2003-03

How intrusion detection can improve software decoy applications

Monteiro, Valter

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/1085

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

HOW INTRUSION DETECTION CAN IMPROVE SOFTWARE DECOY APPLICATIONS

by

Valter Monteiro, Junior

March 2003

Thesis Advisor: Neil C. Rowe

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** March 2003	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE How Intrusion Detection Can Improve Software Decoy Applications	**5. FUNDING NUMBERS**
6. AUTHOR (S) Valter Monteiro, Junior	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the U.S. Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT *(maximum 200 words)*

This research concerns information security and computer-network defense. It addresses how to handle the information of log files and intrusion-detection systems to recognize when a system is under attack. But the goal is not the usual one of denying access to the attacker but providing a justification for deceptive actions to fool the attacker. We implemented a simple demonstration of how two different kinds of open-source intrusion-detection systems can efficiently pool data for this purpose.

14. SUBJECT TERMS Intelligent Software Decoy, Intrusion Detection, Computer Deception, Response Mechanism, Log File Monitor	**15. NUMBER OF PAGES** 85
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

THIS PAGE INTENTIONALLY LEFT BLANK

HOW INTRUSION DETECTION CAN IMPROVE SOFTWARE DECOY CAPABILITIES

Valter Monteiro, Junior
Lieutenant Commander, Brazilian Navy
Electronic Engineering (B.S.), Universidade de Sao Paulo, 1994

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 2003

Author: Valter Monteiro, Junior

Approved by: Neil C. Rowe
Thesis Advisor

J.D. Fulp
Second Reader

Peter J. Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This research concerns information security and computer-network defense. It addresses how to handle the information of log files and intrusion-detection systems to recognize when a system is under attack. But the goal is not the usual one of denying access to the attacker but providing a justification for deceptive actions to fool the attacker. We implemented a simple demonstration of how two different kinds of open-source intrusion-detection systems can efficiently pool data for this purpose.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGEMENTS

This thesis contains many others’ contributions. These include those of the professors and students who worked on the software decoy project, and the group of specialized professors and students whose combined efforts developed this topic. I extend my thanks to these individuals for their support during the last 6 months.

I would especially like to thank Dr. Neil Rowe for all the support, patience and direction that he gave me. He consistently guided me in the right direction and found usefulness in all my work.

I also want to thank Prof. Geoffrey Xie, Prof J.D. Fulp and CDR Nuccio Zuquello. Without Prof. Xie’s friendliness, support and his lab and direct study orientation, this thesis would have been impossible. Prof. J.D. Fulp’s friendship, advice and instruction in teaching in the classroom were instrumental in my learning. Finally, thanks to CDR Nuccio Zuquello, my Brazilian advisor, who sent small e-mails with the right words at the right moment.

Collectively, I also want to thank all of the professors of CISR for teaching me the concepts of good security and all that I know about computer security.

This work is dedicated to my parents who gave me life and the opportunity of education; to my wife, Suzana, for her outstanding support and her extraordinary sense to see the good side of everything; to Stephanno, my oldest son

and best friend, for always encouraging me to spend time with him even for just a while; and finally to Guiga, my special youngest son, who has taught me the real logic of life by simply asking me every day, “Papai happy”? I will always work hard to keep these three special people happy.

I. INTRODUCTION

In the "information age" the world became a more interconnected place. Critical infrastructure, business operations, bank operations, military operations, and communication systems are totally dependent on computer systems that control almost all aspects of life. The global network includes faxes, cellular phones, satellites, and more than 650 million people connected to the Internet.

With this scenario, every year the larger software corporations in the world, such as Microsoft, SUN, Oracle, and Linux Distributors, release a new version of their operating systems (OS), service packs, databases, and desktop and server applications for all of those platforms. Usually security is not the main goal of all of these programs and, even if it were, checking the integrity of all these millions of lines of code is impossible.

Added to this is the number of easily available "hacker" web sites on the Internet providing tools to exploit this ocean of vulnerabilities. A factor urging a change in the approach to defending networks is that hacker tools are becoming more automatic and no longer require deep knowledge to use them. Only a few minutes of exposure could create millions of dollars of losses due to sensitive information becoming accessible by the enemy.

Michael and Riehle suggested in [1] a different defending information systems approach: "Intelligent Software Decoys." This approach borrows ideas from military strategy. Instead of blocking or fighting attackers as soon as they are detected, a decoy system tries to keep them occupied by making them believe that the assault is successful and progressing as expected. To do this, we must decide how to detect the intrusion, how to respond to this attack, and how to implement decoy capabilities.

Implementing a complete decoy system is out of the scope of this research. Therefore, the experimental design that we developed is only a proof of concept. It shows how to coordinate an intrusion-detection system with software decoys and how to improve the performance of software decoy application with an intrusion-detection system. Our research is based on familiar defense concepts like defense-in-depth, intrusion-detection architecture, secure log files, data reduction, the principle of least privilege, and reference monitor. In Chapter II we will define these concepts.

Chapter III will propose a design that is capable of integrating messages read from log files, alerts read from the intrusion-detection alert file, and messages read from all devices installed to protect our system. It borrows some principles from real war strategies, including integration of a diversity of resources to provide more flexibility.

Chapters IV, V and VI will show the experimental design architecture, the data collected from this experimentation and the analysis of the gathered information. As we built the design based on open-source applications, we developed installation guidelines, which are included in the appendices, answering one of the most important objections to open-source: the lack of good documentation.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND AND DEFINITIONS

A. SOFTWARE DECOYS

Since in ancient times Sun Tzu [2] wrote that “all warfare is based on deception”, deception has a very important role in warfare. Dunnigan started his book [3] saying, “The most potent weapon in any soldier’s arsenal is deception.” Deception is an art supported by technology that, when successful, can have a devastating impact on victims. Cohen [4] suggests two way of defeating an enemy in attacks on computers ("information warfare"): have an overwhelming force of some sort (be faster, smarter, better prepared, better supplied, first to strike, better positioned, and so forth), or manipulate the enemy into reduced effectiveness by inducing misperceptions that cause the enemy to misuse their capabilities (i.e., use deception).

In conventional war, the nine main deception types defined by Dunnigan are: concealment, camouflage, ruses, demonstrations, feints, false and planted information, lies, displays and insight. From this list, Neil Rowe [5] explained that only the last 3, lies, displays and insight, are potential defensive tactics for cyberspace. Appropriate deceptive tactics depend on the value of the resources being protected and the danger of the attack [6]. However, the general idea is to limit or confine attacks that get through the first line of defense rather than stop them. Deception differs from honeypots [7] by providing defense not data.

Deception could be one more layer in our defense-in-depth, thus confusing an attack plan for a while. Clearly, in the nanosecond computer world, minutes can be a long period of time. Delays give time to win race conditions against the attacker's automatic tools, permitting the analysis of the attack and a plan to respond.

Before responding to an attacker or outsmarting them, the system must detect the attacker. Thus deception capabilities must be integrated in the defense operations. This would start with a monitoring system, which is described in the next item.

Thinking about the relationship between time and defense, Winn Schwartau in [8] suggested a formula (P > D + R) for a security model. He says that if the time value afforded a system by protection (P) is greater than the amount of time it takes to detect (D) and respond (R) to an attack then a secure environment is evident. A system with deception (C) suggests a new formula: P + C > D + R. This gives us a quantitative justification for deception.

Intelligent software decoys [9], has both a protection and counterintelligence component. The decoy consists of one or more software wrappers placed around a unit of software (*e.g.*, component or method), with each wrapper consisting of a set of rules for detecting and responding to suspicious behavior. Instead of indicating to the attacker that he has been detected, the decoy keeps the attacker occupied by creating the illusion for the attacker that the attack is progressing as expected, using techniques ranging from fake error messages to redirecting the interaction with the attacking computer process to a virtual sandbox.

The goal is threefold: to gather information about the nature of the attack, adjust the system’s defenses based on the intelligence information, and cause the attacker to experience an opportunity cost (*e.g.*, waste attack resources that could have been better applied, or expose sources and methods).

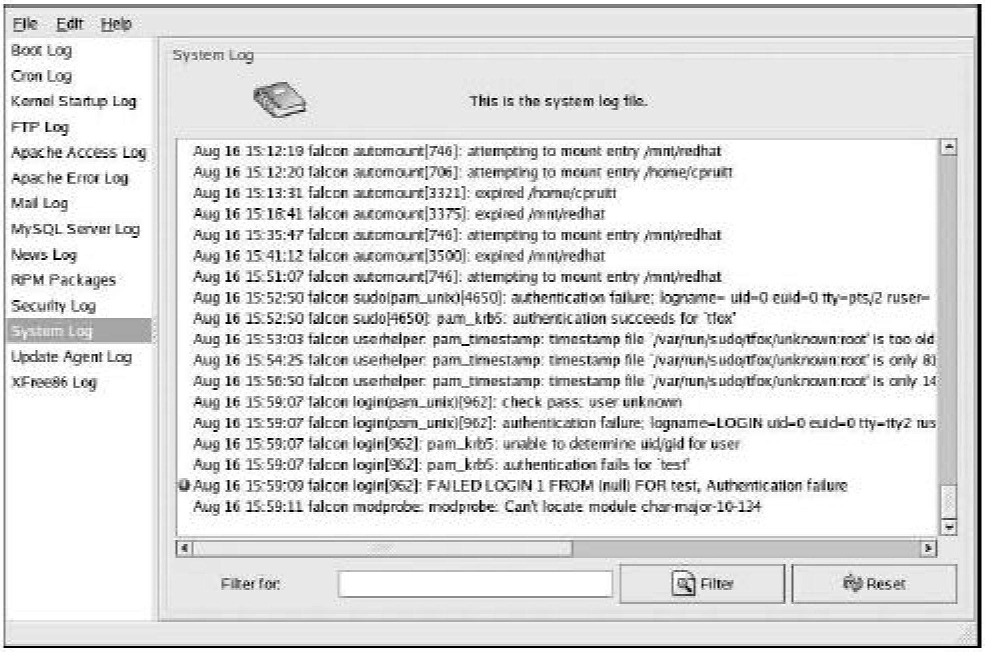
B. LOG FILES

Log files and intrusion-detection systems provide our monitoring mechanisms for a computer operating system. Log files are defined as files that contain messages about the system, including the kernel, services, and applications running on it [10]. Today all operating systems, applications, and network devices have the capability to log information and events that occurred in their environment. Thomas A. Wall wrote in [11] that the better the log stream, the pattern library, and the analysis tools, the better the overall security. He also defines two goals of a monitoring system: reducing the likelihood of an attack going unlogged to as close to zero as is affordable, and increasing the likelihood that the events logged for an attack will be recognized as an attack to as close to 100 percent as is affordable. He also discusses the shape of a logging system, the areas to be logged, the logging mechanisms, the logging system design, log management, and log analysis.

In an ideal network, the system log records every event. This approach is technically very difficult, as few systems have the resources to store all this information. Another difficulty is the human incapacity to check thousands of lines of information of many log files to figure out what is happening. Therefore the log system is configured to reduce this ocean of information by recording only events necessary to detect known common attack patterns, events necessary to detect unusual patterns of access, and information about the continued trustworthiness of the logging system.

Different log files keep different information. For example, there is a default system log file, a log file for security messages, and a log file for kernel events. Some log files are controlled by a daemon called s*yslogd*. In our experimental design based on the Linux platform we will use *syslog*. A list of log messages maintained by *syslogd* can be found in the */etc/syslog.conf* configuration file. *Syslog* is the primary logging mechanism for most Internet-related equipment and the most common network logging mechanism in the TCP/IP world. Syslog runs on all Unix/LINUX systems, and on many other operating systems, including Windows platforms that have adaptors.

The log files that we used in our experiments are in the */var/log* directory. Most log files are in plain text format. In our experiments, we will use a Perl script to read this text and do data reduction. In addition we can use any text editor or *logviewer* to inspect the information as shown in *Figure 1* [10].



```
Aug 16 15:12:19 falcon automount[746]: attempting to mount entry /mnt/redhat
Aug 16 15:12:20 falcon automount[706]: attempting to mount entry /home/cpruitt
Aug 16 15:13:31 falcon automount[3321]: expired /home/cpruitt
Aug 16 15:16:41 falcon automount[3375]: expired /mnt/redhat
Aug 16 15:35:47 falcon automount[746]: attempting to mount entry /mnt/redhat
Aug 16 15:41:12 falcon automount[3500]: expired /mnt/redhat
Aug 16 15:51:07 falcon automount[746]: attempting to mount entry /mnt/redhat
Aug 16 15:52:50 falcon sudo(pam_unix)[4650]: authentication failure; logname= uid=0 euid=0 tty=pts/2 ruser=
Aug 16 15:52:50 falcon sudo[4650]: pam_krb5: authentication succeeds for `tfox'
Aug 16 15:53:03 falcon userhelper: pam_timestamp: timestamp file `/var/run/sudo/tfox/unknown:root' is too old
Aug 16 15:54:25 falcon userhelper: pam_timestamp: timestamp file `/var/run/sudo/tfox/unknown:root' is only 80
Aug 16 15:56:50 falcon userhelper: pam_timestamp: timestamp file `/var/run/sudo/tfox/unknown:root' is only 14
Aug 16 15:59:07 falcon login(pam_unix)[962]: check pass; user unknown
Aug 16 15:59:07 falcon login(pam_unix)[962]: authentication failure; logname=LOGIN uid=0 euid=0 tty=tty2 rus
Aug 16 15:59:07 falcon login[962]: pam_krb5: unable to determine uid/gid for user
Aug 16 15:59:07 falcon login[962]: pam_krb5: authentication fails for `test'
Aug 16 15:59:09 falcon login[962]: FAILED LOGIN 1 FROM (null) FOR test, Authentication failure
Aug 16 15:59:11 falcon modprobe: modprobe: Can't locate module char-major-10-134
```

Figure 1. Log File Monitor Red Hat 8.0

C. PERL SCRIPT – THE LOG MONITOR

A log monitor is a process, or daemon, which monitors log messages produced by the computer system and programs running on it [12]. A properly designed log monitor can recognize unusual activity (or inactivity), alert administrators to problems, gather statistics about system activity, or as in our research, act as the main source for the system to take automatic action against a threat.

A log monitor is an agent, which responds automatically to conditions revealed by one or more system log messages. The response may consist of autonomous actions to handle a situation and/or notification of a human administrator. A stateful log monitor is one that infers the presence of a condition requiring attention by compiling data from more than one log message. Our experiments used a stateful log monitor implemented by a Perl script.

Log monitoring requires string manipulation, and Perl has features that make it one of the most powerful languages for string manipulation. Lutz Prechelt [13] tested 80 implementations of the same set of requirements and compared some properties, such as run time, memory consumption, source text length, and the amount of effort required to write them. The results indicate that for the given programming problem, which regards string manipulation and searches, a “script language” such as Perl is more productive than “conventional languages” such as C, C++ and Java. In terms of run time and memory consumption, “script languages” often turned out better than Java and not much worse than C and C++.

For future implementations of deception capabilities, Perl is also flexible in implementing rules, reading configuration files, reading streams from networks, implementing servers and sockets and manipulating a system’s call. Others Perl advantages include:

- Provides features necessary for large projects like modularization and object-oriented techniques.
- Provides great flexibility for manipulation of strings using regular expressions.

- Allows the use of all system calls including those necessary for network tasks.
- Provides a way to dynamically load a module including code written in C.

D. INTRUSION-DETECTION SYSTEMS

Intrusion-detection systems (IDSs) are important software tools. [14] and [15] provide some background. Some useful definitions:

- Intrusion: Any set of actions that attempt to compromise the integrity, confidentiality, or availability of a computer resource [14].
- Intrusion detection: The problem of identifying actions that attempt to compromise the integrity, confidentiality, or availability of a computer resource [14].
- Monitored system or system: Program, application, host or a network of computer resources that is being monitored [14].
- Intrusion-detection systems: Systems that collect information from a variety of system and network sources and, then, analyze the information for signs of intrusion (attacks coming from outside the organization) and misuse (attacks originating inside the organization) [16].

Crosbie and Spafford in [17] identified desirable characteristics of an intrusion-detection system:

- It must run continually with no human supervision.
- It must be fault tolerant.
- It must resist subversion and monitor itself.
- It must impose a minimal overhead on the systems where it runs.
- It must be as “quiet” as possible, precluding professional attackers from realizing that they are being monitored.
- It must be configurable and expectantly adaptable to changes in the system and to user behavior over time.
- It must be able to detect unknown attacks as much as possible without generating a lot of false positive and false negative.
- It must be able to avoid the situation of being used as a denial of service mechanism.
- It must report and launch automated decoy capabilities as soon as possible after an intrusion or an attack detection.

Intrusion-detection systems can be classified by their data collection mechanisms. We can classify IDSs as direct or indirect [14]. Indirect IDSs could be sub-classified into network-based or host-based with direct data collection mechanisms being sub-classified into internal or external. Table 1 clarifies these definitions.

Data Collection Mechanisms			
Direct		**Indirect**	
Host-based		**Host-based**	**Network-based**
Internal	**External**		

Table 1. Data Collection Mechanisms

Internal IDSs are those whose code is incorporated in the monitored system. We will use both, internal and external, in our experimental design.

Indirect monitoring is the observation of a component through a separate mechanism or tool. Direct monitoring is better than indirect for many reasons [14]. An intruder could potentially alter data from an indirect data source before the log monitor uses it, or it could be affected by non-malicious failures. But the majority of IDSs use some form of indirect monitoring. In our experimental design we will use both direct and indirect.

Network-based IDSs (NIDSs) is the acquisition of data from the network, usually done by capturing packets as they flow through it. Host-based IDSs (HIDSs) process data that originates in computers such as event log files. Network-based IDSs capture and analyze TCP/IP packets, and host-based IDSs process event logs from operating systems, kernels and applications. In our experimental design, we will use both a NIDS and an HIDS. Table 2 [15] summarizes the advantages of NIDSs and HIDSs.

	Network-based IDS	**Host-based IDS**
ADVANTAGES	• Can watch the whole network or any subsets of the network from one location. • Can monitor and detect network attacks (e.g., probes, scans, malicious and anomalous activity across the whole network. • Can become “invisible” for access.	• Can prevent and log abuse of privilege attacks. • Can detect elevated privileges attacks. • Can detect for critical data access and modification.
DISADVANTAGES	• Can not detect host activity. • Can not scan protocols or content if network traffic is encrypted. • Can cause monitoring and detecting to become more difficult on modern switched network. • Can lose some packet when working in high-speed network.	• Can not trace network activity. • Can only work on specific platform. • Can interfere with implemented service activities running in the host. • Can not totally trust the host information, once the machine is compromised.

Table 2.   HIDS and NIDS Comparison

An increasing number of hybrid IDSs use both HIDS and NIDS components to augment the information collected and to better analyze it. Such hybrids are better able to provide tamper-proof operation. If an attacker tries to use the network to launch the attack, they would be monitored; if they launched an attack from the machine, they would also be monitored.

E. LIDS (LINUX INTRUSION-DETECTION SYSTEM)

In our experiments we used LIDS [18] as our HIDS. LIDS provides protection to file and running processes and uses a security kernel. Additionally, LIDS has a built-in portscan detector, which can be used to alert users to the warning signs of a possible intruder, and can send e-mail to the network administrator when a rule is broken. These features could be considered response mechanisms and could be used to launch decoy capabilities.

Besides this, the most important feature of LIDS is its implementation of the reference monitor concept. A reference monitor [19] is an abstraction that allows active entities called subjects to make reference to passive entities called objects, based on a set of current access authorizations. Subjects are processes executing in a particular domain in a computer system. A domain of a process is defined as the set of objects which the process currently has the right to access according to each access mode.

As described in [20], a security kernel is the only method proven to be effective at countering the threats of penetration and subversion of mechanism; therefore it is the only effective method of preventing illicit access to information under protection. A security kernel is defined as the hardware and software that implements a reference monitor. Files, records, and other types of information repositories can be built from primitive objects (read and write), but access control is provided by the reference monitor on the basis of these primitive objects over which it has total control. With regard to information warfare in particular, every security feature must itself be protected so that they can detect and respond. This requires a kernel.

Previous work of our project implemented a deceptive component [21] with security based on kernel modules. LIDS works in the same way by improving Linux security at kernel level. The main advantage of this solution over NAI Wrappers [22] is that LIDS is more comprehensive in its kernel capabilities. In LIDS, the "root" (system administrator) is no longer all-powerful. Some files, directories, and processes protected by LIDS cannot be modified even with the root password. The advantage is that even if a vulnerability were found in a program that is running with root privilege, the damage of its exploitation would be limited.

F. SNORT – A NETWORK INTRUSION-DETECTION SYSTEM

Snort is an open-source free NIDS developed by Martin Roesch [23]. In early 2002, Snort was downloaded over 10,000 times a week to protect government, corporate, home, and education sites. Snort is small at 1.8 Mbytes in the last version (1.9), and extremely configurable, allowing users to create their own rules or even reconfigure its base functionality though its plug-in interface. The schema in Figure 2 shows how Snort works [24].

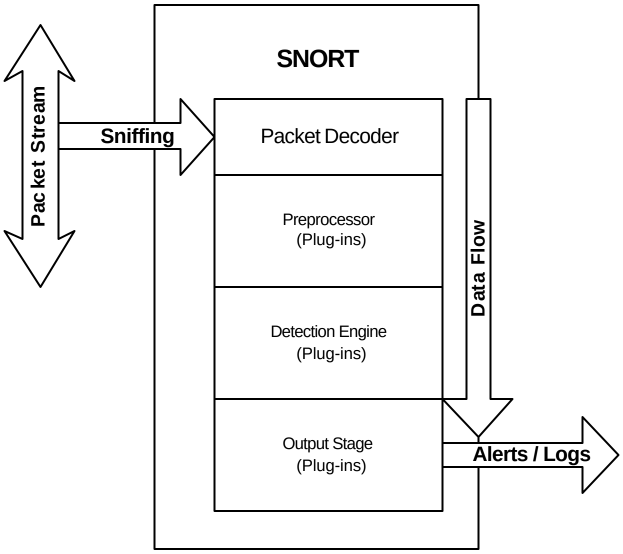


Figure 2. Snort principle of function

Figure 2 shows how the packet is sniffed off the network interface and passed to the packet decoder where it is partitioned into its layers. If any preprocessors have been defined, they act upon the packet. Preprocessors allow Snort to examine and manipulate network traffic data in several useful ways, such as in IP defragmentation, TCP stream assembly, portscan detection, and web-traffic normalization. The preprocessor can maintain state over multiple packets to be more intelligent in its processing. Then the packets are passed to the detection engine where rules examine the processed packet. Snort's official documentation [25] gives details of rule capabilities. If an alert or logging is triggered by the rules, the packet is passed to the output preprocessor for appropriate processing.

There are three basic modes of operation in Snort: sniffer, packet logger, and NIDS. Each is well suited for a particular traffic analysis task. In our experimental design we used Snort in network mode, where it was loaded with a configuration file (snort.conf) containing run-time directives and rules. Also, a Snort packet can be either binary or plain text. For speed and portability, it is best to log to a binary-format file, although we used plain text in our experiments.

G. RESPONSE MECHANISMS OR COUNTERMEASURES

The concept of active response mechanisms or countermeasures in an IDS [26] is a form of the idea of having the IDS capable of automatic reactions to threats. The goal is to prevent further compromise between the attacker and the attacked machine. The following techniques can be used:

- RST emission
- Firewall update
- Routing table update
- Signature-based firewall

An IDS has two parts, data collection and attack response. Data collection is done by sensors that are usually self-contained detection engines, which obtain network packets, search for patterns of misuse, and then report alarms to a data-analysis central command. This approach does have some problems, however, such as difficulty in recognizing denial of service attacks and the creation of race conditions between a packet generated by IDS and the packets send by an attacker [27].

Automated response to intrusions has become a major issue in defending critical systems. Since the adversary acts at computer speeds, systems need the capability to react without human intervention. An infrastructure that supports the development of automated response must allow easy integration of detection and response components to enable experimentation with automated response strategies.

THIS PAGE INTENTIONALLY LEFT BLANK

III. IDS AND SOFTWARE DECOY ARCHITECTURE

A goal of our research is to show that the integration of a NIDS and a HIDS could facilitate deployment of a software decoy. But when we started our research we were not sure how modular NIDSs and HIDSs could be, which is a necessity when integrating them with additional complex modules for deception. We discovered one simple interface that allowed for considerable modularity was to process log files created by the NIDS and HIDS, and make this the input to the deception processing.

A. INTRUSION-DETECTION INTEGRATION

The experimental design contains two types of intrusion detection, NIDS (Snort) and HIDS (LIDS). Another module is a log monitor that collects information from NIDS, HIDS, and the kernel log file, and then does data reduction. Figure 3 illustrates this. As discussed in Chapter II, good IDSs should provide internal direct data collection mechanisms such as those provided by a HIDS. Nonetheless, a NIDS with its indirect collection is valuable as it can anticipate the attack and the reaction. Most attacks start with a probe and scanning and, as seen in Table 2, these can only be caught by a NIDS. Furthermore, NIDS can see malicious network behavior in a variety of forms that a HIDS cannot see. A HIDS can see a portscan, but a NIDS can see the similar attacks on other sites that happened first.

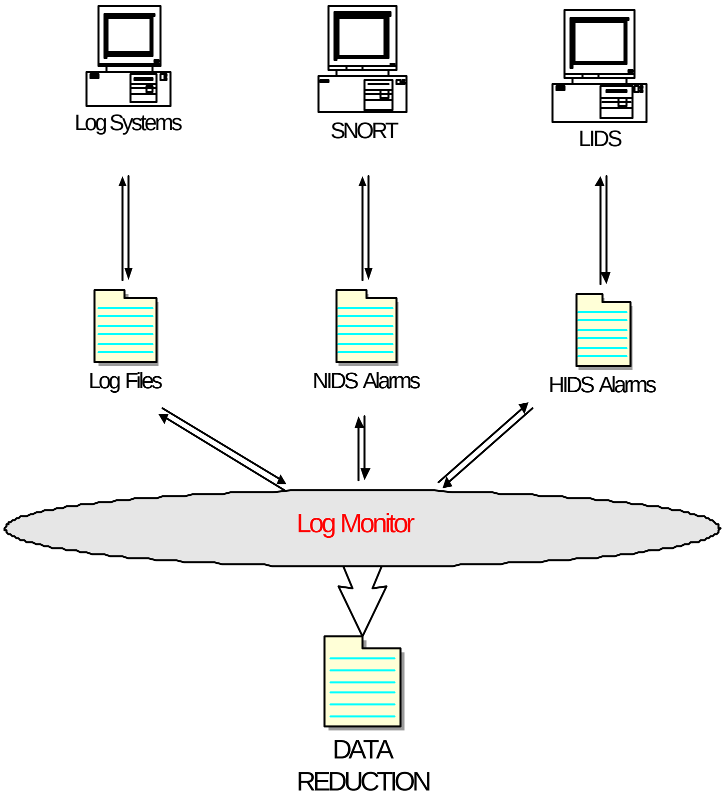


Figure 3. Architecture schema

B. IDS AND SOFTWARE DECOY ARCHITECTURE INTEGRATION

1. How NIDS Can Improve a Software Decoy

Georgios Fragkos [21] selected an exemplar of attack and created a deception for it by using NAI’s Generic Software Wrapper Toolkit [22] to do both detection and decoying. But his deception ends when the attacker tries to interact with the shell since the shell's functionality is not being simulated and the attacker will immediately discover that something went wrong. Since it is thought that professional attacks always will use automatic tools as rootkits, this kind of decoy may only be effective for a few seconds. An alternative could be to transfer the attacker to a safer machine where everything is simulated, such as a honeypot or sandbox, where it would be hard for the attacker to tell that he is being fooled.

NIDS can anticipate the attack, thus improving the performance of the software decoy application. For instance, after NIDS detects a ping it could lie to the attacker and send an ICMP (Internet Control Message Protocol) message saying “host unreachable”, or the software decoy could delay the ping response. Or NIDS could incorrectly inform the attacker as to which ports are open and what vulnerabilities they have.

2. How HIDS Can Improve Software Decoy

After the attacker has gained some privileges on the target machine, data collected from the logs would decrease in importance because the network information could no longer be trusted. The attacker could also launch some cryptographic channels to communicate with the outside world, making the network analysis more difficult. The system developed in our experiments tried to deal with this situation by creating one more layer of defense and one more chance to fool the attacker, the layer above the root discussed in Chapter II (see Figure 4).

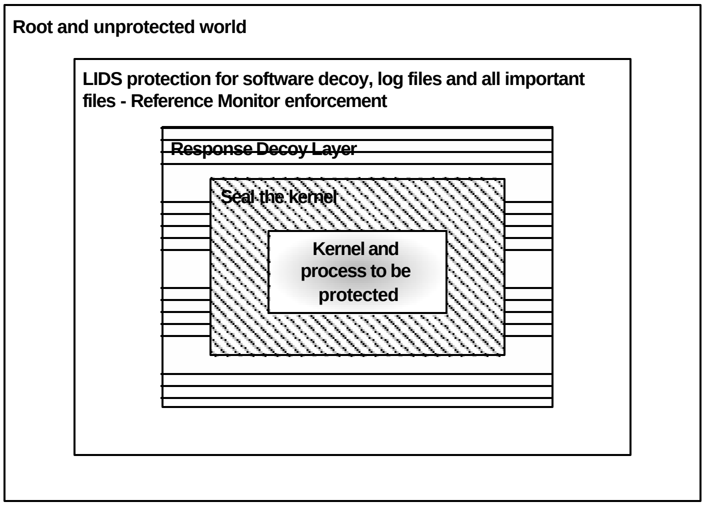

Root and unprotected world
LIDS protection for software decoy, log files and all important files - Reference Monitor enforcement
Response Decoy Layer
Seal the kernel
Kernel and process to be protected

Figure 4. Inner layers of defense-in-depth

As seen in Figure 4 above, the root is no logger the last level to be attacked to reach the control of the host. Even if the attacker gains root privileges, we keep the kernel sealed, the process and services untouchable, and the decoy programs running. This design minimizes the threat to the target system and provides more time for launching decoy capabilities against the enemy, which increases the deception factor in Schwartau formula [8].

3. Data Reduction

Few systems have the resources to store all information generated by various log files. In a medium network the hard disk could be filled by log file information in a few weeks. If the log systems are concentrated in one machine or file, this situation is much worse. Not concentrating all information in one system or file is not reasonable either, as this situation exacerbates the human incapacity to check thousands of lines of information of many log files to figure out what is happening. To make a realistic analysis of the information collected by some log files, the system administrator would have to check each line of each log files, comparing timestamps of the events generated by the all log files or alert files.

Therefore, the best solution is implementation of a log monitor to reduce this ocean of information by, based on a security policy, recording only those events necessary to detect attack patterns and events that are suspicious. The other advantage is that the log monitor works as an interface between the detection and the response. To do this work we used Perl as best explained in Section II-c.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. EXPERIMENTAL DESIGN IMPLEMENTATION

We now describe in more detail an implementation of intrusion-detection systems to support software decoys. The next sections will more thoroughly describe the implementation of each module in Table 3.

Modules	
Type	**Description**
Defense	(a) SNORT as NIDS (b) LIDS as HIDS (c) Kernel Log File
Log Monitor	(d) Perl script
Decoy Mechanisms	(e) Network decoy (f) Host decoy

Table 3. Experimental Design Module Implementation

The modules were implemented for a Linux Red Hat8.0 platform. This platform was chosen because the source code is available, which facilitates instrumentation of the kernel. This feature is very important for the implementation of LIDS (HIDS) as it works as a patch of a “pure” Linux kernel, mandatory in the recompilation of the kernel. Another advantage is that the kernel instrumentation capacity can facilitate future work as the software decoy implementation can also be embedded in the operating system kernel.

A. EXPERIMENTS

As a proof of concept, we tested both a network-based attack and a host-based attack in experiments. Figure 3 shows what was implemented. We installed two services in a protected host: a Web service (Apache2.0) and a SSH service. The SSH server is part of Red Hat 8.0. In the installation of both services the default configuration was used. The simulated attacks tried to break into the protected host to exploit vulnerabilities of a Web and SSH service.

Most attacks are initiated with footprinting, followed by probes and scans. These first steps have the main goal of discovering open ports and known vulnerabilities thus finding the best way to break into the target host [32]. In the first part of an attack the invader bases his action on the network environment, so NIDS could better monitor the invader's actions. In the second part of the attack, after the invader had obtained some privileges, the kernel Log Files and HIDS alerts become essential for attack detection and analysis.

As described in earlier sections, the Perl script (Appendix A) reads the alert file generated by Snort, reads the log file generated by Linux kernel and reads the log file generated by LIDS, writing in a file called *decoy.log* information about the security policy (Table 4), timestamp and IP address. This experiment had the rules shown in Table 4 for its security policy.

Attacker	Detection by	Ploy	Defense Decoy System Reaction
PING	NIDS	HONESTY	(a) Log monitor records the action to a data reduction file; (b) Starts a program to simulate false ports; (c) Delays the response.
SCAN	NIDS and HIDS	LYING	(d) Log monitor records the action to a data reduction file; (e) Responds with false ports as open; (f) Changes the configuration of the border Cisco router, redirecting the attacker to a fake server.
CONNECT (LOGIN)	NIDS and Kernel Log File	LYING	(g) Log monitor records the action to a data reduction file; (h) Launches fake xterm.
BAD ACTIONS	HIDS	LYING	(i) Log monitor records the action to a data reduction file; (j) Reports login.

Table 4. Experiment Security Policy

1. PING

Following Table 4, we prepared our environment to detect any kind of "ping" (attempt to query the status of our protected machine). From an attacker machine (192.168.0.1), we started to ping the protected host (192.168.0.3).

- **Snort:** We wrote a rule to detect any ICMP (Internet Control Message Protocol) packet that has a destination of the protected host:

```
Alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:”ICMP
Packet to Protected Host”; classtype:bad-unknown;)
```

To do this, in the snort.conf (see Appendix C) we defined $EXTERNAL_NET as any IP number but the protected host was assigned IP number 192.168.0.3.

The $HOME_NET is the protected host [25]. This rule generated the following message at the alert file at /var/log/snort directory for each ping:

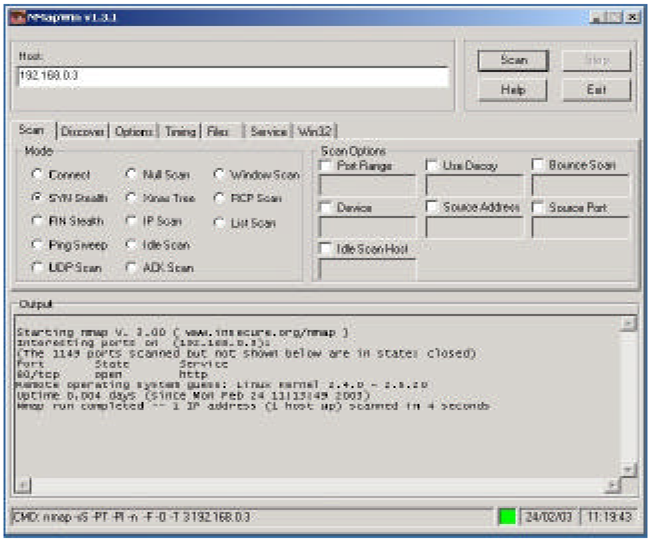
```
02/23-15:16:33.464600   [**]   [1:0:0]   ICMP   Packet   [**]
[Classification: Potentially Bad Traffic] [Priority: 2] {ICMP}
192.168.0.1 -> 192.168.0.3
```

- **LIDS:** Not involved.
- **Kernel Log File:** Not involved.
- **Log Monitor:** Read the alert file, searching for a /PING/ pattern. After matching the PING string at alert file, the Log monitor recorded in decoy.log:

```
HONESTY:  hacker(192.168.0.1)  PING  DEST:192.168.0.3  at  02/23
15:16:33.464600 2003.
```

2. Scan

Using Win?Map, a well-known scanner, from 192.168.0.1 (the attacker machine), we scanned the target protected host (192.168.0.3) (see Figure 5).



Host: 192.168.0.3

```
Starting nmap V. 3.00 ( www.insecure.org/nmap )
Interesting ports on  (192.168.0.3):
(The 1149 ports scanned but not shown below are in state: closed)
Port       State       Service
80/tcp     open        http
Remote operating system guess: Linux Kernel 2.4.0 - 2.5.20
Uptime 0.004 days (Since Mon Feb 24 11:13:49 2003)
Nmap run completed -- 1 IP address (1 host up) scanned in 4 seconds
```

CMD: nmap -sS -PT -PI -n -F -O -T 3 192.168.0.3 | 🟩 | 24/02/03 | 11:19:43

Figure 5. Win?Map application scanning the target machine

- **Snort**: The scan is logged by the Snort portscan preprocessor [30]. It records an alert in the scan.log file and in the alert log file, both in the */var/log/snort* directory. Stream4, another preprocessor, also logged this activity in the */var/log/snort/alert* file. This is the output of that file.

```
02/24-11:27:29.905403  [**] [117:1:1] (spp_portscan2) Portscan
detected from 192.168.0.1: 1 targets 21 ports in 0 seconds [**]
{TCP} 192.168.0.1:60517 -> 192.168.0.3:506

02/24-11:27:31.586498   [**]  [111:9:1]  (spp_stream4)  STEALTH
ACTIVITY (NULL  scan) detection [**] {TCP} 192.168.0.1:60525  ->
192.168.0.3:80
```

The portscan preprocessor is a powerful and flexible scan detector. It checks for TCP connection attempts to more then P ports in T seconds, and UDP packets sent to more than P ports in T seconds [25], where P and T are given in the *snort.conf* file. This portscan can also detect a single “stealth scan” packet as in NUL, FIN, SYNFIN, and XMAS scans. Another benefit of portscan is that alerts only showed once per scan, rather than once per packet, which reduces the amount of redundant information in the alert file.

The stream4 preprocessor provides TCP stream reassembly and stateful analysis capabilities for Snort. Stream4 also gives users the ability to track more than 256 simultaneous TCP streams. Stream4 should be able to scale to handle 32,768 simultaneous TCP connections in its default configuration.

- **LIDS: LIDS can also detect port scans w**hen its optional port scan detector is enabled. Here is an example from its message log file.

```
Feb 24 11:19:39 LIDS kernel: LIDS: (undetermined program) pid 0
ppid 0 uid/gid (0/0) on (null tty): Port scan detected:
192.168.0.1 scanned 1153 closed ports including 575 ports < 1024)
```

- **Kernel Log File:** Not involved.
- **Log Monitor**: Read the alert file generated by Snort, searching for /spp_portscan2/ or /spp_stream4/ patterns. After matching one of these patterns at alert file, the log monitor recorded in the *decoy.log* file:

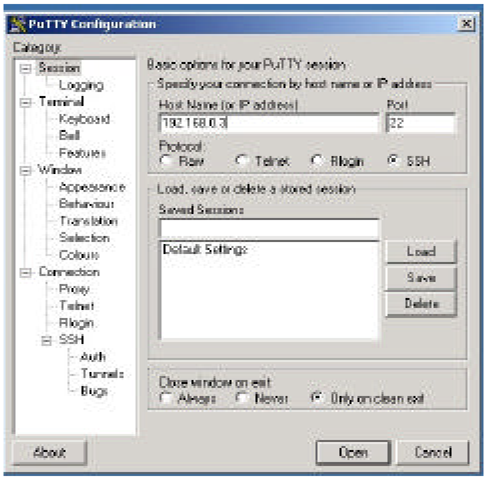
```
LYING:  hacker(192.168.0.1)  SCAN  DEST:192.168.0.3  at  02/24-
11:27:29.905403 2003.
```

The Log monitor also read the kernel message log file generated by LIDS and generated the following record in *decoy.log* file:

```
LYING:  hacker(192.168.0.1)  SCAN  DEST:192.168.0.3  at  Feb  24
11:19:39 2003.
```

3. Connect

In our scenario, see Table 4, after a ping and a scan the attacker tries using the PuTTY application (Figure 6) to connect to the target machine at port 22 (ssh).



Basic options for your PuTTY session

Specify your connection by host name or IP address

Host Name (or IP address): 192.168.0.3

Port: 22

Protocol: ○ Raw ○ Telnet ○ Rlogin ● SSH

Load, save or delete a stored session

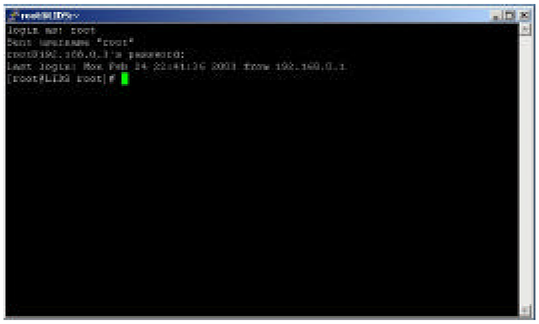
Saved Sessions:

Default Settings

Close window on exit: ○ Always ○ Never ● Only on clean exit

Figure 6. SSH connection from attacker machine

After that, for experimental purposes only, the attacker logs in as root (Figure 7), simulating that the root password was previously known.



```
login as: root
Sent username "root"
root@192.168.0.3's password:
Last login: Mon Feb 24 22:41:36 2003 from 192.168.0.1
[root@LIDS root]#
```

Figure 7. Attacker login

- **Snort:** Snort was configured to detect any attempt to telnet or to connect using the protocol SSH against the protected host. These are the rules:

```
alert tcp $EXTERNAL_NET any -> $TELNET_SERVERS 23 (msg:"TELNET
attempt"; flow:to_server,established; classtype:shellcode-detect;
sid:1430; rev:5;)

alert  tcp  $EXTERNAL_NET  any  ->  $SSH_SERVERS  22  (msg:"SSH
attempt"; flags:S+; classtype:suspicious-login; sid:1431; rev:5;)
```

These rule configurations use the variable $EXTERNAL_NET defined in snort.conf as being any host but the protected host (192.168.0.3). Both TELNET_SERVER and SSH_SERVER are defined as being the protected target host, 192.168.0.3.

Due to the encrypted nature of a SSH connection, detecting such an attacker's attempt to scale privileges is impossible. But we can detect whether encrypted traffic was employed to communicate with the protected host. The following message was logged in the alert file at */var/log/snort* directory:

```
02/24-22:47:10.107040     [**]   [1:1431:5]   SSH   attempt   [**]
[Classification: An attempted login using a suspicious username
was   detected]   [Priority:   2]   {TCP}   192.168.0.1:1884   ->
192.168.0.3:22
```

- **LIDS:** As a SSH is a service available by the protected server, LIDS is not involved.
- **Kernel Log File:** A message to record the SSH connection was recorded in the kernel log file:

```
Feb 24 21:23:01 LIDS sshd(pam_unix)[684]: session opened for user
root by (uid=0)
```

This is misleading because we called the target machine LIDS. The log message above was recorded by the kernel and not by the HIDS called LIDS.

- **Log Monitor**: Read the alert file, searching for an attempt to CONNECT record. After matching the SSH connection recorded by Snort, the log monitor was recorded in *decoy.log* file:

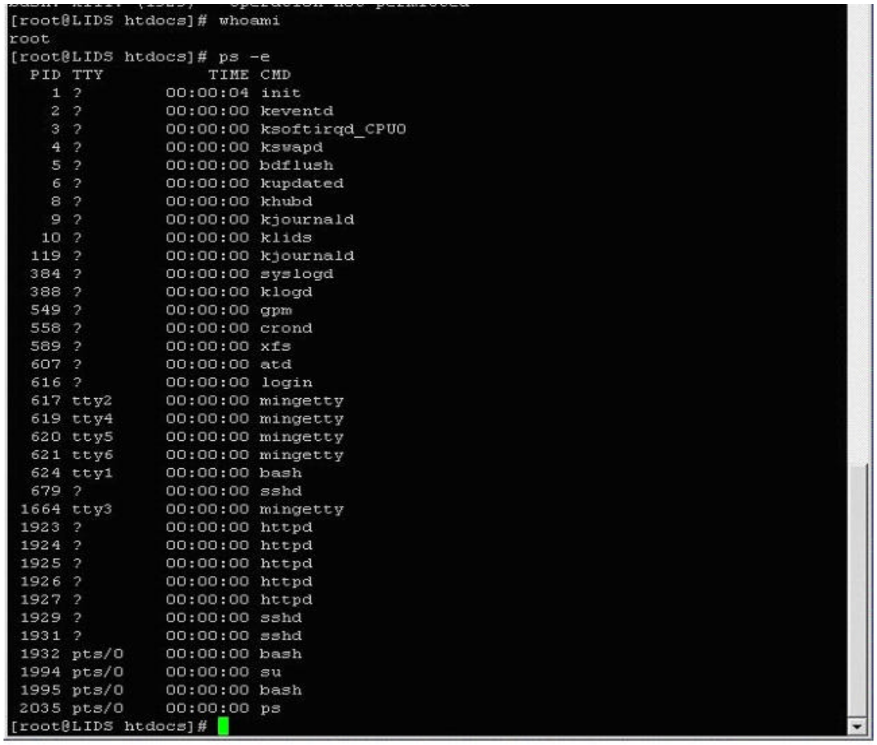
```
LYING: hacker(192.168.0.1) CONNECT SSH DEST:192.168.0.3 at Sat
Feb 21 15:16:33 2003.
```

The log monitor also read the kernel message log file generated by the kernel log and generated the following record in *decoy.log* file:

```
LYING: hacker(192.168.0.1) CONNECT SSH DEST:192.168.0.3 at Feb 21
15:16:33 2003.
```

4. Other Suspicious Actions at Target Machine

Using the PuTTY application, we logged on to the target machine and tried to execute some forbidden actions such as copying files, decompressing files, and killing processes (Figure 8). The rules for this are defined in Appendix B.



```
[root@LIDS htdocs]# whoami
root
[root@LIDS htdocs]# ps -e
  PID TTY          TIME CMD
    1 ?        00:00:04 init
    2 ?        00:00:00 keventd
    3 ?        00:00:00 ksoftirqd_CPU0
    4 ?        00:00:00 kswapd
    5 ?        00:00:00 bdflush
    6 ?        00:00:00 kupdated
    8 ?        00:00:00 khubd
    9 ?        00:00:00 kjournald
   10 ?        00:00:00 klids
  119 ?        00:00:00 kjournald
  384 ?        00:00:00 syslogd
  388 ?        00:00:00 klogd
  549 ?        00:00:00 gpm
  558 ?        00:00:00 crond
  589 ?        00:00:00 xfs
  607 ?        00:00:00 atd
  616 ?        00:00:00 login
  617 tty2     00:00:00 mingetty
  619 tty4     00:00:00 mingetty
  620 tty5     00:00:00 mingetty
  621 tty6     00:00:00 mingetty
  624 tty1     00:00:00 bash
  679 ?        00:00:00 sshd
 1664 tty3     00:00:00 mingetty
 1923 ?        00:00:00 httpd
 1924 ?        00:00:00 httpd
 1925 ?        00:00:00 httpd
 1926 ?        00:00:00 httpd
 1927 ?        00:00:00 httpd
 1929 ?        00:00:00 sshd
 1931 ?        00:00:00 sshd
 1932 pts/0    00:00:00 bash
 1994 pts/0    00:00:00 su
 1995 pts/0    00:00:00 bash
 2035 pts/0    00:00:00 ps
[root@LIDS htdocs]#
```

Figure 8. Process running at protected machine

In Figure 8, we have some process running on the server as httpd (PID 1923) and sshd (PID 1929). The attacker will try to delete a file and copy a file from one directory to another; after that the intruder will try to kill some process.

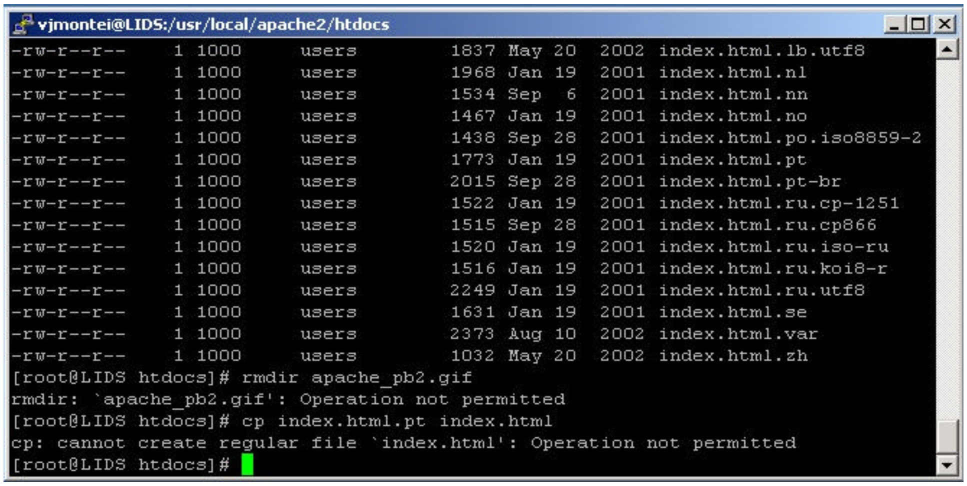
- **Snort:** Not involved.
- **LIDS:** This blocked all of the attacks that the attacker tried to invoke against the target machine. The LIDS configuration needed to accomplish this and allow ssh and the HTTP Server to keep running are described in Appendix B. Following is the output of the message log file logged by LIDS when the attacker

tried to remove and copy a file (Figure 9) to the protected system.

```
Feb 24 23:47:16 LIDS kernel: LIDS: rmdir (dev 3:2 inode 840392)
pid 2031 ppid 1995 uid/gid (0/0) on (pts) : Attempt to rmdir
apache_pb2.gif

Feb 24 23:48:00 LIDS kernel: LIDS: cp (dev 3:2 inode 840382) pid
2032 ppid 1995 uid/gid (0/0) on (pts) : Attempt to open
index.html for writing,flag=32834
```

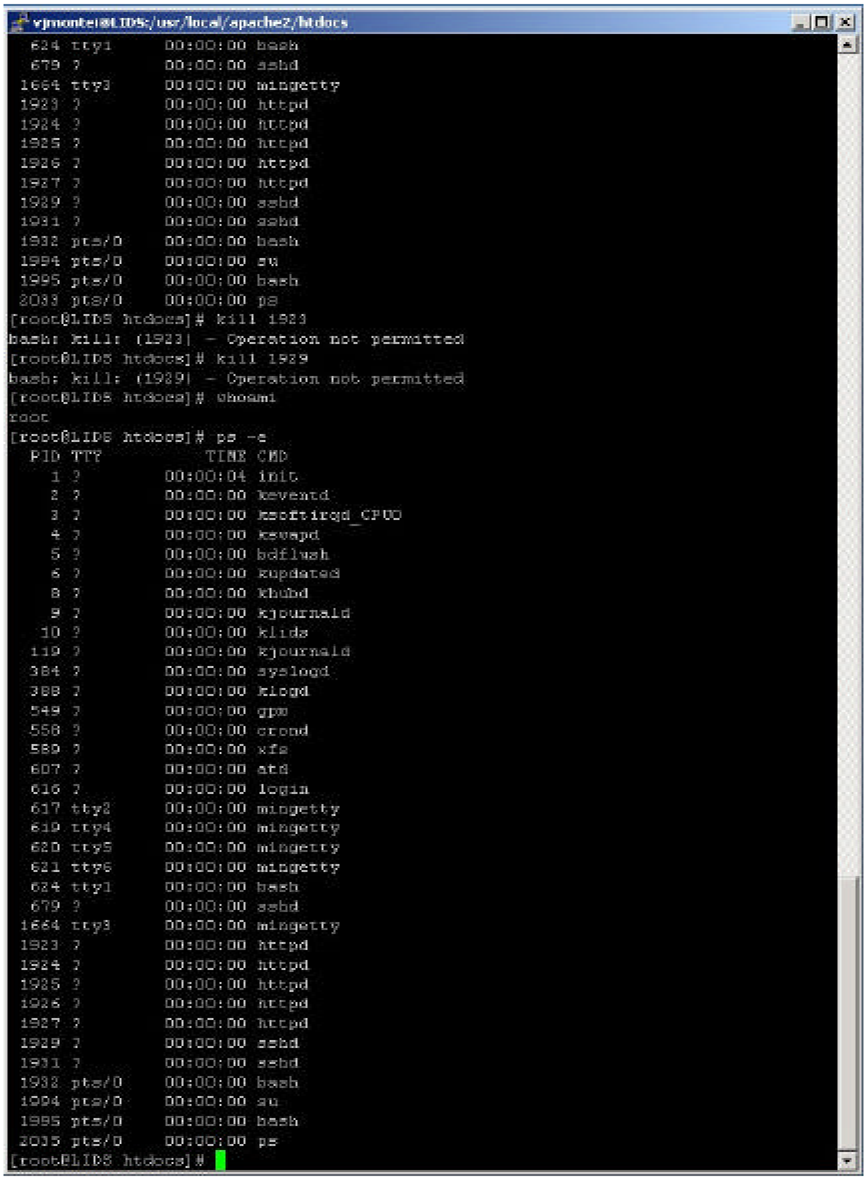
After that, the attacker tried to kill the process httpd. Figure 9 shows the moment that the attacker attempted to check if there was a root privilege; the attacker ultimately realized that the commands, even with root privilege, would not work since the process was still running.



```
-rw-r--r--    1 1000     users        1837 May 20  2002 index.html.lb.utf8
-rw-r--r--    1 1000     users        1968 Jan 19  2001 index.html.nl
-rw-r--r--    1 1000     users        1534 Sep  6  2001 index.html.nn
-rw-r--r--    1 1000     users        1467 Jan 19  2001 index.html.no
-rw-r--r--    1 1000     users        1438 Sep 28  2001 index.html.po.iso8859-2
-rw-r--r--    1 1000     users        1773 Jan 19  2001 index.html.pt
-rw-r--r--    1 1000     users        2015 Sep 28  2001 index.html.pt-br
-rw-r--r--    1 1000     users        1522 Jan 19  2001 index.html.ru.cp-1251
-rw-r--r--    1 1000     users        1515 Sep 28  2001 index.html.ru.cp866
-rw-r--r--    1 1000     users        1520 Jan 19  2001 index.html.ru.iso-ru
-rw-r--r--    1 1000     users        1516 Jan 19  2001 index.html.ru.koi8-r
-rw-r--r--    1 1000     users        2249 Jan 19  2001 index.html.ru.utf8
-rw-r--r--    1 1000     users        1631 Jan 19  2001 index.html.se
-rw-r--r--    1 1000     users        2373 Aug 10  2002 index.html.var
-rw-r--r--    1 1000     users        1032 May 20  2002 index.html.zh
[root@LIDS htdocs]# rmdir apache_pb2.gif
rmdir: `apache_pb2.gif': Operation not permitted
[root@LIDS htdocs]# cp index.html.pt index.html
cp: cannot create regular file `index.html': Operation not permitted
[root@LIDS htdocs]#
```

Figure 9. Attacker deleting and copying files

Next we show what happened when the attacker attempted to terminate process 1929 that was protected by the system (Figure 10).



```
vjmonte1@LIDS:/usr/local/apache2/htdocs
  624 tty1     00:00:00 bash
  679 ?        00:00:00 sshd
 1664 tty3     00:00:00 mingetty
 1923 ?        00:00:00 httpd
 1924 ?        00:00:00 httpd
 1925 ?        00:00:00 httpd
 1926 ?        00:00:00 httpd
 1927 ?        00:00:00 httpd
 1929 ?        00:00:00 sshd
 1931 ?        00:00:00 sshd
 1932 pts/0    00:00:00 bash
 1994 pts/0    00:00:00 su
 1995 pts/0    00:00:00 bash
 2033 pts/0    00:00:00 ps
[root@LIDS htdocs]# kill 1923
bash: kill: (1923) - Operation not permitted
[root@LIDS htdocs]# kill 1929
bash: kill: (1929) - Operation not permitted
[root@LIDS htdocs]# whoami
root
[root@LIDS htdocs]# ps -e
  PID TTY          TIME CMD
    1 ?        00:00:04 init
    2 ?        00:00:00 keventd
    3 ?        00:00:00 ksoftirqd_CPU0
    4 ?        00:00:00 kswapd
    5 ?        00:00:00 bdflush
    6 ?        00:00:00 kupdated
    8 ?        00:00:00 khubd
    9 ?        00:00:00 kjournald
   10 ?        00:00:00 klids
  119 ?        00:00:00 kjournald
  384 ?        00:00:00 syslogd
  388 ?        00:00:00 klogd
  549 ?        00:00:00 gpm
  558 ?        00:00:00 crond
  589 ?        00:00:00 xfs
  607 ?        00:00:00 atd
  616 ?        00:00:00 login
  617 tty2     00:00:00 mingetty
  619 tty4     00:00:00 mingetty
  620 tty5     00:00:00 mingetty
  621 tty6     00:00:00 mingetty
  624 tty1     00:00:00 bash
  679 ?        00:00:00 sshd
 1664 tty3     00:00:00 mingetty
 1923 ?        00:00:00 httpd
 1924 ?        00:00:00 httpd
 1925 ?        00:00:00 httpd
 1926 ?        00:00:00 httpd
 1927 ?        00:00:00 httpd
 1929 ?        00:00:00 sshd
 1931 ?        00:00:00 sshd
 1932 pts/0    00:00:00 bash
 1994 pts/0    00:00:00 su
 1995 pts/0    00:00:00 bash
 2035 pts/0    00:00:00 ps
[root@LIDS htdocs]#
```

Figure 10. Attacker trying to kill httpd process

Here is the output of message log file.

```
Feb 24 23:58:20 LIDS kernel: LIDS: bash (dev 3:2 inode 840369)
pid 1995 ppid 1994 uid/gid (0/0) on (pts) :  violated CAP_KILL

Feb 24 23:58:38 LIDS kernel: LIDS: bash (dev 3:2 inode 840369)
pid 1995 ppid 1994 uid/gid (0/0) on (pts) :  violated
CAP_KILL_PROTECTED  - logging disabled for (60)s

Feb 24 23:58:38 LIDS kernel: LIDS: bash (dev 3:2 inode 840369)
```

```
pid 1995 ppid 1994 uid/gid (0/0) on (pts) : Attempt to kill
pid=1929 with sig=15
```

- **Kernel Log File:** Does not make any kind of record, as the commands are not executed.
- **Log Monitor**: Read the alert file generated by LIDS, searching for patterns that indicate a violation of rules, such as attempting to remove file (/rmdir/), to copy file (/cp/) and kill process (/CAP_KILL_VIOLATION/). After matching the patterns, the log monitor is recorded in ***decoy.log***:

```
LYING: hacker(192.168.0.1) ATTEMPT to RMDIR at Feb 24 23:48:00
2003.
LYING: hacker(192.168.0.1) ATTEMPT to COPY a FILE at Feb 24
23:48:00
LYING: hacker(192.168.0.1) ATTEMPT to KILL PROCESS 1929 at Feb 24
23:58:38 2003.
```

THIS PAGE INTENTIONALLY LEFT BLANK

V. DISCUSSION

Military history suggests it is best to employ a layered, defense-in-depth strategy that includes protection, monitoring, and response [28]. Also, deception should be integrated with operations [29]. This is the strongest point of the architecture developed in the experimental design, which has its structure based on defense-in-depth. We showed that an intrusion-detection system can improve decoy capabilities if the two are integrated together . The detection system cannot be a job of only one machine or technology. Both of these capabilities have to be spread around the defensive structure.

As we do not have, at the present date, a defined decoy policy, we created one (Table 4) based on some experience in the security and network field. With this simple and real example, we demonstrated simple detection and response capabilities. In our experiments, different phases of a simulated attack (PING, SCAN, CONNECTION and BAD ACTIONS), used different kinds of detection technologies that can be associated with different kinds of deception.

In particular, with this architecture and implementation: a log monitor that reads and analyzes all detection alarms and then launches deception capabilities, we have a better coordination between an intrusion-detection system and a software decoy. This integration between defense modules and the log monitor is very important since we can launch different decoys for different situations based on information about the whole arena. This significantly improves the performance of software decoy applications.

VI. CONCLUSIONS

Some concepts of this work, such as defense-in-depth, centralized coordination, surveillance, event registry, and deception, have been widely used by military forces around the world for thousands of years. In the "information age" these tactics and strategies of military forces are migrating to the digital world. This makes sense because the digital world also encompasses; enemies, attacks, networks to be defended, defense planning to be performed, countermeasures to be executed, information warfare to be deployed, technology to be developed, and deceptions to fool the enemy.

Our experiments in this thesis demonstrated the advantages of intrusion-detection software as a component in defense of computer systems, much as any military defense plan should be based on battlefield intelligence information. But intrusion information can be voluminous and needs to be collected and fused by a central "brain" as we described in this research. In the proposed architecture, network intrusion detection, host intrusion detection and log files acted as our defense modules. Their integration with a log monitor became vital for the implementation of decoy capabilities.

This approach is not new and was first proposed in 1980 [30]. Since then a log analysis has been one of the most overlooked aspects of operational computer security. Many organizations spend hundreds of thousands of dollars on intrusion-detection systems (IDS) deployments, but still ignore their firewall logs. Bird [31] suggests that the next wave in security will be to usefully correlate and process the contents of multiple logs and intrusion-detection technology in real time.

A. POSSIBLE FUTURE WORK

More tests and experiments are necessary to best address some issues not covered in this research:

- Synchronization of all modules with timestamps would be useful as the timing of events is a very important issue in this kind of approach.

- Encrypted communication between defense modules and the coordination module could help conceal deceptions.

- Redundancy of all data stored in the coordination module could increase robustness.

- Defense modules could be created with a high level of specialization. The rules and policy could be customized for each environment and each defense module.

- Integration of the intrusion-detection system with other defense technology using the Intruder Detection and Isolation Protocol (IDIP) could provide additional resources to facilitate implementation.

VII. APPENDIXES

APPENDIX A – LOG MONITOR PROGRAM

```
#!/usr/bin/perl

####################################
# Load Configuration
####################################
$decoy_file = "./decoy.log";
$ping_file = "/var/log/snort/alert";
$scan_file = "/var/log/snort/alert";
$connect_file = "/var/log/snort/alert";
$login_root_file = "/var/log/messagens";

###############################################################################
open(PING, $ping_file) or die "can't open PING file: $ping_file: $!\n";
print "Checking $ping_file ...\n";
while(<PING>) {
   next unless /ICMP Packet | Portscan detected | TELNET attempt /;
   if ("$&" == "ICMP Packet") {
      my $status = "PING";
      if (/.* (\d+\.\d+\.\d+\.\d+) -> (\d+\.\d+\.\d+\.\d+)/) {
                   &check_IP_Table($1, $2, $status);
      }#end if
   }#end if
   if ("$&" == "Portscan detected") {
      my $status = "SCAN";
      if (/.* (\d+\.\d+\.\d+\.\d+):\d+ -> (\d+\.\d+\.\d+\.\d+):\d+/) {
                   &check_IP_Table($1, $2, $status);
      }
   }#end if
   if ("$&" == "TELNET attempt") {
      my $status = "CONNECT";
      if (/.* (\d+\.\d+\.\d+\.\d+):\d+ -> (\d+\.\d+\.\d+\.\d+)/) {
                   &check_IP_Table($1, $2, $status);
      }#end if
   }#end if
}#end while

###############################################################################
open (LOG, "/var/log/messages") or die "Can't open /var/log/messagens\n";
print "\nChecking $login_root_file..\n";
while (<LOG>) {
   next unless /session opened/;
   my $status = "LOGIN";
   #$total_good_su++;
   if (/session opened for user (\w) by/) {
      &check_IP_Table($1, undef, $status);
   }#end if
}#end while


###############################################################################
#Area for Sub Rotines
###############################################################################

sub write_decoy_file {
   my $message = $_[0];
   print STDOUT $message;
   open (DECOY, ">>$decoy_file");
   print DECOY $message;
   close (DECOY);
}# end of write_decoy_file

sub check_IP_Table {
   my ($source, $dest, $status) = @_;
   $key_hash = "$source.$status";
   if ($hash{$key_hash}) {
      $hash{$key_hash}++;
                print "\tIP $source has $hash{$key_hash} attempts\n";
   }#end if
   else {
      print "\tNew Hacker Activity: IP number: $source,$status\n";
                &check_hacker($source, $dest, $status);
   }#end elsif
}#end check_IP_Table

sub check_hacker {
   my ($source, $dest, $status) = @_;
#   my $date = localtime();
   if ($status eq "PING") {
      &write_decoy_file("\t\tHONESTY: ");
                &write_decoy_file("hacker($source) $status DEST:$dest\n");
   }#end if
   if ($status eq "SCAN") {
      &write_decoy_file("\t\tLYING: ");
                &write_decoy_file("hacker($source) $status DESt:$dest\n");
                &write_decoy_file("\t\tRedirecting to FAKE SERVER...\n");
   }#end if
   if ($status eq "CONNECT") {
      &write_decoy_file("\t\tLYING: ");
                &write_decoy_file("$hacker($source) $status DEST:$dest\n");
   }#end if
   if ($status eq "LOGIN") {
      &write_decoy_file("\t\tLYING: ");
                &write_decoy_file("$hacker($source) $status DEST:$dest\n");
   }#end if
}#end check_hacker
```

APPENDIX B – LIDS AND INSTALLATION

This appendix provides guidelines to implement LIDS in the LINUX system. This guide is based on the documentation available in the LIDS official home page [42], along with some of our own updates.

Before installing LIDS in the machine, you must download the "pure" kernel developed by Linux (or did you mean Linus Torvalds?); LIDS is a patch of the kernel. Many distributors, including Red Hat, customize their kernel. Although this is not a problem, LIDS only runs over Linux pure kernel. There are many documents and books about Linux Kernel compilation [41].

1. System Environment

Operating System	LINUX RedHat 8	
Kernel	2.4.18	
LIDS version	lids-1.1.1r2-2.4.18	
Unpacked LIDS directory	/usr/src/lids-1.1.1r2-2.4.18	
Configuration directory	/etc/lids	
Configuration Files	Lids.conf	LIDS ACL configuration file
	Lids.cap	LIDS capabilities file
	Lids.pw	LIDS password file
	Lids.net	Lids mail alert configuration file

2. Installation

Before installing LIDS, the kernel must be patched. To do this, download the LIDS patch [42] that matches the specific kernel, which for this research, was lids-1.1.1r2-2.4.18. After that follow the steps:

Step	Commands	Comments
01	cd /usr/src/linux	This is the directory that you unpacked the original Linux kernel.
02	patch –p1 < /usr/src/lids-1.1.1r2-	Patch the Linux kernel

	2.4.18.patch	
03	make clean	Recompile the Linux kernel
04	make xconfig	Open Linux kernel menu. Make sure that the following options are enabled: [*] Prompt for development and/or incomplete code/drives [*] Sysctl support In the LIDS option in the kernel, make sure that the following options are enabled: [*] Linux Intrusion-detection System support [*] Attempt not to flood logs [*] Allow switching LIDS protection [*] Allow reloading config. File [*] Port scanner detector in kernel [*] Send security alerts through network
05	make dep clean bzImage	Complete the kernel recompilation
06	lilo	Do not forget to configure lilo for this new patched kernel
07	cd /usr/src/lids-1.1.1r2-2.4.18	Go to lids directory and install lidsadm an lidsconf with ./configure && make && make install
08	cp /usr/src/lids-1.1.1r2-2.4.18/example/lids.* /etc/lids	This will create the files in order to reboot the machine. To the system work properly we have to change this file according to our system configuration and security police
09	lidsconf –P	Generate the password file lids.pwd
10	reboot	Reboot the machine
11	There are some important commands to deal with LIDS implementation and debug:	
	lidsadm –S -- -LIDS_GLOBAL	Disable LIDS completely
	lidsadm –S -- +LIDS_GLOBAL	Enable LIDS completely
	lidsadm –V	This will produce output that show all LIDS options
12	In the official reference there are some old commands that use lidsadm instead of lidsconf. Use the both commands lidsadm –help and lidsconf –help to make sure about the right command.	

3. LIDS Files Configurations

- **Lids.cap file:**

```
### 0: In a system with the _POSIX_CHOWN_RESTRICTED option defined, this overrides the
restriction
### 0: of changing file ownership and group ownership.
#
+0:CAP_CHOWN

### 1: Override all DAC access, including ACL execute access if _POSIX_ACL is defined.
Excluding
### 1: DAC access covered by CAP_LINUX_IMMUTABLE.
#
+1:CAP_DAC_OVERRIDE

### 2: Overrides all DAC restrictions regarding read and search on files and directories,
including
### 2: ACL restrictions if _POSIX_ACL is defined. Excluding DAC access covered by
### 2: CAP_LINUX_IMMUTABLE.
#
+2:CAP_DAC_READ_SEARCH

### 3: Overrides all restrictions about allowed operations on files, where file owner ID
```

```
must be equal
### 3: to the user ID, except where CAP_FSETID is applicable. It doesn't override MAC and
DAC
### 3: restrictions.
#
+3:CAP_FOWNER

### 4: Overrides the following restrictions that the effective user ID shall match the
file owner ID
### 4: when setting the S_ISUID and S_ISGID bits on that file; that the effective group
ID (or one of
### 4: the supplementary group IDs) shall match the file owner ID when setting the
S_ISGID bit on
### 4: that file; that the S_ISUID and S_ISGID bits are cleared on successful return from
chown(2)
### 4: (not implemented).
#
+4:CAP_FSETID

### 5: Overrides the restriction that the real or effective user ID of a process sending
a signal must
### 5: match the real or effective user ID of the process receiving the signal.
#
-5:CAP_KILL

### 6: - Allows setgid(2) manipulation
### 6: - Allows setgroups(2)
### 6: - Allows forged gids on socket credentials passing.
#
+6:CAP_SETGID

### 7: - Allows set*uid(2) manipulation (including fsuid).
### 7: - Allows forged pids on socket credentials passing.
#
+7:CAP_SETUID

### 8: Transfer any capability in your permitted set to any pid, remove any capability in
your
### 8: permitted set from any pid.
#
+8:CAP_SETPCAP

### 9: Allow modification of S_IMMUTABLE and S_APPEND file attributes.
#
-9:CAP_LINUX_IMMUTABLE

### 10: Allows binding to TCP/UDP sockets below 1024.
#
-10:CAP_NET_BIND_SERVICE

### 11: Allow broadcasting, listen to multicast.
#
+11:CAP_NET_BROADCAST

### 12: - Allow interface configuration
### 12: - Allow administration of IP firewall, masquerading and accounting
### 12: - Allow setting debug option on sockets
### 12: - Allow modification of routing tables
### 12: - Allow setting arbitrary process / process group ownership on sockets
### 12: - Allow binding to any address for transparent proxying
### 12: - Allow setting TOS (type of service)
### 12: - Allow setting promiscuous mode
### 12: - Allow clearing driver statistics
### 12: - Allow multicasting
### 12: - Allow read/write of device-specific registers
#
-12:CAP_NET_ADMIN

### 13: - Allow use of RAW sockets
### 13: - Allow use of PACKET sockets
#
-13:CAP_NET_RAW

### 14: - Allow locking of shared memory segments
### 14: - Allow mlock and mlockall (which doesn't really have anything to do with IPC)
#
+14:CAP_IPC_LOCK

### 15: Override IPC ownership checks.
#
+15:CAP_IPC_OWNER

### 16: Insert and remove kernel modules.
#
-16:CAP_SYS_MODULE

### 17: - Allow ioperm/iopl and /dev/port access
### 17: - Allow /dev/mem and /dev/kmem acess
### 17: - Allow raw block devices (/dev/[sh]d??) acess
#
-17:CAP_SYS_RAWIO

### 18: Allow use of chroot()
#
+18:CAP_SYS_CHROOT

### 19: Allow ptrace() of any process
#
-19:CAP_SYS_PTRACE

### 20: Allow configuration of process accounting
#
+20:CAP_SYS_PACCT

### 21:

### 21: - Allow configuration of the secure attention key
### 21: - Allow administration of the random device
### 21: - Allow device administration (mknod)
### 21: - Allow examination and configuration of disk quotas
### 21: - Allow configuring the kernel's syslog (printk behaviour)
### 21: - Allow setting the domainname
### 21: - Allow setting the hostname
### 21: - Allow calling bdflush()
### 21: - Allow mount() and umount(), setting up new smb connection
### 21: - Allow some autofs root ioctls
### 21: - Allow nfsservctl
### 21: - Allow VM86_REQUEST_IRQ
### 21: - Allow to read/write pci config on alpha
### 21: - Allow irix_prctl on mips (setstacksize)
### 21: - Allow flushing all cache on m68k (sys_cacheflush)
### 21: - Allow removing semaphores
### 21: - Used instead of CAP_CHOWN to "chown" IPC message queues, semaphores and shared
memory
### 21: - Allow locking/unlocking of shared memory segment
### 21: - Allow turning swap on/off
### 21: - Allow forged pids on socket credentials passing
### 21: - Allow setting readahead and flushing buffers on block devices
### 21: - Allow setting geometry in floppy driver
### 21: - Allow turning DMA on/off in xd driver
### 21: - Allow administration of md devices (mostly the above, but some extra ioctls)
### 21: - Allow tuning the ide driver
### 21: - Allow access to the nvram device
### 21: - Allow administration of apm_bios, serial and bttv (TV) device
### 21: - Allow manufacturer commands in isdn CAPI support driver
### 21: - Allow reading non-standardized portions of pci configuration space
### 21: - Allow DDI debug ioctl on sbpcd driver
### 21: - Allow setting up serial ports
### 21: - Allow sending raw qic-117 commands
### 21: - Allow enabling/disabling tagged queuing on SCSI controllers and sending
arbitrary SCSI commands
### 21: - Allow setting encryption key on loopback filesystem
#
-21:CAP_SYS_ADMIN

### 22: Allow use of reboot()
#
+22:CAP_SYS_BOOT

### 23: - Allow raising priority and setting priority on other (different UID) processes
### 23: - Allow use of FIFO and round-robin (realtime) scheduling on own processes and
setting
### 23:   the scheduling algorithm used by another process.
#
+23:CAP_SYS_NICE

Override resource limits. Set resource limits.
### 24: - Override quota limits.
### 24: - Override reserved space on ext2 filesystem
### 24:   NOTE: ext2 honors fsuid when checking for resource overrides, so you can
override
### 24:   using fsuid too
### 24: - Override size restrictions on IPC message queues
### 24: - Allow more than 64hz interrupts from the real-time clock
### 24: - Override max number of consoles on console allocation
### 24: - Override max number of keymaps
#
+24:CAP_SYS_RESOURCE

### 25: - Allow manipulation of system clock
### 25: - Allow irix_stime on mips
### 25: - Allow setting the real-time clock
#
-25:CAP_SYS_TIME

### 26: - Allow configuration of tty devices
### 26: - Allow vhangup() of tty
#
+26:CAP_SYS_TTY_CONFIG

### 27: Allow the privileged aspects of mknod()
###
+27:CAP_MKNOD

### 28:Allow taking of leases on files */
###
+28:CAP_LEASE


### 29: Restricts viewable processes by a user.
#
+29:CAP_HIDDEN

### 30: Allow to kill protected processes
#
-30:CAP_KILL_PROTECTED

### 31: Protect process against signals
#
+31:CAP_PROTECTED
```

- **Lids.conf file configuration**

```
#
#             This file is auto generated by lidsconf
#             Please do not modify this file by hand
```

```
#
0:0::1:0:791855:770:/sbin:0-0
0:0::1:0:840334:770:/bin:0-0
0:0::1:0:2:769:/boot:0-0
0:0::1:0:921136:770:/lib:0-0
0:0::1:0:323201:770:/usr:0-0
0:0::1:0:226241:770:/etc:0-0
0:0::0:0:745099:770:/etc/lids:0-0
0:0::3:0:743370:770:/var/log:0-0
0:0::7:0:745391:770:/var/log/wtmp:0-0
840445:770:/bin/login:7:0:743373:770:/var/log/lastlog:0-0
695028:770:/etc/rc.d/rc:16:-1:-1:30:CAP_KILL_PROTECTED:0-0
695028:770:/etc/rc.d/rc:16:-1:-1:12:CAP_NET_ADMIN:0-0
695028:770:/etc/rc.d/rc:16:-1:-1:21:CAP_SYS_ADMIN:0-0
840422:770:/etc/rc.d/init.d/halt:16:-1:-1:30:CAP_KILL_PROTECTED:0-0
840422:770:/etc/rc.d/init.d/halt:16:-1:-1:21:CAP_SYS_ADMIN:0-0
840422:770:/etc/rc.d/init.d/halt:16:-1:-1:17:CAP_SYS_RAWIO:0-0
840422:770:/etc/rc.d/init.d/halt:16:-1:-1:12:CAP_NET_ADMIN:0-0
0:0::1:0:290881:770:/root:0-0
387985:770:/usr/sbin/sshd:16:0:22-22:10:CAP_NET_BIND_SERVICE:0-0
387985:770:/usr/sbin/sshd:16:0:-1:31:CAP_PROTECTED:0-0
145448:770:/usr/X11R6/bin/XF86_SVGA:16:0:-1:17:CAP_SYS_RAWIO:0-0
340048:770:/usr/bin/ssh:16:0:0-1024:10:CAP_NET_BIND_SERVICE:0-0
0:0::1:0:791855:770:/sbin:0-0
0:0::1:0:840338:770:/usr/local:0-0
0:0::1:0:743375:770:/opt:0-0
0:0::1:0:921138:770:/usr/local/etc:0-0
0:0::0:0:227208:770:/etc/shadow:0-0
0:0::0:0:227644:770:/etc/lilo.conf:0-0
840445:770:/bin/login:1:0:227208:770:/etc/shadow:0-0
840414:770:/bin/su:1:0:227208:770:/etc/shadow:0-0
840414:770:/bin/su:16:0:-1:7:CAP_SETUID:0-0
840414:770:/bin/su:16:0:-1:6:CAP_SETGID:0-0
840369:770:/bin/bash:7:0:293155:770:/root/.bash_history:0-0
840445:770:/bin/login:7:0:745391:770:/var/log/wtmp:0-0
791916:770:/sbin/init:7:0:745391:770:/var/log/wtmp:0-0
791916:770:/sbin/init:7:0:743373:770:/var/log/lastlog:0-0
791915:770:/sbin/halt:7:0:743373:770:/var/log/lastlog:0-0
791915:770:/sbin/halt:7:0:745391:770:/var/log/wtmp:0-0
695030:770:/etc/rc.d/rc.sysinit:7:1:745391:770:/var/log/wtmp:0-0
695030:770:/etc/rc.d/rc.sysinit:7:1:743373:770:/var/log/lastlog:0-0
791945:770:/sbin/hwclock:7:0:226410:770:/etc/adjtime:0-0
791916:770:/sbin/init:16:0:-1:5:CAP_KILL:0-0
840422:770:/etc/rc.d/init.d/halt:16:1:-1:5:CAP_KILL:0-0
791863:770:/sbin/update:16:0:-1:21:CAP_SYS_ADMIN:0-0
0:0::0:0:599268:770:/var/www:0-0
0:0::0:0:339533:770:/usr/bin/consolehelper:0-0
0:0::0:0:226493:770:/etc/ssh/sshd_config:0-0
0:0::0:0:227197:770:/etc/ssh/ssh_host_key:0-0
0:0::0:0:227214:770:/etc/ssh/ssh_host_dsa_key:0-0
387985:770:/usr/sbin/sshd:1:0:226493:770:/etc/ssh/sshd_config:0-0
387985:770:/usr/sbin/sshd:1:0:227197:770:/etc/ssh/ssh_host_key:0-0
387985:770:/usr/sbin/sshd:1:0:227214:770:/etc/ssh/ssh_host_dsa_key:0-0
387985:770:/usr/sbin/sshd:7:0:745391:770:/var/log/wtmp:0-0
387985:770:/usr/sbin/sshd:7:0:743373:770:/var/log/lastlog:0-0
387985:770:/usr/sbin/sshd:16:0:-1:7:CAP_SETUID:0-0
387985:770:/usr/sbin/sshd:16:0:-1:6:CAP_SETGID:0-0
387985:770:/usr/sbin/sshd:16:0:-1:3:CAP_FOWNER:0-0
387985:770:/usr/sbin/sshd:16:0:-1:0:CAP_CHOWN:0-0
387985:770:/usr/sbin/sshd:16:0:-1:1:CAP_DAC_OVERRIDE:0-0
387985:770:/usr/sbin/sshd:16:0:22-22:10:CAP_NET_BIND_SERVICE:0-0
387985:770:/usr/sbin/sshd:16:0:-1:18:CAP_SYS_CHROOT:0-0
387985:770:/usr/sbin/sshd:1:0:227208:770:/etc/shadow:0-0
857457:770:/usr/local/bin/snort:16:0:-1:1:CAP_DAC_OVERRIDE:0-0
857457:770:/usr/local/bin/snort:16:0:-1:13:CAP_NET_RAW:0-0
857457:770:/usr/local/bin/snort:16:0:-1:29:CAP_HIDDEN:0-0
857457:770:/usr/local/bin/snort:16:0:-1:7:CAP_SETUID:0-0
857457:770:/usr/local/bin/snort:16:0:-1:6:CAP_SETGID:0-0
857457:770:/usr/local/bin/snort:7:0:81730:770:/var/log/snort:0-0
```

- **Lids.net file configuration**

```
# LIDS
# Send Alert Message From Network
# for lids 0.9.8
#             xie@gnuchina.org
# -------------------------------------------------------------------

# MAIL_SWITCH = 1 | 0
# 1 , send alert function is on
# 0, send alert function is off

MAIL_SWITCH= 0

# MAIL_RELAY=hex IP:port
# IP11.1 of the machine that will be directly connected by LIDS
# for relaying its mails. Port is usually 25, but who knows...
MAIL_RELAY=127.0.0.1:25

# MAIL_SOURCE=source machine :
# Name of the source machine, used for the ehlo identification.
# Note that a bad name here could make the mail relay refuse your
# mails.
MAIL_SOURCE=decoy.cs.nps.navy.mil

# MAIL_FROM=sender address
#          Sender address, which will also be in the ``from'' field.
MAIL_FROM= LIDS_ALERT@nps.navy.mil

# MAIL_TO=recipient address :
#          Recipient address.
MAIL_TO= vjmontei@nps.navy.mil

# MAIL_SUBJECT= subject :
#          Subject of the mail.
MAIL_SUBJECT= LIDS ALert
```

- **Lids.pw password file**

```
8fee5733a4caef5b1992e25508e0428740f99be7
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C – SNORT FILE CONFIGURATION

```
# Modified by Valter Monteiro - Thesis Research
#--------------------------------------------------
#   http://www.snort.org     Snort 1.9.0 Ruleset
#     Contact: snort-sigs@lists.sourceforge.net
#--------------------------------------------------
# NOTE:This ruleset only works for 1.9.0 and later
#--------------------------------------------------
# $Id: snort.conf,v 1.110 2002/08/14 03:17:58 chrisgreen Exp $
#
###################################################
# This file contains a sample snort configuration.
# You can take the following steps to create your
# own custom configuration:
#
#  1) Set the network variables for your network
#  2) Configure preprocessors
#  3) Configure output plugins
#  4) Customize your rule set
#
###################################################
# Step #1: Set the network variables:
#
# You must change the following variables to reflect
# your local network. The variable is currently
# setup for an RFC 1918 address space.
#
# You can specify it explicitly as:
#
# var HOME_NET 192.160.0.0/24
#
# or use global variable $<interfacename>_ADDRESS
# which will be always initialized to IP address and
# netmask of the network interface which you run
# snort at.
#
# var HOME_NET $eth0_ADDRESS
#
# You can specify lists of IP addresses for HOME_NET
# by separating the IPs with commas like this:
#
# var HOME_NET [10.1.1.0/24,192.168.1.0/24]
#
# MAKE SURE YOU DON'T PLACE ANY SPACES IN YOUR LIST!
#
# or you can specify the variable to be any IP address
# like this:

var HOME_NET 192.168.0.3/24

# Set up the external network addresses as well.
# A good start may be "any"

var EXTERNAL_NET any

# Configure your server lists.  This allows snort to only look for attacks
# to systems that have a service up.  Why look for HTTP attacks if you are
# not running a web server?  This allows quick filtering based on IP addresses
# These configurations MUST follow the same configuration scheme as defined
# above for $HOME_NET.

# List of DNS servers on your network
var DNS_SERVERS $HOME_NET
```

```
# List of SMTP servers on your network
var SMTP_SERVERS $HOME_NET

# List of web servers on your network
var HTTP_SERVERS $HOME_NET

# List of sql servers on your network
var SQL_SERVERS $HOME_NET

# List of telnet servers on your network
var TELNET_SERVERS $HOME_NET

# Configure your service ports.  This allows snort to look for attacks
# destined to a specific application only on the ports that application
# runs on.  For example, if you run a web server on port 8081, set your
# HTTP_PORTS variable like this:
#
# var HTTP_PORTS 8081
#
# Port lists must either be continuous [eg 80:8080], or a single port [eg 80].
# We will adding support for a real list of ports in the future.

# Ports you run web servers on
var HTTP_PORTS 80

# Ports you want to look for SHELLCODE on.
var SHELLCODE_PORTS !80

# Ports you do oracle attacks on
var ORACLE_PORTS 1521

# other variables
#
# AIM servers.  AOL has a habit of adding new AIM servers, so instead of
# modifying the signatures when they do, we add them to this list of
# servers.
var AIM_SERVERS
[64.12.24.0/24,64.12.25.0/24,64.12.26.14/24,64.12.28.0/24,64.12.29.0/24,64.12.161.0/24,64.12.163.0/24,205

# Path to your rules files (this can be a relative path)
var RULE_PATH ../rules

###################################################
# Step #2: Configure preprocessors
#
# General configuration for preprocessors is of
# the form
# preprocessor <name_of_processor>: <configuration_options>

# frag2: IP defragmentation support
# -------------------------------
# This preprocessor performs IP defragmentation.  This plugin will also detect
# people launching fragmentation attacks (usually DoS) against hosts.  No
# arguments loads the default configuration of the preprocessor, which is a
# 60 second timeout and a 4MB fragment buffer.

# The following (comma delimited) options are available for frag2
#    timeout [seconds] - sets the number of [seconds] than an unfinished
#                        fragment will be kept around waiting for completion,
#                        if this time expires the fragment will be flushed
#    memcap [bytes] - limit frag2 memory usage to [number] bytes
#                      (default:  4194304)
#
#    min_ttl [number] - minimum ttl to accept
#
#    ttl_limit [number] - difference of ttl to accept without alerting
#                         will cause false positves with router flap
#
```

```
# Frag2 uses Generator ID 113 and uses the following SIDS
# for that GID:
#  SID     Event description
# -----   -------------------
#   1       Oversized fragment (reassembled frag > 64k bytes)
#   2       Teardrop-type attack

preprocessor frag2

# stream4: stateful inspection/stream reassembly for Snort
#----------------------------------------------------------------------
# Use in concert with the -z [all|est] command line switch to defeat
# stick/snot against TCP rules.  Also performs full TCP stream
# reassembly, stateful inspection of TCP streams, etc.  Can statefully
# detect various portscan types, fingerprinting, ECN, etc.

# stateful inspection directive
# no arguments loads the defaults (timeout 30, memcap 8388608)
# options (options are comma delimited):
#   detect_scans - stream4 will detect stealth portscans and generate alerts
#                  when it sees them when this option is set
#   detect_state_problems - detect TCP state problems, this tends to be very
#                           noisy because there are a lot of crappy ip stack
#                           implementations out there
#
#   disable_evasion_alerts - turn off the possibly noisy mitigation of
#                            overlapping sequences.
#
#
#   min_ttl [number]       - set a minium ttl that snort will accept to
#                            stream reassembly
#
#   ttl_limit [number]     - differential of the initial ttl on a session versus
#                             the normal that someone may be playing games.
#                             Routing flap may cause lots of false positives.
#
#   keepstats [machine|binary] - keep session statistics, add "machine" to
#                         get them in a flat format for machine reading, add
#                         "binary" to get them in a unified binary output
#                         format
#   noinspect - turn off stateful inspection only
#   timeout [number] - set the session timeout counter to [number] seconds,
#                      default is 30 seconds
#   memcap [number] - limit stream4 memory usage to [number] bytes
#   log_flushed_streams - if an event is detected on a stream this option will
#                         cause all packets that are stored in the stream4
#                         packet buffers to be flushed to disk.  This only
#                         works when logging in pcap mode!
#
# Stream4 uses Generator ID 111 and uses the following SIDS
# for that GID:
#  SID     Event description
# -----   -------------------
#   1       Stealth activity
#   2       Evasive RST packet
#   3       Evasive TCP packet retransmission
#   4       TCP Window violation
#   5       Data on SYN packet
#   6       Stealth scan: full XMAS
#   7       Stealth scan: SYN-ACK-PSH-URG
#   8       Stealth scan: FIN scan
#   9       Stealth scan: NULL scan
#   10      Stealth scan: NMAP XMAS scan
#   11      Stealth scan: Vecna scan
#   12      Stealth scan: NMAP fingerprint scan stateful detect
#   13      Stealth scan: SYN-FIN scan
#   14      TCP forward overlap

preprocessor stream4: detect_scans, disable_evasion_alerts
```

```
# tcp stream reassembly directive
# no arguments loads the default configuration
#   Only reassemble the client,
#   Only reassemble the default list of ports (See below),
#   Give alerts for "bad" streams
#
# Available options (comma delimited):
#   clientonly - reassemble traffic for the client side of a connection only
#   serveronly - reassemble traffic for the server side of a connection only
#   both - reassemble both sides of a session
#   noalerts - turn off alerts from the stream reassembly stage of stream4
#   ports [list] - use the space separated list of ports in [list], "all"
#                  will turn on reassembly for all ports, "default" will turn
#                  on reassembly for ports 21, 23, 25, 53, 80, 143, 110, 111
#                  and 513

preprocessor stream4_reassemble

# http_decode: normalize HTTP requests
# ------------------------------------
# http_decode normalizes HTTP requests from remote
# machines by converting any %XX character
# substitutions to their ASCII equivalent. This is
# very useful for doing things like defeating hostile
# attackers trying to stealth themselves from IDSs by
# mixing these substitutions in with the request.
# Specify the port numbers you want it to analyze as arguments.
#
# Major code cleanups thanks to rfp
#
# unicode          - normalize unicode
# iis_alt_unicode  - %u encoding from iis
# double_encode    - alert on possible double encodings
# iis_flip_slash   - normalize \ as /
# full_whitespace  - treat \t as whitespace ( for apache )
#
# for that GID:
#  SID     Event description
# -----   -------------------
#   1       UNICODE attack
#   2       NULL byte attack

preprocessor http_decode: 80 unicode iis_alt_unicode double_encode iis_flip_slash full_whitespac

# rpc_decode: normalize RPC traffic
# ---------------------------------
# RPC may be sent in alternate encodings besides the usual
# 4-byte encoding that is used by default.  This preprocessor
# normalized RPC traffic in much the same way as the http_decode
# preprocessor.  This plugin takes the ports numbers that RPC
# services are running on as arguments.
# The RPC decode preprocessor uses generator ID 106 and does not
# generate any SIDs at this time.

preprocessor rpc_decode: 111 32771

# bo: Back Orifice detector
# -------------------------
# Detects Back Orifice traffic on the network.  This preprocessor
# uses the Back Orifice "encryption" algorithm to search for
# traffic conforming to the Back Orifice protocol (not BO2K).
# This preprocessor can take two arguments.  The first is "-nobrute"
# which turns off the plugin's brute forcing routine (brute forces
# the key space of the protocol to find BO traffic).  The second
# argument that can be passed to the routine is a number to use
# as the default key when trying to decrypt the traffic.  The
# default value is 31337 (just like BO).  Be aware that turning on
# the brute forcing option runs the risk of impacting the overall
# performance of Snort, you've been warned...
#
# The Back Orifice detector uses Generator ID 105 and uses the
# following SIDS for that GID:
#  SID     Event description
# -----   -------------------
#   1       Back Orifice traffic detected

preprocessor bo: -nobrute

# telnet_decode: Telnet negotiation string normalizer
# ---------------------------------------------------
# This preprocessor "normalizes" telnet negotiation strings from
# telnet and ftp traffic.  It works in much the same way as the
# http_decode preprocessor, searching for traffic that breaks up
# the normal data stream of a protocol and replacing it with
# a normalized representation of that traffic so that the "content"
# pattern matching keyword can work without requiring modifications.
# This preprocessor requires no arguments.
# Portscan uses Generator ID 109 and does not generate any SID currently.

preprocessor telnet_decode

# Portscan: detect a variety of portscans
# ---------------------------------------
# portscan preprocessor by Patrick Mullen <p_mullen@linuxrc.net>
# This preprocessor detects UDP packets or TCP SYN packets going to
# four different ports in less than three seconds. "Stealth" TCP
# packets are always detected, regardless of these settings.
# Portscan uses Generator ID 100 and uses the following SIDS for that GID:
#  SID     Event description
# -----   -------------------
#   1       Portscan detect
#   2       Inter-scan info
#   3       Portscan End

# preprocessor portscan: $HOME_NET 4 3 portscan.log

# Use portscan-ignorehosts to ignore TCP SYN and UDP "scans" from
# specific networks or hosts to reduce false alerts. It is typical
# to see many false alerts from DNS servers so you may want to
# add your DNS servers here. You can all multiple hosts/networks
# in a whitespace-delimited list.
#
#preprocessor portscan-ignorehosts: 0.0.0.0

# arpspoof
#----------------------------------------
# Experimental ARP detection code from Jeff Nathan, detects ARP attacks,
# unicast ARP requests, and specific ARP mapping monitoring.  To make use
# of this preprocessor you must specify the IP and hardware address of hosts on # the same layer 2 segme
IP MAC combo per line.
# Also takes a "-unicast" option to turn on unicast ARP request detection.
# Arpspoof uses Generator ID 112 and uses the following SIDS for that GID:
#  SID     Event description
# -----   -------------------
#   1       Unicast ARP request
#   2       Etherframe ARP mismatch (src)
#   3       Etherframe ARP mismatch (dst)
#   4       ARP cache overwrite attack

#preprocessor arpspoof
#preprocessor arpspoof_detect_host: 192.168.40.1 f0:0f:00:f0:0f:00

# ASN1 Decode
#-----------------------------------------
# This is an experimental preprocessor.  ASN.1 decoder and analysis plugin
# from Andrew R. Baker.  This preprocessor will detect abuses of the ASN.1
# protocol that higher level protocols (like SSL, SNMP, x.509, etc) rely on.
# The ASN.1 decoder uses Generator ID 115 and uses the following SIDs for
# that GID:
#  SID     Event description
# -----   -------------------
#   1       Indefinite length
#   2       Invalid length
#   3       Oversized item
#   4       ASN.1 specification violation
#   5       Dataum bad length

preprocessor asn1_decode

# Fnord
#-----------------------------------------
# This is an experimental preprocessor.  Polymorphic shellcode analyzer and
# detector by Dragos Ruiu.  This preprocessor will watch traffic for
# polymorphic NOP-type sleds to defeat tools like ADMutate.  The Fnord detector
# uses Generator ID 114 and the following SIDs:
#  SID     Event description
# -----   -------------------
#   1       NOP-sled detected

# preprocessor fnord

# Conversation
#------------------------------------------
# This preprocessor tracks conversations for tcp, udp and icmp traffic.  It
# is a prerequisite for running portscan2.
#
# allowed_ip_protcols 1 6 17
#      list of allowed ip protcols ( defaults to any )
#
# timeout [num]
#      conversation timeout ( defaults to 60 )
#
#
# max_conversations [num]
#      number of conversations to support at once (defaults to 65335)
#
#
# alert_odd_protocols
#      alert on protocols not listed in allowed_ip_protocols

preprocessor conversation: allowed_ip_protocols all, timeout 60, max_conversations 32000

# Portscan2
#-------------------------------------------
# Portscan 2, detect portscans in a new and exciting way.
#
# Available options:
#       scanners_max [num]
#       targets_max [num]
#       target_limit [num]
#       port_limit [num]
#       timeout [num]
#       log [logdir]

preprocessor portscan2: scanners_max 3200, targets_max 5000, target_limit 5, port_limit 20, time

# Experimental Perf stats
# -----------------------
# No docs. Highly subject to change.
#
# preprocessor perfmonitor: console flow events time 10

####################################################################
# Step #3: Configure output plugins
#
# Uncomment and configure the output plugins you decide to use.
# General configuration for output plugins is of the form:
#
# output <name_of_plugin>: <configuration_options>
#
# alert_syslog: log alerts to syslog
# ----------------------------------
# Use one or more syslog facilities as arguments
#
# output alert_syslog: LOG_AUTH LOG_ALERT

# log_tcpdump: log packets in binary tcpdump format
# -------------------------------------------------
# The only argument is the output file name.
#
# output log_tcpdump: tcpdump.log

# database: log to a variety of databases
# ---------------------------------------
# See the README.database file for more information about configuring
# and using this plugin.
#
# output database: log, mysql, user=root password=test dbname=db host=localhost
# output database: alert, postgresql, user=snort dbname=snort
# output database: log, unixodbc, user=snort dbname=snort
# output database: log, mssql, dbname=snort user=snort password=test

# xml: xml logging
# ----------------
# See the README.xml file for more information about configuring
# and using this plugin.
#
# output xml: log, file=/var/log/snortxml

# unified: Snort unified binary format alerting and logging
# -------------------------------------------------------------
# The unified output plugin provides two new formats for logging
# and generating alerts from Snort, the "unified" format.  The
# unified format is a straight binary format for logging data
# out of Snort that is designed to be fast and efficient.  Used
# with barnyard (the new alert/log processor), most of the overhead
# for logging and alerting to various slow storage mechanisms
# such as databases or the network can now be avoided.
#
# Check out the spo_unified.h file for the data formats.
#
# Two arguments are supported.
#    filename - base filename to write to (current time_t is appended)
#    limit    - maximum size of spool file in MB (default: 128)
#
# output alert_unified: filename snort.alert, limit 128
# output log_unified: filename snort.log, limit 128

# trap_snmp: SNMP alerting for Snort
# -------------------------------------------------------------
# Read the README.SNMP file for more information on enabling and using this
# plug-in.
#
#

#The trap_snmp plugin accepts the following notification options
# [c],[p[m|s]]
# where,
#     c : Generate compact notifications. (Saves on bandwidth by
#         not reporting MOs for which values are unknown, not
#         available or, not applicable). By default this option is reset.
#     p : Generate a print of the invariant part of the offending packet.
#         This can be used to track the packet across the Internet.
#         By default this option is reset.
#     m : Use the MD5 algorithm to generate the packet print.
#         By default this algorithm is used.
#     s : Use the SHA1 algorithm to generate the packet print.
#
# The trap_snmp plugin requires several parameters
# The parameters depend on the Snmpversion that is used (specified)
# For the SNMPv2c case the parameters will be as follows
#  alert, <sensorID>, [NotificationOptions] ,
#         {trap|inform} -v <SnmpVersion> -p <portNumber> <hostName> <community>
#
# For SNMPv2c traps with MD5 digest based packetPrint generation
#
# output trap_snmp: alert, 7, cpm, trap -v 2c myTrapListener myCommunity
#
# For SNMPv2c informs with the 'compact' notification option
#
#output trap_snmp: alert, 7, c, inform -v 2c myTrapListener myCommunity
#
#
# For SNMPv3 traps with
# security name = snortUser
# security level = authentication and privacy
# authentication parameters :
#           authentication protocol = SHA ,
#           authentication pass phrase = SnortAuthPassword
# privacy (encryption) parameters
#           privacy protocol = DES,
#           privacy pass phrase = SnortPrivPassword
#
#output trap_snmp: alert, 7, trap -v 3 -u snortUser -l authPriv -a SHA -A SnortAuthPassword -x D
myTrapListener
#For SNMPv3 informs with authentication and encryption
#output trap_snmp: alert, 7, inform -v 3 -u snortUser -l authPriv -a SHA -A SnortAuthPassword -x
myTrapListener

# You can optionally define new rule types and associate one or
# more output plugins specifically to that type.
#
# This example will create a type that will log to just tcpdump.
# ruletype suspicious
# {
#   type log
#   output log_tcpdump: suspicious.log
# }
#
# EXAMPLE RULE FOR SUSPICIOUS RULETYPE:
# suspicious $HOME_NET any -> $HOME_NET 6667 (msg:"Internal IRC Server";)
#
# This example will create a rule type that will log to syslog
# and a mysql database.
# ruletype redalert
# {
#   type alert
#   output alert_syslog: LOG_AUTH LOG_ALERT
#   output database: log, mysql, user=snort dbname=snort host=localhost
# }
#
# EXAMPLE RULE FOR REDALERT RULETYPE
# redalert $HOME_NET any -> $EXTERNAL_NET 31337 (msg:"Someone is being LEET"; \
#   flags:A+;)

#
# Include classification & priority settings
#

include classification.config

#
# Include reference systems
#
```

```
include reference.config

####################################################################
# Step #4: Customize your rule set
#
# Up to date snort rules are available at http://www.snort.org
#
# The snort web site has documentation about how to write your own
# custom snort rules.
#
# The rules included with this distribution generate alerts based on
# on suspicious activity. Depending on your network environment, your
# security policies, and what you consider to be suspicious, some of
# these rules may either generate false positives ore may be detecting
# activity you consider to be acceptable; therefore, you are
# encouraged to comment out rules that are not applicable in your
# environment.
#
# Note that using all of the rules at the same time may lead to
# serious packet loss on slower machines. YMMV, use with caution,
# standard disclaimers apply. :)
#
# The following individuals contributed many of rules in this
# distribution.
#
# Credits:
#   Ron Gula <rgula@securitywizards.com> of Network Security Wizards
#   Max Vision <vision@whitehats.com>
#   Martin Markgraf <martin@mail.du.gtn.com>
#   Fyodor Yarochkin <fygrave@tigerteam.net>
#   Nick Rogness <nick@rapidnet.com>
#   Jim Forster <jforster@rapidnet.com>
#   Scott McIntyre <scott@whoi.edu>
#   Tom Vandepoel <Tom.Vandepoel@ubizen.com>
#   Brian Caswell <bmc@snort.org>
#   Zeno <admin@cgisecurity.com>
#   Ryan Russell <ryan@securityfocus.com>
#
#=========================================
# Include all relevant rulesets here
#
# shellcode, policy, info, backdoor, and virus rulesets are
# disabled by default.  These require tuning and maintance.
# Please read the included specific file for more information.
#=========================================

#include $RULE_PATH/bad-traffic.rules
#include $RULE_PATH/exploit.rules
#include $RULE_PATH/scan.rules
#include $RULE_PATH/finger.rules
#include $RULE_PATH/ftp.rules
include $RULE_PATH/telnet.rules
#include $RULE_PATH/rpc.rules
#include $RULE_PATH/rservices.rules
#include $RULE_PATH/dos.rules
#include $RULE_PATH/ddos.rules
#include $RULE_PATH/dns.rules
#include $RULE_PATH/tftp.rules

#include $RULE_PATH/web-cgi.rules
#include $RULE_PATH/web-coldfusion.rules
#include $RULE_PATH/web-iis.rules
#include $RULE_PATH/web-frontpage.rules
#include $RULE_PATH/web-misc.rules
#include $RULE_PATH/web-client.rules
#include $RULE_PATH/web-php.rules

#include $RULE_PATH/sql.rules
```

```
#include $RULE_PATH/x11.rules
#include $RULE_PATH/icmp.rules
#include $RULE_PATH/netbios.rules
#include $RULE_PATH/misc.rules
#include $RULE_PATH/attack-responses.rules
#include $RULE_PATH/oracle.rules
#include $RULE_PATH/mysql.rules
#include $RULE_PATH/snmp.rules

#include $RULE_PATH/smtp.rules
#include $RULE_PATH/imap.rules
#include $RULE_PATH/pop3.rules

#include $RULE_PATH/nntp.rules
#include $RULE_PATH/other-ids.rules
#include $RULE_PATH/web-attacks.rules
#include $RULE_PATH/backdoor.rules
#include $RULE_PATH/shellcode.rules
#include $RULE_PATH/policy.rules
#include $RULE_PATH/porn.rules
#include $RULE_PATH/info.rules
#include $RULE_PATH/icmp-info.rules
include $RULE_PATH/icmp.rules
#include $RULE_PATH/virus.rules
#include $RULE_PATH/chat.rules
#include $RULE_PATH/multimedia.rules
#include $RULE_PATH/p2p.rules
#include $RULE_PATH/experimental.rules
#include $RULE_PATH/local.rules
```

LIST OF REFERENCES

[1] Michael,J.B., and Riehle,R.D., Intelligent Software Decoys. Proc. Monterey Workshop: Eng. Automation for Software Intensive Syst. Integration, Monterey, California: Naval Postgraduate School, June 2001, pp.178-187

[2] Sun Tzu, *The Art of War*, Oxford University Press, January 1986

[3] Dunnigan, J.F., and Nofi, A.A, *Victory and Deceit*, second edition: Deception and Trickery in War. San Jose, California: Writers Club Press, 2001

[4] Cohen,F., *A Framework for Deception*, July, 1993.

[5] Rowe, Neil C., *Counterplanning Deceptions to Foil Cyber-Attack Plans*, Proceedings of the 2003 IEEE, workshop on Information Assurance, United State Military Academy, West Point, New York June 2003

[6] Neil C. Rowe, J.Bret Michael, Mikhail Auguston, and Richard Riehle, *Software decoys for Software Counterintelligence*, June 2002

[7] Honeynet Project, *Know Your Enemy*. Addison-Wesley, 2002

[8] Winn Schwartau, Time Based Security, Interpact Press, February 1999

[9] Collection of papers, NPS Web Page – Software Decoy Project

[10] The Official Red Hat Linux Customization Guide, Red Hat 8.0, 2002

[11] Thomas A. Wadlow, *The Process of Network Security*. Addison-Wesley, 2000

[12] Breatt Glass, *Log Monitor in BSD UNIX*, Laramie, presented at BSDCon 2002, San Francisco.

[13] Lutz Prechelt, *An empirical comparison of C, C++, Java, perl, Python, Rexx, and Tcl*, Faukultat fur

Java, perl, Python, Rexx, and Tcl, Faukultat fur Informatik, University Karlsruhe

[14] Diego Zamboni, ***Using Internal Sensor for Computer Intrusion Detection***, PhD Thesis, Center for Education and Research in Information Assurance and Security, Purdue University, August 2001

[15] Proctor, Paul E., ***The Practical Intrusion Detection Handbook***, Prentice-Hall, 2001

[16] The Intrusion Detection Systems Consortium (IDSC), ***An Introduction to Intrusion Detection Assessment***, March 1999

[17] Mark Crosbie and Gene Spafford. Active defense of a computer system

using autonomous agents. Technical Report 95-008, COAST Group, Department

of Computer Sciences, Purdue University, West Lafayette, Indiana,

February 1995.

[18] Build a Secure System with LIDS. URL http://www.lids.org, March 2003

[19] Anderson, J.P., ***Computer Security Technology Planning Study***, ESD-TR-73-51, Vol1, Hanscom AFB, Massachusetts, 1972

[20] Donald L. Brinkley and Roger R. Schell, ***Concepts and terminology for computer Security***, May 1993

[21] Georgios Fragkos, Master’s Thesis , ***An Event-Trace Language for Software Decoys***, September 2002, Naval Postgraduate School

[22] Ko, C.,Fraser, T., Badger, L., Kilpatrick, D., Detecting and Countering System Intrusions Using Software Wrappers, In Proc. 9th USENIX Security Sysposium, Denver, Colorado, August 2000.

[23] Northcutt S., Novak J., ***Network Intrusion Detection,*** Third Edition, New Riders, 2003

[24] SANS Institute, Intrusion Detection Snort Style Booklet, 2002.

[25] Martin Roesh, *Snort Official Manual*, release 1.9, April 2002. URL http://www.snort.org

[26] Krzysztof Zaraska, ***IDS Active Response Mechanisms: Countermeasure Subsystem for Prelude IDS***, July 2002.

[27] Jason Larsen, Jed Halie, ***Understanding IDS Activate Response Mechanisms***, securityfocus, January 29, 2002. URL http://www.securityfocus.com/infocus/1540, March 2003

[28] Gasfinkel, S. and Spafford, G., Practical Unix and Internet Security, O'Reilly & Associates, Inc, 1996.

[29] Fowler, C.A., and Nesbit, R.F., Tactical deception in air-land warfare. Journal of Electronic Defense, Vol 18, No. 6 (June 1995), pp. 37-44 & 76-79

[30] James P Anderson, Computer Security Threat Monitoring and Surveillance, James P. Anderson Co., Fort Washington. PA, April 1980. URL http://csrc.nist.gov/publications/history/ande80.pdf, March 2003

[31] PhD Tina Bird, Marcus J. Ranum, URL http://www.loganalysis.org, March 2003

[32] Valter Monteiro, Neil C. Rowe, Independent Study, Naval Postgraduate School, Monterey, California, December, 2002

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, VA

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, CA

3. Neil C. Rowe
   Naval Postgraduate School
   Monterey, CA

4. J.D. Fulp
   Naval Postgraduate School
   Monterey, CA