Global Information Assurance Certification Paper

Jonathan Rose
GIAC Security Essentials Certification Track 1
Version 1.4b Practical option 1
Local Mentor class in Fairfax Virginia with Millie Ives

Turning the tables: Loadable Kernel Module Rootkits deployed in a honeypot environment

Table of Contents

Abstract

Honeypots are one of the latest technologies available to track and monitor hackers and Internet attackers. They can be generally divided into two different areas, production and research honeypots. Honeypots can also be classified by the amount of system interaction they provide to an attacker and therefore the risk that is involved. First, a very simple, low interaction, low risk honeypot, Back Officer Friendly, is discussed and tested. Next, a new generation of honeypot techniques are discussed, utilizing the advantages of loadable kernel modules for tracking hackers. Finally, and overview of the sebek honeypot system will be discussed focusing on the functionality, advantages and disadvantages of such a system.

Honeypots Overview

Honeypots are a relatively new tool in the field of computer security. A honeypot is a computer that is connected to the Internet in order to study various attacks that are directed towards it. The goal of a honeypot is to monitor the activity and techniques of an attacker without their knowledge. Honeypots were first designed to be bait for computer hackers, being easily attacked computers that intruders would focus on instead of attacking legitimate production servers, while alerting system administrators of the attack. Newer generations of honeypots are designed to monitor and investigate the methods and tools that hackers use, while studying their motives and learning their techniques. The challenge is to keep the hacker on the honeypot, allowing them to transfer tools and malicious code to the target machine, while limiting the hacker's attacks on other computer systems. If an intruder detects the honeypot, the project is compromised. At this point, the attacker may simply not return, deliberately misguide the researchers, or destroy the whole system. Honeynets are comprised of entire networks of honeypot machines, replicating an entire infrastructure of computers and network devices. Honeynets are designed to entice an attacker with more than a simple honeypot, giving them more systems to target. They can consist of any number of servers, or just a single host masquerading as multiple servers. The Honeynet Project describes them as "a tool to learn- specifically, the tools, tactics, and motives of the blackhat community".

A honeypot system is designed with several important parts that function together to monitor the action and techniques of Internet attackers. These parts consist of a network based intrusion detection system, host based intrusion detection systems, remote logging functionality, and keystroke logging. Access control devices must be in place to limit and manage the numbers of outbound connects made by the honeypot. This will limit the damage an attacker can have on other system from the honeypot. A system image should be created before the honeypot is connected to the Internet to build a baseline for comparison after

an intrusion. Computer forensic tools should be ready to analyze the system for deleted files, logs, or any other malicious activity done by the attacker.

Production and Research Honeypots

There are two main types of honeypots, production and research. Lance Spitzner, of the Honeynet project, differentiates between the two by stating, "Production honeypots protect an organization, while research honeypots are used to learn" (Spitzner 44). A production honeypot is designed to replicate a system, such as mail or ftp, to discover weaknesses in these systems designs without actually compromising the real systems themselves. In a production honeypot, the system administrator can watch which vulnerabilities a hacker exploits, allowing them to determine and fix weaknesses in their production systems. Research honeypots are used as learning tools and do not necessarily mirror the production systems. Research honeypots are used to gain information about the tools, methods, and weaknesses that attackers exploit. Research honeypots are a valuable tool for identifying tools and techniques that are not publicly available.

Honeypot Risk and Interaction

The level of user interaction can be used to classify honeypots, which directly affects the degree of risk associated with the honeypot. The higher the interaction of the honeypot, the greater the risk involved is. Before deploying a honeypot, it is important to determine what the goals of the honeypot are and the level of interaction involved will meet these goals, while considering the acceptable amount of risk.

Low risk, low interaction honeypots are easy to setup, yet they are only able to obtain a limited amount of information from the attacker. These honeypots work well if you are attempting to gather limited information on types of attacks and scans on the honeypot, without giving the intruder a full system to interact with. These simple honeypots can emulate services and detect automated attacks and port scanning. The problem with this style of honeypot is that it will not allow the researcher to capture any additional data associated with the attack other than the initial probe. While this may be sufficient for creating alerts and attack signatures, in depth analysis of the malicious code used in the attack would give the researcher a good deal more information on the internal workings of the attack.

High interaction, high-risk honeypots are at the other end of the spectrum for honeypot design. These generally give the attacker a complete system to interact with in an attempt to lure and monitor more sophisticated attackers. By giving an attacker a full system to access, the damage they can do is far greater than a low interaction honeypot. High interaction honeypots allow the researcher

to perform a much more detailed analysis of the attack, tools and methods used by the attacker. Tools can often be recovered from a high interaction honeypot, some even as source code. The risks associated with a high interaction honeypot are far greater than a simple, low interaction honeypot, yet the reward is much greater.

High interaction honeypots are much more difficult to install and configure, and much more challenging to maintain and monitor. "High-interaction honeypots are the extreme of honeypot technologies" (Spitzner 81). The difficulty is that if you are going to give an attacker full access to a system, an experienced hacker will be able to detect most honeypot monitoring mechanisms, or at least become suspicious of the system. This can lead to the disclosure of the honeypot system that could end with catastrophic results to the research being conducted.

Most high interaction honeypots are custom designed based on the goals of the research being done; i.e., what is to be captured. Generally these are normal system installations with the addition of specific software to monitor the action of the intruder. The software components on honeypots can be in the form of system patches, specialized applications or custom tools.

Back Officer Friendly – Low interaction Honeypots

An example of a low interaction honeypot is Network Flight Recorder's Back Officer Friendly (BOF). This is a tool that runs in the background on a computer and detects attempts to connect to various ports. BOF can be downloaded for free from http://www.nfr.com/products/bof/. BOF was installed and configured on a test machine to evaluate its effectiveness. The tool is small, simple to install, and provides very limited functionality. Currently it supports monitoring network ports for BackOrifice, FTP, TELNET, SMTP, HTTP, POP, and IMAP requests. It has the option to fake replies to these services, but only in a very inadequate manner.

A NMAP scan of a Windows 2000 Professional system running BOF displays the following output:

```
[root@localhost root]# nmap -sT -O 192.168.1.5

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Interesting ports on  (192.168.1.5):
(The 1591 ports scanned but not shown below are in state: closed)
Port       State       Service
21/tcp     open        ftp
23/tcp     open        telnet
25/tcp     open        smtp
80/tcp     open        http
110/tcp    open        pop-3
```

135/tcp open loc-srv
139/tcp open netbios-ssn
143/tcp open imap2
445/tcp open microsoft-ds
1025/tcp open NFS-or-IIS
Remote operating system guess: Windows Millennium Edition (Me), Win 2000, or WinXP

Nmap run completed -- 1 IP address (1 host up) scanned in 2 seconds.

As you can see, because BOF is emulating certain services on this host, they appear to be open. A NMAP UDP scan of the same hosts shows BackOrifice open on port 31337.

BOF immediately causes a windows pop-up message to be displayed on the target machine to report the unauthorized scanning activity.

Here is a telnet connection attempt made to the system running BOF.

[root@localhost root]# telnet 192.168.1.5 23
Trying 192.168.1.5...
Connected to 192.168.1.5.
Escape character is '^]'.

login: root

Password: backd00r

Login incorrect

The system fakes the telnet server to the client. Although the option to “fake replies” was set BOF, only the telnet monitor has any level of interaction, which is simply a login and password prompt. All other services display “Service Unavailable” when a connection to the port is initiated.

BOF log displays the information of the NMAP scan, and the attempted telnet login, capturing the login name and password used. BOF also captures another telnet login attempt, which used a long username and password, classifying it as a possible overflow attempt.

Sat Feb 01 08:39:29 Telnet login attempted from 192.168.1.1: user: root, password: backd00r
Sat Feb 01 08:39:29 possible overflow attempt via Telnet from 192.168.1.1 (recvd 127 bytes at Login:)
Sat Feb 01 08:39:29 possible overflow attempt via Telnet from 192.168.1.1 (recvd 230 bytes at Password:)

Sat Feb 01 08:40:42 SMTP connection from 192.168.1.1
Sat Feb 01 08:42:07 HTTP empty request from 192.168.1.1
Sat Feb 01 08:42:37 POP3 connection from 192.168.1.1
Sat Feb 01 08:42:57 SMTP connection from 192.168.1.1
Sat Feb 01 08:42:57 IMAP2 connection from 192.168.1.1
Sat Feb 01 08:42:57 HTTP empty request from 192.168.1.1
Sat Feb 01 08:42:57 POP3 connection from 192.168.1.1
Sat Feb 01 08:42:57 FTP connection from 192.168.1.1
Sat Feb 01 08:42:57 Telnet connection from 192.168.1.1
Sat Feb 01 08:43:09 IMAP2 connection from 192.168.1
Sat Feb 01 09:11:01 Back Orifice saw 0 bytes of garbage from 192.168.1.1

BOF also captured some interesting HTTP attacks while connected to the Internet. IP's are changed to protect the guilty.

Fri Jan 31 20:43:15 HTTP request from aa.bbb.ccc.ddd: GET /scripts/root.exe?/c+dir
Fri Jan 31 20:43:19 HTTP request from aa.bbb.ccc.ddd: GET /MSADC/root.exe?/c+dir
Fri Jan 31 20:43:23 HTTP request from aa.bbb.ccc.ddd: GET /c/winnt/system32/cmd.exe?/c+dir
Fri Jan 31 20:43:27 HTTP request from aa.bbb.ccc.ddd: GET /d/winnt/system32/cmd.exe?/c+dir

BOF is a very simple, easy to use tool, yet provides very little information about the attack itself.

Loadable Kernel Modules – High interaction Honeypots

One example of a high interaction honeypot is one that utilizes a loadable kernel module to monitor and report activities on the server, while letting the attacker complete access to the system. The concepts for loadable kernel modules in honeypot environments are based on loadable kernel modules rootkits that are currently used to hide activity on compromised systems.

Many times when a hacker successfully breaks into a computer, they install a program or set of programs known as a rootkit. These generally replace certain system applications to hide the presence of the intruder and allow for them to access the system in the future. A major disadvantage for the attacker with the standard type of rootkit is that they are easily detected using tools like Tripwire or AIDE, which creates a database that is able to insure the integrity of files on a system. Running Tripwire or AIDE on a compromised system will make it quickly apparent that a rootkit has been installed.

An advanced rootkit now being used by attackers is the loadable kernel module (LKM) rootkit. These rootkits take advantage of functionality in the linux kernel to

subvert the base operating system. To better understand these rootkits, we must first take a look at what LKM's are and how they function.

The kernel is the core of the operating system for Unix systems. The linux kernel, also Solaris and some BSD versions, allows for much of the kernel code to be compiled as modules rather than into the kernel itself. By doing this, the size of the kernel base is smaller; the kernel loads faster and uses less memory. When a specific functionality is required that is not built into the kernel, the kernel loads the required kernel module. These kernel modules become part of the operating system, registering their own system calls and functions. When the functionality is no longer required, the kernel can unload these modules.

When normal programs are run, they rely on system calls to the kernel to find the information they need. System calls open and close files, display system information, read and write to the file system. When a kernel rootkit is installed into the running kernel, it replaces certain system calls with its own system calls. With a malicious system call in place, the attacker has complete control over what the system call does.

One example of this is that when the 'ps' command is used in Linux, it returns a list of running processes. This list of running processes is found when the 'ps' command issues a system call to the kernel for process information. If a kernel rootkit is installed that intercepts this system call, it is able to hide specific programs from the returned list. The 'ps' command output will look normal. The MD5 checksums will match the initial Tripwire or AIDE database to show that the program was not modified. Even if other applications are executed that lists running processes, such as 'top', they will also fail to show the hidden process. Everything user level programs rely on is controlled in the kernel system calls that a malicious attacker has modified.

A new concept for honeypot tools is the use of LKM to track hackers. These are in essence a modified LKM rootkit designed to track and monitor all activity of an intruder while being virtually invisible to the intruder.

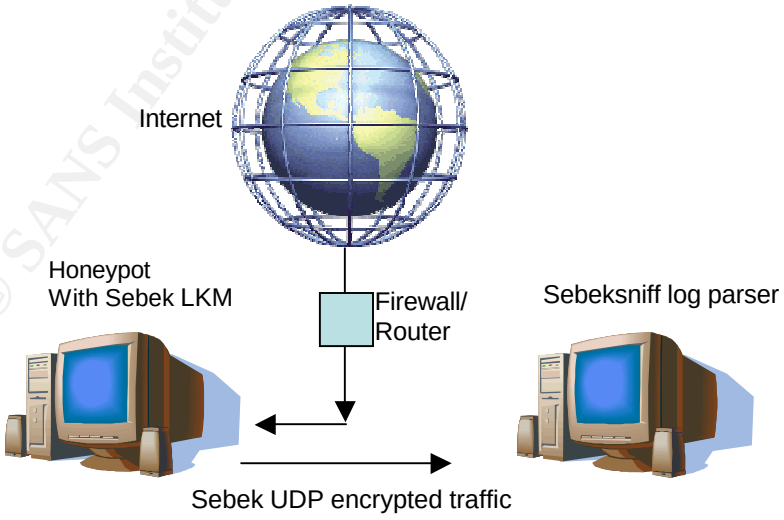
Sebek

Sebek is a system designed for use in honeypots to secretly track attacker's actions and keystrokes. The sebek system is comprised of three separate components that operate together to capture, transmit and analyze activity on a honeypot. Sebek advantages are that it is almost completely invisible and has the ability to capture and record encrypted secure shell commands (SSH) and secure copy (SCP) file transfers. The version of sebek used for this research is sebek version 0.4, written and maintained by Michael Clark and Ed Balas of the Honeynet Project. Sebek can be freely downloaded from http://www.project.honeynet.org/papers/honeynet/tools/sebek-0.4.tar.gz.

The main component of sebek is a LKM based on the Adore rootkit. The Adore rootkit implements file and process hiding through the use of patched system calls. This is used to conceal the activities of the sebek system. Sebek is able to capture and record encrypted session by intercepting unencrypted data while in kernel space through a patched read() system call. Unfortunately, Adore does not have any networking functionality built in to support remote logging. To compensate for the lack of networking in the Adore rootkit, the author of sebek had to create a separate program to handle the remote logging of data collected by the sebek LKM. The sebek device monitor (SDM) handles the remote logging for sebek. It reads from the sebek LKM and transmits this information onto the local area network (LAN). SDM uses a wide variety of techniques to make this remote logging traffic harder to detect. All traffic is transmitted onto the network using the user datagram protocol (UDP). The payload of each UDP packet is encrypted. Ethernet, internet protocol (IP) and UDP header information may be set to any arbitrary number to obscure the source and destination of the traffic. A variable amount of delay is used in between each packet that is transmitted. In cases where there is little or no network traffic, SDM can even be configured to transmit decoys onto the network to help mask it's own traffic.

A remote log host runs the sebek data capture utility, sebeksniff. This application captures all encrypted UDP packets, unencrypts these packets then logs them to a file on the log host.

The third component of the sebek system, sbdump, is a perl script written to read through the raw sebek logs created by sebeksniff and produce human readable output. This perl script is able to parse through the log files and extract all keystrokes and secure copy file transfers.



Example sebek configuration

Sebek Requirements

Sebek installation is simple and straightforward, as long as your system meets the requirements. Sebek currently only works on Linux, and the Linux kernel source must be installed. Unfortunately, sebek will not install on Redhat 8.0 (kernel 2.4-18-14) due to changes in the way the kernel handles loadable modules, so Redhat 7.3 (kernel 2.4.18-3) was used in the testing environment. Building the sebek system creates two tarballs, one for the honeypot named sebek.tar and one for the collector called sebek_collector.tar.

Running Sebek

A shell script called **sebek.sh** provides a configuration and execution control for the sebek components on the honeypot. This program is run on the honeypot to start the sebek system, which installs and hides the LKM and SDM on the machine.

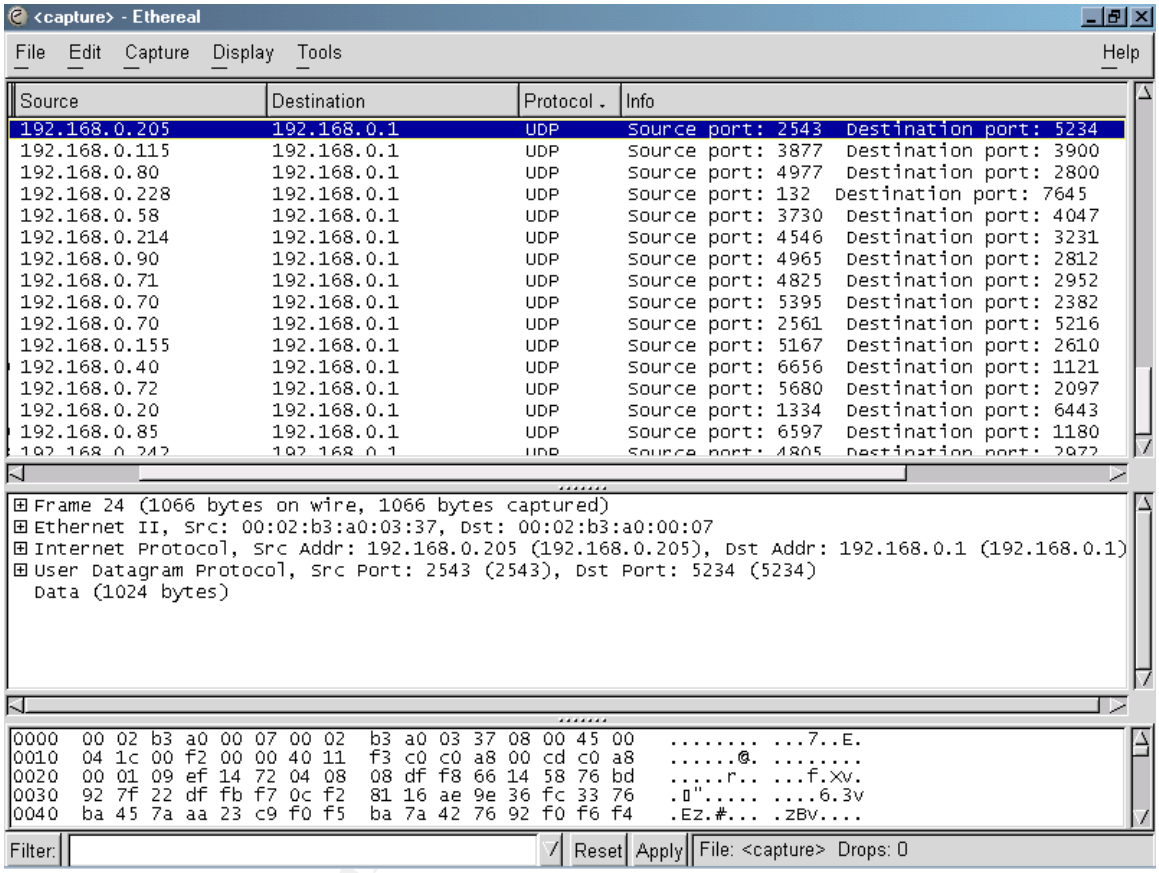
```
[root@localhost sebek]# ./sebek.sh start
File '/tmp/sebek/adore.o' hided.
File '/tmp/sebek/cleaner.o' hided.
File '/tmp/sebek/sdm' hided.
File '/tmp/sebek/ava' hided.
File '/tmp/sebek/sebek.sh' hided.
File '/tmp/sebek' hided.
File '/dev/sebek' hided.
Made PID 13308 invisible.
File './sebek.sh' hided.
```

On the collector machine, sebeksniff is executed with the proper command line options to begin capturing the sebek traffic.

```
[root@localhost collector]# ./sebeksniff -d eth1 -m 7777 -s testtesttest
Device: eth1
Magic: 7777
Sebek monitoring enabled
Sebek symmetric key: testtesttest
write 192.168.0.3:  602 bytes
write 192.168.0.3:  172 bytes
write 192.168.0.3:  41 bytes
write 192.168.0.3:  41 bytes
write 192.168.0.3:  41 bytes
write 192.168.0.3:  41 bytes
write 192.168.0.3:  41 bytes
```

Now the sebeksniff tool is up and running, capturing all traffic sent by the honeypot.

The following screenshot displays the network traffic for the subnet running the sebek system, which includes multiple encrypted UDP packets from spoofed addresses being transmitted to the log host on various destination ports.



Sebek Analysis

To test the functionality of sebek, it was installed on a test computer running Redhat 7.3. The sebek collector was installed on a different computer running Redhat 8.0. A third computer was used to simulate the attacker. All three computers were connected locally to an Ethernet LAN. With this setup, several different scenarios were tested. The first scenario involved logging onto the honeypot locally and running several commands. Next, a SSH connection was made to the machine to run some applications. A review of the captured data was done with sbdump. Sbdump lists the timestamp, userid, process name, tty, file descriptor, and data for each command that was run.

Output of sbdump without parameters:

Sbdump

This is a program to parse sebek logfiles, it is handy
sbdump [-c|-b|-s] filename

-c extracts data gathered character by character
-b extracts data gathered in bulk
-s extracts SCPed files recorded by sebek in the log

The first listing shows the output of sbdump when parsing for keystrokes.

```
[root@localhost collector]# perl sbdump -c 192.168.0.3
00:00:00-1970/01/01 [0:bash:1375:vc/2:0]clear
14:52:25-1971/06/11 [0:bash:1375:vc/2:0]cd
16:39:06-2003/02/17 [0:bash:1375:vc/2:0]whoami
16:39:13-2003/02/17 [0:bash:1375:vc/2:0]httpd -v
15:15:05-1971/06/11 [0:bash:1667:pts:tail /var/log/mess
17:03:14-2003/02/17 [0:bash:1667:pts:0]ps -ax
17:03:36-2003/02/17 [0:bash:1667:pts:0]lsmod
17:04:52-2003/02/17 [0:bash:1667:pts:0]exit
15:20:51-1971/06/11 [0:bash:1792:ptnslookup
17:08:12-2003/02/17 [0:bash:1792:pts:0]exit
```

First, a user that is already logged on runs a few commands, clear, cd, and whoami. This is all from the shell interpreter bash on the server. The following entries with pts indicate that they were being sent remotely, in this case through SSH.

```
[root@localhost collector]# perl sbdump -c 192.168.0.3
00:00:00-1970/01/01 [0:bash:1375:vc/2:0]clear
14:52:25-1971/06/11 [0:bash:1375:vc/2:0]cd
16:39:06-2003/02/17 [0:bash:1375:vc/2:0]whoami
16:39:13-2003/02/17 [0:bash:1375:vc/2:0]httpd -v
15:15:05-1971/06/11 [0:bash:1667:pts:tail /var/log/mess
17:03:14-2003/02/17 [0:bash:1667:pts:0]ps -ax
17:03:36-2003/02/17 [0:bash:1667:pts:0]lsmod
17:04:52-2003/02/17 [0:bash:1667:pts:0]exit
15:20:51-1971/06/11 [0:bash:1792:ptnslookup
17:08:12-2003/02/17 [0:bash:1792:pts:0]exit
[root@localhost collector]#
```

Viewing the raw data from the connection log gives an insight into how the SSH program runs. The log shows the files SSH processes, first of which is /etc/hosts.allow, then /etc/hosts.deny. Next SSH calls PAM to verify the password and username. This information is fairly easy to spot by viewing the raw, unparsed sebek lag created by sebeksniff.

SCP file transfer recovery

Due to the way the Sebek system captures all calls read() system call, it is also able to capture Secure copy (SCP) file transfers to and from the honeypot. Remote SCP of files to the honeypot will show up in the logs with the date and time, program, program data, filename, and the size of data transferred. When an intruder SCP a file to another remote machine from the honeypot, we are able to capture the file as well as the password used to authenticate to the remote server.

Sebek Disadvantages

Although the sebek system represents an advancement for host based honeypots, there are several areas of the system that could be improved to increase the efficiency, reduce detection, and create a better honeypot. One obvious problem detected during the testing of sebek was the error message logged to syslog during the startup of SDM:

localhost kernel: sdm uses obsolete (PF_INET,SOCK_PACKET)

This error message can quickly and easily pinpoint this computer as a honeypot. A better solution would be to get rid of the SDM application altogether. Functionality could be included into the sebek LKM that would hook into the network layer, allowing all sebek related network traffic to remain completely hidden from the sebek machine. This would eliminate the need for a separate process to handle the data transfer. Currently, a skilled attacker would probably become suspicious of encrypted, spoofed UDP packets constantly being broadcast onto the network. Sebek could also be redesigned to work with Redhat 8.0 and later versions of the linux kernel, as well as expanded to other flavors of UNIX. Also, chkrootkit is able to detect the current version of sebek, giving intruders another method to determine if the system is running as a honeypot.

Conclusion

The use of honeypots is becoming more extensive in the computer security industry. Simple, low interaction honeypots do not provide the amount or detail of information that is needed to fully understand the motives and methods of hackers, requiring new honeypot technologies to be developed and deployed. A high interaction honeypot like sebek is able to capture a large amount of data, including encrypted communications, through patched system calls from loadable kernel modules. The sebek system is able to communicate encrypted log data to a remote host for analysis through a highly customizable reporting program. Although the sebek system does have some disadvantages and may be detected, the concept of using loadable kernel modules in a honeypot environment is innovative and should be expanded and improved upon.

References:

Barnett, Ryan C. "--[Honeypots]-- Monitoring and Forensics" URL:http://honeypots.sourceforge.net/ (8 February, 2003).

Gubbels, Kecia. "Hands in the Honeypot" November 3, 2002 URL:http://www.sans.org/rr/intrusion/hands.php (8 February, 2003).

Dai Zovi, Dino. Kernel Rootkits July 4, 2001 URL:http://www.sans.org/rr/threats/rootkits.php (8 February, 2003).

Johnson, Benjamin**. "**My Project – Sebek" URL**:**http://mailman.cs.uchicago.edu/pipermail/cs22800/Week-of-Mon-20020923/000008.html (10 February, 2003)

Miller, Toby. "Detecting Loadable Kernel Modules (LKM)" URL: http://www.incident-response.org/LKM.htm (10 February 2003).

Spitzner, Lance. "Honeypots, Tracking Hackers". Boston: Pearson Education, Inc, 2002.

Martin, William W. "Honey Pots and Honey Nets - Security through Deception" URL:http://www.sans.org/rr/attack/deception.php May 25, 2001

Honeynet Project, The. "Know your Enemy: revealing the security tools, tactics, and motives of the blackhat community". Indianapolis, IN: Addison-Wesley, 2002.

Whitehatinc.com. "NFR BackOfficer Friendly" URL: http://www.whitehatinc.com/nfr/bof.html (9 February 2003).

Honeynet Project, The. "Honeynet Project" URL: http://project.honeynet.org/ (9 February 2003).

AIDE. "AIDE" URL: http://www.cs.tut.fi/~rammer/aide.html (9 February 2003).

Pragmatic / THC. "(nearly) Complete Linux Loadable Kernel Modules, ver 1.0" URL:http://packetstormsecurity.nl/docs/hack/LKM_HACKING.html 03/1999.

Henderson, Bryan. "Linux Loadable Kernel Module HOWTO" URL: http://www.tldp.org/HOWTO/Module-HOWTO/. 21 May 2002.

Rd. "Writing Linux Kernel Keylogger" URL:http://packetstormsecurity.nl/papers/unix/writing-linux-kernel-keylogger.txt June 19th, 2002.

Plasmoid / THC *"Attacking Solaris with loadable kernel modules"* - Version 1.0 (c) 1999 URL: http://www.cs.ucsb.edu/~jzhou/security/slkm-1.0.html