Building a Heterogeneous Honeynet

Javier Prado
jprado1@swarthmore.edu

Heather Jones
hjones2@swarthmore.edu

December 13, 2005

Abstract

A small network of honeypots, each running a different operating system, was constructed inside the campus network at Swarthmore College. Although preexisting honeynet software was used, quite a few difficulties were encountered. These problems, and solutions to those that were solved, are presented in this paper.

1 Introduction

It is critical for any misuse intrusion detection system to know how to be able to detect the latest attacks. With new attacks emerging every day, this can be quite difficult. One tool developed to assist in keeping pace with the attackers is a honeypot: a computer with no purpose other than to be attacked and collect data on the attacks it suffers. A honeynet is an entire network used in this way. Because so many of the attacks in use today involve an intruder gaining control of a computer and then using it to attack other machines, a honeynet operator must make sure that his honeynet cannot only collect data effectively, but can also stop his machines from being used to launch further attacks. This makes the task of building a honeynet from scratch very difficult. Fortunately, there are a number of tools already available to capture and control traffic on a honeynet. The most important contribution, perhaps, is the Honeynet Project's Honeywall Roo, which combines several honeynet technologies onto a bootable CDROM, including the network packet sniffer Snort, a server for the host-based data capture module Sebek, the ability to limit traffic from the honeynet, and a web GUI for administration and data analysis. The idea of the Roo CDROM is to make it easy for any organization to set up its own honeynet.

We set out to build a honeynet using the Honeywall CDROM Roo in order to research the different kinds of attacks that occur on different operating systems. We found, however, that setting up a honeynet is not as easy as it looks. Some of our problems were due to bugs in various parts of the system. Others occurred when we failed to understand key details about how the system was supposed to work. While the latter could be dismissed as due entirely to our inexperience, we believe they provide valuable information about what the makers of the CDROM Roo have assumed that their users will know.

First, we will provide some background about the Honeynet Project, other research on attacks, and details of how the Honeywall Roo works. Next, we will discuss the architecture of our honeynet and the problems we encountered with some of the components of the system. Finally, we will draw conclusions about what we have learned from this experience.

2 Background

2.1 Honeypots and Honeynet.org

Beginning in 1999, the Honeynet Project has been developing and using honeynets to collect and distribute knowledge about the black-hat community [6]. The first phase of the project

Proceedings of the Class of 2006 Senior Conference, pages 9–18,

involved testing the idea that data captured by honeynets could provide useful information about attacks. In the second phase, which began in 2002, a second generation of honeynet technology (Gen II) was developed to be simpler, more interactive, safer, and easier to deploy than previous architectures. The third phase packaged everything necessary to set up a honeynet onto a bootable CD-ROM. This has made it much easier for organizations around the world to set up their own honeynets. The fourth phase involves the development of a centralized data collection system. As the project has progressed, it has expanded to become the Honeynet Research Alliance, with 20 member groups around the globe.

We are using the Honeynet Project's Gen III technology to implement our honeynet. Significant improvements made between Gen II and Gen III include a more automated installation, a web GUI for administration, and a capability for automatic updates [9].

A similar experiment to the one we initially set out to do here was performed as a part of the Honeynet Project. They compared data from Windows, Linux, Solaris and OpenBSD machines, and discovered that each operating system attracted different kinds of attacks. On Windows systems, they saw worms and automated attacks. On Linux systems, they saw attacks originating mainly from Eastern Europe, especially from Romania, that exploited known vulnerabilities or employed automated tools. On Solaris and OpenBSD, they saw more advanced or interesting attacks [6].

2.2 Related Work

In a 2005 paper entitled "A Pointillist Approach for Comparing Honeypots," Pouget and Holz examined how a three-machine honeynet running Windows 98, Windows NT Server and Redhat Linux 7.3 server could be used to make a low-interaction honeypot emulating these three machines more believable [2]. They classified attacks into three categories: Type I attacks targeted only one machine, Type II attacks targeted two out of three machines, and Type III attacks targeted all three machines. Approximately 60 percent of the attacks they recorded were of Type I, and 35 percent were of Type III. Of the few Type II attacks, 88 percent were judged to be Type III attacks for which the message to one of the machines had been lost, 9 percent were due to scanning attacks that targeted every other IP address, and 3 percent were believed to be attacks on the two Windows machines only.

Yegneswaran, Barford and Ullrich used data from 1600 networks around the world to study the global characteristics of internet attacks[11]. In addition to analyzing the volume and distribution of attacks in their 2003 paper "Internet Intrusions: Global Characteristics and Prevalence," they presented a classification of four scan types. A vertical scan examines several different ports on a single machine. A horizontal scan examines the same port on several different machines. A coordinated scan examines the same port on the machines within a subnet and originates from several sources. A stealth scan is a horizontal or vertical scan executed with a very low frequency in order to avoid detection.

Weaver, et al. develop a taxonomy of computer worms based on method of target discovery, carrier, activation, payload, and attacker motivation in their 2003 paper on the subject[10]. Following this taxonomy, target discovery techniques are scanning, pre-generated target lists, external target lists, internal target lists and passive discovery. Carriers include the worm itself, a second channel through which the worm completes an initiated infection, and normal traffic in which the worm embeds itself. A worm can be activated by a human, by a human activity, by a scheduled process, or by itself. The most common payload for a worm is a nonexistent or nonfunctional one, although there are many other possible payloads, including electronic or physical remote control, damage, or denial of service. Attackers may be motivated by experimental curiosity, pride, commercial advantage, criminal gain, random protest, political protest, terrorism, or cyber warfare.

2.3 Honeynet Details

The Know your Enemy series of documents made available by the honeynet.org website has proved to be an invaluable resource. The seminal paper for any honeynet project is the Know your Enemy: Honeynets paper [8]. This paper gives a survey of the basic, fundamental concepts of a honeynet. The two fundamental aspects of a honeynet as they apply to our project are the data control concept and the data capture concept. In data control, the key concept is that it is imperative to circumvent an intruder's ability to attack other systems outside of the honeynet once the intruder has compromised a honeypot within the honeynet. This responsibility primarily rests on the honeywall's ability to detect malicious activity on the network and in each of its honeypots, and then respond in turn to the threat at hand. The paper makes clear that the actual implementation of the data control (our honeywall) is ultimately our decision but the author does provide some suggestions for our implementation. Among the suggestions include the layering of security measures to help obfuscate the presence of the honeywall and for the honeywall to operate in a fail closed manner where any failure of one of our components will result in all network traffic from the honeynet being terminated [8]. The other fundamental concept of a honeynet, data capture, concerns the logging and reporting of an intruders activity, basically the reason why the honeynet exists. For our project, the two primary elements of data capture are the Sebek client and the honeywall's log of network activity.

Because the main goal of this project was to get the honeywall to the point where it could collect useful data, the Sebek client is one of the fundamental components of our project. Sebek is a solution to a problem that has faced the honeynet community: how to observe an intruder's actions without the intruder knowing that he or she is being monitored. The paper discussing this issue [7] shows how simple network monitoring, although a viable solution, is undermined by an intruder's ability to use encryption to obfuscate his or her activity. The only way to capture unencrypted data is to catch it before it is encrypted (i.e., before the attacker sends it onto the network), or after it is decrypted (i.e., once it has reached its final destination). Thus, Sebek provides information on an attacker's actions from the attacked host. In this way, data on the attacker's keystrokes, processes run, files opened, and other system activities can be recorded. The key feature of the Sebek client is that it is incorporated into the kernel of the operating system [1]. Since it is a part of the kernel, it is able to effectively hide from the intruder. Conceptually, this article [8] describes that the user space and the kernel space are mutually exclusive. This being the case, having the Sebek client become a part of the kernel is a solid way of hiding the data capturing component of a honeynet. The Sebek client then stores in a buffer the recorded actions of an intruder, encrypts the information, and sends it to the honeywall to which the honeypot is connected. Because the Sebek client is installed on the attacked host, it provides information about what happens on the host which could not necessarily be gathered from an examination of the network traffic to and from the host, even if this traffic could be decrypted. Thus, Sebek provides a valuable complement to the network data collected by the honeywall[7].

While making the Sebek client a part of the operating system helps to keep it hidden from the attacker, it also creates a problem when honeypots with different operating systems are used. As Windows is even more different from Linux than FreeBSD is from Solaris, we need to ensure that each Sebek client we install is appropriate

[1] Sebek was originally adapted from a blackhat rootkit called Adore [1]. Such rootkits were developed to change the behavior of the kernel without revealing their existence. Recently, Sony has developed a rootkit aimed to enforce usage rules on copyrighted materials [3]. This has produced sharp criticism, not only from advocates of privacy, but also from Windows experts who claim the code is poorly written and can compromise the security of the Windows operating system.

for the operating system that it is going to be installed on. The Honeynet Project has completed much of this process of adapting the Sebek client to each operating system environment. For the Windows environment, the developers have gone so far as to provide a simple executable which automatically incorporates the Sebek client into the kernel of the Windows operating system and leaves the incorporated Sebek client to be customized by the user. For FreeBSD, a loadable kernel module of the Sebek client is available. For Linux, the kernel must be rebuilt to incorporate Sebek. Information regarding the actual Sebek client implementation exists on the Sebek homepage within the honeynet.org website [4].

3 Our Own Honeynet

We wanted to examine the differences between the attacks seen on different operating systems, so we decided to set up a Honeynet with several computers running different operating systems. We chose to use the Honeynet Project's Honeywall Roo bootable CD-ROM, believing that this would make the setup portion of the project virtually effortless. Unfortunately, this did not turn out to be the case. Instead, we found that the setup became a project in itself. What follows is a description of our honeynet, the problems we encountered, how we solved these problems, and what still needs work.

3.1 Honeynet Architecture

Our honeynet consisted of five machines: one running the Honeynet Project's Honeywall Roo (including a Fedora core), one running RedHat Linux, one running Windows XP, one running FreeBSD, and one running Solaris. Because there was no version of Sebek for Solaris that was compatible with the Gen III honeywall, we planned to simply examine the traffic going to and from the machine. After weeks of crawling around behind our computers plugging and unplugging various cables, we obtained a KVM switch to allow us to easily interact with any of the machines except the Solaris using the same monitor, keyboard and mouse. Our honeynet was located within Swarthmore College's campus network; we will refer to this external network as the internet, although being inside of this network did present problems that we would not have seen had we actually been directly connected to the internet. All the individual honeypots were connected to the honeywall through a LanTronix 10 base-T ethernet switch (See Figure 1).

3.2 The Honeywall

The honeywall was a Dell Precision 330 with an Intel Pentium 4 processor. We installed two additional network cards, allowing us two ethernet connections to the internet one as a route to the honeypots and one for remote management of the honeywall and one ethernet connection to our own honeypot network.

We set up the honeywall to run in bridge mode as opposed to NAT (Network Address Translation) mode (See Figures 2 and 3), allowing each honeypot to have its own IP address because we wanted each to appear as a distinct target. We initially had only two network cards because the Honeywall documentation indicated that a third network card was only necessary if remote administration was desired. Yet, this simple setup was not functional because our honeywall software kept trying to find a third network connection. This prevented us from completing a full installation of the honeywall, so we installed a third network card. After this installation, we needed to provide an IP address for honeywall's configuration utility. Since we were behind Swarthmore College's network, we could not simply set an IP address arbitrarily. Because the honeywall intallation did not include a DHCP client or any other similar IP configuration program, we could not determine the IP address from the honeywall machine. We solved this problem by using the live Linux CD Knoppix. This allowed us to run dhclient to get an

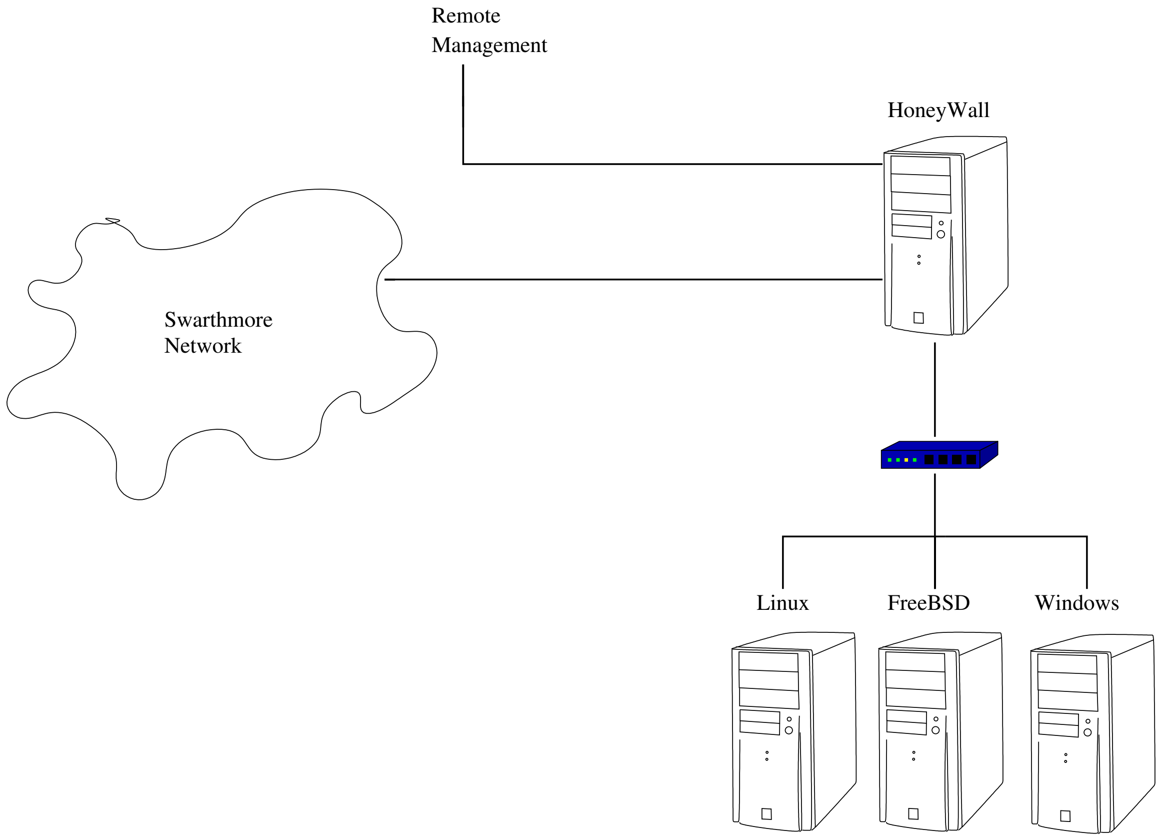

Figure 1: Diagram of Honeynet Architecture

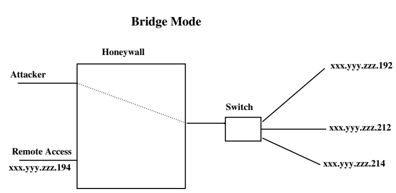

Figure 2: Diagram of bridge mode configuration

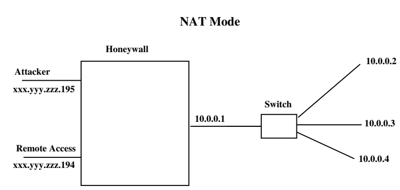

Figure 3: Diagram of NAT mode configuration

IP address which we could then enter into the configuration utility.

One of the main improvements that came with the Generation III honeywall technology was the web interface, Walleye. This GUI allows the honeynet administrator to interact with the honeywall in a more user-friendly environment than the text-based menu utility that can be run on the machine. Using this interface, we were able to observe a small amount of traffic on coming to and from our honeypots mostly bootp, domain or NetBIOS traffic, plus any traffic we created ourselves while testing the system. Supposedly, Walleye identifies what operating system each machine is running. While this occurred successfully for some of the machines connecting to our honeypots, it initially did not work for the honeypots themselves for some unknown reason.

On two occasions, we found that we could not log in to Walleye at all. After we entered in a username and password, the system would begin the login process but load so slowly that it never got to the welcome page. We then logged into the honywall directly to investigate what might be wrong. Reloading the or even rebooting the honeywall did not fix the problem. Eventually, we realized that the directory where the honeywall was storing its data had become full. The first time, moving the data to a directory with more free space fixed the problem. The second time this problem occurred, we realized that we needed more space than was available on the honeynet's hard drive, so we dumped all the data to another machine. We then used the menu utility to clean out the logging directories. In doing so, we noticed an interesting message: apparently the honeywall had not yet been programmed to clean out its MySQL database files. Since the honeywall was still failing to start MySQL, which it uses to make its collected data available to view over the internet and manage the users for Walleye, we initially tried removing several database files that were particularly large, thinking that perhaps the overly large size of the database was preventing it from loading. This did not fix the problem, however, so we restored the files from the external machine to which we had previously dumped all our data. After re-running the startup script for the MySQL daemon, we investigated the error produced. The error itself was not very informative, but it did point us to a log file, which indicated that one of our database tables may have been corrupted. A quick Google search on repairing MySQL tables revealed the program myisamchk. This successfully repaired the table, but then the startup script complained about the next table. After running the program on all of the tables in the database, we were finally able to start the MySQL daemon and successfully log in to Walleye. Looking back at this problem, we found it surprising that the amount of data generated was large enough to cause significant issues with the honeywall machine, since we saw only background noise and test traffic, and no actual attacks. Clearly, for a honeynet that is being used to collect data for research, even for a relatively short term project, the issue of where and how old data should be stored would have to be very carefully considered.

The simplest way to examine traffic on the honeynet is to search for connections by date, time, IP address, and or port from the Walleye welcome page. The results can either be returned as a PCAP file or in "Walleye flow view," the latter being much easier to interpret. At the beginning of the month of December, however, we began encountering an error whenever we tried to see results in Walleye flow view. The error stated that the month "12" was out of the range 0..11, and indicated a file on the honeywall where the problem had occurred. An examination of this file was not enlightening, as it was written in Perl script, with which we have no experience. A search of the Bugzilla bug information server on Honeynet.org produced no results. We tried running YUM (Yellow-dog Updater, Modified) again to get the latest updates for the honeywall, but the only change we observed was that the graph of recent activity on the honeywall, usually displayed on the Walleye welcome page, was no longer loading correctly. By this time, we were several days into the month of December, and another search of the Bugzilla server produced one record. This linked to a possible solution: change one number in the file on the honeywall that had been causing problems [5]. We tried this, and sure enough, it solved our problem. The author of the tip admitted that he did not know what other effects this change might have, but we have not as yet observed any negative repercussions. We did, however, notice that in some cases, Walleye could now recognize the operating system of our Linux and Windows honeypots. We are still unsure why this feature only sometimes works: it does not seem to have any correlation to the type of connection being monitored. We also noticed that when we try to see what packets Snort has dropped using Walleye's "System Status" feature, we see a message saying that no packets have been dropped that day, while checking the same field using the menu utility does display information on a number of packets that have been dropped.

Since data control is a very important part of any honeynet, we wanted to do a quick test of our ability to control the traffic leaving our honeynet. "Roach Motel mode" allows the honeynet operator to allow traffic in but prevent any traffic out. We tried a test that involved using ssh to log in to a honeypot and then ping a machine outside the honeynet. As expected, this test succeeded fully in normal mode, but in Roach Motel mode the user could log in to the honeypot but the ping command did not succeed. We then examined the "Emergency Lockdown" feature. Supposedly, this prevents any traffic in or out of the honeywall except through the management interface. We found that turning on this option did stop an ssh connection in progress to one of our honeypots, and turning off the option allowed this connection to resume. Unfortunately, all connections to the management interface also failed to work with the lockdown option on. This may be an issue unique to our configuration, since we have our honeypots and our remote management interface on the same Class-B network, (the Swarthmore College network), or it may be a problem for any setup. In either case, it would make a situation where a honeynet is monitored exclusively through the remote management interface impossible.

3.3 Linux

The Linux machine was a Dell Precision 330 with an Intel Pentium 4 processor. We initially had the machine running RedHat 7.2 with a 2.4.9 Linux kernel. The current version of Sebek for Linux 2.4 worked with the 2.4.30 kernel. This posed a problem. We updated the kernel but then we discovered that this updated kernel was not completely compatible with the version of Redhat that we were using. This incompatibility manifested itself in the system's inability to recognize the ethernet card. Since internet connectivity is an essential part of a honeynet, we decided to install a more recent version of RedHat. We installed RedHat 9, updated to a 2.4.30 kernel, and installed the Sebek module. This version of Sebek appears to be functioning correctly: packets are dropped and logged at the honeywall, and examining these packets through the web interface shows that individual commands are being recognized.

We were able to get an IP address on this machine by running dhclient, but we could not ping anything beyond our honeywall or reach any sites with a web browser. We determined that this problem must be due to a setting on the honeywall, but we were unable to locate the relevant field in the menu configuration utility. Then, when examining the configuration page on the Walleye web GUI, we found the source of the problem: the "honeypot public address(es)" field had been set to the IP of the external NAT mode interface of the Honeywall, and the honeywall was not in NAT mode. After entering in the specific addresses of the honeypots, we were able to connect to the network beyond our honeywall.

3.4 FreeBSD

The FreeBSD machine was a Dell OptiPlex GXa with an Intel Pentium 2 processor. We had the machine running FreeBSD 5.2.1. We loaded the FreeBSD version of Sebek 3.0.3 onto the machine, and we have observed packets being sent out, but they are considered by the honeywall to be malformed. The current FreeBSD version of Sebek 3.0.3 is still in testing, so we are not sure whether there is a problem with the program or with the way we have it set up. By running dhclient, we were able to get an IP address on this machine, however, we soon began getting error messages that this IP address was not unique. Returning several days later, we found an error message claiming that the machine's IP address was 0.0.0.0 and was already in use. Adding the IP address the machine had obtained previously to the /etc/hosts file resolved this problem temporarily, but it soon resurfaced. We attempted to run dhclient again to obtain a new IP, but the program seemed to be malfunctioning: it ran for several minutes without producing any output. At this point, the network problems of our FreeBSD honeypot are still unresolved.

3.5 Windows

The Windows machine was a Dell Precision 330 with an Intel Pentium 4 processor. The machine was running Windows XP Professional with no service packs or updates installed. We loaded the newly available Windows version of Sebek 3.0.3 onto the machine, and it appeared to be working. When we tried to connect the machine to the internet, however, it began to spontaneously reboot. After a few cycles, it got to the point where it booted up to the login screen and immediately began booting again. We initially assumed this must be due to a virus, but when we examined the network traffic, we were unable to see any connections that may have allowed a virus to reach the machine. We thus concluded that the Sebek client itself was causing the computer to malfunction. We reinstalled the operating system and the Sebek client. This time we saw only a single spontaneous reboot, and after that the machine appeared to function normally. We did see occasional error messages upon login, but we were unable to decipher them, as they were formatted to be sent to Microsoft and not to be readable on the machine. At this point we could see that the machine was sending Sebek packets in response to typed commands, but again the honeywall reported that these packets were malformed. Although this machine did get an IP address, it also could not connect to the internet with a web browser until the aforementioned change to the honeywall configuration (See section 3.3).

4 Conclusion

We have succeeded in building a small honeynet with one fully functional and two partially functional honeypots. Our Linux honeypot collects data using the Sebek client and the honeywall correctly interprets this data. Our Windows honeypot sends improperly formed Sebek packets that cannot be decoded by the honeywall. Our FreeBSD honeypot was previously able to send malformed Sebek packets, but now has problems holding a valid IP address. Our honeywall can limit or fully block outgoing traffic as desired, although a complete system lockdown seems to also block the management interface.

In doing this project, we discovered that the creators of the Honeywall Roo package had made certain assumptions about the knowledge base that their users would have. Those building a Honeynet are expected to know how to set up a network. This may seem obvious, but researchers in a field other than networking, (computer security for example), may have an interest in setting up a honeynet but no previous experience with networks. The honeynet configuration utility also seems to be written for people who are in control of the network on which they are placing their new honeynet. Because we were on Swarthmore College's campus network, this was not the case for us, and we had to briefly run another operating system on our honeywall machine just to get an IP address. In addition, it is assumed that those installing Sebek onto a honeypot know how to work with the kernel of that honeypot's operating system. Installation onto the Windows machine was very simple, although ultimately the installed module did not function correctly. Getting Sebek onto the FreeBSD machine involved inserting a kernel module and stripping the kernel, and this program also did not work as it should have. The Linux version of Sebek also required loading in a kernel module. Because the kernel on the machine we had was significantly older than the version that the Sebek module expected, we had to update the kernel - something we had to learn how to do. The instructions for installing Sebek on Linux also mention that the module will be uninstalled every time the machine reboots unless the module is loaded in a startup script. We have never worked with startup scripts, so this is a task that has not yet been completed.

We also encountered problems specific to the honeynet itself. In less than two months, even with no actual attack traffic being observed, the honeynet can fill up the directory it uses to store data. This causes problems with Walleye or any other program that uses the database of captured traffic. In some cases, this can even cause the tables in the honeywall's MySQL database to become corrupted. Running the program myisamchk can repair these tables, but this takes a significant amount of time with large tables. Also, the honeywall's menu utility command to clean out the logging directories does not clean the MySQL database files. If these become large enough to cause problems even without being stored in the same directory as other large data files, some method for cleaning them out manually would have to be found. In view of the serious problems that large amounts of data can cause, a honeynet operator should have a location to which old data can be transferred for storage and do this transfer frequently.

The Sebek modules for Windows and FreeBSD produce malformed packets. We are unsure whether this problem is specific to the way in which we have the modules configured on our honeypots or whether it would occur for any configuration. In any case, hopefully the next revision of the module for each operating system will include a fix for this bug.

We encountered a problem in the Walleye interface that prevented it from displaying any traffic for the month of December. This was fixed by making a slight modification to a Perl script file on the Honeynet, (See [5]).

Within the "System Status" section of the Walleye System Administration feature, Snort alerts fail to show up, even if they have occurred and are clearly visible via the menu utility on the the honeywall itself. We have not done a thorough investigation of the other options within this section, since most of our focus was on getting the basic features of our honeynet up and working, but this is definitely an area for future work.

During an "Emergency Lockdown," the management interface also appears to be blocked. We are unsure whether or not this problem is specific to our setup, (our management interface is on the same network as the public interface to our honeynet), but it is an important area for further exploration, since it affects whether or not a honeynet can be managed entirely through remote access.

We recognize that the Honeywall Roo is still a relatively new technology, and as such will have many bugs. We hope that this record of our experiences with the software will be helpful both to other researchers beginning a honeynet experiment of their own and to the developers at the Honeynet project who are constantly working to improve the system.

5 Acknowledgements

Thanks to Ken Patton for getting the Honeywall installation started and for being an essential team member throughout this project. Thanks to our sysadimin Jeff Knerr for tolerating us, giving us equipment, feeding us candy, and providing considerable knowledge and assistance. Thanks to the CS97 class for all the help and moral support. Most of all, thanks to Professor Ben Kuperman for inspiring us to try this project and for providing invaluable help every step of the way.

References

[1] Bill McCarty. The honeynet arms race. *"IEEE Security and Privacy"*, pages 79–82, December 2003.

[2] "Fabien Pouget and Thorsten Holz". A pointillist approach for comparing honeypots, 2005. URL www.honeynet.org/papers/individual/DIMVA2005_Pouget_Holz.pdf.

[3] Bruce Schneier. The real story of the rogue rootkit, November 2005. URL http://www.wired.com/news/privacy/0,1848,69601,00.html.

[4] Sebek Homepapge. URL http://www.honeynet.org/tools/sebek/.

[5] Jaime Sotelo. Get an error with walleye and fixed it, 2005. URL http://www.securityfocus.com/archive/119/418383/30/0/threaded.

[6] Lance Spitzner. The Honeynet Project: Trapping the Hackers. *IEEE Security & Privacy*, pages 15–23, April 2003.

[7] "The Honeynet Project". URL http://www.honeynet.org/papers/sebek.pdf.

[8] The Honeynet Project, May 2005. URL http://www.honeynet.org/papers/honeynet.

[9] The Honeynet Project. Honeynet project overview, 2005. URL http://honeynet.org/speaking/honeynet_project-3.0.1.ppt.zip.

[10] Nicholas Wever, Vern Paxson, Stuart Staniford, and Robert Cunningham. A taxonomy of computer worms. *WORM*, 2003.

[11] Vinod Yegneswaran, Paul Barford, and Johannes Ullrich. Internet intrusions: Global characteristics and prevalence. *SIGMETRICS*, 2003.