Project IST-1999-11583

**Malicious- and Accidental-Fault Tolerance for Internet Applications**


# Design of an Intrusion-Tolerant Intrusion Detection System

M. Dacier (Editor)

IBM Zurich Research Laboratory

**MAFTIA deliverable D10**

Public Document

Version 4.3

August 9, 2002

# Revisions

| Rev. | Date | Comment |
|---|---|---|
| 1.0 | 21.09.2001 | Outline of document |
| 1.1 | 18.10.2001 | Revised Outline of document |
| 1.2 | 10.02.2002 | Modifications to put text in sync with D2 |
| 1.3 | 20.04.2002 | Modification on Section related to THOR |
| 2.0 | 10.05.2002 | Major revision on the sections related to false alarm handling |
| 2.1 | 04.06.2002 | Major revision on the section related to RIDAX |
| 2.2 | 18.06.2002 | Major revision of the last chapter |
| 3.0 | 23.06.2002 | Minor revision of the whole text |
| 4.0 | 09.07.2002 | Added middleware example, numerous corrections |
| 4.1 | 17.07.2002 | Numerous corrections |
| 4.2 | 18.07.2002 | Improved references |
| 4.3 | 09.08.2002 | Minor corrections |

# Editor

Marc Dacier

# Contributors

Dominique Alessandri
Christian Cachin
Marc Dacier
Yves Deswarte
Klaus Julisch
Klaus Kursawe
Raffael Marty
David Powell
Brian Randell
James Riordan
Andreas Wespi

# Address

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

# Table of contents

# Chapter 1 Introduction

## 1.1 Motivation

The goal of this document is to help users designing intrusion-detection systems (IDSes) that cannot be easily defeated by attackers. Three different sets of issues need to be addressed, namely:

1. Current IDSes are known to generate an extremely large number of false alarms. As a result, the identification of a real alarm lost within the noise of false alarms quickly becomes an infeasible task for security officers. Attackers know how to that take advantage of the so-called *false positive problem*.
2. By design, an IDS must be such that it is not possible to circumvent the detection mechanisms provided by each individual sensor. Otherwise an attacker can defeat the IDS by using well-known techniques to evade detection. Attackers know how to that take advantage of the so-called *false negative problem*
3. The IDS system itself is a target for the attacker. The IDS must not only be able to detect malicious activities against the system it is supposed to protect but also against itself. Attackers know how *to assume control of the IDS system itself*.

In this document, we offer solutions to these three sets of problems. Any design of an IDS should obey these principles and consider the solutions provided as answers to these problems. Indeed, not addressing one of them opens an avenue for malicious users to defeat the overall system.

This document focuses on the main results obtained in the context of the work package 3, dealing with intrusion detection. For the sake of conciseness, we have decided not to include detailed descriptions but rather synthetic summaries. In the course of the text, the interested reader is accordingly referred to other research reports offering in depth explanations.

In Chapter 1 we consider how the design of IDS can take best advantages of the work carried out in other MAFTIA work packages. It is our goal to illustrate these notions by means of scenarios for the demonstrations planned at the end of the MAFTIA project.

## 1.2 Approach

We have tried to keep this deliverable as self-contained as possible. We have made all efforts possible to make it readable for people who have not read the other MAFTIA deliverables. However, throughout the text, we will refer the reader to other MAFTIA documents, where detailed information about specific topics can be found.

Section 2 of this document will provide a reminder of the MAFTIA terminology used throughout the text. This will help the reader understand what we precisely mean by an “intrusion detection system” and how this is related to other existing work. In that section, we also examine the relationship between intrusion detection and intrusion tolerance.

Section 3 will focus on the problem of dealing with large numbers of false alarms. Based on some real data, we will explain the magnitude of the problem, look at existing

solutions, and propose a novel approach based on a refinement of some existing data-mining approached.

Section 4 considers the issue of false positives. Largely based on the previous WP3 deliverables, namely D3 [Alessandri 2001], we provide a formal model that helps designing systems that could maximize detection coverage, with respect to certain fault assumptions, while minimizing the number of individual IDSes, and hence the cost of the overall IDS.

Section 5 looks at the practical problems of implementing correlation rules, as proposed by the previous section. We investigate how testing techniques can be exploited to take advantage of the diversity of existing IDSes.

Section 6 considers the issue of attacks directly targeted against the IDS itself. Existing threats are considered, and various techniques are proposed, taking advantage, where appropriate, of the MAFTIA middleware developed in the context of the work package 2 (WP2).

# Chapter 2 MAFTIA and related work

## 2.1 Introduction

The purpose of this chapter is to define precisely the terminology used throughout the document. The reader familiar with MAFTIA deliverable D2, entitled "Conceptual Model and Architecture" [Powell & Stroud 2001], might consider skipping this chapter because the material it contains is entirely based on various elements taken from D2. Indeed, it was the purpose of that earlier deliverable to provide a common framework and terminology for the various tasks carried out within the various MAFTIA work packages. Therefore, from a modelling and definition standpoint, the notion of an intrusion-tolerant IDS had already been partially addressed in it.

## 2.2 Fault model

In this section, we define a fault model that is appropriate for reasoning about prevention and tolerance mechanisms aimed at ensuring system security. We first revisit the well-known notions of fault, error and failure introduced in [Laprie *et al.* 1995] and then elaborate on potential causes of security failures.

### 2.2.1 Causal chain of impairments

In [Laprie *et al.* 1995], the notions of fault, error and failure were defined in terms of a causal chain:

- *fault*: adjudged or hypothesized cause of an *error*;
- *error*: that part of the system state that may lead to *failure*;
- *failure*: delivered service deviates from implementing the system function;

i.e., an error is the manifestation of a fault on the system state (where “state” is taken in a broad behavioral sense), and a failure is the manifestation of an error on the service delivered to the system user.

From an intrusion detection/tolerance viewpoint, the need for three types of causally-related impairments can be justified by the following remarks:

- It is necessary to distinguish the internal (detectable) impairment (*error*) from the causing impairment (*fault*) because there may be multiple causes (e.g., intentionally malicious faults vs. accidental faults) that could give rise to the same detectable impairment.
- It is necessary to distinguish the internal (detectable) impairment (*error*) from the external impairment (i.e., failure in the service delivered to a user) that intrusion-tolerance techniques aim to prevent. The alternative viewpoint, in which any detectable impairment is deemed to make the system “insecure” in some sense, would make intrusion-*tolerance* an unattainable objective.

Because of the recursive definition of systems in terms of components, a failure at a given level of decomposition may naturally be interpreted as a fault at the next higher level of decomposition, thus leading to a hierarchical causal chain, as illustrated in

Figure 1. The dotted lines represent a "system boundary," at the level of decomposition or abstraction considered.

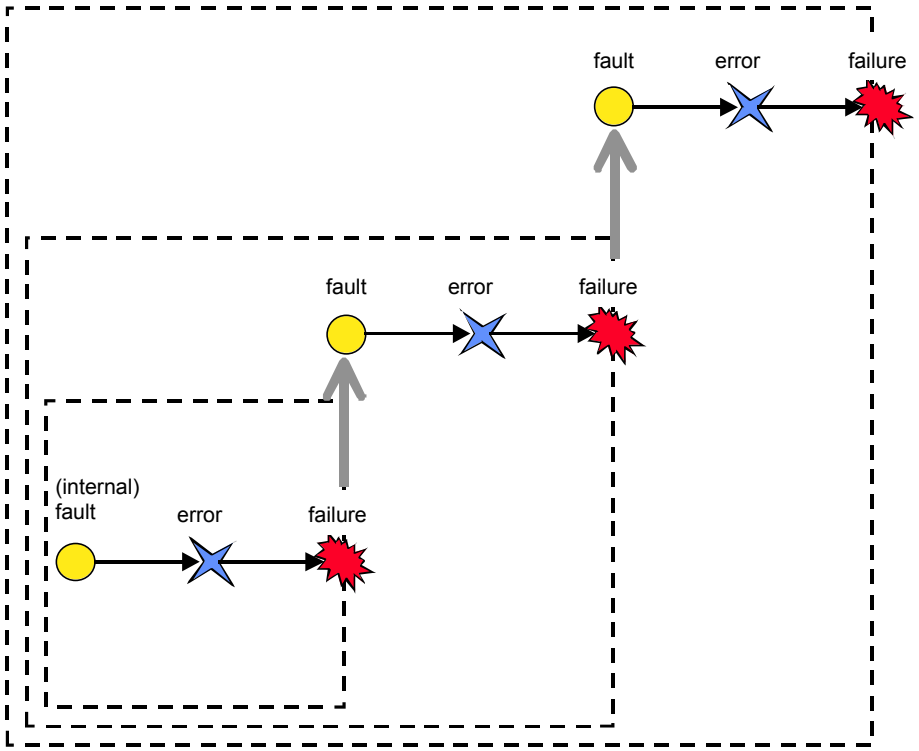



**Figure 1—Hierarchical causal chain of impairments**

The MAFTIA fault model enables such a hierarchical interpretation.

### 2.2.2 Intrusion, attack and vulnerability

An intrusion was defined in [Powell & Stroud 2001] to be a malicious, externally induced, operational fault. Etymologically, the word "intrusion" comes from the Latin *intrudere* (to thrust in) but current usage covers both senses of "illegal penetration" and "unwelcome act." Even a malicious interaction fault perpetrated by an *insider* can thus be classed as an intrusion because the intent is to carry out an operation on some resource that is unwanted by the owner of that resource.

A possible alternative to "intrusion" would be the word "attack." However, it would seem that both terms are necessary, but for different concepts. A system can be attacked (either from the outside or the inside) without any degree of success. In this case, the attack exists, but the protective "shield" around the system or resource targeted by the attack is sufficiently efficacious to *prevent* intrusion. An attack is thus an *intrusion attempt*, and an intrusion results from an attack that has been (at least partially) successful.

In fact, there are two underlying causes of any intrusion (Figure 2):

1. The malicious act or attack that attempts to exploit a weakness in the system,
2. At least one weakness, flaw or *vulnerability*, which is 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.

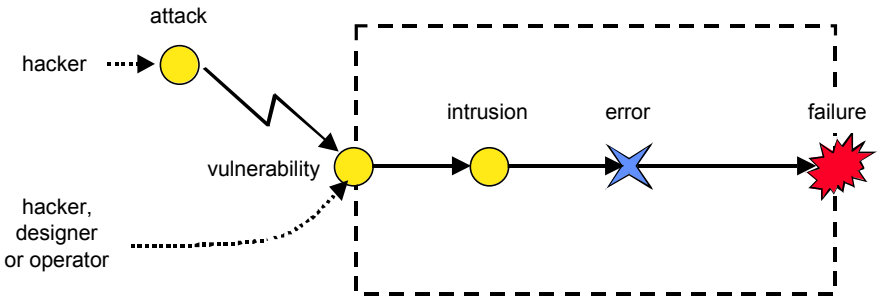



**Figure 2—Intrusion as a composite fault**

## 2.3 Intrusion detection

Whereas the definition of security failure is naturally derived from loss of confidentiality, integrity or availability, there is currently no agreed-upon definition of what might constitute an *error* from the security viewpoint. However, current literature refers to "intrusion detection" which, from the dependability concept viewpoint, might lead one to equate intrusion with "error," rather than "fault."[1] In reality, current literature uses the term "intrusion detection" to cover a *spectrum* of techniques. To paraphrase [Halme & Bauer]:

"Intrusion detection may be accomplished:

- after the fact (as in post-mortem audit analysis)
- in near real-time (supporting SSO[2] intervention or interaction with the intruder, such as network trace-back to point of origin), or
- in real time (in support of automated countermeasures)."

From the dependability concept viewpoint, these three types of intrusion detection can be interpreted, respectively, as:

- off-line fault diagnosis (as part of curative maintenance);
- error detection and on-line fault diagnosis (to an operator-assisted fault treatment facility);
- error detection (as a preliminary to automatic error recovery), or error detection and on-line fault diagnosis (as a preliminary to automatic fault treatment).

Further confusion is introduced by the opposition in [Halme & Bauer] between a "manually reviewed IDS"[3] (called a passive IDS in [Debar *et al.* 1999]) and "Intrusion Countermeasure Equipment (ICE)" or "autonomously acting IDS" (sic) (called an active IDS in [Debar *et al.* 1999]), which clearly go beyond just detection. The notion that an

---

[1] Note, however, that it is also quite common in the literature on tolerance of physical faults to find the term "fault detection" used in one of two ways:

(a) As a clumsy synonym for "error detection" (because detection of an error implies, rather indirectly and perhaps falsely, the "detection" of its cause)

(b) As the designation of a mechanism that seeks out (dormant) faults by running a test procedure to activate them as errors that can be detected by an error detection mechanism.

[2] SSO: System Security Officer.

[3] IDS: Intrusion Detection System.

IDS might include more than just detection, such as also the actions triggered by detection, appears in the Common Intrusion Detection Framework (CIDF) [Porras *et al.*] as well. This framework, which we will re-visit later in this chapter, defines the notion of "response units" that take inputs from other CIDF components to carry out "some kind of action (on their behalf, including) such things as killing processes, resetting connections, altering file permissions, etc."

Here, we prefer to make a distinction between detection *per se* and response, be it manual or automatic. This concurs with the definition given in [NSA 1998], where intrusion detection is defined as: "Pertaining to techniques which attempt to detect intrusion into a computer or network by observation of actions, security logs, or audit data. Detection of break-ins or attempts either manually or via software expert systems that operate on logs or other information available on the network." This is also in line with the charter of the Intrusion Detection Working Group (IDWG) of the Internet Engineering Task Force (IETF) [IDEF], which speaks of "intrusion detection *and* response systems."

However, to conform to the spirit of the NSA definition above, we will avoid using "intrusion detection" in the limited sense of "error detection" but extend it to include some degree of fault diagnosis. To this end, we adopt the following definitions:

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

Our definition of intrusion detection draws attention to the fact that we are particularly interested in detecting errors that may lead to security failure as the ultimate aim of such a system is to provide inputs

- to a system administrator (an SSO) who might carry out further diagnosis and initiate litigation and/or appropriate countermeasures to avert security failures, or
- to an automatic countermeasure mechanism to avert security failures, i.e., to *tolerate* intrusions.

However, the definition also covers a second important aim of intrusion detection, namely that of gathering information about new forms of attack, for which new defenses will need to be devised.

## 2.4 Interpretation of core fault-tolerance concepts

We now consider intrusions in the broader context of intrusion-*tolerance*. We re-examine the notion of fault-tolerance as defined in [Powell & Stroud 2001]. Those concepts distinguish between: (a) *error processing*, aimed at preventing errors from leading to (catastrophic) failure, and (b) *fault treatment*, aimed at preventing the recurrence of errors.

### 2.4.1 Error processing

In the core dependability concepts, error processing covers the set of techniques aimed at removing errors from the computational state, if possible, before the occurrence of a failure. Three error-processing primitives are identified in [Powell & Stroud 2001]:

error detection, damage assessment and error recovery, which we will examine successively in the context of intrusion-tolerance.

#### 2.4.1.1 Error detection

*Error detection* is a necessary preliminary to achieving backward or forward recovery, or compensation by switchover, but is not strictly necessary if compensation is carried out systematically (i.e., fault masking). However, irrespective of the error recovery method employed (if any), error detection is necessary if subsequent fault treatment or curative maintenance actions are to be undertaken.

By definition, error-detection techniques (and indeed, error-processing techniques in general) need to be applied to all errors, irrespectively of the specific faults that caused them.

The intrusion-detection community commonly identifies two categories of error-detection techniques that differ by the type of reference with which observed system behavior is compared [Halme & Bauer].

- Anomaly detection techniques, which compare observed activity against normal usage profiles (in [Debar *et al.* 1999], these are called behavior-based methods).
- Misuse detection techniques, which check for known undesired activity profiles (in [Debar *et al.* 1999], these are called knowledge-based methods).

Here "anomaly" is definitely being used in the traditional sense of "error," whereas "misuse" has an element of fault diagnosis because error patterns related to previously identified intrusions are being searched for.

#### 2.4.1.2 Damage assessment

Damage assessment is an error-processing primitive aimed at evaluating the extent of error propagation before attempting recovery. In traditional fault-tolerance, damage assessment refers, for example, to finding out how many checkpoints to roll back to when doing backward recovery, or how many processes (might) have been affected by an error that has just been detected. In intrusion-tolerance, damage assessment might, for example, be judging which files an intruder has modified so that they can be appropriately restored before someone needs to use them. Vulnerability scanning also comes into play here, as it may be triggered to seek out maliciously implanted vulnerabilities.

#### 2.4.1.3 Error recovery

The core dependability concepts described in [Powell & Stroud 2001] distinguish three forms of error recovery:

- *Backward recovery:* state transformation is carried out by bringing the system back to a previously occupied state, for which a copy (a recovery point) has been saved.
- *Forward recovery:* state transformation is carried out by finding a new state from which the system can operate.
- *Compensation:* state transformation is carried out by exploiting redundancy in the data representing the erroneous state.

## 2.4.2 Fault treatment

In the core dependability concepts, fault treatment covers the set of techniques aimed at preventing faults from being re-activated. Whereas error recovery is aimed at averting imminent failure, fault treatment aims to attack the underlying causes, whether or not error recovery was successful, or even attempted. To take a medical analogy: whereas error processing is concerned with ensuring emergency life support and relieving disease symptoms, fault treatment is concerned with curing the disease, or with providing an autopsy.

Here, we assume that error processing is carried out entirely automatically, whereas fault treatment might be, at least partly, manual, e.g., with the aid of a system security officer or administrator.

Three fault treatment primitives are identified in [Powell & Stroud 2001]: fault diagnosis, fault isolation, and (system) reconfiguration.

### 2.4.2.1 Fault diagnosis

Fault diagnosis is concerned with identifying the type and locations of faults that need to be isolated before carrying out system reconfiguration or initiating corrective maintenance. This includes faults that are judged to be the cause of detected errors, and faults that could cause problems in the future.

In the case of error signals produced by preemptive error-detection mechanisms such as vulnerability scanners and configuration checkers, diagnosis is immediate. However, for error signals from concurrent error-detection mechanisms, it is first necessary to decide whether the underlying cause was an intrusion or an accidental fault.

In the case of intrusions, according to the composite fault model of Section 2.2.2, fault diagnosis can be further decomposed into:

- Intrusion diagnosis, i.e., trying to assess the degree of success of the intruder in terms of system compromise.
- Vulnerability diagnosis, i.e., trying to understand the channels through which the intrusion took place so that corrective maintenance can be carried out.
- Attack diagnosis, i.e., finding out who or which organization is responsible for the attack so that appropriate litigation or retaliation may be initiated.

Note that most currently available intrusion-detection systems do not include any fault 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 important 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 mistyping a password, etc.).

### 2.4.2.2 Fault isolation

In traditional fault tolerance, fault isolation is needed, say, to prevent a faulty transmitter from babbling over a shared bus or to prevent a faulty sensor from continuing to add faulty readings to a pool of redundant measurements. That is, we want to make sure that the source of the detected error(s) is prevented from producing further error(s).

In terms of intrusions, this might involve:

- Blocking traffic from an intrusion-containment domain that has been diagnosed as corrupt, by, for example, changing the settings of firewalls or routers.
- Removing a corrupted file from the system.

Or, with reference to the root vulnerability/attack causes:

- Uninstalling software versions with newly-found vulnerabilities.
- Arresting the attacker.

#### 2.4.2.3 System reconfiguration

The occurrence of faults and the consistent isolation of faulty components naturally lead to a decrease in the number of available fault-free resources, so, in traditional fault-tolerant systems, reconfiguration is sometimes envisaged to effectively re-deploy those resources. As already stated in the core dependability concepts, this may mean abandoning some tasks or services (thus resulting in degraded operation) or re-distributing them among the remaining resources.

Reconfiguration of the system allows a possibly degraded service to be delivered while corrective maintenance is being carried out on faulty resources. After corrective maintenance, further reconfiguration allows repaired or replacement resources to be re-deployed.

In terms of intrusions, possible reconfiguration actions include:

- Changing a voting threshold (say from 3-out-of-5 voting to 2-out-of-3 voting) after two corrupt servers have been isolated, so that a further intrusion can be masked.
- Deployment of countermeasures including more probes and traps (honey-pots) to gather further information about the intruder, and so assist in attack diagnosis.

Actions pertaining to corrective maintenance might be:

- Removing vulnerabilities believed to have contributed to the intrusion:
  - Software revision and upgrade.
  - Deployment of security patches.
- Attacker rehabilitation.

## 2.5 Integrated intrusion detection/tolerance framework

In this section we examine the relationship between intrusion detection and intrusion tolerance. In particular, we will investigate:

- How intrusion detection helps when building an intrusion-tolerant system.
- How an intrusion-detection system can itself be made dependable.

We show how the ideas derived from the core dependability concepts and from intrusion detection might be fitted together in a single integrated framework. It is described what an IDS does towards intrusion tolerance, how the system fits into the overall picture, and how such a system may itself be made intrusion-tolerant.

Our integrated intrusion detection/tolerance framework is illustrated in Figure 3.

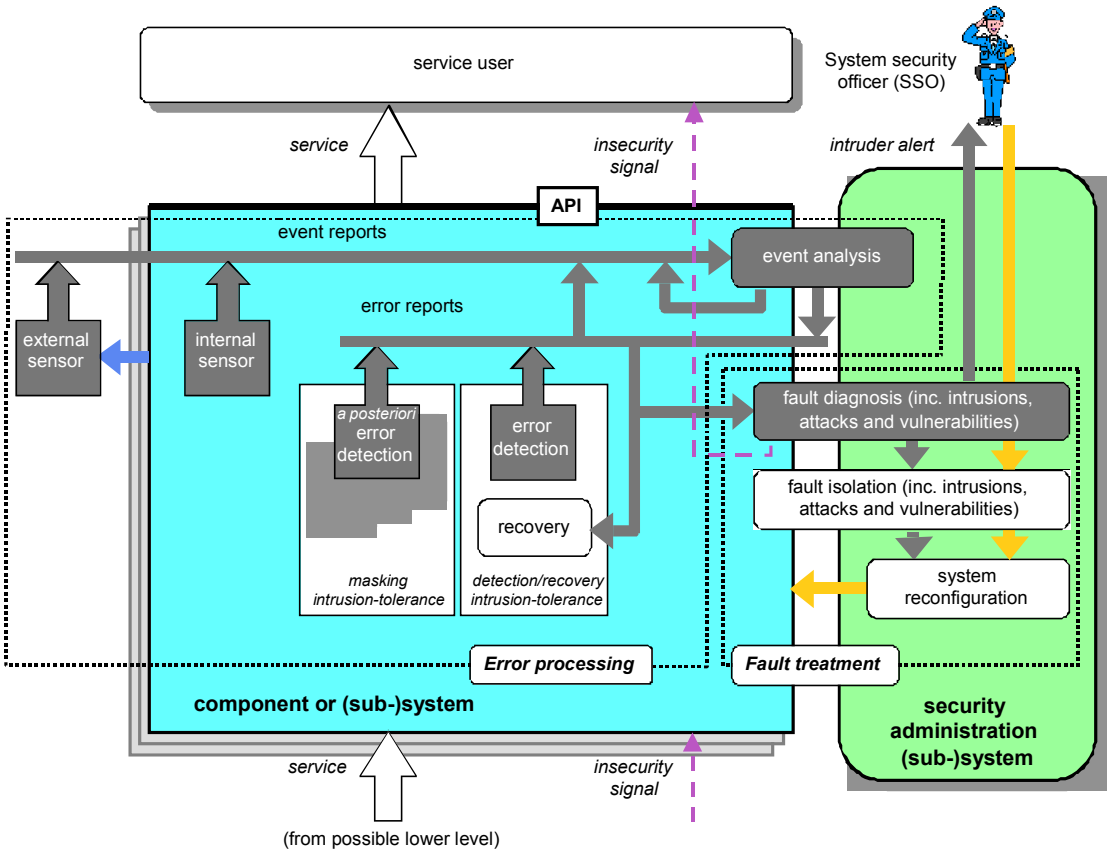



**Figure 3—Integrated intrusion-tolerance framework**

The central part of the figure shows a generic MAFTIA component or (sub-)system. There may be many such components within a MAFTIA system, implementing either end-user application functionality or application support services. An administration (sub-)system manages all such components within a single management domain. Here, we consider only the security aspects of system administration within a single management domain. The security-administration component in this diagram spans over all the layers of the system and, in particular, over those comprising the application. The security-administration component is not specific to an individual application but may provide its service to several different applications within the management domain considered.

Components may be layered. The figure shows a component offering some service over an application-programming interface (API) to some higher-level component, using the service(s) offered by possible lower-level components. In this case, taking inspiration from the "idealized fault-tolerant component" of [Anderson & Lee 1981], these top and bottom interfaces include "insecurity signals" aimed at informing the service user that the service has been (or might have been) compromised. However, such insecurity signals may not be provided by all generic components, at least not autonomously, because a decision to raise such an insecurity signal may involve some system-wide analysis (by the security administration subsystem).

According to the interpretation of the core fault-tolerance concepts in Section 2.4, we further describe Figure 3 in terms of error processing and fault treatment.

### 2.5.1 Error processing

We distinguish two basic generic component types:

- Intrusion-intolerant components
- Intrusion-tolerant components

Both component types are potential sources of error-detection information (in the form of event and error reports). However, intrusion-tolerant components are also capable of acting autonomously to implement error recovery.

Error and event reports can be analyzed in a given context to confirm or deny suspected errors (cf. Section 6.2.1.3). Confirmed errors may or may not trigger automatic recovery. In either case, they are reported to the fault-treatment facilities, which may carry out further analysis toward understanding the root causes of detected errors (fault diagnosis), and then act thereupon (fault isolation and system reconfiguration).

### 2.5.2 Fault treatment

The fault-treatment facilities include the means for diagnosing and isolating faults (including intrusions, attacks and vulnerabilities), and for automatic or manual system reconfiguration (cf. Section 2.4.2). Whereas it seems feasible to implement some degree of fault diagnosis and isolation within the component considered (this would be necessary if the component were to be capable of autonomously raising an insecurity signal), it is expected that it will often be necessary to take into account a more system-wide view. Moreover, such a system-wide view seems essential to carry out meaningful system reconfiguration. For these reasons, Figure 3 shows the fault diagnosis and isolation mechanisms distributed across the generic component(s) and the security administration system, whereas the system reconfiguration mechanisms are internal to the latter, which may possibly be distributed.

From the viewpoint of intrusion detection, the IDS (as defined in Section 2.3, i.e., excluding the so-called response mechanisms) in this integrated framework consists of the set of external and internal sensors, the error-detection mechanisms of any intrusion-tolerant components, and the event analysis and fault diagnosis mechanisms that signal intruder reports to a system security officer. These are shown in dark gray in Figure 3.

# Chapter 3 Handling large numbers of false alarms

## 3.1 Introduction

Practitioners [Broderick 1998], [Manganaris *et al.* 2000] as well as researchers [Axelsson 2000], [Bloedorn *et al.* 2000], [Clifton and Gengo 2000], [Julisch 2001b] have observed that IDSes can easily trigger thousands of alarms per day, up to 99% of which are false positives. This flood of mostly false alarms makes it very difficult to identify the hidden true positives. For example, the manual investigation of alarms has been found to be labor-intensive and error-prone [Broderick 1998], [Dain and Cunningham 2002], [Manganaris *et al.* 2000], [Barbara and Jajodia 2002]. Tools to automate alarm investigation are being developed [Dain and Cunningham 2002], [Debar and Wespi 2001], [Valdes and Skinner 2001] but there is currently no silver-bullet solution to this problem.

This continuous stream of false alarms constitutes a real advantage for attackers interested in defeating the IDS. Indeed, identifying a true positive within a large number of false positives is equivalent to finding a needle in a haystack. Attackers can make the task of the security officers even more complicated by generating nonmalicious traffic which is known to generate false positive. Aside from trying to hide their actions with the help of the surrounding noise, attackers can also take advantage of it to attack the IDS components themselves. Here again, the attack against the IDS will most likely not be detected if run in a stealthy way in a very noisy environment.

From the above, it is clear that an efficient solution must be found to get rid of the existing huge number of false alarms. This is a prerequisite to the design of any intrusion-tolerant system.

This chapter presents a novel semi-automatic approach for handling intrusion-detection alarms efficiently. Central to this approach is the notion of alarm root causes. Intuitively, the root cause of an alarm is the reason for which it occurs. We have made the key observation that a small number of root causes generally accounts for over 90% of all alarms. Moreover, these root causes are mostly configuration problems and do not disappear unless someone fixes them. Note, however, that root causes may be of benign as well of malicious nature.

We have investigated techniques to efficiently handle large groups of redundant alarms. The outcome of this research was a semi-automatic process that consists of two steps:

- Step one, which is conventionally called root-cause analysis, identifies root causes that account for large numbers of alarms.
- Step two removes these root causes and thereby dramatically reduces the future alarm load.

Indeed, the experiments of Section 3.6 show how the one-time effort of identifying and removing root causes pays off by reducing the future alarm load by up to 89%. This is significant as it enables the intrusion-detection analyst to concentrate on the remaining 11% of alarms. As a general rule, root cause identification and removal should be done roughly once a month in order to keep up with changes in the computing environment.

Our work focuses on the first of the above two steps, i.e. on the identification of root causes. To support this step, we have developed a novel technique called alarm

clustering. The motivation for alarm clustering stems from the observation that the alarms of a given root cause are generally *"similar"* (Section 3.4.3 will formally define our notion of similarity). Alarm clustering reverses this implication and groups similar alarms together, assuming that these alarms also share the same root cause. For each alarm cluster, a single so-called generalized alarm is derived. Intuitively, a generalized alarm is a pattern that an alarm must match in order to belong to the corresponding cluster. We show that knowledge of generalized alarms vastly simplifies root-cause analysis.

The novel contribution of this approach is fourfold:

1. Root cause identification and removal are introduced as a safe way to manage intrusion-detection alarms efficiently.
2. We present and validate a novel clustering algorithm that groups similar alarms together and summarizes them by a single generalized alarm.
3. We show that it is usually quite straightforward to derive the root cause of a given generalized alarm.
4. We show that removing root causes can dramatically reduce the future alarm load, thus enabling a much more thorough analysis of the remaining alarms.

The remainder of this chapter is organized as follows: Section 3.1.1 illustrates root cause identification and removal by means of examples. Section 3.2 explains why IDSes trigger so many false positives. Moreover, it surveys related work that also addresses this problem. Section **Error! Reference source not found.** provides some background on root-cause analysis in general, and on how it applies to intrusion detection. Sections 3.4 to 3.6 derive and evaluate the clustering algorithm that we use to support root-cause analysis. Specifically, Section 3.4 introduces our notion of similarity, defines alarm clusters, and proves the general alarm clustering problem to be NP-complete. Section 3.5 presents a novel algorithm that offers an approximation solution to the alarm-clustering problem. This makes alarm clustering a practical technique. Section 3.6 presents our experience with alarm clustering.

### 3.1.1 Some illustrative examples

By way of illustration, here is a short list of root causes that we have observed in our work (related examples by different authors can be found in [Internet Security Systems 2001] and [Paxson 1999]):

1. A broken TCP/IP stack that generates fragmented traffic and thereby triggers “Fragmented IP” alarms.
2. A misconfigured secondary DNS server, which does half-hourly DNS zone transfers from its primary DNS server. The resulting “DNS Zone Transfer” alarms are no surprise.
3. A Real Audio server whose traffic remotely resembles TCP hijacking attacks. This caused our commercial IDS to trigger countless “TCP Hijacking” alarms.
4. A firewall that has Network Address Translation (NAT) enabled funnels the traffic of many users and thereby occasionally seems to perform host scans. Indeed, a NAT-enabled firewall acts as proxy for many users. When these users simultaneously request external services, the firewall will proxy these requests, and the resulting SYN packets resemble SYN host sweeps.

5. A load balancing router that dispatches Web client requests to the least busy server. The resulting traffic patterns resemble host scans that trigger alarms on most IDSes.
6. A network management tool scanning the network or querying sensitive MIB variables triggers alarms on most IDSes.
7. An attacker trying a brute-force password guessing attack against our ftp server.
8. CodeRed scanning for vulnerable servers.

It is instructive to examine the first example of the broken TCP/IP stack in more detail. Obviously, all the resulting “Fragmented IP” alarms have the same source address. Now, let us suppose that the broken TCP/IP stack belongs to a Web server. Then, also the source ports are identical (namely 80). Furthermore, assuming that the Web server is used primarily on workdays, we will observe that the bulk of alarms occur on workdays. In summary, this example shows that all alarms of a given root cause are “similar”. Our alarm clustering algorithm groups these alarms together and summarizes them by a single generalized alarm such as "Your web server triggers many 'Fragmented IP' alarms on workdays!"

Clearly, a generalized alarm like the above facilitates the identification of root causes, but human expertise is still needed. Therefore, alarm clustering only supports the identification of root causes, but does not completely automate it. Moreover, recall that root-cause analysis is only the first in a two-step process. The second step is to act upon the root causes identified. Ideally, one would always remove root causes (e.g. by fixing the broken TCP/IP stack of the above example). Unfortunately, some root causes are not under our control or are expensive to remove. Then, custom-made filtering rules (which automatically discard alarms) or correlation rules (which intelligently group and summarize alarms) can be an alternative. Clearly, this second step requires care as well as human judgment.

## 3.2 Background and related work

As explained above, this research was motivated by the fact that today's IDSes tend to trigger an abundance of mostly false alarms. It is therefore natural to ask for the reasons of this alarm flood. Section 3.2.1 addresses this question, and Section 3.2.2 surveys related work that also investigates solutions to the false alarm problem.

### 3.2.1 Origins of the alarm flood

This section only considers the false alarm problem for knowledge-based systems. For these systems, the abundance of alarms in general, and of false positives in particular, can be attributed to three main factors:

*Underspecified signatures* check conditions that are necessary, but not sufficient for attacks. As a consequence, they also trigger on benign events, which causes false positives. For example, instead of complex regular expressions that can reliably detect many attacks, it is not uncommon to use simple string-matching signatures. There are four reasons for this practice: First, harsh real-time requirements generally preclude the use of precise signatures, which are also more time-consuming to match against the audit data [Ilung 1993], [Ptacek and Newsham 1998]. Second, because of their higher generality, it is attractive to use underspecified signatures that also cover variations of attack [Clifton and Gengo 2000], [Debar and Wespi 2001]. Third, audit sources

frequently lack information useful for misuse detection [Price 1997], [Ptacek and Newsham 1998]. That can make underspecified (or "speculative," if you will) signatures inevitable. Fourth, writing intrusion detection signatures is inherently difficult [Kumar 1995], [Lee and Stolfo 2000], [Mounji 1997], [Ning *et al.* 2001], which favors the creation of buggy or underspecified signatures.

*Intent-guessing signatures* trigger on events that might or might not be attacks. For example, signatures that trigger on failed user logins, DNS zone transfers, overlapping IP fragments, or set URGENT bits are intent-guessing because they assume that these activities are malicious. It has been shown that frequently this assumption is wrong [Bellovin 1993], [Paxson 1999]. (Note that intent-guessing signatures are not underspecified as they reliably detect the events they claim to detect. However, these events are not necessarily attacks.)

*Lack of abstraction*. Today's IDSes tend to trigger multiple low-level alarms to report a single conceptual-level phenomenon. For example, a single run of the nmap scanning tool [Fyodor] triggers hundreds of alarms, namely, one alarm for each probe. Similarly, a network with a small maximum transfer unit (MTU) [Tanenbaum 1996] systematically fragments IP packets. Nevertheless, most IDSes trigger a separate alarm for each fragmented packet. Clearly, this lack of abstraction aggravates the alarm flood.

Axelsson has observed that the use of underspecified or intent-guessing signatures easily leads to an over-proportionally high number of false positives [Axelsson 2000]. Just observe that these signatures are prone to triggering on nonintrusive events. Unfortunately, nonintrusive events are so much more frequent than intrusive events that even their occasional misclassification can easily lead to an unacceptably high false-alarm rate. Axelsson argues that a 90% probability of an alarm being a false one is completely unacceptable because it schools the human operator to ignore the alarm altogether [Axelsson 2000].

### 3.2.2 Data mining

The idea of using data mining to support alarm investigation is not new. [Manganaris *et al.* 2000] mine association rules over alarm bursts. Subsequently, alarms that are consistent with these association rules are deemed “normal” and get discarded. The risk of losing relevant alarms is not considered in this work, whereas we follow the root-cause analysis paradigm to bound this risk. [Clifton and Gengo 2000] use episode mining to guide the construction of custom-made filtering rules. Our work pursues a similar idea, but mines alarm clusters rather than episode rules. Moreover, we offer a more detailed experimental validation. Finally, the work by Dain [Dain and Cunningham 2002] already mentioned is also relevant in this context as it uses data mining techniques to learn correlation rules from hand-labeled training examples.

Other related work comes from Barbara *et al.* [Barbara *et al*. 2001], who use incremental data mining techniques to detect anomalous network traffic patterns in real time. In [Lee and Stolfo 2000] data mining is used for feature construction and training of classifiers that detect intrusions. The goal of these research projects is to construct IDSes more systematically. Our work, by contrast, aspires to use existing IDSes more efficiently. There are many other research projects that also applied data mining to intrusion detection. An overview of these projects and a general treatment of data mining in computer security can be found in a recent book edited by Barbara and

Jajodia [Barbara and Jajodia 2002]. Data mining for fraud detection is investigated in [Fawcett and Provost 1997] and in [Chan and Stolfo 1998].

In the world of telecommunication networks, [Klemettinen 1999] uses association rules and episode rules to support the development of alarm correlation systems. Hellerstein and Ma pursue the same goal by means of visualization, periodicity analysis, and m-patterns (a variant of association rules requiring mutual implication) [Hellerstein and Ma 2000]. However, their research does not capture the notions of alarm similarity and root causes. Furthermore, association rules and episodes have the drawback of generating many uninteresting and redundant alarm patterns [Klemettinen 1999]. Our alarm clustering algorithm does not have these drawbacks.

## 3.3 Root-cause analysis

Root-cause analysis has historically been used to identify the most basic factors that contributed to an incident [Livingston *et al.* 2001]. For example, after a reactor accident, the Energy Department would commonly use root-cause analysis to pinpoint why the accident had happened, so that in the future similar accidents could be prevented. Because prevention of similar accidents is central to root-cause analysis, it has been required that root causes must be factors that one has control over. This motivates the following definition by [Paradies and Busch 1988]:

- *Root cause:* The most basic cause that can reasonably be identified and that management has control to fix.
- *Root-cause analysis*: The task of identifying root causes.

The three key words in the definition of root causes are basic, reasonably, and fix: Root causes must be so basic (or specific) that one can fix them. On the other hand, given that fixing them is the entire point, it is not reasonable to split root causes further into more basic causes. Consequently, these more basic causes are not root causes, and root causes lie at the "highest" level where fixing is possible. Finally, note that a single incident can have multiple root causes.

The above definitions clearly capture our intuitive understanding of the terms root cause and root-cause analysis. Also, it is clear that, as defined in [Powell & Stroud 2001], root causes are nothing but faults and root-cause analysis is equivalent to fault treatment. However, it is very important to stress the fact that we are talking about faults that are the cause of errors in two completely different systems:

- If an intrusion has really taken place on the system S that we are trying to protect, S is in an error state in the sense that this intrusion could lead to a security failure. As explained before, the root causes of this situation are the existence of an attack and of a vulnerability. We refer the interested reader to the discussion in [Powell & Stroud 2001] about the corresponding fault treatment.
- If an IDS has erroneously raised an alarm, S is not in an error state because no security failure will happen. However, we can argue that an error affects the service delivered by the IDS because the latterdeviates from implementing the system function, which is the detection of real attacks. Thus, in this case, the root causes of this situation are to be found in some external conditions but also might be found in the design of the IDS itself. As we will see, fault treatment can be applied on both by modifying the environment or (and) by reconfiguring the IDS.

Practically speaking however, there is no way to determine the situation we are face for a given alarm or set of alarms before having completed the fault diagnosis step. It is only when the causes of errors in terms of localization and nature have been identified that we will be able to figure it out. In the following, we will provide a method to help deal with this fault diagnosis phase and also with the fault treatment in the case of false alarms. We will keep using the terms 'root causes' instead of faults in the following as a way to conveniently express the fact that we are indeed dealing with two separate sets of faults, in a uniform way.

## 3.4 A Framework for similarity and clustering mining

In this and the next section, we consider how root-cause analysis can be carried out in practice. More precisely, we consider the following problem:

> *Given an alarm log and a decent background knowledge of the site from which it was taken; find the root causes that caused the constituent alarms to be triggered*.

Note that the sheer size of alarm logs renders any purely manual solution impractical. Therefore, we have developed a semi-automatic procedure consisting of two steps: The first step finds clusters of similar alarms and describes them in a succinct and intuitive format. This step is fully automatic, and the underlying algorithm is described in this and the next section. The second step is manual and infers the root causes from the clusters of the first step. This second step is discussed in Section 3.6

In the remainder of this section, we present our framework for similarity and clustering. More precisely, we define our notion of alarm similarity and formalize the properties that alarm clusters must have.

### 3.4.1 Introduction and background

This subsection introduces clustering in general, and discusses the problems that are intrinsic to alarm clustering in particular. In general, clustering seeks to group objects so that the objects in a given group/cluster are similar, whereas the objects of different groups/clusters are dissimilar [Han and Kamber 2000], [Han and Kamber 2000], [Jain and Dubes 1988]. Obviously, the notion of similarity is key to this definition. In practice, similarity is easy to define for numerical attributes, but categorical, time, or string attributes pose serious problems [Fasulo 1999], [Ganti *et al*. 1999], [Guha *et al.* 2000], [Han and Kamber 2000]. Unfortunately, alarm messages can contain all of these attribute types:

- *Categorical attributes:* The domain of categorical attributes is discrete and unordered [Han and Kamber 2000]. Examples of categorical attributes include IP addresses, port numbers, and header flags.
- *Numerical attributes:* Examples of numerical attributes include counters (e.g. for the number of SYN packets in a SYN flooding alarm), and size fields (e.g. for the packet size in “Large ICMP Traffic” alarms [CERT 1996-26]).
- *Time attributes:* All alarms must be time-stamped. Note that time should not be regarded as a numerical attribute. Indeed, doing so would mean to ignore its unique semantics (including the notions of periodicity, workdays versus weekends, etc.).

- *String attributes:* String attributes assume arbitrary and unforeseeable text values. To begin with a negative example, attributes such as "Attack Name" or "Recommended Action" are not string attributes because their domains are small and fixed. However, NetRanger's context field [Cisco] is a typical string attribute. The context field is optional, but, when present, it stores the supposedly malicious network packet. For example, the context field frequently contains supposedly malicious URLs or command sequences that seem to constitute an FTP or Telnet exploit. Evidently, the difficulty with string attributes is to extract the information that they carry.

In fact, the mere presence of categorical attributes makes alarm clustering a difficult problem [Ben-Dor *et al.* 1999], [Ganti *et al*. 1999], [Gibson *et al.* 2000], [Guha *et al.* 2000]. The presence of time and string attributes adds further complexity. This section introduces a similarity concept that embraces all of the above attribute types. Furthermore, the clustering problem is restated in the light of this similarity concept.

In a nutshell, to define the similarity between alarms, we use taxonomies on the alarm attributes. Examples of alarm attributes include the source and destination IP address, and taxonomies are generalization hierarchies on these attributes. For example, a taxonomy might state that IP address is a Web server, is a machine in the DMZ etc. Intuitively, two alarms are all the more similar the closer their attributes are related in terms of these taxonomies. We will formalize this notion and define alarm clusters as alarm sets that maximize intra-cluster alarm similarity, while having a user-defined minimum size. In the output, we require that each alarm cluster be summarized by a single "generalized" alarm. We consider this an important feature of our approach because it is not practical to communicate alarm clusters by means of their constituent alarms.

### 3.4.2 Notation

At present, different IDSes report their alarms in different formats. In anticipation of a unifying standard [Erlinger and Staniford-Chen] a similar, yet simplified alarm format will be used. Specifically, alarms are modeled as tuples over a multi-dimensional space. The dimensions are called *alarm attributes* or *attributes* for short. Examples of alarm attributes include the source and destination IP address, the source and destination port, the alarm type (which encodes the observed attack), and the time-stamp (which also includes the date).

Formally, alarms are defined as tuples over the Cartesian product $\mathsf{X}_{1\le i\le n}\mathrm{dom}(A_i)$, where $\{A_1...,A_n\}$ is the set of attributes and $\mathrm{dom}(A_i)$ is the *domain* (i.e. the range of possible values) of attribute $A_i$. Furthermore, for an alarm $a$ and an attribute $A_i$, the projection $a[A_i]$ is defined in the usual way, namely, as the $A_i$ value of alarm $a$. Next, an *alarm log* is modeled as a set of alarms. This model is correct if the alarms of alarm logs are pair-wise distinct – an assumption that we make to keep our notation simple. (Note that unique alarm-IDs can be used to make all alarms pair-wise distinct.)

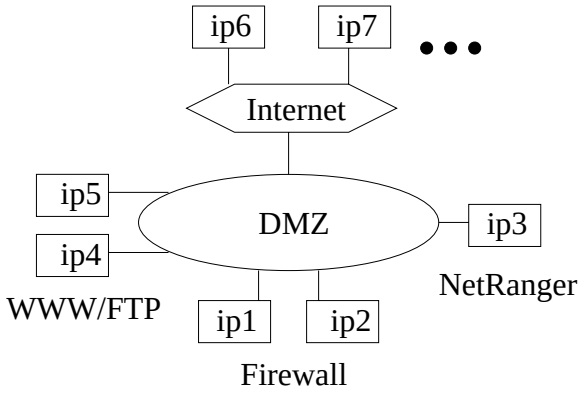
Let $A_i$ be an alarm attribute. A tree $\mathsf{T}_i$ on the elements of $\mathrm{dom}(A_i)$ is called a *taxonomy* (or a *generalization hierarchy*). For two elements y; $\hat{y} \in \mathrm{dom}(A_i)$, we call $\hat{y}$ a *parent* of $y$, and y a *child* of $\hat{y}$ if there is an edge $\hat{y} \to y$ in $\mathsf{T}_i$. Furthermore, we call $\hat{y}$ a *generalization* of $y$ if the taxonomy $\mathsf{T}_i$ contains a path from $\hat{y}$ to y (in symbols: $y \trianglelefteq \hat{y}$). The length of this path is called the *distance* $\delta(y, \hat{y})$ between $y$ and $\hat{y}$. Note that $\delta(y, \hat{y})$ is

undefined if $y \trianglelefteq \hat{y}$ is not satisfied. Finally, $y \trianglelefteq \hat{y}$ is trivially satisfied for $y = \hat{y}$, and $\delta(y, \hat{y})$ equals 0 in this case.

By way of illustration, Figure 4 shows a network topology and the taxonomies one might want to use for IP addresses and port numbers in this environment. Note that the domain of IP addresses is the union of "elementary" IP addresses (i.e. the set $\{p.q.r.s | p,q,r,s \in \{0,...,255\}\}$) and "generalized" IP addresses (i.e. the set {*FIREWALL*, *WWW/FTP*, *DMZ*, *EXTERN*, *ANY-IP*}). Analogously, the domain of port numbers is {1..., 65535, *PRIV*, *NON-PRIV*, *ANY-PORT*}. Next, according to Figure 4c, the IP address *ip1* is a firewall, is a DMZ machine, is any IP address. More succinctly, this relationship can be expressed as $ip1 \trianglelefteq FIREWALL \trianglelefteq DMZ \trianglelefteq ANY\text{-}IP$. Furthermore, we have
$\delta(ip1,ANY\text{-}IP) = 1+\delta(\text{FIREWALL},ANY\text{-}IP) = 1+1+\delta(\text{DMZ},ANY\text{-}IP) = 1+1+1+\delta(\text{ANY-IP},\text{ANY- IP}) = 1+1+1+0 = 3$. Finally, $\delta(ip1,ip2)$ is not defined because $ip1 \trianglelefteq ip2$ is false.

Next, we extend our notation from attributes to alarms. To this end, let $a,\hat{a} \in \bigtimes_{1 \le i \le n} \text{dom}(A_i)$ denote two alarms. Alarm â is called a *generalization* of alarm *a* if and only if $a[A_i] \trianglelefteq \hat{a}[A_i]$ holds for all attributes $A_i$. In this case, we write $a \trianglelefteq \hat{a}$. Furthermore, if $a \trianglelefteq \hat{a}$ holds, then the *distance* $\delta(a,\hat{a})$ between the â alarms *a* and â is defined as $\delta(a,\hat{a}) := \sum_{i=1}^{n} \delta(a[A_i],\hat{a}[A_i])$. If $a \trianglelefteq \hat{a}$ is not satisfied, then $\delta(a,\hat{a})$ is undefined. Finally, in the case of $a \trianglelefteq \hat{a}$, we say that *a* is more *specific* than â, and â more *abstract* than *a*.

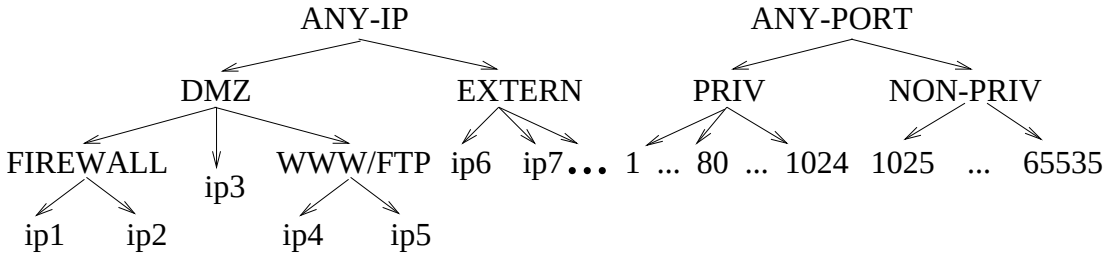
As a convention, we use the symbols $A_1,...,A_n$ to stand for alarm attributes. Furthermore, the symbols $T_1,...,T_n$ are reserved for taxonomies on the respective attributes. Finally, the symbol $L$ will be used to denote an alarm log.




a) Network

| Target-IP | Target-Port |
|---|---|
| ip1 | 80 |
| ip2 | 1234 |
| ip2 | 23 |
| ip4 | 8080 |
| ip4 | 80 |
| ip4 | 21 |
| ip5 | 80 |

b) Sample alarm log




c) Taxonomies for IP addresses and port numbers

**Figure 4—Network, alarm log and taxonomies of our running example**

### 3.4.3 Similarity and Alarm Clusters

We begin by defining similarity. To this end, let $S \subseteq L$ denote a set of alarms. The *cover* of $S$ is the most specific alarm $c \in X_{1 \le i \le n} \mathrm{dom}(A_i)$ to which all alarms $a$ in $S$ can be generalized. Thus, the cover $c$ satisfies $\forall a \in S: a \trianglelefteq c$, and there is no more specific alarm $c'$ ($c' \trianglelefteq c$) that would also have this property. We denote the cover of $S$ by cover($S$). For example, in Figure 4, we have cover({(*ip1*,80),(*ip4*,21)})=(*DMZ*,*PRIV*). Finally, the *dissipation* of $S$ is defined as

$$\Delta(S) := {}^{1}/_{|S|} \sum_{a \in S} \delta(a, \mathrm{cover}(S)). \qquad \text{(Equation 1)}$$

Let us verify that $\Delta(\{(ip1,80),(ip4,21)\}) = {}^{1}/_{2} \times (3+3) = 3$. Intuitively, the dissipation measures the average distance between the alarms of $S$ and their cover. The important thing here is that the alarms in $S$ are all the more similar, the smaller the value of $\Delta(S)$. Therefore, we will strive to *minimize* dissipation in order to *maximize* intra-cluster alarm similarity.

Next, we define the alarm-clustering problem. To this end, let $L$ be an alarm log, $min\_size \in N$ an integer, and $T_i$, $i$=1...,$n$, a taxonomy for each attribute $A_i$. We define:

**Definition 1: Alarm-Clustering Problem:** Let $(L, min\_size, T_1, ..., T_n)$ be an $(n+2)$-tuple with symbols as defined above. The alarm clustering problem is to find a set $C \subseteq L$ that minimizes the dissipation $\Delta(C)$, subject to the constraint that $|C| \geq min\_size$ holds. We call C an alarm cluster or cluster for short.

In other words, among all sets $C \subseteq L$ that satisfy $|C| \geq min\_size$, a set with minimum dissipation has to be found. If there are multiple such sets, then any one of them can be picked. Note that once the cluster $C$ has been found, the remaining alarms in $L \backslash C$ can be mined for additional clusters. One might consider using a different *min_size* value for $L \backslash C$, an option that is very useful in practice.

Imposing a minimum size on alarm clusters has two advantages. First, it decreases the risk of clustering small sets of unrelated but coincidentally similar alarms. Second, large clusters are of particular interest because identifying and resolving their root causes has a high payoff. Finally, the decision to maximize similarity as soon as the minimum size has been exceeded minimizes the risk of including unrelated alarms in a cluster.

Clearly, stealthy attacks that trigger fewer than *min_size* alarms do not yield any clusters, but that is not the point. Indeed, let us recall the practical goal we are pursuing, namely, to identify predominant root causes that account for *large* numbers of alarms. By removing these root causes, we can significantly reduce the number of newly generated alarms. Exactly this reduction is our goal because screening the reduced alarm stream for attacks is much more efficient. Note, however, that, even though important, this screening step is beyond the scope of this article.

Given the need for a practical alarm-clustering algorithm, the following result is relevant:

**Theorem 1: The Alarm-Clustering Problem:** $(L, min\_size, T_1..., T_n)$ is NP-complete.

The proof is obtained by reducing the CLIQUE problem [Papadimitriou 1994] to the alarm-clustering problem. For details, please refer to [Julisch 2002]

In Section 3.5, we will describe an approximation algorithm for the alarm-clustering problem. For now, let us assume that we are able to discover alarm clusters. Then, the question arises how alarm clusters are best presented to the user. As will be shown in Section 3.6, alarm clusters can easily comprise thousands of alarms. Therefore, it is not viable to represent clusters by means of their constituent alarms. Indeed, doing so would mean to overwhelm the user with a vast amount of information that is difficult to make sense of. To solve this problem, we represent clusters by their covers. Note that covers correspond to what we have informally called "generalized alarms."

Representing clusters by their covers works well as long as the covers are reasonably specific. Indeed, abstract covers such as (*DMZ*, *ANY-PORT*) (cf. Figure 4) reveal very little about the alarms the cover was derived from. Specific covers such as (*FIREWALL*, 80) are preferable (even if they do not explicitly reveal the root cause either). Fortunately, by clustering alarms of maximum similarity, we favor clusters that have specific covers. This relationship, which is established by equation (2), is another argument for clustering alarms of maximum similarity (and minimum dissipation).

In order to obtain generalized alarms that are meaningful and indicative of their root causes, we clearly need to consider more attribute types than just the ones in the above examples. In particular, string and time attributes can contain valuable information, and the following subsection shows how to include these attribute types in our framework.

### 3.4.4 Taxonomies for time and string attributes

The time and string attributes of intrusion-detection alarms can be tied into the above notion of similarity. For the sake of brevity, this section will rely on examples, but the generalizations are straightforward.

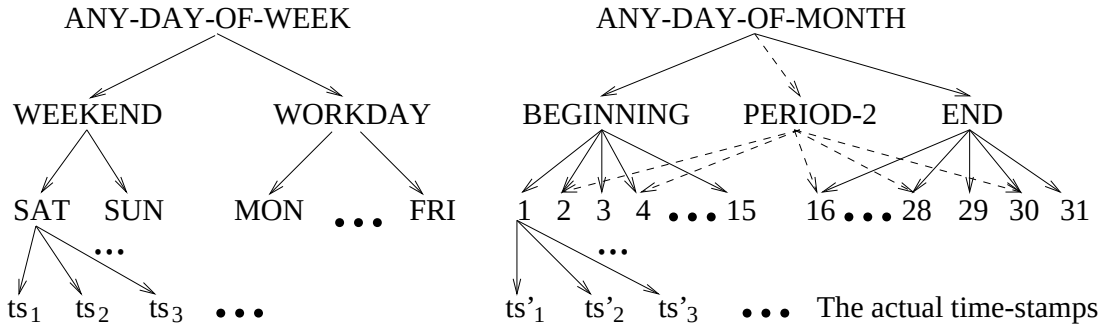



**Figure 5—Sample taxonomies for time attributes**

Let us consider time attributes first. Typically, one wishes to capture temporal information such as the distinction between weekends and workdays, between business hours and off-hours or between the beginning of the month and the end of the month. To make the clustering algorithm aware of concepts like these, one can use taxonomies such as those in Figure 5 (please ignore the dashed arrows for the moment). For example, the left-hand taxonomy shows that the time stamp $ts_1$ can be generalized to the concepts *SAT*, *WEEKEND*, and ultimately, *ANY-DAY-OF-WEEK*.

A limitation of the current framework is that taxonomies have to be trees, whereas directed acyclic graphs (DAGs) are more flexible. Indeed, it might be desirable to use the two taxonomies in Figure 5 simultaneously. However, combining the taxonomies into a single one is not possible because each time stamp would have to have two parents – one that corresponds to its day of the week and one that corresponds to its day of the month. This violates the tree property of taxonomies.

To reach a solution, we replicate the time attribute and assign each taxonomy tree to a separate replica. In this way, one replica plays the role "Day of the Week," while the other plays the role "Day of the Month." Furthermore, each replica is generalized according to its own taxonomy. Similarly, suppose that we want to include the concept of periodicity into the right-hand taxonomy of Figure 5. Again, the most obvious approach (indicated by dashed arrows) is not possible as it destroys the tree property of the taxonomy. Fortunately, the replication trick can be applied here, too.

Next, we consider string attributes. String attributes can assume text values with completely unforeseeable contents. Therefore, the challenge lies in tapping the semantic information of the strings. We solve this problem by means of a feature-extraction step that precedes the actual alarm clustering. *Features* are crisp bits of semantic information that, once extracted, replace the original strings. Thus, each string is replaced by the set of its features. Note that subset inclusion defines a natural taxonomy on feature sets. For example, the feature set $\{f_1, f_2, f_3\}$ can be generalized to the sets $\{f_1, f_2\}$, $\{f_1, f_3\}$, or $\{f_2, f_3\}$, which in turn can be generalized to $\{f_1\}$, $\{f_2\}$, or $\{f_3\}$. The next level is the empty set, which corresponds to "ANY-FEATURE." Evidently, the resulting taxonomy is a DAG, not a tree. To transform this DAG into a tree, we use the above replication trick combined with the pruning of edges.

We consider it the intrusion-detection analyst's responsibility to select features that capture as much semantic information as possible. Fortunately, there are well-established techniques to assist feature selection [Koller and Sahami 1996], [Liu and Setiono 1996], [Yang and Pedersen 1997].

## 3.5 Algorithm for alarm clustering

Given the NP completeness of alarm clustering, we have developed an approximation algorithm. An *approximation algorithm* for the problem ($L$, *min_size*, $T_1$...,$T_n$) finds a cluster $C \subseteq L$ that satisfies $|C| \geq min\_size$, but does not necessarily minimize $\Delta(C)$. Obviously, the further an approximation algorithm pushes $\Delta(C)$ to its minimum, the better it is.

The approximation algorithm we have developed is a variant of attribute-oriented induction (AOI) [Han *et al.* 1992], [Han *et al*. 1993], a well-established technique in the field of data mining. Our modification over the classical AOI is twofold: First, we generalize attributes more conservatively than classical AOI does. Second, we use a different termination criterion, which is reminiscent of density-based clustering [Agrawal *et al.* 1998], [Han and Kamber 2000].

To begin with, our approximation algorithm *directly* constructs the (single) generalized alarm that constitutes the algorithm's output. In other words, the algorithm does *not* make the detour over first finding an alarm cluster and then deriving its cover. The algorithm starts with the alarm log $L$ and repeatedly generalizes the alarms in $L$. Generalizing the alarms in $L$ is done by choosing an attribute $A_i$ and replacing the $A_i$ values of all alarms by their parents in $T_i$. This process continues until an alarm has been found to which at least *min_size* of the original alarms can be generalized. This alarm constitutes the output of the algorithm depicted in Table 1.

**Table 1—Clustering algorithm**

**Input:** An alarm-clustering problem *(L, min_size, $T_1$, ...., $T_n$)*

**Output:** An approximation solution for *(L, min_size, $T_1$, ...., $T_n$)*

**Algorithm:** ;

```
1:    G:= L;                    // Make a copy of. L
2:    loop forever {
3:        for each alarm a ∈ G do {
4:            c := number of alarms x ∈ L with x ⊴ a_i
5:            if c >= min_size then terminate and return alarm a_i
6:        }
7:        use heuristics to select an attribute A_i, i ∈ {1...,n}_i;
8:        for each alarm a ∈ G do        // Generalize a[A_i]
9:            a[A_i]:= parent(a[A_i], T_i)_i
10:   }
```

In more detail, line 1 makes a copy of the initial alarm log $L$. This is necessary as the initial alarm log is needed in line 4. In line 5, the algorithm terminates when a

generalized alarm $a$ has been found to which at least *min_size* alarms $x \in L$ can be generalized. If the algorithm did not terminate, then the generalization step (lines 8 and 9) is executed. Here, selecting an attribute $A_i$ is guided by the following heuristic:

> For each attribute $A_i$, let $f_i \in N$ be maximum with the property that there is an alarm $a^* \in G$ such that $x[A_i] \preceq a^*[A_i]$ holds for $f_i$ of the original alarms $x \in L$. If $f_i$ is smaller than *min_size*, then we know that we will not find a solution without generalizing $A_i$ and, therefore, select $A_i$ for generalization. Note that this will not eliminate the optimal solution from the search space. If, on the other hand, $f_i \geq min\_size$ holds for all attributes, then we select the attribute $A_i$ with the smallest $f_i$ value. Clearly, this choice might not be optimal.

We make no formal claim about the above heuristic except that it works well in practice. Evidently, the heuristic influences the resulting clusters, and we plan to investigate this influence. Also, one could conceive a completely different approximation algorithm, for example one that is based on partitioning or hierarchical clustering ([Han and Kamber 2000], [Jain and Dubes 1988]). Our choice for the above algorithm is based mainly on its simplicity, scalability, and particularly good noise tolerance. However, we acknowledge that there is no such thing as a single best clustering algorithm [Fasulo 1999], [Guha *et al.* 2000], [Han and Kamber 2000].

## 3.6 Experience with alarm clustering

### 3.6.1 Experiment setup

This section describes a clustering experiment we conducted on a log of historical alarms. The alarm log is taken from a commercial IDS, spans a time period of one month, and contains 156,380 alarm messages. The IDS sensor was deployed in a network that is isomorphic to the one shown in Figure 4a. Note that this experiment involves real alarm data that was collected during the day-to-day operation of a commercially used network.

Also, we want to stress that we have applied the same algorithm to many more datasets of various environments. However, this and the next two subsections focus on a single data set in order to allow a detailed discussion of the results.

For the purpose of our experiment, we model alarms as 7-tuples. In detail, the individual alarm attributes are the source and destination IP address, the source and destination port, the alarm type, the time-stamp, and the context field. The context field is optional, but when present, contains the suspicious network packet.

For IP addresses and port numbers, we use the taxonomies in Figure 4c. For time-stamps, we use the taxonomies in Figure 5, but without the dashed part. No taxonomy is defined for the alarm types. Finally, for the context field (a string attribute) we use frequent sub-strings as features. More precisely, let $V := \langle a[Context] \mid a \in L \rangle$ denote the multi-set (or bag) of values that the context field assumes in the alarm log $L$. Then, the Teiresias algorithm [Rigoutsos and Floratos 1998] is run on $V$ in order to find all sub-strings that have a user-defined minimum length and minimum frequency. These sub-strings are the features, and each original string $s$ is replaced by the (single) most frequent feature that is also a sub-string of $s$. Thus, all feature sets have size one. Finally, each feature set can only be generalized to the "ANY-FEATURE" level. An

important strength of this feature-extraction algorithm is that the resulting features are very understandable and interpretable, thus increasing the overall understandability of alarm clusters.

### 3.6.2 Results

This section presents the 13 largest alarm clusters we have found (cf. Table 2). Each line of the table describes one cluster, and the "Size" column indicates the cluster's size. The AT column shows the Alarm Types, for which we have provided mnemonic names below the table. In the table, "any" is generically written for attributes that have been generalized to the root of their taxonomies. Note that only alarm types 1 and 2 have context attributes. Therefore, the context attribute is undefined for all other alarm types. Also occasionally, the port attributes are undefined. For example, the ICMP protocol has no notion of ports [Tanenbaum 1996]. As a consequence, the port attributes of alarm type 5 are undefined. Finally, recall that the names refer to the machines in Figure 4a.

**Table 2—The 13 largest alarm clusters**

| AT | Src Port | Src IP | Dst Port | Dst IP | Time | Context | Size |
|---|---|---|---|---|---|---|---|
| **1** | NON PRIV | EXTERN | 80 | ip4 | any | see text | 54310 |
| **1** | NON PRIV | EXTERN | 80 | ip5 | any | see text | 54013 |
| **1** | NON PRIV | FIREWALL | 80 | EXTERN | any | any | 17830 |
| **2** | NON PRIV | FIREWALL | 21 | EXTERN | any | any | 6439 |
| **2** | NON PRIV | EXTERN | 21 | WWW/FTP | any | any | 4181 |
| **3** | undefined | ip6 | undefined | ip1 | WORKDAY | undefined | 4581 |
| **3** | undefined | ip6 | undefined | ip2 | WORKDAY | undefined | 3708 |
| **4** | NON PRIV | ip1 | 80 | any | any | undefined | 761 |
| **4** | NON PRIV | ip2 | 80 | any | any | undefined | 663 |
| **4** | NON PRIV | FIREWALL | 25 | any | any | undefined | 253 |
| **5** | undefined | EXTERN | undefined | ip4 | any | undefined | 823 |
| **5** | undefined | EXTERN | undefined | ip5 | any | undefined | 711 |
| **6** | undefined | ip7 | undefined | FIREWALL | END-OF-MONTH, TUESDAY | undefined | 861 |

**Alarm Types (AT): 1 = "WWW IIS View Source Attack" ;**
**2 = "FTP SYST Command Attack" ; 3 = "IP Fragment Attack" ;**
**4 = "TCP SYN Host Sweep" ; 5 = "Fragmented ICMP Traffic" ;**
**6 = "Unknown Protocol Field in IP Packet"**

It is worth noting that the clusters in Table 2 cover 95% of all alarms. Thus, we have found a very crisp summary of almost the entire alarm log. Moreover, using this summary for root cause discovery is a huge simplification over using the original alarm log (more on this in the following subsection). To estimate the payoff of resolving the root causes, we have written filters that remove all alarms that match one of the clusters.

We then applied these filters to the alarms of the following month. The result was that the filter automatically discarded 82% of all alarm messages. Thus, if the root causes had been resolved, then 82% less work would have been the payoff in the subsequent month.

### 3.6.3 Discussion

This section shows how one can derive hypothetical root causes from the generalized alarms in Table 2. Of course, the hypothetical root causes have to be validated, but in our practical work, this validation has almost always confirmed our first hypothesis. This shows the value of generalized alarms in root-cause analysis. The following discussion proceeds by alarm type:

- *Alarm type 1*: The first two alarm clusters in Table 2 contain the following (sanitized) sub-string in their context fields:

  *GET /search.cgi/cgi?action=View&VdkVgwKey=http%3A%2F%2Fwww%2Ex%2Egov*

  This request is completely legal and, based on Table 2, it has been issued more than 100,000 times. Thus, the most likely root cause is an under-specified signature. Indeed, a detailed analysis has shown that alarms of type 1 occur whenever a URL contains "%2E," as is the case for the above URL. Finally, the third type-1 alarm turned out to be the dual problem: Internal clients requesting external Web pages, the URL of which contains "%2E".
- *Alarm type 2*: These alarms simply highlight the fact that many FTP clients issue the SYST command—a legal command that returns information about the FTP server. The root cause seems to be a signature that is triggered whenever the keyword "SYST" is sent to the FTP port. This root cause has been validated subsequently.
- *Alarm type 3*: Either ip6 is maliciously sending fragmented packets to the firewalls or there is a router that fragments the packets between ip6 and the firewalls. Our investigation has shown that the second hypothesis is correct.
- *Alarm type 4*: Here, the IDS believes that the firewalls are running host sweeps. In reality, however, the firewalls proxy the HTTP (port 80) and SMTP (port 25) requests of their clients. While exercising this function, the firewalls occasionally contact many external machines at virtually the same time. The resulting traffic resembles host sweeps.
- *Alarm type 5*: After investigating the source IPs, we realized that they all belong to different Internet Service Providers. Therefore, we conjectured that there is some link between fragmented ICMP traffic and modem access to the Internet.
- *Alarm type 6*: At the end of the month, a machine on the Internet starts using an unknown transport layer protocol to communicate with the firewall. As many security tools ignore protocols they do not understand, attackers occasionally use unknown protocols to establish covert channels. A closer investigation of this generalized alarm seems to indicate that, indeed, ip7 is trying to set up a covert channel.

In summary, knowledge of generalized alarms has made it rather straightforward to derive the above root causes. Moreover, by deriving only six different root causes (one per alarm type), we have managed to understand 95% of the 156,380 alarms. We have shown that the future alarm load will decrease by 82% if these root causes are removed.

That, in turn, enables a much more thorough analysis of the remaining alarms. Note that it is beyond the scope of this deliverable to discuss the analysis of the remaining alarms.

### 3.6.4 Additional experiments

We have further evaluated the alarm-clustering technique by using it in 16 additional real-world environments. Following the methodology of the previous sections, we treated the different environments separately from each other. In a first step, we clustered the alarms that were triggered in November 2001 for each environment. The resulting generalized alarms were used to carry out a root-cause analysis. However, given that we were experimenting with historical, real-world, alarm logs, it was not possible to fix or remove the discovered root causes. Therefore, we used filtering to estimate the alarm-load reduction that we could expect. Specifically, we wrote filtering rules that discarded all alarms matching one of the generalized alarms. We applied these filtering rules to the alarms triggered in December 2001, measured the resulting alarm-load reduction, and plotted it in Figure 6.

Let us consider Figure 6 in more detail. For example, 68% of the December alarms in environment 1 were automatically discarded by the filtering rules. Consequently, only the remaining 32% of alarms had to be investigated by the human operator. In environment 13, the alarm load reduction was even 89%. Below the abscissa of the bar chart, a rough characterization of the different environments is given. Specifically, each environment is characterized by its functionality and by the IDS product deployed. Possible IDS products are “A” and “B,” both of which are leading commercial, network-based IDSes. To avoid unintended commercial implications, we do not reveal the product names or vendors of "A" and "B." Finally, an environment can have one of the following functionalities:

- *Intranet*: Denotes an internal corporate network without Internet access.
- *DMZ*: Denotes a perimeter network that is protected by a firewall, but offers services to the Internet.
- *Extranet*: Denotes a network that is shared between multiple cooperating companies, but is not accessible from the Internet.
- *Internet*: Denotes an “unshielded” network with a direct connection to the Internet.

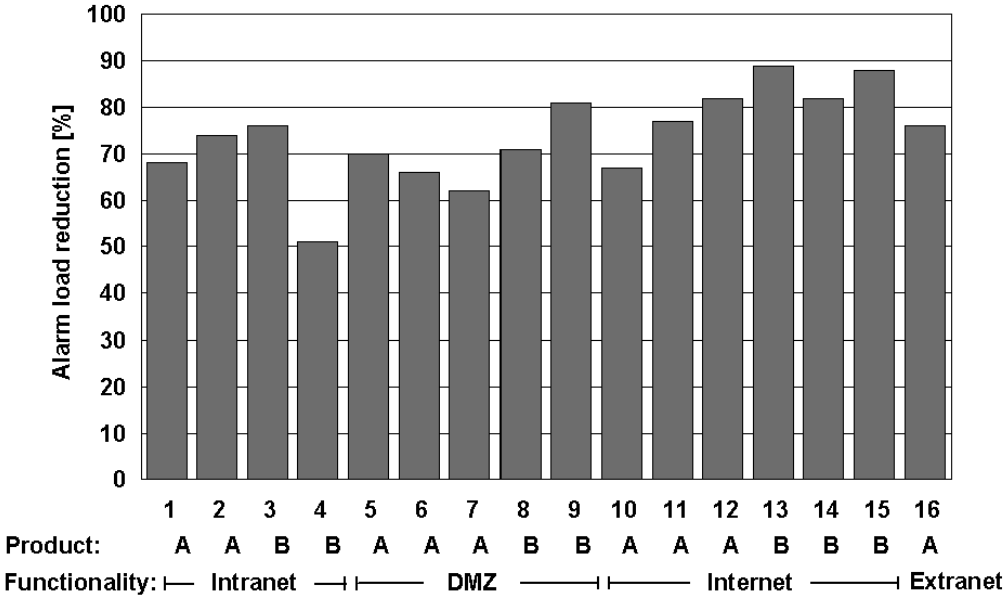



**Figure 6—Alarm-load reduction for 16 different environments in December 2001**

While performing the root-cause analysis for the above environments, we found the output of the alarm-clustering algorithm to be of great help. In fact, using the generalized alarms, we could finish the root-cause analysis in approximately two hours per environment. Occasionally, root-cause analysis was complicated by the fact that generalized alarms contained external hosts that were not part of our administrative domain. In these cases, additional background knowledge about the configuration and usage of these hosts would have facilitated the root-cause analysis. Critics focus on this point and argue that root causes that affect external hosts are difficult to diagnose. Moreover, as these root causes are beyond our control, we generally cannot fix them, but have to resort to alarm filtering or shunning of affected hosts. Both alternatives are claimed to be risky when root causes have not been identified with absolute certainty.

Our objection to this argumentation is twofold: First, we found the scenario sketched by the critics to be the exception, not the rule. Thus, the root-cause analysis paradigm is safe in the majority of cases. Second, even when there is some doubt with respect to the actual root causes, we believe that it is better to take a conscious and explicit decision than to ignore the problem. In fact, root-cause analysis is a structured and controlled process for deciding how to respond to alarms. This response might not be absolutely risk-free, but it is certainly a sound and justified response, and that may well be the best one can hope for.

## 3.7 Preliminary conclusions

In this chapter, we have considered the problem of intrusion-detection systems overloading their human operators by triggering thousands of alarms per day. Our solution to this problem exploits the observation that 90% of these alarms can be

attributed to one out of a small number of root causes. These 90% are highly redundant and distract the intrusion-detection analyst from more stealthy activities. Therefore, we argue that — roughly once a month — one should take the time to identify and remove the most predominant root causes. To make this idea practical, we introduce alarm clustering as a method that supports the discovery of root causes. Furthermore, we show that the future alarm load decreases dramatically if the discovered root causes are removed. Thanks to this reduction in alarm load, analyzing intrusion-detection alarms becomes much more cost-effective and much less error-prone.

This result has the three following consequences on the design of an intrusion-tolerant IDS:

- The design must take into account the operational life of the IDS. Handling large numbers of false alarms involves more than the static definition of technical means, it also implies the definition of a continuous technical process, namely, the clustering process and its consequences: the definition of filtering rules.
- The design must consider the existence of a new type of component, namely, a filtering engine implementing the rules derived from the clustering algorithm. This component is nothing else but the instantiation of the generic notion of the 'event analyzer' defined in Subsection 6.2.1.3 (page 85).
- The analysis suggests the existence of patterns of false alarms for a given environment. Detecting changes in these patterns might lead to the identification of important aspects that would otherwise have remained unnoticed. Using the output of the filtering engine as an input for another type of event analyzer is something we will discuss in Chapter 3.

We consider these results to be extremely encouraging, and they certainly open new research avenues to enhance the results obtained already. They only address one part of our problem though. Indeed, we have to keep in mind that the basic goal of this approach was to reduce significantly—but safely—the number of alarms to be treated by more sophisticated, and hence more expensive, techniques. It has been shown in [Alessandri 2001] that each type of IDS was able to cope with only a subset of all types of attacks. To detect all types of attacks that threaten a given system, the design of the overall IDS must consider using a variety of different individual IDSes. The problem of finding the "best" choice of the set of individual IDSes was left as an open question in D3, we address it in the next section.

# Chapter 4 Efficient combination of IDSes

## 4.1 Introduction

This Chapter is the natural followup of the work done in the MAFTIA Deliverable D3 [Alessandri 2001], in which we had introduced and validated a taxonomy of Intrusion Detection Systems and Attacks. The goal of these taxonomies was to provide an easy way to evaluate the capability of a given IDS to deal with certain types of activities. In this section, we show how to conduct the evaluation work itself. Therefore, we start by defining metrics of individual IDSes and of systems that consists of a combination of several IDSes. The next step describes the technique used to assess systems with respect to certain fault assumptions and the results thereof.

A prototype, named RIDAX, has been built that implements these various concepts. It enables us to evaluate different combination systems of IDSes under various assumptions.

## 4.2 Approach

*Activities* are used to represent IDS input. An activity corresponds to a single event or a sequence of related events. Its intent may be either malicious or benign. Consequently we view *attacks* as representing malicious activities.

For the analysis of a given activity, an IDS has to collect a sufficient amount of data associated to the activity, and examine it. For this, the IDS needs certain capabilities. Our approach describes activities by formulating these requirements in terms of IDS characteristics. More precisely, we describe activities by means of the IDS characteristics required for their analysis. Although this seems relatively simple for a particular IDS and a given activity, activity descriptions become significantly more complex as one wishes to do so for a larger number of IDSes, activities, and activity variants. Each IDS may analyze activities in a completely different manner. This problem had led us to come up with the notion of groups of activities and variants of activities. They are introduced in Section 4.3. Then, in Section 4.4, we introduce a method and several metrics to assess individual IDSes as well as combinations thereof. Finally, in Section 4.5, we present the RIDAX prototype, which implements the method and measures the metrics proposed.

For the sake of conciseness, only the most important notions are described in what follows. We refer the interested reader to [Alessandri 2002] for an in-depth description of the methodology.

## 4.3 Description of activities, activity variants and their components

As explained above, each IDS may analyze activities in a completely different manner. Moreover, depending on its capabilities, an IDS may choose from several different means to analyze a given activity. Therefore, special care has to be exercised to define a scheme that permits efficient and concise description of activities. In order to avoid the creation of large and repetitive descriptions of individual activities or even activity variants, we use a modular approach using multiple components:

1. *Activity groups*: Many activities share important characteristics; for example, they represent an application-layer request or a network-layer PDU. Activity groups express such shared characteristics by describing entire classes of benign activities and attacks.
2. *Activity variations*: Activity variations[4] are rules that express the additional IDS characteristics required for analyzing activities, assuming that the latter have changed slightly.
3. *Expectable alarms*: Every activity variant is associated to a list of expectable alarms.

The following sections describe these components in greater detail, explaining their role in the IDS evaluation process. For BNF definitions of the activity descriptions and examples of activity descriptions, activity groups and activity variations, the interested reader is referred to [Alessandri 2002].

### 4.3.1 Activity groups

We introduce the concept of activity groups in order to simplify the task of creating activity descriptions. In the context of this work, 23 activity groups were created.

Activity groups describe super-classes of activities, i.e., they describe activity characteristics that are shared by several activity classes. The activity group describing argument-based buffer overflows is a good illustration of this:

Activity groups are highly generic building blocks that are specified at a high-level activity scope. As such they are too generic when it comes to describing a specific activity such as an HTTP argument buffer overflow. This led to the concept of "instantiating" activity groups to a lower-level activity scope when used for the description of an activity. This becomes possible by using two activity scopes to specify the validity of an activity group. The first is the high-level activity scope at which the activity group was specified. This scope is fixed by the description of the activity group. The second is what we call the *effective activity scope* and represents a parameter that can be set whenever the activity group is used to describe an activity.

### 4.3.2 Activity descriptions

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

1. Activity groups as described above.
2. IDS capabilities required in addition to the ones formulated by activity groups.
3. The list of activity scopes that are to be used for the instantiation of activity groups.

The second element addresses characteristics of activities that are highly specific to the activity described, i.e. that are too specific to be described by a dedicated activity group. However, we generally try to avoid their inclusions into activity descriptions.

[4] For ease of readability, we refer to "variations" instead of "activity variations" in the following.

The third element has multiple functions. First it specifies the activity scope to which the activity groups have to be instantiated. Its second function, which will be explained in Section 4.3.3, concerns the selection of variations to be applied to the activity.

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

### 4.3.3 Using activity variations to create activity variants

Attacks may be obfuscated in an attempt to elude IDSes [Horizon 1998], [Disk *et al.* 2000], [Ptacek and Newsham 1998], [Puppy 2000], [Sasha and Beetle 2000], [Stewart 1999]. There exist tools such as *Whisker* [Puppy 2000] or *Fragroute* [Song 2002] and its predecessor *Fragrouter* [Song 1999a] that implement such obfuscation techniques. Here we show how such obfuscating transformations of activities can be described by means of activity variations. These are independent of, i.e. not associated to, particular activities. We also provide some background information and explain the difficulties of analyzing altered activities, i.e. activity variants. In a next step we explain how these activity variants can be created by applying variations to activities. Finally we discuss the impact variations have on the evaluation of an IDS when applied to activities.

Knowledge-based IDSes that use some form of attack signature to identify known attacks are susceptible to obfuscation techniques. Generally the goal of varying attacks is to change the appearance of the attack such that existing attack signatures will no longer match. In real-world environments, these techniques unfortunately proved to be fairly effective [Marty 2002]. To make matters worse, most variations can be combined with each other—especially if two variations affect different layers or sub-systems. To render IDS evaluation even more complicated, variations may impact the analysis performed by IDSes in various ways.

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

1. most techniques used to transform an activity also occur in the context of benign activities,
2. the normalization of activities is costly, and
3. in some cases normalization is not possible—at least not in a nonambiguous manner.

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

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

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

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

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

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

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

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

To make matters worse, most variations can be combined with each other—especially if two variations affect different layers or sub-systems. For instance it becomes possible to combine network-layer [Horizon 1998] and application-layer variations [Puppy 2000].

### 4.3.4 Expectable alarms

So far we have developed the description scheme for activities and activity variations, but have not yet explained how the results of their analysis are used to assess IDSes. In order to be able to evaluate the results that an IDS has the potential to produce, a prerequisite is to know the alarms one expects the IDS to generate for any activity. These expectations are tightly coupled to activities, activity groups and variations, and will be described in the following.

Let us therefore assume that an IDS has analyzed an activity based on its description and that the activity was found to have the potential of generating a set of alarms. To judge whether these alarms are to be considered correct, i.e. true positives, we need to extend the description of activities by this additional information. In our solution we specify the expectable alarms per activity group and variation. The specification of an expectable alarm is composed of the following four attributes:

1. *Activity component type*: Activity group, variation or activity.
2. *Activity component name*: Name of the activity component.
3. *Generic alarm name*: Name of the alarm.
4. *Alarm activity scope*: Activity scope at which the alarm condition is formulated.

In assessing the set of alarms that an IDS was found to have the potential to generate for a given activity, we compose the list of expectable alarms and verify whether every activity component for which at least one alarm is to be expected was reported. If, excluding variations, this is not the case, we consider an attack as *not detected* (see also Table 3).

**Table 3—Rating of IDS evaluation results based on lists of expectable alarms**

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

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

## 4.4 Assessment of intrusion-detection systems

Here we develop a process and metrics to evaluate and assess individual IDSes as well as combinations thereof. The IDS evaluation process is described in Section 4.4.1. It determines and rates the alarms that the evaluated IDS has the potential of generating or omitting for a given activity variant. Section 4.4.2 considers the merits of various metrics to assess individual IDSes. Section 4.4.3 proposes a method to take into account the semantics of the alarms generated. This enables us to develop metrics that reflect the usefulness of the information contained in alarms. These results are used, in Section 4.4.4, to assess arbitrary IDS combinations.

### 4.4.1 Evaluation of IDSes

The actual IDS evaluation process (see Figure 7) consists of three steps, which are repeated systematically for every individual IDS and activity variant. Each iteration analyses a given IDS for a given activity variant. In our RIDAX tool this process is fully automated and takes advantage of Prolog's backtracking mechanism to systematically select IDSes and activity variants for evaluation.

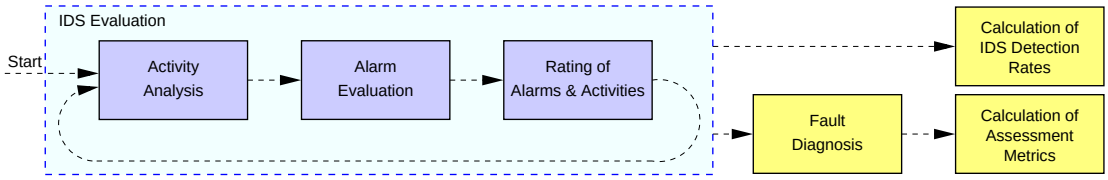



**Figure 7—The three steps of the iterative IDS evaluation process**

Figure 8 provides an overview of the data flow required for each iteration of the IDS evaluation process. The process starts with the analysis of an activity, i.e. activity

variant. As a result this first analysis step provides a description of the capabilities that the IDS being evaluated has used to analyze the activity variant considered. These capabilities are described in terms of IDS characteristics. The process then continues with the alarm evaluation step, which uses the IDS characteristics that were needed to analyze the activity variant considered as input. This alarm evaluation step generates the list of alarms that the IDS being evaluated has the potential of generating for the activity variant considered. In the final step of the IDS evaluation process these alarms are rated whether they represent a true or false positive. Moreover, it is determined whether the IDS has the potential of suffering from false negatives, i.e. of not reporting malicious activities as expected. This analysis also includes the rating of malicious activity variants, i.e. whether they have been detected, partially detected, or not detected (see also Section 4.3.4).

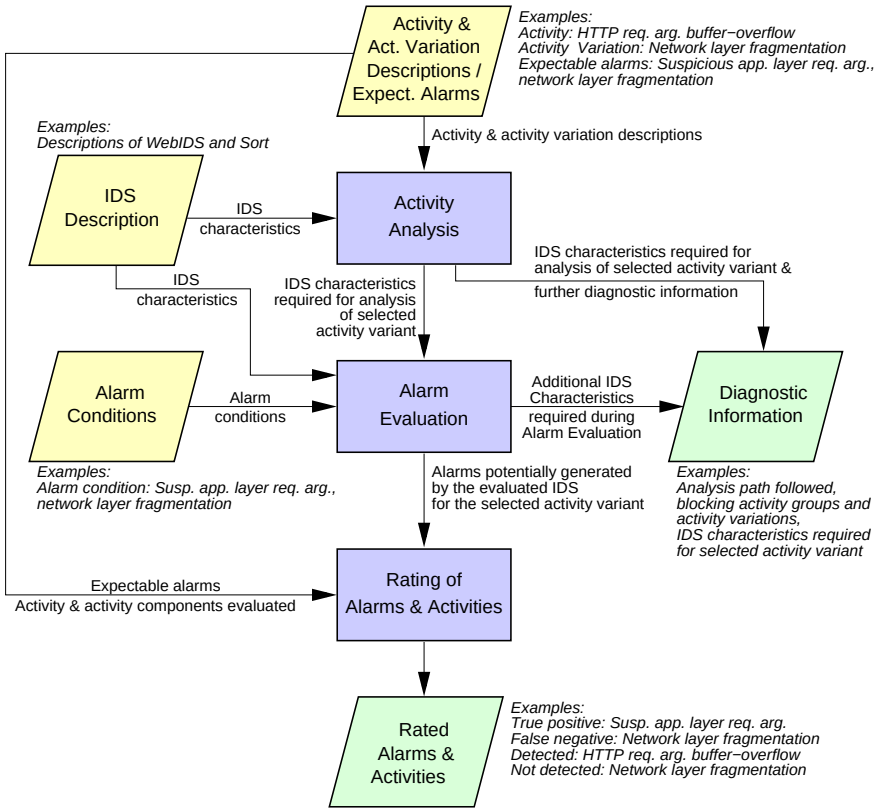



**Figure 8—Overview of the data required in and generated by each iteration of the IDS evaluation process, including examples**

In the following subsections we discuss the three IDS evaluation steps in detail.

#### 4.4.1.1 Activity analysis

The first step of the IDS evaluation process (shown in Figure 8) is the analysis of the manner the IDSes under evaluation analyze activities and their variants. This requires the IDS descriptions and the activity descriptions as input.

The analysis starts with the selection of the activity variant for which the IDS is to be evaluated. This selection is made in a hierarchical manner; i.e. first the activity to be analyzed is selected from the list of available activities. Then the set of variations to be applied to the activity is selected. A new activity is only selected for evaluation if all

applicable combinations of variations have been considered. Similarly, a new combination of variations is only selected if all possibilities of how the IDS considered may analyze the selected activity variant have been examined. This means that the IDS evaluation process might iterate multiple times for the same activity variant—each time exploring a different analysis approach offered by the IDS.

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

The systematic and complete selection of activities as well as the selection of variations rely on the backtracking provided by Prolog [Clocksin and Mellish 1994]. It thereby ensures, in a straightforward manner, that each activity variant is considered, and that all possible analysis approaches are examined.

Once the activity variant has been selected, the activity analysis proceeds by determining how it is analyzed by the IDS under evaluation. The various input items used are shown in Figure 9 in more detail. Owing to the rule-based manner in which activities are described, most of the analysis process is defined implicitly. At this stage we systematically search for ways to analyze the activity variant considered using only the capabilities provided by the IDS under evaluation. While doing so, we record all the IDS characteristics employed for analyzing the activity.

**Example:** *Consider an HTTP argument buffer overflow attack being analyzed by a knowledge-based IDS such as WebIDS[Almgren 1999]. In this case one typically obtains an analysis result that shows that the IDS was analyzing the HTTP instance at the semantic analysis level using a string-matching technique or possibly simply verifying the request length. In addition the result would reflect all sensor items required for the analysis.*

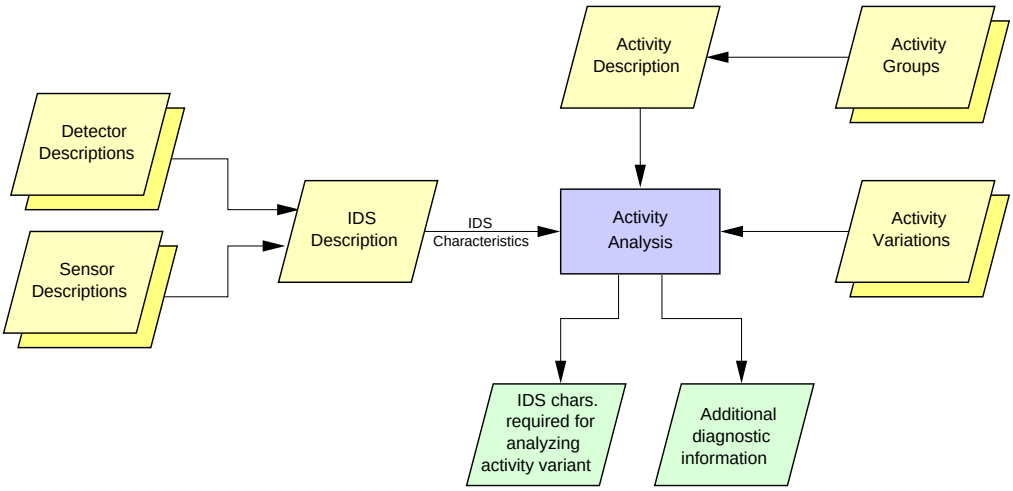



**Figure 9—Input required for and output generated by the activity analysis step**

If the evaluated IDS is incapable of analyzing the activity variant considered, the resulting set of IDS characteristics required for analyzing the activity variant will lack the unavailable IDS characteristics. However, in general more than just the unavailable IDS characteristics will be missing in the result. Typically the IDS characteristics required for analyzing the activity group or variation whose analysis failed as well as the characteristics required for analyzing the omitted activity groups and variations will be missing. The analysis of further activity groups and variations is omitted if there is no alternative way of analyzing the activity variant considered. In the extreme case, i.e. if the IDS considered is not even capable of observing the activity variant, the result will be an empty set of IDS characteristics.

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

#### 4.4.1.2 Alarm evaluation

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

Figure 10 shows the input required for and the output produced by this step. As in the preceding step, the IDS descriptions are required. In addition, the alarm conditions and IDS characteristics that were required to analyze the activity variant considered are required as input. On the output side we find the set of alarms that the IDS under evaluation was found to have the potential to generate. We also obtain the list of

additional IDS characteristics that were required for the generation of the alarms, in addition to some debugging information.

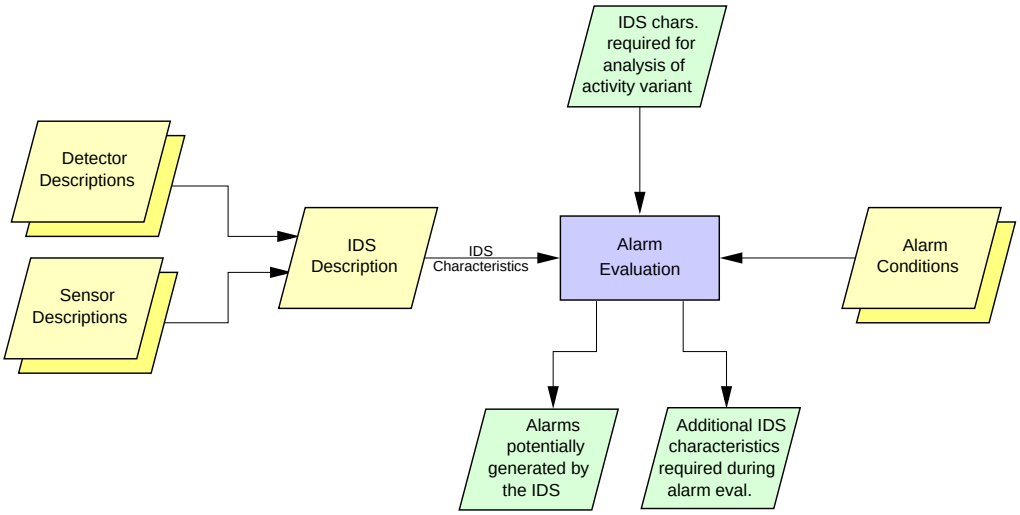



**Figure 10—Input required for and output generated by the alarm evaluation step**

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

#### 4.4.1.2.1 Alarm conditions

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

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

- *Basic HTTP arguments*
- *String matching for HTTP info*

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

As illustrated by the example above, it is possible to specify the requirements an IDS needs to fulfill to be capable of generating the alarm at highly specific activity scope,

e.g. HTTP. Such practice would, however, lead to the repeated description of conditions that only differ in the activity scope that they apply to. This would not only result in an unnecessary large number of alarm conditions, but would also cause an eminent loss of generality. As a consequence we require alarm conditions to be more generic. We achieve this by associating them with two different activity scopes—similar to the method we used for the description of activity groups. This means that alarm conditions are defined at the highest activity scope level possible. Then, during alarm evaluation, we determine the effective activity scope of the alarm based on the activity scopes of the IDS characteristics required for the analysis of the activity variant considered. The effective activity scope of the alarm may not be less specific than the activity scope at which the alarm condition was specified. Technically, the identification of the effective activity scope is achieved by taking advantage of Prolog's inference engine [Diaz 2000]

Remember, the important property is that alarm conditions are independent of both activity and IDS descriptions. However, how does one know which alarm conditions to create if they are independent of activity and IDS descriptions? The 19 alarm conditions that we created in the context of the RIDAX prototype implementation were identified by searching for attack super-classes in the attack classification described in [Alessandri 2001]. We focused our effort on activities and variations for which we actually created descriptions. These alarm conditions are of high generality as they are defined at a high-level activity scope.

#### 4.4.1.2.2 Alarm semantics

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

**Example:** *Referring to above example, and considering an IDS that "simply" checks for the presence of a string. Such an IDS is not capable of providing the information whether the supposedly identified attack was successful as long as the reaction of the process being attacked is not taken into account. This typically is difficult for network-based IDSes but less so for host-based systems because for the latter it is generally easier to get hold of the required information.*

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

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

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

The use of this 5-tuple also helps us differentiate alarm classes based, for instance, on the detection method used by the IDSes. Differentiating alarms that behavior- and knowledge-based IDSes generate is necessary because they differ significantly in their expressiveness and semantics. This difference is also reflected by the corresponding alarm conditions. Knowledge-based IDSes include a limited description of the attack identified in the alarm. They may also include additional diagnostic information such as IP addresses. Such alarms generally refer to identifiers such as CVE [CVE99]. In the case of behavior-based IDS the situation is different. These IDSes commonly only express the fact that suspicious activity was observed by signaling the fact that the system being monitored deviates from its normal behavior. Their alarm identifier may for instance express the fact that a strange, abnormal sequence of instances was identified. However, such an identifier reveals no concrete information about the cause of the nonacceptable sequence.

**Example:** *Considering our HTTP buffer overflow example. Assuming that IDSes such as Snort, WebIDS or DaemonWatcher*[5] *are found to have the potential of detecting this attack, the corresponding alarms could look as follows:*

**Table 4—Example of how various IDSes report a buffer overflow attack**

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

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

#### 4.4.1.3 Rating of alarms and activities

In the last step of the IDS evaluation process, the alarms that IDSes were found to have the potential to generate are rated. This is done based on the expectable alarms for the

[5] These detectors are described in 4.5.2.1, on page 53.

activity components involved (see also Section 4.3.4). These are used to verify whether every activity component for which at least one expectable alarm was specified generated at least one of the expectable alarms. If this is not the case, the activity component concerned is rated as "not detected."

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

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

Concurrently also all the activity components are rated:

- *Benign:* There are no expectable alarms specified for the activity component.
- *Detected:* At least one of the expectable alarms was generated.
- *Not detected:* None of the expectable alarms was generated.

Using these ratings for activity components, we then rate the complete activity variant as explained in Section 4.3.4 (see Table 3, p. 35). Note that regardless of how an activity variant is rated, false positives may be present. In fact, false positives may obfuscate the result such that it becomes even more challenging to draw precise conclusions based only on the set of alarms generated for activity variants.

**Example:** *Consider again the HTTP argument buffer overflow attack (see also the example provided in Section 4.3.4). Moreover assume that the alarm evaluation reveals that the IDS has the potential of reporting the activity group describing the actual argument buffer overflow by means of an alarm. We would rate this activity group as "detected" if the alarm indicates the observation of a suspicious string in the request. The same is true for any other alarm, such as an alarm reporting the observation of a suspicious execution path, that is "expectable" for this activity group. Accordingly any alarm that appears on the list of expectable alarms is rated as a true positive. Assuming that the buffer overflow activity group is the only malicious one referenced in the activity description of the attack, we would consequently also rate the entire attack as "detected." If, however, the same attack was reported only by a nonexpectable alarm, we would rate the attack as "not detected," and the alarm would be rated as a false positive.*

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

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

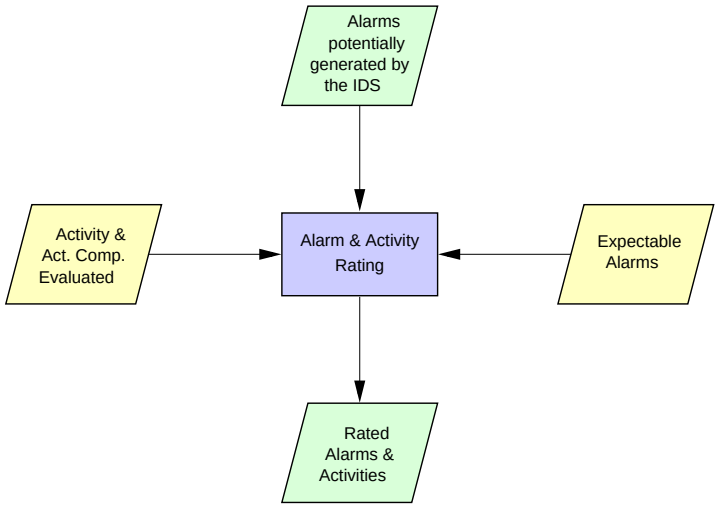



**Figure 11—Input required for and output generated by the alarm and activity rating step**

### 4.4.2 Detection rate of IDSes

In this section we develop simple metrics that allow us to judge IDSes in a manner similar to existing approaches. We have included methods to calculate these metrics in RIDAX, which automatically calculates the results on a per-IDS basis in continuation of the IDS evaluation process. The methods measure the total number of true positives, false positives, and false negatives in a manner that we derived from the concepts of *precision* and *recall*, as used in the information-retrieval field. However, in doing so there are some issues that we need to be aware of:

- The semantics of the alarms generated in the context of our approach differs significantly from that generated by IDS implementations. First, the alarms generated using our approach denote the potential of the IDS to generate a given class of alarms. Second, our approach may report, i.e. describe, activities using multiple alarms. This can be caused by the combination of multiple activity components, but also by single activity components. It is clear that real IDSes may also generate multiple alarms when reporting a single activity. Therefore comparing absolute alarm numbers may result in misleading measurements.
- The number of false negatives determined by our approach cannot be compared with the number of false negatives determined while evaluating real IDSes. This is due to the fact that we rate every alarm that was expectable (see Section 4.3.4) but was not generated as a false negative. However, as explained in Section 4.4.1.3, these false negatives only matter if no true positive reports the activity component in question. When judging coverage provided by an IDS, we focus on whether a malicious activity variant was rated “detected” or not (see Section 4.4.1.3).
- The utility of the information carried by the alarms generated is not measured.
- The measurements are not made in a real environment, and encompass only one instance of every activity variant. However, as explained in the preceding section, other approaches suffer from this environment-related issue to an equal

or greater extent. In spite of this, the results will provide a normalized assessment of the IDSes evaluated.

- These issues provide the motivation for the considerations made in the next section in which we propose an assessment method that enables us to include the utility of the information provided by IDS alarms.

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

- Recall $r$: the ratio between the malicious activity variants detected $m_d$ and the total number of malicious activity variants considered. The total number of malicious activity variants is computed by summarizing the number of detected malicious activity variants $m_d$, the number of partially detected malicious activity variants $m_{pd}$ and the number of nondetected malicious activity variants $m_{nd}$:

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

  As in our approach this measurement is normalized, recall is equivalent to coverage. Coverage describes the ratio of all considered attacks an IDS is capable of detecting, without taking into account the frequency of the attacks.

- Precision $p$: the ratio between the number of true positives $a_{tp}$ and the total number of alarms generated. The latter is composed of $a_{tp}$ and all false positives $a_{fp}$, i.e. all the alarms IDSes generate:

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

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

Note that in above definitions we used both rated alarms ($a_{tp}$ and $a_{fp}$) as well as rated activity variants ($m_d$ etc.). We chose to do so because the resulting definitions of precision and recall reflect the relevant information well. Moreover, the resulting definitions are to some extent comparable with the measurements determined in other approaches such as the Lincoln Lab experiment (see Section 5.2.2). It would have been misleading to use the number of false negatives for the definition of recall, for instance, because the semantics of the false negatives as used in the context of our approach differs considerably from that used in other works.

### 4.4.3 Fault diagnosis based on alarms generated by multiple IDSes

As explained above, existing metrics neither take into account nor measure the utility of the information provided by alarms; they are not suitable for assessing combinations of individual IDSes. As a consequence we propose a method that attempts to take into account the semantics of alarms and, even more importantly, that of alarm sets. This will enable us to develop metrics that reflect the usefulness of the information contained in alarms and alarm sets.

#### 4.4.3.1 Information provided by alarms

In Section 4.4.1.2.2 we introduced a 5-tuple of alarm properties to represent alarms. This representation enables us to distinguish semantically between different alarms without having to describe their semantics explicitly. Finally the location of the IDS, also the environment, and possibly other factors influence the alarm semantics.

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

In the following we wish to assess and exploit the complete information provided by alarms. Hence, we searched for an analogy that would permit us to do so. We found that signals known from information and coding theory [Cover and Thomas 1991] provide a suitable analogy. Accordingly alarms can be viewed as being output signals that result from symbol transmissions over a channel. In this model an IDS corresponds to the channel which transforms symbols, i.e. activity variants, as they are transferred. Owing to the fact that in general information may be lost in transmission, it is not always possible to determine the initial symbol based on the output signal, i.e. the set of alarms an IDS generates. In the following we assess the completeness and utility of the alarms IDSes generate by measuring how well conclusions on the respective activity variants can be drawn based on the set of alarms. This approach also enables the assessment of the potential gains one can achieve in terms of completeness and utility by combining individual IDSes, because such combinations can be viewed as the parallel use of multiple diverse channels.

**Example:** *Consider a network-based IDS such as Snort to represent a broad-band channel and a host-based IDS such as WebIDS a comparably narrow-band channel. This would mean that many signals, which use carrier frequencies that can be transmitted over the Snort-channel, could not be transmitted over the WebIDS channel, because the latter is not capable of transmitting signals at all the frequencies the former can. On the other hand, one can expect the signals that were successfully transmitted over the WebIDS channel to be of better quality than if they had been transmitted over the Snort channel. Reverting to ID, Snort is capable of analyzing SMTP (mail) messages, whereas WebIDS cannot. On the other hand, WebIDS is capable of taking into account the HTTP server's request return code in order to determine the success state of a potential attack, whereas Snort cannot. Similar considerations can be made with respect to variations such as IP fragmentation. (See also Table 4).*

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

#### 4.4.3.2 Fault diagnosis based on alarm sets

In this subsection we develop a method that applies the considerations made in the preceding section to the results obtained by evaluating IDSes as described in Section 4.4.1. The analysis method described is implemented by RIDAX, and is automatically executed as continuation of the IDS evaluation process. It is not related to the method described in Section 4.4.2.

Every alarm that an IDS generates indicates the observation of a presumably suspicious activity. With a probability specific to them, alarms thereby indicate an error that may

lead to a security failure. However, the probabilistic aspects of the relation between activity variants and alarms are not explored in the following, because this would require an in-depth knowledge of the environment, which lies outside the scope of this deliverable.

Here we view alarms and alarm sets as signals indicating a set of possible generating activities. Remember it is irrevelant whether an alarm is a true or a false positive. In some cases false positives may even support the identification of activities or their rating as being malicious or benign. In the following we therefore consider IDSes to be performing a simple uni-directional projection $f^{v}_{IDS}$ that depends on the IDS being evaluated. Figure 12 illustrates this projection. Note that one of the alarm sets can denote the empty set.

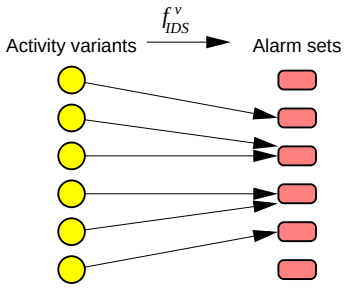



**Figure 12—Projection of activity variants to alarm sets**

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

In order to simplify the analysis we shall only identify the activity but not the activity variant derived from it. This simplification, which is illustrated in Figure 13, can be made without losing important information, because variations generally represent benign alterations of activities.

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

Figure 13c shows that it may not be possible to unambiguously identify the activity that caused a given alarm set. In the figure this is illustrated by alarm set $R_3$. However, knowing this relation may prove quite useful. If, for instance, $a_1$ and $a_2$ are both benign, we can identify the alarm sets $R_2$, $R_3$ and $R_4$ as reports of benign activities that require no further attention. If both activities are considered malicious, $R_3$ at least informs us of a malicious activity in progress. Knowing the above mappings, we can even abridge the list of possible causes to a set of two activities. If, however, one of the activities is

benign and the other malicious, any observation of $R_3$ is unfortunate, because it is impossible to judge whether the detected activity is benign or malicious. This means that rating the cause, i.e. rating the activity, based on $R_3$ has become *ambiguous*.

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

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

Last but not least, one might also find that too many benign activity variants cannot be rated unambiguously. These shortcomings can be addressed by increasing the information provided by the alarms sets used for analysis, which is exactly what was discussed in Section 4.4.3.1. In other words, using alarms generated by multiple diverse IDSes to compose the alarm sets will increase the information they provide. If we use the alarm representation as introduced in Section 4.4.1.2.2, the combination of alarms generated by diverse IDSes becomes straightforward and does not require any changes to our analysis method. Note, however, that combining IDSes may improve some of the issues mentioned, but may make others worse.

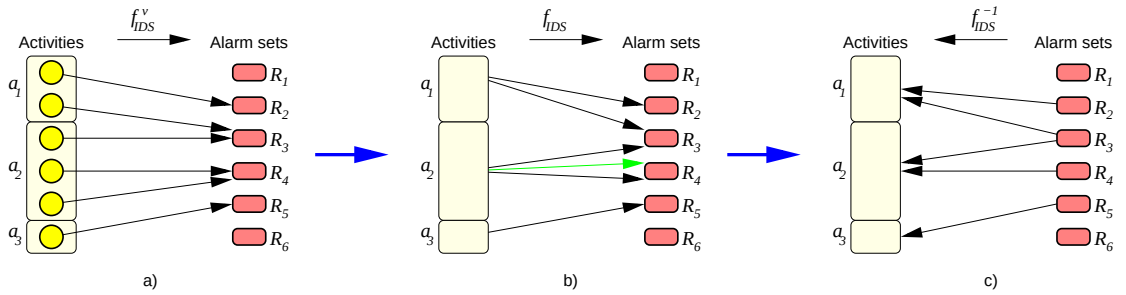



**Figure 13—Projection of activities to alarm sets and vice-versa**

In summary, this method of alarm-set analysis enables us to combine the alarms generated by multiple IDSes by identifying the set of activities that may cause the generation of a given alarm set. Consecutively an activity may be identified if the alarm set considered cannot be associated to any other activity. In a similar, but even simpler fashion, we can also attempt to rate the possible cause of an alarm set as being benign or malicious. If no clear rating can be made, the alarm set is considered ambiguous.

### 4.4.4 Metrics for assessing individual IDSes and their combinations

In Section 4.4.2 we have defined the metrics *precision* and *recall* that enable a relatively simple but limited IDS assessment. These metrics cannot be easily applied to IDS combinations and do not assess the utility of the information provided, both of which are declared goals of this work. In this section we address this issue by proposing a set of metrics that assesses the completeness and utility of arbitrary IDS combinations based on results obtained by analyzing alarm sets as described in the preceding section.

Before starting to define metrics, we need to clarify their goals. It is clear that in the most general case one seeks to measure and optimize *coverage* of the ID architecture envisaged (see Section 4.4.2). When referring to the optimization of coverage, we should be aware that it often does not make sense to aim for total coverage. Often it

suffices to optimize coverage in a given domain, i.e. just for a given set of activity scopes, according to the security policy and the environment to be protected.

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

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

#### 4.4.4.1 Attack recall

The so-called attack recall metric is highly similar to *recall* as defined in Section 4.4.1.3. Instead of considering only individual IDSes, we expand our considerations to a set of IDSes. One notable difference, however, is the fact that we do not distinguish between alarms generated because of variations and alarms with other causes. This is due to the manner alarms are analyzed (see above). In this analysis we do not distinguish between false positives and false negatives, which means that a malicious activity variant is considered *detected* if the IDS evaluated reacts with the generation of an alarm—independent of what the alarm is reporting. Note that it may not be possible to *identify* the activity based on the alarm set observed.

Furthermore, it is clear that one cannot simply add the numbers of detected and not detected malicious activity variants of all the IDSes considered because the domains they cover may overlap. In other words, $m_d$ denotes the number of malicious activity variants that any of the IDSes involved was able to detect. In order not to complicate things further, let us consider activity variants that were detected only partially as "not detected." This results in the definition of $m_{nd}$ as the number of malicious activity variants that were not or only partially detected. Finally we can define *attack recall* $r_a$ as follows:

$$r_a = \frac{m_d}{m_d + m_{nd}}.$$

Again, in our case where only a single instance of every activity variant is considered, $r_a$ provides a measurement for the coverage achieved by the combination of IDSes considered.

#### 4.4.4.2 Attack identification recall

Attack identification recall is similar to the definition of attack recall. The only difference is that we count the number of malicious activity variants $m_i$ for which it was possible to determine the activity that the activity variant was derived from. This results in the following definition of $r_i$:

$$r_i = \frac{m_i}{m_d + m_{nd}}.$$

It is clear that because $m_i \leq m_d$, $r_i$ will never be larger than $r_a$, i.e. $r_i \leq r_a$.

#### 4.4.4.3 Attack identification precision

Attack identification recall provides a measurement in absolute terms. However, in most cases knowing the ratio of detected attacks that can be clearly identified is of greater interest. This is because it provides a measurement for the quality of the detection process. This relative metric $p_i$ can be defined quite easily as follows:

$$p_i = \frac{m_i}{m_d}.$$

#### 4.4.4.4 Rating ambiguity

In measuring recall it is important to verify whether the IDS or the set of IDSes considered provide the required coverage. However, when it comes to assessing the usability of the system, an important parameter is what we call the *rating ambiguity* $r_a$. This absolute metric measures the ratio of all activity variants considered that cannot be rated unambiguously as benign or malicious. The number of ambiguously rated activity variants is composed of malicious ($m_a$) and benign ($b_a$) activity variants. $m$ denotes the total number of malicious and $b$ the total number of benign activity variants considered.

$$r_a = \frac{m_a + b_a}{m + b}.$$

The resulting measurement provides an indication of the operational effort one has to spend because of false positives that cannot be clearly identified as being false positives. As explained earlier, we consider false positives as troublesome only if they cannot be recognized. If we observe a set of alarms that may only have benign causes we can simply discard them, i.e. they do not require any further treatment. Ambiguous alarm sets are troublesome because their generation may be caused by benign activity variants. This is especially annoying because in real-world environments benign activity variants may occur very frequently—far more frequently than the corresponding attacks. So, as a consequence one should focus on composing IDSes such that the overall rating ambiguity is as low as possible.

#### 4.4.4.5 Rating precision

The metric *rating precision* $p_r$ is closely related to the rating ambiguity metric. The two differences are that it is relative instead of absolute and that it assesses the activity variants that can be rated unambiguously. $m_d$ denotes the malicious and $b_d$ the benign activity variants that were detected or reported. $m_{na}$ represents the malicious and $b_{na}$ the benign activity variants that were unambiguously rated as benign or malicious.

$$p_r = \frac{m_{na} + b_{na}}{m_d + b_d}.$$

When measuring the rating precision, we obtain an idea of the usability of the IDS combination considered with respect to the coverage it provides.

## 4.5 RIDAX

### 4.5.1 Implementation: RIDAX, a tool for assessing IDSes

We chose to describe activities, activity components and alarm conditions in a rule-based fashion because it seemed especially well suited for identifying alarms that an IDS potentially generates (see Section 4.4.1). This solution is appropriate for creating and combining the generic and high-level descriptions of activity components introduced in Chapter 1. For the implementation of RIDAX we chose the GNU implementation of *Prolog* by Diaz [Diaz 2000], as it is a rule-based language and meets our requirements that include database connectivity. Database connectivity is implemented by integrating Prolog with the MySQL database [MySql 2000] using their respective C interfaces. MySQL is already used to store IDS descriptions.

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

#### 4.5.1.1 Database structure

During the evaluation process RIDAX stores all the results generated into a database for later evaluation. This database has close relations to the database used to store IDS descriptions. Figure 14 shows a simplified entity relationship diagram of this database, in which all entities taken from the IDS description database are shaded. These entities are not specific to the evaluation result database but solely illustrate the relations to the database developed in the MAFTIA D3 deliverable [Alessandri 2001].

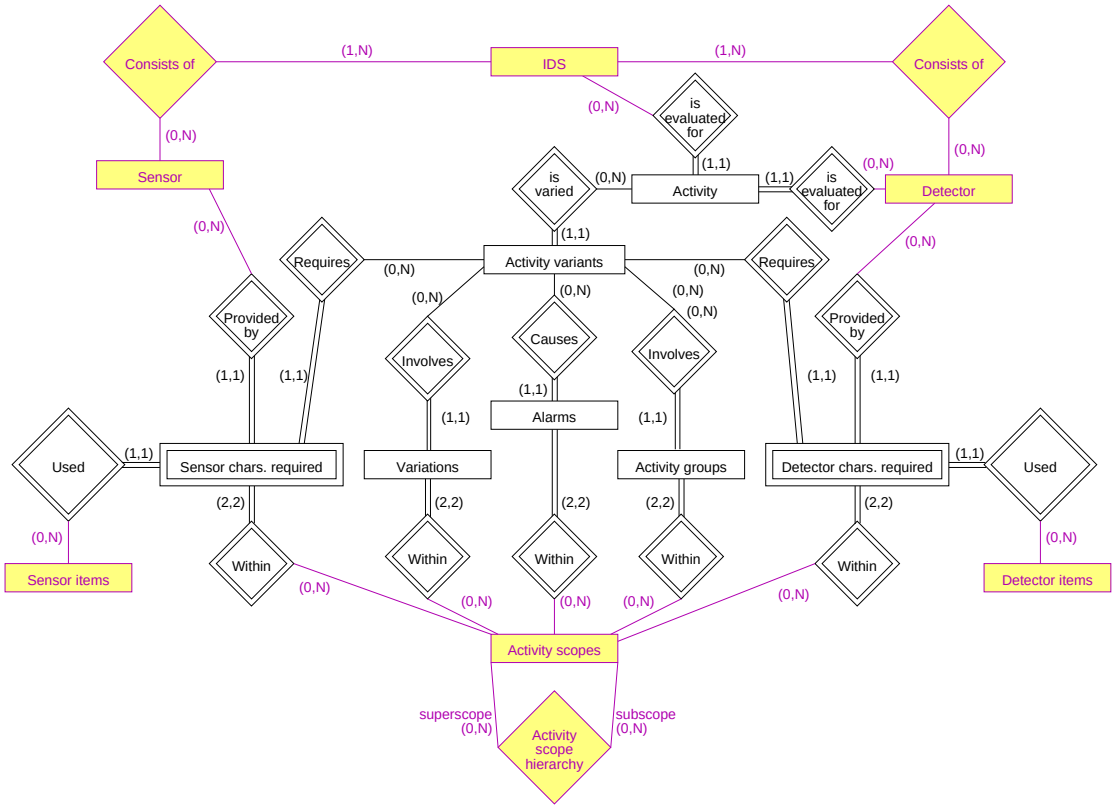



**Figure 14—Entity relationship diagram of the database used to store evaluation results**

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

#### 4.5.1.2 Evaluation steps

In the following we provide an overview of the IDS evaluation steps as they are implemented in Prolog—without going into details of the Prolog code.

##### 4.5.1.2.1 Initialization

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

In a next step all the rules are loaded into the Prolog environment. Some of these rules control the flow of the evaluation process, whereas others represent the descriptions of activities, activity groups, activity variations, alarm conditions etc.

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

##### 4.5.1.2.2 Implementation of activity analysis

After the initialization phase has completed, the Prolog backtracking is used to evaluate all IDSes described for all the activities. This also includes the evaluation of all possible activity variants that may be derived from any given activity. While doing so, all IDS characteristics required during the activity analysis are stored as dynamically asserted Prolog facts and are written to the database. As explained in Section 4.3.3, variations may suppress sensor items or may render the analysis capabilities of detectors useless for further analysis; this is also taken into account by asserting dynamic facts.

##### 4.5.1.2.3 Implementation of alarm evaluation and rating

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

Concurrently all missing alarms are identified, which are then stored as false negatives into the database.

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

#### 4.5.1.2.4 Fault diagnosis and calculation of metrics

Once every IDS has been evaluated for every activity variant, the evaluation process comes to its end. What we have obtained by now is a set of rather large database tables reflecting how the IDSes have analyzed the activity variants and the alarms that the IDSes were found to have the potential of generating. Based on this data, we can calculate the statistical data required for calculating per-IDS precision and recall as defined in Section 4.4.2.

In addition we further analyze the results obtained by identifying and analyzing alarm sets as described in Section 4.4.3. From this analysis we obtain, for every individual IDS but also for every possible combination of IDSes, measurements compiled using the metrics defined in Section 4.4.4.

## 4.5.2 Experiments

In the following we discuss experiments made using RIDAX. We do so by first providing brief descriptions of the IDSes evaluated in the course of these experiments. Then we discuss the attack-detection rates, which are based on rated alarms, using the metrics described in Section 4.4.2. Next, we consider examples obtained while performing the far more advanced alarm-set-based fault diagnosis as developed in Section 4.4.3. We continue by considering measurements obtained by applying the metrics developed in Section 4.4.4 to the results of the alarm-set-based diagnosis. Whereas earlier attack detection rates are discussed primarily for the sake of comparison with other approaches, the measurements obtained using the alarm-set-based analysis enable us to assess the viability of IDS combinations in a manner not possible before.

Our experiments incorporate most of the concepts developed in the course of this work, including the descriptions of the 48 activities identified and seven variations [Alessandri 2002]. Up to two of these seven variations were applied concurrently to each of the 48 activities. This resulted in a total of 928 activity variants considered in our experiments. Of these 498 are considered malicious, and 430 benign. As explained in Section 4.3.3, any given variation can only be applied to activities that involve the activity scope addressed by the variation. Therefore not every activity leads to the same number of activity variants.

Technically RIDAX is capable of applying any number of variations to any activity concurrently. However, we limited this number to two because we felt it necessary to set an upper limit for practical reasons. This limit enables us to investigate the effects of multiple, concurrently applied variations, while providing us with a workable solution. The resulting number of 928 activity variants is believed to represent a meaningful test set because they were generated in a systematic fashion, which assures consistent coverage of a significant portion of the most relevant attack classes.

#### 4.5.2.1 IDSes evaluated

For our experiments we had to select a small number of IDSes from a large list of candidates (see Sobirey's list [Sobirey 1998]). This means that we need selection criteria that are well suited to our primary goal, which is the investigation of potentially achievable gains by combining diverse IDSes. As a result we selected three IDSes, of which we consider five different configurations, by following the following criteria:

- *Diversity:* We require at least one knowledge-based and one behavior-based IDS, as well as at least one host-based and one network-based system.
- *Practicality:* As these experiments are conducted for demonstration and validation purposes, it must be possible to describe the IDSes with only limited effort. This means that the internals of the IDSes need to be available, and, ideally, that they are already well known.

In our experiments we chose to focus on IDSes monitoring network services because various approaches to intrusion detection have been developed in this area. This enables us to control the effort spent for the RIDAX prototype implementation as well as to address both network-based and host-based IDSes, which may either use knowledge- or behavior-based methods. We are aware that we thereby exclude classes of attacks such as those that describe attacks staged by local users. Nevertheless we believe this choice to be viable because, at this point in time, the goal of our experiments is merely to prove and illustrate the validity and flexibility of the concept and not so much the assessment of IDSes at large. Based on these requirements and considerations, we selected the three IDSes listed in Table 5. The last column defines the identifiers for the corresponding IDSes as used in the following discussion.

**Table 5—IDSes evaluated using the RIDAX prototype**

| IDS | Detection method | Information source used | Configuration | ID |
|---|---|---|---|---|
| DaemonWatcher [WeDaDe00, WesDeb99] | Behavior-based | System level log (audit log) | HTTP | DWH |
| | | | FTP | DWF |
| Snort [Roesch99] | Knowledge-based | External raw data (network PDUs) | Simple | SNS |
| | | | Full | SNF |
| WebIDS [Almgren 1999] | Knowledge-based | Application level log (httpd logs) | Normal | WI |

We used a total of five different configurations of these three IDSes for our experiments: Two configurations of DaemonWatcher—one configured for the FTP and one for the HTTP daemon; two configurations of Snort—one using only basic capabilities and one with all additional modules enabled, and one of WebIDS.

##### 4.5.2.1.1 DaemonWatcher for ftpd and httpd

DaemonWatcher [Wespi *et al.* 2000], [Wespi and Debar 1999] by Wespi *et al.* is a behavior-based system that analyses the audit records of processes as they are written by the OS. The system was developed at the IBM Zurich Research Laboratory. For our experiments we consider two configurations of DaemonWatcher. A first configuration covers buffer overflow attacks against the FTP daemon, and the second covers the corresponding attacks against the HTTP daemon.

DaemonWatcher matches the per-process sequences of system calls to system-call sub-sequences stored in a database. The latter represent known benign sequences that were

isolated in a training phase. During this training phase, ideally, all possible execution paths of the executable to be protected are exercised and recorded. From these system-call traces, sub-sequences are isolated using a pattern-extraction algorithm. The resulting sub-sequences describe the complete system-call trace and thereby provide a sufficient description of the normal behavior of the executable to be monitored. As a result DaemonWatcher does not require signature updates for new attacks. However, its disadvantage is that its alarms do not identify the attack staged against the monitored process.

#### 4.5.2.1.2 Snort

Snort [Roesch99] by Roesch is a network-based IDS that is freely available. Snort has become very popular, and is broadly supported by the open-source community. This support consists primarily of new signatures that are being made available on a daily basis and additional modules that extend Snort's detection capabilities. For our experiments we used two configurations of Snort v1.7. One configuration uses none of the resource-intensive extension such as the TCP stream re-assembly module. For the second configuration we consider all these additional features enabled.

Because Snort is a knowledge-based IDS, its signature database needs to be updated as new attacks become known. The signatures are primarily PDU or stream-oriented and support the verification of various protocol flags as well as string matching applied to the payload of the PDU.

#### 4.5.2.1.3 WebIDS

WebIDS [Almgren 1999] by Almgren is a host-based, lightweight IDS tailored to the protection of webserver services (httpd) that was developed at IBM Zurich Research Laboratory. The IDS was implemented in the scripting language Perl [Perl87] and relies on its powerful regular expressions to recognize attacks. In addition the system builds up a list of suspicious hosts and is capable of statistical analysis towards the recognition of flooding attacks. It is clear that the signature database of WebIDS needs to be updated as new attacks become known.

### 4.5.2.2 Detection rates of individual IDSes

In Section 4.4.2 we defined the metrics precision and recall by deriving them from the corresponding definition in the information-retrieval field. The resulting measurements can only be compared in a limited fashion with the results produced by known benchmarking approaches such as the Lincoln Lab evaluation (see Section 5.2.2). A first important difference is the fact that we considered a "normalized" environment in which every activity variant occurs exactly once (see Section 5.2.1). A second difference is that the alarms and activity variants represent classes and therefore cannot be directly compared with their real-world counterparts.

The measurements produced with using RIDAX are shown in Table 6 and illustrated in Figure 15.

**Table 6—Recall and precision of the IDSes evaluated**

| IDS | Detected attack variants | Not detected attack variants | Recall (coverage) | True positives | False positives | Precision |
|---|---|---|---|---|---|---|
| **DWF** | 42 | 456 | **8.4%** | 42 | 0 | **100.0%** |
| **DWH** | 112 | 386 | **22.5%** | 112 | 0 | **100.0%** |
| **SNF** | 72 | 426 | **14.5%** | 277 | 94 | **74.7%** |
| **SNS** | 44 | 454 | **8.8%** | 242 | 92 | **72.5%** |
| **WI** | 79 | 419 | **15.9%** | 164 | 15 | **91.6%** |

The first fact one notes is that DaemonWatcher for HTTP (DWH) not only provides the highest recall but also 100% precision. This raises two questions: How does this come? And, why are the results of DaemonWatcher for FTP different? The main answer to this lies in the environment, i.e. the activities and activity variants used for this evaluation (details can be found in [Alessandri 2002]):

- We have defined nine malicious activities for HTTP, but only four for FTP
- We have defined one variation (hexadecimal encoding of the URL) that can only be applied to HTTP-related activities. Such an additional variation can significantly increase the number of activity variants that can be derived from an activity.
- Because of the detection method used, DaemonWatcher is *not* susceptible to any of the variations considered, i.e. in the context of this work we were not able to identify a meaningful variation that could be used to elude DaemonWatcher.

The items identified illustrate that even our approach suffers some bias caused by the selection of the input data used for the evaluation. However, this impact is limited and well understood. On the other hand, it seems fair to give more weight to HTTP activities owing to the popularity of HTTP and its nature, which permits variations that are not possible in other protocols. For future work it seems advisable to investigate approaches that would permit the results to be weighted on a per-activity and variation basis.

Moreover, DaemonWatcher has the inherent advantages that it does not have to repeat the error-prone network-stack processing and that it does not have to predict how the application monitored might interpret the observed activity.

**Example:** *A simple network-layer variation that fragments IP PDUs will not prevent DaemonWatcher or WebIDS from detecting the actual attack because the host's network stack will reassemble the fragments before the data is passed on to the daemon process. For the same reason it is impossible for these IDSes to detect the variation itself.*

Note that the measurements obtained do not reflect whether the IDSes were able to provide any further information besides the fact that a number of attack variants was detected. Moreover, they do not reflect whether the success state of the attack variants considered is reported by the IDSes. DaemonWatcher, for instance, will only report attacks that were successful, whereas Snort generally cannot provide this information.

Considering Figure 15, we see that Snort has the lowest precision, followed by WebIDS and DaemonWatcher. These differences illustrate the increase in inherent difficulties the lower the level at which the IDS sensor operates is. IDSes that operate based on network PDUs have to make assumptions about how the PDUs will be treated by the target

system. Based on these assumptions, they have to move up further levels of abstraction, trying to predict how the observed data PDUs or data sequences will be interpreted. This excessive crossing of abstraction levels typically results in relatively generic signatures that may not only be evaded, but may also cause false positives. Similar conclusions were drawn by Sekar *et al.* [Sekar *et al.* 1999]. As a consequence their implementation of a network-based IDS only focuses on attacks up to the transport layer, and rarely considers application-layer attacks.

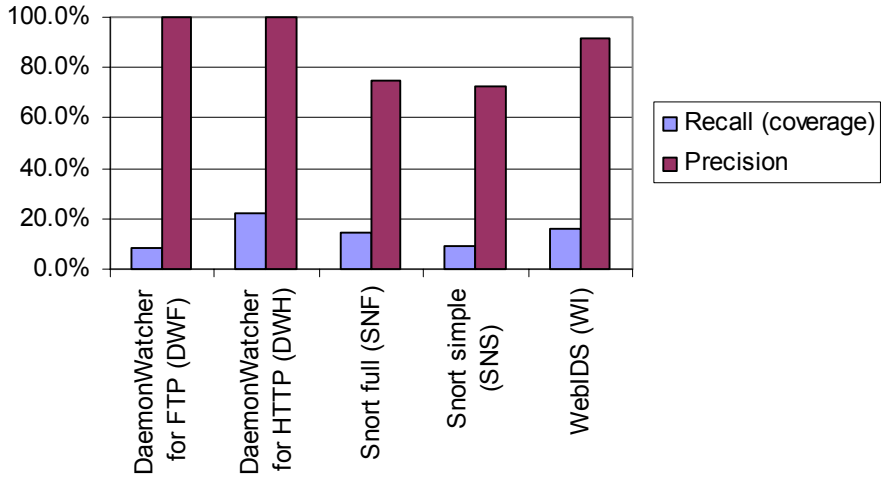



**Figure 15—Chart representing precision and recall**

4.5.2.3 Examples of alarm-set-based fault diagnosis

Especially for large-scale ID architectures, the meaningful processing of alarms is vital. One important factor is certainly the handling of the vast number of false positives one can expect. However, false positives are only part of the problem. With increasing size of the ID architecture, also the difficulty of interpreting the symptoms of a cause that may be malicious increases significantly. Worse, these two issues are closely related and therefore should not be separated. This resulted in the alarm-set-based approach to fault diagnosis developed in Section 4.4.3.

Depending on the number of IDSes combined, we were able to identify up to 72 unique alarm sets. However, when considering individual IDSes such as DaemonWatcher this number may be as small as one. Each of the identified alarm sets may be caused by one or several activities. In the following, we discuss some examples of alarm sets as they were generated in the context of the RIDAX experiments.

We consider the HTTP-argument buffer overflow attack we have already used several times as an example. In our experiments RIDAX was capable of considering 28 activity variants of this attack.

In DaemonWatcher (DWH) each of these 28 activity variants is being reported by an alarm indicating that the execution path of the process implementing the HTTP services is diverting. However, as one might imagine, other activities can have the same effect and are reported by the same alarms. Based on DaemonWatcher alarms only, it is therefore not possible to clearly identify the activity, i.e. the cause, of the alarm. Other IDSes, such as Snort or WebIDS, generate alarms indicating the observation of a suspicious or overly long argument string. However, these alarms do not permit aclear

identification of the activity either because they may also report other, possibly benign, activities. Moreover, in many cases, variations render the activity invisible to these IDSes. In our experiments we found IDS combinations that would report activity variants of this activity with up to eleven different alarm sets. However, it is not our goal to maximize this number. It should be the goal to find solutions that maximize the number of activity variants that, based on alarm sets, can be associated uniquely to the generating activity. In Table 7 we provide an overview of the relevant figures for a selection of IDSes and IDS combinations.

**Table 7—Fault diagnosis results for HTTP argument buffer overflow**

| IDS combination | Number of alarm sets identified for the HTTP-argument buffer overflow activity variants | Number of alarm sets that uniquely identify activity 1 | Number of alarm sets that identify only malicious activities | Number of activity variants that can be associated uniquely to activity 1 | Number of undetected activity variants |
|---|---|---|---|---|---|
| DWH | 1 | 0 | 1 | 0 | 0 |
| SNF | 6 | 0 | 0 | 0 | 18 |
| WI | 3 | 2 | 2 | 21 | 6 |
| DWH, SNF | 7 | 0 | 7 | 0 | 0 |
| DWH, WI | 4 | 2 | 4 | 21 | 0 |
| SNF, WI | 10 | 8 | 8 | 21 | 4 |
| DWH, SNF, WI | 11 | 8 | 11 | 21 | 0 |

In the above table we also included the number of undetected activity variants and the number of alarm sets caused only by malicious activities. These numbers become important in the next section as we assess the utility and the coverage of IDS combinations. As explained in Section 4.4.3.2, an important factor is the number of alarm sets that may denote malicious as well as benign activities. Remember, we do not consider false positives to be problematic as long as they can be recognized as such. On the other hand, false positives are particularly annoying if the attack-similar but benign activity causing them occurs. It is difficult to eliminate them in a generic manner, because they may not be distinguishable from alarm sets reporting malicious activity.

#### 4.5.2.4 Measuring the results of alarm-set-based fault diagnosis

In this section we discuss the measurements made while performing fault diagnosis based on IDS evaluation results. The fault diagnosis and measurements are made as introduced in Section 4.4.3. In contrast to the measurements determined in Section 4.5.2.3, those determined here provide us with indications on the utility of the information provided by alarms. The following discussion is primarily on the metrics defined in Section 4.4.4. It not only considers the results provided by the five individual IDS configurations, but also those provided by all possible combinations thereof.

We start with a table that shows the measurements determined for all possible IDS combinations while performing fault diagnosis as described in Section 4.4.3. In contrast to the measurements discussed in Section 4.5.2.2, this analysis includes alarms reporting variations. In a next step we discuss bar charts and a Venn diagram illustrating the results. Then, in a third step, we discuss plots illustrating how different metrics may influence each other. Finally we discuss the impact and use of alarms reporting

variations, and illustrate their impact by comparing fault-diagnosis results that include them with results where they are excluded.

The measurements shown in Table 8 are ordered by attack recall and by rating ambiguity. Intuitively one wishes to maximize attack recall, i.e. coverage, while minimizing ambiguity. As shown in Table 8 and illustrated in Figure 16, most IDS combinations result in a higher coverage than individual IDSes or smaller sets of combined IDSes do. However, in most cases the rating ambiguity seems to increase or at least to be relatively high. Considering Figure 16, we can identify Snort as having a negative impact on the rating ambiguity. On the other hand Snort generally increases attack recall quite significantly.

**Table 8—Measurements resulting from alarm-set-based fault diagnosis (including variation alarms)**

| IDS combination | Attack recall (coverage) | Rating ambiguity | Rating precision | Attack identification recall | Attack identification precision |
|---|---|---|---|---|---|
| DWF, DWH, SNF, SNS, WI | 55.6% | 19.2% | 57.6% | 18.5% | 33.2% |
| DWF, DWH, SNF, WI | 55.6% | 19.4% | 57.1% | 18.5% | 33.2% |
| DWF, DWH, SNS, WI | 50.0% | 19.2% | 54.4% | 12.9% | 25.7% |
| DWH, SNF, SNS, WI | 49.2% | 20.3% | 51.5% | 18.5% | 37.6% |
| DWH, SNF, WI | 49.2% | 20.5% | 51.0% | 18.5% | 37.6% |
| DWF, DWH, SNF, SNS | 47.8% | 16.6% | 57.7% | 5.6% | 11.8% |
| DWF, DWH, SNF | 47.8% | 16.8% | 57.1% | 5.6% | 11.8% |
| DWF, SNF, SNS, WI | 46.4% | 24.0% | 40.4% | 15.5% | 33.3% |
| DWF, SNF, WI | 46.4% | 24.2% | 39.8% | 15.5% | 33.3% |
| DWH, SNS, WI | 43.6% | 20.3% | 47.5% | 12.9% | 29.5% |
| DWF, DWH, SNS | 42.2% | 16.6% | 53.9% | 0.0% | 0.0% |
| DWF, DWH, WI | 41.4% | 7.4% | 72.5% | 12.9% | 31.1% |
| DWH, SNF, SNS | 41.4% | 17.7% | 50.6% | 5.6% | 13.6% |
| DWH, SNF | 41.4% | 17.9% | 50.0% | 5.6% | 13.6% |
| DWF, SNS, WI | 40.8% | 24.0% | 35.2% | 9.8% | 24.1% |
| SNF, SNS, WI | 40.0% | 25.1% | 31.9% | 15.5% | 38.7% |
| SNF, WI | 40.0% | 25.3% | 31.3% | 15.5% | 38.7% |
| DWH, SNS | 35.7% | 17.7% | 45.7% | 0.0% | 0.0% |
| SNS, WI | 34.3% | 25.1% | 25.3% | 9.8% | 28.7% |
| DWF, SNF, SNS | 33.3% | 20.9% | 33.6% | 5.6% | 16.9% |
| DWF, SNF | 33.3% | 21.1% | 32.9% | 5.6% | 16.9% |
| DWH, WI | 32.9% | 7.4% | 67.0% | 12.9% | 39.0% |
| DWF, DWH | 30.9% | 0.0% | 100.0% | 0.0% | 0.0% |
| DWF, WI | 29.5% | 10.9% | 47.4% | 9.8% | 33.3% |
| DWF, SNS | 27.7% | 20.9% | 26.0% | 0.0% | 0.0% |
| SNF, SNS | 26.9% | 22.0% | 21.5% | 5.6% | 20.9% |
| SNF | 26.9% | 22.2% | 20.8% | 5.6% | 20.9% |
| DWH | 22.5% | 0.0% | 100.0% | 0.0% | 0.0% |
| SNS | 21.3% | 22.0% | 11.3% | 0.0% | 0.0% |
| WI | 21.1% | 10.9% | 32.7% | 9.8% | 46.7% |
| DWF | 8.4% | 0.0% | 100.0% | 0.0% | 0.0% |

Therefore the question arises whether one could combine Snort with other IDSes in such a way that the rating ambiguity decreases. Figure 16 clearly shows that this is possible. However, the improvement does not seem as significant as one might hope. Indeed, in many cases the situation becomes worse. This phenomenon is inherent in the combination of IDSes as illustrated by the following example.

**Example:** *Combining Snort (SNF) with WebIDS (WI) results in relatively high attack recall, but causes the rating ambiguity to increase above 25%. On the other hand combining Snort (SNF) with DaemonWatcher (DWF, DWH) improves the situation from 22.2% down to 16.8% rating ambiguity. Where do these differences come from? Snort and WebIDS partially cover the same attacks, and by combining them one can improve coverage. However, because both systems use similar techniques for detecting an attack, both are susceptible to generating false alarms for similar activities. As a further consequence of using similar techniques, also the semantics of their alarms is similar*[6]*. Because of this it becomes difficult for any fault-diagnosis algorithm to compensate the increased ambiguity caused by the increased number of benign activity variants that may cause alarms. The situation changes significantly when considering the combination of Snort and DaemonWatcher. These IDSes use completely different techniques and information sources for their analysis, and are therefore not susceptible to the same variations. Here it becomes not only possible to increase coverage, but also to compensate weaknesses of the respective IDSes by exploiting the semantic diversity of the alarms they generate.*

Similar observations can be made when considering attack identification recall. However, here the strengths and weaknesses are distributed differently among the IDSes. An IDS that distinguishes well between malicious and benign activity variants is not necessarily as good when it comes to *identifying* the activity causing a given alarm set. An extreme example of this is DaemonWatcher. This IDS distinguishes well between malicious and benign activities. However, as its alarms do not carry semantics beyond the fact that an unusual sequence of system calls was observed, it becomes impossible to determine the cause of such an observation. On the other hand, we are able demonstrate that such systems may significantly increase the expressiveness, i.e. attack identification recall, of IDS combinations.

**Example:** *Considering Figure 16, it is apparent that the combination of WebIDS (WI) and DaemonWatcher for HTTP (DWH) results in an significant increase of the attack identification recall compared with any of the corresponding stand-alone configurations.*

[6] Using dependability terms, this indicates a high probability for the existence of common failure-modes.

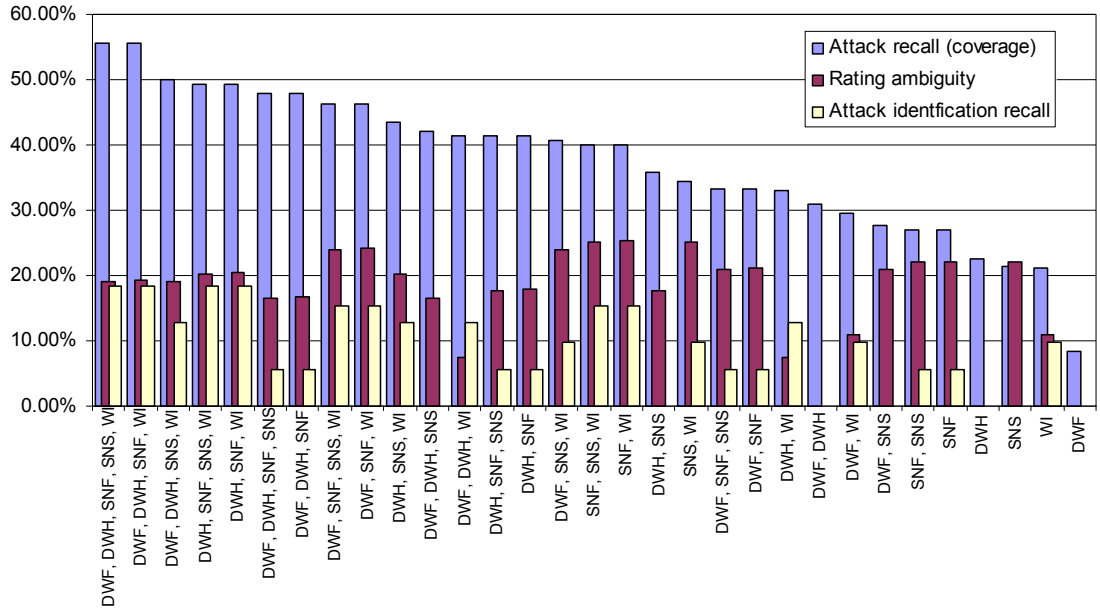



**Figure 16—Attack recall, rating ambiguity and attack identification recall**

Considering attack recall, i.e. coverage, as shown in Figure 16, we already noted that combining IDSes results in increased coverage in most cases. In some cases coverage increases significantly, and in others the increase is less important. The reason for this is quite simple: some IDSes overlap significantly in terms of coverage, and some not at all. Figure 17 illustrates how the five IDS configurations evaluated overlap.

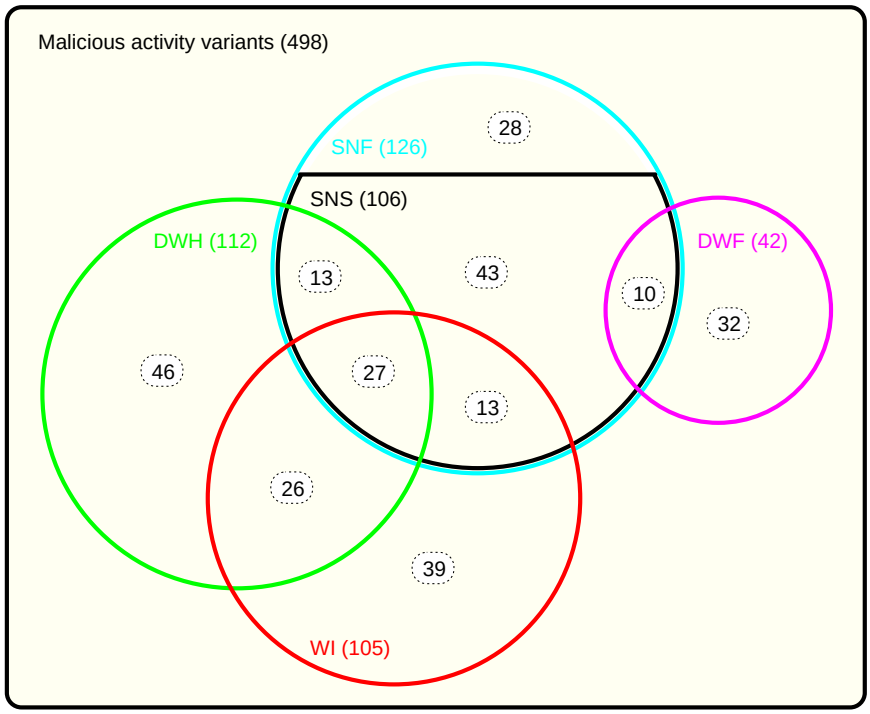



**Figure 17—Venn-diagram showing coverage overlaps of the IDSes evaluated**

When seeking meaningful IDS combinations, it is important to know how the individual IDSes overlap in terms of coverage, in order to determine the coverage provided by a set of IDSes. However, it is even more important to be aware of coverage overlaps when seeking ways to reduce ambiguity and/or increase attack identification recall. This was illustrated in the above example, where we considered the combination of WebIDS and DaemonWatcher for HTTP. For this combination, the gains in terms of coverage are not very important. However, the example clearly illustrates how the diverse semantics of the IDSes' alarms lead to improved ambiguity and attack identification recall. This can be further illustrated in plots such as those shown in Figure 18 and Figure 19. These plots emphasize the fact that not every combination of IDSes leads to improved usability in terms of rating ambiguity or attack identification recall. In Figure 18, for instance, we can identify three bands of IDS combinations. A first band on the x-axis, a second around 10% rating ambiguity, and third around 20% rating ambiguity. Taking a closer look at the data provided in Table 8, we recognize the first band to consist of just the two DaemonWatcher configurations and their combination. The band at the 10% level is defined by WebIDS, and represents the various combinations of WebIDS with DaemonWatcher configurations. The band at 20% is dominated by Snort. Every IDS combination at this level contains at least one of the two Snort configurations.

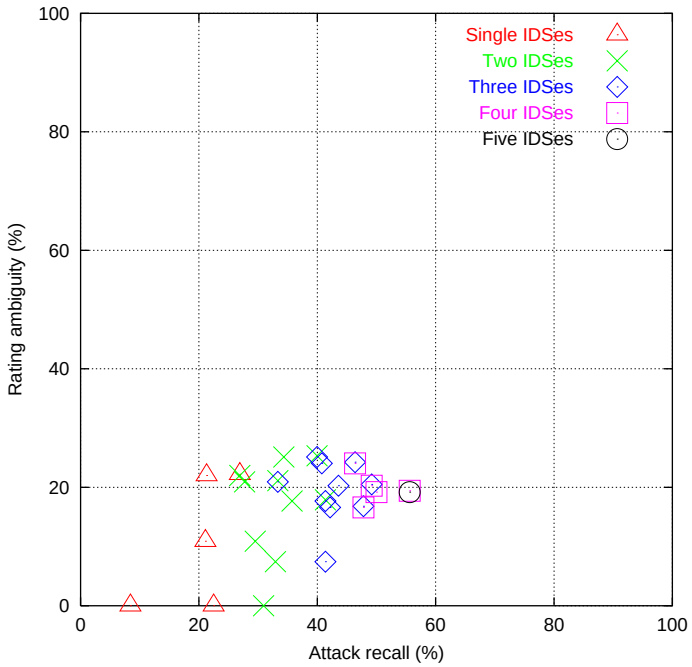



**Figure 18—Attack recall vs. rating ambiguity of IDS combinations**

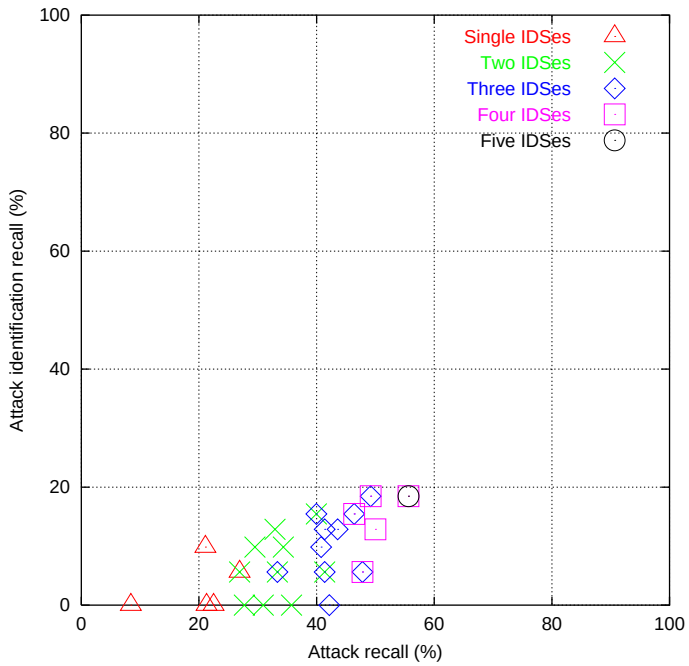



**Figure 19—Attack recall vs. attack identification recall of IDS combinations**

In combination with Table 8 these plots may also be used to investigate the impact of IDS crash failures. Assuming that one out of three combined IDSes fails, these results enable us to assess the degradation in terms of coverage to be expected etc.

So far we considered the *absolute* "recall" measurements only. To complete this discussion we compare attack recall with the *relative* "precision" measurements rating precision and attack identification precision. These measurements are illustrated in the bar chart shown in Figure 20. Also these measurements reflect the inherent strengths and weaknesses of IDSes, such as the fact that DaemonWatcher is excellent at distinguishing between malicious and benign activity variants, but completely fails to identify the activities that cause a given alarm set.

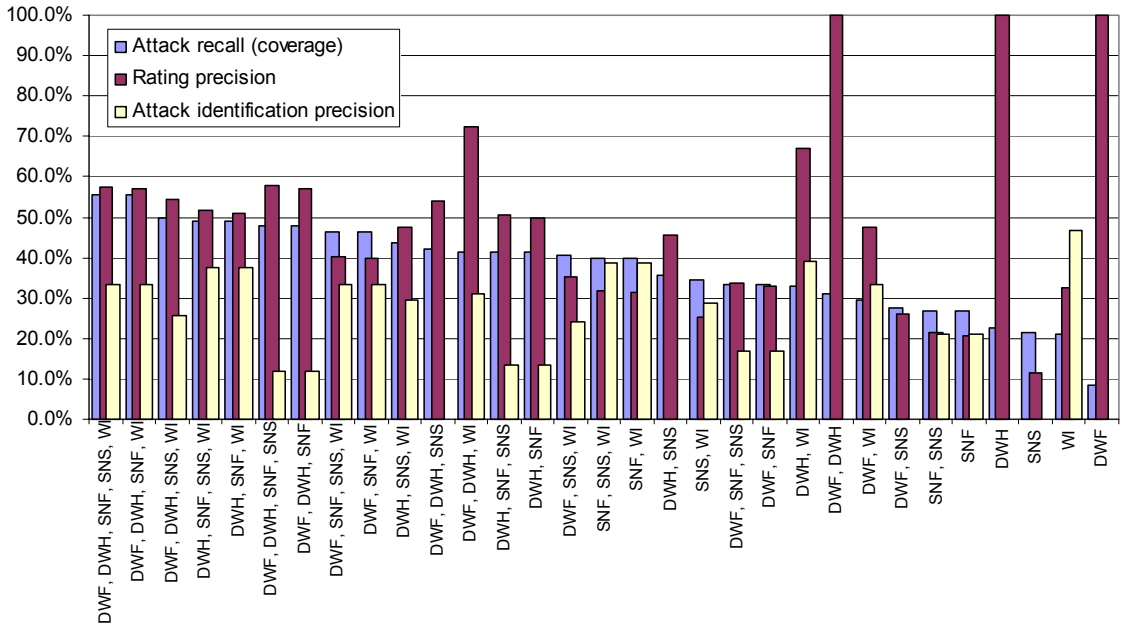



**Figure 20—Attack recall, rating precision and attack identification precision**

4.5.2.5 Use and impact of alarms reporting variations applied to activities

While performing these experiments it became apparent that alarms reporting the observation of variations are of limited use. In fact we felt that they would create misleading measurements for attack recall. To further investigate their influence, we repeated our experiments—disregarding alarms reporting variations.

Our analysis (see Figure 21) revealed that doing so caused both the attack recall and the rating ambiguity to decrease (measurements for rating and attack identification precision are affected even more significantly because they rely on relative metrics). The attack identification recall is not affected by this measure. At first glance the fact that the attack recall decreases appears as a disadvantage. However, if an attack is reported only by means of an alarm indicating the presence of a variation, a suitable reaction to it is difficult. These alarms are highly ambiguous as, in practice, the majority of them report nonmalicious activities. Second, assuming the presence of malicious activity, these alarms hardly provide any useful information about the attack observed. In Figure 21 the generally observable drop of the rating ambiguity nicely illustrates these facts. In most cases the drop in rating ambiguity is far more important than the loss of coverage in terms of attack recall.

As a result we concluded that alarms reporting the presence of variations should be used for adjusting the *severity* one associates with the potential cause of a given alarm set. The fact that an adversary obfuscates its attacks might provide an indication of the tools used and/or the adversary's skills.

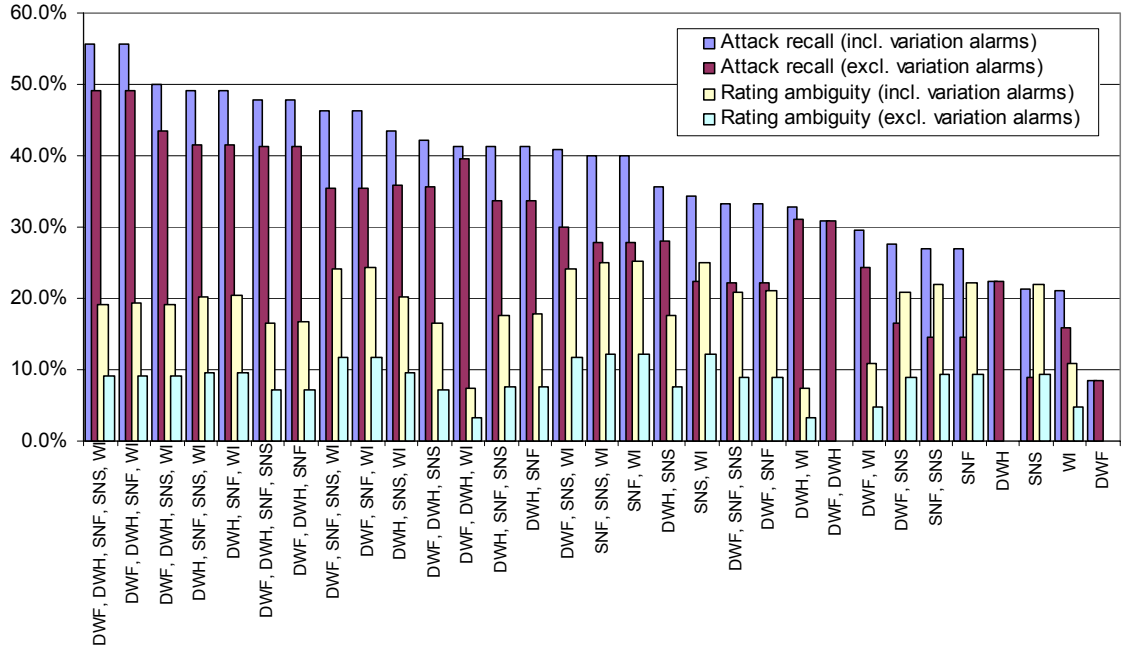



**Figure 21—Attack recall and rating ambiguity including vs. excluding alarms reporting variations**

#### 4.5.2.6 Discussion

Roughly speaking we were able to identify three classes of results. First we were able to reproduce precision and recall measurements for individual IDSes. Second, we were able to identify inter-class relationships between activities and alarm sets, which may serve as the foundation for future work on alarm processing towards root-cause analysis. Finally, we were able to exercise the metrics of attack recall, rating ambiguity etc., which we defined for our alarm-set-based approach to fault diagnosis. These measurements enable an assessment of the potential viability of individual IDSes and, more importantly, of IDS combinations. All these results were obtained using RIDAX to evaluate the 928 activity variants derived in an automated fashion from the 48 activities identified in Deliverable D3. We created these variants by selecting up to two variations out of a set of seven variations described. Of the 928 resulting activity variants, 498 are to be considered malicious.

An important result is the inter-class relationships between activities and alarm sets that we were able to identify. Depending on the number of IDSes combined, we were able to identify up to 72 unique alarm sets. When considering individual IDSes, such as DaemonWatcher, this number is as small as one. Any given alarm set identifies one or several activities that may cause the generation of the alarm set considered. Recalling that all activities and alarms considered in this work represent classes of real-world activities and alarms, our results describe relationships between the two classes. The knowledge of these inter-class relationships will support the development of future alarm-processing algorithms, such as required in future large-scale ID architectures consisting of many highly diverse IDSes. The importance of this knowledge is likely to increase because existing solutions such as IBM's Tivoli Risk Manager [Tivoli 2000] gradually improve their alarm processing towards so-called *root cause analysis* [Julisch 2001b], [Klinger *et al.* 1995], [Paradies and Busch 1988]. As our work proposes an

approach to develop this knowledge, it is hoped that it represents the foundation for future research on this topic.

In addition, in Section 4.5.2.2, we briefly considered detection rates of individual IDSes that were determined in way similar to what is done in other approaches. We chose not develop these measurements much further, as they do not seem well suited to our approach for three reasons:

1. Their underlying metrics cannot be extended to incorporate results from multiple IDSes in a meaningful manner.
2. The measurements obtained do not permit judging the utility of the IDSes in terms of the semantics of the alarms they generate. This specifically includes the ability to distinguish false positives from true positives and the identification of the alarm cause, i.e. the activity.
3. As a consequence the metrics used only seem to provide meaningful information when applied to a real IDS implementation, which is evaluated in the environment for which it is envisaged. Because our approach operates at the more conceptual level of classes and does not include the modeling of an environment, the measurements obtained must be interpreted with care.

Therefore we focused on alarm-set-based fault diagnosis instead. Again one needs to keep in mind that we do not consider a real-world environment, but rather a normalized environment, in which every activity variant is considered exactly once. Identifying the inter-class relationships does not require a model of a real environment. Also, for assessing the viability of IDS combinations a normalized environment proved sufficient. While performing the alarm-set-based fault diagnosis and measuring its results, we were able to validate and quantify a number of common assumptions and to learn numerous lessons:

- Substantial gains in coverage, i.e. attack recall, result from the combination of IDSes with as distinct a coverage as possible (see also Figure 17).
- Improving the rating ambiguity by combining IDSes requires a significant overlap in coverage.
- The techniques used by the combined IDSes should be as diverse as possible.
- It is possible to slightly improve the rating ambiguity while combining IDSes to improve coverage, but this proved to be a difficult task requiring special care.
- Similar observations as made for rating ambiguity measurements apply to attack identification recall measurements.
- Alarms indicating the use of variation techniques increase both coverage and rating ambiguity. One should not consider attacks as detected if they are only reported by an alarm indicating the use of variation techniques. In other words, such alarms should not be included in attack recall calculations because in practice this leads to misleading measurements of coverage.
- It might be meaningful to use alarms that report the use of variation techniques to adjust the severity level associated with a finding presented to the human security officer on duty. Also such alarms may be used to facilitate the identification of the attack tool used by the adversary.
- We were able to validate the common perception [Sekar *et al.* 1999] that IDSes should not cross too many levels of abstraction, e.g. protocol layers, while performing their analysis. Specifically, although generally considered very

convenient to implement, systems applying any of kind string-pattern-matching algorithm to (external) raw data sources are prone to false positives and obfuscation. Furthermore the severity of this issue clearly increases the more abstraction levels an IDS attempts to monitor. An example is the class of network-based systems, which in general are particularly susceptible to false positives because of string mismatches and obfuscation techniques.

In our experiments we always considered the total number of malicious activity variants identified—assuming a normalized environment. In future work one might wish to expand this by weighting the activities and variations according to the coverage required by the security policy and by their importance in the corresponding environments. For instance one might wish to maximize coverage for HTTP-related activities.

When taking into account the environment one should ideally use a model of the environment an individual IDS or the complete ID architecture is envisaged for. If this is not possible, one might consider the creation of multiple environment profiles, each describing a particular class of environments. Such profiles might include descriptions of typical commercial DMZs, Microsoft Windows-dominated intranets, Unix-based environments, etc. At the stage where one weights the evaluation results, one might even choose to combine multiple environment descriptions if appropriate and necessary. One might also want to assess the expressiveness of the alarms generated. In other words, one should not focus on the pure numbers of false and true positives but instead assess how well attacks can be *identified*, and how well false and true positives can be *distinguished*.

Lastly we note that our experiments consider worst-case scenarios only. This means that when considering an attack, we always assumed the attack to be successful. In future work one might extend RIDAX towards considering successful and unsuccessful instances of each attack. This extension should result in more finely grained results for IDSes such as DaemonWatcher that are only able to detect successful attacks.

#### 4.5.2.7 Conclusion

In this chapter we illustrated various assumptions on potential gains achievable by combining IDSes and, for the first time, quantified these gains. More importantly, we were able to identify the class-level relationships between activities and alarm sets, which is hoped to form the foundations for future work moving from “simple” alarm grouping towards root-cause analysis. Moreover, the results obtained correspond to and underline general concepts and issues known in the dependability field. The most important aspect is probably the readily understandable fact that only the combination of sufficiently diverse systems, i.e. IDSes, results in an improved higher-level system, i.e., ID architecture.

We were able to show that a mere counting of false and true positives, as done in typical IDS benchmarks, is insufficient when assessing IDSes—especially when attempting to assess the expectable completeness and utility of IDS combinations. In fact the results obtained in this way may be misleading and significantly biased by the environment used for the benchmarks

It is also clear that when extending our considerations to real IDS implementations care needs to be exercised. Our considerations are made at the conceptual level of activity and alarm classes, and therefore mostly reflect the potential of IDSes and their combinations rather than the behavior of real systems. However, given the conceptual nature of this work, its results seem well suited for supporting the design of an ID

architecture and the development of alarm-diagnosis algorithms that aim towards root-cause analysis.

In the next chapter, we describe a prototype, called *Thor*, which aims at taking advantage of these theoretical notions in order to help designers with the problem of correlating alarms coming from a diversity of sensors.

# Chapter 5 Practical implementation of correlation rules

## 5.1 Motivation

So far, we have considered the problem of the noise generated by huge numbers of false alarms. We have proposed a solution to handle this. Next, we have described a method to assess the best combination of IDSes with respect to certain kinds of fault assumptions. This last result, however, uses models of IDSes rather than their real implementations. We need to find a practical method to verify that real implementations of IDSes will indeed generate the alarms expected when looking at certain activities. Furthermore, we need to find a way to deal with the lack of consistency in the alarms generated by various sensors. Indeed, different sensors use different formats for the alarms they generate. Even worse, there is no common naming scheme for attacks used among the IDSes. In the following, we propose Thor as a first practical approach to deal with these issues.

In order to illustrate our goal let us imagine an environment in which we deploy different ID sensors. All of them gather data and send events to a central console. The console is now confronted with the task of correlating the various alarms with each other and producing a condensed view of what is really happening in the network.

**Table 9—Correlation of alarms**

| Attack | IDS A | IDS B |
|---|---|---|
| PHF | *HTTP_PHF* | *HTTP_GET* |
| PHF, layer-three fragments | *HTTP_PHF* | *IP_FRAG* |

Table 9 is a very simple example of a correlation table. It shows which alarms two different IDSes produce when they are presented with a certain attack. If we are able to provide such a table to the correlation engine, the latter will now be capable of differentiating between the two attacks. If it did not have the table, it would have no chance of telling which sensor generated the more precise result and which attack was actually occurring. The need for this correlation table and the fact that currently there is no method to generate it were the initiator for this work.

### 5.1.1 Thor

From the Prose Edda [Sturluson]

*"Balder the Good had some terrible dreams that threatened his life. When he told the Æsir these dreams, they decided to seek protection for Balder from every kind of peril. Frigg exacted an oath from fire and water, iron and all kinds of metals, stones, earth, trees, ailments, beasts, birds, poison, and serpents, that they would not harm Balder. And when this had been done and put to the test, Balder and the Æsir used to amuse themselves by making him stand up at their assemblies for some of them to throw darts at, others to strike and the rest to throw stones at."*

This is the beginning of a tale from the Norse mythology. Let us now look at the rest of the story to understand why we chose the name *Thor*[7] for this prototype: Loki, the god of mischief, disguised as an old woman, visited Frigg and found out Balder was invulnerable to everything but mistletoe. Loki then made a dart out of mistletoe and tricked the blind god Hod, Balder's brother, into throwing it at Balder, thus killing him. It is believed that Thor found out that Loki was behind this assassination. He went after him and punished him.

Making the connection to our project, Balder stands for an ID sensor, which at first sight is invincible. But by finding the right attacks, such ID sensors can be circumvented. Thor stands for our test system. As the ruler of the gods, he plays the role of supervisor of their activities, just as our test environment does. Loki (the evil attacker) successfully finds a vulnerable spot (an attack the IDS does not generate an alarm for) and is thus able to kill Balder.

## 5.2 Related work

Our main goal is not, per se, to test the IDSes but rather to better understand them in order to combine them. However, it is clear that, from a technical point of view, our approach is very similar to the ones used by people interesting in testing IDSes. It is therefore important to review the current state of the art in that field. In the following, we present the three most important projects.

### 5.2.1 1998 and 1999 DARPA Off-line Intrusion Detection Evaluation

Probably the most discussed IDS test environment was built by Lippman *et al.* for the DARPA off-line Intrusion Detection Evaluation [Lippmann *et al.* 2000], [Lippmann *et al.* 2000b]. Their test environment represents a testbed that simulates real user sessions on a network with a fixed (and rather simple) topology. Intermingled with this background traffic, 300 instances of 38 different attacks are launched. This data is first recorded, and then an *off-line* analysis is performed by the IDSes under inspection. A first portion of all data is used to tune the IDSes. The second portion is then used for the actual evaluation. The results were rather disappointing for most systems. For knowledge-based systems, the reasons had to be found in missing signatures and stealthy techniques to run the attacks. The authors strongly emphasize that the false alarm rates should be interpreted in the context of the testbed and background traffic used.

McHugh criticizes the evaluations extensively [McHugh 2000], [McHugh 2000b]. He claims that too many issues were addressed imprecisely or not at all. The following are just the three main criticisms:

- The generation of background traffic was not conducted in a satisfactory manner. McHugh outlines some of the problems associated with generating background traffic and encourages further research in this area. Specifically he recommends that attacks to the IDSes be made under ideal conditions without

[7] *THOR* also stands for *Test device for Human Occupant Restraint*, also known as *crash test dummies*. It is an unintended, but nicely fitting comparison with our test environment.

background traffic so as to separate detection characteristics from the confounding effects of the background traffic.

- The small and limited network restricted the possibility to test the IDSes' capabilities.
- Justification is not given for the selection of attacks, and the mixing of the attacks into the background data was performed "disadvantageously." McHugh mentions, as an example, that no effort was made to distribute the attacks realistically in the background data. The attack classification is also criticized as it does not describe the attack manifestations in a useful manner.

### 5.2.2 LARIAT

The Lincoln adaptable real-time information assurance testbed (LARIAT) [Rossey *et al.* 2001], [Haines *et al.* 2001] is the continuation of the work presented in the above section. It automates many components of the former system but is not yet capable of doing everything automatically. Alarm collection, for example, has to be done manually. More complex topologies, even firewall devices, are now supported. The most important change is that the analysis is no longer conducted off-line. It also works online.

To control the machines in the network (e.g., for launching the attacks), a central machine can communicate with them. The danger is that both the attack and the control traffic use the same network. Thus it is possible that the control traffic causes the IDS to generate false alarms if it is recognized as being malicious.

### 5.2.3 NSS Intrusion Detection System Test Report

Network Security Services (NSS) released a new version of its *Intrusion Detection System Group Report* (Edition 2) in December 2001 [NSS 2001]. Note that the work is a consumer report, not a research project. In the following discussion, we will not address their product reviews, but focus on the performance tests described therein.

To test the IDSes, a set of attacks was launched in a "*...controlled environment that was as close to a real-life network as [they] could make it...*" and the resulting alarms from the IDSes were collected. The authors unfortunately do not define the terms "controlled environment" and "real-life network" precisely. No justification for their choice of attacks is given, and the provided classification of attacks is ambiguous. We mention this work because they have considered using evasion techniques (*variations*) to test the capabilities of the IDSes. They have used the following two freely available tools: *Fragrouter* [Song 1999b] and *whisker* [Puppy]. However, the report does not clearly specify how an IDS is ranked if it detects all variations of a given attack or only some of them. Also, some conclusions are questionable, to say the least. As an example, for *dragon* [Dragon 2001] and *NFR* [nfr], the report states that they demonstrate *"excellent attack recognition capabilities, as well as a complete resistance to all currently known IDS evasion techniques.*" However, to make such definitive statements about an IDS, it is quite important to know which operating system (OS) it runs on. It might well be that the alarms differ for the same IDS if a different OS is used. In the NSS report, no statement can be found about the OS used to run the IDSes.

## 5.3 Approach

*Thor* is a tool that automates the analysis of IDSes. It was created for this project, and has the following four main capabilities: it can automatically (1) launch *attacks*, (2) vary attacks, (3) collect the *alarms* from the IDSes, and (4) generate reports out of the alarms collected.

*Thor* generates *correlation tables* in the form shown in Table 9. It shows which alarms an IDS has generated for each attack. Thor is not limited to a certain set of IDSes. Integrating a new one is easily possible. New rows are added to the table by either running a new attack, or using an existing attack and applying a *variation* to it, as defined in the previous Section. Variations are useful to change the attacks in a way that could make them invisible to IDSes. It might also be that a variation changes an attack such that the IDS reports a false alarm (i.e., no longer identifies the attack correctly).

For a complete description of *Thor*, its design, implementation and results obtained, we refer the interested reader to [Marty 2002]

## 5.4 Specification

### 5.4.1 Network

The network used by *Thor* has to be very generic and comply with the following requirements:

- It needs a way to simulate an arbitrary networking environment for the IDSes.
- The control traffic (i.e., traffic used to set up systems, control the machines or event messages returned from the IDSes) must not influence the attack traffic, as it would distort the view of the IDSes.
- All the tests we are going to run have to be *reproducible*.

Reproducible tests are important, as otherwise we could not generate the correlation table. This is only possible if the different IDSes are confronted with identical conditions. McHugh [McHugh 2000b] also proposes that attacks be made to the IDSes under ideal conditions without background traffic, so as to separate detection characteristics from the effects of the background traffic.

*Background activity*, or *background traffic*, refers to the fact that a network usually includes machines that communicate with each other, but are not directly involved in an attack. This traffic has the potential to generate *false positives* on the ID sensors. However, we have addressed that issue in Chapter 3, and have proposed a solution to discard false positives semi-automatically. Therefore, by decoupling the problems of false positives and false negatives, we can build a simpler testbed, the results of which will be reproducible. By doing so, we address some of the concerns addressed by McHugh about the Lincoln Lab experiment, as explained in Section 5.2.1

### 5.4.2 Attacks

The generation of attacks should not be reinvented, as numerous tools already exist. Therefore we decided that our system should be able

- to integrate sophisticated attack tools such as Nessus [Deraison 2000],
- to use single attacks from any source, and

- to have multiple source machines that can launch the attacks.

### 5.4.3 Targets

Sources and targets of attacks must be easy to change in the testbed. The user might, for example, want a testbed, where the source is an Intel machine running Linux and the target an Intel machine running some version of Windows. In a next test series, he or she might want to invert roles. This task of reinstalling and resetting all the machines should be automated as much as possible.

The detection capability of certain IDS is a function of the vulnerability of the target. In other words, there are IDS that can only detect attacks against vulnerable machines. Therefore, the vulnerability of the target machine must be a tunable parameter in the testbed. Also, we need to know whether a given attack has been successful.

Finally the testbed must support multiple target machines. We would like to launch attacks from various sources to various targets in the same test series.

### 5.4.4 Alarm collection

A major problem when collecting alarms from different IDSes is that each IDS uses a different representation to describe these alarms. It might for example happen that IDS A does not indicate where a certain attack originated from, but IDS B reports a source address. Or some IDS might use the host names for all the systems, whereas another one uses the IP addresses. To address these problems, we need a mechanism that performs some kind of normalization of the alarms, without destroying any information that could be useful later.

### 5.4.5 Intrusion detection system

*Thor* should have the capability to

- test network-based and host-based IDSes,
- install more than one system at a time and report their behavior.

It is possible that an IDS does not support the forwarding of alarms to a remote machine. For *Thor*, this is essential, as it needs the alarms outside of the IDS. Therefore, the IDS to be tested must have a mechanism that forwards the alarm messages to a central alarm collector.

## 5.5 Design

Having outlined the specifications above, we now develop the design of the *Thor* testbed, which consists of various building blocks.

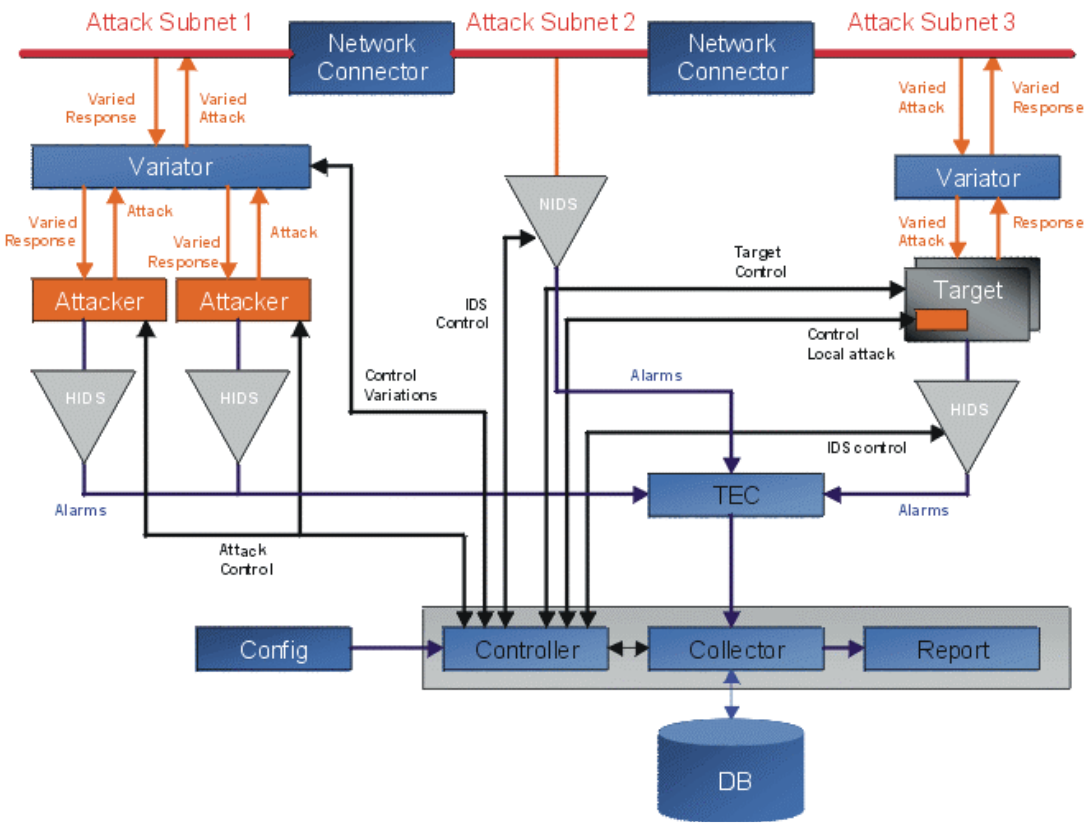



**Figure 22—Component diagram of the Thor testbed**

Figure 22 gives an overview of all the components involved in the system. In the following subsection, we will describe the following elements in greater detail: the *Network, the Attacker, the Target, the Variator, and the Controller.* We refer the reader to [Marty 2002] for a more detailed presentation of all the components.

### 5.5.1 Network

As defined in the specification, we have to support generic network environments. This also means that we need a way to describe the topology, which involves the network layout, the placement of the attackers and targets, and most importantly, the placement of the IDSes. Using a generic way of describing arbitrary topologies would be highly complex. In order to address this issue, we have used so-called *cascading* topologies.

The approach states that if we have a network with two or more IDSes, we can test the network with only one IDS attached at a time. First we run the tests with the only IDS number one attached, and record all the settings and alarms. Then we rerun the exact same tests with the IDS number two attached. Now, because the tests are reproducible, we have results as if both IDSes had been working at the same time. To illustrate the idea a little better, let us look at a slightly more complex network, shown in Figure 23. This topology can be broken up into the simple architecture of Figure 24. After running all the tests, the alarms can be combined as if the two IDSes had been attached at the same time.

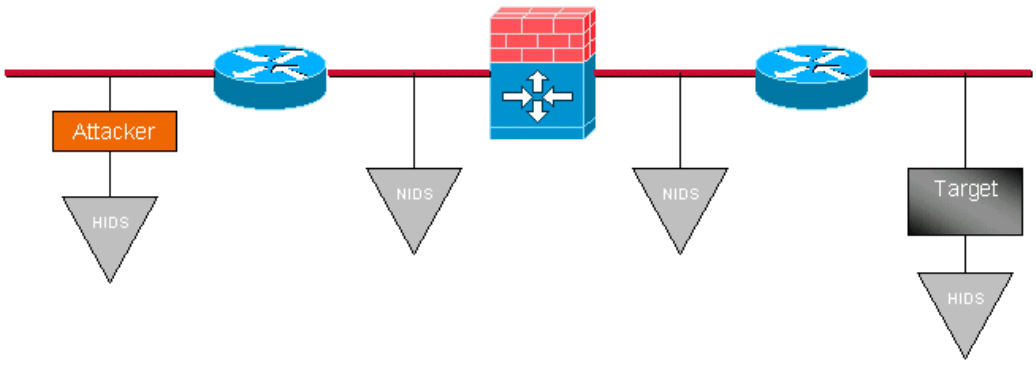



**Figure 23—A complex network structure**

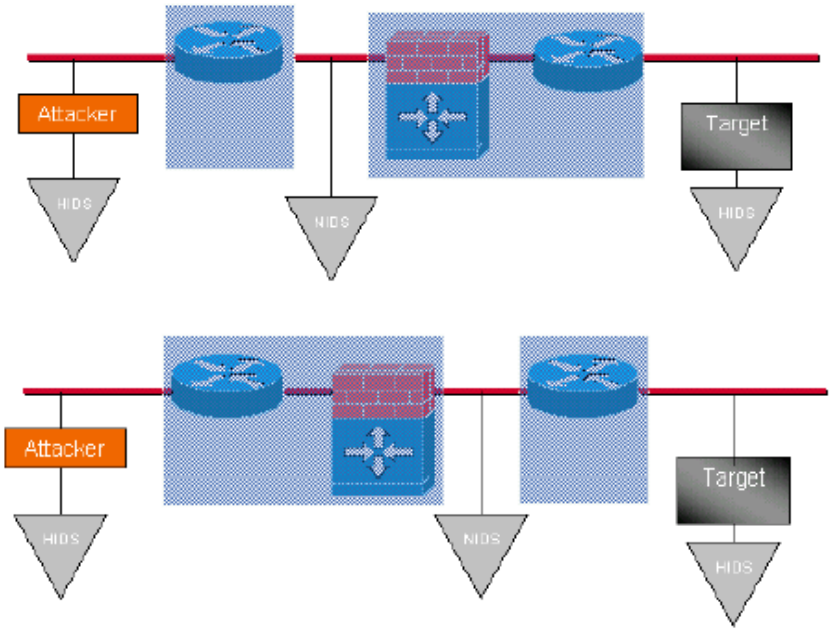



**Figure 24—The network split up into a form that is suitable for our testbed.**

In Figure 25, we show the general setup of *Thor*. As we can see, the *Network Connectors (NC)* hide the network components, such as routers, firewalls, etc. Note that

- an NC can be empty, and that
- the network always has exactly one NIDS, which resides between the two NCs.

This guarantees that the NIDS sees the traffic flowing from the attacker to the target. This is very important, as otherwise the generated alarms would not reflect the capabilities of the IDS. And we even gain some simplicity, as the IDS does not have to be reconnected for every test. It remains statically in its position.

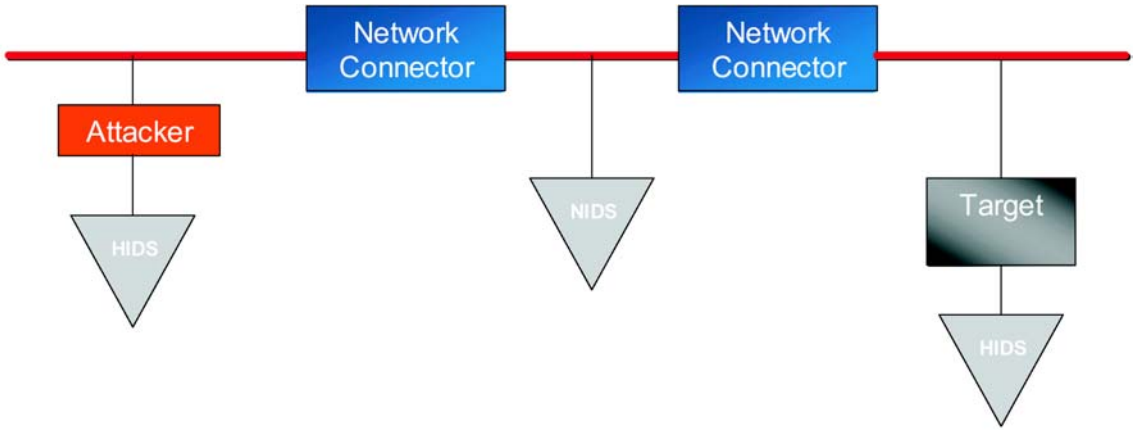



**Figure 25—The generic topology**

Figure 26 depicts a solution for communication between various components. The problem with this approach is that the attack traffic and the control traffic flow on the same network segment, which contradicts the specification requiring the separation of the traffics.

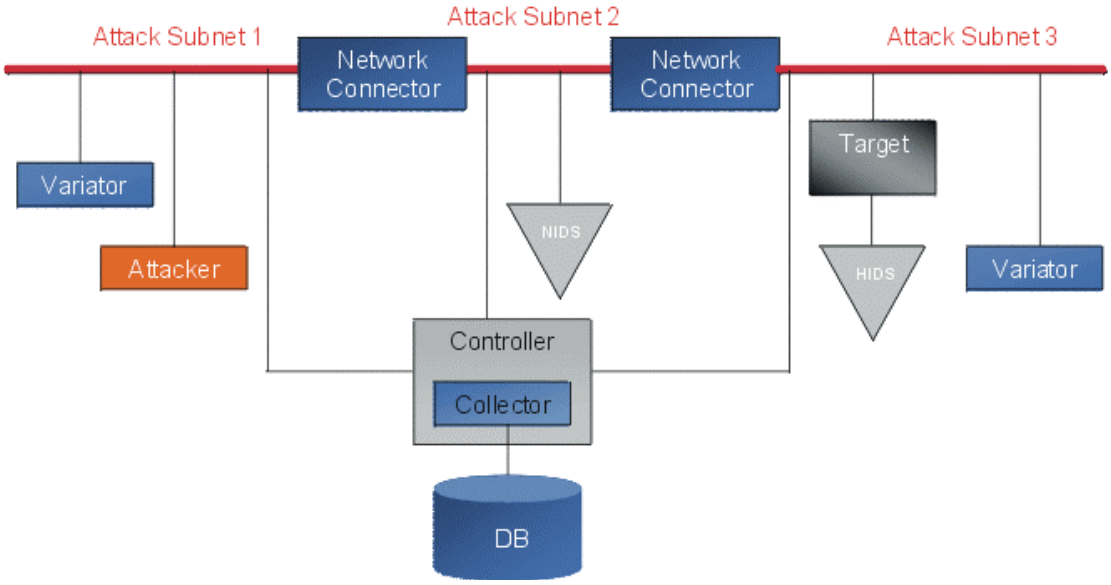



**Figure 26—Shared medium for attack and control traffic**

To eliminate this drawback, we decided to use a different setup, shown in Figure 27. With such a topology, we can guarantee that the control traffic does not affect the attack traffic and, more importantly, the alarm generation of the IDS. In this setup, we use two control networks. This guarantees that the target machine does not get any traffic of the attacker, before it went through the Variators. In this setup, we neglect the fact that the attacker sends its traffic on the same subnet as the control traffic is flowing. This can be done because no IDS, which could be influenced, is attached here.

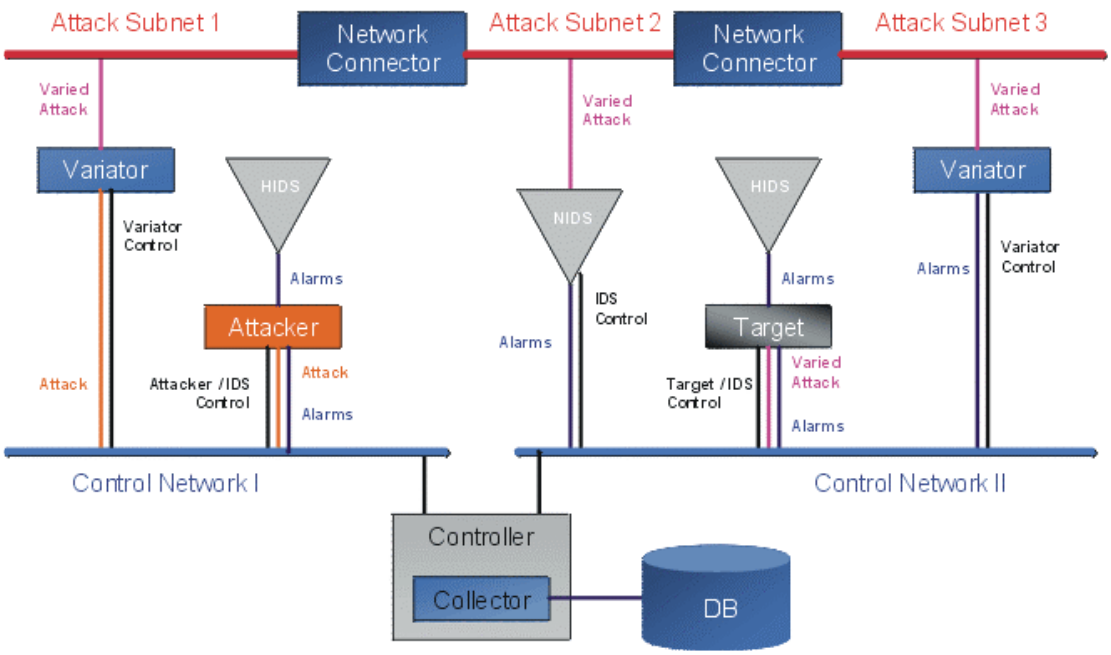



**Figure 27—Separated networks for attack and control traffic**

### 5.5.2 Attacker

The attacker is equipped with the attack-generation tools. The attack machines run a host daemon, as do all other machines, which accepts commands from the controller. These commands will start certain attacks or they could be used to control the machine, for example, to reboot it in case of any error. Certain attacks have the capability of detecting whether the executed attack was successful.

### 5.5.3 Target

For the target machine we need some sort of mechanism that allows us to specifically define which setup we are going to run. In a first step, the operating system needed for the test must be installed on the machine. Then, specific services and configurations are chosen. These pre-attack setups could be handled with either *Internet In a Box* [Whalley *et al.* 2000] or the *Remote Code Browsing* project [Rooney *et al.* 2002]. Both approaches use Vmware [Vmware] to set up the different configurations. Small changes on the machines can be made through scripts launched via the daemon we have running on each machine.

### 5.5.4 Variator

In this section, we address the possibilities of applying variations to the attacks. A first issue is where the variations can be executed to alter the attacks. We have identified three possible locations for doing so:

1. Add the variation directly to the attack, either by changing the attack itself or by putting it in the network stack of the attack machine. This would violate our requirement regarding the orthogonality of the variation.

2. Introduce an additional layer underneath each network layer in the attack machine. Its task would be to add the variations for the network layer above it. This is a very expensive approach and fairly complicated.
3. Use a proxy architecture, in which the attacks are routed through a separate entity that applies the variations.

For obvious reasons, we chose the third option. Using this proxy architecture leaves us some freedom as to where to apply the variations:

- On the *attack* machine.
- In a dedicated proxy, which we call the *Variator*.
- In the *network connectors*.
- In the *target* component.

Applying the variations in the target component does not make sense as the attacker can only achieve this if he or she has already gained control over the machine! By design, we cannot put the variations into the NC because we have no control communication channel.

We are left with the options of applying the variations in the attacker machine or in a separate Variator. Both solutions would be suitable. It is clear, however, that we can provide the same functionalities by using a dedicated variator without having to modify the attacker machine. Indeed, an attack is nothing else than some network packets that leave some machine (the attacker machine). A proxy can take one of these packets, decode it and flip every single bit of the packet, enabling it to do anything one could be doing on the attacker's machine. This the choice we made to implement the notions of variations. As can be seen in Figure 22, the Variator is placed between the attacker and the attack network (or, accordingly, between the attack network and the target machine). This implies that the Variator will act as a gateway for the attacker (or target).

In the setup, we have a total of two Variators. The first one can be used to vary the attack itself, and the second one is here to vary the response. It is not very realistic to assume that an attack towards some system will also result in the response being altered, but it might be interesting to see whether certain IDSes will alert differently if confronted with varied responses.

Using an host-based IDS (HIDS), one is left with placing it on either the source or the target machine. Installing it on the source machine has the impact that the IDS will observe none of the variations, as they are applied only after the attack machine. The same is true when a local attack is executed. Here again we have no possibility to vary the attack, as the attack is executed directly on the target machine. This is of importance when looking at the reports and making statements about the capabilities of the IDSes.

### 5.5.5 Controller

The controller is the component that interacts with all the components in the testbed to control them as well as to gather information as they are up and running. It has the following tasks:

- Setting up the target machines according to the configuration.
- Setting up the source machines.
- Launching the attacks.

- Applying the variations to the attacks.
- Providing interfaces and possible implementations to apply variations to attacks intelligently.
- Communicating with the target machines to read their status and control them (e.g., restarting processes or discovering changed files).
- Communicating with the *Collector* to obtain feedback (alarms) from the IDSes.
- Controlling the IDSes (i.e., providing the possibility to restart them).
- Creating the final reports

Figure 28 shows the building blocks of the controller, all its main components and their communication paths. A detailed description of these elements as well as of the Thor prototype would exceed the scope of this deliverable. Here again, we refer the interested reader to [Marty 2002] for a complete description of the implementation details of this operational testbed. To illustrate its usage we offer some early, and interesting results in the final section of this chapter.

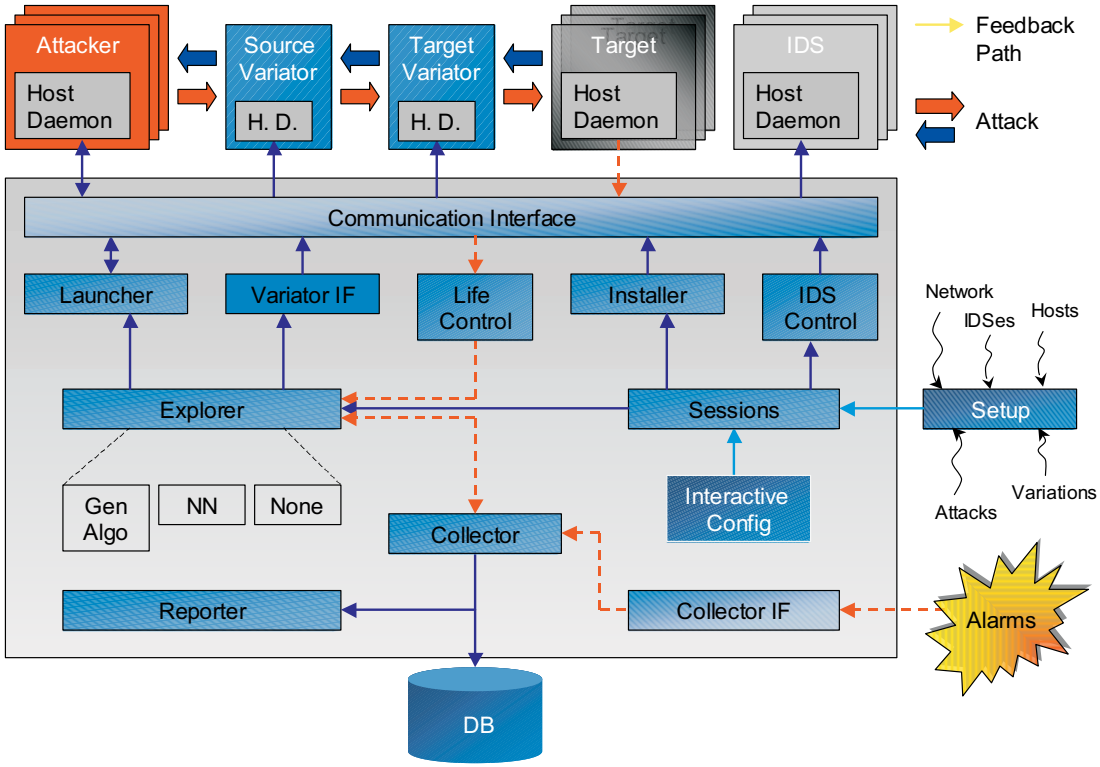



**Figure 28—Design of the controller, its components and their communication paths**

## 5.6 Preliminary results

In this section, we describe only one test scenario for the sole purpose of illustrating the concepts developed in this work.

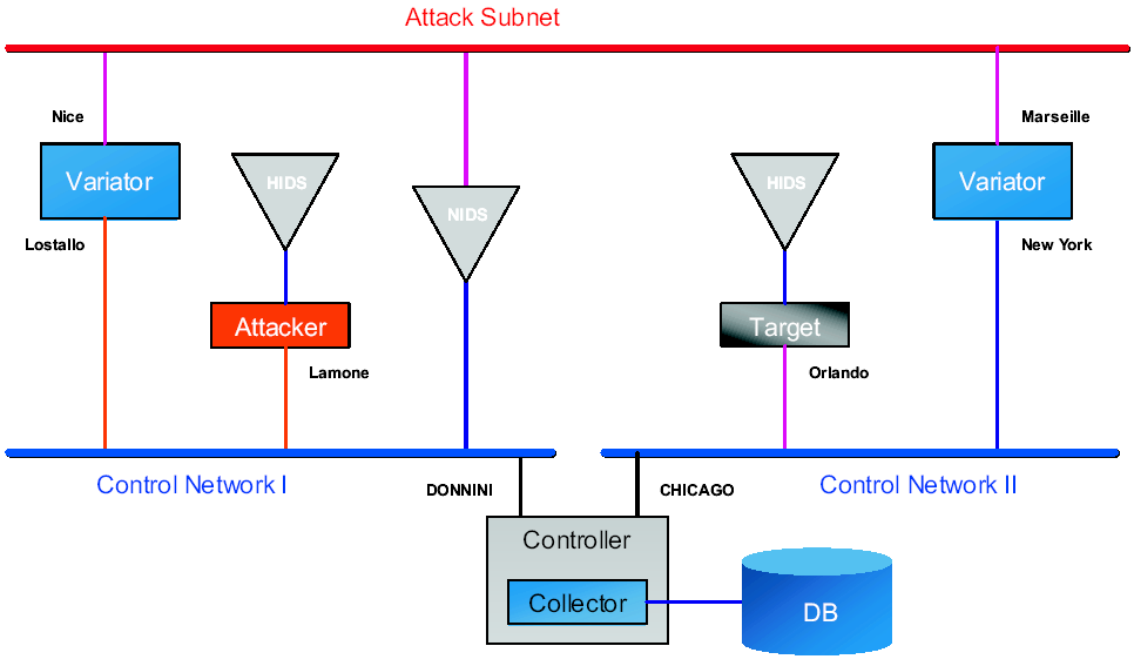



**Figure 29—Topology used for the tests**

The installation used can be described as follows:

- The network topology is shown in Figure 29
- Variators:
  1. RedHat 7.2 standard installation with only the network, messaging and Web tools installed.
  2. Patched kernel enabled to let very small network packets pass.
- Source and Target: RedHat 7.2 standard installation (workstation packages).
- Targets:
  1. Apache 1.3.12 Web server [Apache 2001].
  2. Vulnerable phf-CGI from NCSA Web server 1.5c-export [NCSA 1996].
  3. Purdue Directory Service [Purdue Directory].
- One network-based IDS with the following specification:

  Windows NT 4 Workstation running a well-known commercial product (anonymized for confidentiality reasons).

For the test, the well-known PHF CGI attack [CERT 1996-06] was launched. PHF is a script that allows Web-based queries to the Purdue Directory Service. The CGI attack is known for its ability to abuse the system to execute arbitrary commands on the host machine. This made it possible to download the password file from a target machine. To exploit the vulnerability, the following HTTP request is issued:

*GET target /cgi-bin/phf?QALIAS=x%0a/bin/cat%20/etc/passwd HTTP/1.0*

This, along with a vulnerable phf-CGI, would retrieve the *"/etc/passwd"* file from the target. IDSes should easily detect this attack. However we did not just execute this

attack as is, but introduced IP fragments. This was done by instructing the Variators to use a small MTU[8] to communicate with the target, resulting in fragments on the attack network. These fragments have a size equal to (or smaller than) the MTU. Furthermore, we used *Thor's* capabilities of continuous variations to generate fragments of 20 to 200 bytes. The reason we thought this could be interesting is that some IDSes use a pattern-matching approach to find attacks. If the IDS does not have the capability to reassemble the IP fragments, the pattern never matches, because no single packet will contain the entire signature.

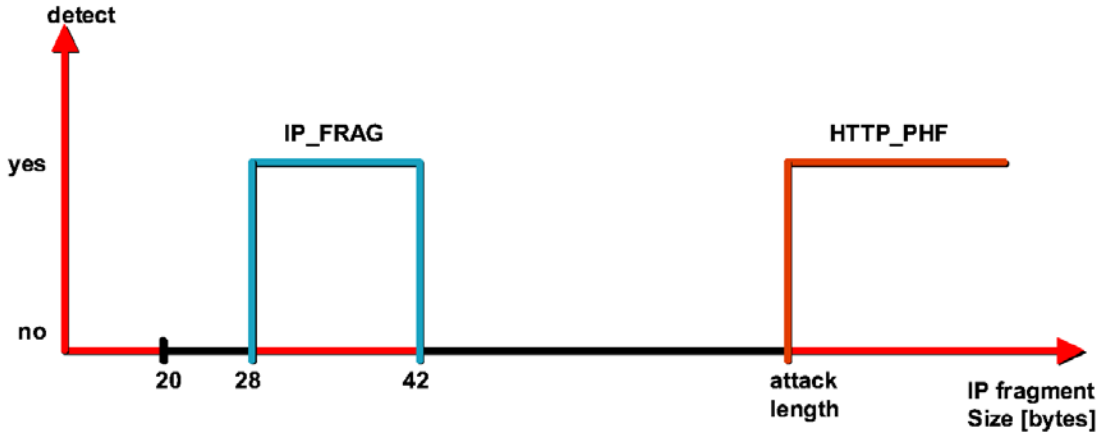



**Figure 30—Alarms generated by the sensor as a function of the IP fragment size**

Figure 30 shows the resulting alarm-generation pattern of the IDS tested. It very nicely shows the boundaries between two distinct alerts messages generated by the very same IDS for the same attack. Table 10 summarizes the observed boundaries and their meaning.

**Table 10—Interpretation of the boundaries shown in Figure 30**

| Fragment Size (in bytes) | Explanation |
|---|---|
| 20 | the standard IP header has a minimum length of 20 bytes. Reducing this size does not make sense, as the IP header would have to be split, which is not possible |
| 28 | IP allows a minimum of 8 bytes for an IP fragment: 20 bytes IP header + 8 bytes IP fragment = 28 bytes |
| 42 | a random value[9] |
| attack length | The length of the attack string in the IP payload |

[8] MTU: Maximum Transmission Unit; it is a parameter of the network interface and specifies the maximum payload size of an Ethernet frame.

[9] 42 also happens to be the answer to the Great Question of Life, the Universe and Everything, as we know from *The Hitchhiker's Guide to the Galaxy* by Douglas Adams.

It is very interesting to find that this sensor does not report anything in the range between 20 and 28 bytes. This is presumably so because IP, according to RFC 791 [RFC791], is not meant to work with fewer than eight bytes of payload. The IDS then detects IP fragments (*IP_FRAG alarm*) up to a size of 42 bytes. As soon as the fragments are greater than 42 bytes, the IDS is blind and no longer detects the attack until the fragments have the size of the attack, which means that no more IP fragments are generated, but the entire request fits in one packet.

# Chapter 6 Intrusion-Tolerant System

## 6.1 Introduction

The previous chapters have proposed methods, processes and tools to address the problem of false positives and false negatives. The purpose of this final chapter is twofold:

- To summarize the results and describe them in terms of design guidelines in a unified model.
- To investigate any remaining weaknesses of the model with respect to threats against the IDS itself.

Section 6.2 offers a brief summary of the generic ID model, based on the MAFTIA concepts developed in [Powell & Stroud 2001]. Section 6.3 provides rationales for this model by summarizing previous results in a bottom-up approach. It highlights intrusion-tolerant features that are offered by this very simple model. Section 6.4 goes one step further by looking at the ID *application* as one specific application among many others that the MAFTIA project is aimed at helping. With this idea in mind, we revisit some of the MAFTIA contributions that we consider to be most appropriate to provide intrusion-tolerant capabilities to the IDS.

## 6.2 Generic intrusion detection model

The following sections offer a brief summary of the ID model defined in MAFTIA deliverable D2 in [Powell & Stroud 2001]. We refer the reader to that document for a complete description of the concepts defined below.

### 6.2.1 High level overview

#### 6.2.1.1 Relationship with the CIDF model

The CIDF model [Porras *et al.*] classifies components of an IDS into four different categories. We recap briefly:

- An *e-box,* or event generator, is a component that gathers event information.
- An *a-box,* or analysis box, analyses event information toward detecting errors and diagnosing faults. The output of an analysis box may provide information to other analysis boxes.
- A *d-box,* or database, provides persistence for the IDS. This facility will take on different forms depending upon use. It may be a complex relational database or it may be a simple text file.
- An *r-box,* or response box, is the portion of the system that acts upon the results of analysis.

Figure 31 presents a refinement of the CIDF model, which explicitly identifies sub-components of the e-box, and the fact that there may be multiple e-, a- and d-boxes.

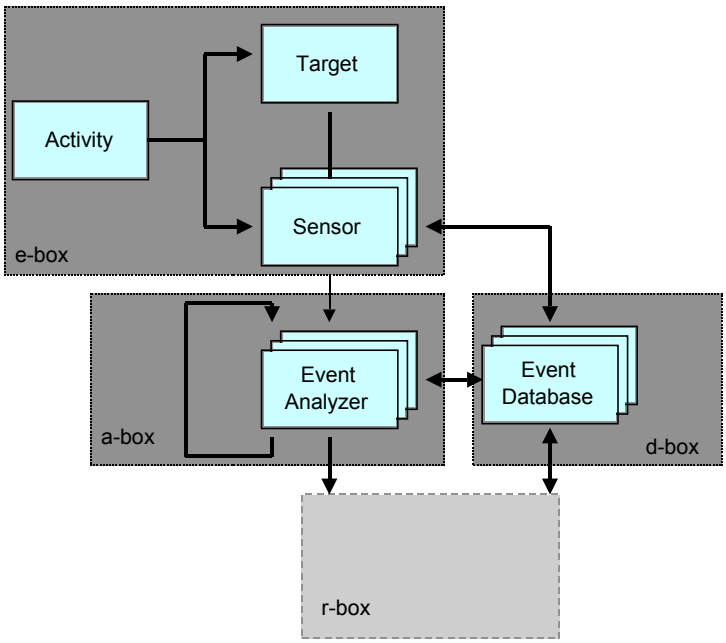



**Figure 31—IDS components**

We note that the decomposition may not correspond to particular physical boundaries. The intention is that the different sensors may generate information stemming from the same root cause, and pass it to a cascading array of analysis components (see Figure 32). In practice most individual IDSes consist of a (small) number of sensors that collect information, which is then analyzed by a (small) number of event analyzers. These generate alarms that may then be forwarded to a central location. In our context these alarms are sent to further event analyzers, also known as *alarm-correlation systems* or *correlation engines*, for further processing (see also Chapter 3 and Figure 32).

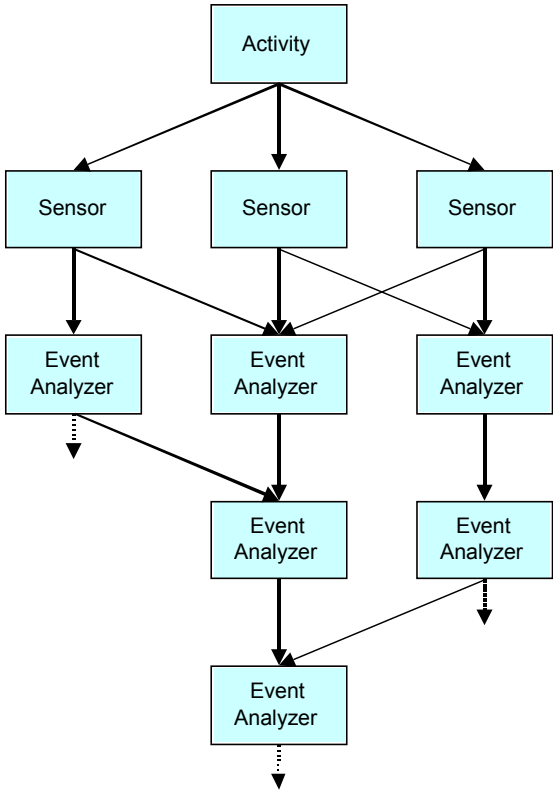



**Figure 32—Cascaded ID topology**

6.2.1.2 Event generator

We subdivide what is termed an *e-box* in the language of the CIDF into three components (*activity, target,* and *sensor*). We have found it necessary to create this subdivision for four reasons:

- To model an *in vivo* system, we need to consider activity in the system.
- To model the real-world of imperfect observation, we need to separate the target of an attack from the sensors used to detect the attack.
- To express several concepts, we need to separate the sensor box from the target.
- To allow several different sensor boxes for a single target.

The role of the sensor is merely to record raw events (no specific ID logic exists in this component). As we have seen, the sensor may very well be imperfect in the sense that it may not sense all raw events of interest.

6.2.1.3 Event analysis

The *event-analysis* boxes successively transform, filter, normalize, and correlate data, adding semantic relevance and reducing volume at each stage. A single event-analysis box may take its input from several different producers (both from sensor boxes and other event-analysis boxes) and may feed its output to several different consumers that are arranged in a arbitrary manner (cf. Figure 32).

6.2.1.4 Event database

The *event database* is implemented to provide persistence for the IDS. This may be for use in off-line error detection, in intrusion analysis, or as evidence justifying response.

This facility has a multi-layered structure similar to that of the entire system. At the lowest level, it may take the form of a simple file. At the highest level, it may be a distributed relational database. We assume that the database may be interactively queried either by the event analysis boxes or by the response boxes that directly require its contents.

## 6.3 Rationales

The above model was defined in [Powell & Stroud 2001] without really providing any strong rationale or explanation. We have now reached a point where we can illustrate this rather theoretical view by means of concrete notions thanks to the results described in the previous sections. In the following, we propose to adopt a bottom-up approach, starting from the simplest instantiation of the model up to its most complex form. In a step-by-step process, we motivate the need for the cascading topology, and we show where the preceding elements of the architecture fit into this model.

Figure 33 describes the simplest case where we have only one sensor and one event analyzer to deal with the entire set of activities considered. A concrete example of such a setup is the sole use of a purely knowledge-based and network-based IDS. In that case, we have only one type of raw data that are being collected by the sensor: the network packets. There is only one event analyzer the signature-matching engine of the IDS.

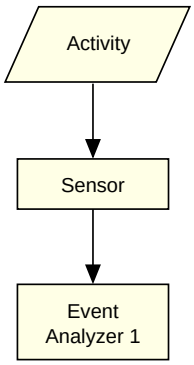



**Figure 33—One sensor and one analyzer**

In Chapter 3, we have discussed at length the risk of generating large numbers of false alarms such a design entails. We have proposed a solution that introduces a new type of event analyzer to automatically discard certain alarms generated by the first event analyzer. Each event analyzer will run specific filtering rules that can be generated thanks to the clustering algorithm described above. This leads us to modify the model represented in Figure 33 to the one shown in Figure 34.

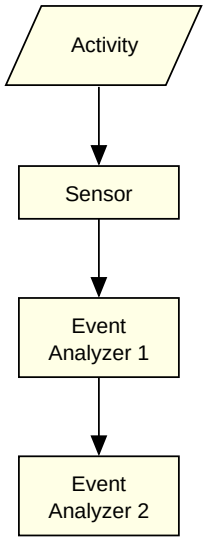



**Figure 34—Second analyzer to handle false positives**

It is worth noting that such a simple setup is already much more efficient than currently deployed solutions. Nevertheless, it is far from being difficult to defeat. Indeed, Chapter 4 and the work described in [Alessandri 2001] have shown that a single sensor will, in most situations, not be sufficient to detect all classes of malicious activities. Indeed, different sensors focus on different views of vulnerabilities and their exploitation. As an example, the detection of a violation of the security policy defined at the application level with a network-based sensor would be computationally infeasible. This leads us to consider IDS systems composed of several sensors working in parallel, as depicted in Figure 35.

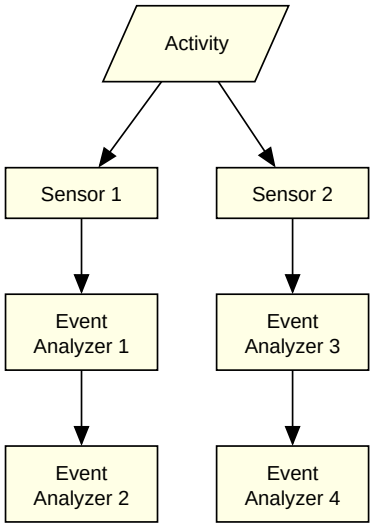



**Figure 35—Second sensor to increase coverage**

Similarly, we have also formally explained the rich variety of event analyzers. We have shown that each of them has various advantages and drawbacks for the detection of certain kinds of malicious activities. We have shown the value of combining more than one event analyzer, and have developed metrics to assess the benefits of doing so. They are two ways to take advantage of the diversity of event analyzers.

- We can either have an event analyzer that combines the output of several other event analyzers. This is the case of the Event Analyzer 5 represented in Figure 36. As explained in Chapter 1, specific correlation rules can be derived for that event analyzer that can take advantage of the combined semantic of alerts generated by the preceding event analyzers.
- Also, as represented by Event Analyzers 1 and 6 in Figure 36, we can have several event analyzers dealing with the same stream of raw events. A simple instantiation of this idea is to have two distinct processes on a given machine, both of which analyzing the same source of information differently (e.g. a behavior-based and a knowledge-based system looking at the same sniffed packets).

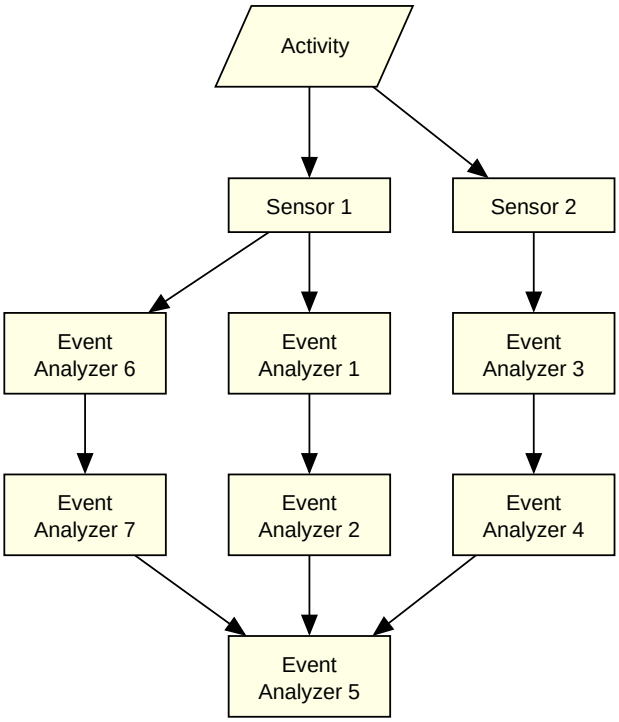



**Figure 36—Second analyzer for the same sensor**

Combining the output of several event analyzers can be used to increase the detection coverage as well as to detect and report attacks against the IDS itself, rendering it intrusion-tolerant. This is possible if, for a given known malicious activity, we know that several event analyzers must generate different alarms. In that case, the absence of an expected alarm in the context of an attack is symptomatic of an error with respect to the IDS service

We illustrate this principle by artificially splitting Event Analyzer 5 into Event Analyzers 5 and 8 in Figure 37. We assume that Event Analyzer 5 implements some form of intrusion masking, as explained before, by correlating the output of several underlying event analyzers. Let us imagine that a user has run an attack in such a way that it evades the detection provided by Event Analyzer 1 but not the one provided by Event Analyzer 6. In that case, the corresponding malicious activity will still be detected when alarms will be combined by Event Analyzer 5. The detection is successful despite the fact that a part of the IDS failed. This is a simple example of *intrusion masking*, in the sense of an intrusion against the IDS system.

Furthermore, let us assume that it is the role of Event Analyzer 8 to verify that the outputs of the various event analyzers are consistent with respect to some expected behavior. In that case, Event Analyzer 8 will be able to raise an alert if Event Analyzer 1 has not generated an expected alarm. Of course, the distinction between Event Analyzer 5 and 8 is rather artificial. In practice, it is quite likely that the same component will not only be able to provide the intrusion-masking capability but will also generate error messages whenever masking intrusions. As explained in [Powell & Stroud 2001] (Section 4.4.1, p. 40), whether masking or detection-and-recovery is used, detected errors and other relevant events are analyzed and reported to the fault treatment facilities. Intrusion-tolerant components are thus a particular kind of internally-monitored components.

The rules used by Event Analyzer 8 are, of course, specific to the underlying other event analyzers. They also are a function of the topology. It is the purpose of the approach developed in Chapter 1 to explore avenues for practical solutions to that problem. In that specific case, one can consider that the Event Analyzer is some sort of *knowledge-based meta-IDSs*, in the sense that it knows which alarm should be raised by whom for which attack.

Similarly, one can think of using some *behavior-based meta-IDS* to enrich the intrusion-tolerant capabilities of our IDS. As an example, let us consider that Event Analyzers 2, 4 and 7 implement the filtering rules described in Chapter 1. On the one hand, these boxes propagate the nondiscarded alerts to Event Analyzer 5. On the other hand, they regularly provide, some statistical information about the accepted and discarded alerts. It is possible to use some behavior-based method to build profiles of the "normal" behavior of these event analyzers. Any change in their behavior indicates that something has changed in the environment or in the IDS system itself. It might be symptomatic of a flood of new attacks (e.g. a new worm spreading) but it might also be due to a nonfunctioning event analyzer (1, 3, and 6 in this case). Here again, such an event analyzer would provide some intrusion-tolerant capability to the IDS.

This model, of course, offers a very simplified view of any real system. The notion of meta-IDS highlights the intrinsicly recursive nature of this model, in the same spirit as the generic MAFTIA component. To come up with a concrete design, topological information will have to be taken into account as well as constraints with respect to the availability of the various sensors and the fault assumptions made. Providing a step-by-step method to guide the design process of concrete IDS lies beyond the scope of this document. The general principles have been given though by means of this model, keeping its recursivness in mind to achieve the required scalability.

So far, we have proposed some rationales for the cascading topology as well as its intrinsic intrusion-tolerant feature. We have also established the link between these features and the work carried out in the other chapters. Note that, by design, the IDS is a distributed application and, therefore, fits the scope of the set of applications that the MAFTIA project, in general, aim at rendering intrusion-tolerant. Therefore, the ID "application" can benefit from the entire body of results obtained in the remainder of the project. It is not our aim to provide an exhaustive list of all MAFTIA results and contribution that are applicable in the ID domain. Instead, we briefly summarize an arbitrary set of the most important, or innovative aspects of them, that we consider especially worthwhile for the IDS designers.

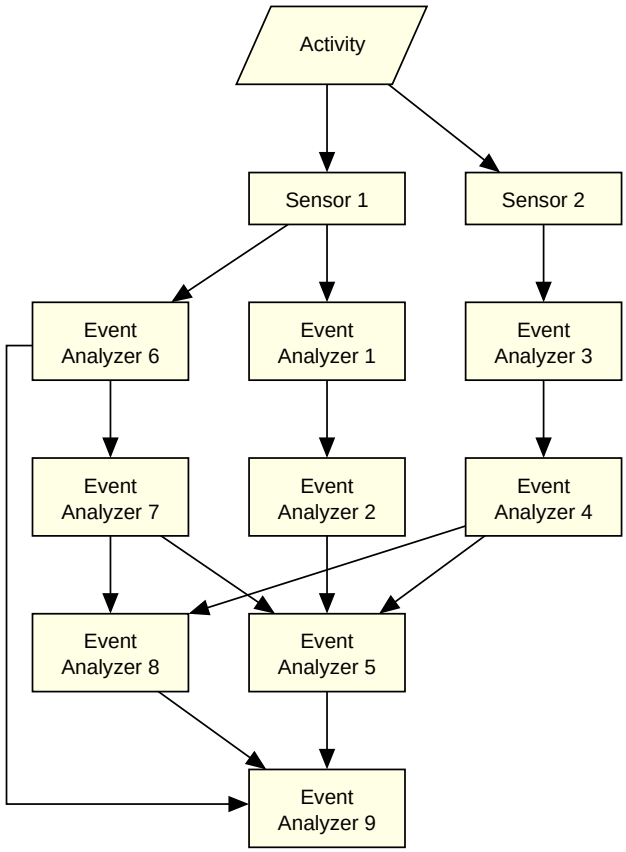



**Figure 37—New analyzer to detect non-functioning IDS**

## 6.4 Relationship with the other MAFTIA work packages.

As for any other distributed application, threats exist against its components: software elements (sensors, event analyzers) and communication channels. In the following, we will not revisit all the threats that the ID application has in common with any other distributed application but instead focus on three new types of threats that are specific to the IDS. Namely, we will study how the IDS can tolerate the fact that

1. an adversary is trying to tamper with the communication channels,
2. a denial-of-service attack can be run against an event analyzer, and
3. an attacker has found a way to assume control over an event analyzer

These various threats and the solution brought forward by contributions of the MAFTIA project are presented in the next three subsections.

### 6.4.1 Channels between ID components

Channels between ID components are of course susceptible to failure. They must thus provide integrity (resistance to message alteration and deletion), authenticity (resistance to message insertion), and quality of service (guaranteed delivery or observable failure). Confidentiality features may be required in settings where logging information could prove dangerously useful to an attacker. This includes, for instance, anonymity (e.g., ensure confidentiality of the identity of a person who has root access) and privacy (e.g., ensure confidentiality of personal data).

The mechanisms for such provisions vary with the channels themselves: a TCB (trusted computing base) may offer all of these for IPC (inter-process communication) whereas network connections may need to resort to redundancy and cryptography.

There are several concerns to be addressed:

- An attacker can interrupt the entire channel.
- An attacker can place a smart filter on the channel that hides only the attacker's activities.
- An attacker can interrupt or hijack the entire channel.
- The channel can be eavesdropped upon.

These problems can be addressed in different ways with different costs:

- A heartbeat event ensures that the channel is alive.
- A cryptographic hash chain that is added to the event stream prevents event deletion.
- Authentication codes prevent event insertion, and event stream hijacking.
- Encryption can prevent the eavesdropping of events.

Such techniques apply not only to transmission but also to storage. Should the logs be stored in a potentially vulnerable location, we can use well-known cryptographic techniques that provide so-called "forward" secrecy [Menezes *et al.* 1996; Schneier, 1996].

These problems have been addressed by MAFTIA in the context of the WP2 work package. The lowest layer of The MAFTIA middleware architecture is the Multipoint Network (MN) module, created over the physical infrastructure [Verissimo & Neves 2001]. Among other things, the MN module provides basic secure channels and message envelopes to protect communication between sites:

- Secure channels are used to protect communication lasting long enough for the concept of connection to make sense. They are based on shared symmetric keys that are used to authenticate messages. The cryptographic signatures are used to protect the integrity of communication against forgery, modification, replay, reorder, and suppression of messages.
- Secure envelopes are used for sporadic transmissions, and seek to achieve site-level transmission security. They resort to per-message security and may use a combination of symmetric and asymmetric cryptography as a way of improving performance.

### 6.4.2 DoS attack against event analyzer

As of today, the most visible threat against existing IDSes consists of denial-of-service attacks. Over the past three years, several sensors have been found to be vulnerable to, sometimes very simple, attacks. Attackers create weird packets to be analyzed by the sensors, and, as a consequence of a design error, the IDS simply crashes. A typical example of such a problem happened at the beginning of 2002 to the Snort IDS,which we have described before:

From [ISS 2002]:

> "[...] It may be possible for remote attackers to send specially crafted ICMP packets to the program, resulting in a segmentation fault that would crash the Snort engine. This attack can be launched from any routable address, and if launched successfully against a Snort-protected network, all IDS functionality may be disabled until Snort is manually restarted.
>
> An exploit has been published that demonstrates a flaw in the ICMP protocol handling functionality. Snort incorrectly handles ICMP "Echo" and ICMP "Echo-Reply" packets that contain less than 5 bytes of ICMP data. If Snort encounters such a packet, it will crash and exit. Packets that are used to exploit this vulnerability can be sent with the "ping" command that is present on most operating systems. [...]"

Other successful attacks exist for various commercial sensors, but have received less publicity. As a consequence, we propose two different methods to address this problem: a watchdog and continuous testing.

#### 6.4.2.1 The immortalizer program

We recommend using a specific, very simple process, watching over each software component of our architecture. This process is called the immortalizer. If no attack happens, the immortalizer process does nothing other than (i) check that the monitor process is neither killed nor modified, and (ii) continuously inform another event analyzer that it is still alive. If the monitored process gets killed, the immortalizer notifies an external event analyzer and restarts the monitorer process immediately. In theory, the “keep alive” heartbeat of the immortalizer could be directly incorporated into the monitored component, but in practice we rarely have access to the code of proprietary software. Furthermore, from a design point of view, sensible pieces of software such as the immortalizer should be kept as simple and as modular as possible. Ideally, they should be simple enough so that a formal proof could be carried out on their code. Therefore, having a separate process is the appropriate solution.

As explained, the immortalizer program is used to keep an active copy of a failure-prone service alive. In our existing implementation, it does so by spawning a child process and monitoring that child. If it should terminate, this fact is logged via the Unix logging facility syslog, and the process is respawned. The immortalizer program itself is extremely simple but general purpose. It is written in C to limit the occurrence of common failure modes that might be experienced in a higher-level program. It does not maintain state for the monitored program nor does it attempt to detect noncrash failures (such as hanging). Immortalize can be configured to periodically send a syslog message indicating that the immortalize process itself is still active. This so-called heartbeat function is included to allow assurance that the immortalizer itself has not crashed.

#### 6.4.2.2 Continuous online testing

Another way of verifying that IDS system is working properly is to periodically launch attacks and to verify that all corresponding and expected alarms are being generated. In order to implement such an idea, we need to provide two different sets of elements to the event analyzer in charge of verifying that all expected alerts have been generated:

1. Some precise information about the type of attacks launched, from where and against what.
2. A mapping that enables the analyzer to deduce from the above information the set of alerts to expect as well as the identification of the sensors involved.

Note that his technique is different from so-called fault-injection techniques. Indeed, goal of the latter is to validate the fault-tolerance assumptions made about some components. Faults are injected into a given system to exercise its fault-tolerant capabilities. In our case, by launching attacks, we are not so much interested in testing the fault-tolerant capabilities of the system under attack, but we simply aim at verifying that the error-detection capabilities work properly. This means that we regularly provide some normal input—which happens to be faults when seen from another point of view—to a given program to test that it works properly. This is continuous online testing. It is agreed that there is a fine line between those two concepts and that they somehow overlap (testing the error-detection mechanism can, from a higher abstraction point of view, be seen as testing the fault tolerance mechanism in general). We emphasis on this issue to illustrate once more the concurrent existence of two different systems: the system under attack and the system protecting it.

From a practical point of view, the results obtained in Chapter 5 can be used directly to provide the event analyzer with the required information about the expected alarms. Also, Thor can be instrumented to implement this continuous testing procedure.

### 6.4.3 Compromised event analyzer

Killing a process is a rather rude approach to compromise it. In the following, we consider the case where a given event analyzer has, somehow, been compromised by a hacker. We furthermore make the assumption that this has not yet been detected. By doing so, the attacker might pursue two different goals:

- Confuse the system officer by having the event analyzer generating numerous false positives.
- Avoid detection of further attacks by instrumenting the event analyzer so that its attacks are no longer reported.

In the first case, it is quite likely that a behavior-based meta-IDS (as described in Section 6.3, p. 89) will quickly notice that something is amiss and will accordingly generate an error message.

The latter case is more interesting as nothing forbids the attacker from hiding his or her traces completely in this way. If we are facing a smart attacker, he or she can modify the code in the event analyzer in such a way that its behavior will be indistinguishable from that of a noncorrupted analyzer, even if we apply continuous testing, as described above, on the system.

However, a solution exists, namely to replicate the event analyzer on several platforms. In this process special care must be taken to avoid the risk of common failure modes so that a majority of the event analyzers always remain noncompromised. By having several event analyzers performing the same treatment on the same in put, we can detect when one of them becomes compromised if we compare their output. The event analyzers can be seen as simple finite state machines that, based on some input received, decide to send events to another event analyzer. If we replicate these finite state machines and want to detect the failure of one them, two options are available:

- Either they all send events to the “upper” event analyzer at any time, and it is the responsibility of the “upper” event analyzer to compare the results and decide whether they are valid (e.g. by using a simple voting mechanism).

- Or, they run a Byzantine agreement protocol before sending the information to the upper level, and send the information only if the agreement protocol succeeds. It is interesting to note that the negative outcome of the agreement might be used as an error that can be sent to the upper event analyzer to inform it that some system is behaving in a nonexpected way, thereby enhancing the intrusion-tolerant capabilities of our IDS.

On the one hand, the algorithmic complexity of the first solution might seem lower, but, on the other hand, we have to consider the fact that the upper level may receive events from a potentially very large number of other event analyzers. The choice between those two solutions will be dictated by the specific constraints of each environment. If the second solution is chosen, we refer the reader to the MAFTIA deliverables [Verissimo & Neves 2001], [Neves & Verissimo 2001], where the abundant literature on Byzantine agreement protocols is presented and where new protocols, developed in the context of MAFTIA, are proposed for synchronous (taking advantage of the TTCB) as well as for asynchronous systems [Cachin, Kursawe & Shoup 2000], [Cachin, Kursawe, Petzold & Shoup 2000].

Note that if we decide to replicate event analyzers, this might open an avenue for new types of attacks. Indeed, an attacker who would have compromised a nonreplicated event analyzer feeding information into a set of replicated event analyzers could try to put these replicas out of sync by sending events to some but not all of them. Here again, it is important to take advantage of the results obtained in the context of the MAFTIA middleware effort. Especially, as explained in [Verissimo & Neves 2001], [Neves & Verissimo 2001] we know that for this specific problem we can use

- reliable broadcast if the ordering of the messages is not important for the replicated event analyzer, or
- atomic Broadcast if replicas could get out of sync if they do not receive the elements in the same order.

The possibilities of using MAFTIA middleware to increase intrusion tolerance of event-analyzer networks are illustrated in Figure 38. The figure shows the option of event analyzers using the Byzantine agreement protocol to agree on a common input and output. The results produced by the agreement protocols are digitally signed, permitting the validation of their validity. In addition, Figure 38 illustrates the usage of the MAFTIA middleware for assuring the consistency of the input provided to event analyzers. Depending on the implementation of the event analyzers, atomic broadcasts may be required to assure that all analyzers receive the events in the same order.

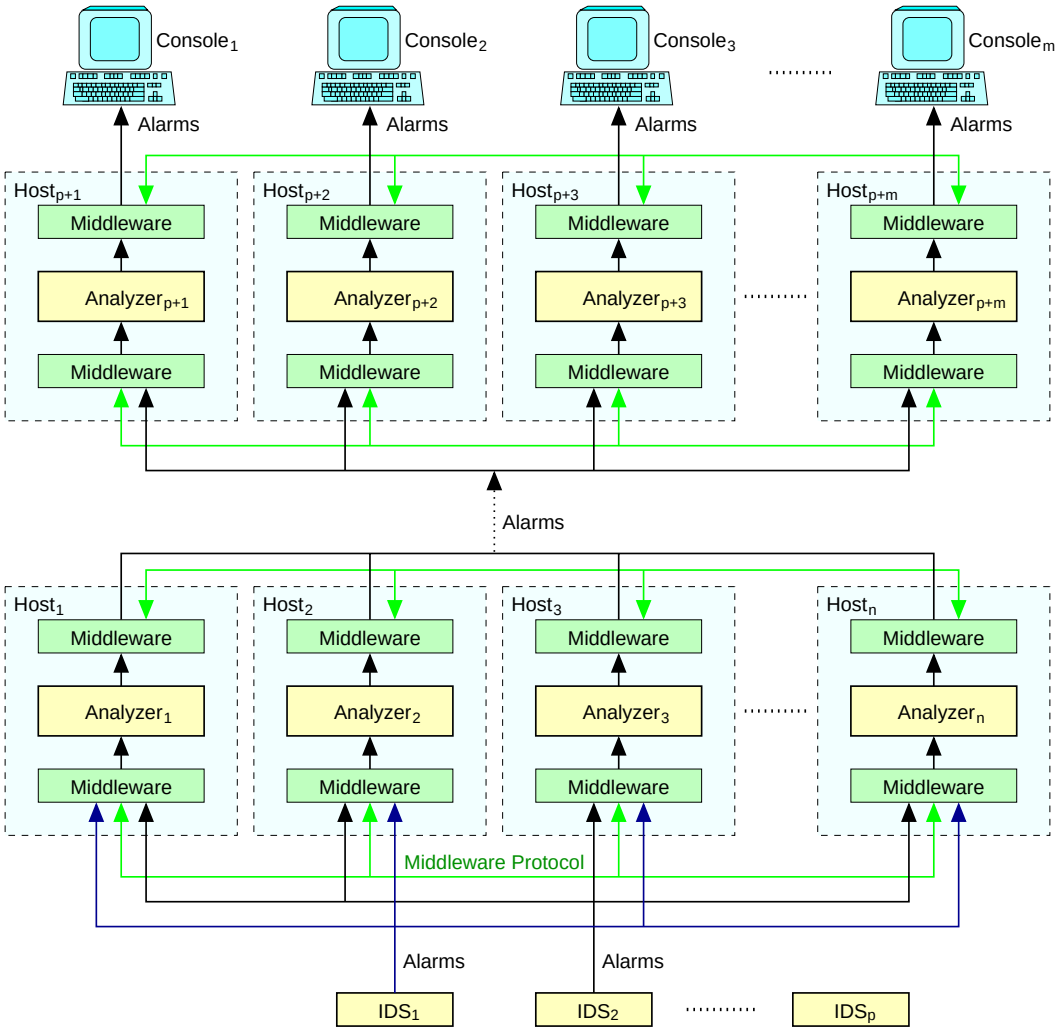



**Figure 38—Example illustrating the two different options of middleware use**

Various algorithms exist and have been presented in the deliverables mentioned above. Two new, efficient results have been proposed for the asynchronous atomic-broadcast protocol [Kursawe 2000], [Kursawe & Shoup 2001] that definitely are worth consideration when designing a setup that would require such properties.

# Chapter 7 Conclusions

In this document, we have considered three main threat categories for intrusion-detection systems:

1. The huge number of false positives they generate offers an easy way for adversaries to flood the system officers with noisy information and, hence, to evade detection.
2. The partial detection coverage offered by each IDS on the one hand and the simplicity of existing techniques to circumvent detection by certain IDSes on the other hand lead to the risk of missing attacks.
3. Distributed intrusion-detection architectures tend to become highly complex, which increases the number of conceivable attacks against their components. Moreover, it renders them vulnerable to attacks, and puts their various elements at risk with respect to attacks against the architectures themselves.

For each problem category, we have reviewed the state of the art, proposed innovative solutions, and investigated their practical application. We have demonstrated how a new clustering algorithm can be used to discover discarding rules that can lead to a very significant reduction of the false-alarm rates. We have proposed a process and several metrics to assess intrusion-detection systems and combinations thereof. We have described an existing prototype, RIDAX, that implements this evaluation method, and we have briefly explained how another prototype, Thor, can be used to design correlation rules to combine the alarms generated by various IDSes efficiently. Last but not least, we have considered the attacks against the IDS itself, and have summarized all the lessons previously learned by means of rationales for a unified model. We have also considered how to enrich that model thanks to other MAFTIA contributions.

# Tables

# Figures

# References

[Agrawal *et al.* 1998]
R. Agrawal, J. Gehrke, D. Gunopulos and P. Raghavan, "Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications," in *ACM SIGMOD International Conference on Management of Data*, pp. 94-105, 1998.

[Alessandri 2001]
D. Alessandri (Ed.), *Towards a Taxonomy of Intrusion Detection Systems and Attacks*, Deliverable D3, Project MAFTIA IST-1999-11583, Research Report, RZ 3366, IBM Zurich Research Laboratory, also available online http://www.MAFTIA.org, 2001

[Alessandri 2002]
D. Alessandri, *Rule-Based Assessment of Intrusion Detection Systems*, RZ 3414, IBM Zurich Research Laboratory

[Almgren *et al.* 2000]
M. Almgren, H. Debar and M. Dacier, "A Lightweight Tool for Detecting Web Server Attacks," in *Network and Distributed System Security Symposium* (NDSS 2000), pp. 157-170, 2000.

[Almgren 1999]
M. 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.

[Anderson & Lee 1981]
T. A. Anderson and P. A. Lee, *Fault Tolerance—Principles and Practice,* Prentice-Hall, 1981.

[Apache 2001]
Apache HTTP Server Project, Apache Software Foundation, available at http://httpd.apache.org, 2001.

[Axelsson 2000]
S. Axelsson, "The Base-Rate Fallacy and the Difficulty of Intrusion Detection," in *ACM Transactions on Information and System Security* (TISSEC), 3(3), 2000, pp. 186-205.

[Barbara and Jajodia 2002]
D. Barbara and S. Jajodia. (Eds.) 2002. *Applications of Data Mining in Computer Security*, Kluwer Academic Publisher, Boston.

[Barbara *et al*. 2001]
D. Barbara, N. Wu and S. Jajodia, "Detecting Novel Network Intrusions Using Bayes Estimators," in *1st SIAM International Conference on Data Mining* (SDM'01), 2001.

[Bellovin 1993]
S. M Bellovin, "Packets Found on an Internet," in *Computer Communications Review*, 23(3), 1993, pp. 26-31.

[Ben-Dor *et al.* 1999]
A. Ben-Dor, R. Shamir and Z. Yakhini, "Clustering Gene Expression Patterns," in *Journal of Computational Biology*, 6(3/4), 1999, pp. 281-297.

[Bloedorn *et al.* 2000]
E. Bloedorn, B. Hill, A Christiansen, C. Skorupka, L. Talboot and J. Tivel, *Data Mining for Improving Intrusion Detection*, 2000, available online http://www.mitre.org/support/papers/tech_papers99_00/.

[Broderick 1998]
Broderick (Ed.), *IBM outsourced solution*, 1998, available online http://www.infoworld.com/cgi-bin/displayTC.pl?/980504sb3-ibm.htm.

[Cachin, Kursawe & Shoup 2000]
Christian Cachin, Klaus Kursawe and Victor Shoup, "Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement using Cryptography," *ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, 2000, pp. 123-132.

[Cachin, Kursawe, Petzold & Shoup 2001]
Christian Cachin, Klaus Kursawe, Frank Petzold and Victor Shoup, "Secure and efficient asynchronous broadcast protocols," *Advances in Cryptology*: Crypto 2001, pp. 524-541, 2001.

[CERT 1996-06]
CERT, *Advisory CA-1996-06*, CERT Coordination Center, available at http://www.cert.org/advisories/CA-1996-06.html, 1996.

[CERT 1996-26]
CERT*, Advisory CA-1996-26: Denial-of-Service Attack via ping*, CERT Coordination Center, available online http://www.cert.org/advisories/CA-1996-26.html, 1996

[CERT 2001-19]
CERT, *Advisory CA-2001-19: "Code Red" Worm Exploiting Buffer Overflow in IIS Indexing Service DLL*, CERT Coordination Center, available online http://www.cert.org/advisories/CA-2001-19.html, 2001

[Chan and Stolfo 1998]
P. Chan and S. Stolfo, "Toward Scalable Learning with Non-Uniform Class and Cost Distributions: A Case Study in Credit Card Fraud Detection," in *4th International Conference on Knowledge Discovery and Data Mining*, 1998, pp. 164-168.

[Cisco]
Cisco Systems, Inc., *NetRanger Documentation*, available online http://www.cisco.com/.

[Clifton and Gengo 2000]
C. Clifton and G. Gengo, G, "Developing Custom Intrusion Detection Filters Using Data Mining," in *Military Communications Int'l Symposium* (MILCOM2000), 2000

[Clocksin and Mellish 1994]
W. F. Clocksin and C. S. Mellish, *Programming in Prolog*, Fourth ed. Berlin, Heidelberg: Springer-Verlag, 1994, ISBN 3-540-58350-5.

[Cover and Thomas 1991]
T. M. Cover and J. A. Thomas, *Elements of Information Theory*. New York, NY: John Wiley & Sons, Inc., 1991, ISBN 0-471-06259-6.

[Cuppens 2001]
F. Cuppens, "Managing Alerts in a Multi-Intrusion Detection Environment," in *17th Annual Computer Security Applications Conference* (ACSAC), pp. 22-31, 2001.

[Cuppens and Miege 2002]
F. Cuppens and A. Miege, "Alert Correlation in a Cooperative Intrusion Detection Framework," in *IEEE Symposium on Security and Privacy*, Oakland, CA, 2002

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

[Dain and Cunningham 2002]
O. Dain and R. K. Cunningham, Fusing Heterogeneous Alert Streams into Scenarios, 2002

[Debar *et al.* 1999]
H. Debar, M. Dacier and A. Wespi, “Towards a Taxonomy of Intrusion Detection Systems,” *Computer Networks*, 31(8), pp.805-822, 1999

[Debar *et al.* 2000]
H. Debar, M. Dacier and A. Wespi, "A Revised Taxonomy for Intrusion Detection Systems," *Annales des Télécommunications*, 55(7-8), pp. 361-378; 2000.

[Debar and Wespi 2001]
H. Debar and A. Wespi, "Aggregation and Correlation of Intrusion Detection Alerts," in *4th Workshop on Recent Advances in Intrusion Detection* (RAID), LNCS, vol. 2212. Springer Verlag, pp. 85-103, 2001.

[Deraison 2000]
R. Deraison, *Nessus*, available at http://www.nessus.org, 2000.

[Diaz 2000]
D. Diaz, *GNU Prolog - A Native Prolog Compiler with Constraint Solving over Finite Domains v1.2.1*, http://www.gnu.org/software/prolog, 2000.

[Disk *et al.* 2000]
J. Disk, Simple Nomad and Irib, "Project Area52: Delirium Tremens," in *Phrack Magazine*, vol. 10(56), http://www.phrack.org/show.php?p=56&a=6, 2000.

[Dragon 2001]
*Dragon IDS*, available at http://www.enterasys.com/ids/, 2001.

[Erlinger and Staniford-Chen]
M. Erlinger and S. Staniford-Chen, *Intrusion Detection Exchange Format (idwg*), available online http://www.ietf.org/html.charters/idwg-charter.html.

[Fasulo 1999]
D. Fasulo, *An Analysis of Recent Work on Clustering Algorithms*, available online http://www.cs.washington.edu/homes/dfasulo/, 1999.

[Fawcett and Provost 1997]
T. Fawcett and F. Provost, "Adaptive Fraud Detection," in *Data Mining and Knowledge Discovery*, 1, pp. 291-316, 1997.

[Fyodor]
*Fyodor*, available online http://www.insecure.org/nmap.

[Ganti *et al*. 1999]
V. Ganti, J. Gehrke and R. Ramakrishnan, "CACTUS - Clustering Categorical Data Using Summaries," in *5th ACM SIGKDD International Conference on Knowledge Discovery in Databases* (SIGKDD), pp. 73-83, 1999.

[Gibson *et al.* 2000]
D. Gibson, J. M. Kleinberg and P. Raghavan, "Clustering Categorical Data: An Approach Based on Dynamical Systems," in *VLDB Journal*, 8(3), pp. 222-236, 2000.

[Guha *et al.* 2000]
S. Guha, R. Rastogi and K. Shim, "ROCK: A Robust Clustering Algorithm for Categorical Attributes," in *Information Systems*, 25(5), pp. 345-366, 2000.

[Haines *et al.* 2001]
J. W. Haines, L. M. Rossey, R. P. Lippmann and R. K. Cunningham, "Extending the DARPA Off-Line Intrusion Detection Evaluations," in *DARPA Information Survivability Conference & Exposition II* (DISCEX '01), Anaheim, CA, vol. 1, ISBN 0-7695-1212-7, 2001, pp. 35-45, 2001.

[Halme & Bauer]
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.

[Han *et al.* 1992]
J. Han, Y. Cai and N. Cercone, "Knowledge Discovery in Databases: An Attribute-Oriented Approach". in *18th VLDB Conference*, pp. 547-559, 1992.

[Han *et al*. 1993]
J. Han, Y. Cai and N. Cercone, "Data-Driven Discovery of Quantitative Rules in Relational Databases," in IEEE Transactions on Knowledge and Data Engineering, 5(1), pp. 29-40, 1993.

[Han and Kamber 2000]
H. Han and M. Kamber, *Data Mining: Concepts and Techniques*, Morgan Kaufmann Publisher, 2000.

[Hellerstein and Ma 2000]
J. L. Hellerstein and S. Ma, "Mining Event Data for Actionable Patterns," in *The Computer Measurement Group*, 2000.

[Horizon 1998]
Horizon, "Defeating Sniffers and Intrusion Detection Systems," in *Phrack Magazine*, vol. 8(54), http://www.phrack.org/show.php?p=54&a=10, 1998.

[IDEF]
IETF, *Intrusion Detection Exchange Format*, http://www.ietf.org/html.charters/idwg-charter.html, (accessed: 5 September).

[Ilung 1993]
K. Ilung. "USTAT: A Real-Time Intrusion Detection System for UNIX," in *IEEE Symposium on Security and Privacy*, Oakland, CA, pp. 16-28, 1993.

[Internet Security Systems 2001]
Internet Security Systems, Inc. 2001, *RealSecure Signatures Reference Guide*, available online http://documents.iss.net/literature/RealSecure/RS_Signatures_6.0.pdf.

[ISS 2002]
Internet Security Systems Security Alert, *Remote Denial of Service Vulnerability in Snort IDS*, January 28, 2002 (Last Revised: February 1, 2002), available at bvlive01.iss.net/issEn/delivery/xforce/alertdetail.jsp?id=advise108.

[Jain and Dubes 1988]
A. Jain and R. Dubes, Algorithms for Clustering Data, Prentice-Hall, 1988.

[Julisch 2001]
J. Klaus, "Whitepaper on Handling Large Amounts of Intrusion Detection Alarms," in *Proceedings of the 17th ACSAC*, New Orleans, December 2001.

[Julisch 2001b]
K. Julisch, "Mining Alarm Clusters to Improve Alarm Handling Efficiency," in *17th Annual Computer Security Applications Conference* (ACSAC), pp. 12-21, 2001.

[Julisch 2002]
K. Julisch, "Using Root Cause Analysis to Handle Intrusion Detection Alarms," in *ACM Transactions on Information System Security*, 2(3), Sept. 2002, pp. 111-138, 2002.

[Klemettinen 1999]
M. Klemettinen, *A Knowledge Discovery Methodology for Telecommunication Network Alarm Data*, Ph.D. thesis, University of Helsinky (Finland), 1999.

[Klinger *et al.* 1995]
S. Klinger, S. Yemini, D. Ohsie and S. Stolfo S., "A Coding Approach to Event Correlation," in *Fourth IEEE/IFIP International Symposium on Integrated Network Management*, Santa Barbara, CA, vol. 4, ISBN 0-412-71570-8, http://www.cs.columbia.edu/ids/research/keypapers/papers/eventcorrelation/isinm95.pdf, 1995, pp. 266-2—77, 1995.

[Koller and Sahami 1996]
D. Koller and M. Sahami, "Toward Optimal Feature Selection," in *13th International Conference on Machine Learning* (ICML-96), pp. 284-292, 1996.

[Kumar 1995]
S. Kumar, *Classification and Detection of Computer Intrusions*, Ph.D. thesis, Purdue University, 1995.

[Kursawe 2000]
Klaus Kursawe, "Optimistic Asynchronous Byzantine Agreement," Research Report, RZ 3202, IBM Zurich Research Laboratory, 2000.

[Kursawe & Shoup 2001]
Klaus Kursawe and Victor Shoup, "Optimistic Asynchronous Atomic Broadcast," Cryptology ePrint Archive, Report 2001/022, 2001.

[Laprie 1992]
J.C. Laprie (Ed.*), Dependability: Basic Concepts and Terminology*, Dependable Computing and Fault-Tolerant Systems, vol. 5. Springer-Verlag, Vienna, 1992.

[Laprie *et al.* 1995]
J.-C. Laprie, J. Arlat, J.-P Blanquart, A. Costes, Y. Crouzet, Y. Deswarte, J.-C. Fabre, H. Guillermain, M. Kaâniche, K. Kanoun, C. Mazet, D. Powell, C. Rabéjac and P. Thévenod, *Dependability Guidebook*, Cépaduès-Editions, Toulouse, 1995.

[Lee and Stolfo 2000]
W. Lee and S. Stolfo, "A Framework for Constructing Features and Models for Intrusion Detection Systems," in *ACM Transactions on Information and System Security* (TISSEC), 3(4), pp. 227-261, 2000.

[Lippmann *et al.* 2000]
R. Lippmann *et al.*, "Evaluating Intrusion Detection Systems: The 1998 DARPA Off-Line Intrusion Detection Evaluation," in *DISCEX'00 - DARPA Information Survivability Conference & Exposition*, Hilton Head, SC, vol. 2, 2000, pp. 12-26.

[Lippmann *et al.* 2000b]
R.Lippmann, J. W. Haines, D. J. Fried, J. Korba and K. Das, "Analysis and Results of the 1999 DARPA Off-Line Intrusion Detection Evaluation," in *Third Intl. Workshop on Recent Advances in Intrusion Detection* (RAID2000), Toulouse, LNCS, vol. 1907*,* ISBN ISBN 3-540-41085-6, 2000, pp. 162-182, 2000.

[Liu and Setiono 1996]
H. Liu and R. Setiono, "A Probabilistic Approach to Feature Selection - a Filter Solution". in *13th International Conference on Machine Learning*, Morgan Kaufmann, pp. 319-327, 1996.

[Livingston *et al.* 2001]
A. Livingston, G. Jackson and K. Priestley, *Root Causes Analysis: Literature Review*, HSE Books. available online http://www.hsebooks.co.uk/, 2001.

[Manganaris *et al.* 2000]
S. Manganaris, M. Christensen, D. Zerkle and K. Hermiz, "A Data Mining Analysis of RTID Alarms," in *Computer Networks*, 34(4), pp. 571-577, 2000.

[Marty 2002]
R. Marty, *Thor: A tool to Test Intrusion Detection Systems by Variations of Attacks*, RZ 3421, IBM Research Zurich Lab, available at www.raffy.ch/projects/ids/thor.pdf, 2002.

[McHugh 2000]
J. McHugh, "The Lincoln Labouratories Intrusion Detection System Evaluation: A Critique," presented at *DISCEX'00 - DARPA Information Survivability Conference & Exposition, Hilton Head*, SC*,* ISBN 0-7695-0490-6, 2000 (article removed from the proceedings).

[McHugh 2000b]
J. McHugh, "The 1998 Lincoln Laboratory IDS Evaluation: A Critique," *in Third Intl. Workshop on Recent Advances in Intrusion Detection* (RAID2000), Toulouse, published in LNCS, vol. 1907, 2000, pp. 143-161, 2000.

[Menezes *et al.* 1996]
A. J. Menezes, P.C. van Oorschot and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.

[Mounji 1997]
A. Mounji, *Languages and Tools for Rule-Based Distributed Intrusion Detection*, Ph.D. thesis, Facultes Universitaires Notre-Dame de la Paix Namur (Belgium), 1997.

[MySql 2000]
MySQL AB, *MySQL Database*, http://www.mysql.com/, 2000.

[NCSA 1996]
NCSA HTTPd Development Team, *NCSA HTTPd*, available at ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd, 1996.

[Neves & Verissimo 2001]
N. F. Neves. and P. Verissimo. (Eds.), *First Specification of APIs and Protocols for the MAFTIA Middleware*, Deliverable D24, Project MAFTIA IST-1999-11583, Aug. 2001, Technical Report DI/FCUL TR-01-06, University of Lisboa, also available online http://www.MAFTIA.org.

[nfr]
*NFR Network Intrusion Detection*, available at http://www.nfr.com/products/NID/, 2001.

[Ning *et al.* 2001]
P. Ning, S. Jajodia and X. Wang, "Abstraction-Based Intrusion Detection in Distributed Environments," in *ACM Transactions on Information and System Security* (TISSEC), 4(4), pp. 407-452, 2001.

[nmap]
Network Mapper (NMAP), *Stealth Port Scanner*, available at http://www.nmap.org.

[NSA 1998]
NSA, *NSA Glossary of Terms Used in Security and Intrusion Detection*, National Security Agency, http://www.sans.org/newlook/resources/glossary.htm, 1998 (accessed: 5 September).

[NSS 2001]
NSS Group, Intrusion Detection Systems Group Test, Edition 2, available at http://www.nss.co.uk/ids, December 2001.

[Papadimitriou 1994]
C. H. Papadimitriou, *Computational Complexity*, Addison-Wesley, 1994.

[Paradies and Busch 1988]
M. Paradies and D. Busch, "Root Cause Analysis at Savannah River Plant," In *IEEE Conference on Human Factors and Power Plants*, pp. 479-483, 1988.

[Paxson 1999]
V. Paxson, "Bro: A system for detecting network intruders in real-time," in *Computer Networks*, 31(23-24), pp. 2435-2463, 1999.

[Perl87]
Perl Mongers, *Perl*, http://www.perl.org/, 1987.

[Porras *et al.*]
P. Porras, D. Schnackenberg, S. Staniford-Chen and M. Stillman, *The Common Intrusion Detection Framework Architecture*, CIDF working group, http://www.gidos.org/drafts/architecture.txt.

[Powell & Stroud 2001]
D. Powell and R. Stroud. (Eds.), *Conceptual Model and Architecture*, Deliverable D2, Project MAFTIA IST-1999-11583, Nov. 2001, Research Report, RZ 3377, IBM Zurich Research Laboratory, also available online http://www.MAFTIA.org.

[Price 1997]
K. E. Price, *Host-Based Misuse Detection and Conventional Operating Systems Audit Data Collection*, M.S.thesis, Purdue University, 1997.

[Ptacek and Newsham 1998]
T. H. Ptacek and T. N. Newsham, *Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection*, Tech. rep., Secure Networks, Inc. January 1998.

[Puppy 2000]
Rain Forest Puppy, *A look at whisker's anti-IDS tactics - Just how bad can we ruin a good thing?*, www.wiretrip.net/rfp/pages/whitepapers/whiskerids.html, 2000.

[Puppy]
Rain Forrest Puppy, *Whisker*, software available online at www.wiretrip.net/rfp/bins/whisker/v2.0/whisker-pr2.0.tar.gz, 2002.

[Purdue Directory]
*Purdue Directory Service*, available at ftp://ftp.purdue.edu/pub/.

[RFC791]
IETF, *Request for Comments: 791*, Internet Protocol, available at http://sunsite.cnlab-switch.ch/ftp/doc/standard/rfc/7xx/791, September 1981.

[Rigoutsos and Floratos 1998]
I. Rigoutsos, I. and A. Floratos, "Combinatorial pattern discovery in biological sequences: The TEIRESIAS Algorithm," in *Bioinformatics*, 14(1), pp. 55-67, 1998.

[Roesch99]
M. Roesch, "Snort: Lightweight Intrusion Detection for Networks," in *LISA '99: 13th Systems Administration Conference*, Seattle, WA, http://www.snort.org/, http://www.usenix.org/publications/library/proceedings/lisa99/full_papers/roesch/roesch.pdf, 1999, pp. 229-138.

[Rooney *et al.* 2002]
S. Rooney, C. Giblin and A. Bussani, *Remote Code Browsing - a network based computation utility*, Research Report, RZ 3416, IBM Zurich Research Laboratory, 2002.

[Rossey *et al.* 2001]
L. M. Rossey., R. K. Cunningham, D. J. Fried, J. C. Rabek, R. P. Lippmann and J. W. Haines, "LARIAT: Lincoln Adaptable Real-time Information Assurance Testbed," *in Fourth International Workshop on Recent Advances in Intrusion Detection* (RAID2001), UC Davis, CA, http://www.raid-symposium.org/raid2001/program.html, 2001.

[Sasha and Beetle 2000]
Sasha and Beetle, "A Strict Anomaly Detection Model for IDS," in *Phrack Magazine*, vol. 10(56), http://www.phrack.org/show.php?p=56&a=11, 2000.

[Schneier 1996]
B. Schneier, *Applied Cryptography,* Wiley and Sons, Inc., 1996.

[Sekar *et al.* 1999]
R. Sekar, Y. Guang, S. Verma and T. Shanbhag, "A High-Performance Network Intrusion Detection System," in *6th ACM Conference on Computer and Communications Security*, 8-17, ISBN 1-58113-148-8, 1999.

[Sobirey 1998]
M Sobirey, *Michael Sobirey's Intrusion Detection Systems page*, http://www-rnks.informatik.tu-cottbus.de/~sobirey/ids.html, November 1998, last update: June 15, 2000.

[Song 2002]
D. Song D, *Fragroute*, http://www.monkey.org/~dugsong/fragroute/, 2002.

[Song 1999a]
D. Song, *Fragrouter - NetworkIintrusion Detection Evasion Toolkit*, Anzen Computing, Manual Page http://www.netflood.net/files/IDS/fragrouter.html, 1999.

[Song 1999b]
D. Song, *Fragrouter*, software available at www.anzen.com/research/nidsbench/, 1999.

[Stewart 1999]
A. J. Stewart, *Distributed Metastasis: A Computer Network Penetration Methodology*, The packet Factory, http://magnificent.sk/d0x/distributed_metastasis.pdf,

References


[Sturluson]
S. Sturluson, "The Death of Balder," in *The Prose Edda*, available at http://www.pitt.edu/~dash/balder.html.

[Staniford *et al.* 2000]
S. Staniford, J. A. Hoagland and J. M. McAlerney, "Practical Automated Detection of Stealthy Portscans," in *ACM Computer and Communications Security IDS Workshop*, pp. 1-7, 2000.

[Tanenbaum 1996]
A. S. Tanenbaum, *Computer Networks*. Prentice-Hall. ISBN: 0-13-394248-1, 1996.

[Tivoli 2000]
Tivoli Systems, *Tivoli SecureWay Risk Manager, User's Guide v3.7*, IBM Corp., http://www.tivoli.com/products/index/secureway_risk_mgr/, 2000.

[Tripwire 1999]
*Tripwire v1.2*, Tripwire Security Systems Inc., available at http://www.tripwiresecurity.com, 1999.

[Valdes and Skinner 2001]
A. Valdes and K. Skinner, "Probabilistic Alert Correlation," in *4th Workshop on Recent Advances in Intrusion Detection* (RAID), LNCS, vol. 2212, Springer Verlag, pp. 54-68, 2001.

[Verissimo & Neves 2001]
P. Verissimo and N. F. Neves (Eds.), *Service and Protocol Architecture for the MAFTIA Middleware*, Deliverable D23, Project MAFTIA IST-1999-11583, January 2001, available online http://www.MAFTIA.org.

[Wespi *et al.* 2000]
A. Wespi, M. Dacier, and H. Debar, “Intrusion Detection Using Variable-Length Audit Trail Patterns,” in *Third International Workshop on Recent Advances in Intrusion Detection* (RAID2000), Toulouse, France, LNCS, vol. 1907, ISBN 3-540-41085-6, http://link.springer.de/link/service/series/0558/bibs/1907/19070110.htm, 2000, pp. 110-130.

[Wespi and Debar 1999]
A. Wespi and H. Debar, “Building an Intrusion Detection System to Detect Suspicious Process Behavior,” in *RAID 99, Workshop on Recent Advances in Intrusion Detection*, West Lafayette, IN, 1999.

[Whalley *et al.* 2000]
I. Whalley *et al.*, "An Environment for Controlled Worm Replication and Analysis," in *the Virus Bulletin Conference*, Orlando, Florida, 2000.

[Vmware]
*VMware*, VMware Inc., available at http://www.vmware.com, 2001.

[Yang and Pedersen 1997]
Y. Yang and J. O. Pedersen, "A Comparative Study on Feature Selection in Text Categorization," in *14th International Conference on Machine Learning*, pp. 412-420, 1997.

[Zamboni 2001]
D. Zamboni, *Using Internal Sensors for Computer Intrusion Detection*, Ph.D. thesis, Purdue University, 2001.