

ABSTRACT

Title of Document: EVALUATING HOST INTRUSION
DETECTION SYSTEMS.
Jesus Molina, Doctor of Philosophy 2007

Directed By: Michel Cukier, Assistant Professor, Department
of Mechanical Engineering

Host Intrusion Detection Systems (HIDSs) are critical tools needed to provide in-depth security to computer systems. Quantitative metrics for HIDSs are necessary for comparing HIDSs or determining the optimal operational point of a HIDS. While HIDSs and Network Intrusion Detection Systems (NIDSs) greatly differ, similar evaluations have been performed on both types of IDSs by assessing metrics associated with the classification algorithm (e.g., true positives, false positives). This dissertation motivates the necessity of additional characteristics to better describe the performance and effectiveness of HIDSs.

The proposed additional characteristics are the ability to collect data where an attack manifests (visibility), the ability of the HIDS to resist attacks in the event of an intrusion (attack resiliency), the ability to timely detect attacks (efficiency), and the ability of the HIDS to avoid interfering with the normal functioning of the system

under supervision (transparency). For each characteristic, we propose corresponding quantitative evaluation metrics.

To measure the effect of visibility on the detection of attacks, we introduce the probability of attack manifestation and metrics related to data quality (i.e., relevance of the data regarding the attack to be detected). The metrics were applied empirically to evaluate filesystem data, which is the data source for many HIDSs.

To evaluate attack resiliency we introduce the probability of subversion, which we estimate by measuring the isolation between the HIDS and the system under supervision. Additionally, we provide methods to evaluate time delays for efficiency, and performance overhead for transparency. The proposed evaluation methods are then applied to compare two HIDSs.

Finally, we show how to integrate the proposed measurements into a cost framework. First, mapping functions are established to link operational costs of the HIDS with the metrics proposed for efficiency and transparency. Then we show how the number of attacks detected by the HIDS not only depends on detection accuracy, but also on the evaluation results of visibility and attack resiliency.

EVALUATING HOST INTRUSION DETECTION SYSTEMS

By

Jesus Molina

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

2007

Advisory Committee:
Professor Michel Cukier, Chair
Professor Jonathan Agre
Professor John S. Baras
Professor Ali Mosleh
Professor Gang Qu

© Copyright by

Jesus Molina

2007

Dedication

To Angelines, Gabriel, world peace and you, especially you.

Acknowledgements

No work can be done alone, and this is no exception. First and foremost, I want to thank my advisor, Dr. Michel Cukier. His commitment to this work was only matched by my own, with weekdays and weekends devoted to review, critique and suggestions. I am also very grateful to my committee members, Dr. John Baras, Dr. Gang Qu, Dr. Ali Mosleh and Dr. Jonathan Agre for their comments and suggestions. Two mentors were especially vital in this long walk, Dr. Virgil Gligor and Dr. Bill Arbaugh. They gave me the strength that I needed to reach the end.

This dissertation required an enormous amount of work, and the members of my laboratory shared the burden. I was lucky to be surrounded by so many talented people. I especially want to thank Robin Berthier, Christine Lu, Joe Gordon and Xavier Chorin for providing support to collect and analyze data. Keith Jarrin provided extensive comments on my presentation, and I can only hope I will be able to do the same with his own dissertation soon.

The understanding of all my colleagues at Fujitsu laboratories of America kept me from ending up in a mad house discussing my dissertation with a flower pot. Dr. Jonathan Agre played a chief role, by not only supporting my efforts, but also taking the extra step of actually being involved in the process. I also want to thank Dr. Ryusuke Masuoka, for being patient and kind, and for believing in my success even when I was not sure myself.

Luis Simon performed the mighty task of cleaning my anarchic bibliography and transforming my references into a thing of beauty. And regarding things of beauty, I owe Rosemarie Salguero a love letter for her hours spent reviewing the manuscript. Finally, friends and family always give support and care. Gabriel Molina, Angelines Terriza, Gloria Terriza, Ariadna Alvarado and Fernanda Azocar gave me that and much more, and there is no word to describe how grateful I am to have them by my side.

Table of Contents

DEDICATION	II
ACKNOWLEDGEMENTS	III
TABLE OF CONTENTS	V
LIST OF TABLES	VIII
LIST OF FIGURES	IX
CHAPTER 1: INTRODUCTION	1
1.1 OVERVIEW OF THE DISSERTATION.....	3
1.2 CONTRIBUTIONS	5
1.2.1 <i>Selecting Characteristics</i>	6
1.2.2 <i>Evaluating Characteristics</i>	6
1.2.3 <i>Evaluating Metrics through Practical Experiments</i>	7
1.2.4 <i>Creation of a HIDS Optimization Framework</i>	8
1.3 PUBLICATIONS.....	9
CHAPTER 2: RELATED WORK	10
2.1 INTRUSION DETECTION SYSTEMS	10
2.2 HOST INTRUSION DETECTION SYSTEMS	12
2.3 EVALUATION OF INTRUSION DETECTION SYSTEMS	14
CHAPTER 3: EVALUATING HOST INTRUSION DETECTION SYSTEMS	17
3.1 HOST INTRUSION DETECTION SYSTEMS	19
3.2 DEFINITION AND DESCRIPTION OF CHARACTERISTICS.....	22
3.2.1 <i>Visibility</i>	22
3.2.2 <i>Detection Accuracy</i>	22
3.2.3 <i>Attack Resiliency</i>	23
3.2.4 <i>Efficiency</i>	24
3.2.5 <i>Transparency</i>	25
3.3 RELEVANCE OF THE PROPOSED CHARACTERISTICS	25
3.4 CONCLUSIONS	28
CHAPTER 4: EVALUATION OF VISIBILITY	29
4.1 EVALUATING AUDIT DATA	31
4.1.1 <i>Measuring Attack Manifestation</i>	31
4.1.2 <i>Measuring Data Quality</i>	32
4.2 EXPERIMENTAL EVALUATION OF VISIBILITY ON FILESYSTEM DATA.....	36

4.2.1 Filesystem Activity.....	36
4.2.2 Experimental Setup	37
4.2.3 Data Analysis	39
4.2.4 Statistics on Filesystem Activity.....	42
4.2.5 Probability of Manifestation per File.....	46
4.2.6 Probability of Manifestation per File and per Attacker Action	49
4.3 CONCLUSIONS	59
CHAPTER 5: EVALUATION OF ATTACK RESILIENCY	59
5.1 DEFINITIONS.....	60
5.2 EVALUATING INDEPENDENCY	63
5.2.1 Studying the HIDS Data Path	64
5.2.2 Defining Privilege Levels and Creating the Independency Score	65
5.2.3 Introduction to Cost Measurements	66
5.2.4 Calculating HIDS Independency.....	67
5.3 IMPROVING HIDS INDEPENDENCY	69
5.3.1 Using Embedded Hardware.....	69
5.3.2 Using Redundant Elements	70
5.3.3 Virtualization.....	71
5.3.4 Trusted Computing.....	72
5.4 EVALUATING RESILIENCY	73
5.5 CASE SCENARIO: EVALUATING SAMHAIN INDEPENDENCY	75
5.6 CONCLUSIONS	80
CHAPTER 6: EVALUATION OF EFFICIENCY AND TRANSPARENCY.....	81
6.1 EFFICIENCY	83
6.1.1 Metrics Linked to Efficiency.....	83
6.1.2 Translating Time Delays into Costs	86
6.2 TRANSPARENCY	88
6.2.1 Metrics Linked to Transparency	88
6.2.2 Relationship with Cost	91
6.3 CASE STUDY: OSIRIS AND SAMHAIN	92
6.3.1 Experimental Setup	92
6.3.2 Results	93
6.4 EVALUATING EFFICIENCY RELATED TO THE DATA COLLECTED.....	98
6.5 CONCLUSIONS	99
CHAPTER 7: INTEGRATING THE CHARACTERISTICS INTO A HIDS COST FRAMEWORK	101
7.1 SURVEY OF HIDS FRAMEWORKS	101
7.2 A COST FRAMEWORK FOR HIDSs.....	105
7.2.1 Integrating Attack Resiliency and Visibility.....	106
7.2.2 Integrating Efficiency and Transparency.....	107
7.2.3 Operational Cost for HIDSs.....	108
7.3 ESTIMATING THE COST OF A HIDS.....	109

7.4 LINKING DETECTION ACCURACY AND VISIBILITY.....	114
7.4.1 <i>Evaluation Methodology</i>	115
7.4.2 <i>Implementing a HIDS Stimulator</i>	116
7.4.3 <i>Sample Attack Evidences</i>	121
7.5 CONCLUSIONS	121
CHAPTER 8: CONCLUSIONS AND FUTURE WORK	123
FUTURE WORK.....	126
APPENDICES	129
APPENDIX A: FILESYSTEM VISIBILITY METRICS	129
A.1 FILES RELATED TO PASSWORD MODIFICATION EVIDENCES.....	129
A.2 FILES RELATED TO DOWNLOAD EVIDENCES	133
A.3 FILES RELATED TO RECONNAISSANCE EVIDENCES	137
REFERENCES.....	140

List of Tables

Table 4-1: Statistics on the Number of Files Targeted	42
TABLE 4-2: PERCENTILE VALUES OF THE FOUR DISTRIBUTIONS OF THE NUMBER OF FILES	43
TABLE 4-3: CORRELATION COEFFICIENTS VALUES	46
TABLE 4-4: PROBABILITY OF MANIFESTATION FOR A SET OF SIGNIFICANT FILES	47
TABLE 4-5: DISTRIBUTION OF THE NUMBER OF SESSIONS PER ATTACK EVIDENCE	51
TABLE 4-6: DISTRIBUTION OF THE NUMBER OF FILES PER ATTACK EVIDENCE	52
TABLE 4-7: SAMPLE FILES FOR RECONNAISSANCE EVIDENCES	55
TABLE 4-8: SAMPLE FILES FOR PASSWORD MODIFICATION EVIDENCES	57
TABLE 4-9: SAMPLE FILES FOR MALWARE DOWNLOAD EVIDENCES	58
TABLE 5-1: VALUES OF θ FOR SPECIFIC ENVIRONMENTS	73
TABLE 5-2: INDEPENDENCY SCORES FOR SAMHAIN	79
TABLE 6-1: EFFICIENCY FOR OSIRIS WITH FILE SIZE OF 142 BYTES	94
TABLE 6-2: EFFICIENCY FOR OSIRIS WITH FILE SIZE OF 13012 BYTES	94
TABLE 6-3: EFFICIENCY FOR OSIRIS WITH FILE SIZE OF 39012 BYTES	94
TABLE 6-4: EFFICIENCY FOR SAMHAIN WITH FILE SIZE OF 142 BYTES	95
TABLE 6-5: EFFICIENCY FOR SAMHAIN WITH FILE SIZE OF 13012 BYTES	95
TABLE 6-6 : EFFICIENCY FOR SAMHAIN WITH FILE SIZE OF 39012 BYTES	95
TABLE 6-7: RESULT OF THE TRANSPARENCY EVALUATION WITH POSTMARK FOR 500/7MB FILES	96
TABLE 6-8: RESULT OF THE TRANSPARENCY EVALUATION WITH POSTMARK FOR A 50/62MB FILES	96
TABLE 7-1: COST RELATED TO HIDS EVALUATION	105
TABLE 7-2. OPERATIONAL COSTS	111
TABLE 7-3: P_D AND $Cost_{IDS}$ FOR DIFFERENT VALUES OF $Cost^R$	113
TABLE 7-4: FIELDS INCLUDED IN THE DATABASE	118

List of Figures

FIGURE 3-1. COMPUTER SYSTEM FINITE-STATE MACHINE MODEL	19
FIGURE 4-1. DETAIL OF SESSION ANALYSIS	40
FIGURE 4-2. DISTRIBUTION OF FILES READ	45
FIGURE 4-3. DISTRIBUTION OF FILES WRITTEN	45
FIGURE 4-4. FILE DISTRIBUTION OF <i>IG</i> , <i>PPV</i> AND <i>NPV</i> VALUES FOR RECONNAISSANCE	55
FIGURE 4-5. FILE DISTRIBUTION OF <i>IG</i> , <i>PPV</i> AND <i>NPV</i> VALUES FOR PASSWORD MODIFICATION EVIDENCES	56
FIGURE 4-6: FILE DISTRIBUTION OF <i>IG</i> , <i>PPV</i> AND <i>NPV</i> VALUES FOR MALWARE DOWNLOAD EVIDENCES	58
FIGURE 5-1. HIDS DATA PATH	65
FIGURE 5-2. EXAMPLE SCHEME FOR HARDWARE HIDSs	70
FIGURE 5-3. SCHEME OF A SECURE KERNEL AS A HIDS	72
FIGURE 5-4. DATA PATH FOR SAMHAIN	77
FIGURE 6-1. TIMINGS RELATED TO EFFICIENCY	85
FIGURE 6-2. SAMPLE EFFICIENCY FUNCTIONS	88
FIGURE 6-3. AVERAGE TIME TO MANIFEST FOR SAMPLE FILES (IN SECONDS)	99
FIGURE 7-1. DETECTION TREE	106
FIGURE 7-2. SAMPLE ROC	110
FIGURE 7-3. MAPPING FUNCTION FOR EFFICIENCY	110
FIGURE 7-4. MAPPING FUNCTION FOR TRANSPARENCY	111
FIGURE 7-5. $Cost_{IDS}$ FOR DIFFERENT INDEPENDENCY SCORES	113
FIGURE 7-6. COMMAND LINE STIMULATOR	118
FIGURE 7-7. SELECTION SCREEN FOR THE HIDS STIMULATOR	119
FIGURE 7-8: UPLOADING AN ATTACK EVIDENCE	120

Chapter 1: Introduction

Intrusion detection systems (IDSs) monitor data (e.g., filesystem files, network packets, and memory activities) collected from a computer system to identify potentially malicious activity and to raise alerts related to the detected events. IDSs are widely deployed tools since security breaches cannot always be prevented. Therefore, discovering an ongoing attack or intrusion (i.e., a successful attack) allows the defender to react and minimize the possible damage of the attack or intrusion.

The detection of an attack is achieved by inspecting data. This data maybe acquired from different resources. Debar [DDW99] and Axelsson [Axe00b], in their respective taxonomies, divided IDSs into two groups depending on the source of the audit data. If the data collected were security logs, the IDS was called a host intrusion detection system (HIDS). If the IDS detection engine consumed network information, the IDS was called a network intrusion detection system (NIDS). While this distinction is still used in the literature, HIDSs have evolved from security log checkers into complex systems, which not only analyze logs, but also other resources including system calls [WFP99] and the files on the filesystem [KS94]. Moreover, while HIDSs were usually implemented as simple processes on the supervised system, HIDSs now include the use of embedded hardware [MA02] and virtualization [Lit05].

The quantitative evaluation of IDSs is a critical research topic that has important practical applications. Such evaluation is necessary, for example, to compare different IDSs and to determine the optimal operational point of the IDS when parameters can be adjusted. Today the focus of IDS evaluation is restricted to the ability of the IDS

detection engine to distinguish between malicious and normal activity. The two metrics to quantify the accuracy of this classification are the probability for true positives, which relates to correctly labeling malicious activity with an alert, and the probability for false positives, which relates to incorrectly labeling a set of non-malicious activity as an alert. Experiments to estimate these metrics have been performed by evaluating the accuracy of the IDS detection algorithm. Detection accuracy is usually leveraged by environmental factors to create composite metrics. These composite metrics better assess an IDS when deployed into a specific environment. Examples of environmental factors are the probability of an attack [Axe00a] and the cost related to IDS functioning [CBS06, JU01, LFM02].

Restricting the evaluation to the accuracy of IDS detection engine is often enough to precisely assess the IDS when applied to NIDSs, which are the most widely deployed IDSs. HIDSs acquire internal data from the system supervised to detect intrusions. Once overshadowed by NIDSs as a secondary tool, HIDSs are subject to renewed attention due to their effectiveness at detecting the emerging threat of insider attacks and the increasing use of point-to-point cryptography. In general, we cannot apply the same assumptions to HIDSs that were used for NIDSs. The different HIDS techniques provide benefits that cannot be quantified or evaluated solely by looking at the detection engine. HIDSs continue to evolve, and we need methods fair evaluation methods.

In this thesis, we analyzed the impact of the distinct HIDS traits by proposing a set of additional quantifiable characteristics. We argued that, for HIDSs, detection accuracy not only depends on the accuracy of the IDS classification engine's detection

algorithm, but also on the specific attributes of the data audited (visibility) and on the difficulty of corrupting the HIDS in the event of an attack towards the system supervised (attack resiliency). Moreover, composite metrics need to incorporate specific HIDS characteristics associated with environmental factors. Hence, we also quantify the impact of the overhead caused by the HIDS on the system supervised (transparency) and the timely detection of an attack or intrusion (efficiency).

1.1 Overview of the Dissertation

Chapter 2 covers the related work to this thesis. The subjects covered are IDSs and their evaluation. The section on IDSs contains an introduction on IDSs, a brief history and a description of IDSs related to existing IDS taxonomies. We described the IDS model used throughout the thesis and the key differences between HIDSs and NIDSs. We then performed a survey of current HIDS technologies. The section on IDS evaluation contains an introduction to security metrics and current evaluation approaches for both NIDSs and HIDSs. We then present recent work on evaluating IDSs.

Chapter 3 describes the concepts of the proposed HIDS evaluation methodology. We introduce a set of characteristics for evaluation: accuracy, visibility, transparency, efficiency and attack resiliency. Visibility evaluates the data collected; accuracy examines the correctness of the detection algorithm; transparency evaluates the impact of the HIDS to the system supervised; efficiency covers the impact of delayed detection of an intrusion; and attack resiliency evaluates the strength of the HIDS to

subversion attack attempts. The characteristics are compared with other proposed characteristics found in the literature. The chapter concludes with a study of the relationship between the characteristics.

Chapter 4 contains the methodology and metrics used to evaluate HIDS visibility. HIDS visibility links the data collected to attack detection. Attacks that do not manifest in the data will not be detected by the HIDS. To measure the effect of the data collected on the detection of attacks, we introduced the probability of attack manifestation and metrics related to data quality (i.e., relevance of the data regarding the attack to be detected). The metrics were applied to evaluate filesystem data, which is the data source for many HIDSs. The experiment consists of setting up honeypots and studying the file actions performed by attackers. The file actions are analyzed to provide the corresponding measures: the probability that a file is utilized by an attacker and the quality of the data related to a specific set of attack evidences.

Chapter 5 explains how to perform an evaluation of attack resiliency. We described a HIDS by means of a data path and study each entity in the data path for possible attacks. We focused on attacks towards the system supervised and calculated the cost of each attack, called an independency score. The independency score is used to estimate attack resiliency, which is the probability that an attack will succeed in subverting the HIDS. We estimated these parameters for the HIDSs Samhain and Osiris.

Chapter 6 introduces the methodology to evaluate efficiency and transparency. Both characteristics are performance-based characteristics. More precisely, these characteristics are the time to receive a notice by the HIDS and the unintended impact

of the HIDS on the host's performance, respectively. In the case of efficiency, we proposed the metric of the time elapsed between data collection and a HIDS alarm. We linked this metric to cost to understand the real impact on the host. For transparency, we measured the reduction of HIDS performance caused by the utilization of shared resources by the HIDS. We linked this metric to the cost related to decreased performance. We evaluated efficiency and transparency for two HIDSs, Osiris and Samhain.

Chapter 7 provides a discussion on how to integrate the proposed characteristics into a single framework for optimization. We discussed several HIDS optimization frameworks based on accuracy and showed that most of them can be framed as a cost framework. We showed how the proposed HIDS characteristics can be integrated into the suggested cost framework. We provided a numerical study of how attack resiliency impacts detection accuracy.

Chapter 8 contains the conclusions, a summary of contributions of this thesis and future work to expand the research initiated in this thesis.

1.2 Contributions

To create a framework for evaluating HIDSs, several research issues were resolved. First a set of characteristics were identified and defined. For each characteristic, we introduced metrics and described a methodology to obtain measures. We developed experiments to estimate these metrics. The experiments included an in depth study of filesystem usage by attackers on four honeypots during 24 days, the creation of a tool called HIDS stimulator to link visibility and accuracy, the evaluation of transparency

and efficiency for two HIDSs, and the creation of attack trees on the Samhain and Osiris HIDSs by launching a set of attacks. To optimize these metrics, we modified an existing IDS cost framework to include these characteristics.

1.2.1 Selecting Characteristics

Detection accuracy is the characteristic evaluated in most IDS evaluations. Previous research lacks the development of an evaluation for additional characteristics of HIDSs. This research showed that a set of five quantifiable characteristics improves current HIDS evaluations. The characteristics are: accuracy, visibility, transparency, efficiency and attack resiliency. We first defined these characteristics and demonstrated their importance to evaluate HIDSs.

1.2.2 Evaluating Characteristics

One reason to select the proposed characteristics was that they are quantifiable. The next step was to identify metrics related to the proposed characteristics. For visibility, we studied the probability of attack manifestation and the quality of the data before entering the detection engine. We showed that the data collected affects the overall accuracy of the system, and that this impact can be measured.

To quantify attack resiliency we introduced the probability of subversion. As data on attempts to modify a HIDS during the course of an attack is rarely available, we estimated these attempts. We restricted the scope of our research to attacks directed towards the HIDS performed as part of an attack towards the system supervised. We introduced the independency score, based on the cost of subverting sections of the HIDS that was then used to estimate the probability of subversion.

Transparency and efficiency had been informally quantified before, but this quantification had not been standardized. For example, previous studies of efficiency never accounted for the time an attack takes to manifest in the data collected by the HIDS. In the case of transparency, most evaluations were composed of a set of arbitrary performance tests, disregarding the type of resources shared between the system supervised and the HIDS. We addressed these problems by providing a methodology to collect the metrics and providing tools to integrate them into cost based frameworks.

1.2.3 Evaluating Metrics through Practical Experiments

To validate the proposed metrics we performed a set of experiments. For visibility we performed an in depth study using honeypot data. The data was collected over a period of 24 days and contained all system calls created as part of a SSH compromise. We developed customized scripts to create a filesystem centric perspective of each compromise, providing statistics of different types of file usage. We studied these results to quantify the probability of attack manifestation for each file on the filesystem. Then we extracted a set of attack evidences (e.g., password modification) and applied the proposed metrics to evaluate the quality of each file to detect the attack evidence. Finally, we measured the time from the start of an attack to the attack manifestation for each file utilized. The study validated the importance of visibility as a characteristic, as only a few files were necessary to detect most attacks. In most cases, without modifying the detector, HIDS results can be improved by selecting the right data to be collected.

The experiment on the honeypots was complemented by the creation of an evaluation tool called HIDS stimulator. Attack evidence samples as observed in our initial experiment collecting attack data were coded and organized to create evidence of attacks. This evidence was ordered depending on the data where they manifested.

We installed two HIDSs, Osiris and Samhain, and applied the methodology for estimating independency and attack resiliency. The evaluation consisted of first finding the data path and then studying the independency of the HIDSs. This second step was performed by launching a sample of attacks towards the different entities of the HIDS data path to assess the complexity and feasibility of a subversion attack. We created a numerical analysis for a Receiving Operating Curve (ROC), showing that the inclusion of the probability of subversion modifies the optimization results for HIDSs.

We evaluated transparency and efficiency on Samhain and Osiris. For transparency we first tried micro-benchmarks for the shared resources (in this case, the filesystem), but the results were inconclusive. We found through experience that macro-benchmarks, which heavily utilize the shared resources are better fitted to understand the performance decrease. Efficiency was also measured and compared for each HIDS.

1.2.4 Creation of a HIDS Optimization Framework

While many IDS optimization frameworks have been proposed, recent research has shown that most of those frameworks can be cast into a cost framework. A cost framework includes weights for each of the detection accuracy metrics. As a result,

the framework provides an estimated cost for the different operational points of the IDS. The optimal IDS operational point appears when the minimum cost is achieved. As part of this research we showed how to include the proposed characteristics in the cost framework. Visibility and attack resiliency directly modified accuracy characteristics, while efficiency and transparency are transformed into costs depending on operational parameters. As many of the proposed characteristics are interrelated (e.g., collecting more data may impact efficiency), the proposed framework provides a way to perform multidimensional optimization for HIDSs.

1.3 Publications

This research led to the publication of two conference articles and the submission of one journal article:

[1] Jesus Molina, Xavier Chorin and Michel Cukier. Filesystem Activity Following a SSH Compromise: An Empirical Study of File Sequences *In Proc. 10th International Conference on Information Security and Cryptology (ICISC 2007)*, Seoul, Korea, November 29-30, 2007 (to appear).

[2] Jesus Molina, Joe Gordon, Xavier Chorin and Michel Cukier. An Empirical Study of Filesystem Activity Following a SSH Compromise. *In Proc. 6th International Conference on Information, Communications and Signal Processing (ICICS 2007)*, Singapore, December 10-13, 2007 (to appear).

[3] Jesus Molina and Michel Cukier. Evaluating Attack Resiliency for Host Intrusion Detection Systems. Submitted to *Journal of Information Assurance and Security*.

Chapter 2: Related Work

In this chapter we first review previous work on intrusion detection systems. Then, we survey host intrusion detection systems. Finally, we discuss existing research on intrusion detection system evaluation.

2.1 Intrusion Detection Systems

The goal of intrusion detection systems (IDSs) is to identify attacks directed towards the system (or systems) supervised. The term “system(s) supervised” in the context of this thesis is used to refer to the computer system(s) monitored by the IDS. The system monitored can be a server, a desktop or any other kind of device (e.g., network equipments) [DW99].

Because computer systems can be the target of a wide range of attacks, preventive defenses like firewalls or access control mechanisms are usually insufficient to deter all possible threats. Historical incidents like the Internet Worm [Orm03] or more recent events, such the apparition of malicious worms like Code Red [MSC02], have shown that systems are far from being secure.

Research on IDSs began with two seminal works. The first was a technical report by Anderson [And80] called “Computer security threat monitoring and surveillance.” This work was later followed by Denning’s “An Intrusion Detection Model.” [Den87] Both works provide the foundation for most of the current IDS research. Since then, many research efforts have appeared to develop IDSs [Por92, Roe99, VK99, LZH02].

Several taxonomies exist to describe the possible approaches for intrusion detection. Representative taxonomies appeared in [DW99] and [Axe00b].

An important distinction is between network intrusion detection systems (NIDSs) and host intrusion detection systems (HIDSs). Historically, HIDSs worked with log files to detect intrusions. NIDSs, on the other hand, inspected network data to detect attacks. As HIDSs now also analyze data other than logs, we can extend the definition for HIDSs to “IDSs that analyze data existing on the system supervised,” while NIDSs are defined as “IDSs that analyze data outside the system supervised.”

Until recently, NIDSs were the preferred choice to supervise systems. Some strong points for NIDSs that made them the favored choice are:

- NIDSs can supervise more than one host at the same time
- The deployment location of NIDSs (outside the host) minimizes problems related to the integrity of the system supervised
- Network data have not yet reached the host, and thus attacks may be prevented

However, NIDSs also have weak points that recently gained relevance:

- NIDSs are unusable if the network data are encrypted
- NIDSs are ineffective against insider attacks not using the network
- NIDSs are relatively simple to evade [PN98]

As a result, HIDSs currently are a necessary tool for many security environments. Another common distinction between IDSs is the analysis technique of the detection engine. The techniques can be divided into anomaly based, where the data are checked against a regular usage baseline to find deviations, misuse based, where the

data are compared to a known database of signatures, and policy based, which uses a set of policy rules to find incorrect states.

2.2 Host Intrusion detection Systems

Host intrusion detection systems analyze existing data on the system supervised to identify attacks. As such, at least part of the HIDS needs to reside on the system supervised. In this section we provide a survey of HIDSs that is by no means comprehensive, but is representative to show the growing importance of HIDSs. We divided the survey into functions of the classification technique of the HIDS (i.e., policy, misuse and anomaly).

Policy-based HIDS: Tripwire [KS04], AIDE [LV], Osiris [Osi] and Samhain [Sam] are software-based HIDSs, which collect data from the filesystem according to a policy. The policy describes a set of features that should be checked, including integrity and other attributes of the filesystem. If the filesystem does not maintain the integrity characteristics specified by the policy, an alarm is raised. I³fs [PKSZ04] performs a similar integrity verification at the operating system level by using a stackable filesystem. Komoku [Mol01, MA02] also verifies file system integrity, but is implemented in the form of a PCI card, accessing the hard disk independently. LIDS [HB] and BlueBox [CC03] implement a certain restrictive policy in the kernel. This is achieved by implementing a set of sensors, which are then embedded in the kernel as kernel hooks. These sensors monitor the system calls and react when the policy is bypassed by examining the actions against a rule set. Finally, the Trusted Computing Group (TCG) [Gro03a] proposed the use of a chip, the Trusted Platform

Module (TPM) [Gro03b] as the cryptographic repository of a series of chained measurements on a system. These measurements are then compared to a policy through a mechanism called attestation. If a deviation exists, the attestation service will raise an alert.

Misuse-based HIDS: Virus and malware scanners, like ClamAV [Koj], are the most common security countermeasures against malicious activity. Virus scanners are HIDSs that verify the content of files against a database of signatures. The database contains patterns, which uniquely identifies viruses and other malware. If a match is found, an alert is raised. The scanning can be done interactively (e.g., on demand) or transparently (e.g., on access). Note that Avfs [MDWZ04] use a stackable file system to implement this functionality.

Anomaly-based HIDS: Historically, HIDSs were anomaly-based IDSs verifying log files on the host. An example is eXpert-BSM[LP01], part of the EMERALD [PN97] framework. Ph [SZ00] monitors the interaction of the processes with the operating system. If a deviation from normal usage is perceived in the system call usage, Ph concludes that something is wrong and that an intrusion has happened. Livewire [GR03] and Hypervisor IDS [Lit05] implement mixed anomaly and policy-based sensors but deploy them in a virtual machine. Virtual machines provide several benefits, including a better internal visibility and an increasing protection from attackers. In Livewire, the sensors are implemented using VMWare [VMw04], while Hypervisor IDS embeds the sensor in the paravirtualized virtual machine monitor Xen [BDF+03].

2.3 Evaluation of Intrusion Detection Systems

A possible approach to evaluate IDSs is to use international evaluation standards, such as the Common Criteria for Information Technology Security Evaluation (CC) [ISO98]. The CC provides a numerical score called the Evaluation Assurance Level (EAL), ranging from one to seven, to measure the technological achievement of a system against a certain security target, called a Protection Profile (PP). While the CC is a useful and accepted evaluation standard, there are serious drawbacks for its use for evaluating HIDSs. First, not all HIDSs come with an EAL from the manufacturer, as EALs are resource intensive to acquire both with respect to time (often months) and cost (often thousands of dollars). Second, as the EAL is tailored to a security target, EALs may or may not include features necessary for evaluation. For example, the PP for an IDS security scanner [Cor05] assumed a non-hostile environment and that no attack would be attempted to subvert it. The security evaluation for the Dragon IDS, [Ent04] assumed that the threat level of the environment was unsophisticated and that the hardware would not be tampered. These two examples show that comparing products by their EAL is only valid in the case that they share the same PP target.

To avoid these problems, IDSs are usually evaluated by analyzing specific characteristics, and creating corresponding security metrics. Good security metrics need to be quantitative. Often the evaluation of HIDSs is qualitative or lacks the necessary information to be rigorous[FOCT02]. Qualitative evaluations are easier to create, and can be useful when performed on a small set of targets. However, for most cases, metrics need to be used to quantify the characteristics analyzed. In particular,

the metrics analyzed so far have been almost always restricted to the rate of detection, referring to the number of attacks detected from a pool of attacks, and the rate of false alarms, which measure the number of alarms raised in the presence of normal behavior. [LHFK00, JU01].

The first IDS evaluation effort was conducted by Puketza [PZC+96]. This work proposes a methodology based on the creation of a software platform for evaluating IDSs. The software platform was composed of scripts generating background traffic and attacks. After this initial effort, others followed [DM02]. However, by far the most recognized effort to evaluate IDSs was performed by Lincoln Labs [LHFK00]. Lincoln Labs' intrusion detection evaluation was the first effort to standardize IDS evaluation, by including several categories of standard network exploits and other forms of malicious traffic. The approach to perform the evaluation was as follows. A set of seven weeks of training data were given to IDS vendors. The data consisted of a mixed set of labeled attacks and normal data. The vendor could then utilize this data to configure their IDS. The actual evaluation was performed with data generated separately. The results included the probability of detection for each IDS, but only if the rate of false negatives was inferior to a threshold. Many critiques appeared afterwards, raising doubts on the validity of the evaluation [McH00] or proposing more standardized approaches for evaluation [AAL+03]. However, most of them propose a similar approach to evaluate IDSs by evaluating the accuracy of the detection engine. A large discussion exists for other characteristics to be also evaluated [MHL+03, Axe00b]. We will discuss in detail the other characteristics in Chapter 3.

As part of the evaluation of HIDSs, we also analyzed the impact of the collected data on the HIDS evaluation. Killourhy [KMT04] explored data driven behavior to create a defense-centric taxonomy based on system call usage. He created an attacker-defender testbed with 25 attacks launched against a target machine vulnerable to all exploits. The trace of system calls was used to divide the 25 exploits based on the type of system call behavior. A similar study to describe system log activity was conducted by Barse and Jonsson [BJ04]. They launched a set of attacks and described their manifestations in different types of system log data. In their follow up paper, they described how to use these manifestations to automatically separate attacks from normal behavior in system logs. The results from these works are mainly qualitative. In [LX01], several information-theoretic metrics (i.e., entropy, conditional entropy, relative conditional entropy, information gain, and information cost) were proposed to study audit data for intrusion detection. The proposed metrics were then applied to three data sets: system calls, BSM logs and network tcpdump data. Entropy based metrics were later generalized in [GDF+06a, GDF+06b] for the analysis of the detection engine.

Chapter 3: Evaluating Host Intrusion Detection Systems

The quantitative evaluation of intrusion detection systems (IDSs) is a critical research topic that has important practical applications. Such evaluation is necessary, for example, to compare different IDSs and to determine the optimal operational point of the IDS. Today, the focus of IDS evaluation is often restricted to the ability of the IDS detection engine to distinguish between malicious and normal activity. The two metrics used to quantify the accuracy of such classification by the IDS are the probability for true positives, which relates to correctly labeling malicious activity with an alert, and the probability for false positives, which relates to incorrectly labeling a set of non-malicious activity as an alert. Accuracy metrics are often leveraged by external environmental factors to create composite metrics. These composite metrics better reflect the qualities of an IDS when deployed in a specific environment. Examples of environmental factors are the probability of an attack [Axe00a] and the cost related to IDS functioning [CBS06, JU01, LFM+02].

Assessing accuracy by focusing on the detection engine is often enough to evaluate IDSs when applied to network intrusion detection systems (NIDSs), which are the most widely deployed IDSs. NIDSs reside outside the host and usually collect network data¹ to detect intrusions. Since a NIDS resides outside the host, it does not directly affect the operation of the host. Since NIDSs collect network data, they have

¹ For the remainder of this thesis, if not explicitly stated, we suppose that the data collected by NIDSs is network data.

the following detection properties: 1) the detection occurs in real time, before the malicious activity appears on the host, and 2) every remote non-encrypted attack appears in the collected data.

Host intrusion detection systems (HIDSs) are another type of IDS that acquire internal data from the host to detect intrusions. Once overshadowed by NIDSs, HIDSs are subject to renewed attention due to their effectiveness at detecting the emerging threat of insider attacks and the increasing use of point-to-point cryptography.

In general, we cannot apply to HIDSs the same assumptions that were used for NIDSs. We argue that the HIDS's detection accuracy not only depends on the accuracy of the IDS classification engine's detection algorithm, but also on the specific attributes of the data audited (*visibility*) and on the difficulty of corrupting the HIDS in the event of an attack (*attack resiliency*). Moreover, composite metrics need to incorporate specific HIDS characteristics associated with external environmental factors. Hence, we also quantify the impact to the performance of the system supervised caused by the HIDS (*transparency*) and the timely detection of an intrusion (*efficiency*).

The structure of this chapter is as follows. We first describe the IDS model we will apply thorough this thesis and then the differences between NIDSs and HIDSs. The following section contains the definition of the proposed characteristics: visibility, accuracy, attack resiliency, efficiency, and transparency. We then make the case on the relevance of having selected these five characteristics to evaluate HIDSs.

3.1 Host Intrusion Detection Systems

A computer system can be represented by a finite-state machine where at any given time the system is in a state $s \in S$, S representing a finite set of states. A transition (α) occurs from one state to another when an input is applied to the system. The next state depends on the previous state and the input. A security policy is a statement that partitions states into authorized (or secure) states, referred as S^a , and unauthorized (or insecure) states, referred as S^{ua} . A secure system starts in a secure state and never transitions to an insecure state. Most formal models for computer security can be considered as interpretations of this general representation [Lan81].

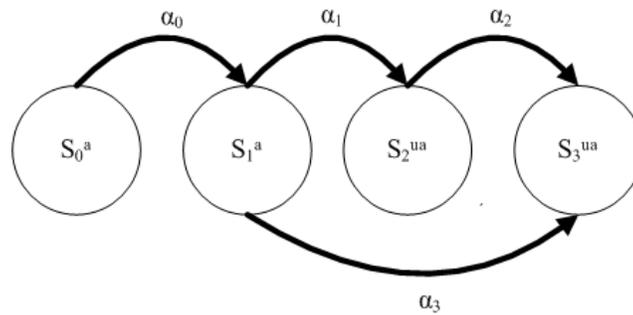


Figure 3-1. Computer System Finite-State Machine Model

An intrusion (or security breach) occurs when the system transitions to an insecure state. We define an attack as the set of inputs, transitions and states that occur as part of an intrusion. A security mechanism is defined as the procedures that enforce a given security policy. We assume that the computer system features a security policy for the three basic security properties: confidentiality, integrity and availability.

We define an IDS as a tool which goal is to accurately detect attacks by evaluating an input vector of audit data D , $D=\{d_0, d_1, d_2,.. \}$, where d_i represents the minimal data unit. There exists often a certain confusion between attack and intrusion. An intrusion

is defined to be “a deliberately-malicious software-domain operational fault that originates externally to the (technical) system boundaries” [PS03]. An attack is “an intrusion attempt and an intrusion result that has been (at least partially) successful” [PS03]. In this thesis, we suppose that every attack leads to an intrusion, and hence detecting any part of an attack leads to the detection of an intrusion. Consequently, we will refer to an intrusion or an attack indistinctively.

An attack is noticeable by the IDS if it detects any attack evidence related to this attack. Attack evidences include inputs by the attacker (i.e., commands sent, keystrokes), transitions and states that occur as part of an intrusion. From the perspective of an IDS, we can describe an attack as a set of attack evidences, $A = \{e_0, e_1, \dots\}$.

If the computer system includes an IDS, we introduce a new state, the alert state. A system transitions to the alert state when the IDS detects an attack. We denote this transition as an alert (AL). The system may transition from a secure state to an alert state if the IDS incorrectly raises an alert. We denote as TP (true positive) the probability of transition from an insecure state to an alert state, and FP (false positive) as the probability of transition from a secure state to an alert state. Additionally, we define 1-TP as the probability of false negatives (FN), and 1-FP as the probability of true negatives (TN). We suppose that once reached an insecure state, the system only transitions back to a secure state when the intrusion is detected and measures have been taken to remove the intrusion.

An IDS is composed of an agent or collection engine, a classifier or detection engine, and a notifier or reporting engine [Bis02]. The collection engine gathers the data and

may dissect the different information provided by the data collected. A single IDS may have more than one collection engine. The detection engine evaluates the information provided by the agent and outputs a decision on the current state of the system. For the remainder of this thesis, we will assume a single detection engine for an IDS and will treat two distinct detection engines as two IDSs even if they share an agent. Finally, the decision output will be transported to the consuming party by the notifier. The consuming party may be human (an administrator) or an automatic tool that may perform a set of actions to minimize or prevent damages to the system (as in the so-called intrusion prevention systems).

HIDSs are defined as a subset of IDSs that consume internal data from the supervised system. Hence, even if the detection engine resides outside the supervised system, we define the IDS as a HIDS as long as the collection is performed on the supervised system. In some instances the detection engine may consume both internal and external data to the supervised system. In that case, we suppose that we can divide the IDS into two different IDSs (i.e., one NIDS and one HIDS) and create a composite output.

We assume that the consuming party acts correctly: a false alarm will be detected as such and the system will be reverted from the alert state to a secure state. In the case of a true positive, the consuming party will take the necessary actions to revert the state to a secure state.

3.2 Definition and Description of Characteristics

This section contains the definition and a description of the proposed characteristics: visibility, accuracy, attack resiliency, efficiency, and transparency.

3.2.1 Visibility

Definition 3-1. Visibility reflects the ability of the HIDS to collect data where an attack manifests.

HIDSs collect audit data from different sources, including the filesystem [KS94, Wot05], system calls [WFP99] and memory [PFMA04]. An intrusion may or may not appear in the audit data. If any evidence of the attack appears in the data collected, we say that the attack manifests in the data. Evidently, if the attack does not manifest in the data, the intrusion will not be detected by the HIDS, hence decreasing the probability for true positives. Hence, it is necessary to study how likely a malicious event is to appear in a set of data. In Chapter 4 we provide metrics to measure the probability that an attack manifests in a set of data. We also provide metrics to evaluate the quality of data collected. This is necessary to optimize the collection of the data most relevant to detect attack evidences. The results modify directly the probability of true positives of the HIDS.

3.2.2 Detection Accuracy

Definition 3-2. Detection accuracy reflects the ability of the detection engine of the HIDS to raise alerts only in the event of an intrusion, which manifests in the audited data.

A detection engine applies different classification techniques to correctly detect an attack in the data collected. The goals of a detection engine are: 1) to detect an attack whenever it manifests in the data fed to the detector, 2) not to raise false alarms otherwise. Current evaluations of IDSs suppose that the results of evaluating the detection engine provide the overall probability of true positives and false positives. This is not true in general: the probability of detection may be modified by other characteristics of the HIDS. The probability of detection (P_D) and the probability of false alarms (P_{FA}), as described in the literature ([AAL+03, LHF+00]), will be used to evaluate the accuracy of the detection engine.

3.2.3 Attack Resiliency

Definition 3-3. Attack resiliency reflects the resistance of the HIDS to subversion attacks in the event of an intrusion.

A HIDS subversion consists in a successful attack against the HIDS that as a result modifies its output. As a part of an intrusion, the attacker may decide to corrupt the HIDS to prevent the occurrence of an alert. In the course of subversion, the attacker may modify any element in the path, from data collection to alert reporting, to compromise the HIDS result. As a result, if the attacker is successful, the attack will be missed, decreasing the probability of true positives. Since a HIDS commonly shares many exploitable elements with the supervised system, this is a fairly common strategy. In Chapter 5 we evaluate the probability that a HIDS will be subverted as

part of an attack towards the system supervised. We also describe the strong relationship between the independency of the HIDS from the system supervised and the probability of subversion. Hence, to estimate the probability of subversion, we describe the independency score, a metric quantifying the level of independency of the system.

3.2.4 Efficiency

Definition 3-4. Efficiency describes the ability of the HIDS to timely detect intrusions.

The ability to timely detect attacks is crucial for most environments. However, not all HIDSs have this ability. In some cases, a delay exists between the initial intrusion and its detection. Typically, IDSs are divided into real-time IDSs, where the attack is detected before the intrusion materialized, and off-line IDSs, where the IDS detects the ongoing or finalized intrusion. This distinction does not provide enough information to precisely study the HIDS achievements. Indeed, some off-line HIDS act faster than others. Moreover, the definition of a real-time for HIDSs is difficult to specify. HIDSs may collect data continuously but only detect an ongoing intrusion when unauthorized states manifest in the collected data. In Chapter 6 we discuss methods to evaluate efficiency. In particular, we demonstrate that efficiency also relates to the data collected, introducing the time to detect as a metric for efficiency. We also associate efficiency as a factor which increases the cost of a HIDS and provide possible mapping functions between efficiency and cost.

3.2.5 Transparency

Definition 3-5. Transparency describes the ability of the HIDS to avoid interfering with the normal functioning of the system supervised.

The functioning of the HIDS may interfere with the system supervised, causing a reduction of performance. Indeed, at least some of the collection engine must reside on the system supervised. Hence, there will be a certain performance impact on the system supervised caused by the HIDS. This undesirable effect could be of major importance in both the process of selecting a HIDS and configuring it. This impact is measured by comparing the performance of the system supervised with and without the HIDS functioning. In Chapter 6 we propose methods to evaluate transparency. As in the case of efficiency, we link transparency to an increased cost of the HIDS, and suggest possible mapping functions between transparency and cost.

3.3 Relevance of the Proposed Characteristics

In this section, we present the criteria used to select the characteristics for evaluating HIDS.

Since IDS evaluation is widely accepted as beneficial and necessary, various characteristics have been recommended. However, the goal of these characteristics often overlaps. Furthermore, we chose not to include characteristics which yielded only qualitative results, and instead focused on quantitative characteristics. Hence, characteristics which were qualitative, such as the ability to detect never-before-seen attacks [PZC+96] and the ability to detect the type of attack [MHL+03], were not

selected. We also discarded characteristics heavily dependent on external elements, such as components which handle the HIDS output (e.g., elements related to decision making, collaboration) or which evaluate the usage of the HIDS itself (e.g., ease of usage). While related to HIDSs, evaluating these external elements depends on either psychological factors, or understanding how the engine that consumes the alert operates.

These requirements provided us with five main types of characteristics: those related to resist attacks, those related to the range of attacks detected, those related to detection performance, those related to performance impact to the supervised system, and those related to the detection engine accuracy.

The study of attack resiliency was first proposed in [Axe00a, MHL+03] as the security and resistance to attacks directed at the IDS. The reason of suggesting this characteristic is that the current motivation of attackers has shifted from notoriety to economic gain. While before the attacker was willing to be discovered, now most attackers want to remain undiscovered. Hence, subverting the HIDS is more attractive as a strategic measure to avoid detection.

The amount of data collected versus the impact to the system due to data collection has already been proposed as an optimization problem for HIDS featuring high visibility [Lit05, GR03]. Many researchers suggest a qualitative description of the data collected. For example, coverage, as proposed in [MHL+03], is included in our definition of visibility: coverage is defined as the type of attacks detected and the data analyzed. Since the type of attacks detected depends on the data audited, visibility provides a quantitative estimation of coverage. Currently, visibility is implicitly

considered in the measurement of detection accuracy. However, separating visibility from detection accuracy has several benefits. From an evaluation perspective, we can divide the intrusion in smaller sets that manifest in the data and are more relevant to each specific HIDS. This will improve recent HIDS evaluations that used an indiscriminate set of attacks drawn from common attack libraries (e.g., Metasploit [Met]). Attack libraries limit the evaluation process because some HIDSs only detect the attack payload. By selecting a payload that always manifest in the data, an evaluation will most likely produce incorrect results, suggesting that the detection engine detects a wider range of attacks. By utilizing the concept of visibility, each data set is associated with a probability that an attack will manifest in it. Thus, estimating visibility prevents inconsistencies in the evaluation when only a small set of the existing attacks manifest in the collected data. For example, in the DARPA evaluation [LHF+00], the HIDSs featured a high level of true positives as the attacks that did not manifest in the data were discarded. Finally, the amount of data collected by certain HIDSs is configurable. Not all data may be collected to prevent an increase of resource usage. Hence, it is advisable to separate the study of the data collected and the detection capabilities.

Efficiency has been previously called timeliness [Axe00a], and associated to HIDS performance [PKSZ04]. Efficiency is often described in a qualitative way [FOCT02, Kah05]. In [LFM+02] the amount of damage caused by an ongoing intrusion was calculated by relating it to the time between the attack and its detection. Currently the study of efficiency is neglected, although the importance of understanding the distinction between real-time and non-real time behavior on a HIDS is acknowledged

[LFM+02]. The time to detect an attack may be the key element to select one HIDS from another, in particular, for systems which defend persistent data, where a late detection will suppose that the protected data was stolen, and hence a significant cost. The importance of transparency has been pointed out in [MHL+03] and described as an economy of resource usage. Moreover, [PZC+96] studied practical measurements for resource usage and resiliency to stress. These latter characteristics can be related as a trade-off function between visibility and transparency, as demonstrated in [SSMF03]. Transparency affects the total cost of the HIDSs, as a performance overhead in the system supervised due to HIDS functioning will likely decrease the productivity of the system. For example, a performance reduction in a busy webserver may suppose significant revenue lose due to missed clients.

3.4 Conclusions

To effectively evaluate HIDS, a set of five characteristics were introduced: visibility, accuracy, attack resiliency, efficiency, and transparency. Each characteristic was defined and described. The rationale for selecting these characteristics was discussed: identifying similar characteristics, discarding qualitative characteristics and also not including characteristics which rely in the evaluation of entities other than the HIDS itself.

Chapter 4: Evaluation of Visibility

One key difference between NIDSs and HIDSs appears in the type of data acquired and inspected by the detector. While, in Chapter 3, we defined NIDSs as IDSs that reside outside the supervised system, in practice, IDSs that inspect network data are defined as NIDSs because it is the case for a large majority of them. As most evaluations focus on NIDSs, the current evaluation methods focus on network data. However, any IDS may only detect attacks, which appear in the data they acquire. Hence, attacks, which do not appear in the data collected, will be missed. In fact, a widely understood limitation of NIDSs is that they are unable to detect attacks that do not utilize the network as part of the attack vector. This is commonly the case of attacks performed by insiders.

HIDSs suffer similar problems due to attacks not appearing in the acquired data, but the impact on detection is not well understood as the data acquired vary. A HIDS that verifies filesystem data will detect an intrusion if the attacker performs some detectable activity in the filesystem. The impact on detection is unknown, as the number of attackers who perform activity on the filesystem has not been studied. This situation is complicated by the fact that, due to performance limitations, not all data of a kind may be acquired by the HIDS. For example, HIDSs inspecting system calls will focus on a particular set, as inspecting all system calls will result in a considerable reduction of performance. Regarding filesystem data, some HIDSs may only inspect one of two files to avoid performance penalties in the case of real time detection. For evaluation, the fact that the data collected affects the outcome of a

HIDS suggests that metrics need to be provided to quantify the impact of the data collected.

In this chapter we first introduce an evaluation framework for visibility. We propose two metrics: probability of attack manifestation and data quality. The first quantifies the probability that an attack exists in the data collected, while the later quantifies the information provided by a data set regarding a specific attack evidence. To support the selection of these metrics, this chapter includes an empirical study of filesystem activity after a security compromise. Quantitative data were collected for all files targeted by attackers while reading, writing, deleting and modifying the file metadata. For each file containing some activity, data collected on file usage are used to estimate the probability of attack manifestation. Each file is also linked to the probability of manifestation of three types of attacker actions: reconnaissance, password modification, and malware download. These results are applied to calculate the metric related to data quality for each file. Finally, using the metrics we determine the most valuable files to audit.

This chapter is structured as follows. Section 4.1 provides the theory on evaluating audit data including the concepts used in this chapter and the theory needed to measure attack manifestation and data quality. Section 4.2 describes in detail an experimental evaluation of visibility on filesystem data. First, we describe the experimental setup and then the data analysis process. Then, statistics on the filesystem activity are presented. Later, we summarize the findings on probability of manifestation per file. Finally, we identify the most relevant files to audit for three different attacker actions.

4.1 Evaluating Audit Data

We refer to an attack that appears in the data audited as an attack that *manifests* in the data. We denote the data audited as D , which, for example, may represent a file or a set of files. The notation for an attack manifestation is similar to the one proposed in [LCT+02]: an attack that manifests in D is denoted as $D \leftarrow A$. An attack that does not manifest in D is represented as $\bar{D} \leftarrow A$. The data collected may also contain manifestations when an attack does not occur, which is denoted by $D \leftarrow \bar{A}$.

An attack manifests in D only if corresponding attack evidence (as defined in Chapter 3), $e \in A$, manifests in D . Evidence can manifest in D (denoted as $D \leftarrow e$) or not (denoted as $\bar{D} \leftarrow e$). IDSs operate by detecting attack evidence in a set of data. As part of the detection process, an IDS extracts a set of *features* from the data audited. Attack evidence manifesting in the data will only be detected by the IDS if it appears on the extracted features. For example, as part of an attack, an attacker may modify the root password. Evidence of the attack is the action of modifying the password, which manifests in several files, including the file containing the password file. A HIDS will be able to detect the attack if it collects data (in this case, the file) and extracts any feature containing manifestations of the evidence (e.g., integrity, file size).

4.1.1 Measuring Attack Manifestation

Let us consider a specific attack A_i , $D \leftarrow A_i$ denotes that A_i manifests in D and $P_M^{A_i}$ is the probability that attack A_i manifests in D : $P_M^{A_i} = \Pr(D \leftarrow A_i)$. Let us assume that there are N attacks and each attack, A_i , has a probability of occurrence, $\Pr(A_i)$. The

overall probability, P_M^A , that attacks manifest in D is given by:

$$P_M^A = \Pr(D \leftarrow A) = \sum_{i=1}^N \Pr(D \leftarrow A_i) \Pr(A_i) \quad (4-1)$$

If we focus on attack evidence, we obtain similar equations. Let us consider a specific evidence of an attack, e_i . $D \leftarrow e_i$ denotes that e_i manifests in D and $P_M^{e_i}$ is the probability that attack evidence e_i manifests in D : $P_M^{e_i} = \Pr(D \leftarrow e_i)$. Let us assume that there are M pieces of attack evidence and each e_i has a probability $\Pr(e_i)$ of occurrence. The overall probability, P_M^e , that attack evidence manifests in D is then given by:

$$P_M^e = \Pr(D \leftarrow e) = \sum_{i=1}^M \Pr(D \leftarrow e_i) \Pr(e_i) \quad (4-2)$$

4.1.2 Measuring Data Quality

An obvious approach to enhance the IDS' probability of detection is to increase the amount of data collected. Indeed, a larger data set may improve the probability of manifestation for each of the attack evidence types, hence improving the probability that an attack manifests in the data. The relevance of the approach, "the more data to audit, the better," depends on the accuracy of the detector. A larger amount of normal activity in the data collected will provoke an increase in the number alarms raised for non-malicious activity, commonly referred as false alarms, particularly in imperfect detectors. Performance is another aspect that needs to be considered, as collecting more data will likely impact the time the IDS needs to process the information. It is

important to keep in mind that even good detectors benefit from a reduction of the amount of data audited.

“Good” audit data is represented by attack evidence manifesting in it and non-attack evidence not manifesting in it. In other words, we need to maximize P_M^e while minimizing $P_M^{\bar{e}}$, where \bar{e} refers to the manifestation of activity which is non-attack evidence. Let us introduce two random variables, E and M , representing attack evidence and attack manifestation, respectively. $E=1$ indicates the presence of attack evidence (e) and $E=0$ indicates the absence of attack evidence (\bar{e}). $M=1$ indicates manifestation in the data, D , and $M=0$ indicates non-manifestation in D . Four combinations are possible:

- $(E=1, M=1)$ represents attack evidence that manifests in D (with probability P_M^e),
- $(E=1, M=0)$ represents attack evidence that does not manifest in D (with probability $1-P_M^e$),
- $(E=0, M=1)$ represents non-attack evidence that manifests in D (with probability $P_M^{\bar{e}}$), and
- $(E=0, M=0)$ represents non-attack evidence that does not manifest in D (with probability $1-P_M^{\bar{e}}$).

This optimization problem is similar to the problem of optimizing a detector for false positives and false negatives. In this regard many metrics have been proposed, including information gain [LX01] and Bayesian metrics [Axe00a].

Information gain is an entropy-based metric adopted from the field of information theory. In intrusion detection, this metric has been previously applied to evaluate the quality of data for misuse detection [LX01] and to evaluate the detection engine [GFD+06a, GFD+06b]. The interpretation of the metric is: given that a receiver and transmitter share information, the information gain represents the average amount of information saved by sharing instead of sending information. The metric is normalized using the overall entropy as a normalizing factor in the denominator. Therefore, the metric will always range between [0, 1]. In our case, the metric represents the information gained about attack evidence once a manifestation in the data set occurred. The normalized information gain (IG) for data set D is:

$$IG = \frac{I(E; M)}{H(E)} \quad (4-3)$$

where $I(E; M)$ describes the mutual entropy between an attack manifestation in data set D and attack evidence, and $H(E)$ consists of the entropy associated with the attack evidence. More precisely,

$$I(E; M) = H(E) - H(E|M) \quad (4-4)$$

with

$$H(E) = -\Pr(e)\log(\Pr(e)) - (1 - \Pr(e))\log(1 - \Pr(e)) \quad (4.5)$$

and

$$\begin{aligned} H(E|M) = & -\Pr(e)P_M^e \log\left(\frac{\Pr(e)P_M^e}{P_M^A}\right) - (1 - \Pr(e))P_M^{\bar{e}} \log\left(\frac{(1 - \Pr(e))P_M^{\bar{e}}}{P_M^A}\right) \\ & - \Pr(e)(1 - P_M^e) \log\left(\frac{\Pr(e)(1 - P_M^e)}{1 - P_M^A}\right) - (1 - \Pr(e))(1 - P_M^{\bar{e}}) \log\left(\frac{(1 - \Pr(e))(1 - P_M^{\bar{e}})}{1 - P_M^A}\right) \end{aligned} \quad (4-6)$$

where P_M^A is the overall probability that attacks manifest in data set D .

Using IG has faced criticism when applied to evaluating IDSs accuracy, as the link between entropy and the accuracy of the detection engine is not intuitive [CBS06]. In our case, however, the evaluation focuses on the data itself and the possible information contained on it rather than on the accuracy of the detector; hence IG is a relevant metric as it provides a measure of the useful information contained on a data set of an attack evidence. Higher values of IG indicate that the detector will be less susceptible to false alarms and missed detections. However, IG fails to accurately quantify the quality of the data where attack evidence rarely manifests, but attack manifestations do signify the occurrence of attack evidence with a high probability. This type of data is also useful, as the information contained will rarely be susceptible to false negatives, therefore providing a “safe” (while insufficient) source of information. Bayesian based metrics [Axe00a] provide more complete information on the data quality for such cases. The positive predictive value (PPV) represents the probability that given that an attack manifests, a certain evidence of an attack occurred.

$$PPV = \Pr[E = 1|M = 1] = \frac{P_M^e \Pr(e)}{(P_M^e - P_M^{\bar{e}})\Pr(e) + P_M^{\bar{e}}} \quad (4-7)$$

In order to understand the behavior in the event of no attack manifestation, a second metric is required called the negative predictive value (NPV), which is defined as:

$$NPV = \Pr[E = 0|M = 0] = \frac{(1 - P_M^{\bar{e}})(1 - \Pr(e))}{(P_M^{\bar{e}} - P_M^e)\Pr(e) + 1 - P_M^{\bar{e}}} \quad (4-8)$$

4.2 Experimental Evaluation of Visibility on Filesystem Data

Monitoring filesystem data is a common method used to detect intrusions. Once a computer is compromised, an attacker may alter files, add new files or delete existing files. The changes that attackers make may target any part of the filesystem, including metadata along with files (e.g., permissions, ownerships, inodes). In this section we apply the proposed techniques to evaluate visibility for filesystem data. We describe an empirical study of visibility that focused on attacker activity after a SSH compromise. First statistical data on the number of files targeted and the associated activity (i.e., read, write, delete, ownership, rights) is reported. Then, we calculate the probability of manifestation of each file in the filesystem. Then, three types of attacker activity (evidences) are considered: reconnaissance actions, password modification, and malware download. For each type of activity, we identify the quality of each file to detect the considered actions. With then identify the most relevant files to audit using the metrics proposed.

4.2.1 Filesystem Activity

Filesystems contain a large amount of information. Acquiring and analyzing all of the data is often infeasible as it translates into severe performance penalties and unacceptably long processing times. Thus, in most cases, only a subset of the filesystem activity is audited. Furthermore, processing files that fail to provide meaningful information to detect intrusions may result in the production of an increased number of alarms in the event of non-malicious file activity (false alarms). Hence, an important goal is to identify the files and corresponding activity monitored

to successfully detect intrusions, while minimizing both collection time and information overload on the IDS.

To describe the significance of each file for detection purposes we introduce the probability of attack manifestation, which provides an estimate on how often a file is opened in the event of an intrusion. However, the activity occurring in a file may not be always related to an attack but could also be the result of non-malicious activity. Therefore, just considering the probability of manifestation is not sufficient to fully assess the relevance of the file for detecting intrusions. Thus, the relevance of using various files as evidence of an attack (e.g., an attacker launching a reconnaissance or downloading some malware) also need to be assessed. This assessment can be based on the metrics proposed in the previous section: information gain (IG), positive predictive value (PPV) and negative predictive value (NPV).

4.2.2 Experimental Setup

To collect data on attacker activity, we used a set of four high interaction Linux honeypot computers. For details regarding the testbed architecture, refer to [PTJC05]. The experimental setup is described in more detail in [RBC07].

Software Configuration: The four honeypots ran on an identical Linux disk image: a slimmed-down installation of Fedora Core 3, updated with the latest patches as of October 10, 2006. Since the primary interaction with the system was via SSH, the installation was conducted in a text-mode environment (the X Window system and associated graphical programs were not installed). To monitor attacker activity, we used the following tools: a modified OpenSSH server to collect password attempts;

syslog-ng to remotely log important system events including logins and password changes; strace [Str] to record system calls made by incoming SSH connections; and the Honeynet Project's Sebek tool [Seb]. We modified the OpenSSH source tree by adding a single line of code that used syslog to record attempted passwords. To prevent attacks directed against strace, the program was concealed as a system script.

User Account and Password: Each honeypot had one privileged root account plus five non-privileged user accounts. Using results from a study that found the most commonly attempted usernames and passwords, we selected five usernames: admin, mysql, oracle, sarah and louise. For each username, we rotated four passwords (i.e., 'username', 'username'123, password, and 123456). After a password modification, the honeypot was redeployed and the next password was used. Two honeypots were set up with strong root passwords. The other two honeypots had root accounts that rotated the four passwords: root, root123, password and 123456.

Honeypot Lifecycle: For a quick turnaround, we used a pre-built disk image and automated scripts to manage the deployment of the honeypot. We monitored the syslog messages coming from each honeypot on 24 hour intervals to check for logins and password changes. The honeypot was redeployed after a password modification to prevent locking out other attackers. Typically, passwords were changed daily. To maximize the attackers' activity on the filesystem following a password change, we waited at least one hour before putting the disk image back onto the honeypot, running the deployment script, and continuing to monitor the live syslog data.

4.2.3 Data Analysis

During the 24 day period from November 14 to December 8, 2006, attackers from 229 unique IP addresses attempted to log into the honeypot 269,262 times (an average of 2,805 attempts per computer per day). According to the syslog data, of 269,262 attack attempts, 824 logged in successfully and 157 changed an account password. Results from an extensive analysis of the syslog data can be found in [RBC07].

The data analyzed in this paper consisted of system call data collected with a tool called strace [Str]. Strace intercepts and records all system calls made by a running process. We launched strace against the sshd daemon, switching on the built-in functionality for strace to record the activity of all the children spawned. To discriminate between compromises, we developed a script to isolate each different compromise among the strace data. We defined a compromise as a successful login followed by a bash session and all its children. Before processing the attack sessions further, all administrative activity required to transfer logs to a central database and to reimage the honeypots were removed. Using the strace data, we found 743 attacks instead of the 824 found by the syslog data [RBC07].

One reason these results differ is due to the difference in defining a compromise using syslog data (i.e., a successful login) versus strace data (i.e., a bash session). Syslog data included SCP and SFTP connections and aborted logins that were not included in the strace data. Moreover, some attackers were able to compromise the strace logging capability. We verified the data collected by strace with data collected by Sebek [Seb]. In particular, we identified strace data collection disruptions by attackers to ensure that such events were rare. We found four sessions in which strace failed to

sessions with a greater number of files read contained some type of activity. To prevent errors caused by the data collection process, we verified that the sessions ended with a specific reading activity that appeared in the login and logout. Using this procedure, a total of 421 empty sessions were found. The large number of empty sessions indicates that automatic tools are used to attempt dictionary attacks. After the tools achieve a successful login, they report the correct login and password to the attacker; no commands are executed.

The remaining analysis focused on the 322 non-empty sessions. The non-empty session were processed to find the files written, read, deleted or whose ownership or rights had changed.

The process consisted of creating a tree containing all processes launched by the attacker in a session. We then separated all system calls created by each process. Customized scripts singled out all filesystem related system calls for each process contained in a session. After this, every system call was parsed and the results stored in a database. The database contained the necessary information to recreate all filesystem activity performed by the attacker in a session. Our records contained the name of the file, type of activity, process linked to the file, and the time of usage from the start of a session. We stored this information by process ID to simplify post-processing. Figure 4-1 provides an example of the analysis of a session. Finally, all files in the database were processed to collect statistics for each session and individual statistics for each file.

4.2.4 Statistics on Filesystem Activity

In this section, we present statistics related to the number of files that were read, written, deleted, or whose ownership or rights changed for the 322 non-empty sessions in the strace data. Only unique file activities were analyzed (i.e., duplicated file activities in the same session were discarded). For example, if a file was read several times in the same session, the reading activity for the file was counted once. However, if the same file was read and written to in the same session, the file was counted twice: once as read and once as written.

Table 4-1 contains the per session minimum, maximum, average and standard deviation of the number of files read, written, deleted or whose rights or ownership changed. As expected, all attacks included many (minimum of 20) different files reads. However, more surprisingly, some attacks consisted of no write or delete activity, but included rights or owner changes. Also as expected, the average number of files read was quite high (144.7) while the average number of files written was low (32.1). More surprising was the low average number of deleted files (6.6). Also interesting was the low average number of files whose rights changed (2.2) and the large number of files whose owner changed (17.2). The average number of files whose owner changed was significantly higher than the number of files deleted and whose rights changed and equal to half the number of files written. The standard deviations varied as a function of the file activity: the number of files whose rights changed had a relatively small standard deviation (5.7), but the other file activities had standard deviations between 41.4 and 96.6.

Table 4-1: Statistics on the Number of Files Targeted

	Read	Write	Delete	Rights	Owner
Minimum	20	0	0	0	0
Maximum	484	656	418	42	852
Average	144.7	32.1	6.6	2.2	17.5
St. Dev.	78.3	90.2	41.4	5.7	96.6

In Table 4-2, we provided a set of percentile values of the distributions of the number of files for the five types of activities. As shown in Table 4-2, most attackers did not delete files (at least 70%), change the file rights (at least 70%) or owner (at least 80%). From Table 4-1, a small number of attacks led to a high number of file ownership changes. Based on Table 4-1, we expected many attacks to include file writing. However, in at least 60% of attacks, no more than two files were written. Such results help to specify the type of attack that was conducted against the honeypot. Few files written related to password modifications, while many files written along with rights modification reflected the action of installing malware. The high number of different files read was linked to reconnaissance. For example, commands like “w” and “ps” read a total of a 120 and 160 of files, respectively. However, the high number of files read complicated identifying files specifically linked to malicious activity.

Table 4-2: Percentile Values of the Four Distributions of the Number of Files

Percentile	Number of Files				
	Read	Write	Delete	Rights	Owner
10%	49	1	0	0	0
20%	57	1	0	0	0
30%	84	1	0	0	0
40%	144	2	0	0	0
50%	150	2	0	0	0
60%	166	2	0	0	0
70%	183	13	0	0	0

80%	202	23	2	2	0
90%	231	73	5	12	2
100%	484	656	418	42	852

Many attacks excluded file deletion, changing the file rights and ownership; plotting their distribution does not add insight. Figures 4-2 and 4-3, respectively, show the distribution of files read and written.

The distribution of files read (Figure 4-2) had two modes at 75 and 150 files read. The first mode appears to be caused by attacker activities involving a password change and/or software download, installation and execution. The reconnaissance actions were limited; the users currently logged on and the processes currently running were never checked. The most frequent reconnaissance action was the “uptime” command that tells how long the system ran; the command does not appear in a majority of the sessions. The second mode occurred because of the reconnaissance command “w” that tells who is logged-on and what was typed in the session. The second mode is also caused by the “ps” command, which lists the processes currently running.

In Figure 4-3, the peak at 10 files written reflects password modifications. Many sessions had 1 or 2 files with a write action. This usually happened when the attacker changed the password, but did not download and install a malicious program. Sessions with 8-15 files containing a write action usually had a program downloaded and installed, and possibly changed a password.

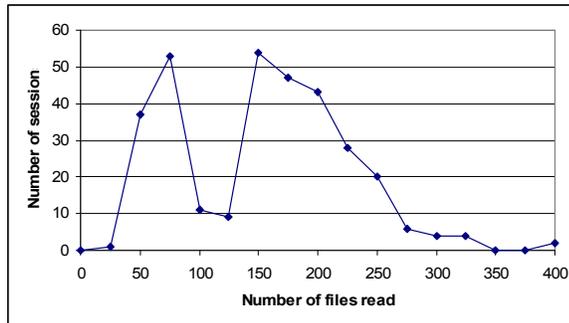


Figure 4-2. Distribution of Files Read

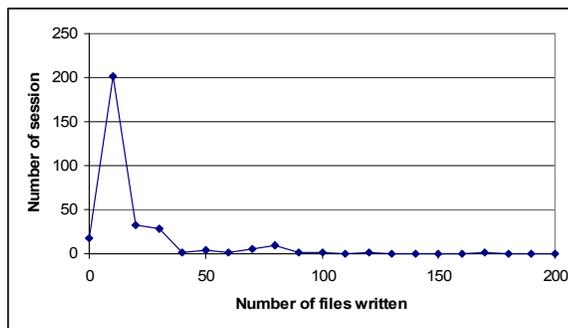


Figure 4-3. Distribution of Files Written

One important statistical result addresses the correlation between the different file characteristics. We applied Guilford's [Gui65] interpretation of the correlation coefficient:

- correlation coefficients lower than 0.2: no correlation,
- correlation coefficients between 0.2 and 0.4: low correlation,
- correlation coefficients between 0.4 and 0.7: moderate correlation,
- correlation coefficients between 0.7 and 0.9: high correlation, and
- correlation coefficients higher than 0.9: very high correlation.

Table 4-3 presents the correlation coefficients for the files read, written, deleted, whose rights or ownership changed. Based on the number of different files, we observed that there is: 1) a low correlation between files read and written, 2) a moderate correlation between files written and those the owner changed, 3) a low correlation between files whose rights and owner changed and 4) a low correlation between files written and deleted and no correlation between files deleted and the other file activities. These results indicate that the number of files read, written, deleted, whose rights or ownership changed are weakly correlated.

Table 4-3: Correlation Coefficients Values

	Read	Write	Delete	Rights	Owner
Read	1				
Write	0.24	1			
Delete	0.14	0.28	1		
Rights	0.32	0.34	0.19	1	
Owner	0.07	0.49	0.00	0.38	1

4.2.5 Probability of Manifestation per File

We compiled the activity for each file that appeared at least once per session, disregarding the type of action that produced the activity. The list included a total of 996 unique files read, 4016 unique files written, 1700 unique files containing attribute modifications and 1500 unique deleted files. For each file, we estimated the corresponding probability of manifestation, P_M^A , by dividing the number of sessions in which the file and corresponding activity appeared by the total number of sessions. Table 4-4 contains an illustrative set of files that will be discussed in this section and the associated probability of manifestation. A more complete list of files and corresponding probability of manifestation can be found in Appendix A.

Table 4-4: Probability of Manifestation for a Set of Significant Files

File	Type of Activity	Number of Sessions	Probability of Manifestation
/usr/lib/libcrack.so.2	Read	165	0.51
/usr/libresolv.so.2	Read	143	0.44
/proc/cpuinfo	Read	110	0.34
/lib/tls/libc.so.6	Read	322	1
/etc/npasswd	Write	161	0.50
/etc/shadow	Write	14	0.04
External malware	Write	121	0.38
/etc/services	Write	9	0.03
/bin/ps	Write	2	0.01

Reading files was the most common activity performed in the filesystem. However, the number of unique files read was small compared to other activities. This is partly because an important part of reading files is invoking libraries during execution and files that provide system information. Most attackers performed common actions that lead to a restricted set of libraries. For example, 165 sessions contained /usr/lib/libcrack.so.2 and 143 sessions contained /usr/libresolv.so.2, which are indications of password and network related activity, respectively. More interestingly, malicious activity was evident from the number of times certain files related to hardware and software information were read. The read action /proc/cpuinfo appeared in 110 sessions and /proc/(PID)/status in 97 sessions. Several files, mainly libraries, always manifested evidence. An example of such a file is /lib/tls/libc.so.6, which was open in every session. This indicates the importance of not only focusing on evidence of a manifestation during an attack but also on a non-manifestation when there is no attack. The file most frequently written, /etc/npasswd appearing in 161 sessions, showed that most attackers attempted password modifications. Also noteworthy was

the corruption of related password files: `/etc/shadow` was corrupted in 14 sessions while creating accounts with blank passwords. Another common malicious act was to install external programs: in a total of 121 sessions new files were created containing malware. Surprisingly, while tools carried diverse names, after decompressing the malware, many shared common files. For example, `unix2.users` appeared in 17 sessions and `psybnc.pid` appeared in another 17 sessions. The attackers' use of the latter file shows an interesting property: all processes written to `psybnc.pid` appeared cloaked as harmless services (`httpd`, `ssh`, `ntpd`, `init`). However, the files used were not cloaked as system files, thus showing that the filesystem data audit is a good vector to detect concealed malware posing as system services. IDSs often monitor written activity on system files, especially binaries to detect rootkits and Trojan horses. However, our results showed few attackers corrupted those files. Only two sessions replaced binary files. In 9 sessions, attackers modified key system files (e.g., `/etc/hosts.allow` and `/etc/services`). Rights or ownership changes appeared most often as part of the malware installation. However, they also appeared on binary files as part of installing rootkits (two sessions) and as part of erasing tracks (two sessions). Finally, files that were deleted were those created by the attacker or in user logs (e.g., `bash_login`), showing the cleanup performed by attackers. An unexpected result was that most attackers did attempt to cleanup after themselves. History files containing activity information were deleted a total of 19 times, by using direct delete commands or targeted cleanup utilities. Leftover installation residues from malware were deleted in a total of 12 sessions.

4.2.6 Probability of Manifestation per File and per Attacker Action

In this section we refine our analysis by considering the activity in each file related to a specific attacker action. Indeed, the probability of manifestation calculated in Section 4.2.5 may be inaccurate as some files may contain activity in sessions that did not provide useful information about a specific attack action. For example, dynamic libraries, such as `libc`, which most systems open as part of their normal operation, provide little information about the attacker's actions. Therefore, we need to measure the information provided by each file for detection purposes. We separated common attacker actions into three different classes of attack evidence. These three classes were identified following the analysis conducted in the experiment (i.e., reconnaissance, password modification, and malware download). For each class we associated a set of commands based on: 1) which commands are usually considered the most relevant for conducting that attacker action and 2) which files targeted by these commands are usually considered the most relevant for conducting that attacker action.

- Reconnaissance (R): Actions are performed by the attacker to gain information about the system's resources, software and its users. We selected the commands `w`, `whoami`, `last`, `ps`, `uname` and `cat` to correspond to "reconnaissance".
- Password (P) modification: Actions where the purpose is to modify a password or create a new account. We selected the commands `passwd`, `userdel` and `useradd` to correspond to "password".

- Malware download (D): Actions are related to downloading external programs and installing them. The common attacker downloads hacking tools including bots and other malware. We selected the commands wget, scp, ftp and curl to correspond to “download”.

The link between a command and file activity is not evident. The same command, issued with different options, may target different files and file activities (e.g., read, write). It is difficult to estimate which options are more commonly used by attackers. For example, “x” and “aux” are the two most common parameters used with ps (which is used a total of 138 times in 97 sessions): “ps aux” was executed 31 times in 24 sessions and “ps x” was executed 82 times in 69 sessions. “ps aux” provides additional information compared to “ps x”, such as the username. To retrieve that information, ps reads /etc/passwd. For this reason, /etc/passwd is not open when executing “ps x” whereas it is open when executing “ps aux”.

We are implicitly making the assumption that evidence of an attack implies an intrusion. In reality, attack evidence might not always reflect the existence of an attack. A password modification might very well be performed by the genuine owner of the system. However, since our analysis is only based on attack data, our objective is to link attack evidence with the related files, not to assess the importance of the chosen evidence of attack.

From the 322 non-empty sessions in the strace data, we did not find any attacker action related to reconnaissance, password modification or malware download (i.e., in 95% of the sessions we observed at least one of these attacker actions) in only 17

sessions. Table 4-5 provides the number (and percentage) of sessions associated with the different attacker actions and combinations of actions.

Table 4-5: Distribution of the Number of Sessions per Attack Evidence

Attack Evidence	Number of Sessions	Percentage
P (only)	29	9.51
R (only)	71	23.28
D (only)	22	7.21
P and R (only)	131	42.95
P and D (only)	62	20.33
R and D (only)	102	33.44
P and R and D	56	18.36

where “P”=password modification, “R”=reconnaissance, and “D”=malware download

We observed that on average, one type of attack evidence is observed in 13.3% of the sessions, a combination of two types of attack evidence are observed in 32.2% of the sessions and that all three types of attack evidence are seen in 18.4% of the sessions. As expected, attackers conducted more than one type of action when launching an attack. However, attackers seem to be interested in particular types of attacks instead of any kind of attack as illustrated by the lower percentage of sessions associated with all three attacker actions.

Table 4-6 contains the number (and percentage) of files and the associated file activity for each type of attack evidence and combination of evidence. The percentages provided in the table are based on the overall total of different files of 257. The files only involved in password modification are read, written, deleted and their rights and ownership changed. The files associated with malware download are read and write. However, the files involved in any other type of attack evidence were

only read. Since the table contains the number of files associated to single or multiple attacker actions, we expected that the number of files common to several actions would be lower than for individual actions. Indeed, only 7 files were common to the three attacker actions. Between 8 and 13 files are common to two attacker actions. And between 49 and 148 files were associated with only one attacker action. These results show that: 1) there are only a few files that can be audited to detect several attacker actions and 2) most of the file activities are read, making it difficult to differentiate attacker action from non-malicious activity.

Table 4-6: Distribution of the Number of Files per Attack Evidence

Attack Evidence	Number of Files	Percentage	Read	Written	Delete	Rights	Owner
P (only)	148	57.59	36	48	36	14	14
R (only)	49	19.07	49	0	0	0	0
D (only)	91	34.41	26	65	0	0	0
P and R (only)	10	3.89	10	0	0	0	0
P and D (only)	13	5.06	13	0	0	0	0
R and D (only)	8	3.11	8	0	0	0	0
P and R and D	7	2.72	7	0	0	0	0
Total Number of Different Files			128	113	36	14	14

We also assessed the relevance of each file associated with the three types of attack evidence separately (e.g. password modification or no password modification). For each of the three types of attack evidence, we empirically estimated the variables involved in the calculation of NIG , PPV and NPV as follows:

- P_M^A (probability that attacks manifest in data set D): number of sessions where the specific file and activity were present divided by the total number of non-empty sessions (322),

- $\Pr(e)$ (probability of attack evidence e): number of sessions that include password modification (166)/reconnaissance (248)/malware download (130) evidence divided by the total number of non-empty sessions (322),
- P_M^e (probability that attack evidence e manifests in D): number of sessions in which the specific file and activity were present for each attack evidence divided by the total number of sessions the specific file and activity were present, and
- $P_M^{\bar{e}}$ (probability that non-attack evidence \bar{e} manifests in D): number of sessions in which the specific file and activity were present for each non-attack evidence divided by the total number of sessions the specific file and activity were not present.

Let us compare respectively the average and standard deviation of IG , PPV and NPV , for all three types of attack evidence:

- Reconnaissance: IG (0.07 and 0.15), PPV (0.87 and 0.26), NPV (0.15 and 0.43),
- Password modification: IG (0.26 and 0.42), PPV (0.92 and 0.21), NPV (0.57 and 0.30),
- Malware download: IG (0.10 and 0.25), PPV (0.86 and 0.28), NPV (0.60 and 0.23).

Based on the values of IG , between the three attacker activities, files associated with password modification can be linked to attacker activity with the highest confidence on average. It is interesting to note that, on average, files associated with

reconnaissance or malware download were poorly linked to the attacker activity. To get a more accurate picture, we need to focus on the values of *PPV* and *NPV*. We observed that the three attack activities led to similar average values of *PPV*. The value of *NPV* helped differentiate the three attacker activities. The value for reconnaissance activity seems to suggest that, on average, files associated with password modification and malware download activity can be linked to these activities with high confidence (since the *PPV* and *NPV* value are high).

The next step is to refine the analysis for each attacker activity. We first discuss a plot indicating the overall results obtained for all files involved in each of the attacker activities and then a sample of files to identify in detail which files are the most relevant for identifying the attacker activity.

Figure 4-4 indicates the number of files associated with reconnaissance evidence with the values of *IG*, *PPV* and *NPV* in 10 bins of 0.1 between 0 and 1. We observe that few files provide a good source of information. However, many files exit with a high value of the *PPV*, reflecting files opened by the attacker to inspect specific aspects of the system's configuration (e.g., header files).

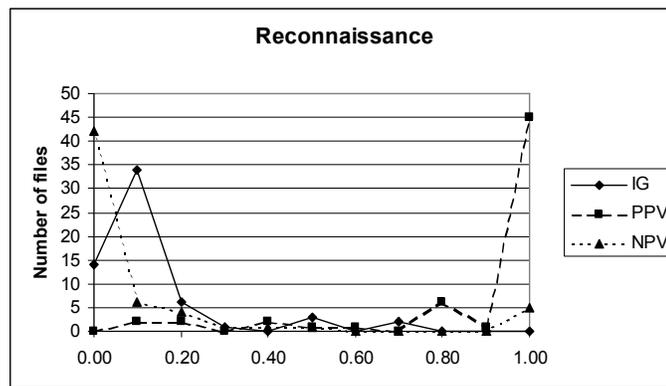


Figure 4-4. File Distribution of IG, PPV and NPV Values for Reconnaissance Evidence

Table 4-7 provides a sample of the files associated with the reconnaissance activity. A more extensive set of results can be found in Appendix A. The most relevant files to identify reconnaissance activity have a high *IG* value. No single file contains all necessary information to identify reconnaissance activity. However, */proc/x/stat* and *proc/x/cmdline* can be associated with high confidence to reconnaissance activity, as manifestations in these files seems to imply the existence of reconnaissance activity (*IG*=0.655 and *PPV*=1). While */proc/loadavg* manifests in more sessions than the previous files, the information provided is less, as it also manifests in other type of actions. Finally, libraries including */lib/tls/libc.so.6* do not provide any information (*IG*=0) as they manifest during reconnaissance activity, but also in many other actions not related to reconnaissance.

Table 4-7: Sample Files for Reconnaissance Evidence

File Name	Activity	IG	PPV	NPV
<i>/proc/x/stat</i>	Read	0.655	1	0.725
<i>/proc/x/cmdline</i>	Read	0.655	1	0.725
<i>/lib/libproc-3.2.3.so</i>	Read	0.440	0.960	0.684
<i>/proc/cpuinfo</i>	Read	0.210	1	0.349
<i>/proc/loadavg</i>	Read	0.195	0.917	0.487
<i>/lib/libnss_files.so.2</i>	Read	0	0.770	0
<i>/lib/tls/libc.so.6</i>	Read	0	0.770	0

Figure 4-5 indicates the number of files associated with password modification evidence with the values of IG , PPV and NPV in 10 bins of 0.1 between 0 and 1. We observe that many files exist with a high PPV and low IG , NPV values. These are temporary files, which were created while modifying the password. Unlike for the case of reconnaissance, there exists a set of good files for identifying a password modification evidence, with $IG=1$ and $PPV=1$, that are mostly common shared libraries.

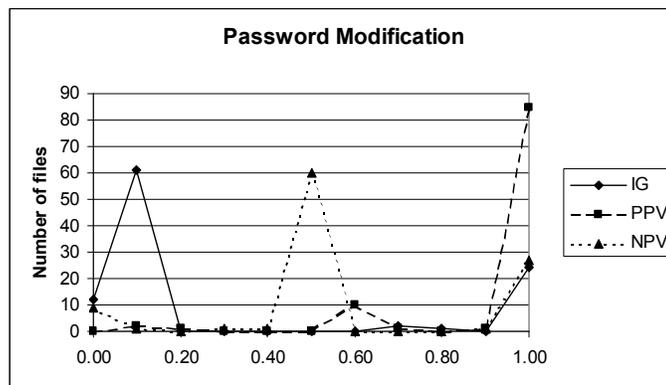


Figure 4-5. File Distribution of IG , PPV and NPV Values for Password Modification Evidence

Table 4-8 provides a sample of the files associated with the password modification activity. A more extensive set of results can be found in Appendix A. As password modification evidences are linked to few commands, each one of a set of libraries provides all the necessary information to single out this activity ($IG=1$). The file most likely to be written as a result of password modification is `/etc/nshadow`, and is therefore linked to an important information gain ($IG=0.9$). However, when the attacker creates accounts with blank passwords, `/etc/shadow` is written instead. The relevance of `/etc/shadow` is reflected by $NPV=1$. Reading `/etc/shadow` is necessary to modify a password. However, it is also a target for reconnaissance activity, as it

contains information about the users, undermining its relevance for password modification. The reading of `/etc/passwd` is an activity performed by many other types of actions, and therefore not suitable to provide high quality information for password modification.

Table 4-8: Sample Files for Password Modification Evidence

File Name	Activity	IG	PPV	NPV
<code>/usr/lib/libgmodule-2.0.so.0</code>	Read	1	1	1
<code>/usr/lib/libgobject-2.0.so.0</code>	Read	1	1	1
<code>/lib/libpam.so.0</code>	Read	1	1	1
<code>/etc/shadow</code>	Read	0,951	0,988	1
<code>/etc/nshadow</code>	Write	0,970	0,988	1
<code>/etc/shadow-</code>	Write	0,021	1	0,495
<code>/etc/passwd</code>	Read	0	0,516	0

Figure 4-6 indicates the number of files associated with malware download evidence with the values of *IG*, *PPV* and *NPV* in 10 bins of 0.1 between 0 and 1. We observed that the three big spikes are associated with files related to writing malware. These files have low *IG* and *NPV* values, since they manifest rarely, but a high *PPV* value since they manifest for this activity only. A set of files, including configuration for network files and shared libraries provide moderately relevant information to identify malware downloads ($IG=0.8$).

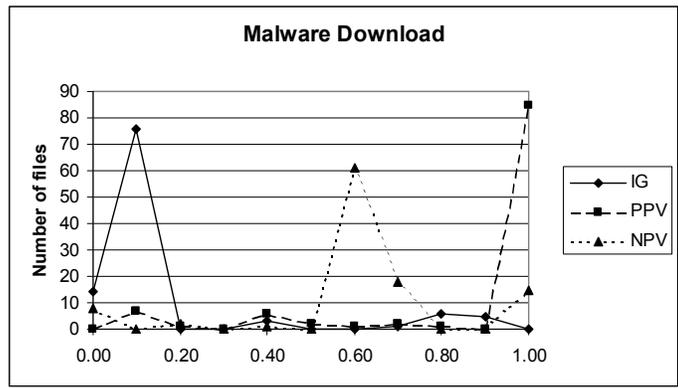


Figure 4-6: File Distribution of *IG*, *PPV* and *NPV* Values for Malware Download Evidence

Table 4-9 provides a sample of the files associated with the malware download activity. A more extensive set of results can be found in Appendix A. Most attackers utilize the wget utility to download malware. Hence, the corresponding configuration file for this utility, /etc/wgetrc provides the most information on malware download activity. We observed that common libraries (e.g., /usr/lib/libk5crypto.so.3 and /usr/lib/libgssapi_krb5.so.2) are a good source of information. In many cases malware was downloaded by a web address instead of using an IP address. That is shown by the usage of /etc/resolv.conf. All independent malware downloaded provides some information. However, as malware manifests in different forms, the information provided by each of these files is not clearly reflected in the value of *IG*. On the other hand, the relevance of the malware file (e.g., sendq.tgz and boti.zip) for intrusion detection is evident by the value of *PPV*, which is always one. Finally, other configuration files, like /etc/localtime, do not provide information on malware download, as its usage also appears in other actions.

Table 4-9: Sample Files for Malware Download Evidence

File Name	Activity	IG	PPV	NPV
/etc/wgetrc	Read	0.831	1.000	0.955
/usr/lib/libk5crypto.so.3	Read	0.822	0.922	1.000
/usr/lib/libgssapi_krb5.so.2	Read	0.822	0.922	1.000
/etc/resolv.conf	Read	0.723	0.939	0.963
sendq.tgz	Read	0.078	1.000	0.632
Boti.zip	Read	0.030	1.000	0.610
/etc/localtime	Read	0.004	0.415	0.697

4.3 Conclusions

In this chapter, we presented two metrics to evaluate the visibility of a HIDS: probability of attack manifestation and data quality. The metrics were then applied to the results of an empirical study of the activity performed by attackers on filesystem data after SSH compromises. The objective was to identify the most relevant files to audit for HIDSs based on filesystem data. As expected, the probability of manifestation was insufficient to fully assess the value of data as an audit source for HIDSs. The proposed metrics, information gain, positive predictive value and negative predictive value provided the necessary information to understand the significance of each file for certain malicious activities. We empirically showed that the link between certain attack evidences (i.e., reconnaissance, password modification, malware download) and files can be quantitatively estimated with the help of the proposed information theory based metrics.

Chapter 5: Evaluation of Attack Resiliency

As already mentioned, HIDSs reside, at least partially, inside the system supervised. Such architecture provides an exposure to attacks. In comparison, every component of a NIDS is deployed outside the system supervised. This separation shows that an attack towards the system supervised will not affect the NIDS. In other words, the state of the system supervised, secure or insecure (as seen in Chapter 3) will not impact the one of the NIDS. While an attacker may still launch attacks directed at the NIDS, in practice, this rarely happens. In most cases, an attacker will choose a simpler strategy to avoid detection, i.e., evading the IDS. Since an attack against a

NIDS will not be affected by the system supervised, the probability of occurrence of such attack can be evaluated with common methods and security metrics (e.g., risk evaluation [Bla01]). On the other hand, HIDSs require a more complex study of attacks against the system supervised. An attacker is likely to escalate privileges on the system supervised if the attack is successful. Since HIDSs usually share many resources with the system supervised, the privilege escalation on the system supervised may provide unrestricted access to HIDS elements. This fact leads to the appearance of new attack vectors for disrupting the normal operation of the HIDS. These attack types are simple and well documented, and hence popular [Half97]. This chapter presents HIDS resiliency as a metric of HIDS strength against attacks towards the HIDS through the system supervised. HIDS resiliency is the probability that the HIDS will not be subverted in the event of an attack against the system supervised. To estimate HIDS resiliency, we introduce a metric reflecting the HIDS's independency. We then estimate these metrics for the Samhain [Wot05] HIDS.

5.1 Definitions

HIDSs may suffer integrity, confidentiality or availability attacks. If the attacker's objective is to perform an attack against the system supervised, availability and confidentiality attacks will be of limited use. Indeed, most availability attacks against the HIDS are considered to be attacks against the system supervised and are labeled as alarms. Confidentiality attacks provide limited information on the system supervised, which is inferred from configuration files and output data.

Integrity attacks are more important with respect to attacking the system supervised. Indeed, a clever change of the HIDS allows the attacker to circumvent the HIDS. If the modification leads the HIDS to miss an otherwise detectable attack, we declare the HIDS to be subverted. While the attacker may compromise the integrity of any HIDS element, we can reduce the target for the attacker to modifying the HIDS's output in the event of an attack towards the system supervised. This definition implies that the attacker's objective is not to gain control of the HIDS, but to change the HIDS output, which may be a simpler problem.

Definition 1: A HIDS, H , with output, O , resulting from a set of data, D , is **subverted** by an attack towards the system supervised if for the same set of data, D , the attacker can change the output to $O' \neq O$.

For the simplest type of HIDS, implemented as a function, h , with inputs $X = \{x[1], x[2], \dots, x[N]\}$ that are classified as normal data or as an attack, $h: X \rightarrow \{0,1\}$, the attacker subverts the system if the HIDS's response is the inverse given X , i. e., "0" instead of "1". Depending on the HIDS deployment characteristics, HIDS subversion varies in complexity. The following definition describes a metric of this complexity:

Definition 2: We define HIDS resiliency to subversion as the probability that the HIDS will not be subverted in the event of an attack to the system supervised.

In the remainder of this thesis, we refer to HIDS resiliency as the HIDS resiliency to subversion attacks. To evaluate HIDS resiliency, we do not make any assumptions about the initial privilege or the privilege level gained on the system supervised. However, we assume that the attacker has no special implicit privilege on the HIDS. If the attacker decides to subvert the HIDS, we need to evaluate the probability of the attacker succeeding. There are two possible routes to perform a subversion attack: through the system supervised as part of the attack to the system supervised, and as a separate attack directed specifically to the HIDS. In the case of HIDSs, usually the easiest approach for an attacker is to subvert the HIDS using the system supervised as the attack vector. We assume that out of band attacks are more costly than attacks through the system supervised. If shared elements exist between the HIDS and the system supervised, an attacker may corrupt or tamper with elements from the system supervised to subvert the HIDS. Hence, a primary goal for the HIDS is to be independent of the system supervised, so that an attack launched against the system supervised will not impact the HIDS resiliency. To estimate resiliency, we first evaluate *HIDS independency* as the level of isolation between the HIDS and the system supervised. The more independent a HIDS is, the more resilient will it be to attacks against the system supervised.

Definition 3: We define HIDS independency as the level of isolation between the HIDS and the system supervised for a certain privilege level on the system supervised.

The study of HIDS independency varies depending on the privilege level achieved by the attacker on the system supervised as part of an attack. Each privilege level may expose certain elements to the attacker and this exposure may exist exclusively for this privilege level.

5.2 Evaluating Independency

If the HIDS is isolated from the system supervised, studying HIDS resiliency does not depend on that host and the problem is similar to that of studying the security of other computer systems. However, achieving complete isolation is not possible on HIDSs. At a minimum, the data collection agent must reside inside the system supervised and, in many cases, other HIDS components are shared with the system supervised. The existence of common elements between the HIDS and the system supervised provides a vector to attack the HIDS.

Definition 4: We define a HIDS as perfectly independent if no shared mechanism exists between the IDS and the system supervised for all privilege levels on the system supervised.

A property of a perfectly independent HIDS is that its resiliency does not depend on the system supervised. The definition does not imply that a perfectly independent HIDS is resilient to all attacks, as other attack vectors to subvert the HIDS may exist without using the host supervised. For example, a PCI card may act as the monitor of a platform and display a very high independency. But if the PCI card features a

network connection (independent from the host), it may be attacked through this connection. However, this vector of attack may be studied separately from the system supervised.

To measure the independency of the system, we introduce the independency score. The independency score is defined as a cost based metric. It describes the effort of subverting an element of the HIDS by the attacker as part of an intrusion to the system.

5.2.1 Studying the HIDS Data Path

A study of HIDS independency starts with identifying the elements employed by the HIDS, from data collection to alert reporting. To subvert the HIDS, the attacker may attempt to launch a range of attacks against any element of the HIDS. The situation is further complicated as most HIDSs require the use of common elements with the system supervised to transfer or modify data (e.g., network card, hard disk controller, kernel driver). Moreover, for complex systems, redundant elements, each one with different properties, may be used for data collection and alert reporting. An attacker may corrupt any element in the HIDS data path from data collection to alert reporting. A simple HIDS model [Bis02] consists of three parts: the agent, which collects the data; the director, which corresponds to the detection engine; and the notifier, which reports the results from the director. We assume that all HIDSs feature a single director, but can consist of many agents and notifiers. As seen in Figure 5-1, both the agent and the notifier may be composed of further active intermediaries. These intermediaries are called proxies. The communication paths between proxies are

called communication channels. In Figure 5-1, we represent each communication channel as S_i , and each proxy as P_i .

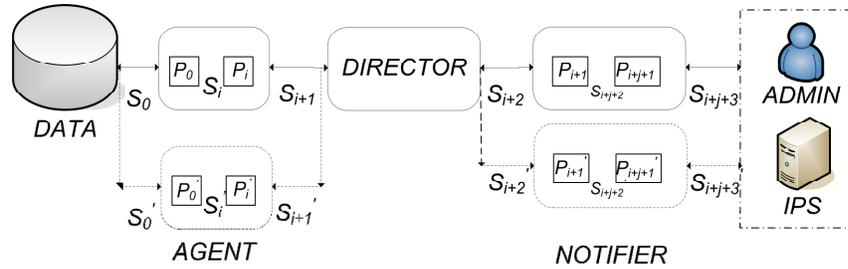


Figure 5-1. HIDS Data Path

We describe each HIDS element in function of the type of resources employed by each proxy and communication channel. For example, a hard disk controller uses firmware and hardware, and a detection engine resides in memory while storing configuration files on a filesystem. An attacker may exploit an element by tampering with any of the shared resources. Normally, subversion attacks, which involve restarting the HIDS, will be notified. Hence we will only evaluate shared resources while the HIDS is running in a normal mode.

5.2.2 Defining Privilege Levels and Creating the Independency Score

Once the HIDS elements identified, the complexity needed to exploit them to subvert the HIDS should be evaluated for the different privilege levels. We assumed that no privilege level on the system supervised immediately led to a privilege level on the HIDS.

While taxonomies of privilege levels differ, for this thesis, we used the taxonomy proposed in [Web98]. This taxonomy refers to the privilege level on the system supervised. For example, the superuser privilege level entails access to the software

executing on the system supervised, including firmware and possibly the BIOS. The physical access privilege level grants access to the system's hardware. Our goal is to evaluate the effort, time and resources needed by the attacker to realize the threat, i.e., subvert the IDS in the event of an attack to the system supervised. We assume that the more complex the modification of a specific HIDS element is, the closer the element is to being isolated from the system supervised, and thus the higher the independency score.

5.2.3 Introduction to Cost Measurements

Security is often measured from the defender's perspective. The simplest method is to estimate the dollar amount saved due to the avoidance of security breaches. The most common measure to calculate the risk of a harmful event is the annual loss expected (ALE), which is computed multiplying the expected rate of loss by the value of the loss. More sophisticated metrics have been proposed to quantify security investment, including the return of investment (ROI) [Bla01] and the internal rate of return (IRR) [GL02]. We find similar methodologies for IDSs: in [LFM+02] a model is proposed built on concepts of risk analysis, by dividing the specific costs into operational costs, damage costs and response costs.

For our methodology, however, the cost inherently relates to the adversary's perspective, rather than the defender's. Our objective is to evaluate the effort, time and further resources needed by the attacker to realize the threat, i.e., subvert the IDS. These metrics are common in cryptography, where the security of a cipher depends on the time an adversary takes to decode a message, given a specific set of initial

conditions. Outside cryptography, the evaluation of security metrics has proven elusive, as many uncertainties exist while assessing the adversaries' cost. To avoid these uncertainties, practical on-site vulnerability analysis is usually preferred, relying on security professionals with the task of attacking the system [JO97]. This approach is often referred in literature as using red-teams [RVK05].

Some theoretical methods exist to calculate this cost without hiring red-teams. The sum of the cost incurred by the attacker to achieve his/her goals is referred by Schechter [Sch04] to the cost-to-break. Schechter's work estimates this cost by measuring the complexity of obtaining the vulnerability in the market, in the same fashion as we rely on the market to find the estimated price for equipment. In the case of software, a bug auction model is proposed for software exploits, where only software with unknown vulnerabilities will have a cost (i.e., software with known vulnerabilities will have no cost for the attacker to be exploited). While Schechter's bug auction model is elegant, it is difficult to apply in practice. However, using the same rationale, we may calculate the cost-to-break by estimating the amount of money needed to hire an expert able to compromise the desired HIDS element. This market model could provide the required costs to evaluate independency.

5.2.4 Calculating HIDS Independency

To calculate the independency for each HIDS element, the simplest metric is the introduction of a relative cost. Its use is a common practice to evaluate IDSs [JU01, LFM+02, CBS06]. The evaluator sets a specific and known cost baseline. If the HIDS evaluation is performed after the HIDS deployment, the simplest technique is to set

the baseline to the cost for the attacker to achieve the privilege level for which independency is evaluated on the system supervised. Once the baseline is set, the relative effort for the attacker to subvert the HIDS needs to be evaluated for each HIDS element. If the HIDS evaluation were performed using another baseline on another system, the practitioner can easily modify the cost by simply specifying the ratio between both baselines. If the element does not contain any shared resource with the supervised system for that privilege level, we assign infinity (∞) as the independency score. In HIDSs where attacks are detected in real or near real-time, the cost to subvert the monitor after the intrusion might be high, as the attacker only has a small window of opportunity to compromise HIDS elements before detection. The cost of subverting the HIDS after the attack is bounded in time by the monitor's reaction time (efficiency). Efficiency will be studied in detail in Chapter 6. Hence a low efficiency increases the cost of certain attacks, e.g., brute force attacks on the HIDS administrator key. Another issue is self-monitoring: some HIDSs monitor their own elements and consequently trigger an alarm if attacked. While performing the cost based analysis, these factors should be taken into account and increase accordingly the independency score.

After assigning a independency score to each HIDS element for a privilege level, we define the lowest independency score assigned to an element for this privilege level as the overall independency score for that privilege level, denoted as $Cost^{priv}$. If the element is redundant, i.e., another data path exists from data collection to alert reporting, only the highest score is used as the overall independency score, as an attacker needs to subvert at least one element on each data path to subvert the HIDS.

Attackers will exploit the shared element with the lowest independency score to subvert the HIDS. Low $Cost^{priv}$ values hence imply low resiliency values.

5.3 Improving HIDS Independency

In this section we describe techniques to improve HIDS independency. The techniques fall into the following categories: 1) decrease the amount of shared resources, beginning with the resources which are easier to exploit; 2) provide redundant paths to avoid single points of failure and to increase the attacker's effort; 3) increase the complexity of exploiting shared resources by using improved access control systems, encryption, or other techniques to mitigate the threat and to increase the independency score of the shared element. Next we discuss some specific techniques based on the previous categories.

5.3.1 Using Embedded Hardware

A technique to reduce the amount of shared resources at the supervisor privilege level consists of transferring most of the unsecured communication channels and proxies to hardware. This will actually increase the possible attack vectors at the physical level. Nevertheless, the cost for an attacker with supervisor access to subvert the system monitor will increase, and most attacks achieve supervisor privilege level. Recent experiences using embedded systems as IDSs can be found in [Mol01] and [PFMA04]. In both cases a single board computer in the form of a PCI card was used as an active monitor of filesystem integrity and memory, respectively.

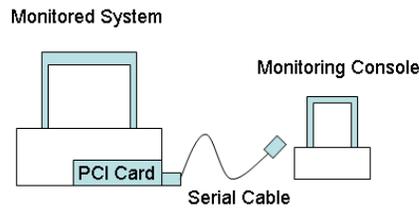


Figure 5-2. Example Scheme for Hardware HIDSs

While intuitively these systems may seem secure, a rigorous analysis of the path is still necessary. In particular, [Mol01] needs an IDE controller to retrieve the data from the hard disk, which is a shared element. As novel hardware continues to appear with field-upgradeable capabilities, we need to account for attacks launched to subvert hardware [HD04]. While extremely rare today, as the expertise of the average attacker continues to rise, the feasibility of such attacks will also increase. This shows that the problem of intrusion detection subversion not only depends on the security of the proxies (i.e., agent, director, notifier), but also on the existence of shared resources. As for [Mol01], a shared proxy also exists for [PFMA04]: the DMA controller and a communication channel, the PCI bus, which may be accessed by both the board and the system supervised.

5.3.2 Using Redundant Elements

In [BGFI+98], the authors propose the use of autonomous agents to overcome a single point of failure. To prevent subversion attacks, they propose to deploy distributed collectors (agents) to monitor data. If one of the nodes failed to report an attack or was subverted, the rest of the detection system continues working. To study

such systems, all redundant paths created by each autonomous agent must be considered, and the cost for the attacker may increase as a result.

5.3.3 Virtualization

The rise of virtualization technologies (e.g., Xen [BDF+03], VMware [VMw04]) provides new techniques to perform intrusion detection. Clients inside virtual machines can be inspected by external agents [GPMB03]. The virtual machine monitor can enforce the separation between shared resources from HIDS and supervised system, even in the case the attacker achieves supervisor privileged on the system supervised. These systems are complex, as the path used by applications to communicate with each other is non-obvious and depends of the specific virtual machine technique implemented. For example, [Lit05] employs Xen to perform intrusion detection, while [GR03] utilizes VMware. These systems, however, display great potential as IDSs.

Finally, current advances in CPU, I/O and DMA isolation [Win03] provide the necessary features to create a Secure Kernel (SK). SK executes in a privileged, secure mode, capable of monitoring the others but inaccessible to them (see Figure 5-4). In this architecture, the virtual machine monitor spawns two or more operating systems. Intel's Safer Computing Alternative (previously known as Lagrande technology [Gra03]) advocates this approach to improve the security of future systems, which provides an even greater separation for shared resources.

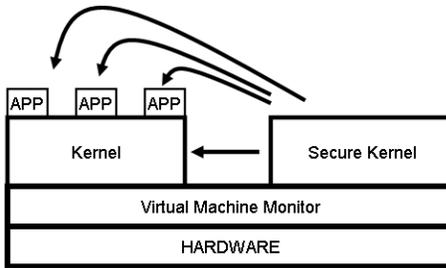


Figure 5-3. Scheme of a Secure Kernel as a HIDS

5.3.4 Trusted Computing

If all proxies in the path between data collection and alert reporting were to be checked for integrity and the data exchanged between them also checked for integrity, the alert report will be trusted or known to be tampered with by the attacker. The Trusted Computing Group (TCG) [Gro03b], formerly known as Trusted Computing Platform Alliance (TCPA), uses this approach to provide a “measured boot”. Every element measures the next element of the booting process by creating a digest of the element. The digest is then stored into an embedded, possibly tamper resistant, cryptographic chip, called the Trusted Platform Module [Gro3a]. As HIDSs have a path between collection and reporting, each element may measure the next one before passing the data. This architecture enables the remote verification (also known as attestation) of all the actors along the HIDS path, so that the output correctness may be verified by means of validating a signature as part of the data. In [SPvD05] a similar method was proposed to perform fine-grained attestation on executing processes to assess the integrity of the output given a certain input.

5.4 Evaluating Resiliency

The independency score evaluates the relationship between the HIDS and the system supervised by identifying shared elements and their possible use as attack vectors against the system supervised. In this section we describe how to estimate HIDS resiliency based on environmental factors and the independency score.

The two main questions we need to answer are: 1) Does the attacker care to be detected? and 2) Does the attacker have the necessary skills to perform an attack against the system supervised?

The motivation for the attacker to launch an attack against the system supervised could depend on the services provided by the system supervised and the type of organization. For example, if the system is an informative web server, the attacker probably will not care about being detected in the event of web site defacement. HIDSs stand in most cases as the last layer of defense. Hence access controls, firewalls and NIDSs possibly have already “cleaned” most of the less dangerous and automated attacks. We introduce the attacker motivation as a variable θ , $0 \leq \theta \leq 1$, where $\theta = 0$ is the extreme of none of the attackers care about being detected (and hence the resiliency will be always 1) and $\theta = 1$ all attackers care and will attempt if possible to subvert the HIDS. To consider only the worst-case scenario, we set θ to 1. Table 5-1 provides some values for θ in specific environments. These values are based on expert judgment. Therefore, the order of magnitude is more important than the absolute value itself.

Table 5-1: Values of θ for Specific Environments

Value of θ	Possible Scenario
0	No subversion attempts
0.1	Unprotected home machine
0.2	Unprotected university machine
0.3	Protected home machine
0.4	Protected university machine
0.5	Small business
0.6	Large business
0.7	Government institution
0.8	Military
1	All attackers attempt subversion

We already discussed how to estimate the independency score of various shared elements compared to the baseline of finding a vulnerability on the system supervised for different privilege levels. Thus, we assume that the attacker has enough skills to reach that privilege level. We now link the independency score introduced when evaluating HIDS independency, $Cost^{priv}$, with the probability of subverting the HIDS. If $Cost^{priv} \ll 1$, the probability that the attacker will succeed in the attempt is quite high, as the attacker has shown enough skill to easily subvert the HIDS, $P_{success}^{priv} \approx 1$. However, for the case of $Cost^{priv} \gg 1$, the probability that the attacker will succeed is quite low, $P_{success}^{priv} \approx 0$. In the extreme case that the HIDS is perfectly independent and $Cost^{priv} = \infty$, we have $P_{success}^{priv} = 0$, as we do not consider out of band attacks.

When discussing the possible probability values of subverting the HIDS, we did not include the attacker's motivation. This motivation is assumed to be independent of the probability of subversion. When calculating the HIDS resiliency, P_{res}^{priv} , we combine both factors in the following equation:

$$P_{res}^{priv} = 1 - \theta P_{success}^{priv} \quad (5.1)$$

The overall resiliency is computed for all possible privilege levels where the resiliency is first weighted by the frequencies of attacks (the frequencies are associated with the level of privilege reached through the attack):

$$P_{res} = \sum_{priv \in S} \alpha^{priv} P_{res}^{priv} \quad (5.2)$$

where S is the set of possible privilege levels on the system supervised and α^s are the frequencies of attacks reaching that privilege level.

5.5 Case scenario: Evaluating Samhain Independency

HIDS independency is a deployment characteristic. Consequently, even for the same HIDS, results may vary depending on how the HIDS is deployed. Our aim is not to evaluate independency of the specific HIDS but rather to illustrate the proposed method for a particular case: evaluating a host-based file integrity verifier. We estimated the independency scores of Samhain [Wot05], an integrity verifier. Samhain is executed as a daemon on the system supervised. It can be run on many platforms (i.e., Windows with Cygwin, Unix, Linux). Samhain works by first scanning the system to create a “baseline” database of the filesystem files. This database is consulted when future scans are performed. If key differences exist between the database and the current scan, such as file size modification, file creation/modification time, new/removed files, etc, they are reported to the logging mechanism. Samhain can be set up to scan at specific time intervals and email the administrator the reports. Logs are generated on the host, on which the daemon is

running, through syslog. Samhain can log through stderr, email, file, pipe, syslog, RDBMS, central log server, external script and IPC message queue. In our implementation, Samhain logs to a local CD burner and reports by an email to the administrator. Samhain was deployed on a server, with an installation of Gentoo Linux. The version of Samhain used was 2.3.1. We deployed Samhain using the default installation options.

To illustrate our method, we restricted the independency evaluation to the supervisor privilege level ($Cost^R$). The selected baseline cost is the cost to achieve root access for an outsider. In this case, the baseline models the effort of finding an unknown vulnerability in the software that runs on the system supervised, which is up to date. We estimated the independency score of the other attack vectors to subvert the HIDS compared to the baseline. In other words, we estimated how much more (or less) effort is needed for the attacker to subvert the HIDS for various attack vectors compared to finding an unknown vulnerability in the software running on the system supervised. Note that the exact values of these independency scores are less important than their order of magnitude so that the most easily launched attack vector is correctly identified.

In our implementation, Samhain reads the files specified by our policy, verifies the integrity against a database and then logs the result on a CD, while sending a message with the logs to an email address. By default, logs, messages and configuration files are unencrypted. The HIDS elements and data path are represented in Figure 5-4.

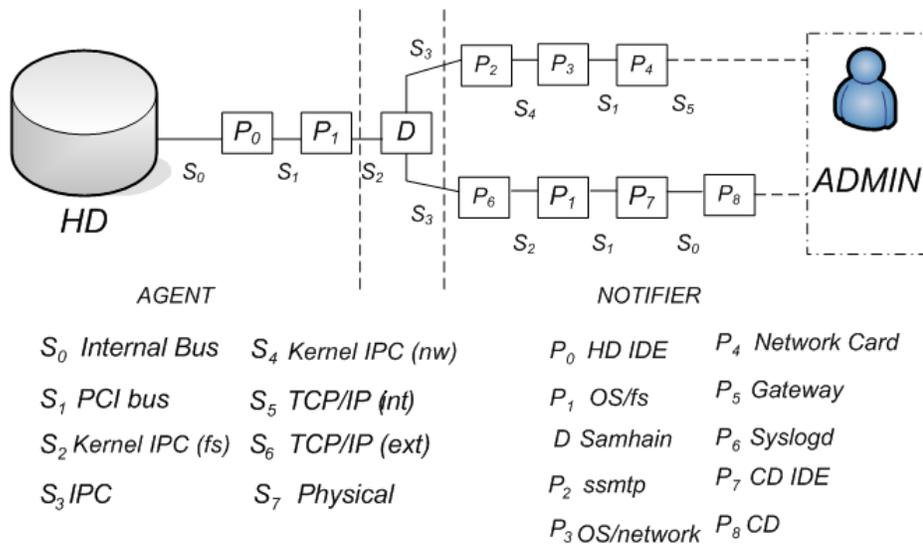


Figure 5-4. Data Path for Samhain

HD is the hard disk containing the files subject to inspection. P_0 represents the hard disk Integrated Drive Electronics (IDE) controller, P_1 the operating system kernel (IDE driver, filesystem) and both elements are part of the agent. The Samhain process acts as the detection engine. Samhain also uses a filesystem to store the configuration files. Two different notifiers are used: consequently we studied both paths. P_2 represents a smtp process, the mail server. P_3 represents the kernel network stack, network card driver and P_4 the network card. For the redundant path, P_6 is the syslogd login daemon, P_7 stands as the CD IDE controller and P_8 the filesystem on the CD. For the communication channels, S_0 are the IDE internal registers, S_1 the PCI Bus, S_2 the kernel to user space communication, S_3 the inter process communication, S_4 the user space to kernel communication and finally S_5 represents the internal network connection. The rest of the path is outside the host and hence out of scope.

After we defined the data path of the HIDS, we evaluated the independency score of each of the shared elements for the supervisor privilege. We compared the different scores of subverting the HIDS for each element for the supervisor privilege level. The results, shown in Table 5-2, are based on expert knowledge. We estimated that modifying the internal operation of hardware proxies or attacking restricted communication paths (e.g., the PCI bus) requires either insider help or an intimate knowledge of the system, so we assigned $Cost^R = 10$ (i.e., an effort 10 times higher than that of finding an unknown vulnerability in the software running on the system supervised). Many current controllers provide a mechanism to upgrade the internal firmware, hence providing an easier attack vector (e.g., replacing the firmware with a corrupted version). While the cost may fluctuate depending on the specific hardware, we set $Cost^R = 5$ for firmware-based hardware.

Software is simpler to subvert: specifically there are tools that can automatically corrupt the detection engine itself, so we assigned $Cost^R = 0.3$. However, the attacker may also corrupt other proxy software, like smtp. As this requires some expert knowledge to correctly modify the communication, we estimated $Cost^R = 0.7$ for both the network OS stack and smtp.

The lowest independency score is associated with S_2 . Many rootkits exist on the Internet that are designed to modify the operating system. Therefore, user space applications will provide deceptive outputs. As rootkits are far more common than patches that target specific binaries, we estimated the independency score to be very low. Hence, if no mitigation techniques are in place, we set $Cost^R = 0.1$. The

existence of two redundant paths for notification does not affect the overall independency score, as the lowest independency score is associated with one of the proxies of the agent.

Table 5-2: Independency Scores for Samhain

Elem.	Description	Resource used	C_i^R
<i>Base.</i>	root privilege	find unknown exploit	1
P_0	HD IDE	hardware/firmware	5
P_1	fs driver	OS	0.8
D	Samhain	memory/filesystem	0.3
P_2	smtp	memory/filesystem	0.7
P_3	network driver	OS	0.8
P_4	network card	hardware/firmware	5
P_6	syslogd	memory /filesystem	0.7
P_7	CD IDE	hardware/firmware	5
P_8	CD	filesystem	∞
S_0	IDE registers	internal bus	10
S_1	PCI bus	internal bus	10
S_2	kernel IPC	OS system call	0.1
S_3	shared libraries	IPC	0.2
S_4	network stack	OS	0.7
S_5	TCP/IP (int)	network connection	5

The amount of shared resources provide the attacker with various possible attack vectors to use to subvert the HIDS. While mitigation techniques may be used, the wide range of shared elements will probably make them either impractical or insufficient. For example, encrypting and signing configuration files and renaming the executable process will increase the independency score for the detection engine. However, the problem of sharing kernel elements will persist.

5.6 Conclusions

In this chapter we introduced HIDS subversion as a technique to circumvent HIDSs. Two metrics were proposed to evaluate the HIDS strength against subversion. HIDS independency is an attack-independent metric, which provides a measure of the isolation between the HIDS and the supervised system. HIDS resiliency is a quantitative, attack-dependent metric, which factors environmental attributes of the deployment scenario, along with the HIDS independency to estimate the probability that a HIDS will not be subverted as part of the attack to the system supervised. We evaluated the independency score of the Samhain HIDS. We found that the independency score reflects the simplicity of targeting the collection engine, in this case, the kernel. These results will be identical for the Osiris HIDS, as both HIDSs utilize the same mechanisms for collecting and reporting.

Chapter 6: Evaluation of Efficiency and Transparency

In this chapter we discuss efficiency and transparency. These characteristics share specificities that motivate their study in the same chapter. Unlike attack resiliency and visibility, transparency and efficiency have been commonly applied in IDS evaluations. However, a detailed evaluation methodology has not been described for these characteristics. One reason might be the simplicity of performing the measurements: efficiency reflects the time for the HIDS to provide a result; and transparency indicates the performance impact of the HIDS to the system supervised. As an example, researchers will conduct a performance evaluation with and without the HIDS to validate the usability of their proposed HIDS [PKSZ04]. However, current evaluations for these characteristics do not follow a common methodology.

The strong relationship between efficiency and transparency also suggests studying both characteristics in a single chapter. Indeed, in many cases, a decrease of efficiency implies a decrease of transparency, and vice versa. Efficiency heavily depends on the resources utilized by the HIDS, and many of these resources might be shared between the HIDS and the system supervised. If the HIDS utilizes more aggressively the shared resources, the HIDS will likely provide the detection results quickly. Many HIDSs implement the detection engine and the reporting engine in separate systems [GRO3a]. This improves the transparency of the system, but the data collection still needs to be realized by utilizing the resources from the host. For example, if the data collected resides in a hard disk, the slower the data is collected the less impact on the HIDS in the case of hard disk contention.

Both efficiency and transparency may have different impacts on the system supervised depending on the environment. For example, efficiency might not be an issue for a desktop machine in an academic environment, where the system can be reimaged without loss of assets. On the other hand, low efficiency in a high security environment might provide comparable losses than a very late detection. As a result, we need to provide tools to allow the optimization and comparison of these characteristics for different deployment settings. We introduce a mapping between both characteristics and cost metrics. This mapping will be useful to integrate these characteristics into a common framework. HIDS frameworks and the integration of HIDS characteristics will be discussed in detail in Chapter 7.

This chapter is structured as follows. Section 6.1 provides the theory on evaluating efficiency. We introduce a set of time variables that can be studied independently. This section also discusses possible mappings between time delays and the cost of the HIDS. Section 6.2 introduces the theory to measure HIDS transparency. We provide a mapping to translate transparency metrics into costs related to decreased performance of the system supervised. In Section 6.3 we apply these metrics to evaluate empirically the efficiency and transparency of two HIDSs. Finally, in Section 6.4 we apply the data collected in the study performed in Section 4 to estimate empirically the delay related to the time an attack takes to manifest in the files collected.

6.1 Efficiency

6.1.1 Metrics Linked to Efficiency

As already mentioned, in most cases, real time intrusion detection in HIDSs is not possible. It usually suffices to discover the intrusion or attempted intrusion (i.e., attack) in a timely fashion (i.e., before the attacker can perform damage to the system). Efficiency is defined by the time that elapses between the start of an intrusion and the output of an associated alert. To measure efficiency, we measure the delays on every channel and proxy used to report the alert, including the processing time of the detection engine. We then add the delays related to the data collected and the time interval between data collections. Using the scheme represented by Figure 5-1 in Chapter 5, the expected value of the efficiency with n proxies is:

$$E[t_{eff}] = E[t_{interval}] + E[t_{manifestation}] + E[t_{analysis}] + E\left[\sum_{i=0}^{n+2} t_{si} + \sum_{j=0}^{n-1} t_{pj}\right] \quad n \in \mathbb{IN} : n > 1 \quad (6-1)$$

The variable $t_{interval}$ (interval between data collections) represents the time between data collections if the HIDS does not perform continuous supervision. For example, active IDSs [DDW99] will not typically request data continuously to avoid stalling the system supervised, as retrieving data will likely affect the host performance, creating a tradeoff between $t_{interval}$ and the performance of the system supervised as described by transparency. The variable $t_{manifestation}$ (time to manifest) characterizes the time between the start of an attack until the data manifests in the data collected by the HIDS. The variable $t_{analysis}$ (time to analyze) is the time required by the detection engine to analyze the data collected and provide a result. Finally, the variable t_{pj} (time

in proxy) describes the time spent at each proxy in the data path, while t_{si} (time in channel) describes the time in each communication channel.

In practice, not every delay in the data path can be measured. Another approach is to calculate the delay for data collection, analysis and reporting separately. However, in many cases, the estimate will be calculated from data collection to alert reporting. This is especially true for black box systems, where it is difficult to divide the timings for the different elements of the system. Performing a less fine-grained evaluation of efficiency results in losing some information about possible optimizations. For example, a HIDS may be very slow because the chosen reporting engine is slow.

We will call the interval between data collection and alert reporting, the time to process ($t_{process}$):

$$E[t_{process}] = E[t_{collection}] + E[t_{analysis}] + E[t_{report}] = E[t_{analysis}] + E\left[\sum_{i=0}^{n+2} t_{si} + \sum_{j=0}^{n-1} t_{pj}\right] \quad n \in \mathbb{IN} : n > 1$$

(6.2)

The variable $t_{collection}$, represents the delay between the data collection and providing the data to the classifier. $t_{collection}$, contains all communications channels and proxies on that section of the data path. Similarly, the variable t_{report} represents the delays between the output result of the classifier and the final reporting of the alert, and it consists on all communications channels and proxies on that section of the data path.

Figure 6-1 shows all the timings described:

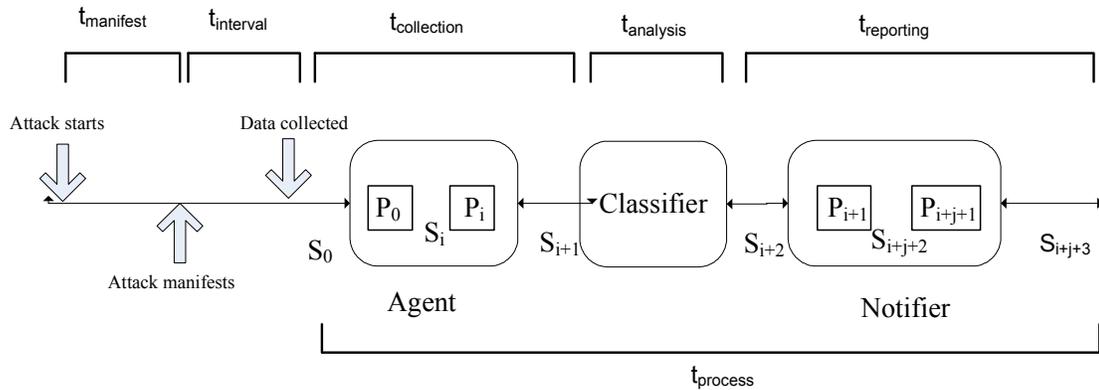


Figure 6-1. Timings Related to Efficiency

The evaluation of efficiency mainly consists of a set of timing measurements in the HIDS itself. We now describe guidelines for the evaluation of the different variables:

- Measuring $t_{interval}$: Many HIDSs schedule checks in a deterministic fashion. As a result, in most cases, $t_{interval}$ will be a fixed value set by the administrator. If the HIDS audits the system at deterministic time intervals, an adversary could evade the HIDS by launching an attack just after an inspection. In [Mol01], randomization between time intervals is proposed to avoid this type of attack.
- Measuring $t_{manifest}$: The time to manifest may be complex to measure, but it is important to note that even if the measurements for the time to process is small, the attacker might have already performed some damage in the system. The reason is that the activity may only manifest in the data collected late in the attack. Time to manifest needs to be measured empirically, by estimating the time for an attack to manifest in the data collected since the attack is launched. A sample study of the time to manifest for filesystem data is

presented in Section 6-4. Note that, in certain instances, the time to manifest may be negative. This happens when the system detects the transition to an insecure state before a security breach actually happens.

Measuring $t_{process}$: We evaluate the delay between the data entering each section of the data path and the output of the required data. In many cases, the estimate will be calculated from data collection to alert reporting. This is especially true for black box systems, where it is not possible to divide the timings for the different HIDS elements. In some cases it will be of interest to divide the time to process into three stages as described in Equation 6-2, or even further as described in Equation 6-1. This separation of elements can provide valuable information for optimizing efficiency. For example, it is common to provide the HIDS reporting capabilities through external programs like sendmail [KS94]. This program may be replaced to provide the same functionality to the HIDS. The time to process varies depending on the amount of data collected. For systems where the data collected is tunable, the evaluator needs to estimate the efficiency for different amounts of data collected, so the trend for the efficiency (e.g., linear with the amount of data, exponential) can be understood.

6.1.2 Translating Time Delays into Costs

A detection delay implies a cost for the supervised system, in the form of further damage and/or due to not applying preventive measures to avoid collateral damage (e.g., canceling stolen credit card data). We introduce the intrusion to alert cost (C_{eff})

that we link to efficiency. C_{eff} is defined as the cost caused by the delay between the beginning of an intrusion and the report of the alert by the HIDS. To calculate this cost, let us suppose an increasing function, $F_{eff}(t)$, mapping delays to costs. For example, the damage created by an intrusion to a server with a database holding credit card records will increase with time. This increase is independent of further intrusions but related to the misuse of the attacker of the stolen data. In general, the function F_{eff} is related to the type of host and the type of attack. The use of mapping functions to costs has been described in [LCT+02], where a mapping function is used to represent the cost to the system due to an attack. In particular, most evaluations provide an implicit translation of HIDS efficiency. A common approximation is to formulate the efficiency function as a step function with three states [LFM+02]. This approximation relies on the fact that for most environments we can divide efficiency measurements into real-time (or early detection), non-real time (late detection) and non-detection (or too late to provide any relief to the intrusion cost). In Figure 6-2 we represent three examples of efficiency functions. Function (1) represents a three costs-step, (2) assumes a cost decreasing exponentially with the detection time, and (3) a cost exponentially increasing with the detection time. Note that all functions are defined only in the time interval between 0 and t_{max} , which is the maximum time for an intrusion to be undetected. Hence, the maximum damage for an undetected intrusion is defined as $C_{max}=F_{eff}(t_{max})$.

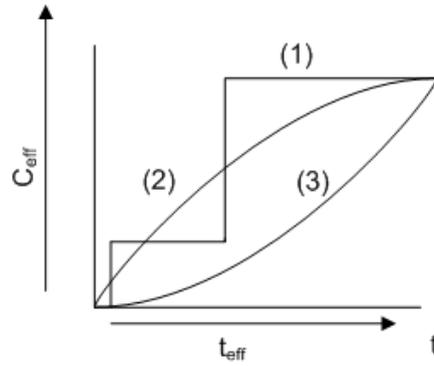


Figure 6-2. Sample Efficiency functions

6.2 Transparency

6.2.1 Metrics Linked to Transparency

As already mentioned, some HIDS elements utilize resources of the system supervised. Typical shared resources include system memory, data buses and storage components. Therefore, the system supervised will likely experience a performance reduction because of the HIDS. HIDS transparency measures this performance reduction.

The performance reduction (P_{tr}) is calculated as the ratio between the normal performance of the system supervised and the performance while the HIDS is running:

$$P_{tr} = \frac{E[\text{Performance with HIDS}]}{E[\text{Performance without HIDS}]} \quad (6-3)$$

Hence, $P_{tr}=0$ indicates no performance reduction (and hence perfect transparency), and $P_{tr}=1$ indicates a 100% reduction of performance (system unusable for the user).

Measurements could be produced by separately evaluating the reduction of performance created by each shared resource and then providing these results along

with the usage of each shared resource by the host. For a HIDS utilizing N shared resources of the system supervised, we have:

$$P_{tr} = \sum_{n=0}^{N-1} Usage_n \times P_{tr}^n \quad (6-4)$$

Where $Usage_n$ refers to the utilization of the shared resource and P_{tr}^n refers to the performance reduction due to that resource. Hence, if the shared resources were identified with exactitude, the best approach to measure transparency would consist of executing a set of benchmarks on the set of shared resources. The appropriate benchmarks would be micro benchmarks that perform a set of short, concise operations on the specific resource, repeating this operation many times. Examples of micro benchmarks are Bonnie++ [Bon] and IOzone [IoZ]. Micro benchmarks would provide a good understanding of the different aspects of the shared resource, but would not provide information on real usage of this resource by an application. Hence, the evaluator needs to estimate $Usage_n$ for each shared resource. Furthermore, to identify, select and create micro benchmarks for each shared resource is not a feasible approach for most HIDSs evaluations. For example, in [Mol01] even if the whole HIDS was implemented as an external PCI card, the host still displayed a noticeable overhead due to contention on the PCI bus, which shows that micro benchmarks might need to be very specific. In this case, providing a micro benchmark for the PCI bus usage and estimating the usage for the PCI bus would have been very difficult.

In most practical cases, the performance reduction can be measured by running a set of high-level performance benchmarks (covering different resources) on the system

with the HIDS activated and then with the HIDS deactivated. Macro benchmarks can be used for such task. Macro benchmarks perform application driven actions for long durations. The actions of the macro benchmark heavily utilize a set of specific system resources. Examples of macro benchmarks are Postmark [Pos] and httperf [Htt]. Macro benchmarks decide the resources to be evaluated, and the Usage_n for each resource. For that reason, macro benchmarks should be picked carefully and the configuration of the benchmark provided along with the evaluation for transparency. Indeed, as macro benchmarks utilize a broader set of shared resources, a modification of the configuration of the macro benchmark may result in heavily utilizing a shared resource instead of another. Hence, an evaluator could produce incorrect results by configuring the macro benchmark with a bias for using more or less heavily a specific resource. We will describe next the commonly used shared resources by each HIDS element. The collection agent will always interfere with the resources containing the data since the collection agent needs to reside on the HIDS. For example, capturing filesystem data will interfere with filesystem related resources (e.g., IDE controlled, PCI bus), and capturing system calls will produce a performance reduction while utilizing shared resources related to the operating system. The detection engine will likely affect the processor and system memory. The shared resources for alert reporting vary greatly, but usually the impact for transparency is minor, as the amount of data transferred is low.

6.2.2 Relationship with Cost

The impact of the performance reduction created by the HIDS on the system supervised varies depending on environmental variables. The performance reduction supposes a decrease of productivity, and hence translates into a cost for the system supervised (C_{tr}). To perform the translation between transparency and cost, a mapping function needs to be provided. The mapping function describes the cost depending on the performance degradation, and will depend of the type of system supervised.

Let us denote the mapping function as $F_{tr}(P_{tr})$. The function is defined in the interval $[0,1]$, and for $(P_{tr}=1)$ provides the cost of a complete shutdown of the system supervised. Hence:

$$C_{max} = C_{tr} = F_{tr}(1) \quad (6-5)$$

For example, a function may suppose that the productivity reduction is proportional to the reduction of performance, i.e., a 50% reduction of productivity translates into a 50% of the cost of a shutdown. For this simple case:

$$C_{tr} = C_{max} P_{tr} \quad (6-6)$$

Due to the high cost of transparency, many HIDSs are only executed for a small period of time. If the HIDS does not run continuously, we will need to provide the proportional cost of running the HIDS for only a short period of time. Hence:

$$C_{tr} = \frac{ExecutionTime}{TimePeriod} F_{tr}(P_{tr}) \quad (6-7)$$

Notice that we assumed that the mapping function is independent of time. This might not be true as the cost due to transparency may be much less if the HIDS is executed

at night. Hence, the final cost might have to be corrected in some cases to account for the time when the HIDS is executed.

6.3 Case Study: Osiris and Samhain

In this section we study the transparency and efficiency of two HIDSs, Osiris [Osi] and Samhain [Sam]. Samhain has already been described in Chapter 5. Osiris' operation and architecture are similar to Samhain [Wot05]. Both feature a client/server architecture, and both examine files to detect integrity variations. However, the collection strategy and the data processing differ greatly. While both are open source projects, we treated both HIDSs as black boxes, only interacting with them through the provided API, usage commands and log files.

6.3.1 Experimental Setup

Both Osiris and Samhain were deployed in the same test environment, a Debian Linux Pentium 4, 1.4 Ghz featuring 256 MB of RAM. Both client and server of the integrity verifiers were deployed in the system supervised. The version of the evaluation copy of Samhain was 4.2.3. The version of the evaluation copy of Osiris was 2.3.5. Both versions were the latest stable releases available at the time of the experiment. We configured the HIDSs to examine exactly the same files for each test, and to verify the exact same integrity settings for each file.

While evaluating efficiency, we tuned both configuration files to evaluate the same exact number of files. These files were created using automated scripts, and each file contained random content with fixed sizes for every file.

To evaluate transparency, we utilized the filesystem macro benchmark Postmark [Pos], version 1.5.1. Postmark provides the performance of a set of filesystem transactions, emulating the behavior of a mail server under heavy load. Specifically, the workload is meant to model a combination of web-commerce transactions. To achieve this, PostMark creates a set of files of random sizes. The files are then subjected to a number of transactions. These transactions consist of a file creation or deletion paired with a file read or append. Each pair of transactions is chosen randomly. Postmarks is a macro benchmark which mimics real usage of the system and heavily utilizes the shared resources related to the data collected by the HIDSs under evaluation (i.e., filesystem).

6.3.2 Results

First we evaluated efficiency for both Osiris and Samhain. Only the programs X server and KDE were running during the evaluation. For both systems the server was started first (called Osiris and Yule, respectively). A baseline for integrity comparison was then created. Before executing the client, a set of 5 files was modified randomly to create alerts. Then the client was launched, indicating the start time for efficiency measurement. We used the logs with the execution times of the process to measure efficiency. We compared the logs created by both applications with the logs related to process execution time on the system supervised to make sure that they were consistent. For Osiris, the logs provided a separation between collection and processing time. For Samhain, the logs did not provide such information but provided results from data collection to alert reporting. For both systems, we studied three

different file sizes: 142 bytes, 13012 bytes and 39012 bytes. For each file size we incremented the number of files to verify, from 1000 to 6000 in 1000 file increments.

Table 6-1: Efficiency for Osiris with File Size of 142 Bytes

Number of files	Average (s)	Std. Dev. (s)	Interval (s)	
1000	4.67	0.58	4.09	5.24
2000	5.33	0.58	4.76	5.91
3000	5.67	0.58	5.09	6.24
4000	6.00	0.00	6.00	6.00
5000	7.00	0.00	7.00	7.00
6000	8.67	1.53	7.14	10.19

Table 6-2: Efficiency for Osiris with File Size of 13012 Bytes

Number of files	Average (s)	Std. Dev. (s)	Interval (s)	
1000	4.67	0.58	4.09	5.24
2000	6.33	0.58	5.76	6.91
3000	8.00	0.00	8.00	8.00
4000	10.00	1.00	9.00	11.00
5000	29.67	1.53	28.14	31.19
6000	58.67	2.08	56.59	60.75

Table 6-3: Efficiency for Osiris with File Size of 39012 Bytes

Number of files	Average(s)	Std. Dev. (s)	Interval (s)	
1000	19.33	1.15	18.18	20.49
2000	33.33	2.08	31.25	35.41
3000	39.00	2.00	37.00	41.00
4000	46.00	2.65	43.35	48.65
5000	76.00	1.73	74.27	77.73
6000	93.00	7.55	85.45	100.55

Tables 6-1,6-2 and 6-3 provide the results of efficiency evaluation for Osiris. The results in Osiris for efficiency heavily depend on the file size, or in other words, on the amount of data collected. For the case of 142 bytes files, Osiris almost does not account for collection time but only for processing time. This can be verified as the results, while increasing, are non linear with the number of files. We have a similar result for file sizes of 13012 bytes, with some non-linearity for the first set of

measurements. For file sizes of 39012 bytes, the results are almost linear with the number of files collected. This may be explained as the bigger files render the processing time negligible and the collection time becomes then the only factor driving efficiency.

Table 6-4: Efficiency for Samhain with File Size of 142 Bytes

Number of files	Average (s)	Std. Dev. (s)	Interval (s)	
1000	222.33	8.14	214.19	230.48
2000	397.67	3.79	393.88	401.45
3000	507.00	23.64	483.36	530.64
4000	621.00	74.72	546.28	695.72
5000	811.00	51.45	759.55	862.45
6000	1029.33	1.53	1027.81	1030.86

Table 6-5: Efficiency for Samhain with File Size of 13012 Bytes

Number of files	Average (s)	Std. Dev. (s)	Interval (s)	
1000	234.00	1.00	233.00	235.00
2000	404.00	1.00	403.00	405.00
3000	539.67	3.21	536.45	542.88
4000	677.33	56.62	620.72	733.95
5000	819.00	28.51	790.49	847.51
6000	1030.33	36.96	993.37	1067.30

Table 6-6: Efficiency for Samhain with File Size of 39012 Bytes

Number of files	Average (s)	Std. Dev. (s)	Interval (s)	
1000	233.33	1.15	232.18	234.49
2000	393.33	23.67	369.66	417.00
3000	544.33	4.04	540.29	548.37
4000	740.67	29.77	710.90	770.44
5000	858.00	14.11	843.89	872.11
6000	1037.00	12.49	1024.51	1049.49

Tables 6-4, 6-5 and 6-6 provide the results of evaluating efficiency for Samhain. The results for Samhain were surprising compared to the ones for Osiris. First, almost all the measurements reflect a 10-fold increase of time delays with compared to Osiris.

We believe that the time difference is because of a common overhead created by

Samhain. A possible reason for this overhead is that Samhain contains more features than Osiris and the databases take longer to load and compare. This hypothesis is supported by the data: unlike Osiris, the size of files does not affect the results, and they remain constant across file sizes. However, the efficiency of Samhain is almost linear with the number of files, independently of the size. This reflects that the collection time is negligible compared with the time to compare.

Next, we calculated the transparency of the system. We performed two tests: one with Postmark utilizing a large number (500) of medium size files (7 MB) and other utilizing large files (62 MB) but a reduced number (50). The reason for these two tests was to evaluate the effect of accessing the hard disk in different ways. A larger number of files will create more activity, but possibly fewer accesses to the hard disk due to caching.

Table 6-7: Transparency Evaluation with Postmark for 500/7MB Files

Type of Access	Average(MB/sec)	Std. Dev. (s)	Interval (s)		P _{tr} (%)
Read no HIDS	10.03	0.19	9.84	10.22	1
Write no HIIDS	13.08	0.24	12.84	13.31	1
Read Osiris	4.20	0.45	3.75	4.66	0.419111
Write Osiris	5.48	0.59	4.89	6.07	0.419003
Read Samhain	5.83	0.35	5.48	6.18	0.581305
Write Samhain	6.81	0.43	6.38	7.23	0.52052

Table 6-8: Transparency Evaluation with Postmark for a 50/62MB Files

Type of Access	Average(MB/sec)	Std. Dev. (s)	Interval (s)		Pe _{ff} (%)
Read no HIDS	12.89	0.24	12.66	13.13	1
Write no HIIDS	15.64	0.29	15.35	15.93	1
Read Osiris	4.98	0.33	4.65	5.30	0.386028
Write Osiris	5.90	0.56	5.34	6.47	0.377499
Read Samhain	10.25	0.45	9.80	10.70	0.795067
Write Samhain	12.26	0.70	11.55	12.96	0.783775

Perhaps a surprising result from the evaluation is that writing provides better performance numbers than reading. These results are due to the caching strategy of current filesystems, where writes are aggressively cached until flushed. Macro benchmarks do not force filesystem flush, as the performance is evaluated from an application perspective. In fact, Postmark will provide similar write/read numbers for most modern filesystems. Osiris transparency is low for both reading and writing (0.42) of smaller files. The numbers are consistent with our experience, as Osiris aggressively collects files. For larger files, Osiris' results degrade, as the hard disk contention increases (0.39 for read, 0.38 for writes). Samhain, on the other hand, deals much better for both small and large files. For small files, the larger number of transactions affects its transparency (0.58 for read, 0.52 for writes). We can see that writes in this case are impacted more evidently. This is consistent with previous results of efficiency, where it looks like most of the time the HIDS is not collecting but processing the information. For the case of larger files, the results show much better results for transparency. As the amount of activity decreases, the HIDS is given more time to process the database.

An important problem is to decide which should be the results provided by the evaluator. As we have seen, a benchmark, with a different configuration, may produce different results for transparency. The solution is to configure the benchmark so that it closely mimics the specific setting of the system while using the shared resources utilized by the HIDS. In this case, we evaluated the transparency of the HIDS for different settings, to provide a broader understanding of transparency results for various settings of the system supervised.

6.4 Evaluating Efficiency Related to the Data Collected

A commonly asked question is to measure how quickly an attack can be detected when monitoring a specific file. Previously we discussed the necessity of an empirical evaluation of the time to manifest ($t_{manifest}$). In this section, we provide numerical results of the time to manifest for files on a filesystem. We measured the timing for the attacker to perform some activity on each file utilized as part of attacks. The experimental setup is the same as described in Chapter 4, Section 2. We created a set of scripts which calculated the time difference between the start of each attack to the first access to the every file which presented activity as part of the attack. In Figure 6-3, we show a reduced set of files displayed using a state machine. Each connector in the state machine refers to a transition from one file to another, and the average time from the beginning of an attack to perform actions on that file. We observed that most attackers modified the password file after checking the system first, which takes on average one minute. Malware is often installed later in the session, consuming an average of 76 seconds if the malware installation occurs after the BEGIN state, increasing the time to an average of 139 seconds for malware written immediately after modifying the password. Installing malware took the longest time, and if it is the last action of the attacker, finalizing this activity will occur on average after 300 seconds have elapsed. An interesting result is how fast an attacker can modify the password file after the beginning of the session, taking only 20 seconds. As a conclusion, this study shows the importance of the time to detect on evaluations. For example, a filesystem HIDSs based on detecting malware will detect the attack later in the session, and most likely configuration files, including the password file will

have been modified by the attacker. On the other hand, if the data collected included the password file, the attack will more likely be in an early stage, and the damage to the system less significant.

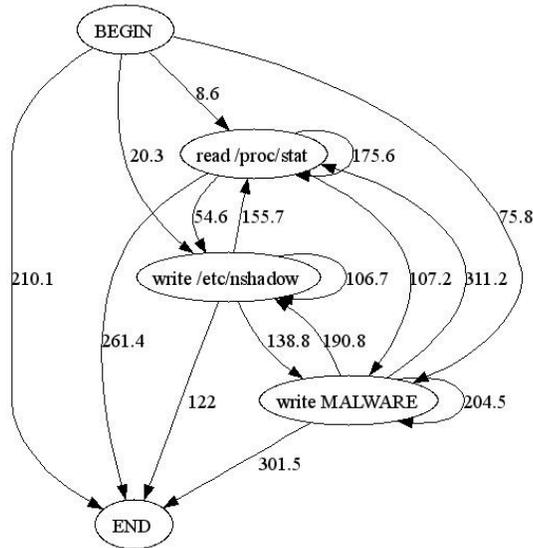


Figure 6-3. Average Time to Manifest for Sample Files (in seconds)

6.5 Conclusions

In this chapter, we introduced two characteristics. Efficiency evaluates the delay from the start of an attack until the HIDS reports the attack. Transparency evaluates the HIDS impact on the performance of the system supervised. For both characteristics, we proposed metrics and techniques to estimate these metrics. Furthermore, we showed how to transform the metrics into costs. These transformations are necessary to account for deployment factors and to integrate the characteristics into HIDS frameworks. Finally, we performed two experiments. In the first one, we evaluated transparency and efficiency for two HIDSs, and compared the results. In the second one, we evaluated empirically the time to attack manifest for filesystem data.

Chapter 7: Integrating the Characteristics into a HIDS Cost Framework

Comparing the characteristics one-to-one for various HIDSs may be of limited value. A HIDS can be superior when considering one characteristic, but inferior for another one. In addition, some characteristics may be related, so one characteristic increasing may result in another one decreasing. Hence, it is important to create frameworks, which provide the capability to compare several characteristics at the same time. In this chapter we will first describe some proposed frameworks for HIDSs. Then we will integrate the characteristics proposed in previous chapters into a cost framework. We will then show two examples on how to evaluate a set of characteristics utilizing the proposed framework. In the first example, we will integrate attack resiliency and detection accuracy to show the impact of different independency scores for HIDS optimization. In the second example, we will present a tool to create a composite evaluation of visibility and accuracy, called a HIDS stimulator.

7.1 Survey of HIDS Frameworks

The goal of a successful HIDS framework is to make possible the comparison between HIDSs, or between operational points of the same HIDS. A HIDS may have different operational points if it provides configurable settings. For example, the classifier may label a set of data as an attack if, after performing some statistical calculations on the data, a threshold in the classifier algorithm is reached. If the

threshold is configurable, the results of accuracy evaluations for the system will differ for each possible setting of the threshold.

If the evaluator is provided with a set of metrics for two HIDSs, and all results are higher for one of the two HIDSs, then the evaluator does not have a decision problem. However, a problem exists if for one metric a HIDS presents better results in the comparison, but for another metric the HIDS has worse results. A solution to this problem is to create a composite result, adding external factors that may help in the decision making process.

All techniques for optimization and comparison so far integrate only the metrics for detection accuracy. As discussed in Chapter 3, the two main accuracy metrics are the number of true positives in a certain universe of attacks, referred to as the probability of detection (P_D) and the number of false positives in a certain universe of valid actions collected by the HIDS, referred to as the probability of false alarms (P_{FA}). As two metrics are provided, a framework is necessary to integrate both into a single, composite metric.

A first attempt to create composite metrics was proposed by Axelsson [Axe00b]. In this work, the attack ratio (p_a) is presented as an important factor while evaluating IDSs for different environments. The positive predictive value (PPV) and negative predictive values (NPV) (as shown in Chapter 4) are proposed as possible composite metrics to integrate p_a , P_{FA} and P_D .

In [JU01], a set of external factors were proposed to both provide a comparison framework and to find the optimal operational point of an IDS. These external factors

were the cost of errors: the cost of not detecting an attack (C_{FN}) and the cost of erroneously labeling normal data as an intrusion (C_{FP}). Additionally, as proposed by Axelsson [Axe00b], the attack ratio (p_a) was also included in the framework. By using these external factors, the overall cost of implementing a HIDS was proposed as:

$$Cost_{IDS} = p_a C_{FN} (1 - P_D) + (1 - p_a) C_{FP} P_{FA} \quad (7-1)$$

The equation can be described as follows. A cost is associated with the event of missing an intrusion. The probability of missing an intrusion is the probability that an attack occurs p_a , multiplied by the probability that the attack is not detected ($1 - P_D$). There is an additional cost associated with an alert raised in a secure state. The probability of such an event is the probability that an attack did not happen ($1 - p_a$), multiplied by the probability that a false alarm was raised (P_{FA}).

Once the overall cost is calculated, given a set of pairs composed by the probability of detection (P_D) and probability of false alarms (P_{FA}), the evaluator calculates the total operational costs for each pair, and decides depending on the minimum cost value:

$$(P_D, P_{FA}) = \min_{P_D, P_{FA} \in D} (Cost_{IDS}) \quad (7-2)$$

where D is the set of (P_D, P_{FA}) pairs for evaluation. The subset D is called the receiver operating characteristics (ROC) [GSS99] of the HIDS if the pairs come from different operational points of the same HIDS. ROCs are commonly represented by curves. The curves are created by plotting each pair in a graph. The X-axis of the graph represents P_D , and the Y-axis corresponds to P_{FA} . If the HIDS only presents one operational point, the representation will be a single dot in the graph. Given a ROC, if

our goal is to optimize the operation of the HIDS, then we resolve the following equation:

$$(P_D, P_{FA}) = \min_{P_D, P_{FA} \in ROC} (Cost_{IDS}) \quad (7-3)$$

The specific calculation of the values for the different costs associated with the IDS (C_{FN} , C_{FP}) was studied in [LFM+02]. C_{FN} was defined as the maximum cost incurred by the intrusion plus the cost of raising an alert. C_{FP} was set to the cost of raising an alert and the cost of clearing the false alarm.

In [GFD+06a], a different approach is taken, by calculating the relationship between the input and the output of an IDS using information theory. A new metric is introduced, the intrusion detection capability, described as the relation between the mutual information between the input and the output and the entropy of the input. The approach used to create this relationship is similar to the one utilized in Chapter 4 to quantify data quality. The main reason to introduce this metric is to avoid the problem of guessing the different operational costs associated with each detection result by the classifier.

However, in [CBS06], the author demonstrates that previous frameworks can be represented in terms of a cost framework, given that additional costs are included for each possible outcome. These costs define the following outcomes: reacting to an intrusion, missing an intrusion, raising a false alarm and not raising an alarm in a secure state, respectively. In Table 7-1, we summarize the meaning of the different costs. The following equation represent the extended framework including the four possible situations:

$$Cost_{IDS} = p_a(C_{FN}(1-P_D) + C_{TP}P_D) + (1-p_a)(C_{FP}P_{FA} + C_{TN}(1-P_{FA})) \quad (7-4)$$

Table 7-1: Cost Related to HIDS Evaluation

Cost	Description	Associated Probability
C_{TN}	Normal HIDS operation	$(1-p_a)(1-P_{FA})$
C_{FN}	Cost of not reacting to an intrusion	$p_a(1-P_D)$
C_{FP}	Cost of a false alarm	$(1-p_a)P_{FA}$
C_{TP}	Cost of reacting to an intrusion	p_aP_D

A limitation of all approaches based on cost, as pointed out in [GFD+06b], is that the evaluator sets the costs arbitrarily. Hence, there is no standard manner to calculate the costs except using expert knowledge. In the next section we address the problem by linking operational costs with transparency and efficiency.

7.2 A Cost Framework for HIDSs

In this section we introduce a modified framework to estimate the operational cost of the HIDS. The starting point is the cost framework as described in [JU01, CBS06], represented in Equation 7-4. Our objective is to introduce the characteristics described in the previous chapters, and utilize the metrics proposed to complement detection accuracy in this framework.

Our motivation to modify the estimation of the operational cost follows two observations. First, for HIDSs the probability of true positives (TP) is not only dependent on detection accuracy, as represented by the P_D , but also on visibility and attack resiliency. Second, the costs due efficiency and the cost due to transparency can be utilized to estimate the operational costs in the framework.

7.2.1 Integrating Attack Resiliency and Visibility

An attack not detected by the HIDS (false negative) may fall in one of these three possibilities: 1) not manifested in the inspected data (the attack was not visible by the HIDS), 2) manifested but not detected by the HIDS (the attack evaded the HIDS) and 3) manifested and detected, but the HIDS was corrupted to prevent the alert (HIDS subversion). In Chapter 5 we defined the probability of attack manifestation to quantify the probability that an attack manifests in the data, and, in Chapter 6, attack resiliency as the probability that an attack will not be subverted in the course of an attack towards the supervised system.

We provide a graphical representation of the associated detection tree in Figure 7-1. In the proposed detection tree, P_D is complemented with the other two proposed metrics (P_M and P_{res}) to provide a more accurate quantification of TP and FN .

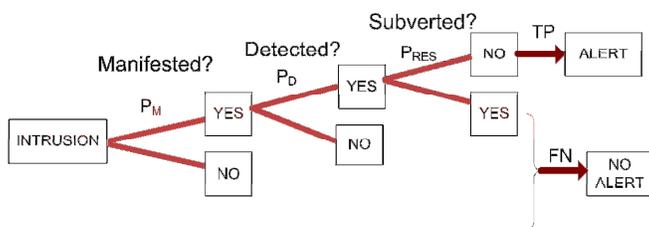


Figure 7-1. Detection Tree

Note that P_M , P_D and P_{res} are not independent. The probability of detection is conditional to the manifestation of an attack. The probability of attack resiliency is conditional to the event that the attack manifested in the data collected and that the attack was detected. Hence:

$$\begin{aligned}
TP &= P_M P_D P_{res} \\
FN &= 1 - P_M P_D P_{res}
\end{aligned}
\tag{7.5}$$

7.2.2 Integrating Efficiency and Transparency

The operational cost described in [JU01, CBS06] was chosen by guessing reasonable cost values. However, some of these costs are directly related to the system efficiency and transparency. In this section we study the relationship between operational costs, efficiency and transparency.

The performance overhead created by the HIDS in the system supervised modifies the operational cost when both the system supervised and the HIDS operate normally. In other words, when no attacks exist, and no alert is reported, the only cost that applies to the operational cost is the transparency cost. As we have seen previously, in Section 7-1, the cost of the HIDS while in normal operation was referred as C_{TN} . If we suppose that no other operational cost applies, we can set the normal operation cost of the HIDS as the transparency cost. Hence:

$$C_{TN} = C_{tr} \tag{7-6}$$

In Chapter 6 we pointed out that the cost for true positives (C_{TP}) depends on the system efficiency, due to the damage that an ongoing intrusion may have caused.

Hence, we set C_{TP} to the cost caused by the delay on detecting an intrusion:

$$C_{TP} = C_{eff} = F_{eff}(t_{eff}) \tag{7-7}$$

Moreover, we set cost for missed intrusions to the maximum cost of the function that maps efficiency to costs (F_{eff}). As the function is always increasing (i.e., taking

longer to detect an intrusion will only result in additional costs), we set the cost of missed intrusions to the limit of the efficiency function F_{eff} :

$$C_{FN} = C_{eff}^{max} = \lim_{t \rightarrow \infty} F_{eff}(t) \quad (7-8)$$

7.2.3 Operational Cost for HIDSs

We can now replace the previously defined costs in Table 7-2 by the costs related to both efficiency and transparency. Hence:

$$Cost_{IDS} = p_a(C_{eff}^{max}(FN) + C_{eff}TP) + (1 - p_a)(C_{FP}FP + C_{tr}(1 - TP)) \quad (7-9)$$

By utilizing the new characteristics, the final operational cost provides more details than before for both comparison and optimization.

A remaining cost, which still needs to be estimated and is not related to characteristics of the HIDS is the cost of a false positive. If we substitute Equation 7-9 with the results of integrating attack resiliency and visibility, we have the following result:

$$Cost_{IDS} = p_a((P_M P_D P_{res}(C_{eff})) + (1 - P_M P_D P_{res})(C_{eff}^{max})) + (1 - p_a)(C_{FP}P_{FA} + C_{tr}(1 - P_{FA})) \quad (7-10)$$

This result integrates the proposed characteristics into a single cost framework. While providing improvements compared to previous cost frameworks, this framework still requires a considerable amount of expert knowledge input. Next we will describe two case studies on how to utilize these results in practice.

7.3 Estimating the Cost of a HIDS

We have estimated the different characteristics for two HIDSs, Samhain and Osiris. However, calculating the total cost of these two HIDSs applying equation 7-10 will be of limited value. First, for both systems the independency score is very low, providing similar final costs. Second, to avoid the problems related to the low transparency, both Samhain and Osiris are usually deployed with long interval (t_{interval}) between executions. This basically dilutes the results for efficiency, as the interval between executions is comparatively much more important than the other terms related to efficiency. Also, transparency is affected, as the total executing time is short, increasing the transparency of the system. A more interesting use for the cost framework is to show the effect of improving a characteristic in the HIDS. Next, we show how the operational point of Osiris and Samhain will be affected by modifying their independency.

A key problem in HIDS evaluation is, given the ROC of a HIDS, to find the optimal operational point given a certain environment, $Cost_{IDS}^* = \min_{(P_D, P_{FA}) \in ROC} Cost_{IDS}$. This section illustrates how different values of $Cost^{priv}$, as described in Chapter 5 will modify the value of $Cost_{IDS}$, where $Cost_{IDS}$ is calculated by using equation 7-10. For demonstration purposes, we will use ROC curves and attack probability values from the DARPA 1998 data set [LHF+00]. In Figure7-2, we can see the ROC curve we will use for our analysis. We will suppose that the probability of manifestation is equal to one, and does not affect the evaluation.

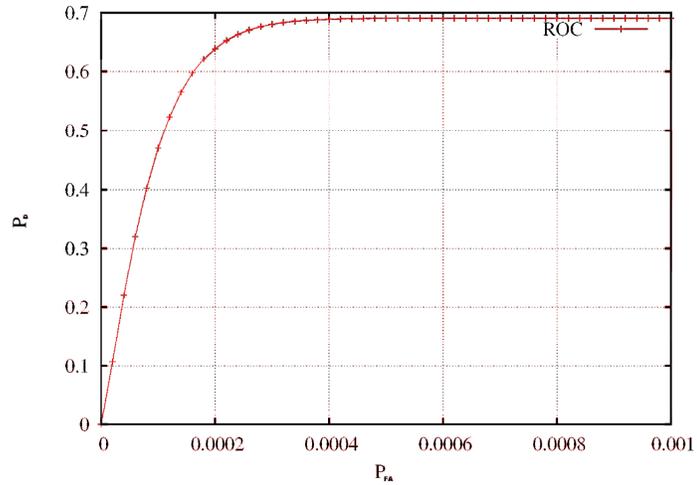


Figure 7-2. Sample ROC

Given that the mapping functions for efficiency (F_{eff}) and transparency (F_{tr}) are as shown in Figure 7-3 and Figure 7-4 respectively, we use the results of the Samhain HIDS to calculate the related costs.

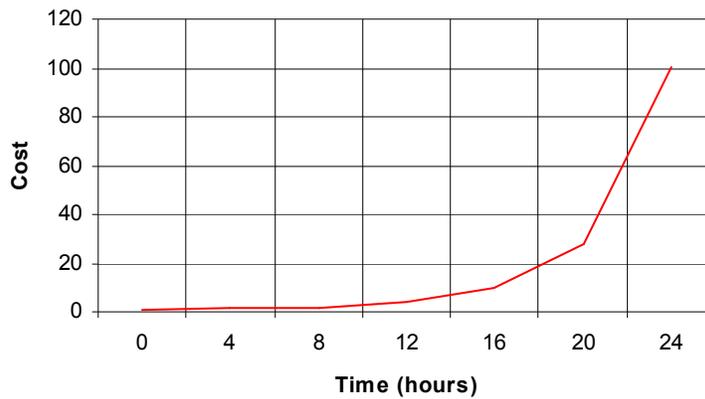


Figure 7-3. Mapping Function for Efficiency

We suppose that for our deployment scenario the mapping function for efficiency corresponds to an exponential, while the mapping function for transparency is linear with the performance decrease caused by the HIDS. The cost for an intrusion

increases rapidly once the attacker has enough time to install malware and/or steal data from the system, hence our choice for the sample mapping function. A linear function represents well the behavior of a web server for cost due transparency, as reduced transparency will impact the number of connections lost, and hence lost revenue.

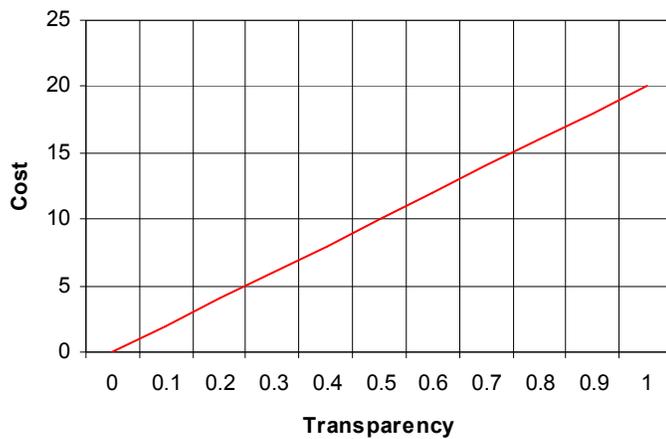


Figure 7-4. Mapping Function for Transparency

We now evaluate the results corresponding to Samhain using these results. The C_{eff}^{max} corresponds to $F_{tr}(1)$, while C_{eff} corresponds to the efficiency of the system. In this case, we suppose that the HIDS is executed every six hours, and then add the values of efficiency as shown in Section 6. As we do not execute continuously the HIDS, the transparency is weighted for the time executed. The efficiency and transparency operational costs and the cost wrongly reacting to an intrusion are as described in Table 7-2:

Table 7-2. Operational Costs

C_{tr}	3×10^{-5}
C_{eff}	2
C_{eff}^{max}	100
C_{FP}	5

We set the value of the probability of intrusion, p_a , $p_a = 6.25 \times 10^{-5}$ as published in the DARPA evaluation. We also suppose that all attackers care about being detected and will attempt to subvert the IDS, with $\theta = 1$ as described in Chapter 5. In the study of attack resiliency, for a given privilege level, we described how the probability of a subversion attack succeeding, $P_{success}^{priv}$, was linked to the relative cost compared to the baseline of finding an unknown vulnerability in the software running on the system supervised (independency score). We denoted this cost, for a given privilege level, as, $Cost^{priv}$. In Chapter 5 we also provided guidelines to estimate the probability of a successful subversion ($P_{success}^{priv}$) by using the independency score. We introduce the following simple mapping function as an estimate of $P_{success}^{priv}$ that has the properties described in Chapter 5, Section 4:

$$P_{success}^{priv} = \frac{1}{1 + Cost^{priv}} \quad (7-11)$$

For the initial evaluation, let us set $P_{res} = 1$. Applying Equation 7-11 to the HIDS, substituting, and resolving for each point in the ROC curve from Figure 7-2, we find that the optimal operation point of this HIDS is $P_D = 0.6534, P_{FA} = 0.00022$, with an expected cost of $Cost_{IDS} = 0.00335$.

Our goal is to optimize the operational point given different values of $Cost^R$. For this study we suppose the frequency of attacks in privilege levels other than root is negligible. The optimal operational point changes as presented in Figure 7-5.

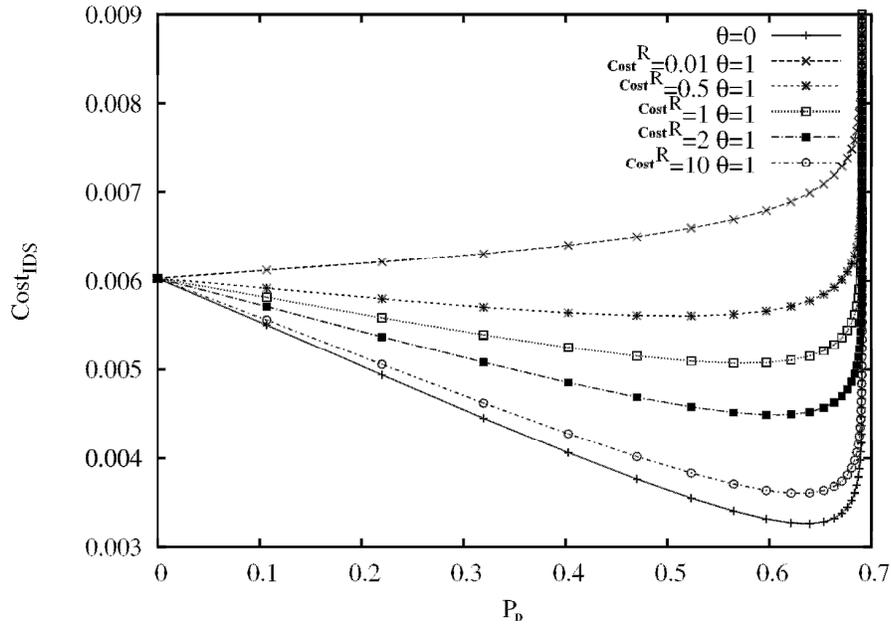


Figure 7-5. $Cost_{IDS}$ for Different Independency Scores

The Operational Costs are: $C_{tr} = 3 \times 10^{-5}$, $C_{FP} = 5$, $C_{eff} = 2$, $C_{eff}^{max} = 100$

Table 7-3 provides the optimal probability of detection and the associated cost for the different values of $Cost^R$.

Table 7-3: P_D and $Cost_{IDS}$ for Different Values of $Cost^R$

$Cost^R$	P_D	$Cost_{IDS}$
0.01	0	0.0061
0.5	0.5234	0.0056
1	0.5649	0.0051
2	0.6213	0.0045
10	0.6397	0.0036

The HIDS is unusable for $Cost^R = 0.01$: the low independency provides no cost benefit compared with not having a HIDS, for any operation point of the provided ROC curve. This fact demonstrates the importance of the study of independency, indicating that a HIDS with a good detection engine will only benefit the organization if it is not easily subverted. For the other HIDSs a different optimal operational point should be used to compensate for the reduction in the probability of detection and the increased comparative weight of the probability of false alarms. This solution is of course dependent on the operational costs as provided by transparency and efficiency, but the importance of resiliency should be stressed as we have chosen a very low p_a . Increasing the value of p_a will provide even more extreme results, as the increased number of attacks will provoke a spike in the number of subversion attempts. In this case we assumed the worst-case scenario, with $\theta = 1$. However, we believe it makes sense for HIDSs to evaluate the operational point on the pessimistic side.

7.4 Linking Detection Accuracy and Visibility

As already mentioned, the rate of true positive depends on the probability of attack manifestation in the data collected and on the probability of detection of the classifier. So far, in practical evaluations, this distinction has not been implemented. In this section, we discuss an evaluation methodology, which links visibility and detection accuracy.

Our approach relies on a new tool called a HIDS stimulator. To create the HIDS stimulator, sample attack evidences were coded and organized depending on the data where they manifested. This set of data can be linked to its probability of

manifestation (i.e., probability that an attack manifests in that set of data). Then each attack manifesting in the data collected by the HIDS is launched to provide the probability of detection for the HIDS for these attacks.

7.4.1 Evaluation Methodology

As mentioned, the metrics used to estimate detection accuracy are the probability of detection and the probability of false alarms. In practice, the probability of detection is estimated as the percentage of detected malicious activity, while the probability of false alarms is estimated by the percentage of wrongfully raised alarms. As shown previously, the probability of detection is not equal to the probability of true positives, as visibility also needs to be considered. A detection engine may have a very high probability of detection, but the total number of attacks detected can be small if the attacks rarely manifest in the data collected.

The proposed methodology consists on the following steps:

- Create a set of attack evidences
- Pair each evidence with the data where it manifests
- Provide the probability of manifestation for the different data sets where the evidence manifests
- Evaluate the HIDS detection accuracy only with evidences that manifest in the data acquired by the HIDS (visible evidences)

Finally, we evaluate all the visible evidences to calculate the probability of true positives:

$$TP = \sum_{i=1}^n P_M^{e_i} P_D^{e_i} \quad (7-12)$$

7.4.2 Implementing a HIDS Stimulator

Many tools exist to evaluate NIDSs [AAL+03, Met]. These tools launch remote exploits over unencrypted network traffic or scan for remote vulnerabilities. However, there are no such equivalent tools for HIDSs. This is why we decided to develop a tool called, HIDS stimulator. The HIDS stimulator applies the evaluation methodology described in the previous section to provide a more accurate estimation of the probability of true positives.

The HIDS stimulator runs on the machine that has the HIDS installed. It should be run by a user who has root privileges, thus emulating the capabilities a third party who gained root access would have. The HIDS stimulator will not generate any external network traffic, separating its tests from those typical of NIDS testers. Therefore, system administrators can gauge the effectiveness of their HIDS, as well as enhance their understanding of what is detected by their HIDS.

One must use a combination of the log produced by the HIDS stimulator and the log produced by the HIDS under test to determine which tests are detected. Due to the nature of some HIDSs the user may have to wait some time before any detections are reported. Therefore, one must have a good understanding of the reporting functionality of the HIDS they are testing in order to evaluate it.

The first implementation of the HIDS stimulator was a command line tool created in Java for portability. The stimulator launched a set of attack evidences, in the form of modular plugins. The modular nature of the HIDS stimulator was based on the

directory structure presented by its plugins directory. Any subdirectories within the directory constitute a set of tests sharing similar data manifestation (e.g., filesystem, system calls). Inside a given test type directory the attack evidence exists, as well as two subdirectories: testdata and testlog. For a given name of a test, *testname*, three files corresponding to that test may exist. The first was *info-testname.txt*, which consisted on a text file containing information of the test. The second was *run-testname*, which was an executable that runs the desired evidence. The third and final file that could be associated with an evidence was *clean-testname*, which performed the task of undoing any changes created by the *run-testname* executable. Attack evidence types could be added simply by creating subdirectories within the tests directory, and tests per type could be added simply by creating the required files. The tool will automatically pick up these additions. A graphical view of the directory structure can be seen in Figure 7-6.

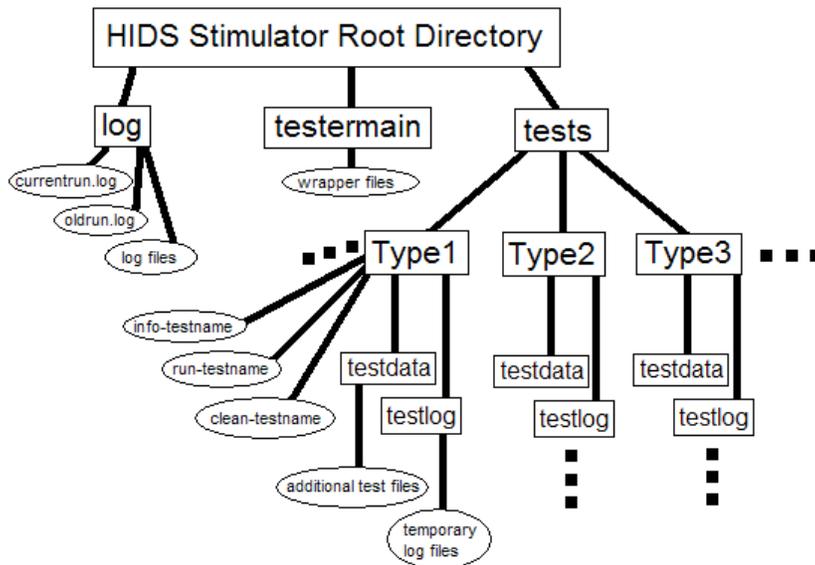


Figure 7-6. Command Line Stimulator

While adding more test evidences, we realized that the command line version was insufficient for several reasons. Attack evidences may manifest in more than one type of data, so we needed to support this feature without copying each evidence over different directories. The command line approach also failed to provide searching capabilities to select different type of evidences. The information files, while useful, were not standardized. This created problems when the evidences were created by different people. Finally, there was no way to filter the attack evidences by basic properties, e.g. operating system where the evidence can be launched. To solve these problems, we decided to move the implementation to a database. The database contained all the relevant information of each attack evidences. Meanwhile, the evidences themselves resided in a single plug-in directory. The database was created in Java and contains a total of eight fields, which maybe expanded in the future. Table 7-4 summarizes the fields implemented in the database.

Table 7-4: Fields Included in the Database

Name of Field	Description
Name of evidence	String representing the given name to the evidence
Executable type	String providing the type of executable (shell script, ruby)
Cleanup name	Name of the API to call for cleanup
Origin	String describing the origin of the attack evidence
About	Information of the functionality provided by the attack evidence
OS type	Set of operating systems where the evidence can be launched
Evidence key	Unique identifier for the evidence
Data sets	The sets of data where the attack manifests

We set as requirements for the database to be lightweight and easily integrated with a Java program for portability. However, the implementation needed to be compatible with SQL commands. The solution used in this second implementation of the HIDS

stimulator was Java DB, also known as Apache Derby. Apache Derby is a java based SQL compliant database that is now part of Java version 6. But because some user may utilize older versions of JAVA, the required files to implement the database are included as part of the program.

To access the database a graphical interface was created, where the user could search and select the evidences. The evidences could be selected in different ways, for example, filtering by specific information or by name. A screenshot of the selection process for attack evidences can be seen in Figure 7-7.

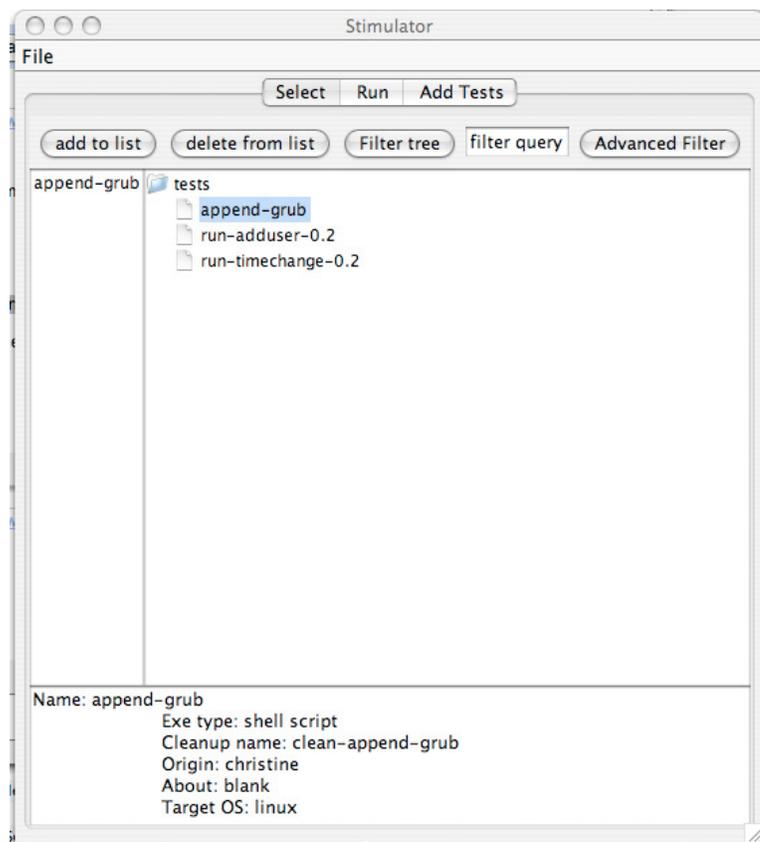


Figure 7-7. Selection Screen for the HIDS Stimulator

The interface also provides the possibility to upload new attack evidences. The graphical interface detects the type of executable and labels accordingly, providing

more flexibility in the language and platform the tests could be created. The process of uploading an attack evidence was greatly simplified from the previous version. Through the graphical interface, the user can select the file to be added as an attack evidence. To reduce the file clutter, the functionality to revert to the previous state was implemented through an argument passed to the evidence executable. Once a file is selected, a window pops up and the creator is given the possibility to fill specific fields, e.g. the platform where the evidence can be launched. These options were provided as a simple multichoice environment, to be used later to filter. We can see a screenshot of the process of uploading an attack evidence in Figure 7-8.

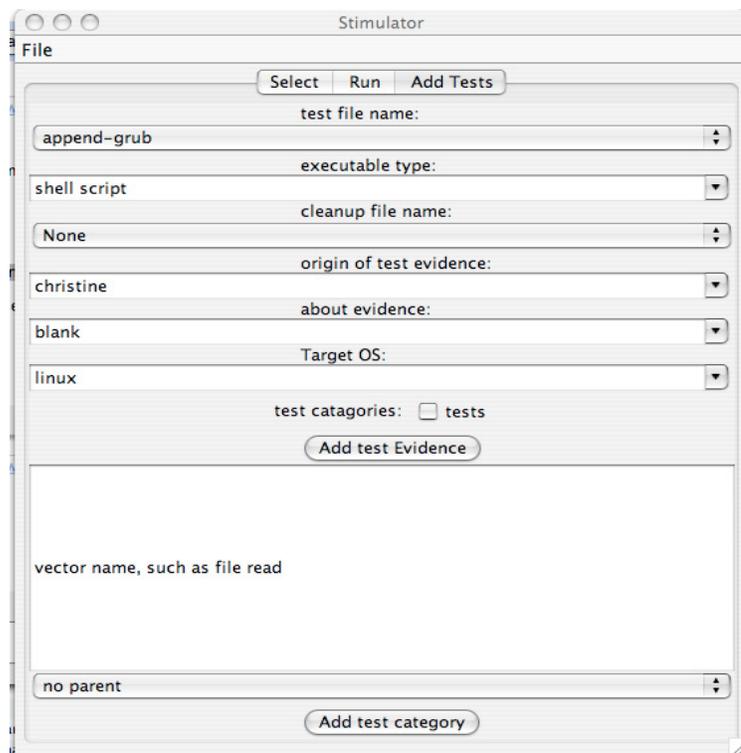


Figure 7-8: Uploading an Attack Evidence

The outcome of the launched evidence appears in the graphical interface. The results still need to be paired with the HIDS log.

7.4.3 Sample Attack Evidences

Currently, 16 attack evidences have been implemented, most of them taken from common activities performed by attackers, or by extracting payloads from attack libraries. Next we describe some coded attack evidences:

- Time change test: a filesystem evidence that modifies the timestamp of a random executable. It first backs up the original timestamp before running the test. The test itself changes the timestamp, and the cleaner restores the original timestamp.
- Add user test: a file system test that checks to see whether /etc/ can be read from, can be written to, and can make a copy of /etc/passwd. If those conditions pass, then the test runs a shell script that adds a user to /etc/passwd named 'Sploit', with the password 'ABC'. The cleaner file restores the modified /etc/passwd with its original values, thus preserving the original state.

Other tests on filesystem data checked random file creation on sensitive directories, file corruption, log tampering and changing permissions of random files. For system memory data set we launched rogue processes, and then kill them. In the future we expect to expand the library of evidences to successfully evaluate HIDSs under the proposed methodology

7.5 Conclusions

In this chapter, we surveyed previous efforts to create IDS frameworks. A link exists between the operational costs of the HIDS, related to efficiency and transparency, and

the operational cost defined by previous cost frameworks. We used this relationship to integrate efficiency and transparency into the cost framework. We then showed how to integrate the metrics defined for attack resiliency and visibility as part of the total calculation for probability of true positives. Finally, we provided two practical examples on how to integrate characteristics. First, we showed how the relationship between accuracy and attack resiliency can affect the calculation of the optimal operational point of a HIDS. Then we propose a tool, which integrates the study of visibility and accuracy by creating a database of attack evidences. The attack evidences are labeled depending on the data set where they manifest, allowing the use of only evidences that can be detected by the HIDS evaluated.

Chapter 8: Conclusions and Future Work

In this dissertation, we researched methods to improve the evaluation of host intrusion detection system (HIDSs). We showed that HIDSs were now important security tools, but their evaluation was still performed using the evaluation methods proposed for network intrusion detection systems (NIDS). Hence, an evaluation of a HIDS based the outcome solely on the evaluation of detection accuracy. However, we showed that evaluating only detection accuracy is insufficient for HIDSs, as they portray specific characteristics that also need to be considered. To solve this problem we explored distinct HIDS traits, proposing a set of quantifiable characteristics. The proposed characteristics were the ability to collect the data where an attack manifests (visibility); the ability of the detection engine of the HIDS to raise alerts only in the event of an intrusion, when the intrusion manifest in the data collected (detection accuracy); the ability of the HIDS to resist subversion attacks in the event of an intrusion (attack resiliency); the ability to timely detect attacks (efficiency); and the ability of the HIDS to avoid interfering with the normal functioning of the system supervised (transparency).

For evaluating efficiency and transparency, some practical quantification had been performed, but no methodology existed to perform the evaluation. For the evaluation of visibility and attack resiliency, little research had been conducted, and to the best of our knowledge, no efforts for quantification existed previous to this one.

We identified metrics related to each of the proposed characteristics. To quantify visibility, we studied the concepts of probability of attack manifestations, and the

quality of the data before entering the detection engine. We empirically applied the proposed metrics for visibility on filesystem data. The data required for the evaluation was collected over a period of 24 days and contained all system calls created as part of a SSH compromise. We studied these results to quantify the probability of attack manifestation for each file on the filesystem. Then we extracted a set of attack evidences (e.g., password modification) and applied the proposed metrics to evaluate the quality of each file to detect the attack evidence. We successfully identified the most relevant files to audit for HIDSs based on filesystem data. Our study demonstrated that both the probability of manifestation and quantification of data quality are necessary to provide a full assessment of visibility.

The probability of subversion was introduced as a metric to quantify attack resiliency. Quantifying attack resiliency was difficult, as data on attempts to modify a HIDS during the course of an attack is rarely available. To overcome this limitation, we suggested estimating how often these subversion attempts will be successful. We restricted the scope of our research to attacks directed towards the HIDS performed as part of an attack towards the system supervised. We introduced the independency score, based on the cost of subverting sections of the HIDS. The independency score was then used to estimate the probability of subversion. We showed the feasibility of this approach by evaluating the independency score for the Samhain HIDS. Our results showed that the independency score was very low, making it very simple for the attacker to subvert the HIDS. Thus, the empirical results confirmed the necessity of considering attack resiliency while evaluating HIDSs.

We found that the quantification of transparency lacked standardization. Previous evaluations of transparency [PZC+96] were composed by a set of performance tests, disregarding the type of resources shared between the system supervised and the HIDS. We proposed the percentage of performance reduction as the metric to quantify transparency. Then, we studied the use of performance micro benchmarks [Bon, Ioz] and macro benchmarks [Htt, Pos] to evaluate the transparency of the HIDS. We found that micro benchmarks provide results that are more accurate, but are more difficult to apply practically. This is because more than one micro benchmark might be necessary for a successful evaluation, and therefore the evaluator needs a deeper understanding of the HIDS functionality to pick the right set of micro benchmarks. Macro benchmarks, on the other hand, provide a simpler method to evaluate transparency, but have to be configured properly to provide a good evaluation. Then, we evaluated the transparency of Samhain [Sam] and Osiris [Osi], two HIDSs based on filesystem data. The evaluation was performed using the Postmark [Pos] macro benchmarks with different configurations. The evaluation showed better transparency results of Samhain over Osiris, but also that modifications in configuring the macro benchmark affected the evaluation.

We studied approaches to evaluate efficiency. We quantified efficiency using timing measurements. We introduced a set of timings that are useful to measure the overall efficiency. In particular, we showed empirically that the efficiency of the system may be modified by the type of data collected. This is because a delay may exist between the start of an attack and the manifestation of the attack in the data collected. The experiments were based on studying the time for an attack to create activity on a set

of files on a filesystem. After this, we evaluated efficiency for both Osiris and Samhain. The results showed that Samhain takes ten times more time to provide the results than Osiris, while perform the same type of detection. This shows the importance of evaluating efficiency to understand the benefits of a HIDS.

A major problem while comparing several metrics is to decide between two HIDS when one is superior while evaluating a characteristics, but inferior while evaluating another. As multiple metrics were proposed, we provided a framework to create a composite metrics. The proposed framework is based on a cost framework. We showed that a link exists between the operational costs of the HIDS, related to efficiency and transparency, and the operational cost defined by previous cost frameworks. We used this relationship to integrate efficiency and transparency into the cost framework. We then showed how to integrate the metrics defined for attack resiliency and visibility as part of the total calculation for probability of true positives. We provided two examples of integrating characteristics for evaluation. In the first example, we demonstrated how the optimal operational point of a HIDS is modified by the independency score. In the second, we presented a tool to integrate the evaluation of visibility and detection accuracy.

Future Work

This work presented new evaluation methods for HIDSs, demonstrating the importance and feasibility of including additional characteristics for evaluation. However, this study is only a starting point to better characterize the functioning of a HIDS through evaluation.

The study of visibility is promising for a better understanding of the role of data in intrusion detection. We empirically quantified a set of data, the filesystem, but the research community will benefit from the quantification of visibility for other data sets, like system calls or memory. In particular, systems based on data collected from the host, and then having the data sent to a server for remote verification [Gro03a] are gaining acceptance to improve the security of internal networks. Completing the study of visibility will provide information to optimize the type of data collected while creating these systems.

As part of the study of attack resiliency, we presented a set of approaches that may improve the independency score of a system. However, the exact impact of these approaches (e.g., virtualization [Lit05, GPMB03] in improving the independency score of the HIDS featuring these improvements is unknown. In this respect, a practical evaluation is necessary to evaluate attack resiliency for HIDSs implementing these improvements.

While evaluating transparency, the performance tests showed that configuration decisions have an effect on the results. Moreover, as several types of macro benchmarks and micro benchmarks exist, selecting the right tool for evaluation needs to be the subject of further study. It will be beneficial to perform an analysis of existing macro benchmarks and micro benchmarks, considering specifically what type of system resources are evaluated, and how modifying the configuration will affect the evaluation of the performance overhead.

We proposed the use of mapping functions to link efficiency and transparency to the operation cost of the HIDSs. These mapping functions may vary depending on

environments (e.g., type of system supervised). Similar environments will have similar mapping functions, and hence further research could be performed exploring methods to create more realistic curves based on empirical data. As an example, an empirical study on how a time delay on detecting an intrusion will reflect into financial losses for different types of environment will provide more realistic mapping functions to link efficiency with cost. Similarly, an empirical study on how performance overhead affects different systems (e.g., lost productivity in a desktop, lost connectivity in a webserver) will help defining the mapping curves for transparency.

The evaluation of each characteristic independently is a time consuming process. However, most of the proposed evaluations for each characteristic can be automated. We showed initial work in that direction with the implementation of the HIDS stimulator. Future work may lead to the implementation of the proper tools to automate the measurement of the proposed characteristics, and to automate the calculations required to integrate the measurements into a cost framework. As an example, the tool could provide the user with a choice between different environments and different types of HIDSs. The evaluation suite will perform the required calculations depending on the information provided by the user (e.g., selecting the right macro benchmark, choosing a suitable mapping function).

Appendices

Appendix A: Filesystem Visibility Metrics

A.1 Files Related To Password Modification Evidence

FILE NAME	P _M	Activity	IG	PPV	NPV
/usr/lib/libgmodule-2.0.so.0	0.515528	read	1.000	1.000	1.000
/usr/lib/libgobject-2.0.so.0	0.515528	read	1.000	1.000	1.000
/usr/lib/libglib-2.0.so.0	0.515528	read	1.000	1.000	1.000
/lib/libpam.so.0	0.515528	read	1.000	1.000	1.000
/usr/lib/libuser.so.1	0.515528	read	1.000	1.000	1.000
/lib/libpam_misc.so.0	0.515528	read	1.000	1.000	1.000
/lib/security/../../lib/security/pam_succeed_if.so	0.512422	read	0.973	1.000	0.994
/lib/security/../../lib/security/pam_permit.so	0.512422	read	0.973	1.000	0.994
/etc/pam.d/other	0.512422	read	0.973	1.000	0.994
/etc/pam.d/passwd	0.512422	read	0.973	1.000	0.994
/lib/security/../../lib/security/pam_unix.so	0.512422	read	0.973	1.000	0.994
/usr/lib/libcrack.so.2	0.512422	read	0.973	1.000	0.994
/lib/security/../../lib/security/pam_env.so	0.512422	read	0.973	1.000	0.994
/etc/pam.d/system-auth	0.512422	read	0.973	1.000	0.994
/lib/security/pam_stack.so	0.512422	read	0.973	1.000	0.994
/lib/security/../../lib/security/pam_limits.so	0.512422	read	0.973	1.000	0.994
/lib/security/../../lib/security/pam_deny.so	0.512422	read	0.973	1.000	0.994
/lib/security/../../lib/security/pam_cracklib.so	0.512422	read	0.973	1.000	0.994
/usr/lib/libpopt.so.0	0.518634	read	0.973	0.994	1.000

/etc/shadow	0.521739	read	0.951	0.988	1.000
/etc/nshadow	0.5	write	0.900	1.000	0.969
/usr/lib/cracklib_dict.pwd	0.5	read	0.900	1.000	0.969
/usr/lib/cracklib_dict.pwi	0.5	read	0.900	1.000	0.969
/usr/lib/cracklib_dict.hwm	0.5	read	0.900	1.000	0.969
/lib/libnsl.so.1	0.568323	read	0.710	0.902	0.993
/lib/libcrypt.so.1	0.596273	read	0.659	0.865	1.000
/etc/security/opasswd	0.413043	read	0.608	1.000	0.825
/etc/localtime	0.897516	read	0.095	0.571	0.970
/etc/shadow-	0.021739	write	0.021	1.000	0.495
/etc/passwd-	0.021739	write	0.021	1.000	0.495
/etc/login.defs	0.021739	read	0.021	1.000	0.495
/etc/skel/.bash_profile	0.018634	write	0.018	1.000	0.494
/etc/gshadow	0.018634	write	0.018	1.000	0.494
/etc/default/useradd	0.018634	write	0.018	1.000	0.494
/etc/skel/.zshrc	0.018634	write	0.018	1.000	0.494
/etc/skel/.bash_logout	0.018634	write	0.018	1.000	0.494
/etc/skel/.bashrc	0.018634	write	0.018	1.000	0.494
/etc/shadow+	0.018634	write	0.018	1.000	0.494
/etc/passwd+	0.018634	write	0.018	1.000	0.494
/bash/.bashrc	0.009317	write	0.009	1.000	0.489
/bash/.zshrc	0.009317	write	0.009	1.000	0.489
/bash/.bash_logout	0.009317	write	0.009	1.000	0.489
/bash/.bash_profile	0.009317	write	0.009	1.000	0.489
/home/zidan/.zshrc	0.006211	write	0.006	1.000	0.488
/home/zidan/.bashrc	0.006211	write	0.006	1.000	0.488

/home/zidan/.bash_logout	0.006211	write	0.006	1.000	0.488
/home/zidan/.bash_profile	0.006211	write	0.006	1.000	0.488
/home/admin/.zshrc	0.003106	write	0.003	1.000	0.486
/etc/group.6755	0.003106	write	0.003	1.000	0.486
/etc/passwd	0.003106	write	0.003	1.000	0.486
/etc/group.4831	0.003106	write	0.003	1.000	0.486
/etc/shadow.6455	0.003106	write	0.003	1.000	0.486
/etc/gshadow.6455	0.003106	write	0.003	1.000	0.486
/etc/passwd.5988	0.003106	write	0.003	1.000	0.486
/etc/passwd.5989	0.003106	write	0.003	1.000	0.486
/home/admin/.bash_profile	0.003106	write	0.003	1.000	0.486
/proc/self/attr/fscreate	0.003106	write	0.003	1.000	0.486
/etc/shadow.5988	0.003106	write	0.003	1.000	0.486
/etc/shadow.5989	0.003106	write	0.003	1.000	0.486
/etc/gshadow.5988	0.003106	write	0.003	1.000	0.486
/etc/gshadow.5989	0.003106	write	0.003	1.000	0.486
/etc/shadow.10687	0.003106	write	0.003	1.000	0.486
/etc/passwd.4831	0.003106	write	0.003	1.000	0.486
/etc/shadow.10608	0.003106	write	0.003	1.000	0.486
/etc/passwd.6755	0.003106	write	0.003	1.000	0.486
/etc/group.6455	0.003106	write	0.003	1.000	0.486
/home/admin/.bash_logout	0.003106	write	0.003	1.000	0.486
/etc/shadow	0.003106	write	0.003	1.000	0.486
/etc/shadow.4831	0.003106	write	0.003	1.000	0.486
/etc/group.10687	0.003106	write	0.003	1.000	0.486
/etc/passwd.10687	0.003106	write	0.003	1.000	0.486

/etc/shadow.6755	0.003106	write	0.003	1.000	0.486
/etc/gshadow.10687	0.003106	write	0.003	1.000	0.486
/etc/gshadow.6755	0.003106	write	0.003	1.000	0.486
/etc/group.10608	0.003106	write	0.003	1.000	0.486
/etc/group.5988	0.003106	write	0.003	1.000	0.486
/etc/group.5989	0.003106	write	0.003	1.000	0.486
/etc/passwd.10608	0.003106	write	0.003	1.000	0.486
/etc/gshadow.4831	0.003106	write	0.003	1.000	0.486
/etc/gshadow.10608	0.003106	write	0.003	1.000	0.486
/home/admin/.bashrc	0.003106	write	0.003	1.000	0.486
/etc/passwd.6455	0.003106	write	0.003	1.000	0.486
/tmp/own.so	0.003106	read	0.003	1.000	0.486
/usr/lib/libuser/libuser_files.so	0.003106	read	0.003	1.000	0.486
/etc/libuser.conf	0.003106	read	0.003	1.000	0.486
/usr/lib/libuser/libuser_shadow.so	0.003106	read	0.003	1.000	0.486
/lib/security/\$ISA/pam_limits.so	0.003106	read	0.003	1.000	0.486
/proc/sys/kernel/ngroups_max	0.034161	write	0.002	0.636	0.489
/var/run/utmp	0.664596	read	0.000	0.126	0.257
/etc/group	0.990683	read	0.000	0.043	0.009
/usr/share/locale/locale.alias	0.549689	read	0.000	0.034	0.301
/etc/passwd	1	write	0.000	0.516	0.000
/lib/libselinux.so.1	1	read	0.000	0.516	0.000
/proc/filesystems	1	read	0.000	0.516	0.000
/lib/libnss_files.so.2	1	read	0.000	0.516	0.000
/proc/mounts	1	read	0.000	0.516	0.000
/lib/libdl.so.2	1	read	0.000	0.516	0.000

/etc/nsswitch.conf	1	read	0.000	0.516	0.000
/etc/selinux/config	1	read	0.000	0.516	0.000
/lib/tls/libc.so.6	1	read	0.000	0.516	0.000

A.2 Files Related To Download Evidence

FILE	P _M	Activity	IG	PPV	NPV
/etc/wgetrc	0.375776	read	0.831	1.000	0.955
/usr/lib/libk5crypto.so.3	0.437888	read	0.822	0.922	1.000
/usr/lib/libgssapi_krb5.so.2	0.437888	read	0.822	0.922	1.000
/usr/lib/libkrb5.so.3	0.437888	read	0.822	0.922	1.000
/lib/libcom_err.so.2	0.440994	read	0.811	0.915	1.000
/lib/libresolv.so.2	0.444099	read	0.799	0.909	1.000
/lib/libssl.so.4	0.391304	read	0.764	0.968	0.959
/lib/libnss_dns.so.2	0.397516	read	0.764	0.961	0.964
/lib/libcrypto.so.4	0.428571	read	0.724	0.913	0.978
/etc/resolv.conf	0.406832	read	0.723	0.939	0.963
/usr/lib/libz.so.1	0.431677	read	0.713	0.906	0.978
/etc/hosts	0.413043	read	0.699	0.925	0.963
/lib/tls/librt.so.1	0.531056	read	0.367	0.708	0.940
/lib/tls/libpthread.so.0	0.537267	read	0.357	0.699	0.940
/usr/share/locale/locale.alias	0.549689	read	0.352	0.689	0.945
sendq.tgz	0.055901	write	0.079	1.000	0.632
/usr/lib/libkrb4.so.2	0.040373	read	0.038	0.923	0.618
/usr/lib/libdes425.so.3	0.040373	read	0.038	0.923	0.618
boti.zip	0.021739	write	0.030	1.000	0.610
neo.tgz	0.021739	write	0.030	1.000	0.610

udp.pl	0.015528	write	0.021	1.000	0.606
/home/admin/.netrc	0.012422	read	0.017	1.000	0.604
doki.tgz	0.012422	write	0.017	1.000	0.604
psyBETA.tgz	0.012422	write	0.017	1.000	0.604
bot.tgz	0.012422	write	0.017	1.000	0.604
psybnc.tgz	0.012422	write	0.017	1.000	0.604
/usr/lib/libidn.so.11	0.009317	read	0.013	1.000	0.602
/usr/lib/libcurl.so.3	0.009317	read	0.013	1.000	0.602
sshll.tgz	0.009317	write	0.013	1.000	0.602
playerz.tar	0.009317	write	0.013	1.000	0.602
Adi.tgz	0.009317	write	0.013	1.000	0.602
robot.tar	0.006211	write	0.008	1.000	0.600
psy.tgz	0.006211	write	0.008	1.000	0.600
Unreal3.2.5.tar.gz	0.006211	write	0.008	1.000	0.600
a3	0.006211	write	0.008	1.000	0.600
p.tgz	0.006211	write	0.008	1.000	0.600
teambot2.tar.gz	0.006211	write	0.008	1.000	0.600
sniff.tar.gz	0.006211	write	0.008	1.000	0.600
all.tar	0.006211	write	0.008	1.000	0.600
b3	0.006211	write	0.008	1.000	0.600
/lib/libutil.so.1	0.10559	read	0.006	0.529	0.611
/home/mysql/.netrc	0.003106	read	0.004	1.000	0.598
/var/log/trans/dorothy_120213.log.tgz	0.003106	read	0.004	1.000	0.598
/var/log/trans/wtmp_120213	0.003106	read	0.004	1.000	0.598
/var/log/trans/sshd_120212.log.tgz	0.003106	read	0.004	1.000	0.598
john.tar	0.003106	write	0.004	1.000	0.598

88.tgz	0.003106	write	0.004	1.000	0.598
psyBETA.tgz	0.003106	write	0.004	1.000	0.598
rai.tar.gz	0.003106	write	0.004	1.000	0.598
scan.tgz	0.003106	write	0.004	1.000	0.598
2k.tar.gz	0.003106	write	0.004	1.000	0.598
pico.tgz	0.003106	write	0.004	1.000	0.598
botteam3.tar.gz	0.003106	write	0.004	1.000	0.598
john.tar.gz	0.003106	write	0.004	1.000	0.598
psybnc-linux.tar	0.003106	write	0.004	1.000	0.598
boti.tar	0.003106	write	0.004	1.000	0.598
local.tar.gz	0.003106	write	0.004	1.000	0.598
botteam2.tar.gz	0.003106	write	0.004	1.000	0.598
mirkforce.tar.tar	0.003106	write	0.004	1.000	0.598
teambot.tar.gz	0.003106	write	0.004	1.000	0.598
/home/admin/vteam.zip	0.003106	write	0.004	1.000	0.598
crisi.tgz	0.003106	write	0.004	1.000	0.598
hm-udp.pl	0.003106	write	0.004	1.000	0.598
index.html	0.003106	write	0.004	1.000	0.598
reales.tgz	0.003106	write	0.004	1.000	0.598
holy	0.003106	write	0.004	1.000	0.598
je.tgz	0.003106	write	0.004	1.000	0.598
deep.tar.tgz	0.003106	write	0.004	1.000	0.598
bash.tgz	0.003106	write	0.004	1.000	0.598
bogdaneRk.tar.gz	0.003106	write	0.004	1.000	0.598
acycmech.tar.gz	0.003106	write	0.004	1.000	0.598
dare-mech2.tar	0.003106	write	0.004	1.000	0.598

shv.tgz	0.003106	write	0.004	1.000	0.598
26.tgz	0.003106	write	0.004	1.000	0.598
putty.exe	0.003106	write	0.004	1.000	0.598
psybnc-linux.tar.gz	0.003106	write	0.004	1.000	0.598
psyBNC-2.3.2-7.tar.gz	0.003106	write	0.004	1.000	0.598
web.tgz	0.003106	write	0.004	1.000	0.598
/home/admin/access.tar	0.003106	write	0.004	1.000	0.598
andra.txt	0.003106	write	0.004	1.000	0.598
teambot1.tar.gz	0.003106	write	0.004	1.000	0.598
alecs.tgz	0.003106	write	0.004	1.000	0.598
axe.tgz	0.003106	write	0.004	1.000	0.598
sir.tgz	0.003106	write	0.004	1.000	0.598
hienaXmech.tgz	0.003106	write	0.004	1.000	0.598
udp.txt	0.003106	write	0.004	1.000	0.598
flood-udp.tar	0.003106	write	0.004	1.000	0.598
2.6.jpg	0.003106	write	0.004	1.000	0.598
flood.tar.gz	0.003106	write	0.004	1.000	0.598
/etc/localtime	0.897516	read	0.004	0.415	0.697
/lib/tls/libc.so.6	1	read	0.000	0.404	0.000
/lib/libselinux.so.1	1	read	0.000	0.030	0.000
/lib/libnss_files.so.2	1	read	0.000	0.400	0.000
/lib/libnsl.so.1	0.568323	read	0.000	0.033	0.108
/lib/libcrypt.so.1	0.596273	read	0.000	0.094	0.138
/etc/host.conf	0.086957	read	0.000	0.321	0.588
/etc/passwd	1	read	0.000	0.086	0.000
/etc/services	0.096273	read	0.000	0.387	0.595

/dev/tty	0.031056	read	0.000	0.400	0.596
/proc/mounts	1	read	0.000	0.030	0.000
/lib/libdl.so.2	1	read	0.000	0.396	0.000
/etc/nsswitch.conf	1	read	0.000	0.400	0.000
/var/run/utmp	0.664596	read	0.000	0.086	0.353
/etc/selinux/config	1	read	0.000	0.030	0.000
psync.conf	0.015528	write	0.000	0.200	0.593

A.3 Files Related To Reconnaissance Evidence

FILE	P _M	Activity	IG	PPV	NPV
/proc/x/stat	0.683230	read	0.655	1.000	0.725
/proc/x/cmdline	0.683230	read	0.655	1.000	0.699
/proc/uptime	0.711180	read	0.454	0.961	0.699
/proc/stat	0.711180	read	0.454	0.961	0.684
/lib/libproc-3.2.3.so	0.704969	read	0.440	0.960	0.349
/proc/cpuinfo	0.341615	read	0.210	1.000	0.487
/proc/loadavg	0.636646	read	0.195	0.917	0.329
/proc/x/status	0.301242	read	0.179	1.000	0.315
/proc/tty/drivers	0.270186	read	0.157	1.000	0.315
/proc/sys/kernel/pid_max	0.270186	read	0.157	1.000	0.315
/proc/self/stat	0.270186	read	0.157	1.000	0.444
/var/run/utmp	0.664596	read	0.117	0.879	0.238
/etc/issue	0.034161	read	0.017	1.000	0.237
/proc/x/enviro	0.031056	read	0.015	1.000	0.237
/proc/x/statm	0.031056	read	0.015	1.000	0.237
/lib/libproc.so.2.0.6	0.031056	read	0.015	1.000	0.236

/usr/include/proc.h	0.027950	read	0.014	1.000	0.236
/usr/include/hosts.h	0.027950	read	0.014	1.000	0.236
.bash_history	0.027950	read	0.014	1.000	0.234
/var/log/wtmp	0.018634	read	0.009	1.000	0.231
raw.set	0.006211	read	0.003	1.000	0.231
mech.session	0.006211	read	0.003	1.000	0.231
/usr/lib/+c0d.init	0.006211	read	0.003	1.000	0.231
/usr/lib/named/named.sn	0.006211	read	0.003	1.000	0.231
/tmp/info_tmp	0.006211	read	0.003	1.000	0.231
ssh.log	0.006211	read	0.003	1.000	0.231
/tmp/own.so	0.003106	read	0.002	1.000	0.231
/dev/shm/shv5/bin/.sh/shdcf2	0.003106	read	0.002	1.000	0.231
m.lev	0.003106	read	0.002	1.000	0.231
vhosts	0.003106	read	0.002	1.000	0.231
/tmp/.stats	0.003106	read	0.002	1.000	0.231
/tmp/.init2	0.003106	read	0.002	1.000	0.231
/usr/local/games/.kde/bin/.sh/shdcf2	0.003106	read	0.002	1.000	0.231
82.146.pscan.22	0.003106	read	0.002	1.000	0.231
ARSEX3	0.003106	read	0.002	1.000	0.231
58.10.pscan.22	0.003106	read	0.002	1.000	0.231
/usr/include/prthead.h	0.003106	read	0.002	1.000	0.231
Neo.seen	0.003106	read	0.002	1.000	0.231
/etc/inittab	0.003106	read	0.002	1.000	0.231
m.ses	0.003106	read	0.002	1.000	0.231
m.set	0.003106	read	0.002	1.000	0.231
/tmp/.procs	0.003106	read	0.002	1.000	0.231

vuln.txt	0.003106	read	0.002	1.000	0.231
128.8.pscan.22	0.003106	read	0.002	1.000	0.231
ports	0.003106	read	0.002	1.000	0.231
logo	0.003106	read	0.002	1.000	0.003
/etc/group	0.990683	read	0.000	0.120	0.126
/etc/shadow	0.521739	read	0.000	0.049	0.000
/proc/meminfo	1.000000	read	0.000	0.567	0.121
/usr/share/locale/locale.alias	0.549689	read	0.000	0.129	0.168
unix1.users	0.077640	read	0.000	0.040	0.157
/etc/hosts	0.413043	read	0.000	0.314	0.000
/lib/libnss_files.so.2	1.000000	read	0.000	0.720	0.000
/etc/nsswitch.conf	1.000000	read	0.000	0.720	0.000
/etc/passwd	1.000000	read	0.000	0.722	0.000
/etc/mtab	1.000000	read	0.000	0.747	0.114
/etc/localtime	0.897516	read	0.000	0.740	0.226
mech.set	0.009317	read	0.000	0.333	0.228
/etc/rc.d/rc.sysinit	0.006211	read	0.000	0.500	0
/lib/tls/libc.so.6	1.000000	read	0.000	0.770	0.725

References

- [AAL+03] Nicholas Athanasiades, Randal Abler, John G. Levine, Henry L. Owen, and George F. Riley. Intrusion detection testing and benchmarking methodologies. In *Proc. of the Information Assurance Workshop (IEEE IAW'03)*, pages 63–72, March 2003.
- [And80] James P. Anderson. Computer security threat monitoring and surveillance. 1980.
- [Axe00a] Stefan Axelsson. The base-rate fallacy and the difficulty of intrusion detection. *ACM-Transactions on Information and System Security -TISSEC*, 3(3):186–205, 2000.
- [Axe00b] Stefan Axelsson. Intrusion-detection systems: A Taxonomy and Survey. Technical Report 99–15, Department of Computer Engineering, Chalmers University of Technology, SE–412 96, Göteborg, Sweden, March 2000.
- [BDF+03] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *Proc. of the Nineteenth ACM Symposium on Operating Systems Principles (SOSP '03)*, pages 164–177, Bolton Landing, NY, USA, October 2003.
- [Bes03] Jean-Luc Besson. Next generation intrusion detection and prevention for complex environments. Master's thesis, Department of Information Technology University of Zurich, Switzerland, 2003.
- [BGFI+98] Jai Balasubramanian, Jose Omar Garcia-Fernandez, David Isaco, Eugene Spafford, and Diego Zamboni. An architecture for intrusion detection using autonomous agents. In *Proc. of the Fourteenth Annual Computer Security Application Conference (ACSAC '98)*, pages 13–24, 1998.
- [Bis02] Matt Bishop. *Computer Security: Art and Science*. Addison Wesley, New York, 2002.
- [BJ04] Emilie Lundin Barse and Erland Jonsson. Extracting attack manifestations to determine log data requirements for intrusion detection. In *Proc. of the Annual Computer Security Applications Conference (ACSAC'04)*, pages 158–167, 2004.
- [Bla01] George Robert Blakley. An imprecise but necessary calculation. *Secure Business Quarterly: Special Issue on Return of Security Investment*, 2(1):20–25, 2001.
- [Bon] bonnie++. <http://www.die.net/doc/linux/man/man8/bonnie++.8.html>.

- [Bru99] David Brumley. Invisible intruders: rootkits in practice. *login; Magazine, Intrusion Detection Special Issue*, 9:<http://www.usenix.org/publications/login/1999-9/features/rootkits.html>, 1999.
- [CBS06] Alvaro A. Cardenas, John S. Baras, and Karl Seamon. A framework for the evaluation of intrusion detection systems. In *Proc. of the IEEE Symposium on Security and Privacy (IEEE security'06)*, pages 63–77, Washington, DC, USA, 2006. IEEE Computer Society.
- [CC03] Suresh N. Chari and Pau-Chen Cheng. BlueBoX: A policy-driven, host-based intrusion detection system. *ACM Transactions on Information and System Security*, 6(2):173–200, 2003.
- [Cor05] Science Application International Corporation. Intrusion detection system scanner protection profile. Technical report, 2005.
- [DDW99] Herver Debar, Marc Dacier, and Andreas Wespi. Towards a taxonomy of intrusion-detection systems. *Computer Networks*, 31(8):805–822, 1999.
- [Den87] Dorothy Denning. An intrusion detection model. *IEEE Transactions on Software Engineering*, 13:222, 1987.
- [DM02] Herver Debar and Benjamin Morin. Evaluation of the diagnostic capabilities of commercial intrusion detection systems. In *Proc. of the Recent Advances in Intrusion Detection (RAID '02)*, volume 2516, pages 177–191, 2002.
- [EGK00] Steven T. Eckmann, Vigna Giovanni, and Richard A. Kemmerer. STATL: An attack language for state-based intrusion detection. In *Proc. of the Workshop on Intrusion Detection Systems (ACM '00)*, pages 43–57, 2000.
- [Ent04] Enterasys. Enterasys dragon intrusion defense system. Technical report, 2004.
- [FOCT02] Glenn Fink, Karen F. O'Donoghue, Brett L. Chappell, and T. G. Turner. A metrics-based approach to intrusion detection system evaluation for distributed real-time systems. In *Proc. of the International Parallel and Distributed Processing Symposium (IPDPS '02)*, pages 204–228, 2002.
- [GDF+06a] Guofei Gu, Prahlad Fogla, David Dagon, Wenke Lee, and Boris Skoric. Measuring intrusion detection capability: an information-theoretic approach. In *Proc. of the ACM Symposium on Information, Computer and Communications Security (ASIACCS '06)*, pages 90–101, New York, NY, USA, 2006. ACM Press.

- [GDF+06b] Guofei Gu, Prahlad Fogla, David Dagon, Wenke Lee, and Boris Skoric. Towards an information-theoretic framework for analyzing intrusion detection systems. In *Proc. of the European Symposium on Research in Computer Security (ESORICS '06)*, pages 527–546, 2006.
- [GL02] Lawrence A. Gordon and Martin P. Loeb. The economics of information security investment. *ACM Transactions on Information and System Security*, 5(4):438–457, 2002.
- [GPMB03] Tal Garfinkel, Ben Pfaff, Jim Chow Mendel, and Rosenblum Dan Boneh. Terra: A virtual machine-based platform for trusted computing. In *Proc. of the 19th Symposium on Operating System Principles (SOSP '03)*, pages 145–159, 2003.
- [GR03] Tal Garfinkel and Mendel Rosenblum. A virtual machine introspection based architecture for intrusion detection. In *Proc. of the Network and Distributed System Security Symposium (NDSS '03)*, pages 124–136, February 2003.
- [Gra03] David Grawrock. LaGrande architecture. Technical Report SCMS-18, Intel, September 2003.
- [Gro03a] Trusted Computing Group. Trusted computing platform alliance (TCPA) main specification, version 1.1b. Technical report, 2003.
- [Gro03b] Trusted Computing Group. Trusted computing platform alliance (TCPA) main specification, version 1.1b. Technical report, 2003.
- [GSS99] Anup K. Ghosh, Aaron Schwartzbard, and Michael Schatz. Learning program behavior profiles for intrusion detection. In *Proc. of the Workshop on Intrusion Detection and Network Monitoring*, pages 51–62, Berkeley, CA, USA, 1999. USENIX Association.
- [Gui65] Joy Paul Guilford. *Fundamental Statistics in Psychology and Education*, 4th ed. McGraw-Hill, New York, 1965.
- [Hal97] Halflife. Bypassing integrity checking systems. *Phrack Magazine*, 7, Issue 51(51):<http://www.trust-us.ch/phrack/show.php?p=51&a=9>, September 1997.
- [HB] Xie Huagang and Philippe Biondi. LIDS: linux intrusion detection system. <http://www.lids.org>.
- [HD04] James Hendricks and Leendert Van Doorn. Secure bootstrap is not enough: Shoring up the trusted computing base. In *Proc. of the 11th SIGOPS European Workshop (ACMSIGOPS 04')*, pages 230–247, September 2004.
- [Htt] httpperf. <http://www.hpl.hp.com/research/linux/httpperf/>.

- [IoZ] Iozone filesystem benchmark. <http://www.iozone.org>.
- [ISO98] ISO/IEC Standard 15408. *Common Criteria for Information Technology Security Evaluation*, version 2.0 edition, November 1998.
- [JO97] Erland Jonsson and Tomas Olovsson. A quantitative model on the security intrusion process based on attacker behavior. 23:136–144, April 1997.
- [JU01] John E. Gaffney Jr. and Jacob W. Ulvila. Evaluation of intrusion detectors: A decision theory approach. In *Proc. of the Symposium on Security and Privacy (IEEE security'01)*, pages 50–71, 2001.
- [Kah05] Erkan Kahraman. Evaluating it security performance with quantifiable metrics. Master's thesis, Institutionen for Data- och Systemvetenskap, DSV SU/KTH, June 2005.
- [KMT04] Kevin S. Killourhy, Roy A. Maxion, and Kymie M. C. Tan. A defense-centric taxonomy based on attack manifestations. In *Proc. of the International Conference on Dependable Systems and Networks (DSN '04)*, pages 102–123, 2004.
- [Koj] Tomasz Kojm. ClamAV. <http://www.clamav.net>.
- [KS94] Gene H. Kim and Eugene H. Spafford. The design and implementation of tripwire: A file system integrity checker. In *Proc. of the Conference on Computer and Communications Security (ACM '94)*, pages 18–29, 1994.
- [Lan81] Carl E. Landwehr. Formal models for computer security. *ACM Comput. Surv.*, 13(3):247–278, 1981.
- [LCT+02] Wenke Lee, Joao B. D. Cabrera, Ashley Thomas, Niranjana Balwalli, Sunmeet Saluja, and Yi Zhang. Performance adaptation in real-time intrusion detection systems. In *Proc. of Recent Advances in Intrusion Detection (RAID '02)*, pages 252–273, 2002.
- [LFM+02] Wenke Lee, Wei Fan, Matthew Miller, Salvatore J. Stolfo, and Erez Zadok. Toward cost-sensitive modeling for intrusion detection and response. *Journal of Computer Security*, 10(1/2):5–22, 2002.
- [LHFK00] Richard Lippmann, Joshua W. Haines, David J. Fried, Jonathan Korba, and Kumar Das. The 1999 DARPA off-line intrusion detection evaluation. *Computer Networks*, 1(34):579–595, 2000.
- [Lit05] Lionel Litty. Hypervisor-based intrusion detection. Master's thesis, University of Toronto, 2005.

- [LP01] Ulf Lindqvist and Phillip A. Porras. eXpert-BSM: A host-based intrusion detection solution for sun solaris. In *Proc. of the Annual Computer Security Applications Conference (ACSAC'01)*, pages 240–251, 2001.
- [LV] Rami Lehti and P. Virolainen. AIDE: Advanced intrusion detection environment. <http://www.cs.tut.fi/rammer/aide.html>.
- [LX01] Wenke Lee and Dong Xiang. Information-theoretic measures for anomaly detection. volume 134, pages 130–143. IEEE Computer Society Washington, DC, USA, 2001.
- [LZHH02] Peter Lichodziejewski, A. Nur Zincir-Heywood, and Malcolm I. Heywood. Host-based intrusion detection using self-organizing maps. In *Proc. of the International Joint Conference on Neural Networks (IEEE '02)*, pages 233–240. IEEE, May 2002.
- [MA02] Jesus Molina and William A. Arbaugh. Using independent auditors as intrusion detection systems. In *Proc. of the 4th International Conference on Information and Communications Security (ICICS'02)*, pages 291–302, Singapore, December 2002.
- [McH00] John McHugh. Testing intrusion detection systems: A critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Transactions on Information and System Security*, 3, Issue 4(4):262–294, November 2000.
- [MDWZ04] Yevgeniy Miretskiy, Abhijith Das, Charles P. Wright, and Erez Zadok. AVFS: An on-access anti-virus file system, 2004.
- [Met] Metasploit Web site. <http://www.metasploit.com/>.
- [MHL+03] Peter Mell, Vincent Hu, Richard Lippmann, Josh Haines, and Marc Zissman. An overview of issues in testing intrusion detection systems. Technical Report NIST IR 7007, National Institute of Standards and Technology, August 2003.
- [Mol01] Jesus Molina. Using independent auditors for intrusion detection systems. Master's thesis, University of Maryland at College Park, 2001.
- [MSC02] David Moore, Colleen Shannon, and Klein Claffy. Code-red: a case study on the spread and victims of an internet worm. pages 273–284. ACM Press New York, NY, USA, 2002.
- [Orm03] Hilarie K. Orman. The morris worm: A fifteen-year perspective. *IEEE Security & Privacy*, 1(5):35–43, 2003.

- [Osi] Osiris. <http://osiris.shmoo.com/>.
- [PFMA04] Nick L. Petroni, Timothy Fraser, Jesus Molina, and William A. Arbaugh. Copilot - a coprocessor-based kernel runtime integrity monitor. In *Proc. of the USENIX Security Symposium (USENIX '04)*, pages 179–194, 2004.
- [PKSZ04] Swapnil Patil, Anand Kashyap, Gopalan Sivathanu, and Erez Zadok. I3fs: An in-kernel integrity checker and intrusion detection file system. In *Proc. of the 18th USENIX Conference on System Administration (LISA '04)*, pages 67–78, Berkeley, CA, USA, 2004. USENIX Association.
- [PN97] Philip A. Porras and Peter G. Neumann. Emerald: Event monitoring enabling responses to anomalous live disturbances. In *Proc. of the 20th National Information Systems Security Conference*, pages 353–365, Baltimore, Maryland, USA, OctOctober 1997. NIST, National Institute of Standards and Technology/National Computer Security Center.
- [PN98] Thomas H. Ptacek and Timothy N. Newsham. Insertion, evasion, and denial of service. *Secure Networks, Inc.*, 1, January 1998.
- [Por92] Philip A. Porras. STAT - a state transition analysis tool for intrusion detection. Master's thesis, Computer Science Department, University of California, Santa Barbara, June 1992.
- [Pos] Postmark. <http://www.netapp.com>.
- [Ps03] David Powell and Robert Stroudand. *Conceptual Model and Architecture for MAFTIA*. University of Newcastle upon Tyne, Computing Science; Universidade de Lisboa and Zurich Research Laboratory, 2003.
- [PTJC05] Susmit Panjwani, Stephanie Tan, Keith M. Jarrin, and Michel Cukier. An experimental evaluation to determine if port scans are precursors to an attack. In *Proc. of the International Conference on Dependable Systems and Networks (DSN '05)*, pages 602–611, 2005.
- [PZC+96] Nicholas J. Puketza, Kui Zhang, Mandy Chung, Biswanath Mukherjee, and Ronald A. Olsson. A methodology for testing intrusion detection systems. *IEEE Transactions on Software Engineering*, 22(10):719–729, 1996.
- [RBC07] Daniel Ramsbrock, Robin Berthier, and Michel Cukier. Profiling attacker behavior following ssh compromises. In *Proc. of the International Conference on Dependable Systems and Networks (DSN '07)*, pages 119–124, 2007. IEEE Computer Society.

- [Roe99] Martin Roesch. Snort: Lightweight intrusion detection for networks. In *Proc. of the Conference on System Administration (USENIX '99)*, pages 229–238. USENIX Association, 1999.
- [RVK05] Helayne T. Ray, Raghunath Vemuri, and Hariprasad R. Kantubhukta. Towards an automated attack model for red teams. *IEEE security and Privacy*, 3(4):18–625, July-August 2005.
- [Sam] Samhain. <http://www.la-samhna.de/samhain/>.
- [Sch04] Stuart Schechter. *Computer Security Strength & Risk: A Quantitative Approach*. PhD thesis, Harvard University, Cambridge, Massachusetts, 2004.
- [Seb] Sebek. [http://www.honeynet.org/tools/sebek/Related work](http://www.honeynet.org/tools/sebek/Related%20work).
- [SPvD05] Elaine Shi, Adrian Perrig, and Leendert van Doorn. Bind: A fine-grained attestation service for secure distributed systems. In *Proc. of the Symposium on Security and Privacy (IEEE security'05)*, pages 154–168, 2005.
- [SSMF03] Lambert Schaelicke, Thomas Slabach, Branden Moore, and Curt Freeland. Characterizing the performance of network intrusion detection sensors. In *Proc. of Recent Advances in Intrusion Detection (RAID '03)*, pages 155–172, 2003.
- [SZ00] Eugene H. Spafford and Diego Zamboni. Intrusion detection using autonomous agents. *Comput. Networks*, 34(4):547–570, 2000.
- [Str] Strace. <http://sourceforge.net/projects/strace>.
- [VK99] Giovanni Vigna and Richard A. Kemmerer. Netstat: A network-based intrusion detection system. *Journal of Computer Security*, 7(1):37–71, 1999.
- [VMw04] Inc VMware. Vmware gsx server 3.1. 2004.
- [Web98] Daniel J. Weber. A taxonomy of computer intrusions. Master's thesis, Massachusetts Institute of Technology, June 1998.
- [WFP99] Christina Warrender, Stephanie Forrest, and Barak Pearlmutter. Detecting intrusions using system calls: Alternative data models. *Proc. of the Symposium on Security and Privacy (IEEE security'99)*, 145:123–132, 1999.
- [Win03] WinHeck. AMD platform for trustworthy computing. Technical report, Microsoft Corporation, September 2003.
- [Wot05] Brian Wotring, *Host Integrity Monitoring Using Osiris and Samhain*. McGraw-Hill, New York, 2005.

