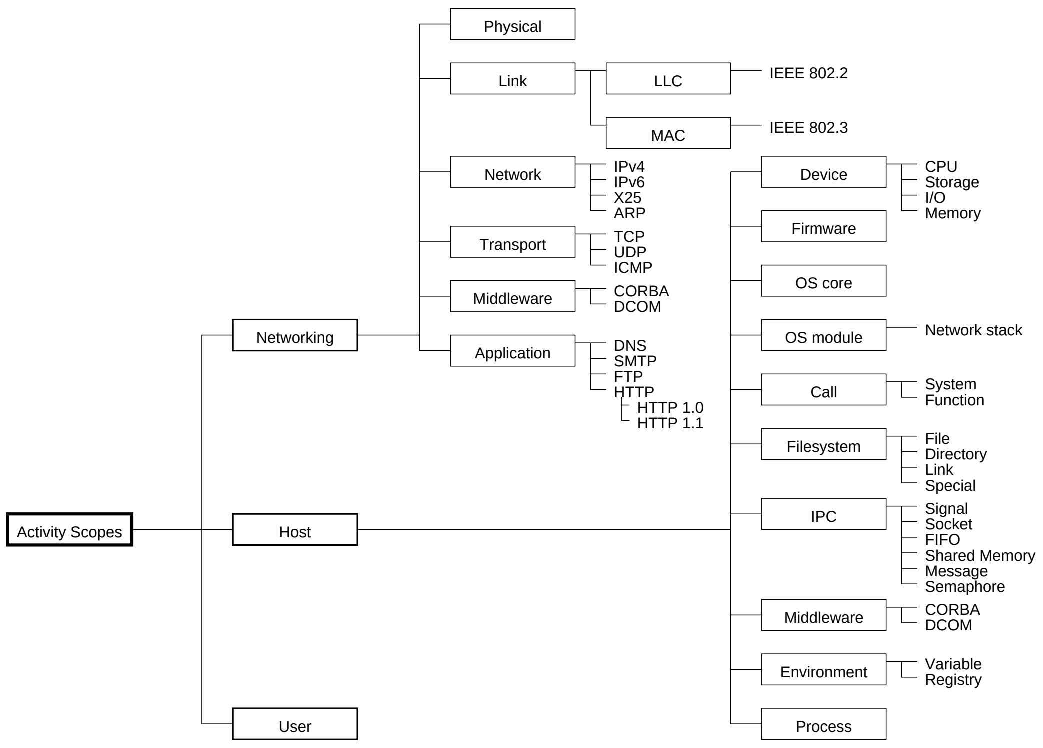


Figure 10 – Activity scope tree including specific activity scopes

The tree shown in Figure 10 can be extended if needed. For instance, further protocols may be added as needed.

3.1.2.1 Networking related activity scopes

As a first step we are considering the *networking activity scopes* used to describe networking related IDS characteristics. For further details we recommend Tanenbaum [Tanenb96]. Please note that the protocols listed below are merely examples illustrating the conceptual foundation of the activity scope tree.

Physical layer	The physical layer of communication systems is not of importance for today's IDSes. However, as technology evolves, this aspect may become relevant some day.

Table 1 – Networking related activity scopes–physical 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 2 – Networking related activity scopes–link layer

Network layer	The network layer generally provides a routable addressing of entities and may also offer the service of reliable communication i.e., a connection based service. However the most commonly used implementation of the network layer used today is IPv4 (internet protocol version 4) 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	*The 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 to 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 3 – Networking related activity scopes–network layer

Transport layer	The transport layer is the layer where in today's networks reliable communication is implemented. 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 used. The handling of TCP streams is an important but non-trivial issue for network-based IDSes.*
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 IDSes 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 IDSes.*

Table 4 – Networking related activity scopes–transport layer

Middleware	The monitoring of middleware protocols is–if actually 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 5 – Networking related activity scopes-middleware

Application layer	The monitoring of application layer protocols is a tedious task for IDSes 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 IDSes 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.*

Table 6 – Networking related activity scopes–application layer

3.1.2.2 Host related activity scopes

As already mentioned, it is not so straightforward to identify the activity sub-scopes for the *host activity scope*. The following activity scopes have been defined by identifying system components as they can be found in computing systems. The list focuses on security relevant activity scopes by refining areas of special interest to IDSes.

Devices	Devices are hardly covered by today's IDSes. 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 7 – Host related activity scopes–devices

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 IDSes, but with the ongoing development of technology [VMware00] that enables the operation of several independent virtual machines on the same physical system this may change.

Table 8 – Host related activity scopes–firmware

OS core	The operating system is used to control and to manage the system. Further it provides various services.

Table 9 – Host related activity scopes–OS core

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 very prominent target. If the network stack encounters a failure this often propagates to OS core and may therefore lead the whole system to failure. 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 10 – Host related activity scopes–OS modules

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 do 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 deamon-watcher by Wespi et al.* [WeDaDe00, WesDeb99]*, to obtain an indication on what a process is actually doing.*
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].

Table 11 – Host related activity scopes–calls

Filesystem objects	The filesystem is a very important 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, besides ordinary files, directories and links, the notion of special files such as named pipes, and sockets or device files.

Table 12 – Host related activity scopes–filesystem objects

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 very 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	*Semaphore are used for synchronization purposes e.g., to prevent the concurrent access to a resource.*

Table 13 – Host related activity scopes–IPC

Middleware	As mentioned, certain components of commonly used middleware technology can only be monitored on the host level. This is the reason for the repeated listing here.

Table 14 – Host related activity scopes–middleware

Environment	The behavior of a process may be influenced by its environment, this makes the environment an important attack interface for adversaries.
Variable	*Environment variables are copied at process creation time 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 whole system. The modification of the registry can, sometimes, influence processes already running.*

Table 15 – Host related activity scopes–environment

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 httpd (world wide web) and sendmail (mail service). Typical applications are web browsers or text processors.

Table 16 – Host related activity scopes–process

The final, first level activity scope to discuss is the *user activity scope*. We have not divided the user activity scope into lower level activity sub-scopes because there are no user components that would seem to be of special interest to an IDS. Nevertheless the user activity scope is needed as it may be used to describe an IDS' ability to relate observations to a user or possibly a group of users.

3.1.3 Functional activity scopes

So far we have developed an activity scope tree that enables us to classify objects and protocols according to their logical location in a system. We have pursued a pragmatic approach–focusing on items relevant to ID. This being said we have to admit that we have not yet been able to develop the activity scopes to the point required for compact and generic evaluation of IDSes–the goal being to remain above the implementation level when describing IDSes.

In particular protocols that have been assigned to a generic activity scope may vary significantly in the way they operate and the functionality they provide. Such functional differences have significant consequences for the IDSes that are analyzing protocols. For example, the analysis of a connection oriented protocol requires the IDS to perform far more sophisticated operations such as keeping the state of a protocol session, compared to the analysis of a connectionless i.e., datagram based, protocol. The two protocols may represent activity (sub)-scopes of the same generic activity scope e.g., both TCP and UDP are transport layer protocols. Because of the significant differences in the way they operate it is not possible to describe an IDS at the generic activity scope with respect to these protocols.

In order to address this obvious concurrent lack of generality and specificity we are introducing so-called *functional activity scopes* to refine the existing classification. The goal of these functional activity scopes is the detailed description and evaluation of IDSes at the level above the implementation of protocols etc., thereby still providing sufficiently detailed descriptions to make the evaluation viable.

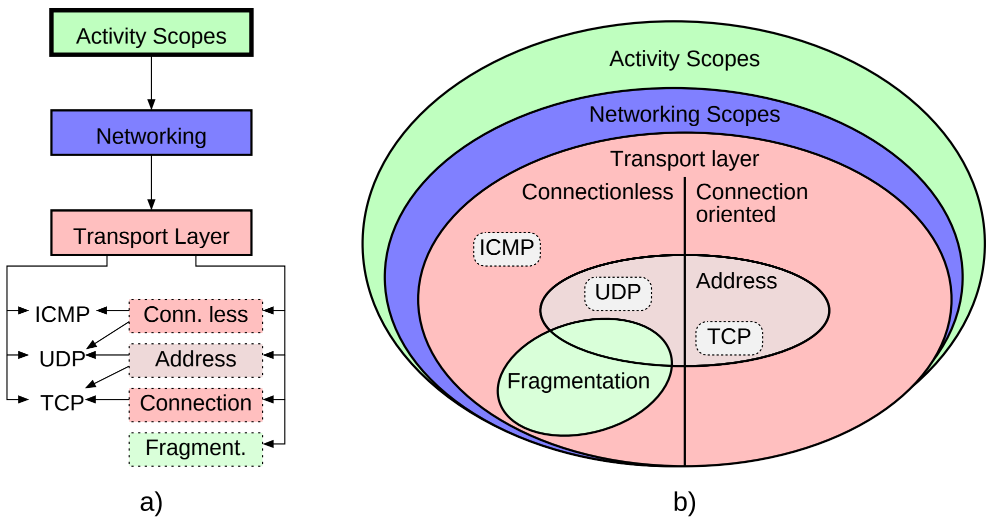


Figure 11 – Example of functional activity scopes

Figure 11 shows two views of an example illustrating functional activity scopes. Both views include the generic activity scopes networking and transport layer. Further both views show the transport layer activity (sub-)scopes ICMP, UDP, and TCP. Latter activity scopes are grouped into groups i.e., functional activity scopes, reflecting common functional properties. The example shown in Figure 11 introduces functional activity scopes that groups activity scopes according to whether an activity scope,

- is connectionless,
- is connection oriented,
- provides fragmentation[3] and/or
- provides addressing on the transport layer.

The resulting graph, shown in Figure 11a, is no longer a tree. In order to keep the graph directed and non-cyclic we have introduced directed arcs that by definition point from higher-level activity scopes to lower-level activity scopes. These arcs will have to be taken into account by the algorithms used for the evaluation of IDSes.

[3] We are not aware of a transport layer protocol that supports fragmentation. This functional activity scope is mentioned for illustration and for the sake of completeness.

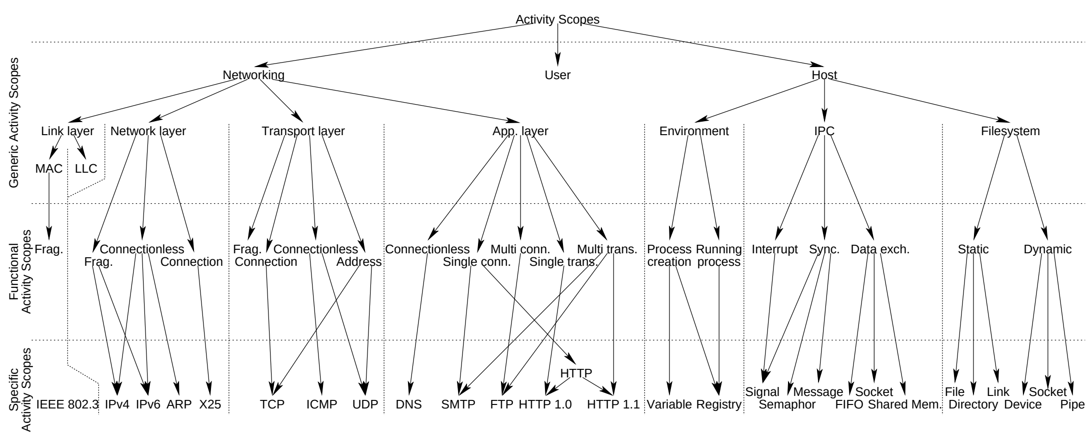


Figure 12 – Activity scope graph including functional activity scopes

Figure 12 shows the activity scope graph that includes the functional activity scopes defined in the following. Please note, this graph does not show the arcs pointing from the generic activity scopes to the specific activity scopes for readability reasons. Instead we indicated these associations with vertically arranged dotted lines. In addition the graph does not show all the specific activity scopes mentioned as examples in the following.

As mentioned, we now introduce and describe functional activity scopes by first mentioning their higher level activity scope. The names of the functional scopes are highlighted in italic font. We also provide examples wherever possible. Please note that we are again focusing on issues relevant to ID only.

3.1.3.1 Networking related functional activity scopes

As before we start by discussing the networking scopes first:

MAC	Medium access control layer, see Table 2.
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 17 – Networking related functional activity scopes–MAC layer

Network layer	See Table 3.
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 a 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 IDSes should recompose these fragments before they analyze the data. This is not so complicated, but it is costly in terms of CPU and memory required. Certain IDSes do not reassemble fragments for exactly those reasons. An adversary can fool those IDSes 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 18 – Networking related functional activity scopes–network layer

Transport layer	See also the example shown in Figure 11 and Table 4.
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. Examples: UDP, TCP.*
Connection oriented	*The analysis of connection oriented protocols imposes additional costs on the IDS that is 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 IDSes 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 IDSes 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 19 – Networking related functional activity scopes – transport layer

Application layer	See Table 6.
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 require one single connection only. This means that an IDS has to monitor only one transport layer connection in order 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 needs 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. For the purpose of this work we are relaxing the definition of transactions. 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 e.g. 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.*
Single transaction	*Application protocols that do not support multiple transactions within a session are generally simpler to analyze for an IDS because it need keep limited protocol state information only. Examples: HTTP version 0.9 and 1.0.*

Table 20 – Networking related functional activity scopes–application layer

3.1.3.2 Host related functional activity scopes

Within the host scopes we have been able to identify far less functional activity scopes than for the networking activity scopes:

Environment	See Table 15.
Process creation	*Environment elements belonging to this functional scope influence a process at the time the process is created. Examples: Unix and Windows environment variables, Windows registry.*
Running process	*Environment elements may influence processes already running. Example: Windows Registry.*

Table 21 – Host related functional activity scopes–environment

IPC	Inter-process communication, see Table 13.
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 22 – Host related functional activity scopes–IPC

Filesystem object	See Table 12.
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.*

Table 23 – Host related functional activity scopes–filesystem object

For the user activity scope we have not been able to identify functional activity sub-scopes relevant to ID.

3.2 Static activity characteristics

Now that we have introduced the notion of activity scope and have thereby identified all the activity scopes known to be relevant to ID we can now go back to the discussion of the activity taxonomy i.e., develop the static activity characteristics.

As mentioned, for the detection process performed by an IDS the location of the vulnerability enabling an intrusion is of limited importance only. However, the vulnerability has a significant influence on the set of interface objects and the affected object required for the description of the corresponding attack. In fact the vulnerability implicitly defines the avenues for possible attacks. Loosely 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 or more generally the activity. Furthermore the vulnerability implicitly defines the data sources i.e., once again the interface objects and affected objects an IDS has to monitor to detect a given attack.

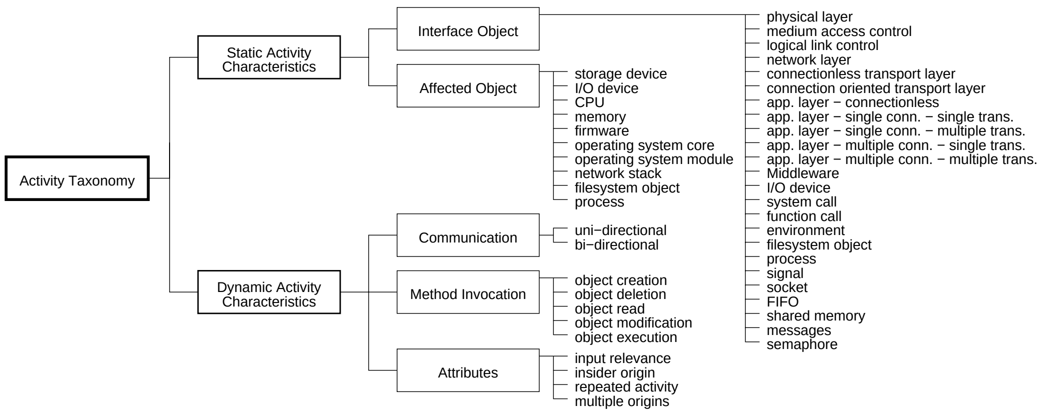


Figure 13 – Overview of activity taxonomy

In the following we discuss the activity taxonomy as shown in Figure 13–starting with the static activity characteristics. The taxonomy elements are developed based on the aim of the attacks described.

3.2.1 Affected object

It is the goal of a given attack to intrude a target object. More formally speaking, the goal is to change the state of the target object to a failure state that represents a security policy violation. 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 the fact that the fault i.e., the vulnerability, that enables a successful attack is not necessarily located in the affected object.

In the following we list the affected objects chosen from the activity scopes for the development of the activity taxonomy. They have been identified by isolating all 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 activity scopes at different levels in the activity scope graph discussed in Section 3.1 and shown in Figure 10. Please note that many of the scopes listed in the following are listed for reasons of completeness. They are listed because it is conceivable that they may play the role of an affected object, although we are currently not aware of an attack targeting them, and because the scope graph imposes their listing

- 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 here because it is a prominent attack target.
- Filesystem object
- Process

3.2.2 Interface object

Whenever an attack is executed it targets what we call the affected object. In order to interact with this object the adversary has to involve one or several *interface objects,* or in other terms 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 a given attack. This allows us to limit the choice of interface objects to the proximity of the attacked object. In other words, it makes no sense to consider the adversary's keyboard as a potential information source for the detection of a webserver attack, because the adversary's keyboard is not relevant to the detection of evidence of the attack.

Most of the affected objects listed above may also be used as an interface object. However, there is an important list of communication objects that was not listed among the affected objects that can figure as an interface for an attack. The categories of protocols listed below refer to depictions of said protocols. In addition the list of interface objects was developed based on the notion of activity scopes introduced in Section 3.1. A slight difference in the affected objects identified earlier is that the functional activity scopes (see Section 3.1.3) are used to refine the model. 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 between the connection oriented transport layer and the connectionless datagram service as it may be a differentiator for an IDS' ability to detect an ongoing attack.

- Connectionless transport layer
- Connection oriented transport layer

In the case of application layer protocols we differentiate even more. We do so not only by distinguishing among protocols based on connectionless and connection oriented services but also by considering the number of transactions that can be executed within 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 IDSes.

- 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. A typical example is HTTP version 1.1 that supports persistent connections. We have observed IDS not being able to recognize 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 IDSes 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 a 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 non-trivial task for IDSes.

We now continue with activity scopes that are primarily host oriented. The middlware scope is both a network oriented and a host oriented activity scope. We do not distinguishing between the host and the network portion of middleware here, because in general one cannot distinguish between host-based and network-based use of middleware[4].

- Middleware
- I/O device
- Operating system module: Some operating systems such as Linux 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 series of different mechanisms has been developed over time:

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

[4] For the description of an activity involving middleware it is not relevant to distinguish between the portion observable on the network and the portion observable on the host because one can assume that both are present. 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.

3.3 Dynamic activity characteristics

The dynamic activity characteristics of this taxonomy focus on observable attack relevant characteristics. These characteristics will, to a large extent, enable us to evaluate the attack recognition and identification capabilities of IDSes. Please note that because we are focusing on real-time and transition based IDSes, we do not model the impact of attacks. In other words we focus on observable evidence of attacks and not on the (possibly) resulting internal state change of the affected object.

In the model shown in Figure 5 we already identified three sets of dynamic activity characteristics that describe the interaction among objects. We distinguish the set of characteristics describing the inter-object *communication*, the *method invocation*, and some additional *activity attributes*.

Separating the inter-object communication and the method invocation characteristics, supports capturing 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.

3.3.1 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 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[5].

Please 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 him to do so or because the target has already become unresponsive e.g., crashed.

3.3.2 Method invocation

The second set of dynamic activity characteristics is the set of methods invoked within the context of the affected object (see also Figure 5). The identification of these methods is relatively straight-forward:

- Object creation: A new object is created. This generally occurs within 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.

[5] DNS stands for Domain Name Service–it is the directory service used on the Internet to translate host names to IP addresses.

- Object execution: The 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 [CA1395] and attacks involving special character [CA0696, CA0797].

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

3.3.3 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: *The creation of a file cannot be distinguished from the exploitation of a race condition.*

In order to address these types of 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 where input data is relevant.
- Repeated activity: Certain types of activities can only be clearly classified 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 clearly identify a given attack. Examples are attacks such as Smurf [CA0198] or distributed denial-of-service attacks such as Trinoo [CIN0799]. Those attacks end up sending a large amount of PDUs with arbitrarily forged sender addresses towards a target.

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

3.4 Attack Classification

Now that the activity taxonomy has been defined we could start developing activity assumptions by exercising all the possible combinations of activity characteristics. Although this would result in a large number of activities that could then be used for the evaluation of IDSes, this approach is unpractical mainly because of the sheer number of activities to be determined.

Instead we are pursuing another approach where we identify the *activity classes* that are of interest to ID, and thereby to the evaluation of IDSes as well, by classifying real-world attacks. An activity class is naturally defined by a given combination of dynamic and static activity characteristics. As implied, these ID-relevant activity classes are identified by describing real-world attacks using the activity taxonomy developed in this chapter.

Given that the data used for the attack classification is representative, it also allows us to validate the activity taxonomy developed here. This can be done by verifying that there is not a small number of activity classes accommodating the vast majority of all attacks classified.

As indicated, such an attack classification needs to be based on a representative set of attacks. We have met this requirement by classifying attacks taken out of IBM's vulnerability database VulDa. VulDa is described in more detail in Appendix B:. In the time available we have classified[6] 358 out of more than 800 attacks that are explicitly documented in VulDa and that have been collected over a period of 4 years. The attacks were chosen at random.

Example 1: *Before we take a closer look at the statistics of the resulting classification we consider an attack example to illustrate the way attacks were classified. A good example can be based on 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 classification:*

- *Affected object: Filesystem object*
- *Interface objects: Process, application layer (connection based, single or multi transaction)*
- *Dynamic activity characteristics: Bi-directional communication, object read, input is relevant.*

Example 2: *One can also execute the exercise in the opposite direction. We could for example ask the question whether an attack exists that is targeting a process using local, e.g. signals, and network, e.g. TCP, communication means concurrently. In fact, just recently, a theoretic attack against some FTP daemons has been discovered [Zalews01] where a remote attacker is able to inject executable data over the network that can then be activated by 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 arbitrary command. Please note that this attack has not yet been successfully run against a FTP daemon, but it has been proven that the attack is theoretically possible–even though it is considered to be rather difficult.*

- *Affected object:Process*
- *Interface objects: Transport layer (connection oriented), signal, system call*
- *Dynamic activity characteristics: bi-directional communication, object execution, input is relevant, repeated activity.*

As mentioned, we have classified 358 attacks along the lines of the two examples described above. Due to the descriptive nature of the activity taxonomy, which permits a relatively fine-grained description of activities, or in this case attacks, it is no surprise that 184 classes were used to classify the 358 attacks. Based on this material we were able to produce significant statistical results that are discussed in Chapter 5: . These results will support us when choosing representative activities to evaluate IDSes.

[6] Please note that the taxonomy developed here has been integrated into the database maintenance process. This means that the classification of attacks has become an ongoing effort continuously resulting in a more and more representative classification.

After all, such a classification can be used to validate the viability of a taxonomy. For instance, if a large number of attacks should fall into the same class, this would be a sign for a taxonomy that is too coarse. On the other hand if almost every attack should form its own class, this would be a sign for the fact that the taxonomy is likely to be too fine-grained.

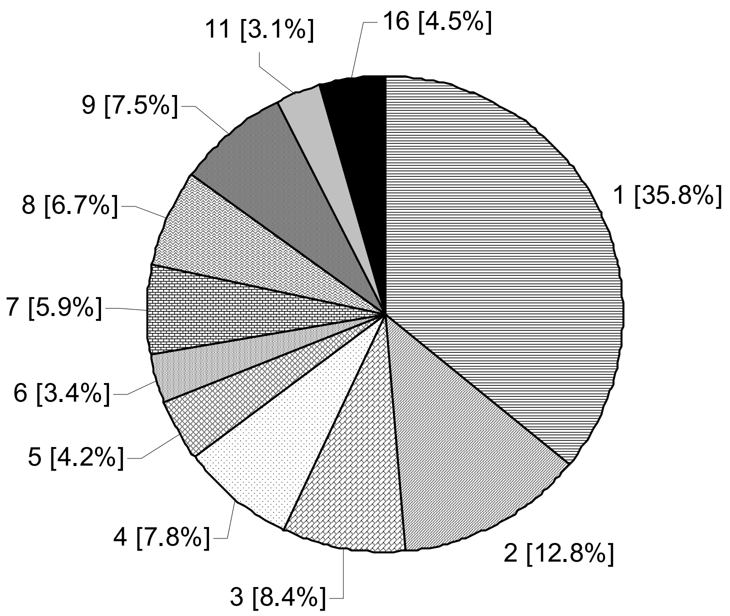


Figure 14 – Distribution of attacks on activity class sizes

As one can see in Figure 14 the taxonomy developed here is rather fine-grained. This figure shows for example that 35.8% of the classified attacks form their own class. On the other hand one can see that also rather large classes with up to 16 class members, accommodating 4.5% of the attacks classified, have been identified.

Figure 14 shows the distribution of attacks on activity class sizes based on the most fine-grained classification provided by the activity taxonomy developed in this chapter. In Chapter 5: we discuss various classification results that are more coarse in the sense that not every set of activity categories provided by the activity taxonomy was taken into account. This discussion allows the identification of very important attack meta-classes, knowledge that shall be reused when evaluating IDSes.

3.5 Discussion

The results of this attack classification reveal facts that correspond to the common reception of attacks as discussed in Appendix A:

- A large number of attacks that involve some form of network communication, are in fact one of today’s major problems.
- The figures in Appendix A show that processes are of high importance, on the one hand they serve as a target in the first steps of an attack and on the other are often misused as an interface to attack other objects.
- Processes are rarely used as interface objects in combination with communication protocol layers which also makes sense as it clearly distinguishes between attacks that run locally and attacks that are staged remotely.

- The statistical results and our ability to explain them make us confident that the underlying taxonomy is sound.

However, there is one point that we shall mention here. Considering the results presented in Section A.3 and the list of affected objects identified a bit earlier in Section 3.2.1 we notice that not all the affected objects actually show up in the diagram shown in Figure 29. So one might ask the question why we actually introduced those seemingly superfluous affected objects. The reasons are two-fold. On the one hand we were aiming at defining a taxonomy that offers the best possible coverage which was the reason why we derived our taxonomy from a typical computing system that included said affected objects. On the other hand we believe that one has to classify even more attacks according to this taxonomy to actually find attacks that fall into those rare categories. After all this classification is based on attacks as posted on the bugtraq mailing list [SecFoc] or discussed in a CERT advisory etc. In these forums attacks with low impact or attacks that require the attacker to have some important privileges to actually launch them are not heavily discussed i.e., published. In other words those attacks are generally not considered to be of high importance.

3.6 Conclusion

Based on the results of this attack classification we believe the activity taxonomy to be viable to serve as the foundations of the activity assumptions because the classes of attacks identified are sufficiently diverse and detailed. Also the identified attack classes relate well to the attacks described in the past. As mentioned the space spawned by this taxonomy contains a large number of elements. In order to be able to handle this challenge we will, when doing the actual evaluation of IDSes, select elements that were proven to be relevant by this attack classification. In addition we foresee an increase in generic activity meta-classes that can be combined to dynamically extend the number of activities used during the evaluation process.

Chapter 4: IDS taxonomy

In this chapter we develop a taxonomy for IDSes that is fine-grained enough without being too detailed so that it becomes possible to evaluate IDSes for their potential of discovering activities threatening the security policy, e.g. attacks.

Simply classifying an IDS into the class of behavior-based or knowledge-based (see Figure 6, p. 3) systems does not enable us to draw precise conclusions neither about the attacks an IDS is able to detect nor about the amount of false positives it could generate. In addition the other sets of taxonomy categories shown in Figure 6 do not permit such conclusions. In order to do so we propose an IDS taxonomy that is far more fine-grained. The proposed taxonomy aims at describing the capabilities of an IDS with respect to the analysis of activities.

This taxonomy shall provide the needed framework to evaluate IDSes and to determine whether a given IDS has the potential of detecting a given activity as threatening the security policy, e.g. an attack. Also, it enables us, per IDS, to determine the relative costs of the detection process in terms of memory, CPUs cycles etc.

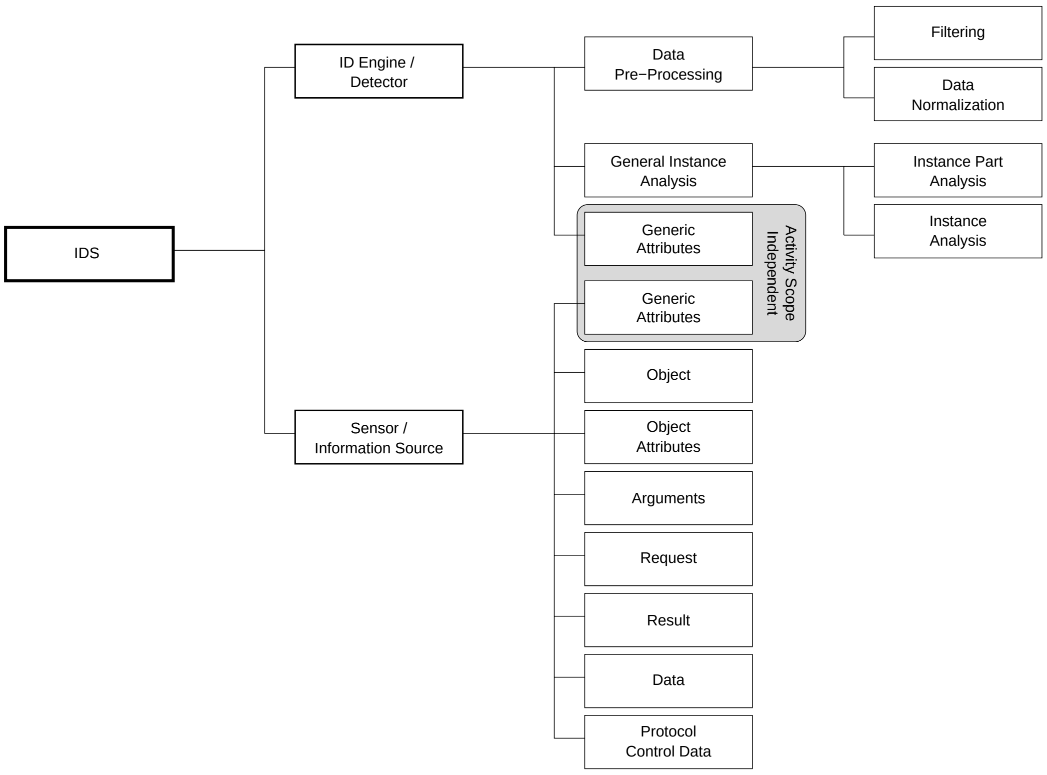


Figure 15 – IDS taxonomy overview

Figure 15 provides an overview of the taxonomy we propose for the classification and description of IDSes. In the following sections we develop and refine this taxonomy. We do so by first introducing a relatively simple model of IDSes that allows us to distinguish the data and functionality provided by IDS components. In a next step we develop the characteristics i.e., the taxonomy elements, required for the description of said components. The semantics of the majority of these characteristics are then defined with respect to the activity scopes introduced in Section 3.1. A minority of the characteristics do not need to be defined with respect to activity scopes because they describe generic properties such as delay. In a further step we introduce a notion of cost that can be associated with every IDS property.

Some insights on how the taxonomy can be used to classify IDSes and how to represent this classification in a database conclude this chapter.

As a side note, please note that in ID it is often not possible to separate *fault diagnosis* and *error detection* clearly. This particularly applies to the more popular knowledge-based IDSes because they are trying to identify signs of known attacks. Once these systems recognize a known attack, they detect the *error that may lead to security failure* (i.e., error detection) and at the same time they 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 number. The ability to fault diagnosis is far more limited in the case of behavior-based systems 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.

4.1 Intrusion detection system model

IDSes are generally separated into a portion that deals with gathering information from an information source (also called *sensor*) and a second portion that performs (security) error detection by analyzing the collected data (also called detector, analyzer or *ID engine*).

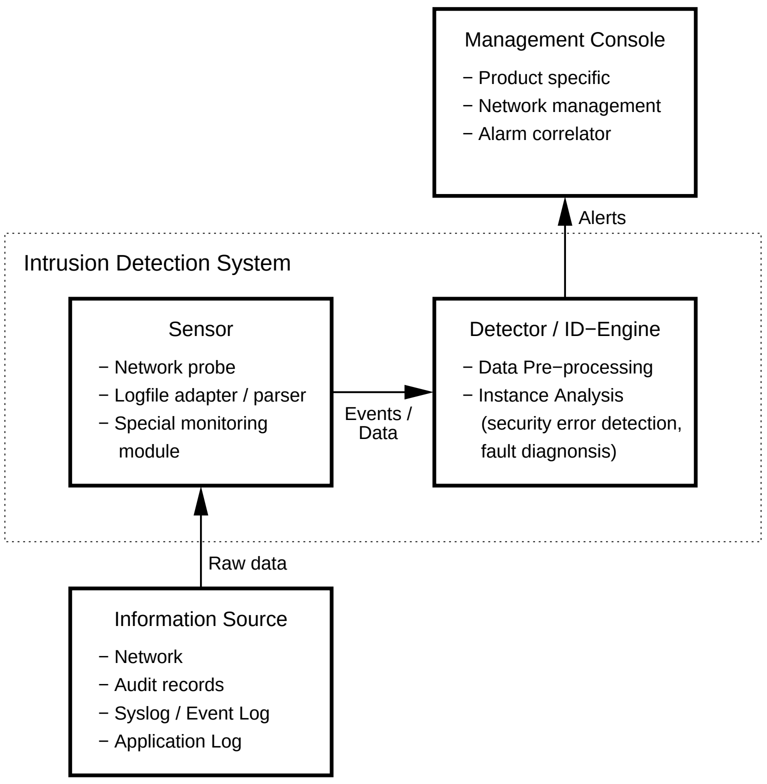


Figure 16 – Intrusion detection system model

Figure 16 illustrates this relatively simple IDS model[7]. This simplistic model leads to the definition of the two main subtrees in our taxonomy, as represented in Figure 15. The simplest form of an IDS consists of a single sensor that provides data to the ID engine which

[7] This model is in compliance with the current IETF efforts pursued by the intrusion-detection working group (IDWG) to standardize the messages exchanged among ID components [DeHuDo00, WooErl01].

performs the analysis of the data presented. However, in the real world IDSes are a bit more complex than shown in Figure 16:

- A more complex system may have several sensors that supply data to an ID engine.
- A slightly more complex system may be using one sensor that supplies data to several ID engines.
- An even more complex system may consist of several sensors and ID engines.

It is worth noting that it is not always possible to draw a clear line between ID sensors and ID engines, as shown in Figure 16. An example of such a problematic system is the work described by Kerschbaum et al., in [KeSpZa00], where the IDS is embedded into the operating system.

Sections 4.2 (resp. 4.3) describe the characteristics of sensors (resp. ID engines). Each characteristic is defined by means of an attribute. Most attributes are boolean variables. In other words, in most cases the attribute X describing characteristic Y of a given IDS component will be set to true if and only if that component has the characteristics Y. A few attributes are non-boolean variables that may take a value from a predefined set of values.

4.2 Sensor

In our model sensors are systems that transform the information provided by an information source into a form suitable for further analysis by the ID engine.

To understand and model the semantic of the information the sensor passes to the ID engine, we should take a close look at its internal. In most cases (e.g. commercial products) this is not possible. Fortunately, most sensors are very simple and 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 the IDS, by looking at the information sources it uses, and by investigating the diagnostic output the ID engine generates along with the alarms.

One of the first statements made in this document is that the quality of today's IDSes is generally rather inferior, with some exceptions. So, what about the quality of ID sensors? The data transformation performed by ID sensors generally leads to loss of information. This means that the amount of information that a sensor provides to the ID engine is a subset of the information available at the information source. This reduction has several reasons. 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. misperception of an ethernet Protocol Data Unit (PDU) or because some portion of the system 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 amount of traffic monitored surpasses the capacity of the monitoring port the switch starts dropping packets. As we shall see later even if the monitoring port and the host running the IDS are well equipped (e.g. gigabit-ethernet), information might be dropped at the ID engine for similar reasons.*

Please note that in this work we do not investigate the probabilistic aspects of whether a given or a given activity might or might not get processed for such reasons.

In the following, we describe the characteristics of the sensors in terms of two families: *activity scope independent* and *activity scope dependent* attributes. The resulting sensor description will enable us to answer the question whether a given sensor is providing sufficiently detailed information to enable the ID engine to properly detect and to recognize an attack.

4.2.1 Activity scope independent sensor attributes

The sensor attributes described in the following are independent of activity scopes. This means they do not have to be considered with respect to an activity scope in order to define their semantics.

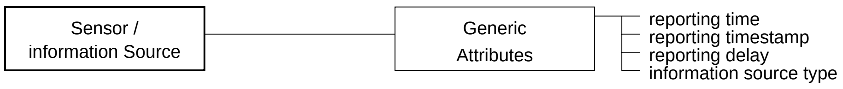


Figure 17 – Overview of activity scope independent sensor attributes

As Figure 17 shows, we have identified the following four activity scope independent sensor attributes (the table headings describe the attribute and the follow-up rows describe the permitted values):

Reporting time	The reporting time denotes the point in time at which evidence of an activity is observed and reported to the ID engine. This property is not so important for the detection process itself, but it may influence timeliness of the IDS.
Post-execution	*In the most common case a sensor will pass on the data to the ID engine 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 IDSes anymore as it analyzes activities before they actually happen. Such systems may deny an activity from being executed and are therefore merely policy enforcement systems. For an example of such a system see the work by Hutchison and Welz* [HucWel00].

Reporting delay	The delay between the point-in-time an activity is observed i.e., identified, and the point-in-time the activity is reported to the ID engine. Based on our expertise and experience with various sensors, we arbitrarily express that property in terms of the following ranges.
Less than 3 seconds	*It takes the sensor less than 3 seconds to pass the data describing the activity observed on to the ID engine.*
Less than 1 minute	
Less than 15 minutes	
More than 15 minutes	
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 ID engine.*

Reporting timestamp	The reporting timestamp denotes the timestamp a sensor assigns to an observation when it is reported to the ID engine. Many ID sensors do not provide such a timestamp and leave it up to the ID engine to set a timestamp whenever it finds something worth reporting to the administrator i.e., whenever the IDS is issuing an alarm.
None	*The sensor is not providing 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 been started.*
End of activity	*The timestamp provided by the sensor to the end of an activity.*

Table 24 – Activity scope independent sensor attributes

4.2.1.1 Information source type

The sensor attributes just introduced are of importance when considering the diagnostic capabilities of an IDS, but hardly affect the detection capability of an IDS. This is different for the last activity scope independent sensor attribute, the *information source type*.

The information source type determines the view the IDS has from 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 IDSes are classified based on what they call *audit source location* (see also Figure 6). However, knowing the location where information may be gathered is important and useful for the description of IDSes but only implicitly describes the inherent properties of the information gathered from the respective sources. For instance, when a network-based IDS is gathering an URL from the network, it is impossible for the IDS to clearly determine how the webserver software will be interpreting the URL–especially when the presence of attack obfuscation techniques are assumed. The situation is different when considering 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 terms, 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 does share the same view on the data.

When making these considerations we were able to identify 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 non-transformed view of the data as it is sent by an activity. In contrast, log data sources represent a view on the manner in which received data was interpreted by the monitored system and not a view 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 monitored system. The raw data class is sub-divided into an *external* and an *internal* class of information sources. External information sources provide a view on the raw data before it reaches its destination–the monitored system. This is where classical network-based IDSes based on network sniffers fit (examples: [CiscoNR99, ISSNet99, Paxson98, Paxson99, Roesch99]). Internal raw data information sources access the raw data at the monitored system and within the same activity scope as the monitored system it does. So far mainly network-based IDSes that inspect the data on a host as it traverses the network stack have been implemented [ISSSer00]. However, it is conceivable that such sensors may also be implemented directly into applications. For instance one could envisage a webserver sensor module that inspects the 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 i.e., sources provided by *applications* and so-called *meta* information sources that typically already include interpretations of observed activities. 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 IDSes[8].

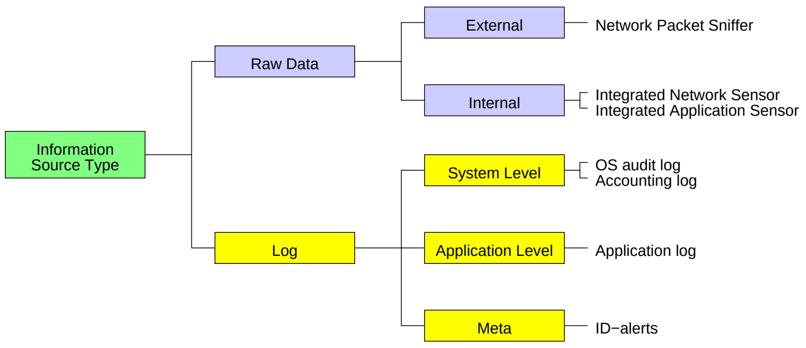


Figure 18 – Overview of information source type taxonomy

Figure 18 provides an overview of information source type taxonomy just introduced and also includes the information source type examples mentioned above. In the table below we describe said source type examples in more detail.

Information source types	The value set used for the description of this attribute has been derived and extended from the audit source location category of the IDS taxonomy by Debar et al. [DeDaWe00] as shown in Figure 3, p. 3.
Data external	*Network packet sniffer: The IDS sensor sniffs its data from the network, e.g. of such systems are snort* [Roesch99] *and Bro [Paxson98, Paxson99].*
Data internal	*Integrated network sensor: The IDS sensor monitors the data as it traverses the network stack, e.g. at the socket interface. An example of this system is [ISSSer00].* *Integrated application sensor: This type of sensor is similar to the previous one. However, in contrast it is integrated into the application daemon to be monitored rather than into the networking stack itself. For instance, it is conceivable to build a sensor module that can be loaded into daemons such as the apache webserver software [Apache].*

[8] Please note that this type of information sources operate at a different level that is not key to this work, which is why we do not further investigate them here.

System level log	*OS audit log: The IDS sensor analyzes the audit log, e.g. C2 audit log as written by the AIX operating system [LCRM98]. An example of this system is daemon-watcher by Wespi et al. [WDDN98, WeDaDe00, WesDeb99].*
	Accounting log: The IDS sensor analyzes system account logs that report accounting information such as login sessions from users, consumed system resources etc. Examples of such systems are IDES and NIDES developed by SRI [JLADGJ93, lunt90a].
Application level log	*Application log: The IDS sensor gathers the log information provided by applications. An example of an IDS operating on the application log is WebIDS [Almgre99].*
Meta log	*ID-alarms: ID-alarms fall into meta log class and stand for pre-processed information as it may be generated by a data reduction tool or by another IDS. In fact the output i.e., the alarms, generated by an IDS can be considered to be pre-processed log data. An example of this system is RiskManager offered by Tivoli Systems [TRM00].*

Table 25 – Activity scope independent sensor attributes–Information source types

4.2.2 Activity scope dependent sensor attributes

This section deals with information items that shall be discussed in close relation to the notion of activity scope introduced earlier in Section 3.1. More precisely we define the semantics of information items represented by attributes within the context of a given activity scope.

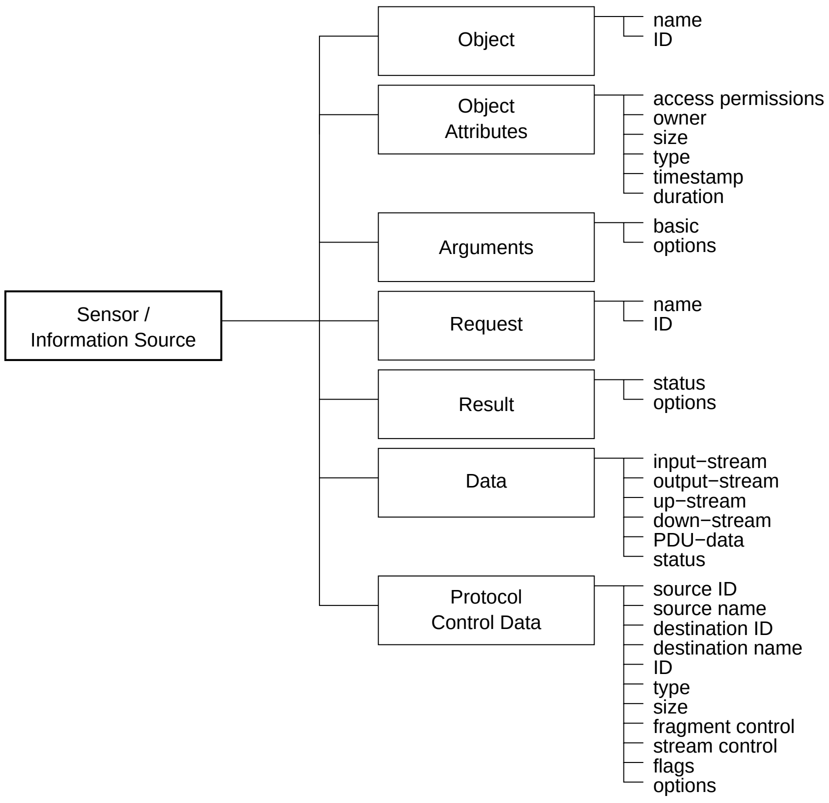


Figure 19 – Overview of scope dependent sensor attributes

All the attributes shown in Figure 19 are explained in the following. In Appendix C: we provide extensive examples of their semantics which can be defined with respect to a given activity scope. These semantic definitions are made on the highest activity scope level possible or on the level of functional activity scopes (see Section 3.1.3) where applicable. It was actually one of the goals, pursued by the introduction of the functional scopes to avoid semantic definitions on highly specific activity scopes such as protocols. Also please note that if the semantics of a sensor attribute are defined on a given generic activity scope, it does not necessarily mean that the attribute exists on every descendent activity scope.

The following table defines all the attribute categories and their attributes:

Object	The object category is made of two attributes (for examples see Table 45, p. 3):
Name	*The name of an object is generally human readable and, in most cases, uniquely identifies an object.*
ID	*An object identifier, in most cases, uniquely identifies an object by a numerical or possibly alphanumerical identifier.*

Table 26 – Activity scope dependent sensor attributes–object

Object attribute	The object attribute category provides additional attributes required to describe an object. Please note that the state of an object is captured in the data category. See also Table 46, p. 3.
Type	*If an IDS is able to obtain the information on the type of an object the IDS can differentiate between similar objects in the same activity 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 attribute denotes timestamps such as creation time or login time.*
Duration	*The duration attribute denotes duration's such as lifetime or the time consumed.*

Table 27 – Activity scope dependent sensor attributes–object attributes

Argument	The argument category is used to represent arguments supplied to calls, process, requests etc. We distinguish only two different properties (see also Table 47, p. 3):
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 28 – Activity scope dependent sensor attributes–arguments

Request	The request category is also very small. It is used to name calls, request etc. See also Table 48, p. 3.
Name	*The name of the request.*
ID	*The ID of the request made.*

Table 29 – Activity scope dependent sensor attributes–request

Protocol control data	The protocol control data category is a bit more complex than most of the other categories. It needs to capture the variety of protocols that have been defined. See also Table 49, p. 3.
Source ID	*The source ID typically denotes the source address of the PDU considered.*
Source name	*Like source ID, but instead of a numerical ID a name is used.*
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 destination ID, but instead of a numerical ID a name is used.*
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 variety of functionality.*
Options	*Protocols often offer a number of other fields and options that are not covered by the attributes listed above. We summarize those fields and options here.*

Table 30 – Activity scope dependent sensor attributes–protocol control data

Data	The data category is quite complex as well. In this category all properties that can be considered as data of any kind are collected. 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 ID engine. A typical example is the fragmentation of IP PDUs, which may make it impossible to analyze the payload of IP PDUs for IDSes 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 ID engine. See also Table 50, p. 3.
Input stream	*This attribute represents the stream of data fed into an object, e.g. a process.*
Output stream	*This attribute represents the stream of data generated by an object, e.g. a process.*
Up-stream	*This attribute 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 attribute represents the data that is returned by a server to a client.*
PDU-data	*The attribute PDU-data represents the data portion of a PDU. This attribute 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.*

Table 31 – Activity scope dependent sensor attributes–data

Example: *An IDS such as WebIDS [Almgre99] that parses webserver logs in the common log format (CLF) [Weinma98] will be able to provide the following information to the ID engine (considering the HTTP activity scope only):*

- *Request name*
- *Basic arguments*

In addition the sensor provides protocol control data attribute source ID within the network layer activity scope.

4.3 Intrusion detection engine

The ID engine, also called detector, performs *error detection* for errors that may lead to security failure and to some degree *fault diagnosis*. It does so based on the information gathered by the ID sensor as indicated earlier and as shown in Figure 16. The ID engine is generally the most complex component of the IDS. It is also the component that varies the most among the different approaches proposed and implemented by the ID community.

In general we were able identify three categories of attributes that separate the description of ID engine characteristics. The sets are illustrated in Figure 15 and are named as follows:

- Generic characteristics: The category of generic ID engine attributes is very similar to the one of the of IDS sensors. These attributes are also independent from activity scopes.
- Data pre-processing: The data pre-processing attribute category contains a set of activity scope dependent attributes that describe the operations an ID engine is able to perform on the data provided by the sensor before the data is analyzed for signs of errors that may lead to security failure.
- Instance analysis: The attributes of the instance analysis category are also activity scope dependent and describe the ID engine's abilities to perform security error detection, which generally includes some limited degree of fault diagnosis.

In the following we first discuss the generic activity scope independent attributes and then continue with the two activity scope dependent attribute categories.

4.3.1 Activity scope independent ID engine attributes

As mentioned, the set of generic activity scope independent ID engine attributes is similar to the respective set used to describe sensors. Namely the alarm timestamp and the alarm delay attributes shown in Figure 20 have very similar definitions to the respective sensor attributes discussed in Table 24.

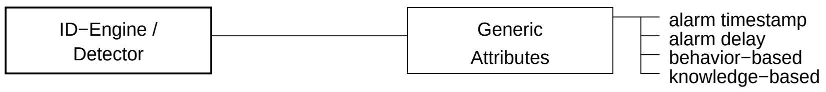


Figure 20 – Overview of activity scope independent ID engine attributes

However, the set of activity scope independent ID engine attributes, also contains two attributes that do not appear in the set of activity scope independent sensor attributes. These attributes are used to distinguish detection methods.

After all we have been able to identify the following set of activity scope independent ID engine attributes:

Alarm timestamp	This attribute describes the point-in-time an alarm timestamp denotes, with respect to the information reported by a sensor. In other terms this attribute tell us whether the alarm timestamp refers to the beginning or to the end of a sequence of sensor reports that led to the generation of an alarm. The possible values are identical to the ones of the *reporting timestamp* attribute discussed in Table 24.
Alarm delay	This attribute describes the delay the described ID engine 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 possible values are identical to the ones of the *reporting delay* attribute discussed in Table 24.
Behavior-based	This attribute is a boolean variable and is used to denote the fact that the described ID engine applies a behavior-based detection method or at least uses a component that is behavior-based. See also Section 2.2.1.
Knowledge-based	This attribute is a boolean variable and is used to denote the fact that the described ID engine applies a knowledge-based detection method or at least uses a component that is knowledge-based. See also Section 2.2.1.

Table 32 – Activity scope independent ID engine attributes

4.3.2 Data pre-processing ID engine attributes

In the most general case the ID engine has to pre-process the data obtained from the sensor because the data may not yet been in a form for the instance analysis where the actual security error detection is done.

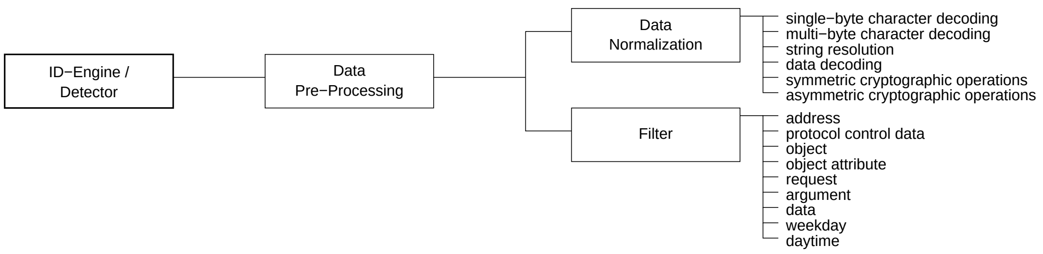


Figure 21 – Overview of data pre-processing ID engine attributes

On one hand this means filtering of data that is either irrelevant or for some reason not of further interest. On the other hand it is often possible to represent data in various different but valid ways. In order for the ID engine to be able to analyze this data, it needs to be normalized.

In the following we first discuss the representation of data normalization characteristics and then in a next step the filtering of data.

Data normalization	Application layer protocols in particular, often offer several different ways to express the same fact i.e., it is often possible to formulate an information item using varying syntactical expressions that have the same semantic meaning. For examples see Table 51, p. 3. Such a high degree of freedom in the representation of data enables adversaries to slightly modify their attacks such that it becomes significantly more difficult for IDSes 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 is performing sanity checks on the data that 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 the said sanity checks. Generally speaking, data normalization is especially important for knowledge-based systems. This is because an ID engine's inability to normalize data may result in attack signatures not matching.
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 ID engine.*
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 IDSes that perform cryptographic transformations on the data they observed. However, it is conceivable that IDSes perform such transformations on the data they observe. In our case this applies to information items as they were identified in Section 4.2.2. For instance, it is conceivable that a network-based IDS that is monitoring a webserver, using SSL (secure socket layer) to encrypt customer data and transactions, is holding a copy of the webserver's private key. Knowing the webserver's private key enables the IDS to monitor encrypted https traffic. Please 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 concerns, risks created by the fact that private keys are stored at multiple places etc.*

Table 33 – Data pre-processing ID engine attributes–data normalization

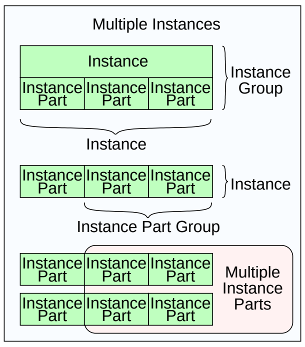
Filtering	Filtering of data reporting activities may be an important measurement for the elimination of false positives or undesired alarms in general. A typical example is the filtering based on network addresses if a given host is known to cause many false alarms, even though the host is known to be harmless. For instance, this may be caused by a broken implementation of the host's TCP/IP stack that is generating many fragmented packets[9]. Another cause might be a host that is used to scan the network for vulnerabilities and would therefore cause a real storm 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 4.2.2. In addition to those data categories address data, weekday, and daytime have been added.
Object	*See Table 26.*
Object attributes	*See Table 27.*
Arguments	*See Table 28.*
Request	*See Table 29.*
Data	*See Table 31.*
Protocol control data	*Excluding address data. See Table 30.*
Address	*Address data is listed separately as it is one of the most important information items used in filtering rules.*
Weekday / *Daytime*	*We have extended the list by two notions of time period–weekday and daytime. These notions of time period may be required from an IDS to eliminate alarms, known to be caused by harmless regularly occurring activities. Examples are, scheduled DNS zone transfers.*

Table 34 – Data pre-processing ID engine attributes–filtering

[9] Older Cray operating systems were known to erroneously set the fragmentation bit on every packet they sent to the network.

4.3.3 Instance analysis ID engine attributes

In this section we develop the IDS taxonomy elements used to represent the error detection and fault diagnosis capabilities of ID engines. In order to do so we introduce the concept of so-called *instances*. As already done for other attributes, the semantic of an instance is defined by the activity scope to which it relates. More precisely, an instance represents the instantiation of a given activity scope.



a)

Instance Notion	App. Protocols	Trsp. / Netw. / Link Layer	Processes
Instance Part	prot. statement	fragment	thread
Instance Part Group	transaction part	fragments of a PDU	threads of a process
Multiple Instance Parts	parts of transactions	fragments	threads
Instance	transaction	connection / PDU	process
Instance Group	session	–	application / service
Multiple Instances	transactions	connections / PDUs	processes

b)

Figure 22 – Overview of instance analysis (excl. bi-directional instance analysis)

As illustrated in Figure 22, an ID engine analyzes instances not only at the instance level. Depending on the activity scope of an instance, the ID engine may be analyzing *instances, instance parts,* or both. The concept of instances and instance parts can be illustrated by considering an instance of the activity 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 related to each other–last but not least because the receiver needs to be provided with the information required for the recomposition of 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 are therefore not building a group, e.g. two processes that implement two different services, it happens that they influence each other nevertheless. This is a fact that has to be addressed in this taxonomy as well. In the following we call an ID engine able to perform analysis across *multiple instances* capable of performing *cross-instance analysis,* respectively *cross-instance part analysis* if the ID engine is able to perform analysis across multiple instance parts that do not belong to the same instance. In fact many security problems arise because instances interact in a way they were not designed for or interact even though no interaction at all was foreseen.

The structure of the taxonomy as shown in Figure 23 considers instance and instance part group analysis to be a sub-branch of cross-instance, respectively cross-instance part analysis. We do so because the former can be considered to be a subset of the latter. More precisely the analysis of an instance group can be considered to be cross-instance analysis of instances that share common criteria. The same naturally also applies to instance parts.

When combined with the notion of activity scopes, the concept just introduced provides a generic and systematic way to describe the capabilities of an ID engine. However, one might ask–how does this concept of instances relate to activities? As mentioned, activities are described by the combination of IDS characteristics required to process a given activity and to possibly identify it as an attack. This means that activity descriptions include combinations of instance analysis related ID engine characteristics as well. However, an activity can typically be analyzed in several different ways i.e., based on instances of several different activity scopes. For example, a typical buffer overflow attack can be detected based on a deviation of the sequence of system calls made by a process or based on the observation of a too long input string. Please note that therefore, although one might be tempted to do so, instances cannot be equated to activities or the reports of activities. After all instance analysis related ID engine characteristics are one of the most important elements of the process of ID.

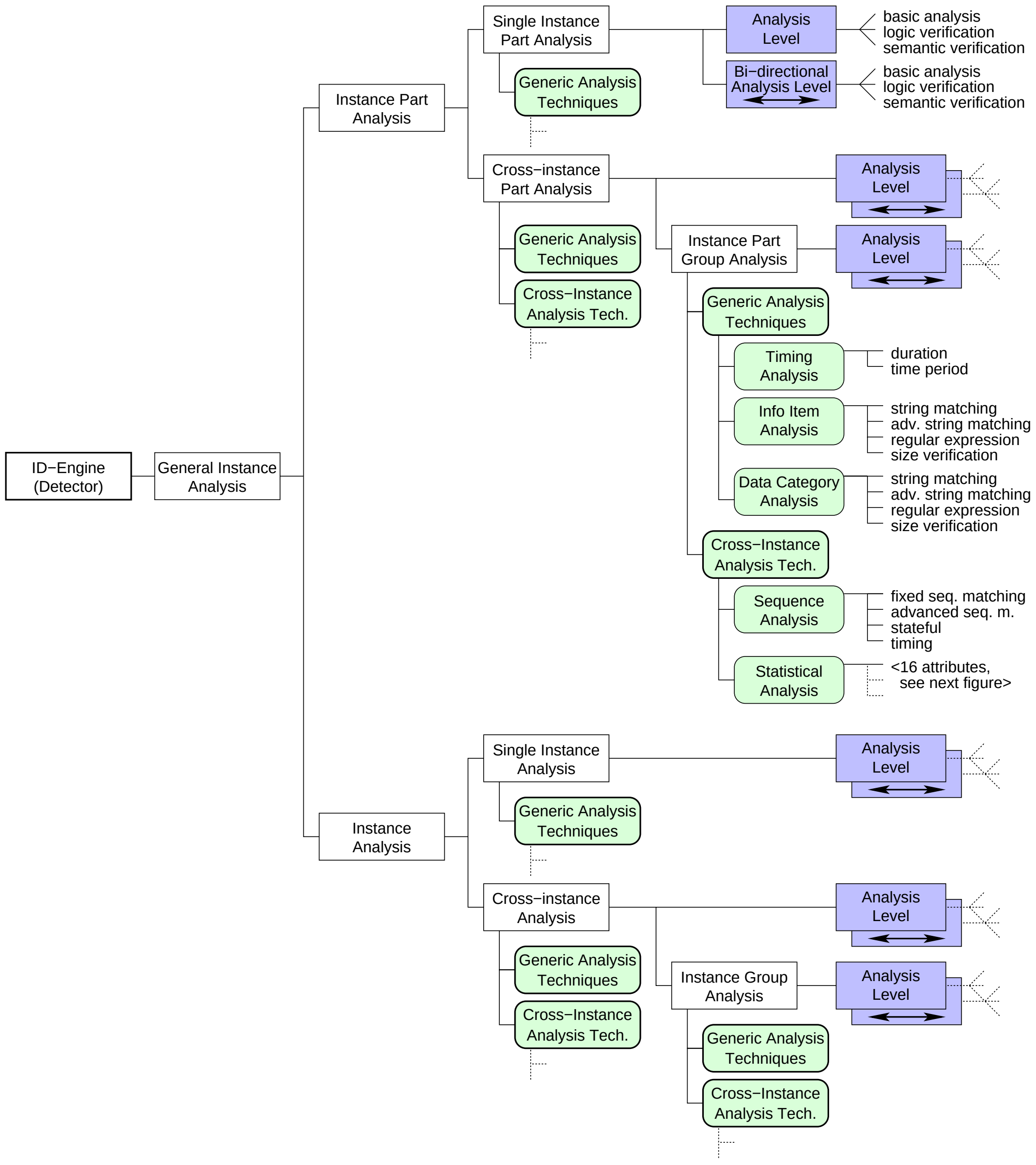


Figure 23 – Overview of instance analysis ID engine attributes

Figure 23 shows an overview of all the instance analysis related ID engine attributes that we have identified and that shall be discussed in the following. The figure shows the two branches of what is called general instance analysis. The first branch illustrates the description scheme for the analysis capabilities of an ID engine with respect to *instance parts*, whereas the second branch does the same for *instance* analysis. The two branches are identical except for the fact that one describes instance analysis and the second instance part analysis. When looking more closely at the two subsets one can find the different types of instance and instance part analysis identified above.

However, Figure 23 shows even more than that. In the following we discuss the various sub-sets of attributes i.e., the *analysis levels* and *analysis techniques*, that we have identified in order to characterize and describe ID engines. The analysis techniques are used to describe the way the ID engine analyzes a given instance whereas the analysis levels are used to describe the level at which the analysis is performed. Finally we provide a series of examples in C.3.

4.3.3.1 Instance analysis levels

As shown in Figure 26, for every type of instance and instance part analysis we distinguish between *analysis levels* and *bi-directional analysis levels*. The introduction of the notion bi-directional analysis levels is motivated by the fact that an ID engine often has to analyze protocols and other instances that are of bi-directional nature. Because most protocols and other instances such as system and function calls are bi-directional by definition and the analysis of such instances requires additional capabilities from the ID engine, the attributes describing the instance analysis levels have been divided into the two said subsets.

When considering the different analysis levels, we distinguish three different analysis levels that range from *basic analysis* over *logic verification* to *semantic verification*. These analysis levels were inspired by Dobson's [Dobson89] abstraction levels (see also Section 2.4.3) and need to be interpreted separately for each instance analysis type.

In the following we define these three analysis levels at a conceptual level. In addition we provide interpretation guidelines with respect to the various instance analysis types. Please note that every higher-level analysis level comprises any possible lower-level analysis level.

Also please note that every instance analysis level has to be considered in conjunction with an activity scope in order to define its semantics. In Section C.3.1 we provide a series of examples where we demonstrate how the various analysis levels have to be interpreted with respect to activity scopes. It is also worth mentioning that 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.

Basic analysis	Basic analysis denotes the fact that a given ID engine performs a very low-level analysis such as simply recognizing the thing of interest. The thing of interest can be an object, a request etc. and is defined by the respective instance or instance part-analysis attribute sub-branch and the effective activity scope.
Single instance and *instance part analysis*	*The ID engine 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 52 and Table 53.*
Instance and *Instance part group analysis*	*The ID engine associates instances, respectively 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 ID engine. See also Table 54 and Table 56.*
Cross-instance and *cross-instance part analysis*	*The ID engine associates instances and instance parts that are formally unrelated.*

Table 35 – Instance and instance part analysis levels – basic analysis

In Table 35 we provide 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 bi-directional instance and instance part analysis levels because they are highly similar to what is already defined in Table 35. They just extend the respective definitions towards the analysis of bi-directional instances and instance parts. We also do so in the following definitions of the analysis levels *logic verification* and *semantic verification*.

Logic verification	The analysis level logic verification denotes the fact that a given ID engine verifies the thing of interest at the logical level. As before, the thing of interest is defined by the respective instance or instance part analysis attribute sub-branch and the effective activity scope. In most cases this is equivalent to syntax verification.
Instance and *Instance part analysis*	*The ID engine 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 do not need to be complete or logically correct to be considered as an instance. In fact many attacks are manifested by incomplete instances. See also Table 52 and Table 53.*
Instance and *instance part group analysis*	*The ID engine verifies the logical relation among instances and instance parts belonging to the same group. See also Table 54 and Table 56.*
Cross-instance and *cross-instance part analysis*	*As we are not aware that cases where a logical dependency among instances and instance parts, unrelated by definition, exist, we do not further define this particular level.*

Table 36 – Instance and instance part analysis levels–logic verification

Again we omit the bi-directional instance analysis types for the same reasons as already mentioned. A simple example that can be used to illustrate bi-directional logic verification is TCP. There an ID engine is considered to be performing bi-directional logic verification if it verifies that the TCP-PDUs exchanged in both directions fulfill the protocol specification.

Semantic verification	If an ID engine performs at the semantic level it means that it is verifying the semantic correctness and acceptability of the thing of interest. The thing of interest is defined by the respective instance or instance part analysis attribute sub-branch and the effective activity 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 ID engine verifies the semantic correctness of the instances and instance parts. See also Table 52 and Table 53.*
Instance and *instance part group analysis*	*The ID engine verifies the semantic correctness of the relation among instance and instance part group members. See also Table 54 and Table 56.*
Cross-instance and *cross-instance part analysis*	*The ID engine verifies the semantic consistency and acceptability among instance parts and instances. See also Table 55.*

Table 37 – Instance and instance part analysis levels–semantic verification

Again by referring to the bi-directional analysis, we illustrate bi-directional semantic verification using the example where an ID engine detects the fact that a protected, non-public web page was revealed to the public. In order to do so, the ID engine has to recognize that a HTTP request that is asking for a protected page is fulfilled by the webserver.

Example: *To illustrate the taxonomy elements just introduced, we consider once more WebIDS [Almgre99]. WebIDS mainly operates within the HTTP activity scope. Within this activity 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 cannot only detect suspicious requests (semantic instance verification), it can also determine whether they were successful (bi-directional semantic instance verification). After all it does not perform semantic verification across requests, but it 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). This description is not complete.*

4.3.3.2 Generic analysis techniques

Figure 23 shows not only the instance analysis levels but also the instance analysis techniques. The techniques developed and described in the following are of a relatively high level. They merely describe the resulting ID engine capability rather than the implementation used to achieve a given capability. For instance, the stateful analysis of a sequence can be achieved using various different techniques such as state machines or petri nets. Although we agree that state machines and petri nets are not the same, they both represent a stateful technique to analyze sequences.

As shown in Figure 23, the generic 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 an ID engine 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 23, we have identified three sub-sets of attributes which describe techniques that can be applied on single instances and single instance parts. Their identification was relatively simple–for instance, considering any kind of PDU–what is the information one can potentially use for further analysis? There are basically three things. First one can record the header and the data portion of the PDU for analysis. In addition one knows the point in time at which the PDU was observed. Similar considerations can be made for most other activity scopes and for cross-instance and cross-instance part analysis. Using more general terms we have therefore identified the following three sets of generic analysis technique attributes: *information item analysis techniques*, *data category analysis techniques,* and *timing analysis techniques*. Please note that information item analysis techniques include all the information items described in Section 4.2.2 except the data items described in Table 31–these data items are covered by the data analysis techniques instead.

Before starting a discussion on the various attributes used to describe these techniques it is worth mentioning that not every single technique represented by one of the attributes described in the following, is applicable for every type of instance analysis.

4.3.3.2.1 Timing analysis techniques

The attribute sub-set describing the timing analysis techniques is the smallest and consists of two members only:

Timing analysis techniques	

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

Table 38 – Instance and instance part timing analysis attributes

4.3.3.2.2 Information item analysis

Especially, but not solely, when verifying the logical and semantic correctness of instances, instance parts etc., information items often need to be analyzed for their content. As illustrated in Figure 23, we have identified four additional attributes for the description of the respective ID engine characteristics. Three of them address the content and one the size of the information item. Please note that when referring to information item within this context, we refer not to any information item provided by the sensor, but the information items belonging to the data category (described in Table 31).

Information 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 a bit further than just 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. In case the ID engine has a basic analysis of the instance in question only, size verification can be seen as a very limited syntax check. In case the ID engine is able to verify the logical correctness of an instance the size check may be applied in addition to identify suspiciously-sized instance elements.*

Table 39 – Instance and instance part information item analysis attributes

It is important to note that an ID engine might perform information item analysis without verifying the syntactical correctness of the instance. This is actually an important cause of false positives and false negatives. For example, IDSes 'blindly' applying string matching on protocol statements may result in erroneous reports of suspicious strings that are harmless or even normal within the context they appeared. 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.

4.3.3.2.3 Data category analysis

The set of attributes described above within the context of information item analysis explicitly excludes the data category as introduced in Table 31. This is done because in real-world settings the data portion of an instance is often not as easily accessible as other information items related to an instance. An example is HTTP where the HTTP request is often treated differently than the data associated with the request i.e., the document served to the client or the data posted by the client. Also ID engine's capabilities with respect to the data category items may vary because of said reason.

We do not repeat the attribute definitions here because they are identical to the attributes as defined in Table 39.

4.3.3.3 Cross-instance analysis techniques

When considering multiple instances or instance parts concurrently an ID engine can apply additional analysis techniques than the ones just identified. The two additional attribute sub-sets that we were able to identify describe the ID engine's capabilities to verify the *sequence* of instances and instance parts and to analyze the *statistical* properties of instance and instance part sequences.

4.3.3.3.1 Sequence analysis techniques

The attributes that we identified for the different types of sequence analysis are quite similar to what we have identified for the analysis of information items in Section 4.3.3.2.2. However their semantic is a bit different as they address state transitions rather than string processing or information analysis in general.

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 a bit further than just the identification of known sequences by allowing the use of 'don't care' placeholders or wildcards.*
Stateful sequence analysis	*In the case of stateful sequence analysis the ID engine 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.*

Table 40 – Instance and instance part sequence analysis attributes

4.3.3.3.2 Statistical analysis

The last set of attributes discussed here is the set of attributes describing the way a given ID engine analyzes statistical properties of sequences. Such analysis is required for the detection of port scans or flooding attacks as it is implemented by IDSes such as Bro [Paxson98, Paxson99] or Snort [Roesch99].

The description of these attributes that we provide in the following varies from the attributes identified thus far because we have identified four sets of characteristics that have to be combined.

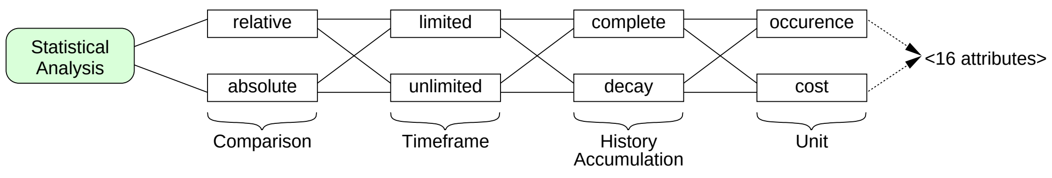


Figure 24 – Characteristics of statistical instance and instance part analysis

As illustrated in Figure 24, the combination of these four sets of characteristics that contain two characteristics each, results in a total of 16 attributes that are used to describe statistical instance and instance part analysis techniques.

The following tables describe the four sets of characteristics that we have identified for describing the statistical instance analysis performed by ID engines.

Comparison	Relative vs. absolute measure
Relative	*An ID engine is considered to be performing relative measures of class instances and instance parts if it is comparing the measures made for one class to measures made for another class.*
Absolute	*An ID engine performing absolute measures of instances and instance parts is just considering a class of instances without taking other classes into account.*

Table 41 – Statistical instance and instance part analysis–comparison

Timeframe	Limited vs. unlimited timeframe measure
Limited	*The ID engine 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 ID engine simply counts or accumulates instances, instance parts, or measurable properties of these. This means that the timeframe is unlimited.*

Table 42 – Statistical instance and instance part analysis–timeframe

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

Table 43 – Statistical instance and instance part analysis–history accumulation

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

Table 44 – Statistical instance and instance part analysis–unit

We have identified the categories of statistical characteristics defined above 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 that were found in the context of audit log analysis i.e., these techniques were less general than what we have identified above.

Example: *We again use the already well-known WebIDS [Almgre99] to illustrate the statistical analysis attributes. WebIDS does not compare statistical information collected for a group of HTTP with the information collected for another group. 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 a HTTP request was made and not on the cost (e.g., number of served bytes) associated with a request. This characterization leads to the following statistical analysis attributes that apply to WebIDS for the HTTP activity scope:*

- *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*

4.4 Representation of IDS descriptions

In the sections above we have described an extensive and flexible taxonomy of IDSes. A couple of IDSes have been described to validate this approach. Their descriptions have been stored in a database in order to provide a rich environment to realize the evaluation of the IDSes themselves. We have implemented this IDS description storage facility using a combination of open-source software that consists of

- a SQL database MySQL [MySql],
- an Apache webserver [Apache],
- a PHP [PHP] scripting interpreter module, and
- 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 obtains significant support in the Linux community and is easy to use and to 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 to populate the DB which we implemented by using the phpMyAdmin package that we operate on an Apache webserver. We have extended the phpMyAdmin slightly to simplify the population of the DB.

4.4.1 Database structure

The database developed in the context of this work consists of three groups of tables. The first group of tables documents and reflects the taxonomy developed in this chapter and the notion of activity scopes including the activity scope graph as introduced in Section 3.1. 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., foreign table keys. The last group of tables has not yet been developed and shall be used to store the results of IDS evaluations.

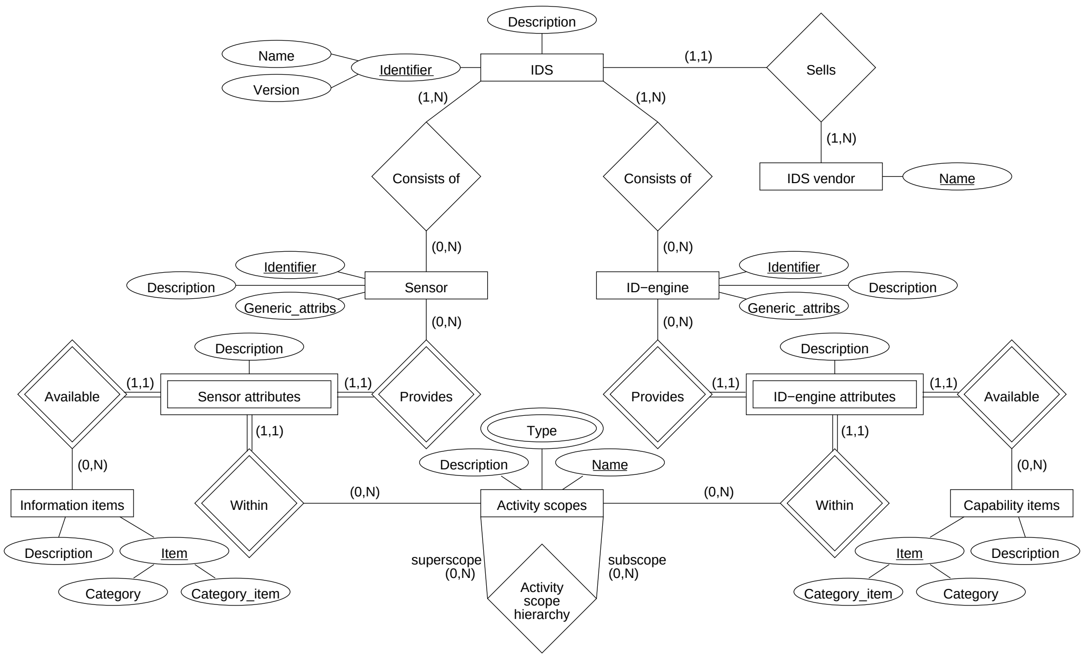


Figure 25 – Entity relationship diagram of the database used to store IDS descriptions

The entity relation diagram shown in Figure 25 captures the first two groups of tables mentioned. The diagram itself is based on the notation by Elmasri and Navathe [ElmNav94].

The high degree of symmetry in the entity relationship diagram is apparent. This is caused by the fact that an IDS entity consists of one or several *sensor* entities and one or several *ID-engine* entities. The representation of the sensor and ID-engine attributes is very similar. Both use the combination of the *activity scope* entity and the sensor, respectively the ID-engine, specific property description entities to represent their properties. In the case of the sensor entity the attribute description entity describes the various *information items* a sensor can potentially provide (see also Section 4.2.2). The attribute description entity affiliated with the ID-engine entity describes all the *capabilities* an IDS could potentially offer for the analysis of activities (see also Sections 4.3.2 and 4.3.3). As indicated in Figure 25 by double lined borders, both attribute entities, i.e. sensor information items and ID-engine capabilities, are so-called weak entities. A weak entity type is defined by the fact that its entries become unique only when considering an externally supplied element as well. In this particular case all the relevant primary key information is supplied externally. In addition to the attribute entities both, the sensor and the ID-engine entity are used to represent the generic attributes as described in Sections 4.2.1 and 4.3.1.

The activity scope entity reflects the concepts introduced in Section 3.1. It is not only used within the context of the representation of IDS subsystem properties but also for the representation of the activity scope hierarchy i.e., the activity scope dependency graph. This is achieved by the introduction of a relation that defines one activity scope to be activity sub-scope of another activity (super-) scope.

Last but not least the entity relationship diagram also requires documentary information to be included in the description of IDSes. A series of description fields 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.

4.5 Discussion

The IDS taxonomy developed in this chapter represents a pragmatic approach to describing IDSes. It is important to note that the potential set of taxonomy elements is used to classify an IDS, otherwise the taxonomy would fail to comply with the criteria listed in Section 2.2.

The taxonomy as it has been developed thus far carries the danger of not being non-ambiguous i.e., not repeatable in the very same manner. This danger arises because of the hierarchical nature of the activity scopes introduced in Section 3.1. The hierarchical nature of the activity scopes allow an IDS to be described at a more specific or a more generic level where both descriptions would be equally valid. This issue can be avoided by first defining a classification policy that defines the level of detail at which IDS shall be classified. One could for example agree that IDSes should be described as generically as possible or as detailed as possible. However, this issue becomes even more complicated due to the fact that detailed information about the capabilities of an (commercial) IDS is often 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. In most cases one can also deduce quite a significant amount of information by considering the information sources used by the IDS more closely. By doing so it is often possible to deduce facts such as the way IP fragments or TCP streams are being processed by an IDS.

Another point worth mentioning is the fact that the distinction between behavior-based and knowledge-based IDSes becomes less obvious. This is due to the fact that most properties used to describe IDSes can be used to describe both of these types of IDSes. We have compensated for this lack by introducing separate properties that allow us to describe the detection method employed by the IDS. This shall have an influence on the semantics of the alarms an IDS model is generating.

4.6 Conclusion

The combination of the taxonomy introduced and the notion of activity scope introduced in Section 3.1 provides us with a flexible and practical instrument to describe IDSes. This taxonomy has been developed to describe the functional aspects i.e., the capabilities of IDSes such that one can in a next step evaluate IDSes for their potential to detect attacks, generate false positives etc.

Chapter 5: Conclusions

5.1 Contributions

In this work we have developed a flexible and extensible way to describe IDSes that is based on a taxonomy which itself is based on the so-called *activity scopes*. The activity scopes are used as a basis not only for the description of IDSes but also to describe the static activity characteristics identified within the context of the *activity taxonomy.* This taxonomy builds the basis for the so-called *activity assumptions*. This concept provides us with a well-structured foundation for the identification of activities that can be used to evaluate IDSes based on their description as developed in this work. The activity taxonomy was developed with the identification of activities that threaten or even cause violation of the security policy e.g., attacks or intrusions in mind. We have then verified the viability of the activity assumptions concept and actually exercised it by means of an *attack classification* where 350 attacks were classified according to the activity taxonomy.

Finally it is our strong belief that this work contributes important concepts derived from the dependability domain to the field of ID and thereby to the goals pursued by MAFTIA.

5.2 Future directions

This work opens several avenues for future work. The natural continuation of this work is the evaluation of IDSes. Based on the attack classification described in this document it becomes possible to identify a representative set of activities that can consecutively be used to evaluate IDSes in a systematic manner.

It is the goal of our future work to develop–within the context of MAFTIA–an intrusion tolerant ID architecture. This architecture shall become dependable i.e., intrusion tolerant, mainly due to the use of diverse IDSes. The immediate continuation of the work described in this document–the evaluation of IDSes–shall enable us to eliminate common failure modes of IDSes used within the ID architecture. Further it shall enable us to maximize the coverage in terms of errors that may lead to security failure detected by the ID architecture.

Appendix A: Attack classification

In Chapter 3 we developed a taxonomy of activities towards the characterization of activities that are security relevant, in the sense that they threaten or violate the security policy. In Section 3.4 we then explained how this taxonomy was used to classify 358 attacks as described in IBM's vulnerability database VulDa.

In the following we provide an extensive discussion of the statistical results that were created based on said attack classification.

A.1 Distribution of dynamic fault characteristics

As mentioned in the case of the dynamic activity characteristics (Section 3.3) we allow the combination of several dynamic activity characteristics to describe an attack i.e., we are using the potential set.

In Figure 26 we only consider the communication model combined with the method invocation model. By considering the communication characteristics (uni- or bi-directional) it is apparent that more than half of the classified attacks involve some communication mean. In fact a large number of attacks only involve bi-directional communication (105 attacks) or execution within object context (45 attacks).

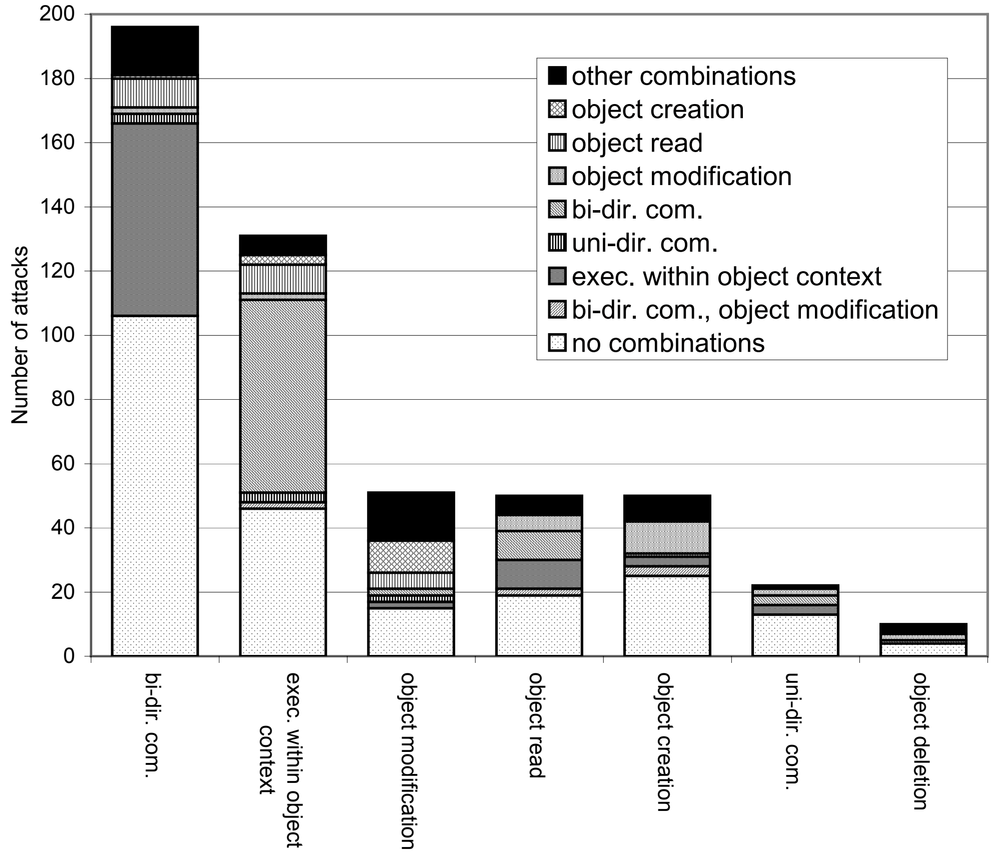


Figure 26 – Histogram of dynamic activity characteristics, excluding attributes

Please note that histograms shown in Figure 26 and Figure 27 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 26 and 86 combinations in the data used for Figure 27. Figure 26 shows only 14 combinations of dynamic activity characteristics, the remaining combinations collected in the group are called *other combinations*. Figure 27 shows 21 different combinations.

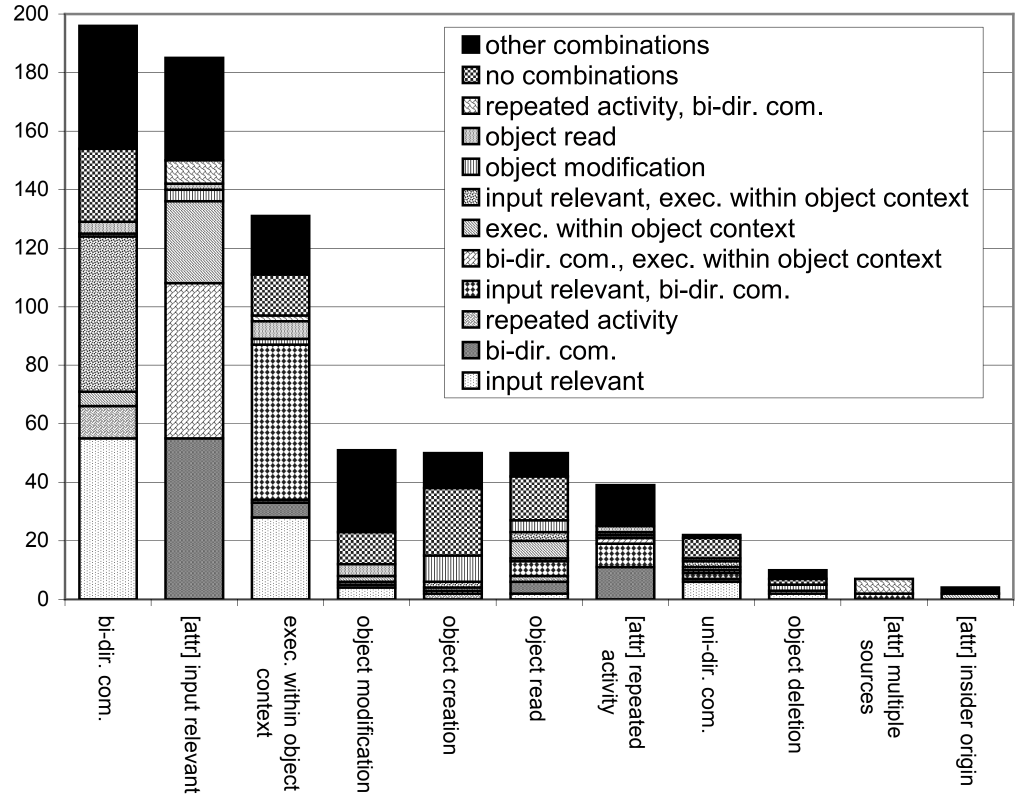


Figure 27 – Histogram of dynamic activity characteristics, including attributes

Figure 27 incorporates the dynamic activity attributes as well (see Section 3.3.3). 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 in 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 and exec. within the object context* portion of the first bar (53 attacks). In a similar way one can identify any kind of potential buffer-overflow or special character attack by considering the second bar (90 attacks). Further combinations of characteristics exist that involve other characteristics in addition to those just mentioned above. For instance, a special character attack against a webserver that reveals the password file also involves the read object characteristics.

A.2 Distribution of interface objects

The histogram shown in Figure 28 shows the distribution of interface objects used to stage attacks. The total size of the bars shows that processes and application protocols are the most frequently used interface objects. Taking a closer look at the respective bars one can further deduce that application layer protocols i.e., all protocol layers and processes, are rarely used in combination to attack another object. This is not surprising and it enables us to distinguish clearly between attacks executed locally and remote attacks.

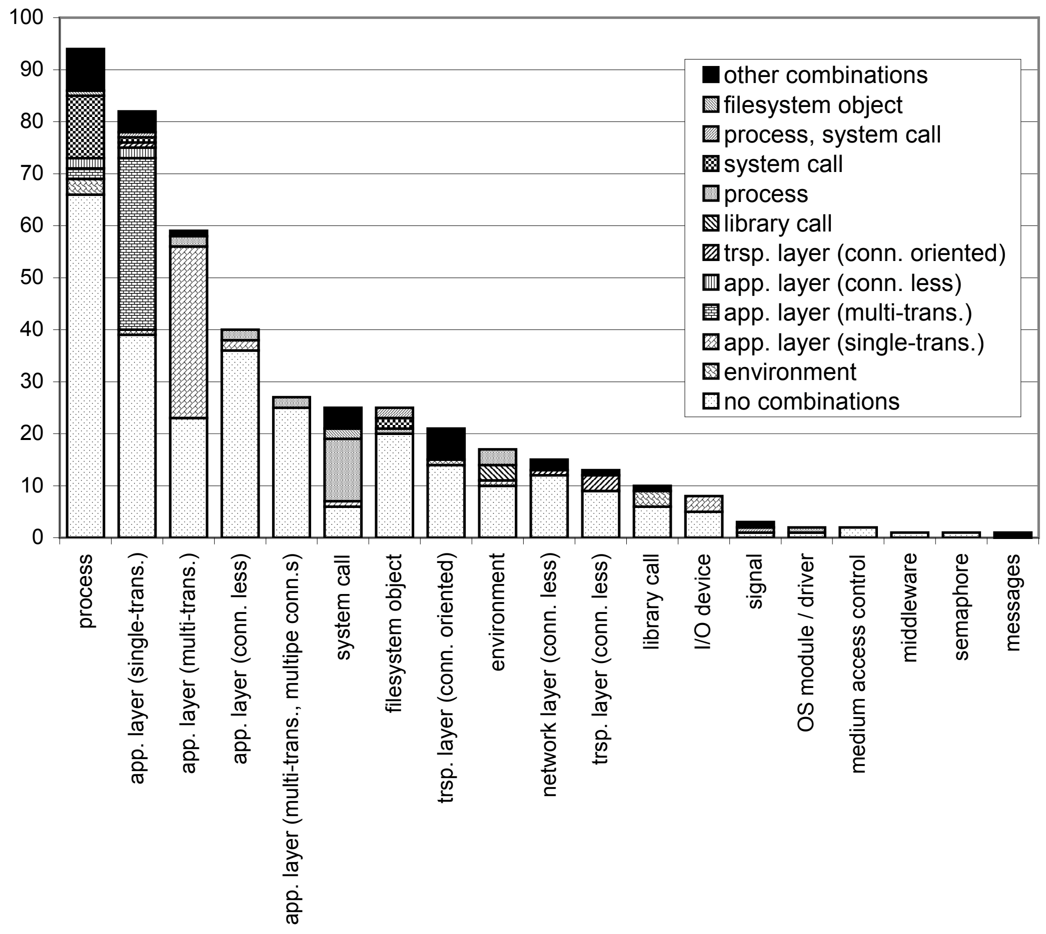


Figure 28 – Histogram of interface objects

Moreover, it is apparent that the filesystem objects and the environment are not used in combination with communication protocol layers to attack another object. This is a fairly reasonable result, because filesystem objects are typically the affected objects of remote attacks i.e., they typically do not serve as interface objects. In the environment things are similar. Although the environment was involved in some remote attacks involving the telnet protocol, remote attacks involving the environment are very rare.

A.3 Distribution of affected objects

The distribution of the affected objects shown in Figure 29 shows the clear dominance of processes. They figure as the most prominent targets of attacks. As to be demonstrated later in more details this corresponds to the observations made in A.2 which showed a large number of attacks involving any kind of application layer protocol. Those attacks typically affect network services that are commonly implemented by daemon processes. Attacks against the filesystem objects are typically targeted towards sensitive files such as the password file.

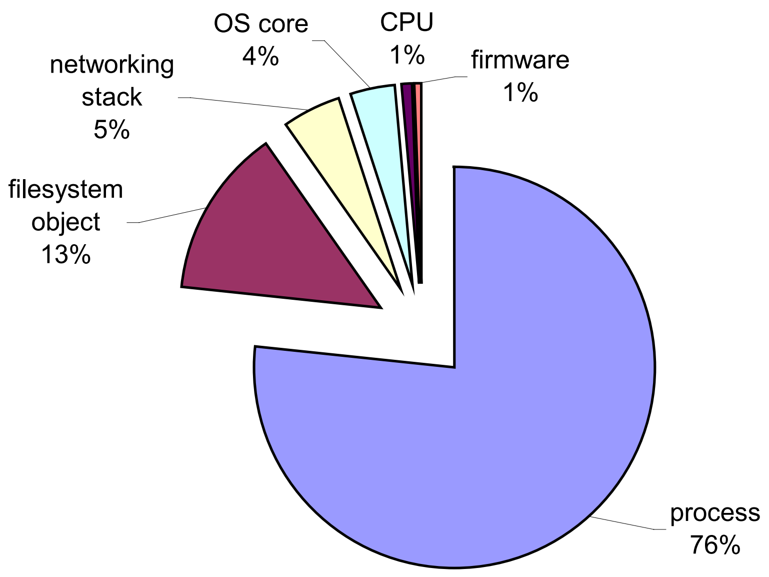


Figure 29 – Distribution of affected objects

Please note, we are not classifying the impact of attacks here. The impact of an attack is described by the failure-state to which an object may be forced. For instance, an attack that is staged against a process or a file may result, in both cases, in command-line level access for the attacker. So the affected object can be a process or the filesystem respectively. However the impact of the successful attack is a (remote) shell being provided for the attacker.

A.4 Dynamic activity characteristics with affected objects

When considering the eight most frequent combinations of dynamic fault characteristics, we obtain a similar picture as in Figure 26. However, it is apparent that the combination of the activity characteristics bi-directional communication and execution within object context is even more frequent than the sole execution within object context. This has already been observed in Section A.1 and it demonstrates the importance of the attacks against (server) processes. The latter can be verified by the fact that almost all the attacks falling into this category, target processes.

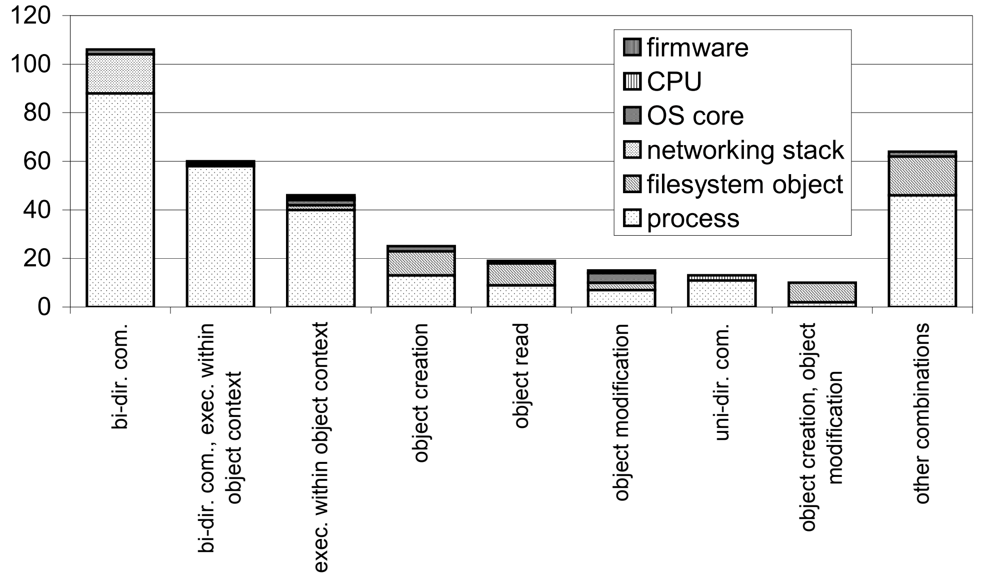


Figure 30 – Histogram of dynamic activity characteristics with affected objects

Moreover we can identify the class of attacks that are using a communication mean to attack the networking stack of system. Most of these attacks are denial-of-service attacks and attempt to crash the whole system by sending some malformed PDUs to the host that put the networking stack into an undefined state, called a failure state when successful. Finally we can identify the important class of attacks that affect the filesystem by means of object creation, modification, or reading. These attacks are typically staged on the local host.

A.5 Interface objects with dynamic activity characteristics

The distribution of the combination of interface objects is less bursty than the one found when considering the affected objects. At first glance, processes are by far the most important class, see first column Figure 31. However, a detailed analysis shows that the classes involving application layer protocols sum up to an even more significant number.

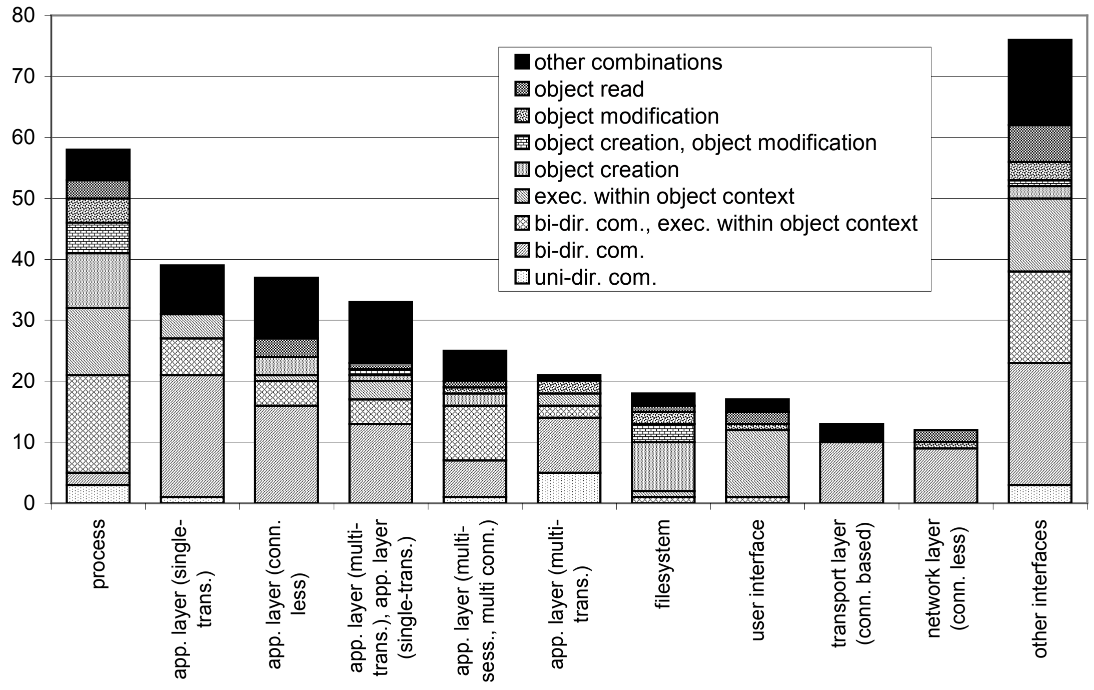


Figure 31 – Histogram of interface objects with dynamic activity characteristics

Further, 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 the columns).

A.6 Interface objects with affected objects

When considering the relationship between affected objects and interface objects we can identify the interfaces most frequently used to attack a given object. Besides the dominance of processes as attack target objects and as attack interface objects we can identify filesystem objects to be attacked merely using a process as an 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.

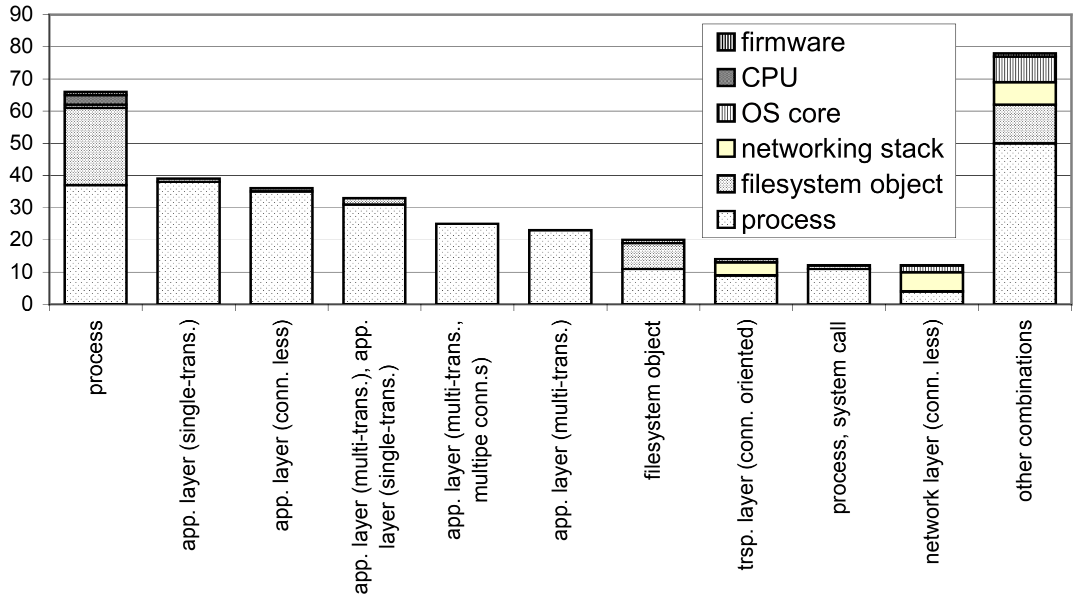


Figure 32 – Histogram of interface objects with affected objects

Besides this, one can observe the CPU to be attacked by processes only, this can be explained by the fact that CPUs are generally attacked by exercising some malicious, possibly invalid, command sequence. In most cases those attacks result in a denial-of-service by crashing the whole system. Another observation one can make is 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.

Appendix B: The vulnerability database

All the work described here is based or at least influenced by the experience and data that was collected within the context of the maintenance process of the vulnerability database, *VulDa*. The daily work with VulDa [DacAle99] and all those new vulnerabilities that are discovered on a daily basis allowed us to gain an in-depth understanding of the nature of real-world security issues and hacking activity in particular.

Over time this database evolved to a rather complex system whose detailed description lies outside the scope of this document. We therefore focus on the core functionality of the database and the data and functionality used in the context of the work described here.

B.1 Motivation and history

Back in 1996 members of the IBM Zurich Research Laboratory started to build-up a repository of attacks. At this point it was difficult to get hold of attacks because they were not generally 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 the so-called vulnerability description files.

B.2 Database structure

The design of the database has improved significantly over time. However, the core of VulDa has remained the same. On the one hand 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, SecurityFocus [SecFoc], SANS [SANS], NIAP [NIAP97], or the CSRC [CSRC] etc. On the other hand vulnerability descriptions provide highly structured information about vulnerabilities and their corresponding attacks.

The population of the database is automated to a high degree and is mostly achieved by unattended batch processes that are gathering data, archiving data, converting data and indexing data. However, the creation of the vulnerability descriptions cannot be automated because their creation requires an in-depth understanding of computer security.

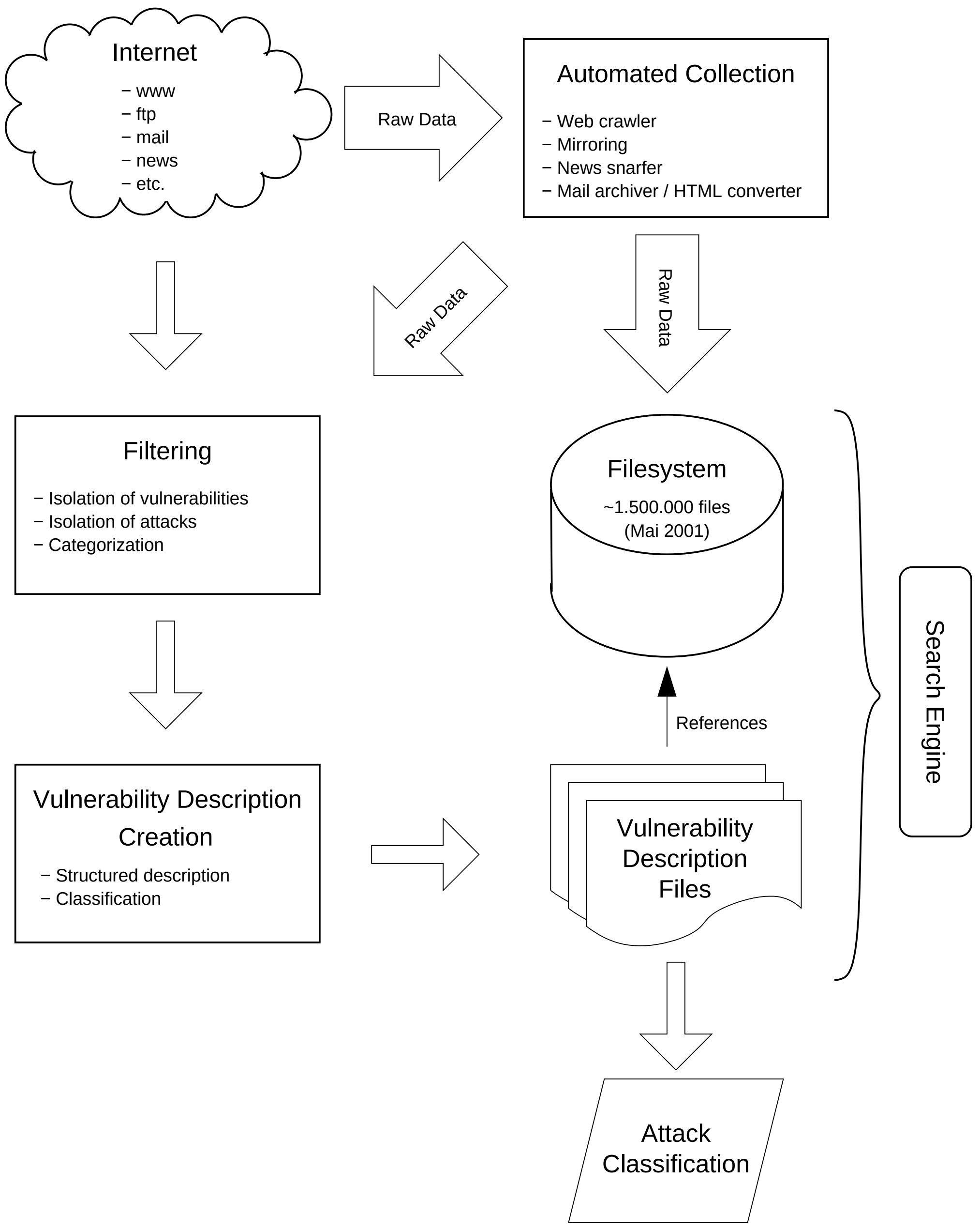


Figure 33 – Data flow in VulDa

Figure 33 shows the data flow within VulDa. As mentioned, for the maintenance, control, state tracking, indexing, and searching a series of processes implemented with tools such as shell scripts, Perl, GDBM (GNU database manager), or MySQL [MySql] are used. The database can be accessed from within IBM by means of a webserver that offers various ways for searching the various categories of documents.

B.3 Vulnerability descriptions

As mentioned, the vulnerability descriptions represent a highly important part of the database. The task of creating vulnerability descriptions requires that the author have a good background in networking, computing systems, and security in general. The data is stored in so-called vulnerability description files that are divided into sections. Those sections contain attribute-value pairs that are used to express the various aspects of the vulnerability and attack described.

As just indicated, VulDa's vulnerability descriptions are well structured and clearly separate vulnerabilities and attacks. Generally speaking one can state that VulDa's vulnerability descriptions represent a super-set of the information provided by other efforts established just recently. Examples of similar efforts are ICAT [MBDRM] or the Bugtraq ID pursued by SecurityFocus [SecFoc].

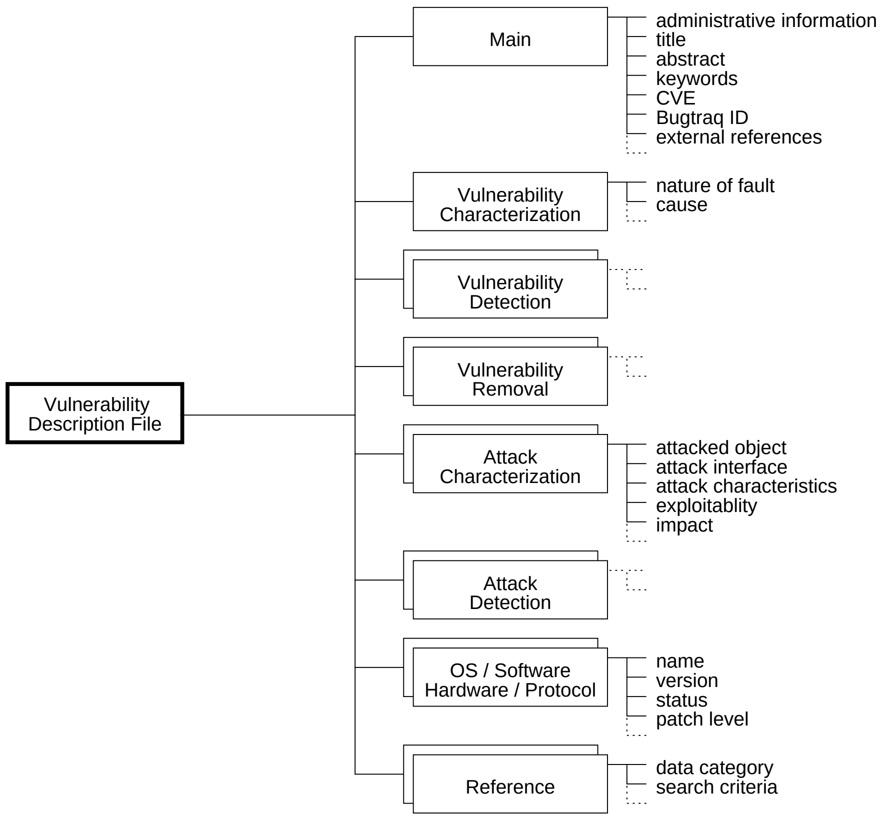


Figure 34 – Overview of the vulnerability description structure

In the following we briefly discuss the purpose of the various sections shown in Figure 34. We only highlight the details believed to be of general interest or that are used within the context of the overall work described here.

The *main* section of the vulnerability description files contains generic information about the document. Besides information used for administrative purposes this section also contains the document title, abstract, keywords, external references, and last but not least CVE identifiers and Bugtraq IDs (see also Section 2.2.3.1).

In the section *vulnerability characterization* the vulnerability is classified according to various criteria such as the cause of the fault e.g., design fault, implementation fault etc. Further the vulnerability is classified according to commonly well known fault characteristics such as insufficient input validation, privilege abuse etc. Among other criteria this section also denotes the system component where the fault is located.

The *vulnerability detection* and *vulnerability removal* sections are not explained in detail here. However, it is worth noting that the section describing the detection of the vulnerability also includes information about testing tools e.g. a scanning tool and the way those tools would report the vulnerability. The *vulnerability removal* section describes the various ways in which the vulnerability can be removed e.g., disabling of a service, reconfiguration etc. It is permitted to include several of the sections just described within one vulnerability description simply because there might be several different ways to detect a vulnerability or to remove a vulnerability.

The *attack characterization* section characterizes one of the ways the previously described vulnerability can be exploited i.e., the way the fault can be activated. Besides attributes that describe the immediate impact of the attack, the exploitability (remote or local), prerequisites etc., this section contains the information that was used to classify attacks. This information formed the basis for the results presented in Section 3.4 and in Appendix A. As there is often more than one way to exploit a given vulnerability, several attack characterization sections are supported.

The *attack detection* section consists of properties which allow us to specify how one can detect an attack using a given IDS. Because the vulnerability description may contain the description of several attacks and because various existing IDSes may detect a given attack in different ways, several attack detection sections are allowed.

The *OS, software, hardware,* and *protocol* sections have an almost identical structure. They are used describe 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. In this way it becomes possible to describe the difference in terms of version and patches between vulnerable and safe versions of a product or protocol i.e., the vulnerable and safe versions can be described 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 yet another patch will remove the vulnerability again.

The *reference* sections are used to link the vulnerability description with further information sources. It is possible to provide references to every document found in the database– including other vulnerability descriptions, exploits, advisories, RFCs etc. It is important and completely natural that information about a given vulnerability be discovered and published gradually. It also seems obvious that the frequent updating of references in several hundreds or thousands of vulnerability descriptions is not feasible in practice. This led us to a solution that was proven to be very efficient from a maintenance standpoint and also from a usability standpoint. Taking advantage of the fact that the documents in VulDa can be searched by categories we included search patterns per category in the reference sections that are resolved at runtime. This allows us to refer very efficiently and in an always up-to-date manner to a whole series of subsequent documents e.g. a mailing list thread or a series of advisories. Experience has shown that given that the search patterns were formulated carefully enough in the first place, even over time the number of non-relevant document references generated can be kept to an almost negligible minimum. To further extend the expressiveness of the references generated it is possible to rank the generated references.

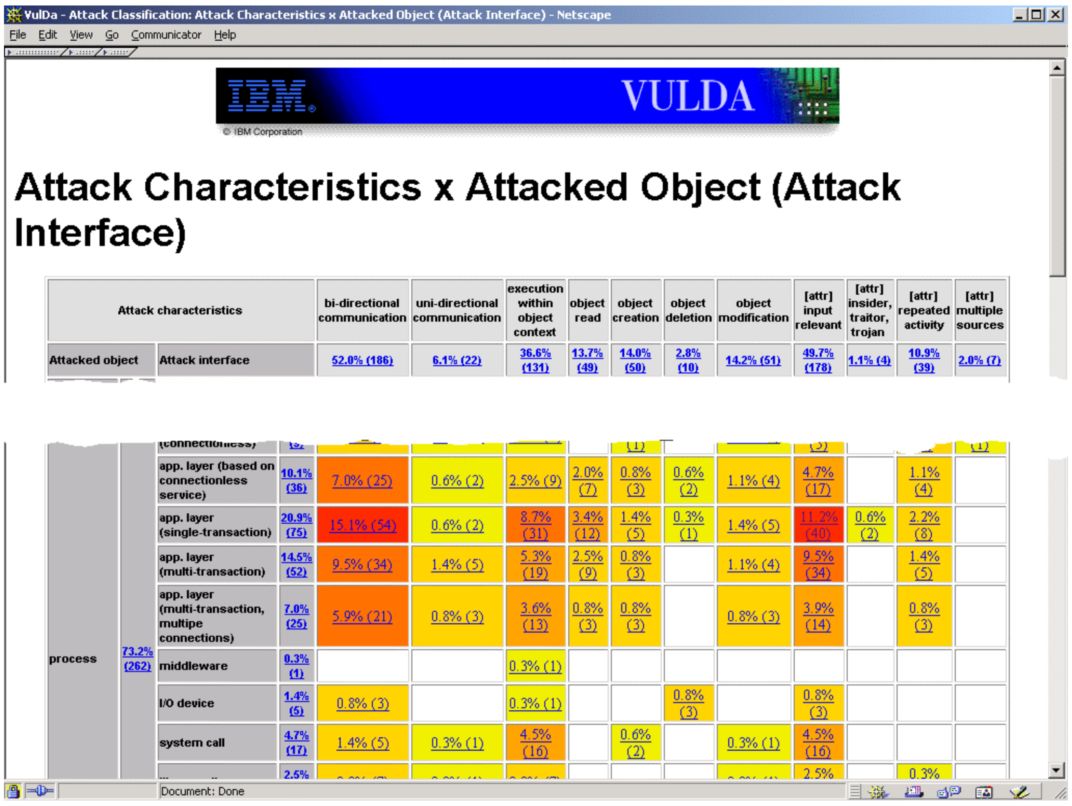
B.4 Results

Over the years the VulDa database has grown into a large repository of security relevant documents separated by categories such as advisories, vulnerability descriptions, attacks, tools etc. VulDa offers a flexible search facility that allows the searching of documents by categories or the browsing of vulnerability descriptions by various criteria such as OS etc. Using the same infrastructure, security tools such as a network security scanner were integrated.

It should be noted that the technology used to implement this database was not the most recent available. It was not the goal of this project to demonstrate what the most advanced software products are able to do. Besides using public domain software such as Perl [Perl87] we were pragmatically focusing our efforts on the concepts and the content of the database.

B.4.1 Attack classification

The classification of attacks described in vulnerability descriptions is the most visible result achieved with respect to this work. The data obtained by this classification was used to generate the results discussed in Section 3.4.


VulDa - Attack Classification: Attack Characteristics x Attacked Object (Attack Interface) - Netscape
File Edit View Go Communicator Help
IBM
VULDA
© IBM Corporation
Document: Done

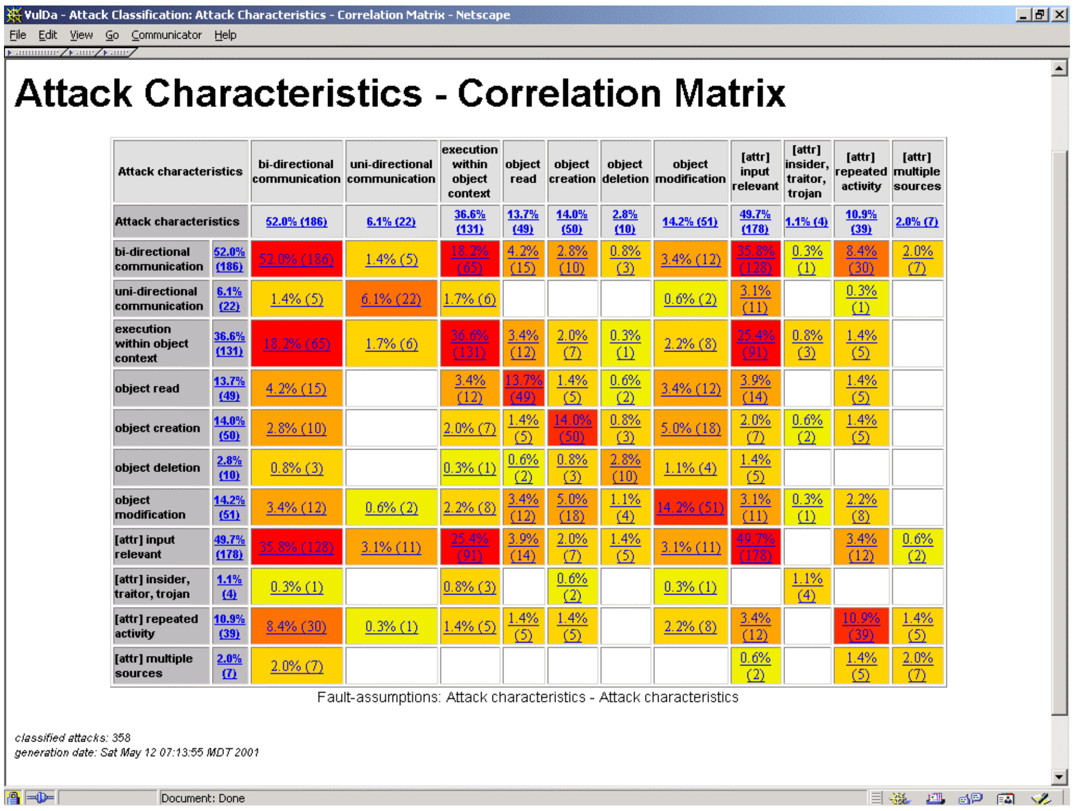
Attack Characteristics x Attacked Object (Attack Interface)

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
Attacked object		Attack interface		52.0% (186)	6.1% (22)	36.6% (131)	13.7% (49)	14.0% (50)	2.8% (10)	14.2% (51)	49.7% (178)	1.1% (4)	10.9% (39)	2.0% (7)
		(connectionless)	[illegible]	[illegible]	[illegible]	[illegible]		(1)		[illegible]	(5)		[illegible]	(1)
		app. layer (based on connectionless service)	10.1% (36)	7.0% (25)	0.6% (2)	2.5% (9)	2.0% (7)	0.8% (3)	0.6% (2)	1.1% (4)	4.7% (17)		1.1% (4)	
		app. layer (single-transaction)	20.9% (75)	15.1% (54)	0.6% (2)	8.7% (31)	3.4% (12)	1.4% (5)	0.3% (1)	1.4% (5)	11.2% (40)	0.6% (2)	2.2% (8)	
		app. layer (multi-transaction)	14.5% (52)	9.5% (34)	1.4% (5)	5.3% (19)	2.5% (9)	0.8% (3)		1.1% (4)	9.5% (34)		1.4% (5)	
		app. layer (multi-transaction, multipe connections)	7.0% (25)	5.9% (21)	0.8% (3)	3.6% (13)	0.8% (3)	0.8% (3)		0.8% (3)	3.9% (14)		0.8% (3)	
process	73.2% (262)	middleware	0.3% (1)			0.3% (1)								
		I/O device	1.4% (5)	0.8% (3)		0.3% (1)			0.8% (3)		0.8% (3)			
		system call	4.7% (17)	1.4% (5)	0.3% (1)	4.5% (16)		0.6% (2)		0.3% (1)	4.5% (16)			
		[illegible]	2.5%	[illegible]	[illegible]	[illegible]				[illegible]	2.5%		0.3%	

Figure 35 – Statistics of attack classification superposing attacked object, attack interface and attack characteristics

Besides the statistics discussed in Section 3.4 we have produced a series of HTML tables shown in Figure 35 and Figure 36 that support the evaluation of IDSes in a practical manner. The tables not only provide an overview of the most frequent classes of attacks but they also allow us to obtain the list of vulnerabilities belonging to a given class by clicking on the corresponding table field. This functionality simplifies the task of describing activities that belong to a given class and supports us in systematically identifying the activities to be used for the IDS evaluation.

Figure 35 shows a table that combines the static activity characteristics discussed in Section 3.2 (rows of the table) and the dynamic activity characteristics discussed in Section 3.3. Please note that every attack may qualify for several static and dynamic activity characteristics. This means that vulnerability descriptions might qualify for several class fields in the table–which is a completely valid situation.


VulDa - Attack Classification: Attack Characteristics - Correlation Matrix - Netscape
File Edit View Go Communicator Help

Attack Characteristics - Correlation Matrix

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		52.0% (186)	6.1% (22)	36.6% (131)	13.7% (49)	14.0% (50)	2.8% (10)	14.2% (51)	49.7% (178)	1.1% (4)	10.9% (39)	2.0% (7)
bi-directional communication	52.0% (186)	52.0% (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	36.6% (131)	18.2% (65)	1.7% (6)	36.6% (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	2.0% (7)	2.0% (7)							0.6% (2)		1.4% (5)	2.0% (7)

Fault-assumptions: Attack characteristics - Attack characteristics

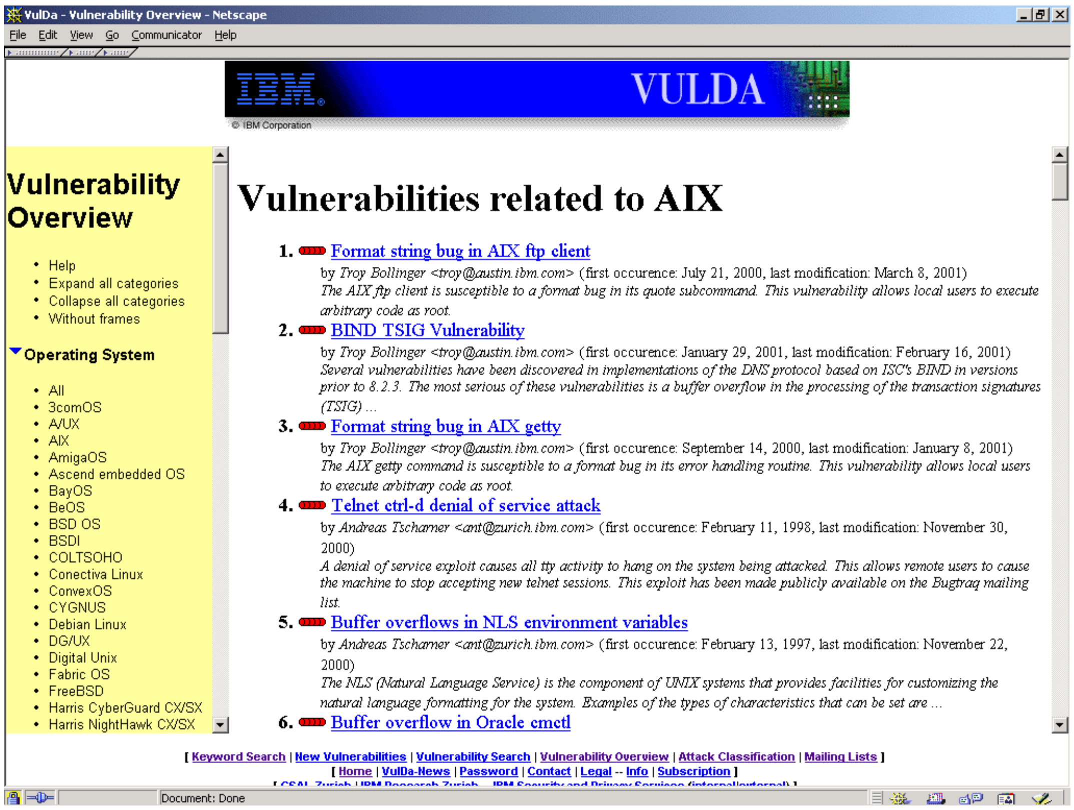
classified attacks: 358
generation date: Sat May 12 07:13:55 MDT 2001

Figure 36 – Statistics of concurrent occurrences of attack characteristics

Figure 36 shows a different example of a table that is generated based on the classified attacks. In this example one can find the various ways the dynamic activity characteristics i.e., attack characteristics in the figure, are combined within one attack. Please note that this table reflects only combinations of two characteristics per field. This means that attack classes combining three or more attack characteristics do not appear in this table.

B.4.2 Vulnerability browser

The vulnerability browser–also called vulnerability overview–provides a browser-stylish interface that enables a user to search for categories of vulnerabilities and attacks easily. The interface is implemented based on the mentioned search engine. As shown in Figure 37 and Figure 38, 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.


VulDa - Vulnerability Overview - Netscape
File Edit View Go Communicator Help
IBM
VULDA
© IBM Corporation
Document: Done

Vulnerability Overview

- Help
- Expand all categories
- Collapse all categories
- Without frames

Operating System

- All
- 3comOS
- A/UX
- AIX
- AmigaOS
- Ascend embedded OS
- BayOS
- BeOS
- BSD OS
- BSDI
- COLTSOHO
- Conectiva Linux
- ConvexOS
- CYGNUS
- Debian Linux
- DG/UX
- Digital Unix
- Fabric OS
- FreeBSD
- Harris CyberGuard CX/SX
- Harris NightHawk CX/SX

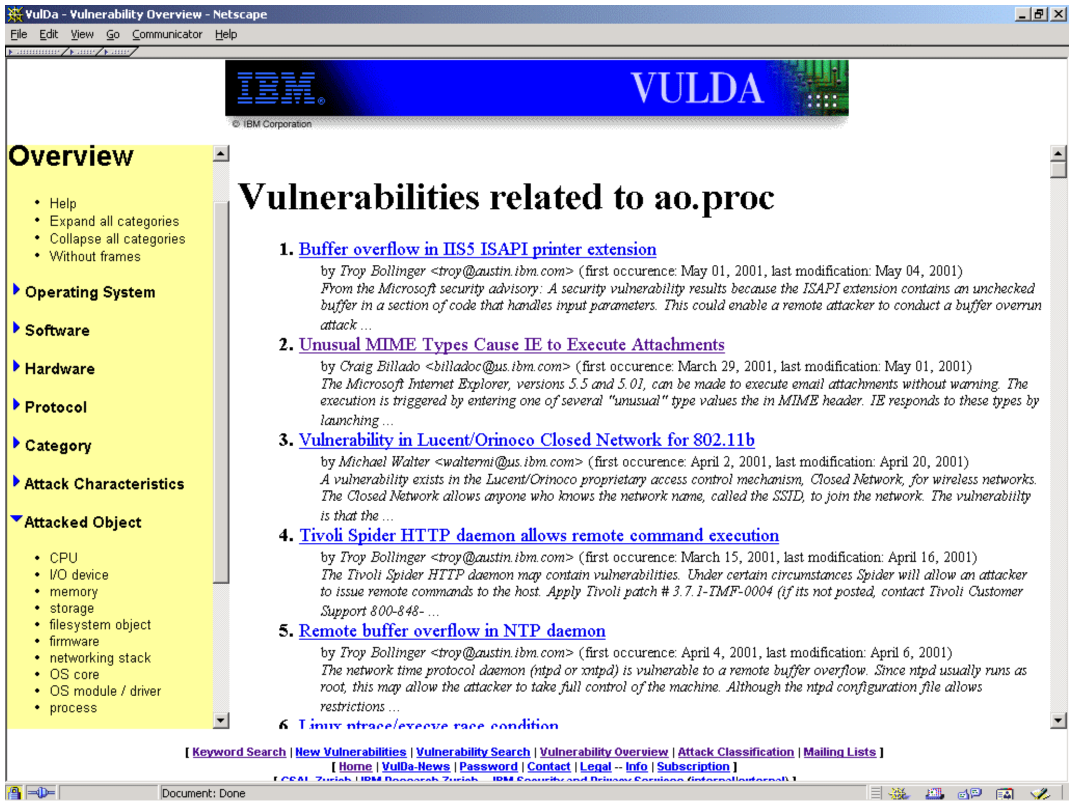
Vulnerabilities related to AIX

1. Format string bug in AIX ftp client
by *Troy Bollinger <troy@austin.ibm.com>* (first occurence: July 21, 2000, last modification: March 8, 2001)
The AIX ftp client is susceptible to a format bug in its quote subcommand. This vulnerability allows local users to execute arbitrary code as root.
2. BIND TSIG Vulnerability
by *Troy Bollinger <troy@austin.ibm.com>* (first occurence: January 29, 2001, last modification: February 16, 2001)
Several vulnerabilities have been discovered in implementations of the DNS protocol based on ISC's BIND in versions prior to 8.2.3. The most serious of these vulnerabilities is a buffer overflow in the processing of the transaction signatures (TSIG) ...
3. Format string bug in AIX getty
by *Troy Bollinger <troy@austin.ibm.com>* (first occurence: September 14, 2000, last modification: January 8, 2001)
The AIX getty command is susceptible to a format bug in its error handling routine. This vulnerability allows local users to execute arbitrary code as root.
4. Telnet ctrl-d denial of service attack
by *Andreas Tscharner <ant@zurich.ibm.com>* (first occurence: February 11, 1998, last modification: November 30, 2000)
A denial of service exploit causes all tty activity to hang on the system being attacked. This allows remote users to cause the machine to stop accepting new telnet sessions. This exploit has been made publicly available on the Bugtraq mailing list.
5. Buffer overflows in NLS environment variables
by *Andreas Tscharner <ant@zurich.ibm.com>* (first occurence: February 13, 1997, last modification: November 22, 2000)
The NLS (Natural Language Service) is the component of UNIX systems that provides facilities for customizing the natural language formatting for the system. Examples of the types of characteristics that can be set are ...
6. Buffer overflow in Oracle cmctl

[Keyword Search | New Vulnerabilities | Vulnerability Search | Vulnerability Overview | Attack Classification | Mailing Lists]
[Home | VulDa-News | Password | Contact | Legal -- Info | Subscription]
[CSAL Zurich | IBM Research Zurich -- IBM Security and Privacy Services (internal/external)]

Figure 37 – Vulnerability browser–example with the operating system AIX

Figure 37 shows an example that lists all vulnerabilities known to affect IBM's AIX operating system.


VulDa - Vulnerability Overview - Netscape
File Edit View Go Communicator Help
IBM
VULDA
© IBM Corporation
Document: Done

Overview

- Help
- Expand all categories
- Collapse all categories
- Without frames

Operating System

Software

Hardware

Protocol

Category

Attack Characteristics

Attacked Object

- CPU
- I/O device
- memory
- storage
- filesystem object
- firmware
- networking stack
- OS core
- OS module / driver
- process

Vulnerabilities related to ao.proc

1. Buffer overflow in IIS5 ISAPI printer extension
by *Troy Bollinger <troy@austin.ibm.com>* (first occurence: May 01, 2001, last modification: May 04, 2001)
From the Microsoft security advisory: A security vulnerability results because the ISAPI extension contains an unchecked buffer in a section of code that handles input parameters. This could enable a remote attacker to conduct a buffer overrun attack ...
2. Unusual MIME Types Cause IE to Execute Attachments
by *Craig Billado <billadoc@us.ibm.com>* (first occurence: March 29, 2001, last modification: May 01, 2001)
The Microsoft Internet Explorer, versions 5.5 and 5.01, can be made to execute email attachments without warning. The execution is triggered by entering one of several "unusual" type values the in MIME header. IE responds to these types by launching ...
3. Vulnerability in Lucent/Orinoco Closed Network for 802.11b
by *Michael Walter <walterml@us.ibm.com>* (first occurence: April 2, 2001, last modification: April 20, 2001)
A vulnerability exists in the Lucent/Orinoco proprietary access control mechanism, Closed Network, for wireless networks. The Closed Network allows anyone who knows the network name, called the SSID, to join the network. The vulnerabilty is that the ...
4. Tivoli Spider HTTP daemon allows remote command execution
by *Troy Bollinger <troy@austin.ibm.com>* (first occurence: March 15, 2001, last modification: April 16, 2001)
The Tivoli Spider HTTP daemon may contain vulnerabilities. Under certain circumstances Spider will allow an attacker to issue remote commands to the host. Apply Tivoli patch # 3.7.1-TMF-0004 (if its not posted, contact Tivoli Customer Support 800-848- ...
5. Remote buffer overflow in NTP daemon
by *Troy Bollinger <troy@austin.ibm.com>* (first occurence: April 4, 2001, last modification: April 6, 2001)
The network time protocol daemon (ntpd or xntpd) is vulnerable to a remote buffer overflow. Since ntpd usually runs as root, this may allow the attacker to take full control of the machine. Although the ntpd configuration file allows restrictions ...
6. Linux ptrace/execve race condition

[Keyword Search | New Vulnerabilities | Vulnerability Search | Vulnerability Overview | Attack Classification | Mailing Lists]
[Home | VulDa-News | Password | Contact | Legal -- Info | Subscription]
[CSAL Zurich | IBM Research Zurich -- IBM Security and Privacy Services (internal/external)]

Figure 38 – Vulnerability browser–example with the attacked object process

Figure 38 shows an example that lists all vulnerabilities where it is known that the corresponding attacks are affecting processes. This example makes use of the classification made for the validation of the activity assumptions developed within the context of this work.

B.4.3 Integration with security software

The flexibility of the vulnerability description files and the search engine used on VulDa enabled us to integrate security software such as network security scanner with the database. In the example mentioned, the security auditing tool would generate an HTML report containing 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 a vulnerability identifier such as CVE numbers or Bugtraq IDs. By clicking on the link VulDa returns the list vulnerabilities matching the search criteria specified by any of the mentioned ID types.

Appendix C: Examples of IDS characteristics with respect to activity scopes

The examples provided in the following are given to illustrate the way the pairs of attributes and activity scopes are interpreted. The following tables are therefore by no mean meant to cover the complete activity scope graph.

In some cases several semantic interpretation seem possible. In such cases we generally choose and document the one that seems most suitable for the evaluation of IDSes and ID in general.

C.1 IDS sensor characteristics

Activity scope / attributes	**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[10] e.g., signal name.	Denotes a unique device name.	The IDS sensor is able to gather the name of a middlware object from the network[11].	The IDS sensor is able to gather the name of a middlware object on the host.	Denotes the name of the executable that was 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 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 45 – IDS sensor characteristics – Objects (see also Table 26)

[10] This is not be confused with IPC objects linked to a filesystem object as done on Unix systems.

[11] 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 an IDS sensor perspective this difference does matter because the sensor may be monitoring object operations on a host or on the network only.

Activity scope / attributes	**User**	**Filesystem object**	**IPC**	**Device**
Type	Indicates the role of a user e.g., the fact that the 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.

Table 46 – IDS sensor characteristics–Object attributes (see also Table 27)

Activity scope / attributes	**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. Typical examples are the HTTP header fields that follow the HTTP request statement.

Table 47 – IDS sensor characteristics–Arguments (see also Table 28)

Activity scope / attributes	**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 48 – IDS sensor characteristics–Request (see also Table 29)

Activity scope / attributes	**MAC layer**	**Network layer**	**Transport layer**
Source / destination ID	In the MAC layer context the source / destination ID denotes the MAC address of the PDU sender / destination.	In the network layer context the source / destination ID denotes the network address of the sender / destination e.g., IP address.	In the transport layer context the source / destination ID generally denotes the transport layer source / destination address e.g., TCP port number.
Source / destination name	N/A	In the network layer context the source / destination ID denotes the network name of a system e.g., hostname as stored in DNS.	In the transport layer context 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 possibly to extend the header information with fields for source routing etc.	Protocols such as TCP use optional header fields to negotiate connection parameters.

Table 49 – IDS sensor characteristics–Protocol control data (see also Table 30)

Activity scope / attributes	**Filesystem object**	**Process**	**Device**	**TCP**
Up- / down-stream	N/A	N/A	N/A	TCP provides a reliable bi-directional 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 activity scope the status data represents the content of a filesystem object e.g., the content of a file.	The status data of a process can be seen as the memory image of the process.	In the device context the content of physical memory or of a storage device can be considered status data. This includes the status of device registers.	

Table 50 – IDS sensor characteristics–Data (see also Table 31)

C.2 ID engine data pre-processing characteristics

Activity scope / attributes	**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 being accessed, but it may be able to obfuscate the attack from an IDS. It is also 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., the infamous pipe attack, CVE-1999-0203.	The arguments passed to a system or function call may be quoted in an unusual way.	Filenames may be escaped in various ways.
Data decoding	In HTTP POST data is often encoded in base64.	N/A	Mail messages, particularly the attachments, are often base64 encoded.	N/A	The content of files may be compressed.

Table 51 – ID engine characteristics–data normalization (see also Table 33)

C.3 ID engine instance analysis characteristics

C.3.1 Instance analysis levels

Activity scope / attributes	**Call**	**Process**	**Transport / network / link layer**	**Application layer**
Basic analysis	The ID engine is able to recognize calls.	The ID engine is able to identify a thread of a process. If a process consists of a single thread only this also falls into this category. Basic instance part analysis for processes is for example required in combination with system call sequence analysis, where the ID engine 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 ID engine 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 ID engine verifies the plausibility and the policy compliance of the instance part.

Table 52 – Single instance part analysis

Activity scope / attributes	**Call**	**Process**	**Transport / network layer**	**Application layer**
Basic analysis	N/A	The ID engine is able to identify a process and its threads.	The ID engine 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 ID engine is able to identify a group of protocol statements i.e., a transaction.
Logic verification	N/A	N/A	The ID engine is able to recompose the instance from its parts by following the protocol specification. Further the ID engine is able to recognize suspicious protocol sequences e.g. stealth TCP scanning [CIN0498].	The ID engine is able to verify the correctness of the protocol sequence.
Semantic verification	The ID engine is able to analyze the impact of a call e.g. to verify its return values to be acceptable with respect to the call arguments.	N/A	Inconsistent but logically correct parts are recognized. Overlapping fragments are recognized. Connection segment retransmissions or overlapping fragments that no longer contain the same data are also recognized.	In the case of HTTP one expects the ID engine to be able to identify the fact that a protected document was revealed. In the case of SMTP one would expect the ID engine to recognize the fact that a confidential document is sent to a receiver outside of the organization.

Table 53 – Single instance analysis

Activity scope / attributes	**Call**	**Process**	**Application layer**
Basic analysis	The ID engine is able to associate calls to a process or to a user. This may additionally require some degree of process analysis.	The ID engine is able to identify a process group.	The ID engine 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 simplistic check.
Logic verification	The ID engine is able to verify the logical correctness of a sequence of calls made by a process or by a user.	N/A	The ID engine is able to analyze the instances i.e. transactions executed within a session independently.
Semantic verification	The ID engine 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 there is dependency among the instances of a group, the ID engine is able to verify consistency of this dependency and to verify the acceptability of sequences.

Table 54 – Instance group analysis

Activity scope / attributes	Call	**Transport *<u></u>/ network / link* layer**	**Application layer**
Basic analysis	N/A	N/A	N/A
Logic verification	N/A	N/A	N/A
Semantic verification	The ID engine is able to verify the semantic correctness and acceptability of a sequence of calls (the calls may be made by several independent processes).	The ID engine is able to identify unacceptable sequences of instances.	If there is dependency among instances, the ID engine is able to verify consistency of this dependency and to verify the acceptability of sequences.

Table 55 – Cross-instance (multi-instance) analysis

Activity scope / attributes	**Connection oriented transport / network / link layer**	**Application layer**
Basic analysis	The ID engine is able to associate PDUs flowing in both directions.	The ID engine is able to identify the server response.
Logic verification	The ID engine is able to verify that the PDUs flowing in both directions are consistent with respect to the protocol definition.	The ID engine 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	Is able to verify the acceptability of the bi-directional instance observed e.g., the ID engine is able to detect the attempt of TCP connection hijacking.	The ID engine is able to verify that the server's response is acceptable with respect to the request sent by the client.

Table 56 – Bi-directional instance part group analysis

Tables

Figures

References

[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.

[Almgre99] Magnus Almgren, "Design and Implementation of a Lightweight Tool for Detecting Web Server Attacks," Master's thesis, Uppsala: University of Uppsala, Sweden, Department of Scientific Computing, 1999, pp. 60.

[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.

[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.

[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.

[Axelss99] Stefan Axelsson, "The Base-rate Fallacy and its Implications for the Difficulty of Intrusion Detection," presented at 6th ACM Conference on Computer and Communications Security, Singapore, 1999.

[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.

[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.

[CA1201] CERT Coordination Center, "Superfluous Decoding Vulnerability in IIS," *http://www.cert.org/advisories/CA-2001-12.html*, 2001, last update: May 15, 2001.

[CA1395] CERT Coordination Center, "Syslog Vulnerability - A Workaround for Sendmail," CERT Coordination Center, Pittsburgh, PA, Advisory CA-95.13, 1995.

[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.

[CheBel94] W. R. Cheswick and S. M. Bellovin, *Firewalls and Internet Security - Repelling the Wily Hacker*. Reading, MA: Addison-Wesley Publishing Company, 1994.

[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.

[Cohen95] F. B. Cohen, *Protection and Security on the Information Superhighway*. New York: John Wiley & Sons, 1995.

[CSRC] National Institute of Standards and Technology (NIST), "Computer Security Resource Center (CSRC)," *http://csrc.nist.gov/.*

[CVE99] The MITRE Corporation, "Common Vulnerabilities and Exposures," *http://cve.mitre.org/*, 1999.

[CVE033301] CVE editorial board, "CAN-2001-0333," *http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2001-0333*, 2001.

[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, (in preparation), 2001.

[DacAle99] Marc Dacier and Dominique Alessandri, "VulDa: A Vulnerability Database," presented at 2nd Workshop on Research with Security Vulnerability Databases, Purdue University, IN, 1999.

[DCWMS99] Robert Durst, Terrence Champion, Brian Witten, Eric Miller, and Luigi Spanguolo, "Testing and Evaluating Computer Intrusion Detection Systems," *Comm. of ACM*, vol. 42, July 1999.

[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.

[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.

[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.

[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.

[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.

[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: June 7, 2001.

[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.

[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, 1999.

[JiSiIr00] Jitsu-Disk, Simple Nomad, and Irib, "Delirium Tremens," *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.

[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 Krsul, Eugene Spafford, and Mahesh Tripunitara, "Computer Vulnerability Analysis," COAST Laboratory, Purdue University, West Lafayette, IN, Technical Report 1998.

[Krsul98] Ivan Victor 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.

[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.

[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.

[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.

[MBDRM] Peter Mell, Elizabeth Boteler, Derek Dye, Michael Reilly, and David Marks, "ICAT Metabase," *http://icat.nist.gov/.*

[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.

[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.

[MeyPra87] F. Meyer and D. Pradhan, “Consensus with Dual Failure Modes,” presented at The 17th International Symposium on Fault-Tolerant Computing Systems, Pittsburgh, PA, 1987, pp. 214--22.

[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.

[OstWey] 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.

[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.

[PBSVW88] D. Powell, G. Bonn, D. Seaton, P. Verissimo, and F. Waeslynck, "The Delta-4 Approach to Dependability in Open Distributed Computing Systems," presented at 18th IEEE International Symposium on Fault-Tolerant Computing Systems (FTCS-18), Tokyo, Japan, 1988, pp. 246--51.

[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.

[Powell95] David Powell, "Failure Mode Assumptions and Assumption Coverage: A Revised Version," LAAS-CNRS, Toulouse, France, Research Report 91462, *ftp://ftp.laas.fr/pub/Publications/1991/91462.ps,* 1995.

[Power96] R. Power, "Current and Future Danger: A CSI Primer of Computer Crime & Information Warefare," CSI Bulletin 1996.

[Roesch99] Marty Roesch, "Snort - The Lightweight Open Source Network Intrusion Detection System," *http://www.snort.org/*, 1999.

[SANS] Consortium, "SANS (System Administration, Networking, and Security) Institute," *http://www.sans.org/.*

[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.

[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.

[Sobire98] Michael Sobirey, "Michael Sobirey's Intrusion Detection Systems page," *http://www-rnks.informatik.tu-cottbus.de/~sobirey/ids.html*, November 1998.

[Spaffo88] E. H. Spafford, "The Internet Worm Program: An Analysis," Purdue University, Technical 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.

[Sundar96] Aurobindo Sundaram, "An Introduction to Intrusion Detection," *ACM Crossroads Student Magazine*, pp. 10*, http://www.acm.org/crossroads/xrds2-4/intrus.html*, 1996.

[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 generateion internet*, first ed. New York: John Wiley & Sons, Inc., 1996,

ISBN 0-471-13088-5.

[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.

[VMware00] Inc. VMware, "VMware Workstation v2.0," *http://www.vmware.com/,* 2000.

[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.zurich.ibm.com/pub/Other/RAID/Prog_RAID98/Table_of_content.html*, 1998.

[Webster] Website, "Merriam-Webster's Collegiate Dictionary," *http://www.britannica.com/bcom/dictionary/.*

[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.

[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.

[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.