THESIS FOR THE DEGREE OF LICENTIATE OF ENGINEERING

Extending Intrusion Detection with Alert Correlation and Intrusion Tolerance

DAN GORTON

Department of Computer Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2003

Extending Intrusion Detection with Alert Correlation and Intrusion Tolerance

DAN GORTON

Technical Report no. 27 L
ISSN 1651-4963
School of Computer Science and Engineering

Department of Computer Engineering
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone: +46 (0)31-772 10 00
www.ce.chalmers.se

Printed in Sweden
Chalmers Reproservice
Göteborg, Sweden 2003

Extending Intrusion Detection with Alert Correlation and Intrusion Tolerance

DAN GORTON
Department of Computer Engineering
Chalmers University of Technology

Thesis for the degree of Licentiate of Engineering, a degree that falls between M.Sc. and Ph.D.

Abstract

Intrusion detection is an important security tool. It has the possibility to provide valuable information about the current status of security. However, as enterprises deploy multiple intrusion detection sensors at key points in their networks, the issue of correlating messages from these sensors becomes increasingly important. A correlation capability reduces alert volume, and potentially improves detection performance through sensor reinforcement or complementarity. Correlation is especially advantageous when heterogeneous sensors are employed because of the potential to aggregate different views of the same incident. This thesis studies a number of different properties of intrusion alert correlation, such as standard formats and similarity metrics. A conceptual framework is suggested, followed by three different case studies. Firstly, a router based IDS is implemented and analyzed. The quality of the event source is found to be unreliable and the consequences for intrusion detection and correlation are evaluated. Secondly, a case study of live traffic analysis is performed using heterogeneous intrusion alert correlation. A successful correlation is presented. Thirdly, the possibility to implement intrusion alert correlation using open source tools is evaluated. A simple prototype is implemented. However, even if the performance of the intrusion detection systems increases, there will always be intrusions. One way to remedy this problem is to use fault tolerant techniques in a new setting, providing intrusion tolerance. This results in a system that will continue to provide a valid service, possibly with a performance penalty, in spite of current intrusion attempts. This thesis studies different ways to implement intrusion tolerant services. Additionally, an intrusion tolerant reverse proxy server is implemented and analyzed. All in all, we show the value of intrusion alert correlation and intrusion tolerance in different settings.

Keywords: computer security, vulnerability, intrusion, intrusion detection, intrusion alert correlation, intrusion tolerance.

List of included papers and research efforts

This thesis is based upon the following papers and research efforts:

- Heterogeneous Sensor Correlation: A Case Study of Live Traffic Analysis. Dan Andersson[1], Martin Fong, and Alfonso Valdes. Presented at IEEE Assurance and Security, United States Military Academy, West Point, NY, June 2002.
- Testing Intrusion Detection Systems - A Survey. Dan Andersson, and Helén Svensson. Presented at the Fourth Nordic Workshop on Secure IT Systems NORDSEC'99, Kista, Sweden, 1-2 November 1999.
- An Adaptive Intrusion-Tolerant Server Architecture. A. Valdes et al. Technical Report. System Design Laboratory, SRI International. Feb, 2002.
- Firewalls as an Event Source for Intrusion Detection. Draft. Dan Andersson. SRI International, System Design Laboratory, 2000.

[1] Dan Andersson changed name to Dan Gorton as of October 11, 2003.

Acknowledgement

I would like to start out by thanking my supervisor Prof. Erland Jonsson for guiding me through this licentiate thesis. Thanks also to the current and former members of the security research group at Chalmers University of Technology, Department of Computer Engineering: Magnus Almgren, Stefan Axelsson, Ulf Gustafson, Hans Hedbom, Håkan Kvarnström, Stefan Lindskog, Ulf Lindqvist, Emilie Lundin, Lars Strömberg, and Helén Svensson.

Several external parties have sponsored this dissertation.

AerotechTelub Communications should have an eloge for supporting the last part of the thesis work and in particular Joakim Karlmark, Joakim Kruse, Jerker Löf, Magnus Svensson, and Sten Sörenson.

Thanks to WM-data Försvarsdata, I had the chance to start with my (part time) Ph.D. studies. Special thanks to: Krister Gustavson, Johan Hallén, Staffan Jackson, Claes Thagemark, and Marie Thorzell.

Thanks also to Telia Research Infosecurity for interesting projects and interesting discussions about security around the coffee table: Nils Daniles, György Enderz, Anders Hansmats, Jessica Gunnarsson, Håkan Kvarnström, Ola Sjögren, Roberto Zamparo, as well as Mattias Jorstedt (who was working on his masters thesis at the time).

Thanks to SRI International, I had the chance to participate in cutting edge intrusion detection research. As an International Fellow working together with the EMERALD-team at SDL (System Design Laboratory) I took on the work to analyze the possibility to IDS'ify router logs, implementing Snort into the EMERALD suite, and develop the first intrusion tolerant (web proxy) demonstrator. Special thanks to: Magnus Almgren (again), Martin Fong, Ulf Lindqvist (again), Phil Porras, and Alfonso Valdes.

Last, but most important. I had not been able to finish this work if it had not been for my wife Sara Gorton, and my closest family: Kerstin, Stefan, and Mikael Bolling, and Lars, Christina, and Pia Gorton.

And, thanks everybody else that I have forgotten to mention above!

Contents

1 Introduction

Intrusion detection is an important security tool. It has the possibility to provide valuable information about the current status of security. However, several reports show that the volume of alerts generated from intrusion detection systems can be overwhelming [25][85]. The alert volume often stems from the intrusion detection system generating too elementary alerts, e.g. through the use of a one-to-one mapping between events and alerts. Additionally, the performance of intrusion detection may be further hampered by the generation of false alarms, sometimes in combination with an alarmingly low detection rate [87]. Furthermore, as enterprises deploy multiple intrusion detection sensors at key points in their networks, the issue of correlating messages from these sensors becomes increasingly important. A correlation capability has the potential to reduce alert volume and improve detection performance through sensor reinforcement or complementarity. Correlation is especially advantageous when heterogeneous sensors are employed because of the potential to aggregate different views of the same incident.

The recent interest in intrusion alert correlation has resulted in several papers from several research groups, including groups at IBM, MIT, and SRI International [10][12][23][24] [25][26]. These recent developements in intrusion alert correlation clearly have considerable potential. In this thesis we study a number of different properties of intrusion alert correlation, such as standard formats and similarity metrics. A conceptual framework is suggested, followed by three different case studies. Firstly, a router based IDS is implemented and analyzed. Secondly, a case study of live traffic analysis is performed using heterogeneous intrusion alert correlation. Thirdly, the possibility to implement intrusion alert correlation using open source tools is evaluated.

However, even if the performance of the intrusion detection system increases through the use of intrusion alert correlation, there will always be intrusions. One way to remedy this problem is to use fault tolerant techniques in a new setting, providing intrusion tolerance. This results in a system that will continue to provide a valid service, possibly with a performance penalty, in spite of current intrusion attempts [36]. This thesis studies different ways to implement intrusion tolerant services. Additionally, an intrusion tolerant reverse proxy server is implemented and analyzed.

2 Organization

The rest of this thesis is organized as follows. Section 3 gives the problem statement. In section 4, a short background of security and intrusions is given. This is followed by an introduction of basic properties of intrusion alert correlation and intrusion tolerance, including surveys of related research.

In section 5, intrusion alert correlation is analyzed further and we present a set of generic properties of the surveyed intrusion alert correlation methods.

Section 6 includes three different intrusion detection and alert correlation experiments. The first experiment presents an intrusion detection system analyzing event records created by access rules in routers. The second experiment presents a 24 hour live traffic analysis correlating Snort alert messages together with alerts from various sensors in the EMERALD[1] suite. The third experiment presents an analysis of the possibility to customize current Open Source tools to create an alternative to expensive commercial Intrusion Alert Correlation (IAC) systems.

In section 7, an experiment extending intrusion detection with intrusion tolerance is presented. An intrusion tolerant web proxy server is implemented and the additional value of using intrusion tolerant web proxies is analyzed.

Section 8 presents a summary of the licentiate thesis and section 9 presents the main results.

[1] http://www.sdl.sri.com/projects/emerald.

3 Problem Statement

The research reported in this thesis has focused on ways to extend intrusion detection with intrusion alert correlation and intrusion tolerance.

The assumption has been that existing computing systems mostly consists of various combinations of COTS (Commercial off the shelf) software and operating systems. Additionally, these systems are already under the protection of basic security measures (such as authentication servers, access control, firewalls, intrusion detection systems etc.).

When the possibility of making individual COTS software more secure (using e.g. configuration options) has been exhausted, other options need to be evaluated. This thesis has analyzed two additional measures: correlation of intrusion alerts for better intrusion detection performance, and intrusion tolerance i.e. using traditional fault tolerant techniques in a "new" setting.

3.1 Research Questions

The process of surveying, testing and evaluating different intrusion alert correlation, and intrusion tolerance methods includes the following research questions:

- What techniques and methods exist for intrusion alert correlation?
- What techniques and methods exist for intrusion tolerance?

With a sound knowledge of the answers to the questions above, four different systems were implemented and evaluated. The main research questions are:

- What are the implications of using router based logs in the intrusion detection and correlation process?
- What is the additional value of using intrusion alert correlation together with heterogenous intrusion detection sensors?
- Is it possible to customize current Open Source to create effective intrusion alert correlation systems?
- What is the additional value of using intrusion tolerant web proxies?

3.2 Methods

To answer the research questions above the following methods were used:

- Survey of intrusion alert correlation and intrusion tolerant services
- Experimental evaluation of implemented prototypes

4 Background on Security and Intrusions

This section starts out with a brief presentation of some important concepts concerning computer and network security. This is followed by an initial presentation of intrusion detection, intrusion correlation (which we partition into intrusion event correlation and intrusion alert correlation) and intrusion tolerance.

4.1 Computer Security

The goal of computer security is to provide *trust* in a computing system (in this thesis defined as hardware, software, and data) primarily through three different attributes: *confidentiality*, *integrity*, and *availability* [1][2][3][4][5]. The purpose of confidentiality is to limit access to computing systems to only authorized users (could be a person, process, or system). The purpose of integrity is to limit the possibility to alter or destroy computing resources (both maliciously and accidentally). The purpose of availability is to make the computing system provide proper service to authorized users when expected.

To implement security into a computing system, different security services are needed. In a network security setting five generic security services are needed (at least): *authentication*, *access control*, *confidentiality*, *integrity*, and *non-repudiation* [6]. The purpose of authentication is to verify a user provided identity. The purpose of access control is to restrict access to objects or services to only authorized users. The purpose of non-repudiation is to provide proof of delivery and sender identity.

A computing system is not secure if it has one or more *vulnerabilities*. A vulnerability (or *security flaw*) is most often defined as a weakness in a computing system that allows a violation of the *security policy* [3].

There exist several taxonomies trying to classify vulnerabilities using different perspectives. One of the most referenced taxonomies is the taxonomy of computer software security flaws published in 1994 [70]. The taxonomy partitions security flaws according to three different aspects: how did the flaw enter the software (genesis), when did it happen (time of introduction), and where can it be found (location).

The genesis of a security flaw is first divided into intentional and inadvertant entry. An intentional entry of a security flaw may be malicious or non-malicious, where malicious entry includes trojan horses, trap doors, and time-bombs. Non-malicious entry includes design flaws that can be used to a attack the security of a system, e.g. using covert channels. Inadvertent entry of a security flaw includes various forms of logical errors, like validation errors, domain errors, serialization and aliasing errors, inadequate authentication, and violation of boundary conditions.

The following table presents the taxonomy of security flaws with respect to genesis according to Landwehr et al. [70]:

<table>
<tr><td rowspan="10">Genesis</td><td rowspan="7">Intentional</td><td rowspan="3">Malicous</td><td rowspan="2">Trojan horse</td><td>Non-replicating</td></tr>
<tr><td>Replicating</td></tr>
<tr><td colspan="2">Trapdoor</td></tr>
<tr><td colspan="2">Logic/Time Bomb</td></tr>
<tr><td rowspan="3">Non-Malicious</td><td rowspan="2">Covert Channel</td><td>Storage</td></tr>
<tr><td>Timing</td></tr>
<tr><td colspan="2">Other</td></tr>
<tr><td rowspan="5">Inadvertent</td><td colspan="3">Validation Error (Incomplete/Inconsistent)</td></tr>
<tr><td colspan="3">Domain Error (Including Object Re-use, Residuals, and Exposed Representation Errors</td></tr>
<tr><td colspan="3">Serialization/Aliasing (including TOCT-TOU)</td></tr>
<tr><td colspan="3">Identification/Authentication Inadequate</td></tr>
<tr><td colspan="3">Boundary Condition Violation (Including Resource Exhaustion and Violable Constraint Errors)</td></tr>
<tr><td colspan="3">Other Exploitable Logic Errors</td></tr>
</table>

Table 1: Flaws by genesis [70].

Landwehr et al. then continues with the taxonomy of security flaws with respect to time of introduction. Flaws are said to be introduced during either development, maintenance, or operation. Introduction of flaws during development is further divided into introducing flaws during the requirement, specification, and design phase, and during source and object coding.

The last part of the taxonomy classifies flaws with respect to location. In this case, security flaws are first divided into being located in software or hardware. The security flaws in software are then divided into operating system, support, and applications.

The following tables present the taxonomy of security flaws with respect to time of introduction and the taxonomy of security flaws with respect to location [70]:

Time of Introduction	During Development	Requirement/Specification/Design
		Source Code
		Object Code
	During Maintenance	
	During Operation	

Table 2: Flaws by time of introduction [70].

Location	Software	Operating System	System Initialization
			Memory Management
			Process Management/ Scheduling
			Device Management (including I/O, networking)
			File Management
			Identification/Autentication
			Other/Unknown
		Support	Privileged Utilities
			Unprivileged Utilities
		Application	
	Hardware		

Table 3: Flaws by location [70].

One or more flaws or vulnerabilities may be exploited by a *threat*, an instantiation of which is called an *attack*. I.e. an attack is an attempt to violate the security controls of a computing system. A successful attack is in turn called an *intrusion*.

There also exist some efforts trying to create taxonomies of threats, *threat agents* (the perpetrators or intruders) and intrusion techniques. One early example is the taxonomy of threat agents presented in 1980 by J.P. Anderson, which partitions threat agents into three different categories according to the table shown below [69]:

	Penetrator Not Authorized to Use Data/Program Resources	Penetrator Authorized to Use Data/Program Resources
Penetrator Not Authorized Use of Computer	Case A: External Penetration	N/A
Penetrator Authorized Use of Computer	Case B: Internal Penetration	Case C: Misfeasance

Table 4: General cases of threats [69].

Another intrusion taxonomy, presented by Neumann and Parker, uses nine different classes of intrusions [68]:

Class of intrusions	**Description**
NP1 External Misuse	External misuse concerns misuse external to the system
NP2 Hardware Misuse	Hardware misuse includes passive and active side effects
NP3 Masquerading	Masquerading pertains to techniques for spoofing and inpersonation
NP4 Subsequent Misuse	Subsequent misuse relates to preparations for futher misuse
NP5 Control Bypass	Control bypass refers to circumvention of intented controls
NP6 Active Resource Misuse	Active resource misuse concerns violating system and data integrity

Table 5: Intrusion Taxonomy by Neumann and Parker [68].

Class of intrusions	**Description**
NP7 Passive Resource Misuse	Passive resource misuse includes violations against data confidentiality
NP8 Misuse via Inaction	Misuse via inaction relates to knowing a potential problem without doing anything about it
NP9 Indirect Aid	Indirect aid referes to off-line planning of an intrusion

Table 5: Intrusion Taxonomy by Neumann and Parker [68].

The taxonomy by Neumann and Parker was later refined by Lindqvist and Jonsson [67]. With the experience from realistic intrusion experiments performed at Chalmers University of Technology, seven different subclasses were added to the taxonomy. NP5, bypassing intended controls, is divided into password attacks, spoofing privileged programs, and utilizing weak authentication. NP6, active misuse of resources, is divided into exploiting inadvertent write permissions, and resouce exhaustion. Finally, NP7, passive misuse of resources, is divided into manual and automatic browsing.

Updated Categories		
NP5 Bypassing intended controls	Password attacks	Capture
		Guessing
	Spoofing privileged programs	
	Utilizing weak authentication	
NP6 Active misuse of resources	Exploiting inadvertent write permissions	
	Resource exhaustion	
NP7 Passive misuse of resources	Manual browsing	
	Automated searching	Using a personal tool
		Using a publicly available tool

Table 6: Updated intrusion taxonomy by Lindqvist and Jonsson [67].

Apart from the taxonomies presented above, there exist several others, sometimes using other classifiers. Some examples are the CERT intrusion taxonomy by Howard [66], and an intrusion taxonomy based on audit traces presented by Kumar [89].

4.2 Intrusion Detection

In the previous section an intrusion was loosely defined as a successful attack. As the terms related to intrusions are central to this thesis some stricter definitions are needed.

When it comes to the definition of intrusion I chose to use the definition of intrusion that is used in the Ph.D. thesis presented by Ulf Lindqvist [71]:

"**Intrusion** is a successful event from the attacker's point of view and consists of

1. an attack in which a vulnerability is exploited, resulting in
2. a breach which is a violation of the explicit or implicit security policy of the system."

When it comes to defining intrusion detection, two main approaches exist. The first approach only includes the detection element of intrusion detection. This can be exemplified by the definition presented by the NSA Glossary of Terms in Security and Intrusion Detection [72]:

> "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."

The second approach adds the element of response to intrusion detection. This can be exemplified with the definition presented by Amoroso [33]:

> "Intrusion detection is the process of identifying and responding to malicious activity targeting at computing and network resources."

The term "intrusion detection" can be somewhat misguiding. The reason for this is that the word intrusion relates to a successful attack, but, even unsuccessful attacks are (and should be) recognized by the intrusion detection system. The more accurate term *attack detection* has been suggested.

A computerized automated system that helps in the intrusion detection process is called an *intrusion detection system* (IDS). The IDS, in turn, is managed by the *site security officer* (SSO), i.e., the security administrator. Lindqvist relates the functionality of the intrusion detection system to the use of traditional preventive mechanisms with the following instructive figure [71]:

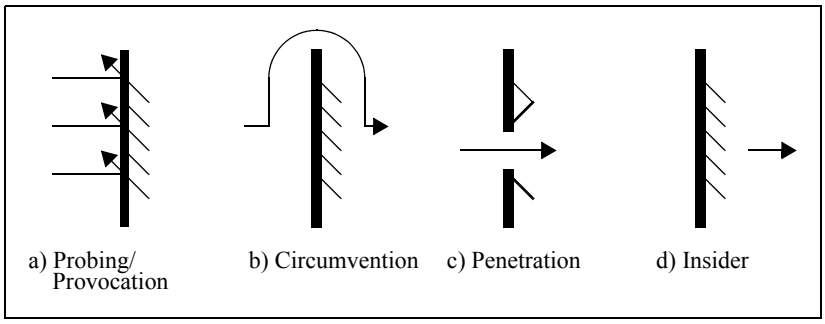

Figure 1. Events that should trigger IDS response [71].

To favor interoperability between different intrusion detection systems (and parts thereof), a framework called Common Intrusion Detection Framework (CIDF) was proposed [63]. In CIDF, the intrusion detection system is composed of four basic parts called event generators (E-box), analysis engines (A-box), storage mechanisms (D-box), and response units (R-box). The event generator collects events and sends them to the analysis engine. The analysis engine analyzes events according to its detection method. The storage mechanisms store information needed by the intrusion detection system. The response unit is used to initiate some kind of manual or automatic response.

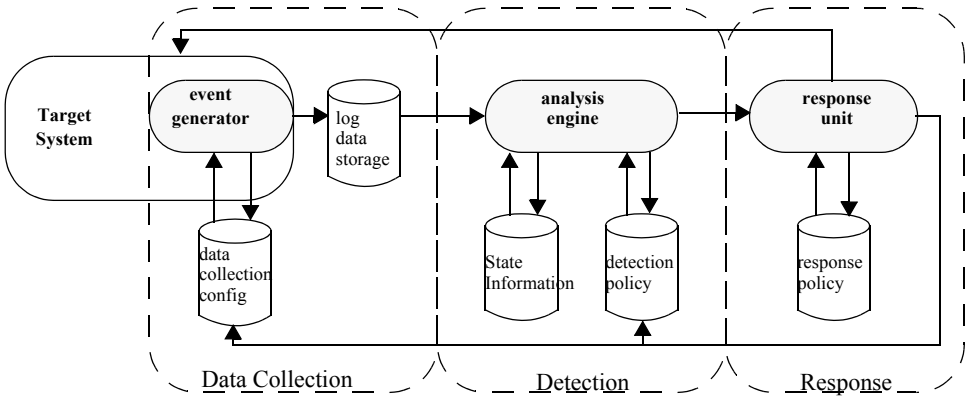

Figure 2. Generic model of IDS components based on CIDF [73].

Additionally, CIDF describes means for the different boxes to communicate, including Generalized Intrusion Detection Objects (GIDO), protocols, and Application Programming Interfaces (API). In theory this would mean that event generators from vendor A, B and C should be able to send events to an analysis engine from vendor D. So far, in practice, using different intrusion detection systems in concert has turned out to be a hard problem to solve.

Apart from CIDF, there exist several other standardization efforts within (or related to) the area of intrusion detection. Some examples are IDMEF, IDIP, SASL, IDXP and BEEP [7][63][64][65]. Of these, the Intrusion Detection Message Exchange Format (IDMEF) will be analyzed further in this thesis as it provides a standard format to alert messages generated by IDS [7].

There also exists several recent efforts of creating a taxonomy of intrusion detection systems. Axelsson used time of detection, granularity of data-processing, source of audit data, response of detected intrusions, locus of data-processing, security, and degree of inter-operability as classifiers [74]. Debar et al., on the other hand, presented detection method, behavior on detection, audit source location, and usage frequency as classfiers in their paper “Towards a taxonomy of intrusion-detection systems” [75]. This taxonomy is then later revised by Debar et al. and the classifier “detection paradigm” is added to the taxonomy (see figure 3) [100].

The main detection methods are generally divided into anomaly and misuse detection [3]. This differs from the taxonomies above. Axelsson uses the term anomaly detection, but has changed misuse detection in favor of signature detection [74]. Debar et al. have instead opted for the terms behavior and knowledge based detection [75]. However, all versions mean approximately the same, namely:

- Anomaly detection refers to a detection method that compares current behavior with profiles of previous behavior, primarily using statistical means [3][98]
- Misuse detection refers to a detection method that compares current events, e.g. log entries, against known bad events, often represented by intrusion signatures stored in a knowledge base [3]

Additionally, Axelsson defines a third method called signature inspired, which takes both the normal behavior of the system and the behavior of the perpetrator into account [74].

When it comes to response methods, the most common terms are passive or active response [3]. Axelsson uses these terms, while Debar et al. uses active response and passive alerting. Once again, they both mean approximately the same thing, i.e.:

– Passive response refers to response methods that only alert the SSO about the attack or intrusion, i.e. it does nothing to try to mitigate it

– Active response refers to response methods that try to mitigate the harm resulting from the attack or intrusion

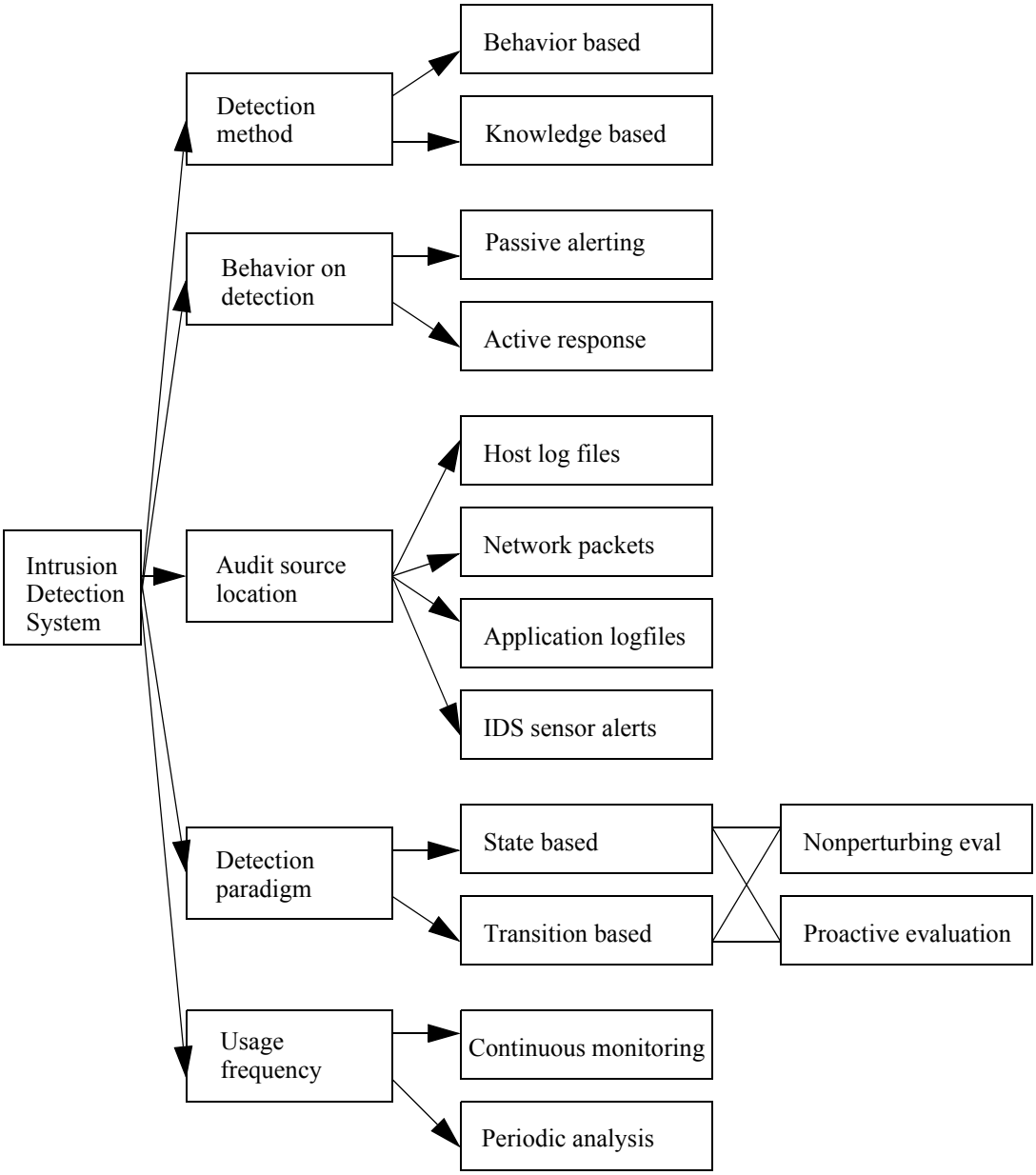

Figure 3. The revised taxonomy by Debar et al. [100].

Depending on the type (network data or logfiles) of the audit data, IDSs are commonly divided into being either network, host, or application based. Debar et al. adds one additional category called IDS sensor alerts, while Axelsson includes both application based and IDS sensor alerts in the host based category. (In this case we equate IDS sensor alerts with logs from network equipment like firewalls and routers.) Bace, on the other hand adds a category called target based [3]. This category includes sensors that regularly compute cryptographic integrity checks to detect alterations on remote systems (e.g. like Tripwire [40]).

We are not going to analyze all of the remaining classfiers for IDS included above. Instead, we conclude that there does not exist an agreed upon taxonomy for IDS and that more research on this subject is needed.

Although the use of different intrusion detection methods has the possibility to mitigate the harm resulting from attacks and intrusions, it is not without its problems. Three research questions deserve special attention; how to secure the IDS itself, how to improve the effectiveness, and how to improve the efficiency of intrusion detection.

The security of the IDS is very important. This includes protecting all phases in the intrusion detection and response process, as well as the protection of distributed security policies etc.

The effectiveness of intrusion detection referes to the ability to correctly classify audit events as being intrusive or not. This most often includes a binary decision, but the decision could also include a percentage of certainty.

Efficiency, on the other hand, refers to the ability to effectively process incoming events. A low efficiency may lead to an IDS being flooded with audit data, resulting in lost or late events.

If we return to the ability to correctly classify audit events as being intrusive or not, the following four situations may occur:

	Intrusive Event	Non-intrusive Event
Intrusive Decision	True Positive	False Positive
Non-intrusive Decision (including missed events)	False Negative	True Negative

Table 7: Four different possible outcomes.

To measure the effectiveness and efficiency of current IDS, it is very important that there exist relevant metrics. The most commonly used metrics being the detection rate and the false alarm rate (scores above T are flagged as intrusive) [97]:

- Detection rate = (#attack sessions with score > T) / (Total #attack sessions)
- False alarm rate = (#normal sessions with score > T) / (Total #normal sessions)

There exists several research efforts that have studied various metrics used to measure the effectiveness and efficiency of IDS.

In 1998, DARPA, for the first time, in a large scale, evaluated research projects in the field of intrusion detection [90][93][94][97]. The idea was to find promising technologies for integration in a military-wide demonstration of intrusion detection technology, called the IA:AIDE project (Information Assurance: Automated Intrusion Detection Environment). The DARPA evaluations used raw data, captured from a live network, during the generation of test data. The raw data was then sanitised and synthesised for both off-line and on-line evaluations. The purpose of this was to make the evaluations as "real" as possible.

The work performed by IBM aimed at providing a comparative evaluation of intrusion detection techniques [92]. The workbench should enable comparison of the respective efficiency of the prototypes developed at IBM. Two general requirements were defined. The workbench must be able to generate a wide variety of normal usage patterns. The workbench must also be able to cope with a heterogeous environment, i.e. an environment with several different IDS. Three methods of simulation were suggested. Using a generic session building tool, using test suites provided by the development laboratories, and recording and replaying normal data generated by actual users.

One of the main motivations for the work performed at UC Davis was to decide to what extent one can rely on IDS and to what extent one must rely on other security mechanisms [95][96]. Puketza et al. suggests the use of equivalence partitioning. This means that the test cases should be partitioned into classes, and then a representative subset of the test cases will be created by selecting one or more intrusions from each class. The classes should be selected such that, within each class, either the IDS detects every intrusion, or the IDS does not detect any intrusion. Then one test from each class can be selected to represent the class in the final set of test cases. The selected test cases should then be tested during three different test phases, with different conditions. The basic functional test phase tests if the IDS in ideal conditions can detect each attack. The variance-test phase tests if the IDS can detect variations of the attack. And finally, the stress-test phase tests whether or not the IDS will respond to the attack during extreme conditions. These three tests later evolved into intrusion identification tests, resource usage tests, and stress tests. Chung et al. also presents a methodology for automatically transforming a sequence of intrusive scripts into a set of parallel intrusive scripts, with the possibility to simulate an advanced intruder who might parallel the intrusive commands and issue them from different sources in an attempt to reduce the risk of being detected by an IDS [91].

4.3 Intrusion Correlation

The increasing use of intrusion detection systems gives rise to an even more increasing stream of alerts that need to be taken care of. All of these alerts have a relatively high chance of being a false alarm. To make things worse, most intrusion detection systems have their own alert format, making it hard to create a common picture when analyzing alerts from heterogeneous sensors. To clarify, the current problem for intrusion detection systems is:

> *To be able to analyze and react to a huge volume of alerts, generated in different formats, coupled with a relatively high false alarm rate*

One way to remedy the above problem is to use different correlation methods. Correlation is not new in intrusion detection research; however, when correlation is mentioned it is often unclear what is really meant. According to Amoroso [33], this is also the reason why research in this particular area has been scarce:

> "Correlation processing has not been the focus of much research in computer and network security. We believe that this stems from its lack of clear definition. It also stems from the tendency to equate complex statistical analysis with correlation - an equating we believe to be misleading".

Amoroso then continues by defining intrusion correlation, as follows [33]:

> "Intrusion correlation refers to the interpretation, combination, and analysis of information from all available sources about target system activity for the purposes of intrusion detection and response".

We will refine this definition slightly, by noting that intrusion correlation is (at least) two different problems; the older problem of intrusion event correlation, and the more recent problem of intrusion alert correlation:

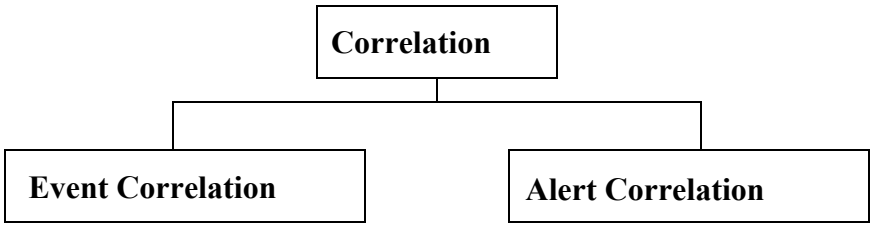

Figure 4. The problem of intrusion correlation can be divided into at least two sub-problems; that of intrusion event correlation, and that of intrusion alert correlation.

The biggest difference between intrusion event correlation and intrusion alert correlation is that intrusion event correlation analyses neutral events, meanwhile intrusion alert correlation analyses identified misuse or anomalies. This relation between events and alerts is also stated in the IDMEF specification [7]:

> "Generally, every time an analyzer detects an event that it has been configured to look for, it sends an Alert message to its manager(s). Depending on the analyzer, an Alert message may correspond to a single detected event, or multiple detected events. Alerts occur asynchronously in response to outside events".

4.3.1 Intrusion Event Correlation

We define intrusion event correlation as:

> *Intrusion event correlation refers to the interpretation, combination, and analysis of neutral events from all available sources, about target system activity for the purposes of intrusion detection and response.*

Amoroso [33], mostly focuses on intrusion event correlation. In this thesis we will focus on the problem of intrusion alert correlation. However, the two problems are interrelated

and some of the ideas presented by Amoroso will be referenced here. Three different types of intrusion correlation are identified [33]:

- Single-session versus multiple-session network correlation
- Real-time versus after-the-fact correlation
- In-band versus all-band information

Single session network correlation refers to the correlation of information related to a stream of packets between two endpoints. Three important correlation related problems are identified for single session network correlation:

- Association processing (i.e. context management)
- Time-to-store (i.e. memory management)
- Bi-directional build (i.e. keeping state information)

Amoroso then adds five correlation related problems related to multiple session network correlation:

- Remote sessions
- Same source or destination end points
- Time inconsistencies
- Patterns in unrelated sessions
- Connecting unrelated sessions

According to Amoroso, the basic difference between real-time and after-the-fact correlation is that real-time analysis can not use a "look forward" function, i.e. use the ability to "jump forward in time" during batch-processing of stored event logs for instance.

With respect to in-band versus all-band information correlation, Amoroso defines in-band as consisting of "computing and networking activity that is inherent to the target system". All other relevant information is defined as out-of-band information. Together these two form all-band information. Using out-of-band information is not without problems. These include [33]:

- A common format for all-band information
- Different confidence levels for all-band information
- Subjective interpretation by the SSO

4.3.2 Intrusion Alert Correlation

The recent interest in intrusion alert correlation has resulted in a number of publications from several different research groups.

IETF is developing the Intrusion Detection Message Exchange Format (IDMEF) draft standard [7].

IBM has developed an Aggregation and Correlation Component (ACC) [23]. Two kinds of correlation relationships handle duplicate alerts and consequences of alerts. Aggregation relationships are used to group alerts into attack situations according to various selection criteria.

M2D2 is an attempt to create a formal data model for the alert correlation process [32]. Information gained from vulnerability scanners are correlated together with alerts from intrusion detection systems. Additional leverage is gained from information about the monitored system and information about the security tools used.

SRI International has presented several papers concerning a probabilistic method implemented in the EMERALD architecture [10][12][25]. Hierarchical correlation is performed using four different concepts: (alert) feature overlap, feature similarity, minimum similarity, and expectation of similarity.

The M-Correlator is an extension to the EMERALD architecture, adding information about the relevance with respect to the target operating system, priority with respect to criticallity of the target and importance of the alert type, as well as incident ranking based on Bayes networks [26].

MIT Lincoln Laboratory presents an alert correlation model similar to the one presented by SRI International [24]. The main difference is the use of automatic optimization of correlation parameters using training sets of tagged alerts.

The "Prerequisite" Approach presents the concept of a hyper-alert, including information about facts (about the intrusion), prerequisites, and consequences of intrusions [27][28][29][30].

The MIRADOR project includes an intrusion alert correlation component called CRIM [31][34]. Incoming alerts are clustered and merged, and processed into intrusion scenarios.

We define intrusion alert correlation as follows:

> *Intrusion alert correlation refers to the interpretation, combination, and analysis of intrusion alerts, together with information external to the intrusion detection system, with the purpose of intrusion alert refinement and intrusion scenario building.*

As is clear from earlier sections, different intrusion detection methods/systems are preferable in different situations and in different environments. This leads to a situation where the best result from intrusion detection systems, should be achieved when employing several different intrusion detection systems throughout the security domain.

Furthermore, it can be argued that deployment of diverse sensors within a single security domain provides a more complete coverage of the attack. However, sensor diversity has resulted in a new difficulty, namely, interpreting the reports from numerous sensors using diverse approaches. As a result, the potential benefit from sensor diversity is partly lost, because, for example, it is difficult for the SSO to ascertain which reports pertain to the same or to different incidents.

Intrusion correlation provides a number of potential advantages in the operation of an *enterprise sensor suite*. The most obvious benefit is the reduction in the number of alerts that a security officer must address. When considering a *single* sensor, a correlation engine can reduce alert volume by organizing numerous alerts that are part of an ongoing attack. In the case of *heterogeneous* sensors, a correlation engine should recognize when reports from multiple sensors refer to the same incident. Another benefit of using correlation is that it can enhance detection capability, and give a more complete picture of attacks that an individual sensor may observe only partially. In addition, reports from several sensors employing diverse analytical techniques may reinforce each other and therefore enhance the confidence of the detection, thereby minimizing the number of false positives and false negatives.

4.3.3 Research related to Intrusion Alert Correlation

This section surveys different research efforts related to IAC. The main purpose is to identify common features in the surveyed methods.

4.3.3.1 Intrusion Detection Message Exchange Format (IDMEF)

Most of the papers summarized in this survey are focused on the correlation of alert messages conforming to IDMEF (Intrusion Detection Message Exchange Format) [7]. It is therefore due time for a short presentation of the IDMEF (work in progress) model and its different (sub) classes (see figure 5).

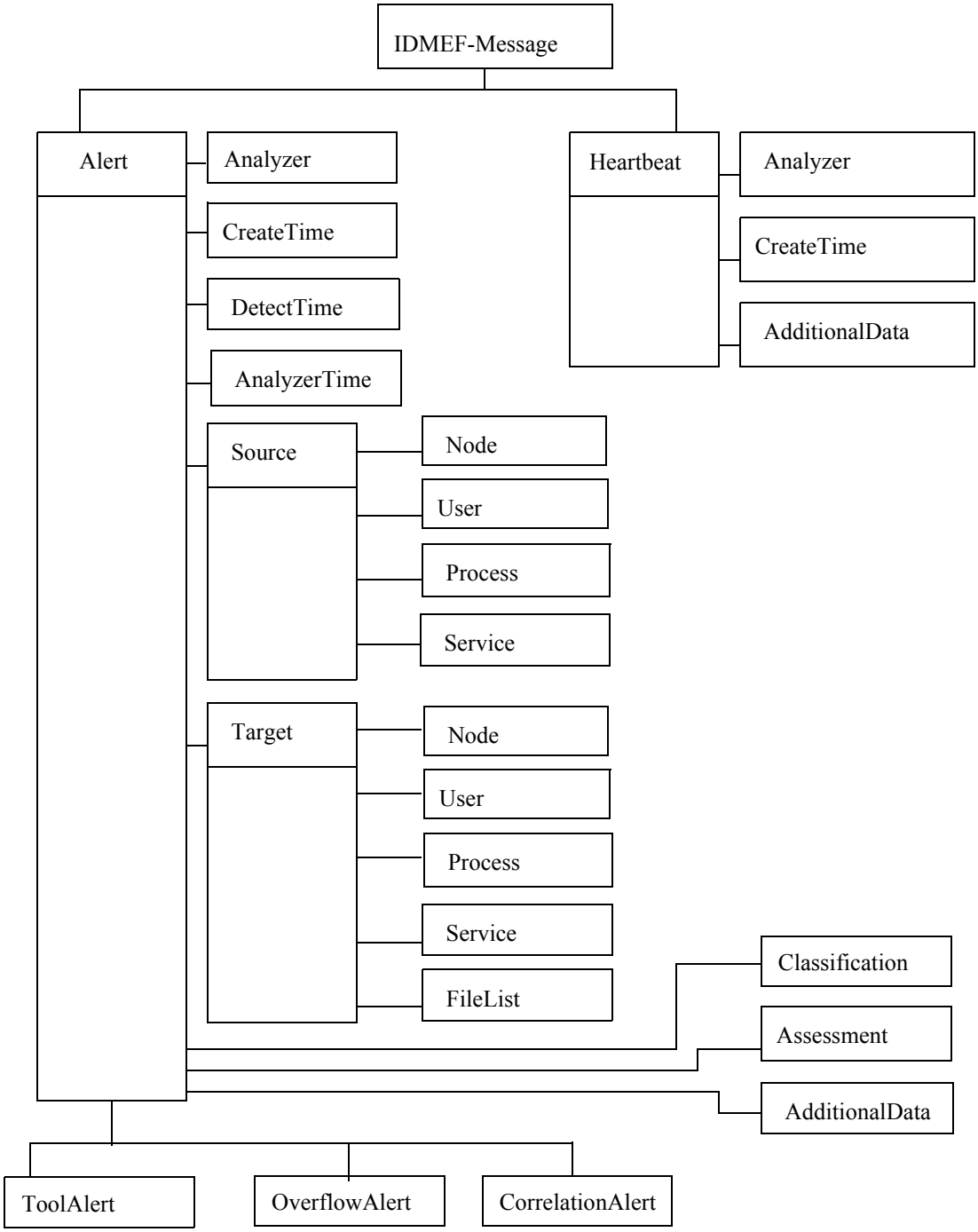

Figure 5. A simplified version of the IDMEF model as of January 30, 2003 [7].

From the figure above it can be seen that an IDMEF message can be either an alert or a heartbeat message. An alert message may be specialized further into a tool, overflow, and correlation alert.

The Alert class includes one attribute (ident[1]) and several aggregate classes described below. Alert Ident is unique within the sensor, and to make the alert unique within the intrusion detection infrastructure, it needs to be correlated together with the analyzerid (in the analyzer class presented below).

The specialized alert class called ToolAlert includes three aggregated classes (Name, Command, and Alertident). The purpose of the tool alert is to add information about identified attack tools (scripts etc.) generating the events that produced earlier alert messages.

The specialized alert class called OverflowAlert includes three aggregated classes (Program, Size, and Buffer). The purpose of the overflow alert is to add information to buffer overflow attacks, i.e. the size of the contents in the buffer, the actual contents itself, and the program that the buffer overflow tried to execute.

The most interesting specialized alert class is the CorrelationAlert, and it includes two aggregated classes (Name, and Alertident). The purpose of the correlation alert is to provide a means to group alerts together. This is accomplished by adding the name of the correlation method together with a list of the previously seen alerts (alertident and optionally analyzerid).

The Analyzer class includes seven attributes (analyzerid, manufacturer, model, version, class, ostype, osversion) and two aggregate classes (Node, and Process). The analyzer class has two main purposes during correlation. Firstly, Analyzerid (together with ident values) may be used to uniquely specify which sensor produced a specific alert message. Secondly, a combination of the other attributes may be used to identify which "type" of sensor generated the alert message. The first information is important when performing inter sensor correlation. The latter information is important for the translation of "alert name" into "alert class" (see section 5.2.2 below).

The three Time classes (detect, create, and analyzer time) each have one attribute (ntpstamp) representing different aspects of date and time. First comes the DetectTime, which contains the time the original event record (or the first in a composite event) was detected. Second comes CreateTime, which is the time when the alert, or heartbeat, was created. Last comes the AnalyzerTime, which should contain a last time stamp, just before the alert is sent for further processing, i.e. we have: detect time <= create time <= analyzer time. Of these three classes, only analyzer time is required. The time classes have two main purposes during correlation. Firstly, making it possible to pinpoint the actual time of the

[1] The class and attribute names within parentheses are further presented in [7].

intrusion, and secondly, making it possible to correlate alerts that have appeared close in time (together with additional constraints).

The Source class includes three attributes (ident, spoofed, and interface) and four aggregate classes (Node, User, Process, and Service). The main purpose of the source class is obviously to identify the offending party/parties. The problem of fabricated "source" information is evaluated using the spoofed attribute, which can be set to three different distinct values (unknown, yes, and no). The interface attribute is a special case and should be used by a multi-homed network intrusion detection system to indicate the interface registering the event.

The Target class includes three attributes (ident, decoy, and interface) and five aggregate classes (Node, User, Process, Service, and FileList). The purpose of the target class is to specify possibly affected entities. The decoy attribute in the target class is used to indicate if the target is a decoy, and it can be set to the same values as the spoofed attribute in the source class.

The Assessment class includes no attributes, but three aggregate classes (Impact, Action, and Confidence). The Impact class is used to specify the impact of the event on the target(s). The Action class is the only class that handles information about intrusion response. The Confidence class is used to indicate the analyzers assessment of the correctness of the generated alert message.

The Classification class includes one attribute (origin) and two aggregate classes (Name, and Url). The main purpose of the Classification class is to provide an alert name, and indicate where additional information may be found. The origin attribute is used when correlating heterogeneous sensors generating different alert names for the same or similar combination of events. Origin can be set to four distinct values (unknown, bugtraqid, cve, and vendor-specific). Vendor specific alert names require specific handling to map the alert name to a generic alert name or alert class.

The AdditionalData class includes two attributes (type, and meaning), where type specifies the type of data provided according to ten fixed values (boolean, byte, character, date-time, integer, ntpstamp, portlist, real, string, and xml). The purpose of this class is to provide means to extend the basic IDMEF data model. Obviously, this will be a problem for heterogeneous alert correlation.

4.3.3.2 Aggregation and Correlation Component (ACC)

IBM has developed a prototype called the aggregation and correlation component (ACC) [23]. The model is organized into four different layers:

- The Probe layer contains information about the signature generating the alert, as well as the probe sending the alert
- The Target layer contains information about the target(s). Multiple targets may be represented by a headcount
- The Source layer carries information about the possible source(s) of the alert. This may include an indication about the source being spoofed
- The Detailed target layer includes additional target specific information

The purpose of the aggregation and correlation algorithm is to form groups of related alerts using a small number of relationships. Two different relationships are used:

- Correlation relationships
- Aggregation relationships

The correlation relationship is further divided into two different relationships, called Duplicates and Consequences, which are used to group alerts according to explicit rules. These groups are then processed together.

Duplicate relationships between alerts are defined in advance using a duplicate definition file. The list of attributes of the newly arrived alert must be equal to the attributes of a previous alert to be considered a duplicate. A severity value, found in the duplicate definition file, is then used to set the severity level of the correlated alert. Additionally, repeated instantiations of the same duplicate group may generate a new signature (event of class X repeated Y times) with a higher severity (initial severity times Y).

Consequence relationships between alerts, i.e. if we get an alert of type X, then we should also get an alert of type Y, are defined in advance using a consequence definition file. New alerts may start a consequence chain, and they are also checked to see if they are a consequence of a previous alert. If a wait period expires without the required consequence happening, a simulated alert representing the missed alert is created with the severity value from the consequence definition file.

1	Alert/Source/Target	A	S	T
2.1	Source/Target	*	S	T
2.2	Alert/Target	A	*	T
2.3	Alert/Source	A	S	*
3.1	Source	*	S	*
3.2	Target	*	*	T
3.2	Alert	A	*	*

Table 8: Attack situations; from specific to more general [23].

The aggregation relationship is used to aggregate alerts into seven different (attack) situations according to table 8 above. The different situations are defined in advance using a situation definition file, consisting of four different terms: alert class (A), source token (S), target token (T), and severity level. The severity level is later used to identify the most severe attacks, while alert class, source token, and target token are tested for seven different combinations of equality.

When the sum of the severity levels of the aggregated alerts exceeds the severity level in the situation definition file, an alarm is generated. If several situations reach the alarm level, only the most specific situation is allowed to generate an alarm. The values specified in the wildcard fields in the table above are collected into an alert property list. (One list for situation-2 alarms, two lists for situation-3 alarms.)

Additional features include special handling of authorized scans, including comparison against previous scans, and re-evaluation of situations using two different kinds of triggers: situation re-evaluation, and multi-situation assessment. Re-evaluation of the severity value may be done if the generated entries in the alert property list(s) exceed a predefined threshold, or if the alert property list(s) contains any of a predefined set of sublists. Multi-situation assessment concerns re-evaluation of severity values when related situations exist.

4.3.3.3 M2D2

M2D2 uses a formal data model to include external information in the alert correlation process [32]. Four different information types are handled: information about the monitored system, information about known vulnerabilities, information about security tools (vulnerability scanners and intrusion detection systems), and information generated by the

security tools, e.g. scans and alerts. A relational database is used to store information from IDS and scanners, together with product information from the ICAT vulnerability database [78].

The monitored system is modelled from both the "network node" and "product" perspective. The nodes in the network are modelled using Vigna's TCP/IP networks topology formal model [99]. Some extensions are made, e.g. adding mappings between hostnames and IP addresses. The products are modelled as quadruples using vendor_id, product_id, version_id, and type. Additionally, a product is classified into four different main product classes according to their use. This includes OperatingSystem, Server, LocalApp, and Other. The server class may be further divided into server specific classes, e.g. httpServ and ftpServ.

Identified vulnerabilities, which are said to affect configurations, are modelled using information from the ICAT database [78]. A mapping function is used to translate non-CVE/CAN vulnerability names into CVE/CAN vulnerability names [86]. Furthermore, the prerequisites (called requirements) for exploiting a vulnerability are modelled using three (access) classes: Remote, RemoteUser (remote login with user access privileges), and Local. The consequences of using the vulnerability are in turn modelled using three classes: CodeExec, DoS, and Info.

Security tools are modelled as either IDS or vulnerability scanners, both of which can create reports (which in turn may be mapped to vulnerabilities). IDS are host or network based and use misuse or anomaly detection methods. Scanners only affect hosts.

According to Moring et al. [32], an alert is modelled as "a kind of event, since it reflects the state of an IDS. Such definitions are compliant with the previous definitions and enables to elegantly model the fact that alerts produced by an IDS may be the events from other tools' point of view. In other words, current IDS are low level event consumers and alert producers but future IDS may be alert consumers and high level alert producers". The current model covers events at several different layers: IPv4, TCP, UDP, HTTP network events, and web server log events.

The paper then continues with examples covering different aggregation techniques. One function groups all host and network based alerts, referring to a named target, together. Another function has the ability to identify all systems potentially vulnerable to an incoming alert. An alert similarity function is used to identify IDS, reactive to the same set of "basic" events. By comparing the number of incoming (true or false positive) alerts, referring to the same event(s), with the number of false negative alerts (from non-reactive IDS), different measures can be taken. The trustworthiness of an alert can be adjusted up or down, and malfunctioning IDS's can be identified for example.

The model is very flexible and offers several possibilities for aggregation of alerts [32]:

- caused by the same event
- referring to the same vulnerability
- caused by events belonging to the same TCP/IP session
- based on temporal relations

4.3.3.4 EMERALD

SRI has introduced a probabilistic approach to alert correlation [10][12][25]. To be able to handle heterogeneous alerts, a generic alert template (not IDMEF) is used. The correlation is then performed hierarchically. "Threads" are used to correlate alerts relating to the same incident on the same sensor ("same" sensor, alert class, source, and destination address). "Security incidents" are then composed of the same incidents correlated over several sensors. It is then possible to create "correlated attack reports" by correlating over several alert classes.

The correlation approach considers four different concepts called feature overlap, feature similarity, minimum similarity, and expectation of similarity.

When a new alert arrives it is first compared to a previous alert, called a meta alert. Features that the alert and the meta alert have in common, called the feature overlap, is then evaluated according to feature specific similarity functions, returning a similarity value between 0 and 1. Relevant features include the source of the attack, the target (hosts and ports), the class of attacks, and timing information.

The attack class is generated using a probe/sensor specific translation function from alert signature to alert class. The similarity between different alert classes is defined in advance using a matrix of similarity containing 14 different alert classes (invalid, privilege-violation, user-subversion, denial-of-service, probe, access-violation, integrity-violation, system-environment-violation, user-environment-violation, asset-distress, suspicious-usage, connection-violation, binary-subversion, and action-logged).

Expectation of similarity is a situation specific weighting factor and it may differ depending on the circumstances, e.g. attacks using spoofed IP addresses have a low expectation of similarity concerning the source address, while other attack types have a high expectation of similarity.

Minimum similarity is also a situation specific value that expresses necessary but not sufficient conditions for correlation. Certain features can be required to match exactly (minimum similarity for these is unity) or approximately (minimum similarity is less than unity, but strictly positive) for an alert to be considered as a candidate for fusion with another.

Setting minimum expectation for some features to unity (not normally recommended) causes the system to behave like a heuristic system that requires exact matches on these features.

To conclude, the overall similarity between two alerts is zero if any overlapping feature matches at a value less than the minimum similarity for the feature (features for which no minimum similarity is specified are treated as having a minimum similarity of 0). Otherwise, overall similarity is the weighted average of the similarities of the overlapping features, using the respective expectations of similarity as weights.

4.3.3.5 M-Correlator

The Mission Impact Intrusion Report Correlation System (M-Correlator) is an extension to the EMERALD system developed at SRI International. The M-Correlator uses information external to IDS alerts in the following way [26]:

- Filters are used by alert subscribers to filter out low-interest alerts classes
- A relevance score is calculated with respect to the prerequisite of the attack (with regard to OS version etc.) and (user supplied/nmap) information about the target environment
- A priority score is calculated with respect to the criticality of the target, and the importance of the alert class (User supplied information.)
- Incident ranking is performed with respect to a combination of alert priority and alert relevance (calculated above), and the probability-of-success value in the alert message
- Related alerts are combined using an attribute-based clustering algorithm

An implementation of Bayes networks is used to calculate incident ranking. The main branches of the incident rank tree consist of outcome (from the perspective of the sensor), and priority and relevance (with respect to information stored in the knowledge base).

4.3.3.6 MIT Approach

A similar approach to the one developed at SRI has been chosen by MIT Lincoln Laboratory [24]. To perform correlation, alerts are partitioned into five attack categories called discovery, scan, escalation, denial-of-service, and stealth. New alerts are possibly added to existing intrusion scenarios after the evaluation of the probability that one attack category is followed by another, the time difference between alerts, and the proximity of source IP addresses. The parameters used for correlation are optimized using training sets of tagged alerts.

A new alert is compared to the latest alert in all existing scenarios. The new alert is then joined with the scenario with the highest probability score, providing the score is above a minimum threshold. Otherwise, the alert opens up a new scenario.

The time-based quantity is defined by a sigmoid function (a step function at x=0), using two parameters, where different transitions between alert classes will generate a different shape of curve.

The IP address based quantity is computed in two steps. First, the number of 1 bits in a common IPv4 subnet mask is computed. Second, a transition specific function, using five parameters (for 0, 8, 16, 24, and 32 bits in common), is then used to produce the proximity value.

4.3.3.7 The "Prerequisites" Approach

The idea behind this approach is that earlier stages of an attack are often preparations for later ones [27][28][29][30]. This leads to a correlation system using an artefact called "hyper-alert" that includes information about attack types: facts, prerequisites (for intrusion), and (possible) consequences (of intrusion). The free variables in "prerequisite" and "consequence" are attributes in "fact", which in turn encode information from the alert.

"Fact" is represented as a relational schema where the tuples representing a hyper-alert include a "begin time" and "end time". The instantiation of a single hyper-alert can be restricted using time constraints; for example, the time difference between the earliest begin time and the latest end time must be lower than a fixed duration.

Instantiated hyper-alerts may then be correlated if the consequence(s) of one hyper-alert fulfils some (or all) of the prerequisites of the second hyper-alert. An hyper-alert is said to prepare for another if the consequences in the first hyper-alert implies at least one prerequisite in the second hyper-alert, and the end times of all consequences are earlier than the begin time of the prerequisite(s). Additional restrictions may be applied using time spans and attack types.

The correlated hyper-alerts are then used to create hyper-alert correlation graphs. More formally, a "hyper-alert correlation graph CG = (N, E) is a connected graph, where the set N of nodes is a set of hyper-alerts, and for each pair of nodes n_1, n_2 in N, there is an edge from n_1 to n_2 in E if and only if n_1 prepares for n_2" [27].

Three different methods are suggested for the handling of hyper-alert correlation graphs:

- Adjustable graph reduction. Hyper-alerts of the same type may be aggregated if their respective begin and end times are within a set time interval

- Focused analysis. The attributes of the hyper-alerts must satisfy specified constraints, using AND, OR, and NOT operators when comparing attribute values against constants, to be considered for hyper-alert correlation graphs
- Graph decomposition. The attributes of the hyper-alerts must satisfy specified constraints, using AND, OR, and NOT operators when comparing attribute values against each other, to be considered for hyper-alert correlation graphs

4.3.3.8 CRIM

CRIM is an implementation within the MIRADOR project, which implements alert clustering, alert merging, and alert (scenario) correlation [31][34].

Alerts, conformant to IDMEF, are sent to an alert (data)base management function where they are transformed and inserted in a relational database as facts. Redundancy is avoided, and previously seen entities (e.g. targets) are not inserted again.

Alerts corresponding to the same occurrence of an attack are identified, using similarity expert rules, and linked into clusters.

The similarity expert rules are focused on classification, time, source, and target. In this case, the similarity check will consist of series of comparisons between tuples in the database. Two alerts are similar with respect to classification if the original alert names are related to the same (MIRADOR) attack name. Two alerts are similar with respect to time if the difference between the respective detecttimes are within a certain limit. Similarity between sources and between targets is a harder problem. Correspondence tables are used for look-ups between IP addresses and host-names, and port numbers and service names. Both source and target can consist of a node, a user, a process and a service. Tuning is needed to find the optimal similarity function. Additionally, both time similarity and source and target similarity are dependent on the alert classification (creating “expected” similarity requirements).

The information pertaining to the various alerts in the alert cluster are then merged into a new global alert. New alerts will then either create new global alerts, or update (and even merge) existing ones.

A likelihood coefficient is created for each global alert with respect to the number of non-reactive IDS. Additionally, alert clusters cannot be updated after a predetermined interval and thus become stable clusters. Also, merging a new alert with a global alert requires conflict identification functionality, checking that the new alert is consistent with the existing ones.

The alert classification of the global alert becomes the union of the correlated alerts. Sources and targets are merged into unique entities. The detecttime and createtime of the

global alert become an interval, and analyzertime is bound to the global alert (not to the correlated alerts).

The global alerts are then sent for post-condition/pre-condition correlation, with the purpose of creating scenario alerts. (The correlation rules are automatically generated off-line, modulating attacks manually described by five different fields: attack pre-condition, attack post-condition, attack scenario, detection scenario, and verification scenario.)

The correlation function may also cope with missing alerts. Transitive relationships are evaluated using virtual alerts (making up for the missing one). If all correlation restrictions apply, an IDMEF compatible scenario alert is created despite the missing alert.

The plan for the future is to use the post-condition/pre-condition functionality to be able to predict, and possibly hinder, the most probable next step of the unfolding intrusion.

4.4 Intrusion Tolerance

Traditional security countermeasures have focused on intrusion prevention (e.g. using authentication services, access control and firewalls), and intrusion detection (including intrusion response). However, despite the increased use of these countermeasures, there will always (i.e. in a foreseeable future) be intrusions.

There exist several reasons for this observation, however, one of the most important ones is the following statement (I do not know who used this first though):

> "You must find and fix all vulnerabilities to make the system secure, but you only need to find one flaw to make an intrusion".

Furthermore, even if the intrusions are detected by an intrusion detection system, and an appropriate response is taken, there will always be a time interval during which the attacked service cannot be trusted. The reason for this could be false alarms, batch processing or just instantaneous effects of the attack. Thus, there exists a demand for services that can deliver their specified service even during various attack scenarios.

A promising way to approach this problem is intrusion tolerance. An intrusion tolerant system is capable of self-diagnosis, repair, and reconstitution, and continues providing service to legitimate clients (with possible degradation) in the presence of intrusions [36].

4.4.1 From Dependability to Intrusion Tolerance

The dependability research community is separate from the security community. However, lately, these two research areas have started to get more closer together with transfer of concepts from one area to the other.

Dependability is defined as "that property of a computer system such that reliance can justifiably be placed on the service it delivers" [49]. Some of the most important aspects of dependability have been visualized in the Dependability Tree [49]:

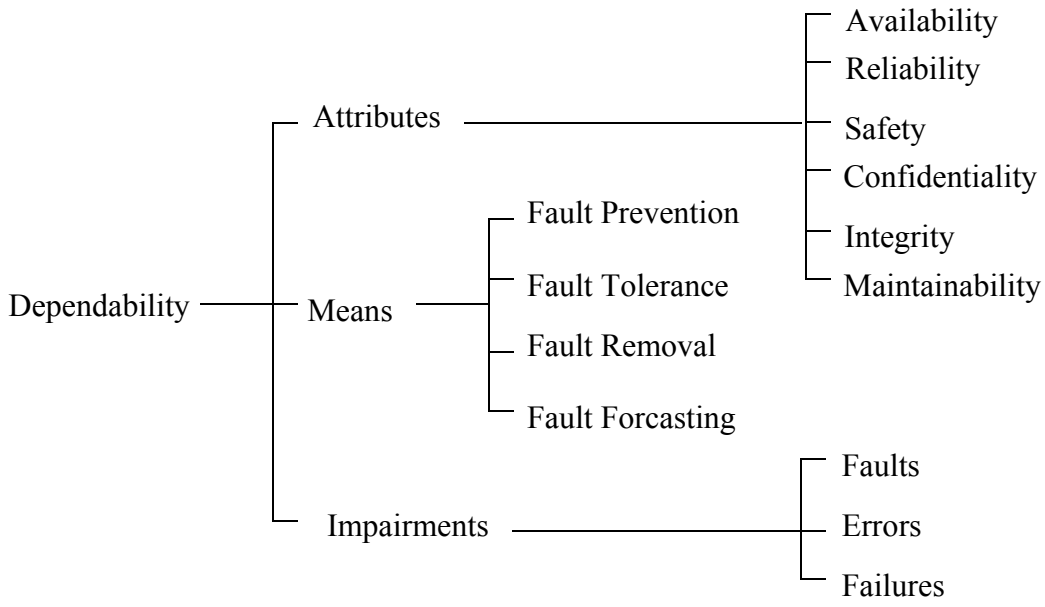

Figure 6. The Dependability Tree [49].

The recent MAFTIA project transfers the means of dependability (above) into the security domain by equating intrusion and vulnerability with fault. Furthermore, they present the following six methods as being meaningful in the security domain [49]:

- Vulnerability prevention: how to prevent the occurance of introduction of vulnerabilities
- Intrusion prevention: how to prevent the occurance of intrusions
- Intrusion tolerance: how to provide a service capable of implementing the system function despite intrusions
- Vulnerability removal: how to reduce the presence (number, severity) of vulnerbilities

- Vulnerability forecasting: how to estimate the presence, creation and consequences of vulnerabilities
- Intrusion forecasting: how to estimate the presence and consequences of intrusions

The rest of this section will focus on intrusion tolerance, which has been defined as[36]:

> "An intrusion tolerant system is capable of self-diagnosis, repair, and reconstitution, and continues providing service to legitimate clients (with possible degradation) in the presence of intrusions".

Next, we will present some research efforts concerning implementing intrusion tolerance into various services.

4.4.2 Research related to Intrusion Tolerance

4.4.2.1 An Intrusion Tolerant IDS

One part of the MAFTIA project addresses the security of the IDS itself [51]. It is observed that the IDS, as a distributed system, suffers from the same threats as other distributed componets within the MAFTIA architecture, plus three additional threats specific to IDS [51]:

- An adversary is trying to tamper with the communication channels
- A denial-of-service attack can be run against an event analyzer
- An attacker has found a way to assume control over an event analyzer

To hamper the threat against the communication channel they propose four different solutions, the use of: heartbeat messages, hash chains, authentication codes, and data encryption .

A denial-of-service attack against an IDS may be remedied using continues monitoring of the IDS itself. Two different solutions are proposed. The first uses a (immortalizer) program that monitors the integrity and aliveness of the IDS. The second uses continual online testing to verify that all expected alerts are generated [51].

To remedy the last threat presented above, i.e. that the attacker has assumed control over an event anlyzer, they present two different methods: using a behavior based meta-IDS, detecting strange behavior from the IDS, and replication of the event analyzer together with some kind of majority voting of the end result [51].

Additional intrusion tolerant features, not specific to intrusion detection, are handled by other parts of the MAFTIA architecture (not presented here) [79].

4.4.2.2 An Intrusion Tolerant Web Service

One current research project is called SITAR (A Scalable Intrusion-Tolerant Architecture for Distributed Services) [48]. The SITAR project is rather interesting, as it has taken almost the same approach as our current work. To our knowledge, the two projects have not had any knowledge of each other. The SITAR project models system behavior, using a state transition model, when the system is under a specific attack using a given system configuration [48]. The web-based architecture is built around five important components; proxy servers, acceptance monitors, ballot monitors, adaptive reconfiguration, and audit control. The proxy servers act as front-end, reverse proxies, to a set of COTS web servers.

Every proxy machine includes a network based intrusion detection system, watching for external attacks as well as the behavior of the other proxies. Depending on the threat level, different restrictions apply for e.g. user authentication, degree of redundancy (one request sent to several COTS), and increased auditing. These functions are implemented using an adaptive reconfiguration module (ARM) that is responsible for the overall security configuration. The acceptance monitors validate replies from, and detects intrusions in, the COTS servers. The reply and the result of the acceptance test are then sent to a ballot monitor. The ballot monitors decides on the final reply, if redundant requests are sent to COTS, using majority voting or Byzantine agreement [80]. The result is then announced to the proxy. The audit control module, audits trails from the other components, as well as performs regular diagnostic tests. The diagnostic tests are both positive and negative, and the test results are sent to the ARM. The SITAR project is planning to implement and test the architecture described above.

4.4.2.3 An Intrusion Tolerant Private Key Storage

The ITTC (Intrusion Tolerance via Threshold Cryptography) project at Stanford University has focused on an intrusion tolerant private key storage [52]. The private keys are protected during generation and storage using a set of share servers in combination with threshold cryptography. Using a combinatorial t-out-of-k algorithm the private keys never need to be reconstructed at a single site. Instead each client (e.g. a web server protected by SSL) contacts a randomized set of t of the share servers. The contacted share servers then perform their respective part of the requested cryptographic function, either sign, decrypt, or hash-sign, and returns the reply to the client. When the client has received all k replies it combines them to create the output from the (whole) cryptographic function.

4.4.2.4 An Intrusion Tolerant Authentication Service

The development of this intrusion tolerant authentication service was partly supported by the Delta-4 project [36][77]. Two variations are presented using symmetric and asymmetric encryption techniques. The asymmetric techniqe was not implemented and it is not included here.

A major problem with a central authentication server is that it easily becomes a single point of failure. This problem has been remedied by the separation of duties according to the following (symmetric) scheme [77]:

- The authentication server functionality is split between N different sites
- Each site is administered by its own SSO
- Each SSO has a personal master smartcard, which has been initialized using one mother smartcard
- The user's smartcard memory is split into N areas, with each area being initialized and by one of the N master smartcards

The user smartcard registration process requires the participation of every SSO, which in turn means that a single SSO cannot create false smartcards.

The authentication protocol is then performed as follows [77]:

- The user site broadcasts its authentication request to all the authentication sites
- Each authentication site performes independently the authentication by using the secret data
- Each authentication site broadcasts its decision to the other authentication sites
- Each one votes on the decisions
- Each one sends the final result to the user site
- The user site votes on the replies to be sure it has been successfully authenticated by a majority of the authentication sites; if so, it can send other types of requests to the authorization server in order to access remote servers

The protocol above is able to handle intrusions into a minority of the sites.

4.4.2.5 An Intrusion Tolerant Authorization and Directory Service

The development of this intrusion tolerant authorization and directory service was partly supported by the Delta-4 project [36].

Using the ticket received from the authentication servers above, the client contacts the authorization server with an authorization request. Two different requests exist. Either a request for a service, or a request for access to an object. The information needed to make an access decision is stored in a directory service (X.500) and it includes information about [36]:

- Access control lists
- Reference information (which gives direct access to the object or server)
- Additional information according to object type

The access control information is distributed among several servers to enable separation of duty. The level of replication is dependant on the secrecy of protected object. Information that is not sensitive can be replicated on every site, meanwhile sensitive information is protected using a secret sharing threshold algorithm that only needs participation of a majority of the sites.

Finally, the authorization protocol uses majority voting in the same way as is done by the authentication service presented above. A successful authorization results in an additional ticket for further contact with remote objects or servers.

4.4.2.6 An Intrusion Tolerant Persistent File Service

The development of this intrusion tolerant persistent file service was partly supported by the Delta-4 project [36].

Two operations will be presented, file write and file read. We assume that the client has previously contacted the authorization and directory service. A successful authorization results in the authorization server sending one ticket back to the client and one ticket to the intrusion tolerant persistant file service. The client ticket includes the fragmentation key, which is used to partition the file during write and read operations. The server ticket includes the hashed file name together with the authorized operations and a user certificate.

If we assume that a user has created a file on his workstation and wants to store it on the intrusion tolerant persistant file service, the following protocol is used [36]:

- The file is partitioned into pages of a fixed size. If necessary, padding is used to fill empty space on the last page
- A cryptographic checksum is added to every page
- The page is encrypted using a symmetric encryption algorithm in CBC mode

- Every page is then partitioned into a fixed number of fragments, where the individual fragments get their names using a hash algorithm over the fragmentation key (from the client ticket), the file name, the index of the page, and the index of the fragment
- A tag containing a hash over the file name and the fragment name is added to every fragment
- The client broadcasts the fragments in random order on the local subnet
- The storage servers uses a pseudo random algorithm locally to decide if the fragment should be stored on the server or not. The algorithm includes information about the required level of redundancy, and storage space left on the server

When the client, at a later time, wants to read a file from the storage servers, the client broadcasts the names of the wanted fragments (also in random order). The storage servers hashes the fragment name together with the hashed file name (from the server ticket), and compares it with the TAGs that are stored together with the fragments.

The security of the intrusion tolerant persistent file service is dependent on that only a minority of the servers are compromised. Additional security is added by sending the fragments in randomized order over the network.

4.4.2.7 Intrusion Tolerant Enclaves - An Intrusion Tolerant Group Communication Service

The original Enclave group communication system has a centralized architecture with one group leader serving a set of group members [81][82]. This system has implemented several security measures, but the central groupleader is still a single point of failure. To remedy this problem a new intrusion tolerant version of Enclaves is presented [76].

In the intrusion tolerant version of Enclaves, the single responsibility of the central group leader is split between N groupleaders (servers). A successful intrusion would need to compromise more than f leaders according to the following formula [76]:

$$3f + 1 \leq N$$

This, in turn, requires that the intrusion tolerant Enclave system must be able to manage secure group communication in spite of up to f leaders being compromised. To accomplish this, three problems need to be addressed: authentication of group members, coordination of leaders, and group-key management.

To make the authentication of group members resilient of up to f faulty leaders, a group member needs to authenticate against $2f + 1$ leaders. (Each group member authenticates using a symmetric key that is only known by the group member and each specific leader.)

The final authentication decision is then dependant on the decision of $f+1$ non-compromised leaders. A successful authentication results in the generation of a session key for further communication.

The purpose of coordinating the leaders is to make sure that more than $f+1$ leaders agree on a successful authentication before letting the group member join the group. The following broadcast based leader coordination protocol is used when leader L_i has successfully authenticated A (the notation $<>\sigma_i$ denotes a signed message) [76]:

After successful authentication of A,
L_i sends <Propose, i, A, $n_i>\sigma_i$ to all leaders

After receiving $f+1$ valid <Propose, j, A, $n_j>\sigma_j$
from different leaders, L_i sends <Propose, i, A, $n_i>\sigma_i$
to all leaders if it has not already done so

When L_i receives $n-f$ valid <Propose, j, A, $n_j>\sigma_j$
from $n-f$ distinct leaders, L_i, accepts A as a new member

Figure 7. The leader coordination protocol [76].

Each time a group member joins (or leaves) the group a new group key needs to be generated. The proposed group key management protocol uses a variant of the secret sharing algorithm presented in [83]. Using this scheme, each leader is able to learn its share of the secret without knowing the (whole) secret. Any combination of at least $f+1$ leaders is then enough to recompute the group key.

5 A Generic Intrusion Alert Correlation Procedure

In this section, we present a generic alert correlation procedure. It identifies common solutions to common (IAC) problems, identified during the survey of IAC research efforts presented in section 4.3.3.

5.1 A Generic Procedure

After a synthesis of the surveyed IAC research efforts we propose the following generic IAC procedure:

1. Create alerts using a standard format
2. Transform alerts into tuples in a relational database
3. Correlate duplicate alerts within sensor
4. Correlate duplicate alerts across sensors
5. Refine alert information using out-of-band information
6. Build attack scenarios and identify the next step

Each of the six steps of the generic IAC procedure are described further below.

Create alerts using a standard format.
All of the surveyed research groups are using some kind of standard format for their alert reporting. Furthermore, most groups are using some variant of IDMEF to suit their needs. The use of markup languages, in this case XML, simplifies the correlation process in that it is easy to identify attributes that are up for correlation. Some parts of IDMEF are still potentially hard to correlate between heterogeneous sensors, such as information in the Classification and AdditionalData classes.

Transform alerts into tuples in a relational database.
The next step in our generic procedure is to transform the standard format alerts into tuples in a relational database. Various relations are created for the purpose of alert correlation and aggregation. The relations typically mimic the underlying structure of the standard format used for alert reporting, e.g. IDMEF.

Correlate duplicate alerts within sensor.
When the database has been populated with alerts, the first part of the correlation process identifies alerts pertaining to the same attack instance, reported by a single unique sensor.

These alerts are duplicates in a broad sense, and they are often the result of a sensor generating too primitive alert reports, i.e. a sensor missing internal IAC possibilities. The most common attributes to evaluate during this step are:

- Same sensor
- Same time
- Same attack class
- Same source
- Same target

All of the above attributes will require a high similarity value or equality to be considered for intrusion alert aggregation.

Correlate duplicate alerts across sensors.
The second step of the correlation process also identifies alerts pertaining to the same attack instance, but this time, the correlation is performed across sensors. The additional values of this step are sensor reinforcement and complementarity. Compared to within sensor correlation, this step often only requires that the alert class and time attributes are similar enough. The reason for this is that different sensors will have (hopefully small) time differences, and possibly also report attacks using different attack classes. The most common attributes to correlate are:

- Same (or close in) time
- Same (or related) attack class
- Same source
- Same target

Refine alert information using out-of-band information.
So far, the alert correlation process has only considered information internal to the standard format alerts. This may result in correlated alerts with little or no connection to the protected environment. To refine the correlated alerts, out-of-bound information (i.e. information external to the alert) is used during additional processing. The most common out-of-band information is:

- Environmental information. Information about the architecture, including hardware, operation system and running services

- Resource classification. Information about the severity of a breach in a service
- Vulnerability information. Additional information about already identified vulnerable services (using various vulnerability scanners)
- Attack information. Additional information about vulnerable architectures, including vulnerable hardware, operating systems and services
- Post mortem analysis. Additional controls with the purpose of validating attacks
- Attack visibility information. Information about which intrusion detection systems should have seen the attack

The additional information is used to re-evaluate the correlated alert reports, including the possibility that the attack succeded, the severity of the attack, and the priority of the corralated alert report.

Build attack scenarios and identify the next step.
The last part of the correlation process identifies attack scenarios. An attack scenario covers different attack classes, reported over a length of time, across all sensors. During this step it is important to evaluate the possibility that one attack class will follow another, and the possibility that an attack spreads from system to system.

This concludes our proposed IAC procedure. The following section will present some of the most important help constructs needed during the IAC process.

5.2 Help Constructs

Many of the surveyed IAC models have stumbled on the same problems, e.g. related to the high number of alert messages and the diversity of sensors. These are normally solved using various help constructs. The most important ones are:

- Similarity metrics
- Alert classes
- Merging and conflict resolution functions
- Look-up functions

These help constructs are further described below.

5.2.1 Similarity Metrics

During alert correlation, the different features of a (standard format) alert are compared to the corresponding features of another alert. Some correlation approaches require a perfect match between compared features, while others tolerate small deviations, i.e. it is enough if the features are similar. Additional ways to correlate alerts could be by evaluating feature separation, including the recognition of common time gaps between events, and feature covariance, including the recognition of things varying together, e.g. increasing IP and/or port numbers [13].

The following are some of the most commonly compared features together with the similarity metrics used:

- Time. The most common metric used is based on time intervals with “hard” thresholds, i.e. the compared alerts must be close enough in time. It is also possible to use “soft” intervals that decrease the possibility of a positive correlation when the interval increases. It is important to remember that an intrusion alert message generated in IDMEF may include up to three different time values, i.e. detect, create, and analyzer time
- Reporting sensor. During the correlation of duplicate alerts within a single sensor, the sensor must be uniquely identified. Using IDMEF, equality is required for the analyzerid attribute. During the correlation of duplicate alerts across sensors this problem is a non-issue
- IP addresses. Five different cases exist (care must be taken to the possibility of IP addresses being spoofed):
  1. Equality on IP addresses
  2. IP addresses within the same subnet
  3. IP addresses have enough common bits in the address mask
  4. IP addresses are within known groups of IP address (e.g. in the same domain, known bad hosts, or known own hosts)
  5. No restrictions on IP addresses whatsoever (which can be used during correlation of distributed attacks)
- Port numbers. Usually requires a perfect match between the compared lists of port numbers, or an overlap that is high enough (possibly with respect to the size of the port lists)

– Alert classification. During the correlation of duplicate alerts within a single sensor, a perfect match may be required. During the correlation of duplicate alerts across heterogeneous sensors the requirement of a perfect match may potentially be relaxed

5.2.2 Alert Classes

Alert classes are used to group similar alert types together. The main reason for this is to remedy the problem of non-uniform alert naming, i.e. the lack of a functioning alert taxonomy. Several alert taxonomies exist; however, they are neither complete, nor consistent with each other [86][88]. The alert class is most often generated through a lookup function from a sensor specific alert name to a generic alert class name (internal to the correlation engine). Additionally, during scenario correlation, a transition matrix may be used to map the possibility that one alert class will follow another. For example it is more probable that a port scan is followed by a buffer overflow attack, than the other way around.

Example. During the implementation of Snort into the EMERALD architecture a special mapping function was developed that translated Snort native alert format (i.e. snortid, and revision) into EMERALD internal format (alert class). Every snortid-revision combination was analysed with respect to EMERALD alert class. During the time, this meant that almost 1,500 snort alerts were mapped onto 14 different EMERALD alert classes.

5.2.3 Merging and Conflict Resolution Functions

A successful correlation will result in the merging of information from two or more alert messages, and the new correlated alert will become a superset of its parts (i.e. the merged alerts). Sometimes the information in the various alerts may be conflicting. This may require special handling in the form of precedence and priority. Additionally, we identify two different kinds of merging: irreversible and reversable merging. If the merging process is reversible, it is possible to separate the correlated alert into its parts: If the merging process is irreversible, then this is not possible.

Example. If two alerts, containing two different source IP addresses, are merged into one list of source addresses, and the same is done with their respective port lists, then the process is irreversible (without additional means).

If we return to IDMEF, the main classes of information that need to be merged during correlation are time, source, target, and alert classification. Depending on the implementation this may result in various operations on lists, or in adding or updating tuples in a relational database. In CRIM, this process is handled as follows [34]:

- Time. An interval is formed for each of DetectTime and CreateTime. The oldest time value is placed in the new alert's DetectTime and CreateTime. The most recent time values are then placed in AdditionalData
- Source/target. Either existing tuples are updated with addtional information, or new ones are created
- Alert classification. The alert classification is the union of its parts (with unique values)

5.2.4 Look-up Functions

Finally, various look-up functions are used during the correlation process. This includes functions to translate between IP addresses and host names, port numbers and service names, and vice versa. Additionally, time functions are used to enable a global time base.

One important factor during correlation is the freshness of information. If the look-up is not performed close enough to the event, alert or correlation, then stale information may be fed into the correlation process.

6 Intrusion Correlation Experiments

This section presents three different intrusion detection and correlation experiments.

Firstly, a router (firewall) based intrusion detection system is implemented. The purpose is to evaluate its applicability to intrusion detection. A weakness in the logging mechanism is identified and analyzed with respect to intrusion alert correlation.

Secondly, a case study of heterogeneous alert correlation is presented. The purpose is to evaluate the effects of intrusion alert correlation using a live-traffic analysis. Snort alert messages are correlated together with alert messages from EMERALD. A ten-to-one reduction in the number of alert messages is accomplished.

Lastly, a log-based intrusion detection system is implemented. The purpose is to evaluate the possibility to use Open Source software for the creation of an alert correlation system. Some alert correlation results are presented.

6.1 Router Based Intrusion Detection

The router based intrusion detection experiment investigated the hypothesis that log data produced by packet filtering gateway routers (firewalls) constitutes a good source for traces of novel attack techniques that are not yet publicly known.

We also describe how such router logs could be used as an event source for real-time misuse detection sensors such as the EMERALD sensor suite. Cisco routers were used for filter experiments and the collection of large logs based on real network traffic, but the results apply to other router brands with similar filtering and logging functionality.

One of the problems with signature-based misuse detection sensors is to keep the knowledge base updated with new attacks. This is especially true for attacks not yet publicly known. Therefore, it is important for IDS designers to continuously investigate new attack traces and to write signatures that are as general as possible.

We believe that collected gateway/firewall logs constitute a good source for traces of novel attacks/probe techniques that are not yet publicly known. Analysis of large amounts of such logs could lead to the synthesis of signatures that could be incorporated into intrusion detection sensors. Alternatively, this log data can enhance the detection capacity of probabilistic systems [10].

6.1.1 Firewalls from an Intrusion Detection Perspective

6.1.1.1 Firewalls and Distributed System Security

The section describes some basic concepts of firewalls and distributed system security.

Distributed system security can be based on host-based security where every single computer is made secure, or network-based security where strategic points in the network are secured or, ideally, a combination of both. The problems with relying on host-based security include the number of systems that need to be protected, and that end-systems are often added and reconfigured dynamically. Even small companies can have their offices separated by a vast distance, making it very difficult to maintain host-based security. This has forced most organizations to implement network-based security instead. The main issue is to implement a security policy at strategic points in the network. In the early years of distributed systems, this meant to filter inbound and outbound traffic to external networks, that is networks beyond the organization's own administrative control.

Definition of Firewall: A firewall consists of one or more filtering devices. The main objective is to separate two or more networks by applying filtering of network traffic, and data, according to a predefined (network) security policy.

There exist two basic firewall techniques, called packet filtering and proxying.

Packet filtering mainly looks at information contained in the network and transport header of the datagram. This information tells the firewall which client and server programs (ports) that communicate. The end-systems talk directly to each other, and the filter does not interpret or control what is contained in the data portion of the datagram.

Proxying is a technique that uses a helper program located on an intermediate host, between the client and server programs. The proxy sits in the application layer, listens to client requests, contacts the destination server on behalf of the calling client, and sends the requested data back to the calling client. This means that there is no direct traffic between the client and the server. A proxy makes it possible to make filtering decisions on the contents of the data, and it is an excellent point to introduce authentication of users.

In the remainder of this paper, we focus on firewalls that are based on a router platform rather than a general-purpose computer. Typically, router-based firewalls only perform stateless packet filtering on a single-packet basis, although some router models can be equipped with special software to perform stateful (dynamic) filtering.

Compared to host-based firewalls, router-based firewalls normally have inferior hardware resources. For example, there is less memory and often no disk available. This has an impact on performance, as well as on security. As an example, logging is not recommended on routers already working near the top of their CPU capacity. This in turn means that intrusion detection analysis, via logging, is best suited on routers that are not heavily loaded. In addition, the logging that is actually done does not contain as much information as logs generated by host-based firewalls.

Another problem with router based firewalls is that they do not complain about faulty configurations. A single mistake can easily result in an access list that permits everything without the administrator noticing. One positive thing though, is that routers have less software and do not provide many services. This should make them less vulnerable to direct attacks.

Cisco routers are used for the experiment and examples in this thesis. Most of the examples require at least Cisco IOS version 11.2 or comparable (which supports the log-input keyword). There exist some relatively new features that will not be covered. The reason for this is that we want the thesis to be general enough to be applicable to a broader range of router-based firewalls (Cisco as well as other brands). However, some of the new features are relevant to security [55].

6.1.1.2 Firewalls as an Event Source for Intrusion Detection

Most discussions concerning intrusion detection in a firewall environment are focused on where to place the sensors, that is, outside or inside the firewall. Both methods have their pros and cons.

One reason to place an intrusion detection sensor outside the firewall is to be able to monitor "unfiltered" traffic. There is a good possibility that it is in this traffic that we will find the first attempts using novel attack types. However, placing the intrusion detection sensor outside the firewall also means that it will be at greater risk of being attacked itself.

This risk can be mitigated by placing the sensor directly on the firewall. However, placing a sensor on the firewall will probably lead to a substantial performance loss (depending on the analysis method used).

An alternative, in this case, would be to make better use of the firewall logs. As the logs are probably already being generated, analyzing them off-line will not affect the firewall performance significantly. Due to the firewall's first line of defence, we believe that collected gateway/firewall logs should constitute a good source for traces of novel attacks/ probe techniques that are not yet publicly discussed.

6.1.1.3 Cisco Router Log Structure

During the experiment we only looked at information generated by logged access-lists.

There are two different types of access-lists, standard and extended. The standard access-list has the following syntax [55]:

> Access-list number {permit|deny} source wildcard-mask [log|log-input]

This means that we make access decisions by only looking at the source address or network. If we want to make more fine-grained access-decisions, as is often the case in a firewall environment, we can use extended access-lists instead [55]:

> Access-list number {permit|deny} [protocol|protocol-keyword] source source-wildcard [port] destination destination-wildcard [port] [established] [log|log-input] [other options]

Both kinds of access-lists have the possibility to include the log or log-input keyword. If either of those keywords is used and the rule is triggered, a log message is generated. The log message can for example be logged locally, sent via SNMP, or sent via Syslog. In this experiment we will only consider the syslog approach.

The log message has the following syntax [55]:

Date:time:facility-severity-mnemonic:description

If the log-input keyword is used (instead of log) additionally the receiving interface of the router is logged. This information is useful when doing a trace-back.

A log message can be at most 80 characters long and it always begins with a %-sign. (Date and time must be specifically configured on the router.) The following is an example showing what kind of information that get logged by routers when using syslog for transport:

Jun 20 07:50:46.708: %SEC-6-IPACCESSLOGP: list 140 denied udp 192.168.7.5 (137) -> 192.168.5.255 (137), 1 packet

Syslog messages are grouped into facilities (depending on the sender) and every message has a severity level. Cisco routers by default use the syslog facility LOCAL7 and the severity level Warning. When a rule is triggered the following information is supposed to be available for analysis:

- Date and time
- Permit/deny
- Protocol identifier
- IP source address
- IP destination address
- TCP SYN
- TCP ACK/RST
- TCP source port
- TCP destination port
- UDP source port
- UDP destination port
- ICMP type numbers
- Additionally the address of the incoming interface of the router

This information is comparable to the “reduced log data set” presented by Northcutt [56], i.e. a minimum amount of information that is needed for an analysis.

6.1.1.4 The Quality of Router Generated Information

The quality of the log data is one of the true down sides of routers compared with host-based firewalls.

Firstly, logging is not a prioritized service in the router. This means that all violating packets will not generate log entries, “producing” a high false negative rate.

Secondly, when the log keyword is used together with an access-list, only the first packet that triggers the rule will generate an immediate log entry. To save disk space a time-out will then suppress logging of duplicate packets for five minutes. All duplicate log messages are then summarized in a new log message, containing the actual message as well as the number of duplicates that were received.

Thirdly, Syslog is UDP based, which means that the syslog message is not guaranteed to arrive at the syslog server.

To summarise, for intrusion detection analysis this means that we may lose log entries, as well as timing information for individual packets. Additionally, the combination of Syslog and UDP makes it possible to easily create successful attacks against confidentiality, integrity and availability.

6.1.1.5 How to Refine the Information

To make the available information as good as possible we have to refine it, i.e. we have to write our filtering rules in a way that suits intrusion detection. This pretty much comes down to writing the rules as specific as possible. The following basic techniques are available for filtering:

1. Permit/deny IP traffic between certain hosts or networks
2. Permit/deny TCP traffic between certain hosts or networks
3. Permit/deny UDP traffic between certain hosts or networks
4. Permit/deny ICMP traffic between certain hosts or networks
5. Permit/deny TCP traffic between certain programs/ports on certain hosts or networks
6. Permit/deny TCP traffic with the ACK/RST bits set between certain hosts or networks
7. Permit/deny TCP traffic with the ACK/RST bits set between certain programs/ports on certain hosts or networks

8. Permit/deny TCP traffic with no ACK/RST bits set between certain hosts or networks. This filters out e.g. the initial SYN packet
9. Permit/deny TCP traffic with no ACK/RST bits set between certain programs/ports on certain hosts or networks. This filters out e.g. the initial SYN packet
10. Permit/deny UDP traffic between certain programs/ports on certain hosts or networks
11. Permit/deny ICMP traffic of a certain type between certain hosts or networks

Of these, we will mostly be using techniques 6-11. The other rules are all too general to be of any real value in intrusion detection.

To know how we are going to use the above techniques we have to decide which part of the traffic that we are interested in, and that we can analyze. Porras and Valdes discuss the following ways to categorize TCP/IP event stream gateway traffic [39]:

- Discarded traffic, i.e. traffic not allowed by the access-lists
- Pass-through traffic, i.e. traffic allowed by the access-lists
- Protocol-specific traffic, e.g. traffic using TCP or UDP
- Unassigned port traffic, i.e. traffic to unblocked, not used, ports
- Transport management messages, i.e. TCP state information like SYN, SYN/ACK, and RST
- Source-address monitoring, i.e. monitoring traffic from known good or bad addresses
- Destination-address monitoring, i.e. monitoring traffic to given internal hosts
- Application-layer monitoring, i.e. analyzing traffic to specific ports

It is possible to write access-lists to filter out all of the above categories. However, we cannot use application layer monitoring in a deeper sense.

6.1.1.6 Detection Methods

Porras and Valdes also describe some measures concerning signature-based network traffic analysis [39]:

- Single event analysis, i.e. one event is enough for flagging intrusive behavior
- Threshold analysis, i.e. flags intrusive behavior with respect to accumulated behavior

- Application-layer transaction analysis, i.e. sweeping the data part of the traffic searching for malicious behavior. This is not possible with router log data
- Datagram parsing of transactions directed at high numbered unused ports, profiling traffic against well-known network applications (like telnet and ftp). This is not in an easy way possible with router log data

In our case it is only the first two methods that we can use, i.e. single event analysis and threshold analysis. Most of the following attacks that we reference in this thesis are further described in [56].

6.1.2 A Router Based Intrusion Detection Experiment

6.1.2.1 Experiment Setup

For the evaluation of homogeneous sensor correlation, we considered a couple of Cisco routers as sensors in the network:

- Cisco 1605. This system was set up in the test lab at SDL, SRI. One side of the router was connected to the rest of the lab network. The other side was connected to a laptop running RedHat Linux. The router was configured to log any traffic going through the router
- Cisco 7500. This system is a production system at SRI. Logs from the router were transferred to a lab PC for analysis. No changes were made to the configuration of the router, the log-entries were used as is

The laptop running RedHat Linux was installed with different tools, like nmap [59], and nessus [60]. The setup is presented in the figure below:

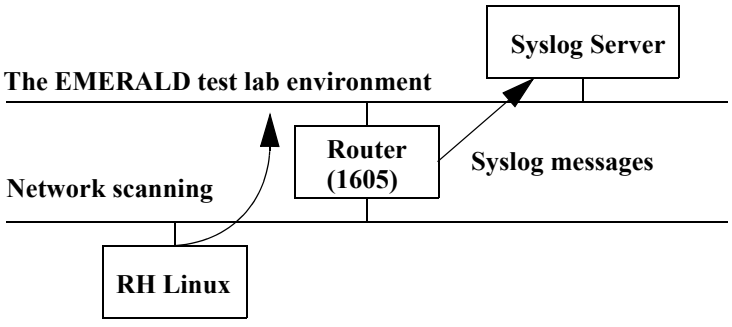

Figure 12. The experimental setup.

The prototype, called *RIDS*, has been tested on Redhat Linux version 6. RIDS has been developed using Cisco logs, without special firewall features. RIDS analyses syslog mes-

sages generated by ICMP, TCP, UDP, and ESP access-lists. No control is made to see if the connection is inbound or outbound, i.e. a source address could actually be a destination address. No control is made to see if the access attempt was permitted or denied.

6.1.2.2 Detection Methods Used

As was stated in section 6.1.1.6 above, two different detection methods are applicable for router based analysis: single event, and threshold analysis.

Single event analysis can be used to detect, e.g.:

- Spoofed connection attempts. Generating a bogus, often a non-existing or internal (as far as the router is concerned), source IP address with the purpose to traverse packet filters, create DoS attacks, and make forensics harder
- Connection attempts to known Trojan horses (default ports). Public lists with known "hot" ports could be used for this
- Connection attempts to known vulnerable ports (e.g. used in exploit scripts). Same as above
- The Land DoS attack. The attack uses the same IP address, and optionally the same port number, for both source and destination
- TCP-broadcasting. A session can only be established between one client and one server at a time, i.e. 192.168.1.255 over TCP is not a valid choice
- The echo-chargen attack
- ICMP and UDP echo request

The most important factors concerning threshold analysis are the number of connections during a certain time frame. Another factor of success is how the packets are counted, i.e. the selection criteria.

Ranum et al. describes the usage of histograms in the analysis engine of the NFR network intrusion detection system [54]. The histograms generate alerts based on previously unseen values, or values exceeding a specified range.

Vern Paxon elaborates on scan detection in the Bro intrusion detection system [57]. The Bro intrusion detection system uses two different kinds of metrics called distinct-peers, and distinct-ports. By counting the number of distinct destination addresses per source address an address scan can be detected. Likewise, by counting the number of distinct destination ports per source address a port scan can be detected. Bro can detect some forms of stealth scanning. This is possible because the algorithm does not care about the order of addresses/ports scanned, or timing information between successive connection attempts.

Northcutt mentions that they have tried time ranges of 1-3 seconds, 1-5 minutes, 24-hours, up to 60 days [56]. They have also had some success using the Shadow network intrusion detection system with a threshold of 5-7 connections to different hosts during a 1-hour time frame.

Threshold analysis should be able to detect, e.g.:

- SYN flooding
- Network mapping
- Port scans
- Random destination port scans
- Inverse mapping
- FTP bounce attacks
- Some stealth attacks
- Novel attacks (with repetitive behavior)

It is also very important to realize that we cannot find every attack that is out there. The following attacks are examples that cannot, in an easy way, be identified looking at router log data:

- Non-standard combinations of flags in TCP
- CGI-vulnerabilities
- The Christmas Tree DoS attack (UAPRSF)
- The WinNuke DoS (out-of-band data to TCP port)
- The Teardrop DoS attack (overwriting UDP fragments)
- The Ping-of-death DoS attack (large packets)

6.1.2.3 Issues concerning the Analysis of Router Log Data

During initial testing of the router in the test lab, it became apparent that routers (at least the ones that we tested) only generate a couple of log entries per second. In the logs that we analyzed, we found at most three log entries generated within the same second. This means, for example, that during portscans, only some packets/datagrams violating the current log policy will generate a log entry.

The high number of false negatives affects router based sensors in such a way that it is almost impossible to build state information looking only at the log entries. The reason for this is that most connections will reach the established state in less than a second. And even if it would take longer time to set up a session, the return packet/datagram would have to compete with other packets/datagram, arriving at the same time, to get logged. This in turn means that there is less reason for "syncronizing" the different log policies used at the router's interfaces.

Another interesting aspect with the router generated log entries, is the granularity of the generated log entries, i.e. the level of information presented. Filtering rules that do not specify restrictions on port (for TCP and UDP) or type (ICMP) level, will not generate log entries containing that information. The following is an example of log entries generated by filtering rules without port level restrictions:

```
Aug  8 16:07:47: %SEC-6-IPACCESSLOGP: list 101 permitted tcp a.b.c.253(0) ->
a.b.c.254(0), 359 packets
Aug  8 16:08:46: %SEC-6-IPACCESSLOGP: list 101 permitted udp a.b.c.2(0) ->
a.b.c.254(0), 1 packet
```

Both log entries are generated by accesslist number 101. The source and destination IP addresses are presented, but the port numbers are clearly missing, i.e. the parenthesis should have included the affected port number, not only a zero.

The same is true for log entries pertaining to ICMP (missing type information within the parenthesis):

```
Aug  8 16:07:30: %SEC-6-IPACCESSLOGDP: list 101 permitted icmp a.b.c.40 ->
a.b.c.253 (0/0), 76 packets
```

To generate port/type information, at least one reference to a port is needed per network interface.

It is also important to take the accumulation of log entries into account when designing an analysis method for Cisco log files. The strategy is said to work as follows:

1. If a packet gets logged, a timer is started
2. If a duplicate packet arrives within five minutes, only a counter will be increased
3. When the five minutes have passed a summary log message is produced, and the timer is reset to zero
4. If a new packet arrives within the five minutes a new summary will be produced. (We are back at step 2.) If not, no summary is produced and the timer is stopped

This strategy was verified for TCP:

```
Aug  8 14:58:18 130.107.12.254 141: Aug  8 16:07:47: %SEC-6-IPACCESSLOGP:
list 101 permitted tcp 192.168.1.253(0) -> 192.168.1.254(0), 359 packets
Aug  8 15:03:16 130.107.12.254 143: Aug  8 16:12:45: %SEC-6-IPACCESSLOGP:
list 101 permitted tcp 192.168.1.253(0) -> 192.168.1.254(0), 12 packets
```

We can see that the first log entry is produced 14:58:18, and that the next summary log entry is produced after approximately five minutes, i.e. 15:03:16.

The strategy was additionally verified for UDP:

```
Aug  8 16:13:44: %SEC-6-IPACCESSLOGP: list 105 permitted udp
192.168.2.240(0) -> 192.168.2.255(0), 1 packet
Aug  8 16:18:42: %SEC-6-IPACCESSLOGP: list 105 permitted udp
192.168.2.240(0) -> 192.168.2.255(0), 17 packets
Aug  8 16:23:40: %SEC-6-IPACCESSLOGP: list 105 permitted udp
192.168.2.240(0) -> 192.168.2.255(0), 2 packets
```

Once more, we can see the approximate five minute suppression of duplicate packets: 16:13:44, 16:18:42, and 16:23:40.

However, there seem to exist counter examples, like the following for TCP:

```
Aug 10 08:23:41: %SEC-6-IPACCESSLOGP: list 101 permitted tcp
192.168.1.253(4781) -> 192.168.2.40(111), 1 packet
Aug 10 08:23:47: %SEC-6-IPACCESSLOGP: list 101 permitted tcp
192.168.1.253(4781) -> 192.168.2.40(111), 3 packets
```

We can see that the summary log entry is produced only six seconds after the previous log entry, i.e. the first log entry is produced 08:23:41 and the summary 08:23:47.

Additionally, this anomaly was verified for ICMP:

```
Aug  9 09:21:18: %SEC-6-IPACCESSLOGDP: list 101 permitted icmp 192.168.1.253
-> 192.168.2.40 (8/0), 1 packet
Aug  9 09:23:00: %SEC-6-IPACCESSLOGDP: list 101 permitted icmp 192.168.1.253
-> 192.168.2.40 (8/0), 1 packet
```

To sum things up, there are three things to keep in mind when designing an analysis method for logs generated by routers:

- The false negative rate can be very high
- Log entries may be delayed for several minutes
- It is fairly easy to generate false log entries using spoofing techniques

To minimize the impact of false negatives, we must minimize the number of logging rules. I.e. log on as few interfaces and rules as possible, and preferably log on the interface with the lowest bandwidth (e.g. the WAN link).

6.1.2.4 Chosen Analysis Method

The analysis method was implemented in C [58]. Two tools were developed, one for "near real-time" analysis, the other one for batch processing. Only log entries generated by the external network interface of the router were analysed.

With the above knowledge in mind, we decided to go for the following method. Firstly, every log entry, representing a violation against the rule-base, is transformed into a internal format including the following variables:

- time
- status
- protocol
- type
- source IP
- source port
- destination IP
- destination port
- icmp
- number of packets

Secondly, single event analysis is made using a list of known bad port numbers. A match generates an alert of the format:

```
Warning: Possible TCP backdoor attempt on port 31
15:22:26 permitted tcp A.B.C.D 1648 (Ethernet0.E.F.G) A.B.C.D 31 #1
```

Thirdly, threshold analysis is performed using an event correlation algorithm that correlates the internal format into a linked list structure. The number of packets in the correlated event are incremented with individual values registered in each log entry.

At regular intervals the linked list structure is searched for correlated events reaching the set threshold. A sample run on a log file generated by the Cisco 7500 is presented in figure 13.

***** APPROXIMATE TIME: 15:04:47 *****					
*** Logged Connections ***					
Source IP	Destination IP	Port or Type	Port	Packets (proto)	Time
A.B.C.D					
	C.D.E.F				
		1560	23	#1 (tcp)	15:04:47
PREPROCESS WARNING: Priority _%SYS-5-CONFIG_I:_ is not supported					
WARNING: Possible TCP backdoor attempt on port: 31					
15:22:26 permitted tcp A.B.C.D 1648 (Ethernet0 E.F.G) A.B.C.D 31 #1					
WARNING: Possible TCP broadcast attempt: A.B.C.255					
15:33:27 permitted tcp A.B.C.D 33267 A.B.C.255 80 #1					
***** LAST PART OF LOG FILE *****					
WARNING: Possible port scan from A.B.C.D to 59 different ports					
*** Logged Connections ***					
A.B.C.D					
	G.H.I.J				
		0/0		#41 (icmp)	15:30:12
		32523	21	#5 (tcp)	15:08:15

Figure 13. Example output from RIDS when analyzing a router (sys)log.

The timing information is gathered from the analyzed syslog entries. Each log entry is correlated first according to its source IP address, then according to its destination address and destination port. This results in a set of "incidents" sorted on IP source address, destination address, and port number. Duplicate entries are aggregated and the accumulated number of packets is increased accordingly.

The preprocessor warning shows various problems with the input log file, like unsupported (not analyzed) protocols and misformatted entries.

Lastly, the warning messages either present the result from the single event analysis together with the offending log entry, or present the result from the threshold analysis together with the aggregated log entries.

6.1.3 Conclusions

As mentioned earlier, the experiment investigated the hypothesis that log data produced by packet filtering gateway routers (firewalls) constitutes a good source for traces of novel attack/probe techniques that are not yet publicly discussed. The examples above have shown that this is not always the case.

The false negative rate, which in this case means missed alerts, can be substantial. This may lead to a situation where it is not applicable to use the router log data at all. However, the alerts generated should be useful for both sensor complementarity and sensor reinforcement. Together with alerts from other sources, it should be possible to create more complete intrusion scenarios during the alert correlation process.

The automatically delayed log entries are best suited for batch processing, but may be interesting for "real-time" analysis depending on the time requirements. However, one draw-back is that the individual packets violating the security policy will not get time stamped.

Additionally, the possibility for a perpetrator to create false positives, which in this case means spoofed alerts, must be minimized. This can partly be accomplished using regular network security techniques using e.g. stateful inspection firewalls together with anti-spoofing.

6.2 Heterogeneous Sensor Correlation

In this section we will present a case study of the deployment of several sensors listening to live network traffic, and the application of correlation technology in a real network environment. Specifically, alerts from the Snort Open Source IDS are correlated together with alerts generated by the EMERALD IDS suite. The research prototype at SRI uses probabilistic techniques to aggregate diverse sensor reports according to a set of feature similarity functions.

The probabilistic correlation engine employed in this live traffic analysis has been presented in an earlier paper, and the interested reader should consult that report for a more in-depth treatment of the details of the approach [10].

6.2.1 Sensor Correlation

All alerts are reported using a standard alert template. The template supports the concept of alert threading, making it possible for a single sensor to consolidate a large number of alerts from a single attack into a smaller number of alerts and updates. The figure below shows an alert thread [10]:

```
Thread ID 69156 Class=portsweep   BEL(class)=0.994 BEL(attack)=1.000
2001-06-15 17.34:35 from x.y.148.33 ports 1064 to 1066
duration=0.000 dest IP a.b.30.117
3 dest ports: 12345{2} 27374{3} 139
```

Figure 14. EMERALD alert thread.

Systems such as Snort generate a very large number of messages for certain attacks (e.g., portsweeps), presenting a difficult management problem for the security officer, as well as possibly burying more important alert messages [8]. SPADE mitigates this difficulty to a degree [13], but early on we identified the need to infer synthetic threads as an important capability of a correlation engine. This is achieved by enforcing high minimum expectation similarity (described below) on the sensor itself, that is, the thread must come from a single sensor, the attack class, as well as the source and target (IP and ports).

6.2.1.1 Probabilistic Correlation Approach

Alerts from multiple sensors can be fused into attack scenarios using the concept of meta alerts. A meta alert is a set of one or more alerts considered similar enough to be grouped together. The meta alert itself includes a list of thread IDs, so the operator is always able to

examine in detail the alerts that contributed to the meta alert report. The example presented in figure 15 shows an attack that transpired over several days [10]:

Meta Alert Thread 248
Source IPs source_IParray: x.y.148.33 x.y.148.47
Target IPs target_IParray: a.b.30.117 a.b.6.232 a.b.8.31 a.b.1.166 a.b.7.118 a.b.28.83 a.b.19.121 a.b.21.130 a.b.6.194 a.b.1.114 a.b.16.150
From 2001-06-15 17:34:35 to 2001-06-21 09:19:57
Correlated_alert_priority –1
Ports target_TCP_portarray: 12345{4}, 27374{4}, 139{3}
Number of threads 10 Threads: 69156 71090 76696 84793 86412 87214 119525 124933 125331 126201
Fused: PORT_SCAN

Figure 15. EMERALD Meta alert thread.

The probabilistic alert fusion approach considers feature overlap, feature similarity, minimum similarity, and expectation of similarity.

For two alerts (typically a new alert and a meta alert), we begin by identifying features they have in common (feature overlap). Such features include:

- the source of the attack
- the target (hosts and ports)
- the class of the attack
- time information

With each feature, we have a similarity function that returns a number between 0 and 1, with 1 corresponding to a perfect match.

Expectation of similarity is also a number between 0 and 1, and it expresses our prior expectations that the feature should match if the two alerts are related, considering the specifics of each. We can consider expectation of similarity as a feature weighting that can vary based on the type of correlation being performed.

The correlation component also enforces situation-specific minimum similarity. Certain features can be required to match exactly (minimum similarity for these is unity) or approximately (minimum similarity is less than unity, but strictly positive) for an alert to be considered as a candidate for fusion with another. Setting minimum expectation for some features to unity (not normally recommended) causes the system to behave like a

heuristic system that requires exact matches on these features. Minimum expectation thus expresses necessary but not sufficient conditions for correlation.

The overall similarity between two alerts is zero if any overlapping feature matches at a value less than the minimum similarity for the feature (features for which no minimum similarity is specified are treated as having a minimum similarity of 0). Otherwise, overall similarity is the weighted average of the similarities of the overlapping features, using the respective expectations of similarity as weights:

$$SIM(X, Y) = \frac{\sum_j EjSIM(Xj, Yj)}{\sum_j Ej}$$

X = Candidate meta alert for matching
Y = New alert
j = Index over the alert features
Ej = Expectation of similarity for feature j
Xj, Yj = Values for feature j in alert X and Y, respectively (may be list valued)

Figure 16. The EMERALD similarity function [10].

The expectation of similarity and minimum similarity are set by the mode of correlation:

- Thread correlation attempts to correlate a large number of alerts from the same sensor pertaining to the same attack (for example, a large number of alerts pertaining to an IP address scan). In this case, minimum similarity for the sensor and the attack class are unity, expectation of similarity in time is high, and expectation of similarity in target IP is low
- Incident correlation potentially correlates events from more than one sensor, so expectation of sensor similarity is low, expectation of time similarity is high, and expectation of incident class similarity is medium-high (since different sensors may have different classifications for the same attack)
- Scenario mode relaxes time similarity and includes an asymmetric transition matrix that expresses the similarity of an attack of one class following an attack of the previous class. This matrix is hand-coded on our judgement

In our terminology, a meta alert is an alert output from the correlation component. It may consist of a single raw sensor alert, if such an alert does not satisfy any of the correlation modes. Threading is the most prevalent correlation mode that takes place in practice. Raw alerts and threads may be further correlated into incidents and scenarios. Our meta alert content includes a field for the mode of correlation which applies to the particular meta alert.

Our experience has been that thread correlation achieves alert count reduction of a factor of ten on average for Snort, and about a factor of three for EMERALD sensors, which perform a degree of within-sensor threading.

When the system decides to fuse two alerts, based on aggregate similarity across common features, the fused feature set is a superset of the features of the two alerts. Feature values in fused alerts are typically lists, so alert fusion involves list merging. For example, suppose a probe of certain ports on some range of the protected network matches in terms of the port list with an existing probe that originated from the same attacker subnet, but the target hosts in the prior alert were to a different range of our network. The attacker address list has the new attacker address appended, and the lists of target hosts are merged. The port list matches and is thus unchanged.

6.2.1.2 Feature Similarity Functions

The features presently considered in the probabilistic correlation component include:

- sensor identification (identifier, location, name)
- alert thread
- incident class
- source and target IP lists
- target TCP/UDP port lists
- source and target user id
- time

Computations are only over features that overlap in the alert to be merged and the candidate meta alert into which it is to be merged. Incident signature is used as well, but with a low expectation of similarity as these vary widely across heterogeneous sensors.

Some of the more important similarity functions are presented below.

Alert Thread Similarity.
If an alert from a sensor has a thread identifier that matches the list of sensor/thread identifiers for some meta alert, the alert is considered a match and fusion is done immediately. In other words, the individual sensor's determination that an alert is an update of or otherwise related to one of its own alerts overrides other considerations of alert similarity.

Incident Class Similarity.
Incident class similarity is based on a notion of proximity, which at present is the result of our judgment. The proximity of class A to B reflects how reasonably an attack currently of incident class A may progress to class B. Note that this is not symmetric; we more strongly expect an exploit to follow a probe than the other way around. The actual classes used are from the EMERALD Incident Handling Knowledge Base (IHKB) and are listed in section 6.2.2.2 below. Note that some "default" classes such as "invalid" and "action logged" are reasonably proximal to most other classes. The reason for this is because there does not exist an agreed upon standard for attack naming, and reports from heterogeneous sensors for the same incident may use different attack names or classes. As such, we do not want to reject potential matches based on this field alone.

For two alerts that are extremely close in time, it is possible that the alerts may not be in time order. In this case, incident class similarity is the greater of SIM(X,Y) and SIM(Y,X).

Source and Target Similarity.
Most features are potentially list-valued. For lists, the notion of similarity generally expresses the fraction of the smaller list that is contained in the larger. For source IP addresses, similarity also attempts to express the notion that the addresses in question may come from the same subnet.

If the meta alert has received reports from host sensors on different hosts, we do not expect the target host feature to match. If at least one report from a network sensor has contributed to the meta alert and a host sensor alert is received, the expectation of similarity is that the target address of the latter is contained in the target list of the former.

In determining whether an exploit can be plausibly considered the next stage of an attack for which a probe was observed, we expect the target of the exploit (the features host and port) to be contained in the target host and port list of the meta alert.

Deciding whether the attacker is similar is somewhat more complicated. In the case of an exact match of originating IP address, similarity is perfect. We assign high similarity if the subnet appears to match. In this way, a meta alert may potentially contain a list of attacker addresses. At this point, we consider similarity based on containment. In addition, if an attacker compromises a host within our network (as inferred by a successful outcome for

an attack of the root compromise class), that host is added to the list of attacker hosts for the meta alert in question. Finally, for attack classes where the attacker's address is likely to be spoofed (for example, the Neptune SYN flood attack), similarity expectation with respect to attacker address is assigned a low value.

Time Similarity.
Time similarity is a step function that drops to 0.5 after one hour. A close match in time is expected only when the system operates in "incident" mode. Thread and scenario aggregation may be over time intervals of days.

6.2.1.3 Multistage Attack Scenarios

To recognize a multistage attack scenario, correlation can be achieved by appropriate similarity settings at various levels in the sensor hierarchy.

Firstly, at the single sensor level, threads can be used to aggregate alarms that belong to the same attack. For sensors where the thread concept is not supported, threads can be synthesized.

Secondly, when minimum expectation of similarity on the sensor identifier is suppressed, and expectation of similarity is relaxed, incidents from several heterogeneous sensors can be correlated into a single incident report. A moderately high expectation of similarity on the attack class is enforced. This need not be unity, because the attack may be reported as belonging to different attack classes depending on the sensor used. In addition, minimum expectations on the source and target address are still enforced.

Thirdly, when minimum expectation of similarity on the attack class is relaxed, various steps in a multistage attack scenario can be reconstructed. Each stage in an attack may itself be a correlated security incident as described above.

In this fashion, it is possible to recognize a staged attack composed of, for example, a probe followed by an exploit to gain access to an internal machine, and then using that machine to launch an attack against a more critical asset.

6.2.2 A Correlation Experiment with Snort Alerts

In this section we present the correlation experiment, which included a one-day live traffic analysis. Using a live traffic analysis is faced with a dilemma, trading off the controlled conditions of a test environment, versus the realism of actual Internet traffic. We have opted for the latter, because we wish to consider the "added value" aspect of sensor correlation, and we need realistic alert volumes and false positive rates.

6.2.2.1 Experiment Setup

To evaluate the utility of heterogeneous sensor correlation, we considered a diverse set of intrusion detection sensors. By "diverse," we mean diversity of analytical technique. The following sensors were deployed:

- EMERALD eXpert-Net. This rule-based system keeps a moderate amount of session states, and has signatures for a number of attacks against TCP, HTTP, FTP, SMTP, ICMP, and UDP protocols [11]
- EMERALD eBayes TCP. This system detects attacks visible in TCP header data using Bayes inference [12]. It is particularly effective against general probe attacks
- Snort 1.8 is a widely deployed open source network IDS, which currently contains more than 1200 rules [8]. We ran Snort with the IDMEF XML plug-in from Silicon Defense [9], and we also implemented a translator to convert the XML alerts to EMERALD binary messages

The EMERALD sensors were installed on a dual-processor platform with a passive interface to the network gateway and a private interface for alert reporting and administrator functions. The private interface is not visible on the monitored network. We have been developing this appliance under various EMERALD programs for some months, and have extensive experience using it to examine live traffic.

Snort was installed on an Intel i686 machine listening to the same interface. In a day of monitoring, Snort observed 7.8 million packets. According to Snort diagnostics, no packets were dropped. Due to the Code Red activity at the time of monitoring, we do not know if this volume is higher than average for the network in question.

6.2.2.2 Live Traffic Analysis

The monitoring period included a set of probes launched by our central IT staff, which gives us some known attack data. Additionally, the monitoring period encompassed a period of high activity of the Code Red and Code Red V2 worms [14].

The probabilistic correlation engine considers attack class as one of the features in its similarity matching algorithms. The mapping between attack classes and attack signatures is implemented in the EMERALD incident handling knowledge base (IHKB), which is shared by all EMERALD sensor and correlation components. Currently, all signatures are mapped into 14 Incident Handling Knowledge Base (IHKB) classes (see table 9).

EMERALD IHKB Classes		
ACCESS VIOLATION	DENIAL OF SERVICE	SUSPICIOUS USAGE
ACTION LOGGED	INTEGRITY VIOLATION	SYSTEM ENVIRONMENT CORRUTPION
ASSET DISTRESS	INVALID	USER ENVIRONMENT CORRUPTION
BINARY SUBVERSION	PRIVILEGE VIOLATION	USER SUBVERSION
CONNECTION VIOLATION	PROBE	

Table 9: EMERALD Incident Handling Knowledge Base (IHKB) classes.

During the case study, all Snort alerts were assigned to the fallback "ACTION LOGGED" class. (However, a complete mapping of Snort alerts to EMERALD incident classes has been done.) An advantage of probabilistic techniques is that this approach produces slightly lower fidelity results, but the technique is sufficiently robust to tolerate this as a minor deficiency.

Because of the overall architecture of EMERALD, we are able to deploy a correlation capability at one or more points in a monitoring network, and can in fact correlate correlated alerts. We chose to separately correlate the Snort alerts, the EMERALD alerts, and the entire set. The first function of correlation, as presented in the introduction, is to reduce the raw number of alert reports that a security administrator must examine. Table 10 presents totals for a one-day collection period in our laboratory, starting at 10 a.m. PDT, August 6-7, 2001.

Sensor	**Raw Alerts**	**Correlated Alerts**
Snort	4816	487
EMERALD	1586	523
Composite	6402	869

Table 10: Result from correlation experiment at SRI International.

As described above, it is possible that a raw alert will fail to correlate with any other alerts. In this case, the corresponding correlated alert will consist solely of the contents of the single contributing raw alert. Therefore, the set of correlated alerts contains information for all of the raw alerts. We observe that correlation achieves about a 10 to 1 reduction in Snort alerts, and about 3 to 1 for EMERALD alerts. This occurs because the EMERALD sensors attempt to thread alerts, as we have previously discussed.

6.2.2.3 Sensor Complementarity

The most frequent Snort alert for which there is no corresponding EMERALD alert is the DNS named version attempt. The full Snort set contains 815 alerts of this class, which reduce to 224 correlated alerts, or 47% of the total Snort correlated alerts. To the degree that we can ascertain, we believe most, if not all, of the sessions in question reflect innocuous activity, and these are most likely false alarms. Of these, one alert correlated with an alert from the Bayes sensor, which considered the session's combination of ports highly unusual. In addition to the DNS request, the same session accessed ssh. This detection was from the Bayes sensor's anomaly detection capability and implies that the port combination was extremely unusual but not necessarily a probabilistically good match to any of the misuse models in the Bayes sensor's knowledge base.

The Bayes sensor also detected port sweeps such as sessions that probe for NETBUS and SUB7 [15][16]. The approach used is probabilistic, not based on a rule such as "so many ports within a given interval of time" nor on a rule such as a match to a pattern of ports corresponding to a known probing script such as MSCAN. The Bayes sensor was the only sensor in the mix to report this apparent probe.

One likely DNS sweep detected by the EMERALD Bayes and eXpert-TCP sensors probed 255 IP addresses on port 53. This was not reported by Snort. The correlator assembled the following apparent multistage attack scenario from several reports from Snort and the EMERALD Bayes sensor:

- An attempt to connect to port 1243, detected by the Bayes sensor
- A near-simultaneous DNS request from a different IP address in the same subnet as above, detected by Snort
- Nine hours later, a connection attempt to port 3128, detected by both sensors, from the same subnet
- One hour later, a connection attempt to port 3150, detected by the Bayes sensor, from the same subnet

The composite alert is an apparent probe distributed in time and source for ports 53, 1243, 3128, and 3150. Formation of this composite alert highlights the utility of the techniques

we are proposing. The Bayes and Snort sensors have complementary coverage, and the ability of the former to detect probes with no prior signature is essential to seeing parts of this attack. In live operation, the initial alert is reported as it is generated, and future alerts are considered updates to this one. The ability to correlate over time and across sensors is the final critical capability needed to form a more complete picture. Snort reports a number of ICMP-related alerts, such as "ICMP Ping," "ICMP Source unreachable," and "ICMP Source quench," for which there is no EMERALD counterpart. For these, the reports in the correlated set come only from Snort. We have not yet investigated these to see which, if any, are valid alerts. The originating address of the "Source unreachable" alerts is our network switch, which leads us to believe that the traffic is legitimate.

6.2.2.4 Sensor Reinforcement

In addition to the cases cited above, the sensors agreed on a large number of apparent attacks. As mentioned previously, the monitoring period contains a set of scripted vulnerability probes launched by the central IT office at SRI, intending to uncover vulnerabilities that may be unknown to individual laboratory domains. All the sensors in the suite generate numerous alarms from these probes.

Even though sensors may alert in response to the same incident, we found (as expected) that the calls are often different. For example, signature systems may have an explicit signature for BACK ORIFICE [17], whereas the Bayes sensor detects this as an extremely unusual access attempt. To permit correlation of these messages, the correlation engine must explicitly allow imperfect matches, and should work at the incident class rather than signature level. Our probabilistic engine satisfies this requirement.

Code Red activity over the period in question was sufficiently significant to warrant further discussion. Both Snort and eXpert-TCP were used with signatures for this worm that were very new at the time of our analysis. The Bayes sensor potentially detects Code Red activity if several target IP addresses are probed; in this case, the alert is a TCP ADDRESS SWEEP that increases in confidence as more addresses are probed. Snort detects Code Red and Code Red II with the rule "WEB-IIS .ida attempt."

To the best of our ability to analyze the results ex post facto, all Code Red attempts that trigger the Snort "WEB-IIS .ida attempt" are also detected by one or both of the EMERALD sensors. The correlation facility identified repeated re-infection attempts from the same source or source subnet. We observed Code Red traffic from several sources within the same subnet in numerous correlated Code Red alerts. The correlation engine effectively aggregates these reports across time, sensor, and source subnet. As we have no IIS servers in our facility, all of these attempts failed.

Code Red II often triggers the Snort "WEB-IIS cmd.exe" rule in addition to the "WEB-IIS .ida attempt" rule, although we observed some instances of "cmd.exe" with no "WEB-IIS .ida attempt" alert. We observe that the payload field for these reports looks like that of the alerts related to Code Red, so we are not certain why the traffic did not trigger the prior rule as well. Since EMERALD did not alert for these sessions, correlation sheds no more light on this issue.

6.2.3 Conclusions

We have presented a case study of practical IDS alert correlation using heterogeneous sensors, test-driving the IDMEF standard as far as its ability to provide adequate content for such a study. This was carried out at a time when the Internet as a whole was subject to a fairly high attack frequency due to variants of the Code Red worm. We sought to explore the ability of heterogeneous sensors and sensor correlation to provide alert volume reduction, complementary coverage, and sensor reinforcement.

Considering Snort as a widely deployed sensor that does not thread alerts, we were able to reduce alert volume by an order of magnitude. The alert volume reduction for the EMERALD sensors is less, in part because these sensors implement alert threading. We may argue that 869 alerts a day is an unacceptably high number of reports for an administrator to comprehend. However, 224 Snort DNS alerts can be assigned lower priority, and we must recognize that many of the rest are very likely originating from the Code Red worm.

We were able to identify likely attacks where sensor heterogeneity and correlation brought out a more complete picture of what actually happened. We have identified enhancements we would like to make to the IDMEF standard to extend the functionality of the EMERALD correlation engine. However, even at this stage of the game, the probabilistic approach of the correlation engine has proven itself sufficiently robust to be extremely useful to a security administrator.

6.3 Design of an Alert Correlation Tool using Open Source Software

The main objective of this research project was to investigate the possibility to add automated log analysis to the current syslog architecture. Parts of this effort were conducted as a Master's thesis project [35], supervised by the present author. The hypothesis was that it should be possible to use existing open source tools to build a customized intrusion detection and correlation system utilizing the collected log information. The architecture consisted of various brands and versions of routers, servers and switches sending syslog messages to a central log host.

There are several reasons for evaluating open source tools instead of going for COTS-based log analysis and intrusion detection systems directly, including:

- The cost of COTS is still significant
- No single COTS would be able to cover all brands and versions within the log architecture
- The possibility to customize COTS according to all local requirements is often minimal

We will start out by presenting some syslog basics followed by implementation specific issues.

6.3.1 Syslog from an Intrusion Detection Perspective

6.3.1.1 A Centralized Syslog Architecture

A common use of the syslog protocol is to create a centralized syslog architecture. Applications, on the local systems, have their level of logging set through application specific configuration files. Log messages are then directed to the local syslog subsystem, where further filtering of log messages is done with respect to the rules specified in the syslog.conf file. Local analysis of generated log messages is now possible, however, a huge efficiency gain is possible by directing several syslog feeds to a central log server, thus creating a centralized syslog architecture. This also has important ramifications for security, as the remote transport of syslog messages gives an additional window of opportunity to detect the attack before the log messages are deleted by the intruder.

The centralized syslog architecture enables central management of enterprise wide logs. This (should) include backing up the generated log file(s), as well as regular auditing. A too common solution is to only rely on manual audits, after the fact. The preferred solution

is of course to add automatic log analysis in the form of log based intrusion detection systems.

In the following sections, the various parts of the syslog subsystem are explained in more detail, as well as the implications of using syslog for intrusion detection.

6.3.1.2 Syslog Basics

Syslog is the standard system logging facility on Unix platforms. It first appeared in BSD 4.2. Syslog consists of three parts: the client library, syslog, the log daemon, syslogd, and the configuration file, /etc/syslog.conf. By default, the log files are stored in /var/log on the local file system.

All users in the Unix system are able to use syslog (you do not have to be root) to send messages to a syslog server. This can be done by incorporating support for the the syslog client library or through the use of a shell command called logger.

6.3.1.3 Syslog.conf

Any user can use the syslog subsystem to generate log messages. However, it is the syslog.conf file that decides how the messages should be treated. (Please note that some of the functions described below are system dependent.)

Syslog.conf consists of lines with two fields called selector and action. Every log message received by the syslog subsystem is compared to the syslog.conf file. If a matching selector is found, i.e. the correct facility and a level that is at the same or at a lower level, the corresponding action is performed. The chosen action can be one of the following:

- /abs_path_to_log_file_or_device Append message to selected file or device
- @loghost Send message to another loghost
- user1, …, userN Write message to specified user(s), if logged in
- * Write message to all users that are logged in
- | program Pipe message to a program

6.3.1.4 Syslogd

The usual way to start up syslogd is through the rc startup files. By default syslogd reads the syslog.conf file at startup. However, this can be changed using the –f option on the command line. Some security by obscurity can be accomplished by using the –f option to direct syslogd to another configuration file. This means that an intruder that by default only looks at the syslog.conf file, which has been left in the system on purpose, can be tricked to remove the wrong logfiles, or attack a fake remote log server (a honeypot).

Syslogd listens for log messages from the kernel, from local processes, and from the network. If syslog is going to receive messages over the network, it must be started with the –r option (in Linux), and if it is going to forward syslog messages between hosts, it must be started with the –h option.

6.3.1.5 Attacks against Syslog

UDP-based servers, like syslogd, are vulnerable to denial-of-service (DoS) attacks. The reason for this is that UDP-based servers accept data without doing any kind of "handshake" protocol, that is, not verifying the true origin of the sender. This means that servers that use UDP with no additional application layer security, like verifying digital signatures, gladly accept spoofed packets.

In the case of syslog, an intruder can flood syslogd with log messages, filling the local filesystem (to crash the system or to insert crafted log messages), or rotating the logs to hide previous activity.

As syslog messages are sent in the clear, coded as ASCII, they are inherently easy to sniff. It is also possible to intercept, change, and also discard packets without the sending or receiving parts noticing.

A general recommendation is that syslogd should not be reachable from the outside world to prevent denial-of-service attacks. Also, to minimize the risk of spoofing, all relevant firewalls should implement anti spoofing filters.

6.3.1.6 Using Syslog for Intrusion Detection

The basic format of the syslog message was previously presented in section 6.1.1. Briefly, every syslog message has the following syntax [55]:

Date:time:facility-severity-mnemonic:description

It is the description part of the syslog message that includes application specific information. It is this information that is most interesting during the log analysis. A web server will for example log information about processed requests including source, destination, requested page, and if the request was granted or not. A router is able to log information about logged access rules, etc.

In a centralized syslog architecture, all this logging will require effective hardware and software resources. However, and here comes the main problem, the description part of the syslog message has not been standardized in any way. This means that it has been up to the application developers to decide what to send to the syslog facility. Even worse,

there does not exist a normalized log format, i.e. every application developer has been free to use his/her own format.

From an intrusion detection system's point of view, this means that log messages from diverse operating systems and applications, generated using different formats, must be supported for full coverage. This is not the way commercial intrusion detection systems work today. Most often, only a fixed set of operating systems and applications are supported.

This leads us back to our initial hypothesis, namely that it should be possible to use existing open source tools to build a customized intrusion detection and correlation system using collected log information.

6.3.2 A Correlation Experiment using Open Source Tools

6.3.2.1 Customizing Open Source Tools

Several open source tools were surveyed with special focus on tools within the areas of log analysis, alert correlation and presentation. Among those tools, three were chosen for customization:

- Checksyslog [18]
- Simple Event Correlator (SEC) [19]
- Analog [20]

The first tool, Checksyslog, is a regular expression based log analysis tool written in perl [18][21]. The purpose of this tool is to perform pre-filtering in the form of audit data reduction of input log files. This means that everything known to be non-malicious is removed from the log file.

```
#
# very noisy BIND
#
named\[\d+\]: Connection refused
named\[\d+\]: .*.reloading nameserver
named\[\d+\]: Lame delegation
named\[\d+\]: Lame server
named\[\d+\]: ns_forw:
named\[\d+\]: .*.points to a CNAME
named\[\d+\]: sysquery:
```

Figure 17. Excerpt from the sample rule file distributed with checksyslog [18].

The way checksyslog handles audit data reduction is the following. Compare each log entry against a list of known good behavior. If there is a match, remove the log entry from the output. Figure 17 presents some "anti-signatures" that can be found in the sample rule file that is distributed with checksyslog.

The second tool, Simple Event Correlator (SEC), is an advanced log analysis tool [19], written in perl [21], with the possibility to create contexts (event storage space) and perform correlation.

The actions supported by SEC are primarily focused on reporting matching events, and the maintenance of contexts. Contexts can e.g. be created, updated with additional events, reported (e.g. the events within the context can be written to a log file), and deleted.

Session analysis can be performed on services that generate known start and stop messages (see bold text in figure 18). This includes e.g. telnet, and ftp. Figure 18 is an example of how to handle sessions generated by ftp [19].

```
type=single
continue=takenext
ptype=regexp
pattern=ftpd\[(\d+)\]: \S+ \(ristov2.*FTP session opened
desc=ftp session opened for ristov2 pid $1
action=create ftp_$1

type=single
continue=takenext
ptype=regexp
pattern=ftpd\[(\d+)\]:
context=ftp_$1
desc=ftp session event for ristov2 pid $1
action=add ftp_$1 $0; set ftp_$1 1800 (report ftp_$1 /bin/mail root@localhost)

type=single
ptype=regexp
pattern=ftpd\[(\d+)\]: \S+ \(ristov2.*FTP session closed
desc=ftp session closed for ristov2 pid $1
action=report ftp_$1 /bin/mail root@localhost; delete ftp_$1
```

Figure 18. Example rules from SEC creating ftp sessions [19].

The rules above create, maintain, report and delete (either after a timeout of 1800 seconds or when the session closes) ftp sessions, which are (uniquely) identified by their specific PID (process id) numbers.

When it comes to misuse detection, it may be enough to create simpler rules, mapping an event to an action that writes event information into a specified log file.

The third tool, Analog, is written in C [20], and it is primarily a tool for analyzing web server logs with the purpose of creating various forms of statistics. The (output) statistics are delivered in HTML format for presentation using a regular web browser.

During the survey of possible presentation tools, we found that Analog had previously been used in a program called fwanalog to present firewall statistics [22]. A closer look, showed that Analog is rather configurable, especially if the input can be transformed into something looking like a regular web log, e.g.:

```
webclient - - [14/Aug/2002:14:02:35 +0100] "GET path/webpage HTTP/1.0"
 - -
```

Figure 19. An example of Common Log Format (CLF).

We used the following simple conventions, when rewriting SEC output messages (i.e. intrusion alerts):

- Source of attack is transformed into web client
- Target of attack is transformed into path to web page
- Alert class is transformed into viewed web page
- Date and time is transformed into a new format

Using the transformations above, we end up with something looking like:

```
source - - [14/Aug/2002:14:02:35 +0100] "GET target/DoS_attack HTTP/1.0”
 - -
```

Figure 20. A sample intrusion alert log entry suitable for Analog.

Additionally, changes are then needed to specific language files (to change the vocabulary from web server statistics to intrusion detection statistics), and also a couple of changes in the analog configuration file. The implemented demonstrator was developed to create the following statistics:

- General Summary
- Monthly Report. A presentation of the total number of alerts per month

- Daily Report. A presentation of the total number of alerts per day during a week
- Hourly Report. A presentation of the total number of alerts per hour during a day
- Attacker Report. A listing of the most common attackers
- Target Report. A listing of the most commonly targeted systems
- Target and Alert Type Report. A listing of the most common target and alert type combinations

A sample output from the demonstrator is presented in figure 21:

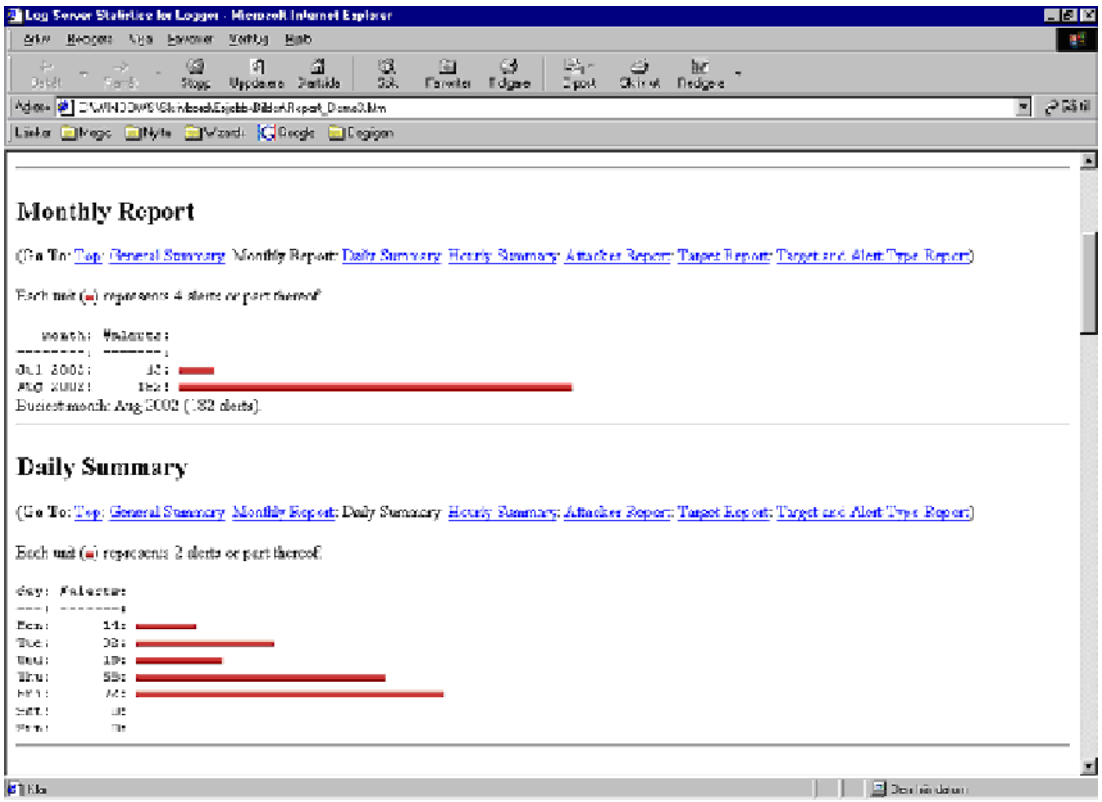

Figure 21. Example output from customized Analog [35].

6.3.2.2 Choosen Analysis Method

After an initial assessment of the collected log data and the chosen Open Source tools the following intrusion detection and correlation architecture was suggested:

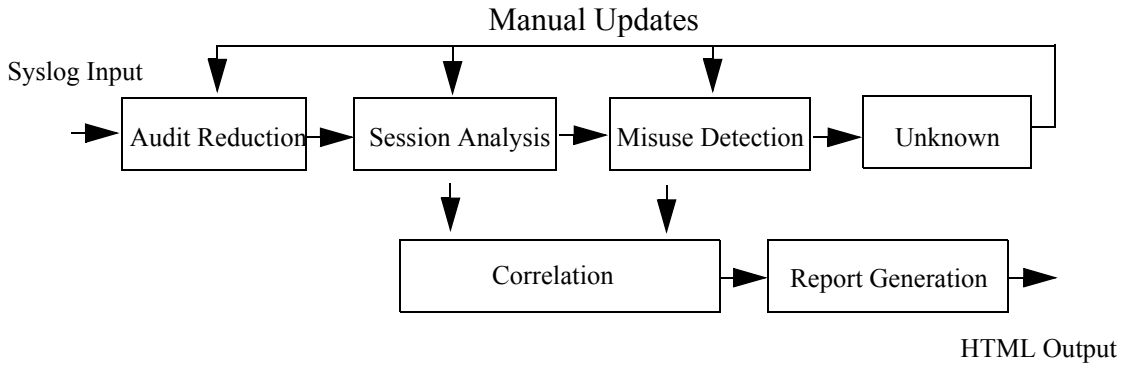

Figure 22. Proposed analysis method.

Checksyslog was chosen for audit data reduction, SEC for session analysis, misuse detection, and correlation. Analog was chosen for presentation purposes.

Audit data reduction, which was first presented by Anderson in 1980 [69], increases the chances of finding misuse and anomalies. The reason for this is that the factor between good behavior and misuse or anomalies decreases.

Session analysis and misuse detection represents basic intrusion detection methods. The output from both of these methods are intrusion alerts. By using a common alert template for both methods, it is possible to perform intrusion alert correlation on the combined output. The correlated alerts are then written to yet another log file, which can be prepared for presentation using Analog.

All log entries that are left after audit reduction, session analysis, and misuse detection, represent events that have not been seen in the system before, i.e. they represent new and possible anomalous behavior. In our model, it is left to the site security officer (SSO) to classify new events into known good behavior, sessions, or misuse, and then update the rule-base accordingly.

6.3.2.3 A Correlation Experiment

To verify the proposed analysis method, a set of simple experiments was performed. A Swedish Internet Service Provider (ISP) provided a 250 MB log file. The log file was collected from a central log host, hosting a syslog server. The log entries, in turn, were col-

lected from various sources in the company network, and included syslog messages from host systems, routers and switches.

We then created a basic set of misuse detection signatures using SEC. A test run with the log file generated more than 1,000 alerts. This is clearly not manageable, which is also stated in [35].

The most common alert types generated were alerts concerning access and authentication failures in SNMP, rsh, telnet, and ssh. The alerts were not investigated further and it is unclear if all alerts were valid.

To filter out the most interesting alerts, different correlation rules were created, once again using SEC. The main findings included:

- Five correlated alerts were created by identifying attacker and victim pairs with more than one attack type in common
- Ten correlated alerts were created by identifying attackers that connected to more than a set threshold of victim hosts (in this case three)
- Ten correlated alerts were created by identifying victims with more than a set threshold of attackers connecting (in this case three)

Even though the experiment was quite simple, the usefulness of the proposed analysis method was clearly shown.

6.3.3 Conclusions

The experiment showed that it was possible to build a simple intrusion alert correlation tool by customizing Open Source tools. Compared to similar Open Source log analysis tools, like Swatch [62], the demonstrator excelled in both correlation capabilities and in the presentation of the result.

Although the experiment was limited in size, it should be possible to extend it to a (more) distributed environment. However, this would require local customizations of audit reduction filters, and session and misuse detection signatures.

Finally, it is important to realise that a customized intrusion alert correlation system, like the one presented, will need a lot of effort to create new signatures, keeping up with new threats. A potential ease could be to regularly update the rule-base with new misuse signatures from e.g. the Snort Open Source project [8]. The additional maintenance cost must in this case be compared to the cost of updates and support from commercial alternatives.

7 An Intrusion Tolerant Web Service

Even if the performance of the intrusion detection system increases through the use of intrusion alert correlation, there will always be intrusions. One way to mitigate this problem is to use fault tolerant techniques to provide intrusion tolerance. This results in a system that will continue to provide a valid service, possibly with a performance penalty, in spite of current intrusion attempts [36].

In this section, I will present work performed within the Dependable Intrusion Tolerance (DIT) project at SRI International.

7.1 An Intrusion Tolerant Architecture

The purpose of the DIT project was to evaluate the possibility to use intrusion tolerant techniques together with the existing EMERALD intrusion detection infrastructure. The intrusion tolerant design principles presented in this section are described in [53], and the interested reader should consult that report for an in-depth treatment of the details of the approach. Additionally, we only look at one single tolerance proxy. Information about the multi proxy scenario configuration is found in [53].

7.1.1 Architecture Components

The architecture used within the DIT project is composed of four major components: application servers, tolerance proxies, IDS, and a firewall (see figure 23).

The redundant application servers are used to provide contents to requesting web browsers. The application servers are implemented with diversity in mind. Different hardware, operating systems, and applications are used to minimize the risk of all web servers being vulnerable to the same attack or failure modes. All servers were instrumented to support MD5 checksums according to HTTP/1.1 [61].

The tolerance proxies are then used to provide a secure front-end to the application servers. They mediate client requests to one or more application servers depending on the currently selected security policy. When several proxies are used, one is chosen to be the master proxy. The remaining auxiliary proxies monitor all communication to verify the behavior of the proxy leader. If a majority of the auxiliary proxies agree on that the master proxy is out of scoop (compromised or otherwise not functioning correctly), a new master proxy is selected.

The IDS is used as one part of the monitoring subsystem described below.

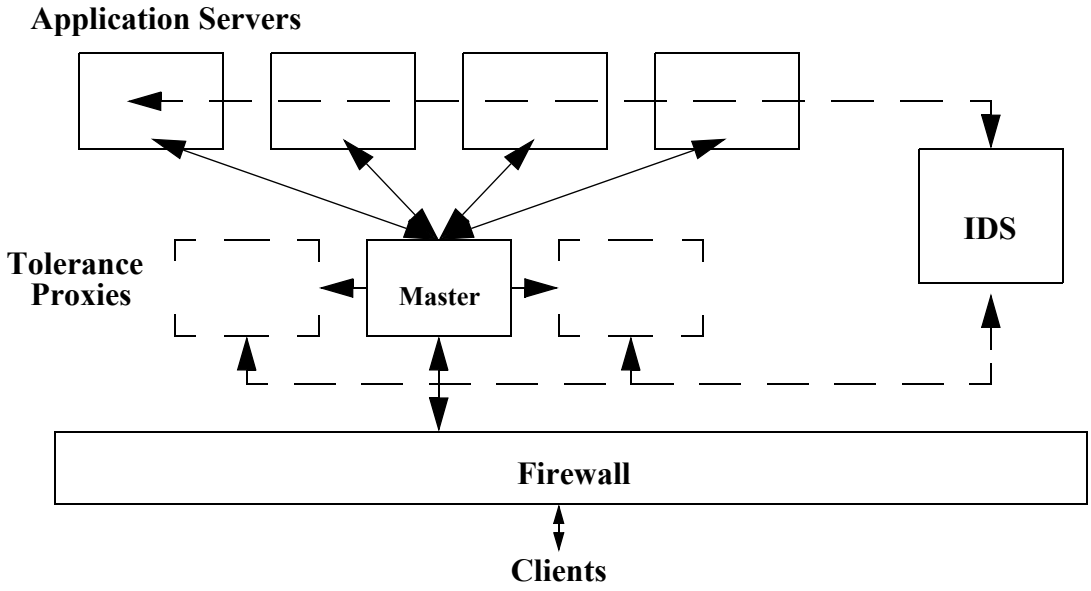

Figure 23. Schematic view of the intrusion tolerant server architecture [53].

The firewall is used minimize the exposure of the intrusion tolerant system. Only web requests are allowed to pass through from the outside.

7.1.2 Monitoring Subsystem

The monitoring subsystem, which is implemented on the network, on the application servers, and on the tolerance proxies, uses four complementary techniques: IDS, content agreement, challenge-response protocols, and on-line verifiers.

The intrusion detection systems include the EMERALD host, network, and protocol sensors [38][39]. The use of diverse sensors provides complementary coverage, potentially leading to an increased detection rate. Additionally, correlation techniques can be used to lower the number of alerts, and reducing the false alarm rate [10][12][25].

The tolerance proxies implement content agreement by computing MD5 checksums over the requested web content [41]. Failing application servers are identified by comparison with peers.

Two different challenge-response protocols are used, namely, heartbeat messages and integrity checks. The heartbeat messages recognize down servers. The integrity check protocol functions like a remote Tripwire [40]. The client sends a nonce to the integrity check server, which computes a response by concatenating the nonce and the static web content

and feeding them to a one-way hash function. The client compares the response with a recomputed hash value. Any difference will produce an alert.

The on-line verifiers are used to verify the behavior of the tolerance proxies. The abstract properties of the tolerance proxies must hold in every situation. Unexpected states produce alerts.

7.1.3 Agreement Regimes

Agreement regimes are used by the tolerance proxies to specify the level of replication needed for each incoming client request. Four different regimes are supported:

- benign mode (called single mode in [53])
- duplex mode
- triplex mode
- full agreement mode

In benign mode, no current intrusion or intrusion attempts are known, and the client requests are sent directly to one of the redundant application servers. The load-balancing algorithm uses a random scheme to remedy the possibility that an intruder can make intelligent guesses about which of the redundant servers is chosen next. By not knowing the hardware, operating system, and application of the target, it becomes more difficult to succeed with an attack on the first try, making the attack easier to detect for an intrusion detection system.

In duplex mode, the client request is sent to two servers with identical content. This could be the default state of the web service, or a state that is invoked as part of an intrusion detection response process. A mismatch between the replies from the servers means that a security violation may be in progress, i.e. we have detection. In this case the proxy could send a default web page back to the client, explaining that the web service is unavailable for the moment. This should be followed by either passive or active response. However, as we still do not know which one of the two server systems that is affected, we must initiate triplex mode.

In triplex mode, the client request is sent to three servers with identical content. If we assume that an attack always affects only one of the three systems, we should be able to identify the compromised server. Even though we have a detection, the proxy is able to send a valid response back to the client, using majority voting. The remedy, in this case, could be a complete repair of the identified system.

Full agreement mode is used when mismatches continue to occur even after a complete repair during triplication. This means that at least five server systems are needed, and that the minority servers are taken off-line for a complete repair.

7.1.4 Adaptive Response

The system may use a different response depending on the current situation (policy) and on the alerts received. Different possibilities include:

- Temporarily blocking the offending addresses
- Go to a higher agreement regime, e.g. from benign to duplex
- Increasing the coverage and frequency of heartbeat and integrity check messages
- Make one of the application servers unavailable during a rebuild
- Halting web service and alerting the SSO
- Tracing the intruder

7.2 Design of an Intrusion Tolerant Reverse Proxy

The design presented in this section was the first prototyping of the tolerance proxy performed at SRI. The purpose of it was to provide quick answers to the rest of the project team, as well as to provide a demonstrator for an upcoming event. The design of the demonstrator was performed during the summer of 2001, and the DIT project has evolved since then.

However, the first intrusion tolerant web service was implemented using four different servers:

- Intrusion-tolerant Squid (reverse) proxy on Redhat 6.2
- Netscape FastTrack 4.1 (iPlanet) server on Redhat 6.2
- Apache 1.3.20 server on FreeBSD 4.2
- Microsoft IIS 5.0 server on MS Windows 2000

The web servers were regular COTS. They were installed using as much default settings as possible. Apache had experimental support for MD5 checksums, while FastTrack and IIS had this functionality added manually.

7.2.1 Choosing a Suitable Platform

For efficiency reasons, our ambition was to reuse code from some of the Open Source, high availability projects. Several different platforms were evaluated according to our purposes:

- Linux Virtual Server
- Apache HTTP Server
- TIS Firewall Toolkit
- Squid Web Proxy Cache

Linux Virtual Server is a kernel patch that implements load balancing using IP masquerading and different load balancing algorithms [37]. However, a kernel based solution would be difficult to port to other operating systems, which was our plan. (The reason for this is to remedy the situation of the proxy being a single point of failure.)

The Apache HTTP Server has high availability, load balancing, and proxy features built into some of its modules [42][43][44][45]. However, none of the modules could be used directly for our purposes, as the only way to provide load balancing was to rewrite the URL.

The TIS Firewall Toolkit had two proxies that looked interesting to use, http-gw, and plug-gw [47]. However, http-gw is designed to serve only trusted hosts, i.e. using it as a reverse proxy could result in a security incident. Plug-gw, on the other hand, only provided a bare minimum of code reuse, and no load balancing.

Our choice fell on the Squid Web Proxy Cache, which also provided a load-balancing feature [46]. This could be done using either accelerator mode and a redirection capability, i.e. similar to Apache, or by using a separate patch that turns the proxy into a reverse proxy.

By choosing to install the rproxy patch, Squid becomes a reverse proxy with several different load-balancing algorithms to choose among. Incoming client requests are then processed in the following order:

- Accept http request
- Read client request
- Check access restrictions
- Check URL validity
- Redirect request if necessary

- Check if the requested content is in the web cache (if used)
- Choose one of the peer servers for delivery according to load-balancing algorithm
- Get the requested content from the chosen application server
- Return content to client
- Update cache if needed

After a code walk-through, we decided to add a new load-balancing algorithm. The new load-balancing algorithm was implemented using a random distribution algorithm, with additional intrusion tolerant features described below.

7.2.2 Implementing the Intrusion Tolerant Reverse Proxy

Our main function was called getRandomParent() and it was passed a pointer to a request-structure as input argument. The request-structure includes information about which application servers are available, the client URL, and other information important to serve a reverse proxy request. The return value from getRandomParent() was a pointer to a peer-structure, including the chosen application server to send the request to. The Squid configuration file, squid.conf, was then configured to use the new random algorithm.

Two new definitions were added, PEER_NOT_COMPROMISED, and PEER_COMPROMISED. They were then used to flag a peer, i.e. an application server, as compromised, by adding one additional variable, the integer *state*, to the peer-structure. When Squid is first started, this state variable is initialised as PEER_NOT_COMPROMISED, for all peers, i.e. applications servers. Later, when the squid function peerHTTPOk() checks if the chosen peer is ok, we added one extra test to only choose servers that were not flagged as compromised.

In the first implementation, we used a counter value to time-out the different security policies. Id_max_counter and id_current_counter was used to implement the timer.

Additionally, we added a suspicious-value for intrusion detection reports. The purpose of which was to be used in an ageing algorithm for incoming intrusion alerts. As long as the suspicious-value is below a certain value, the server is flagged as either UP (functioning correctly), SUSPICIOUS, or COMPROMISED. The suspicious-value was initialised to 0.0, i.e. no suspicion. Due to time constraints the suspicious-value technique was not fully implemented.

To compare the different results from the applications servers, we used the MD5 algorithm [41]. This was done by collecting the web content from one of the application servers, and collecting the MD5 values of the requested web content from the remaining

application servers. If all MD5 checksums agreed, the content was assumed to be valid. If there was a difference, further processing was needed to pinpoint the failing application server.

The getRandomParent() function consisted of the following parts:

- Initialization
- Health-checks
- Timer adjustments
- Control of external information
- Choose one of the application servers to be SERVER1
- Perform additional fault localisation according to the current agreement regime
- Return chosen application server to reverse proxy code

At first, the function initializes all used variables, as well as performs some simple health checks, e.g. checking that the chosen agreement regime is valid. The timer, id_current_counter, is then decreased by one. If the timer reaches zero, and the agreement regime is higher than benign, the agreement regime is lowered one step, and the counter is again set to id_max_counter.

Before another round of choosing one of the application servers, the getRandomParent() function checks if there is any external information available. This information comes from an external alert manager.

If one of the application servers is flagged as compromised, that application server is taken off service, and cannot be chosen by the random load-balancing algorithm. After some time, maybe after a total rebuild of the application server, the application server is flagged as on-line again. This triggers the proxy to add the application server to the set of possible application servers, and the application server can once again be chosen by the load-balancing algorithm.

After possible deletions and/or additions of application servers, the function chooses one of the available application servers to service the request. Depending on the current agreement regime the chosen server is either delivered directly to the reverse proxy code, or one or more additional application servers are chosen.

In benign mode, the chosen application server is delivered directly.

In duplex mode a second application server is chosen. The MD5 checksums are compared, and if they agree, one of the servers is delivered back to the reverse proxy code. If they do

not agree, the triplex agreement regime is entered and one additional application server is chosen. The three MD5 checksums are then compared and the failing application server is identified and taken off-line. One of the majority application servers is then delivered back to the reverse proxy code.

If the proxy is in a high agreement regime (duplex or triplex), and no differences are found between the MD5 checksums, the regimes will eventually time out and switch to a lower regime. This means that after a long enough time period, or in this case, after a certain amount of valid MD5 comparisons, the agreement regime will be back at the lowest level, benign mode.

7.2.3 Initial Testing

The intrusion tolerant reverse proxy was implemented and tested using a fictional attack scenario. During the execution of the proxy a log trace was written to a file in the local file system. The log contains information about the traversal between different policy regimes and the reason for changing policy.

The log trace could for example include one of the following examples during a content agreement check using MD5 with three diverse application servers (the server name is presented together with the MD5 checksum):

```
SERVER1: XXXXXXXXXXXXXX
SERVER2: XXXXXXXXXXXXXX
SERVER3: XXXXXXXXXXXXXX
MD5 checksums agree

SERVER1: XXXXXXXXXXXXXX
SERVER2: XXXXXXXXXXXXXX
SERVER3: XYXYXYXYXYXYXY
Compromised host: SERVER3
```

Figure 24. Example output in triplex mode.

Alternatively, the log trace looks like the following when only using two application servers:

```
SERVER1: XXXXXXXXXXXXXX
SERVER2: XYXYXYXYXYXYXY
Disagreement in reported MD5 checksums
```

Figure 25. Example output in duplex mode.

The log trace below shows a fictional attack scenario with a recently started intrusion tolerant system, where different events trigger a traversal between different policy levels, i.e. agreement regimes (see table 11).

Table 11: Fictional attack scenario.

IN THE LOG	DESCRIPTION
DUPLEX:	The intrusion tolerant proxy starts in duplex mode
BENIGN:	After a timeout, the intrusion tolerant proxy enters benign mode
Served content from SERVER1	Server1 is randomly selected from [1, 2, 3]
BENIGN:	
Served content from SERVER2	Server2 is randomly selected from [1, 2, 3]
BENIGN	
Severed content from SERVER1	Server1 is randomly selected from [1, 2, 3]
BENIGN:	
Served content from SERVER3	Server3 is randomly selected from [1, 2, 3]
Suspicious event reported from IDS component	IDS component reports suspicious events. Unclear which application server is affected
DUPLEX:	The intrusion tolerant proxy enters duplex mode
SERVER3: XXXXXXXXXXX	Server3 is randomly selected from [1, 2, 3]
SERVER1: YYYYYYYYYYY	Server1 is randomly selected from [1, 2]
Disagreement in reported MD5 checksums	The checksums differ

Table 11: Fictional attack scenario.

IN THE LOG	DESCRIPTION
TRIPLEX:	The intrusion tolerant proxy enters triplex mode
SERVER3: XXXXXXXXXXX	Server3 is still selected
SERVER1: YYYYYYYYYYY	Server1 is still selected
SERVER2: YYYYYYYYYYY	Server2 is randomly selected from [2]
Compromised host: SERVER3	Server3 is supplying the minority answer
Served content from SERVER1	Server1 is selected from [1, 2]
DUPLEX:	The intrusion tolerant proxy enters duplex mode
SERVER2: ZZZZZZZZZZZZ	Server2 is randomly selected from [1, 2]
SERVER1: ZZZZZZZZZZZZ	Server1 is randomly selected from [1]
MD5 checksums agree	The checksums agree
BENIGN:	After a timeout, the intrusion tolerant proxy enters benign mode
Served content from SERVER2	Server2 is randomly selected from [1, 2]
Server3 rebuilt	Server3 is reported as rebuilt
TRIPLEX:	The intrusion tolerant proxy enters triplex mode
SERVER1: SSSSSSSSSSSSS	Server1 is randomly selected from [1, 2, 3]
SERVER3: SSSSSSSSSSSSS	Server3 is randomly selected from [2, 3]

Table 11: Fictional attack scenario.

IN THE LOG	DESCRIPTION
SERVER2: SSSSSSSSSSSSS	Server2 is randomly selected from [2]
DUPLEX:	After a timeout, the intrusion tolerant proxy enters duplex mode
BENIGN:	After a timeout, the intrusion tolerant proxy enters benign mode

In table 11 the following events are described. Duplex mode is chosen to mitigate spurious reboots caused by attackers. After a time-out, with no suspicious events happening, the tolerance proxy enters benign mode, i.e. using a one-to-one mapping between requests and application servers. When an IDS component reports a suspicious event, the tolerance proxy once again enters duplex mode to be able to verify if one of the application servers is serving invalid data. When a mismatch is found, triplex mode is entered, and a third application server is polled. A majority decision is then used to identify the failing application server. The tolerance proxy then continues to serve data from the two remaining application servers. After a while, the third server is put on-line again and triplex mode is entered. The scenario ends with two time-outs, leaving the intrusion tolerant system in benign mode.

7.3 Conclusions

We have shown that our single intrusion tolerant web proxy is able to deliver valid contents, even though some of the server systems may be down or otherwise delivering invalid contents. Compared to existing high availability solutions, our system adds an extra layer in the form of content validation. However, we have introduced a new weakness; the proxy represents a single point of failure. One way to limit this new exposure is to implement multiple proxies using Byzantine Agreement [80].

An additional problem with the intrusion tolerant web service that needs to be addressed is the comparison of replies from web requests. It has been shown that even valid replies may differ between different COTS web servers [84].

Finally, the use of Open Source software resulted in fast prototyping. However, due to superfluous functionality (resulting in too much code), the recommendation is to rewrite the intrusion tolerant proxy from scratch.

8 Summary

In this section we discuss the main results of the licentiate thesis.

8.1 A Generic Alert Correlation Procedure

In the Intrusion Correlation section the most important research question was; what techniques and methods exist for intrusion alert correlation?

To be able to answer this question a survey of related research was performed. As can be seen from the survey, the different alert correlation approaches are beginning to converge. It is thus possible to identify generic building blocks that are used by a large part of the surveyed alert correlation approaches.

The generic building blocks found performed the following tasks:

- The generation of alerts pertaining to a standard format
- Transforming the alerts into tuples in a relational database
- Correlating alerts of the same attack type
- Correlating alerts of different attack types
- Refining alert information using out-of-bound information
- Building scenarios to be able to identify (and respond to) the next step in the ongoing intrusion scenario

It is important to remember that the survey focused on hierarchical intrusion correlation models. The main reason for not including distributed alert correlation models was the lack of research in this area.

8.2 Intrusion Correlation Experiments

The next part of the thesis investigated an additional set of questions:

- What are the implications of using router based logs in the intrusion detection and correlation process?
- What is the additional value of using intrusion alert correlation together with heterogeneous intrusion detection sensors?
- Is it possible to customize current Open Source to create effective intrusion alert correlation systems?

The first experiment demonstrated problems when using access control lists in routers as event generators. The main problems were:

- The false negative rate can be very high
- Log entries may be delayed for several minutes
- It is easy to generate false log entries (i.e. false positives)

The false negative rate, which in this case means missed alerts, can be substantial. This may lead to a situation when it is not meaningful to use the router log data at all. However, the alerts generated should be useful for both sensor complementarity and sensor reinforcement. Together with alerts from other sources it should provide additional value during the alert correlation process.

The second experiment illustrated the value of intrusion alert correlation in that it is able to greatly reduce the number of alerts reported. In the experiment performed a reduction of an order of magnitude was gained. If we were to add additional out-of-band information during the alert correlation process, the reduction would probably be even greater. Thus, we showed that it was possible to create a lower number of correlated alert reports, with the potential to minimize the time needed before intrusion response can be started.

The third experiment demonstrated that it was possible to build an effective alert correlation tool using (several) existing Open Source tools. Although the tool is somewhat primitive, it is definitively a more complete “solution” than current Open Source log analysis tools like Swatch [62].

Given the interest from several research groups, IDMEF is a model that helps solving some of the most urgent problems concerning the lack of standard formats within the intrusion detection community.

8.3 An Intrusion Tolerant Web Service

As it is clear that there will always be intrusions and that intrusion alert correlation only solves part of the problem, one additional avenue of defence is to build systems that, to some degree, are immune against intrusions, i.e. intrusion tolerant systems.

The purpose of the last part of the thesis was to answer the following set of questions:

- What techniques and methods exist for intrusion tolerant services?
- What is the additional value of using intrusion tolerant web proxies?

Obviously, many of the techniques and methods from the fault tolerant community look promising for intrusion detection and intrusion tolerance. In this thesis, we have presented

a brief survey of current research together with an initial effort to evaluate the use of an intrusion tolerant web service.

The results show that the intrusion tolerant web proxy is able to deliver valid results, even though some of the server systems may be down or otherwise delivering invalid data. Compared to existing high availability solutions, the intrusion tolerant web proxy adds an extra layer in the form of content validation.

Despite the recent interest in intrusion tolerance additional work efforts will be needed to evaluate the existing knowledge. Trade-offs in the form of availability and integrity protection need to be evaluated.

9 Results

The main findings of my thesis may be summarized in five paragraphs.

Firstly, the survey of related research concerning intrusion alert correlation is, to our knowledge, the first survey done in this area. Research papers from several different research groups are summarized, and it is clear that most of the surveyed groups have been working with the same practical problems, and that this has resulted in similar solutions to common problems. Based on this knowledge, a conceptual framework for intrusion alert correlation was suggested.

Secondly, during the evaluation of the feasibility to use router log data as a source for intrusion detection, we discovered that router access lists may be a poor event source. Log entries are potentially generated late, or not at all, and it is easy to spoof log entries. One way to mitigate this situation is to use the log entries for intrusion alert correlation purposes, and not for stand alone intrusion detection, i.e. the value of the generated log entries may increase when correlated with other sources.

Thirdly, during the live traffic analysis, alerts from various sensors within the EMERALD suite were successfully correlated together with alerts from the network based Open Source IDS Snort. Our results showed that the total number of alerts could be reduced by an order of magnitude, and that multistage attack reports could be successfully created. We were also able to identify attacks where sensor heterogeneity and correlation brought out a more complete picture of what actually happened.

Fourthly, during the evaluation of the possibility to use current Open Source tools for intrusion alert correlation, a prototype was implemented and tested. We showed that it was possible to create correlated attack reports, and that this may be an alternative to buying expensive COTS.

Lastly, the potential benefit of intrusion tolerance was evaluated. A prototype was implemented and analysed. We showed that it was possible to use different fault tolerant mechanisms, e.g. redundancy and diversity, to be able to tolerate some degree of intrusions.

All in all, we have shown the value and possibilites of intrusion alert correlation and intrusion tolerance in different settings.

10 Abbreviations

ACC	Aggregation and Correlation Component
API	Application Programming Interface
ARM	Adaptive Reconfiguration Module
BEEP	Blocks Extensible Exchange Protocol
CGI	Common Gateway Interface
CIDF	Common Intrusion Detection Framwork
CISL	Common Intrusion Specification Language
CLF	Common Log Format
COTS	Commercial Off The Shelf
CVE	Common Vulnerabilities and Exposures
DARPA	Defense Advanced Research Projects Agency
DIT	Dependable Intrusion Tolerance
DoS	Denial-of-Service
EMERALD	Event Monitoring Enabling Responses to Anomalous Live Disturbances
ESP	Encapsulating Security Payload
FTP	File Transfer Protocol
GIDO	Generalized Intrusion Detection Object
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IAC	Intrusion Alert Correlation
ICMP	Internet Control Message Protocol
IDIP	Intruder Detection and Isolation Protocol
IDMEF	Intrusion Detection Message Exchange Format
IDS	Intrusion Detection System
IDXP	Intrusion Detection Exchange Protocol

IETF	Internet Engineering Task Force
IHKB	Incident Handling Knowledge Base
IP	Internet Protocol
ITTC	Intrusion Tolerance via Threshold Cryptography
MAFTIA	Malicious- and Accidental-Fault Tolerance for Internet Applications
M-Correlator	Mission Impact Intrusion Report Correlation System
SASL	Simple Authentication and Security Layer
SEC	Simple Event Correlator
SIM	Similarity Function
SITAR	Scalable Intrusion-tolerant Architecture for Distributed Services
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SSL	Secure Sockets Layer
SSO	Site Security Officer
TCP	Transmission Control Protocol
TOCTTOU	Time Of Check To Time Of Use
UDP	User Datagram Protocol
WAN	Wide Area Network
XML	Extensible Markup Language

11 References

[1] C. P. Pfleeger. Security In Computing. Prentice Hall International, Inc. ISBN 0-13-185794-0, 1997.

[2] W. Ford. Computer Communications Security - Principles, Standard Protocols and Techniques. Prentice Hall, Inc. ISBN 0-13-799453-2, 1994.

[3] R. G. Bace. Intrusion Detection. Macmillan Technical Publishing. ISBN 1-57870-185-6, 2000.

[4] Office for Official Publications of the European Communities. Information Technology Security Evaluation Criteria. June 1991. Version 1.2.

[5] Common Criteria Implementation Board. Common Criteria for Information Technology Security Evaluation, Part I: Introduction and General Model. August 1999. Version 2.1.

[6] International Standards Organization. Information Processing Systems - OSI - Basic Reference Model - Part 2: Security Architecture. ISO 7498-2. February 1989.

[7] D. Curry and H. Debar. Intrusion Detection Message Exchange Format. http://www.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-10.txt. August 2003.

[8] M. Roesch. http://snort.sourcefire.com/. May 2002.

[9] Silicon Defense. http://www.silicondefense.com/idwg/snort-idmef/. May 2002.

[10] A. Valdes and K. Skinner. Probabilistic Alert Correlation. In Proceedings of the 4th International Symposium, Recent Advances in Intrusion Detection (RAID) 2001, Springer-Verlag Lecture Notes in Computer Science.

[11] U. Lindqvist and P. Porras. Detecting Computer and Network Misuse Through the Production-Based Expert System Toolset (P-BEST). In Proceedings of the 1999 IEEE Symposium on Security and Privacy IEEE Computer Society Press, Oakland, CA. May, 1999.

[12] A. Valdes and K. Skinner. Adaptive, Model-Based Monitoring for Cyber Attack Detection. In Proceedings of the third International Workshop, Recent Advances in Intrusion Detection (RAID) 2000, Springer-Verlag Lecture Notes in Computer Science, October 2000.

[13] J. Hoagland, J. McAlemy, and S. Stanniford. Practical Automated Detection of Stealthy Portscans. http://www.securityfocus.com/library/3019. May 2002.

[14] F-Secure Virus Descriptions. Code Red. http://www.europe.f-secure.com/v-descs/bady.shtml. May 2002.

[15] F-Secure Virus Descriptions. NetBus. http://www.europe.f-secure.com/v-descs/netbus.shtml. May 2002.

[16] F-Secure Virus Descriptions. SubSeven. http://www.europe.f-secure.com/v-descs/subseven.shtml. May 2002.

[17] F-Secure Virus Descriptions. Back Orifice. http://www.europe.f-secure.com/v-descs/backori.shtml. May 2002.

[18] CheckSyslog. http://www.jammed.com/~jwa/hacks/security/checksyslog/checksyslog-doc.html. August 2002.

[19] SEC Simple Event Correlator. http://www.estpak.ee/~risto/sec/sec.pl.html. August 2002.

[20] Analog. http://www.analog.cx. August 2002.

[21] Perl. http://www.perl.org. August 2003.

[22] Fwanalog, http://tud.at/programm/fwanalog. August 2002.

[23] H. Debar and A. Wespi. Aggregation and Correlation of Intrusion-Detection Alerts. In Proceedings of the 4th International Symposium, Recent Advances in Intrusion Detection (RAID) 2001, Springer-Verlag Lecture Notes in Computer Science, October 2001.

[24] O. M. Dain and R. K. Cunningham. Building Scenarios from a Heterogeneous Alert Stream. In IEEE Workshop on Information Assurance and Security, United States Military Academy, West Point, NY, 5-6 June 2001.

[25] D. Andersson, M. Fong, and A. Valdes. Heterogeneous Sensor Correlation: A Case Study of Live Traffic Analysis. In IEEE Information Assurance Workshop, United States Military Academy, West Point, NY, June 2002.

[26] P. A. Porras, M. W. Fong, and A. Valdes. A Mission-Impact-Based Approach to INFOSEC Alarm Correlation. In Proceedings of the 5th International Symposium, Recent Advances in Intrusion Detection (RAID) 2002, Springer-Verlag Lecture Notes in Computer Science, October 2002.

[27] P. Ning, D.S. Reeves, and Y. Cui. Correlating Alerts Using Prerequisites of Intrusions. Technical Report TR-2001-13, North Carolina State University, Department of Computer Science, 2001.

[28] P. Ning and Y. Cui. An Intrusion Alert Correlator Based on Prerequisites of Intrusions. Technical Report TR-2002-1, North Carolina State University, Department of Computer Science, 2002.

[29] P. Ning, X.S. Wang, and S. Jajodia. Modelling Requests among Cooperating Intrusion Detection Systems. Computer Communications 23(17):1702-1715, Elsevier Science, 2000.

[30] P. Ning, Y. Cui, and D. S. Reeves. Analyzing Intensive Intrusion Alerts via Correlation. In Proceedings of the 5th International Symposium, Recent Advances in Intrusion Detection (RAID) 2002, Springer-Verlag Lecture Notes in Computer Science, October 2002.

[31] F. Cuppens and A. Miège. Alert Correlation in a Cooperative Intrusion Detection Framework. In Proceedings of 2002 IEEE Symposium on Security and Privacy, 2002.

[32] B. Moring, L. Me, H. Debar, and M. Ducassé. M2B2: A formal Data Model for IDS Alert Correlation. In Proceedings of the 5th International Symposium, Recent Advances in Intrusion Detection (RAID) 2002, Springer-Verlag Lecture Notes in Computer Science, October 2002.

[33] E. Amoroso. Intrusion Detection - An Introduction to Internet Surveillance, Correlation, Trace Back, Traps, and Response. Intrusion.Net Books. ISBN 0-9666700-7-8. 1999.

[34] F. Cuppens. Managing alerts in multi-intrusion detection environments. In 17th Annual Computer Security Applications Conference (ACSAC). New-Oreans, December 2001.

[35] M. Jorstedt. A method for log analysis. Masters Thesis in computer security, Chalmers University of Technology, Department of Engineering, Sweden, 2002.

[36] Y. Deswarte, L. Blain, and J.-C. Fabre. Intrusion tolerance in distributed computing systems. In Proceedings of the International Symposium on Security and Privacy, pages 110-121. IEEE press, May 1991.

[37] W. Zhang, S. Jin, and Q. Wu. Creating Linux Virtual Servers, http://www.linuxvirtualserver.org. Technical report, 1999. August 2001.

[38] P. Porras and P. Neumann. EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. In National Information Security Conference, Oct 1997.

[39] P. Porras and A. Valdes. Live traffic analysis of TCP/IP gateways. In Proceedings of the Symposium on Network and Distributed System Security. Internet Society, Mar 1998.

[40] Tripwire white papers, http://www.tripwire.com. August 2001.

[41] R. Rivest. The MD5 message digest algorithm. Internet Engineering Task Force, RFC 1321. Apr 1992.

[42] R. S. Engelschall. Load Balancing Your Web Site. Technical report. 1998. http://www.webtechniques.com/archives/1998/05/engelschall/. August 2001.

[43] The Apache Software Foundation. Apache HTTP Server Version 1.3: Apache 1.3 URL Rewriting Guide. http://www.apache.org. August 2001.

[44] The Apache Software Foundation. Apache HTTP Server Version 1.3: Module mod_proxy. http://www.apache.org. August 2001.

[45] The Apache Software Foundation. Apache HTTP Server Version 1.3: Module mod_rewrite URL Rewriting Engine. http://www.apache.org. August 2001.

[46] Team Squid. Squid: A User's Guide. http://www.squid-cache.org. August 2001.

[47] Trusted Information Systems. TIS Firewall Toolkit Documentation. http://www.fwtk.org. August 2001.

[48] F. Wang, F. Gong, C. Sargor, K. Goseva-Popstojanova, K. Trivedi, and F. Jou. SITAR: a scaleable intrusion tolerance architecture for distributed services. In Second IEEE SMC Information Assurance Workshop, 2001.

[49] Project IST-1999-11583. Reference Model and Use Cases. MAFTIA deliverable D1, August 25, 2000.

[50] Project IST-1999-11583. Towards a Taxonomy of Intrusion Detection Systems and Attacks. MAFTIA deliverable D3, Version 1.01, September 6, 2001.

[51] Project IST-1999-11583. Design of an Intrusion-Tolerant Intrusion Detection System. MAFTIA deliverable D10, Version 4.3, August 9, 2002.

[52] T. Wu, M. Malkin, and D. Boneh. Building intrusion tolerant applications. http://www.stanford.edu/~dabo/ITCC. August 2001.

[53] A. Valdes, M. Almgren, S. Cheung, Y. Deswarte, B. Dutertre, J. Levy, H. Saidi, V. Stavridou, T. E. Uribe. An Adaptive Intrusion-Tolerant Server Architecture. Technical Report. System Design Laboratory, SRI International. Feb, 2002.

[54] M. J. Ranum, K. Landfield, M. Stolarchuk, M. Sienkiewicz. Implementing a Generalized Tool for Network Monitoring. In the Proceedings of the Eleventh System Administration Conference (LISA '97), San Diego, California, October 1997.

[55] Cisco Connection Online. http://www.cisco.com. August 2000.

[56] S. Northcutt. Network Intrusion Detection - An Analyst's Handbook. New Riders Publishing, 1999.

[57] V. Paxon. Bro: A System for Detecting Network Intruders in Real Time. Technical report, 1999.

[58] S. P. Harbison and G. L. Steeler Jr. C - A reference Manual, forth edition. Prentice Hall. ISBN 0-13-326224-3. 1995.

[59] NMAP Network Mapping Tool. http://www.nmap.org. August 2000.

[60] Nessus. http://www.nessus.org. August 2000.

[61] R. Fielding et al. RFC 2616, Hypertext Transfer Protocol -- HTTP/1.1. http://www.w3.org/Protocols/rfc2616/rfc2616.html. August 2003.

[62] S. E. Hansen, and E. T. Atkins. Automated System Monitoring and Notification With Swatch, In Proceedings of the Seventh System Administration Conference (LISA VII). 1993.

[63] IDWG. http://www.ietf.org/html.charters/idwg-charter.html. August 2003.

[64] SASL. http://rfc.net/rfc2222.html. August 2003.

[65] IDIP. http://www.networkassociates.com/us/nailabs/research_projects/adaptive_network/intrusion_tracing.asp. August 2003.

[66] J. D. Howard. An Analysis of Security Incidents On The Internet 1989-1995. PhD thesis, Carnegie Mellon University, Pittsburgh, Pensylvania. 1997.

[67] U. Lindqvist and E. Jonsson. How to Systematically Classify Computer Security Intrusions. In Proceedings of the 1997 IEEE Symposium on Security and Privacy, pp. 154-163, Oakland, California, May 4-7, 1997.

[68] P. G. Neumann and D. B. Parker. A summary of computer misuse techniques. In Proceedings of the 12th National Computer Security Conference, pages 396-407, Baltimore, Maryland, October 10-13, 1989. National Institute of Standards and Technology/National Computer Security Center.

[69] J. P. Anderson. Computer security threat monitoring and surveillance. Technical report, James P Andersson Co., Box 42, Fort Washington, PA 19034, USA, April 15, 1980.

[70] C. E. Landwehr, A.R. Bull, J.P. McDermott, and W.S. Choi. A taxonomy of computer program security flaws. ACM Computing Surveys, 26(3):211-254, September 1994.

[71] U. Lindqvist. On the Fundamentals of Analysis and Detection of Computer Misuse. PhD thesis, Department of Computer Engineering, Chalmers University of Technology, Göteborg, Sweden 1999.

[72] NSA Glossary of Terms in Security and Intrusion Detection. http://www.sans.org/resources/glossary.php. June 2003.

[73] E. Lundin and E. Jonsson. Survey of Intrusion Detection Research. Technical Report nr. 02-04, Department of Computer Engineering, Chalmers University of Technology, Göteborg, Sweden 2002.

[74] S. Axelsson. Intrusion Detection Systems: A Survey and Taxonomy. Technical Report 99-15, Department of Computer Engineering, Chalmers University of Technology, Göteborg, Sweden 1999.

[75] H. Debar, M. Dacier, and A. Wespi. Towards a taxonomy of intrusion-detection systems. Computer Networks 31 (1999) 805-822, IBM Research Division, Zurich Research Laboratory, Switzerland.

[76] B. Dutertre, V. Crettaz, and V. Stavridou. Intrusion-Tolerant Enclaves. In Proceedings of the IEEE Symposium on Research in Security and Privacy, Oakland, 2002. System Design Laboratory, SRI International, Menlo Park, CA.

[77] L. Blain and Y. Deswarte. A Smartcard Fault-Tolerant Authentication Server. In 1st Smart Card Research and Advanced Application Conference (CARDIS'94), Lille, France, 1994.

[78] ICAT METABASE . http://icat.nist.gov/icat.cfm. August 2003.

[79] MAFTIA homepage. http://www.newcastle.research.ec.org/maftia/programme/. August 2003.

[80] D. K. Prahan. Fault-tolerant computer system design. Prentice Hall PTR, 1996.

[81] L. Gong. Enclaves: Enabling Secure Collaboration over the Internet. In IEEE Journal on Selected Areas in Communications, 11(5):567-575, Apr. 1997.

[82] B. Dutertre, H. Saidi, and V. Stavridou. Intrusion-tolerant Group Management in Enclaves. In International Conference on Dependable Systems and Networks (DSN'01), pages 203-212, Göteborg, Sweden, July 2001.

[83] C. Cachin, K. Kursawe, and V. Shoup. Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement Using Cryptography. In Proceedings of the 19th Annual Symposium on Principles of Distributed Computing, Portland, OR, July 2000.

[84] M. Almgren and U. Lindqvist. Application-Integrated Data Collection for Security Monitoring. In Proceedings of the 4th International Symposium of Recent Advances in Intrusion Detection (RAID) 2001, Springer-Verlag Lecture Notes in Computer Science, October 2001.

[85] S. Manganaris, M. Christensen, D. Zerkle, and K. Hermiz. A Data Mining Analysis of RTID Alarms. In 2nd Workshop on Recent Advances in Intrusion Detection (RAID), 1999.

[86] Common Vulnerability and Exposures (CVE). http://www.cve.mitre.org/. August 2003.

[87] H. Debar and B. Morin. Evaluation of the Diagnostic Capabilities of Commercial Intrusion Detection Systems. In Proceedings of the 5th International Symposium of Recent Advances in Intrusion Detection (RAID) 2002, Springer-Verlag Lecture Notes in Computer Science, October 2002.

[88] Bugtraq ID. http://www.securityfocus.com/bid. August 2003.

[89] S. Kumar. Classification and Detection of Computer Intrusions. PhD thesis, Purdue University, West Lafayette, Indiana, August 1995.

[90] T. Champion, and R. Durst. Air Force Intrusion Detection System Evaluation Environment. In 2nd Workshop on Recent Advances in Intrusion Detection (RAID), 1999.

[91] M. Chung, N. Puketza, B. Mukherjee, and R. A. Olsson. Simulating Concurrent Intrusions for Testing Intrusion Detection Systems; Parallelizing Intrusions. In Proceedings of the 18th National Computer Security Conferens, 1995.

[92] H. Debar, M. Decier, A. Wespi, and S. Lampart. An Experimentation Workbench for Intrusion Detection Systems. IBM Research Division, Zurich Research Laboratory, 8803 Ruschlikon, Switzerland, March 1998.

[93] R. Durst, T. Champion, B. Witten, E. Miller, and L Spagnuolo. Testing and Evaluating Computer Intrusion Detection Systems. Communications of the ACM, July 1999.

[94] R. Lippmann, R. Cunningham, D. Fried, I. Graf, K. Kendall, S. Webster, M. Zissman. Results of the DARPA 1998 Offline Intrusion Detection Evaluation. In 2nd Workshop on Recent Advances in Intrusion Detection (RAID), 1999.

[95] N. Puketza, B. Mukherjee, R. Olsson, and K. Zhang. Testing Intrusion Detection Systems: Design Methodologies and Results from an Early Prototype. In Proceedings of the 17th National Computer Security Conferens, 1994.

[96] N. Puketza, M. Chung, R. A. Olsson, and B. Mukherjee. A Software Plattform for Testing Intrusion Detection Systems. IEEE Software, vol 14, no. 5, September/October 1997.

[97] DARPA Intrusion Detection and Evaluation program at MIT's Lincoln Labs. http://www.ll.mit.edu/IST/ideval/index.html. June 1999.

[98] D. E. Denning. An Intrusion-Detection Model. IEEE Transactions on Software Engineering, vol. SE-13, no. 2, february 1987.

[99] G. Vigna. A topological characterization of tcp/ip security. Technical Report TR-96.156, Politecnico di Milano, 1996.

[100] H. Debar, M. Dacier, and A. Wespi. A Revised Taxonomy for Intrusion-Detection Systems. Research Report, IBM Research, Zurich Research Laboratory, Switzerland, 1999.