A METHODOLOGY FOR INTELLIGENT HONEYPOT DEPLOYMENT

AND ACTIVE ENGAGEMENT OF ATTACKERS

By

Christopher R. Hecker

RECOMMENDED:

Advisory Committee Co-Chair

Advisory Committee Co-Chair

Chair, Department of Computer Science

APPROVED:

Dean, College of Engineering and Mines

Dean of the Graduate School

August 5, 2012

Date

A METHODOLOGY FOR INTELLIGENT HONEYPOT DEPLOYMENT AND ACTIVE ENGAGEMENT OF ATTACKERS

A

DISSERTATION

Presented to the Faculty

of the University of Alaska Fairbanks

in Partial Fulfillment of the Requirements

for the Degree of

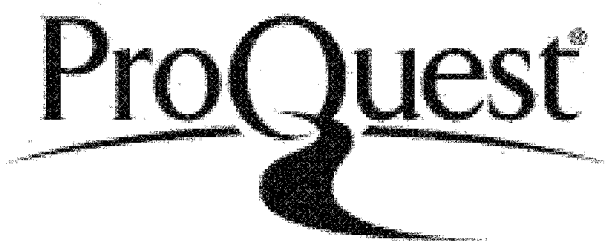
DOCTOR OF PHILOSOPHY

By

Christopher R. Hecker, M.S.

Fairbanks, Alaska

August 2012



ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Abstract

The internet has brought about tremendous changes in the way we see the world, allowing us to communicate at the speed of light, and dramatically changing the face of business forever. Organizations are able to share their business strategies and sensitive or proprietary information across the globe in order to create a sense of cohesiveness. This ability to share information across the vastness of the internet also allows attackers to exploit these different avenues to steal intellectual property or gather information vital to the national security of an entire nation. As technology advances to include more devices accessing an organization's network and as more business is handled via the internet, attackers' opportunities increase daily. Honeypots were created in response to this cyber warfare. Honeypots provide a technique to gather information about attackers performing reconnaissance on a network or device without the voluminous logs obtained by the majority of intrusion detection systems. This research effort provides a methodology to dynamically generate context-appropriate honeynets. Administrators are able to modify the system to conform to the target environment and gather the information passively or through increasing degrees of active scanning. The information obtained during the process of scanning the environment aids the administrator in creating a network topology and understanding the flux of devices in the network. This research continues the effort to defend an organization's networks against the onslaught of attackers.

Table of Contents

List of Figures

Page

List of Tables

Page

List of Appendices

Acknowledgments

This research effort would not have been possible without the support, encouragement and guidance of many people. I would like to extend my appreciation to the following people for their support during this endeavor.

I want to give God the honor and praise for blessing me with the opportunity to pursue this degree, granting me the wisdom along the way and placing people in my life to help me accomplish this task.

I want to thank my co-chairs Dr. Brian Hay and Dr. Kara Nance for having patience during this long process and providing the guidance that has helped me achieve this goal. I would like to thank the remainder of my committee, Dr. Jon Genetti, Dr. Joseph Hawkins and Dr. Orion Lawlor, for providing feedback during my presentations and offering insight on my project.

Last but not least, I want to thank my wife and my boys. They have been more than understanding during this entire process. Their love and support have propelled me through the long nights of writing. I want to thanks my entire family for their encouragement and urging to complete my degree.

Without my family, friends and advisers this would not have been possible, so thank you!

Chapter 1: Problem Statement

1.1 Overview

The connectivity provided by the internet has dramatically affected the world in which we live. With almost all business transactions, academic research, and personal communication utilizing and relying on this avenue, the need to defend ourselves against attackers has become essential. The observable rise in computer incidents since 1999, as seen in Figure 1, from automated and targeted attacks from across the globe has produced a multibillion dollar industry to identify and mitigate unauthorized intrusion attempts. Reducing the high number of false positives in these intrusion detection system (IDS) logs has led to the development of a decoy technology whose only purpose is to be attacked and compromised to aid the system administrator in determining the method of both the penetration and execution for future attacks. Introduced in the 1990s as honeypots, these systems have allowed researchers and corporations to capture tools and identify vulnerabilities in their systems.

Since honeypots are decoy systems and they offer no production value to the environment, they provide the ability to monitor the network without the painstaking analysis that inherently comes with monitoring a production device. Forensically analyzing a production device means delving into the voluminous logs and programs necessary to conduct business. On a non-production device, attacks are able to be distinguished from the network noise and the source of the attack quickly pinpointed. It is this effectiveness and efficiency that have facilitated the evolution of honeypot systems.

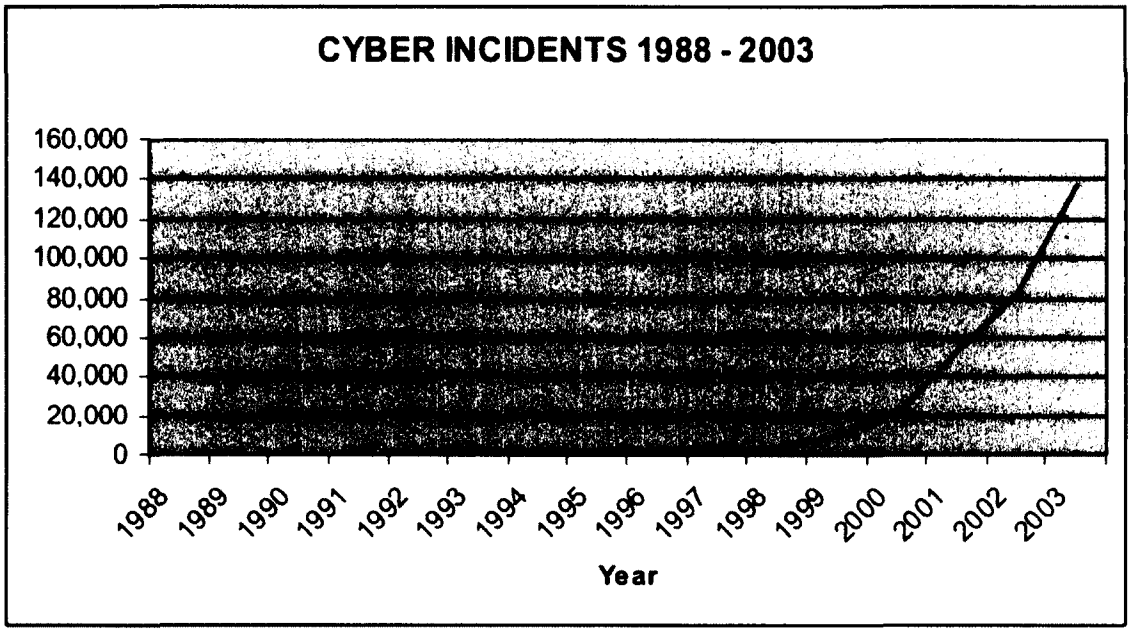


Figure 1, Cyber incidents reported to the Computer Emergency Response Team (CERT) by third parties within the U.S. [Turk, 2005]

1.2 Detailed Problem Description

Due to the variability of cyber-attacks and the capability to conceal the compromise in a device, new tools need to be developed and old tools need to be expanded to assist in the war that faces every network attached to the internet. In 1998, John Howard and Thomas Longstaff wrote a report for the Sandia National Laboratories which presented a matrix of possible computer and network attacks [Howard & Longstaff, 1998]. This matrix is still relevant today since the taxonomy of incidents has not changed only the methods used by the attacker. As shown in Figure 2, it is apparent that the combination of possible attacks based on the tools, vulnerabilities, actions, targets, and unauthorized results are exorbitant.

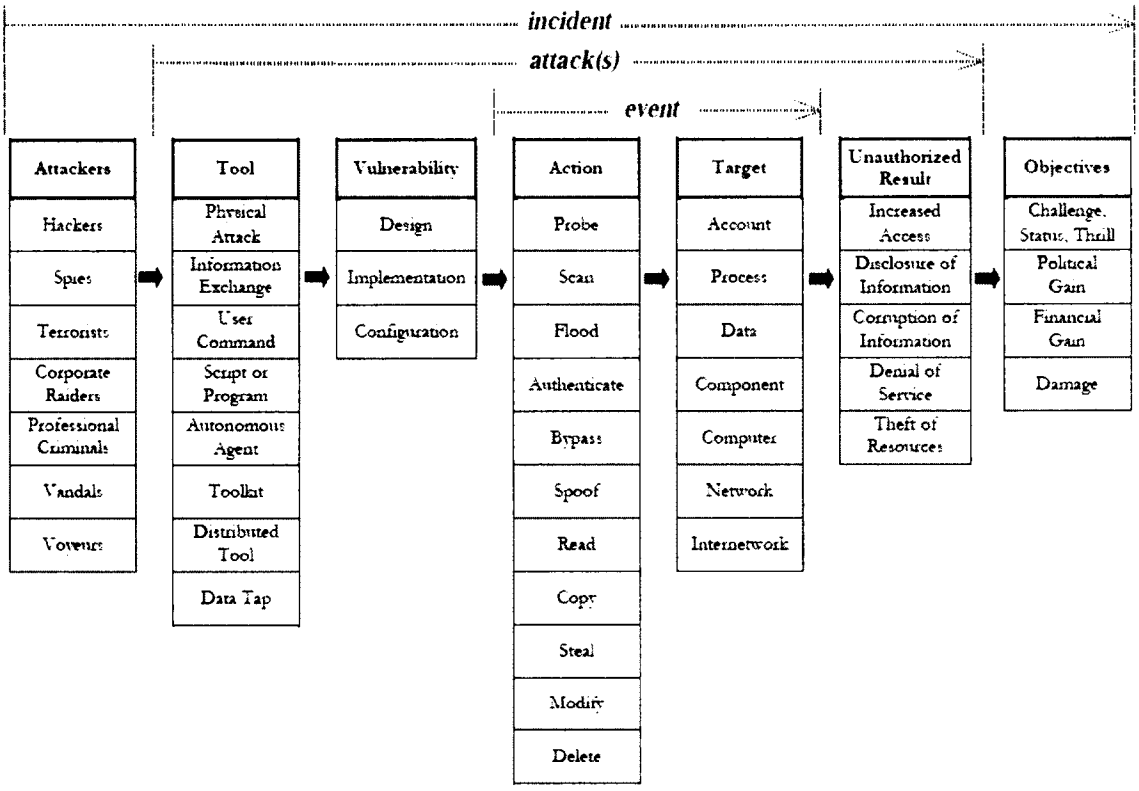


Figure 2, Computer and network attacks [Howard & Longstaff, 1998]

As these attacks occur, identification of the methods and tools used by the attacker to compromise the system becomes a fundamental technique to understanding current and future events. Most current ID systems rely on signatures of known malicious software or tools to identify and mitigate threats. But with malware polymorphism, code packing, and simple semantic changes even the modern IDS can be beaten. These abilities to conceal both tools or malware, make it necessary to have the piece of suspected code run on a live system to confirm it can be used for nefarious means and to determine its effects on the system.

In the past, attackers would target large government systems to collect sensitive information. But the news is full of articles where small businesses and individuals are being attacked at an increased rate [Acohido, 2011; Freedom, 2011; Larose, 2012; Srivastava, 2012] and attacks on corporations are often being orchestrated utilizing social engineering [Clayton, 2012; Freri, 2011]. The security model for protecting a network is changing from guarding the perimeter with firewalls and IDSs to watching for indicators that the network has been compromised from within, via the social engineering route. Thus it is extremely important to have the infrastructure in place to monitor the internal network for signs of compromise. There are three specific issues which need to be addressed to change the mindset in this modern cyber warfare to allow honeypots to be used to their fullest potential.

1.2.1 Opposition to deploying honeypots on an organization's network

"Therefore, just as water retains no constant shape, so in warfare there are no constant conditions...He who can modify his tactics in relation to his opponent, and thereby succeed in winning, may be called a heaven-born captain." - Sun Tzu, The Art of War [Tzu, 1910]

Honeypots are used in research arenas, antivirus companies, internet service provider (ISP) sector and federal government agencies due to their ability to collect statistics about types of attacks, location of attackers, and tools used to perpetrate an attack. Yet some security professionals are leery about using such technology in an organization's production network environment. Some of the arguments that these security professionals express about deploying honeypots are: exposing the company to unnecessary risk by luring attackers to the network, generating operational overhead that

most organizations cannot handle by producing the same information that IDSs do, and just knowing an organization is under attack isn't useful [Higgins, 2006]. These arguments are addressed in subsequent sections of this paper.

Organizations continue to underestimate the uses and benefits of honeypots, due to their lack of knowledge of them. Organizations must be willing to employ new technologies to discover the attempts to steal information or compromise their systems, because attackers are definitely not afraid of exhausting their every resource to gain access to the organization's network.

1.2.2 Self-adapting Honeynet

"Hence that general ... is skillful in defense whose opponent does not know what to attack."
- Sun Tzu, The Art of War [Tzu, 1910]

"All warfare is based on deception." - Sun Tzu, The Art of War [Tzu, 1910]

Many design ideas and advancements have led to the creation of honeyfarms which are deployed in elaborate architectures. Some honeyfarms have the ability to route attackers from low interaction honeypots (LIH) or redirectors to high interaction honeypots (HIH). Other architectures rapidly deploy high interaction honeypots based on the number of attackers. These honeyfarms rebut the argument that honeypots are "like putting a stick into the ground and hoping a guy running at you runs into it instead" [Higgins, 2006]. They offer the ability to deploy large number of honeypots which increase the ability to snare an attacker. Yet none of these architectures enable the user to model the individual honeypots after an organization's infrastructure.

With the continual updates and patches of our operating systems, installation of new applications and devices, new smart devices being released every month, telecommuters coming and going, and administrators better securing the work environment; production networks are a place of constant change. Yet HIHs are still deployed after manually creating a template which takes considerable time to keep the honeynet consistent with the production network. Advancement has been made in the area of low interaction honeypots; one form of network scanning is utilized to assist in the generation of configuration files which allows the low interaction honeypots to adjust to the changing environment. But even these systems do not have an automated method to re-scan the network then deploy a new honeynet in response to the changes.

Kansas State University researchers were recently awarded a five year grant from the Air Force Office of Scientific Research to study the "Understanding and quantifying the impact of moving target defenses on computer networks" [Tammen, 2012]. The researchers will construct a working model of a network system which chaotically changes its configuration to impede an attacker's attempts to learn about the environment. While this idea will make it more difficult for the attackers to map out a organization's network topology, the attacker will still be able to perform reconnaissance on the nearby devices and attempt to detect their vulnerabilities. Upon discovering a vulnerability, it is a matter of relocating the device and executing the code even if the network topology has "changed."

As the computer networks become more diverse, with the proliferation of smart phones, tablets, and a wide assortment of computers and operating systems; it is

necessary for the network administrators to have a honeynet system which self-adapts to the changing environment. Low and high interaction honeypot must be configured "on the fly" so the honeynet becomes indistinguishable from the production devices. Additionally, the honeypots must be sporadically distributed about the network in ample numbers to increase the probability of sensing a breach. An example self-adapting honeynet architecture is shown in Figure 3.

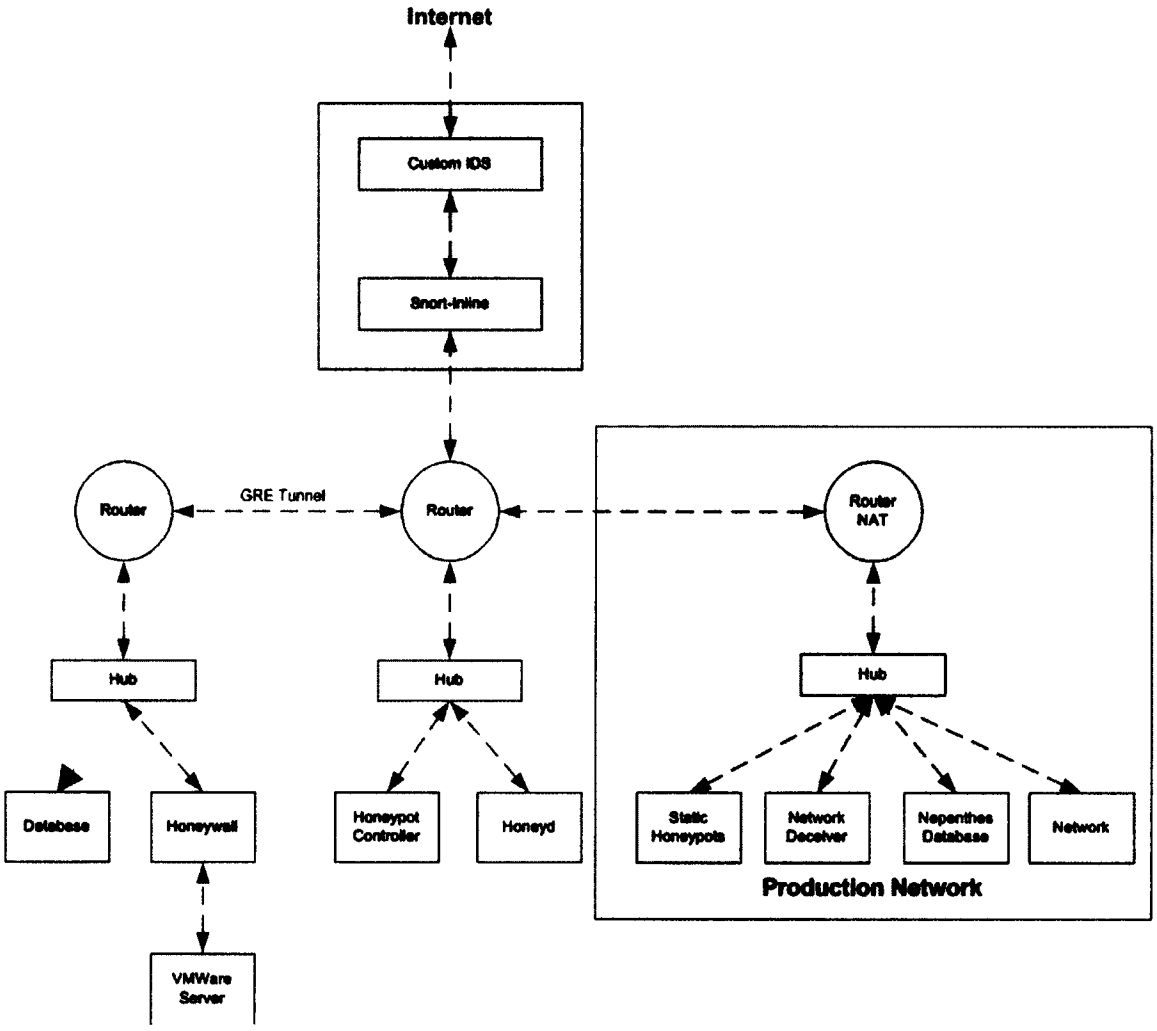


Figure 3, Network Diagram [Hudak, 2008]

1.2.3 Gathering more "valuable" intelligence

"If you know the enemy and know yourself, you need not fear the result of a hundred battles.
If you know yourself but not the enemy, for every victory gained you will also suffer a defeat.
If you know neither the enemy nor yourself, you will succumb in every battle."
- Sun Tzu, The Art of War [Tzu, 1910]

With the limited time that administrators have to identify, decontaminate, and reimage a compromised device it is imperative that effective and efficient tools be at their disposal. Honeypots are such a tool by providing value to model threats and techniques used by attackers [Dacier, Pouget & Debar, 2004], and for finding compromised hosts and learning how to repair them [Jackson, Levine, Grizzard & Owen, 2004]. Unfortunately, high interaction honeypots currently require considerable time for their initial setup. Virtual machines (VM) have reduced the time to re-deploy a honeypot after infection, but constructing the template and updating the VMs to accurately represent the production devices is still time consuming. To further complicate the issue, administrators might not have the current network topologies from which to build their high interaction honeypots. As users install new applications, installation of network devices, telecommuters return from travel, and smart phones are brought into the workspace; the administrator has the difficult task of keeping the network topology up-to-date.

Low interaction honeypots are much easier to configure and self-adapt to a production environment. However, there are limitations to the amount of data able to be collected through a low interaction honeypot. Simple low interaction honeypots respond during the TCP three-way handshake to imitate a real device. Some low interaction

honeypots respond a bit more by imitating the actual service that the attacker is trying to engage. In order to enhance the interaction, thus gather more information about the attack, scripts need to be implemented or brought to a higher level [Pouget & Holz, 2005]. During Pouget and Holz's research, it was determined 40 times more packets were exchanged with high interaction honeypots' real services than with the low interaction honeypots' scripts.

Proper location, time and bait will allow a honeypot to gather information on more advanced attackers [Spitzner, 2004]. Since organizations need to gather valuable information about the attackers that are plaguing their networks and the vulnerabilities they are exploiting, the honeynet has to be configured to resemble their production networks. Attackers are using more targeted attacks on specific organizations to gather information that the attackers find worthwhile. Deploying replica honeypots amongst the real production devices will cause the attackers to search for the information that they desire. Administrators will be alerted to the reconnaissance and the information the attackers want by examining the type of honeypot that was explored.

1.3 Problem Statement

Although the techniques and tools involved in honeypot deployment have continued to evolve for the past 15 years, there are still important research issues which need to be addressed. This research effort addresses some of the identified limitations in current honeypot technology. For honeypots to be useful in a corporate environment the honeynet must represent the production devices on the network. This forces the network

administrators to constantly reconfigure their honeypot systems, which takes considerable time and effort. The amount of time required to reconfigure the honeynet is dependent on the dynamic nature of the environment. In addition to creating devices that match the environment in which they reside, honeynets must be adaptive in the ways that the honeypots are concealed amongst the production devices. If the honeynet is deployed using an algorithmic approach, then the attacker is able to avoid the honeypots upon deducing the pattern. Deploying the wrong honeypots and/or neglecting to take the necessary steps to avoid detection will ensure that no actionable intelligence will be gathered.

This research investigates the effectiveness of scanning a production network in order to dynamically create honeypots to represent the production network, with the benefits of a combined low and high interaction honeynet.

Chapter 2: Foundational Work and Current Approaches

2.1 Literature overview

While intrusion detection is a very broad field, I will include aspects of this topic as it relates specifically to honeypots and similar technologies. This literature review is a summary and analysis of articles, journal publications, conference papers, and books that relate to the field of honeypot systems.

2.1.1 History and Theoretical work of Intrusion Detection Systems (IDSs)

Since the sixties, people have been connecting computers together to share information. Most of the institutions using this network of computers were research, academic or government. Then in 1979, USENET allowed users to hold discussions and publish information through newsgroups [Hauben, 2002]. USENET along with email, telnet, and FTP gave more access and ability for users to share or search for information. During the seventies and eighties many system administrators, researchers and academics were concerned about computer security and the threat of people using the internet to compromise computer systems. In the early eighties, James Anderson wrote a paper regarding the ability to use audit trails and surveillance to ensure the security of clients' computers [Anderson, 1980]. This was the beginning concept to intrusion detection. In the mid-eighties, Dorothy Denning and Paul Neumann did work on the first real-time IDS named the Intrusion Detection Expert System (IDES) [Denning & Neumann, 1985; Denning, 1987], which used rule-based architecture to detect known malicious attacks.

This IDES design was implemented using anomalous behavior detection to identify possible network intruders or internal people abusing their network access [Lunt & Jagannathan, 1988]. IDES reported on several behaviors (time and location of login, connect time duration, CPU and I/O activity, and protection violations) that were compared with the baseline individual user profile to discover whether the behavior was anomalous. These workstation audit records of user behavior were recorded into an Oracle database. Lunt planned to further investigate ways, in future iterations, to increase the detection abilities which discriminate between normal and anomalous activity.

Although accounts were published in the late eighties and early nineties of attacks and compromises, many administrators were not persuaded to fully lock down their computers by implementing security policies. As browsers from Mosaic and Microsoft became commonplace, the internet exploded with information due to the ease and accessibility of sharing and retrieving that information. New vulnerabilities introduced by the internet and the browsers allowed new attack vectors to be exploited with greater efficiency. In addition, there was always the threat of attacks from knowledgeable users from a previous era. As operating systems and other programs continue to grow in complexity, often reaching millions of lines of code, the security vulnerabilities are continually being exploited by attackers through review of the source code or the use of tools. Since the time of the mid-nineties the proliferation of tools and programs to attack computers has continued to grow, which increased the number of individuals and decreased the knowledge level needed to attack computers systems.

System administrators have the daunting task of securing their networks and information technology (IT) infrastructure but still allowing the users access to information. This has especially been a problem at academic institutions, where the atmosphere has been one of openness and any restrictions are thought of as negative by the user community. The tradeoff of having high availability of data while ensuring the confidentiality/integrity of the data is the conundrum faced by every organization. Even programs originally installed by default on many computers to allow information sharing or allow users to see whether an individual was using their computer such as sendmail and fingerd would be used as future attack points [Kehoe, 1992]. So the administrators not only had to keep information flowing but they also had to be aware of vulnerabilities in a time that update servers and security bulletins were not available or prevalent. The government has been funding research regarding the transfer of information via a network since the mid-1960s [Leiner et al., 1997; Leiner et al., 2009]. It was not until the 1970s and 1980s that the government outlined ways to improve intrusion detection techniques [Anderson, 1972a; Anderson, 1972b], but it was not until the beginning to mid-nineties that network based technologies became widely available [Bruneau, 2003; Kemmerer & Vigna, 2003]. This early technology was not sufficient to detect all attacks, as with the technology still in use today, though it did provide the system administrators some foresight of how to better secure their networks.

System administrators have been watching and logging information from attackers for over 50 years, ever since the first computers were connected over phone lines in 1965 [Leiner et al., 2009]. Stories of Cliff Stoll, watching attackers from

Germany browse through government and university computers while recording their every move, demonstrated that even "secure" systems were vulnerable to attack [Stoll, 1990]. Bill Cheswick also documented attackers trying to break into AT&T's computers by using the "chroot" jail in UNIX to study their tactics [Cheswick, 1990]. As the internet progressed in the nineties and access to computers increased, the amount of network traffic increased exponentially. The sheer volume of the information prevented administrators from manually watching individual networks and shifting through log files for anomalies. Software and hardware based tools, like Snort [Snort, 2012], Wheel Group NetRanger, and internet Security Systems (ISS) RealSecure were created to reduce the amount of unwanted traffic into private networks and were used to report the most serious, novel, or significant offences to the administrators.

2.1.2 Applications of Intrusion Detection Systems (IDS) and Tools

Many tools have been developed to help administrators find signs of intrusion and report the incidents. There are two main categories of intrusion detection systems: anomaly detection and misuse detection. These systems can be implemented for a single host or from the perspective of a network segment (or even an entire network). The anomaly based device uses a baseline to characterize events happening on a computer or network. The characteristics used in the baseline could be packet types, login attempts, tasks performed by a user, or the flow of network traffic between two computer systems. If any of the observed events deviate from the "norm," then they are reported as suspicious and potentially dropped. This produces a high rate of false positives but aids in the detection of new attacks. Misuse based IDSs have a predefined set of rules or

signatures that must be obeyed by the arriving packets. Any variance will prompt a response from the device. Contingent on the quality of the signatures, the misuse IDS has the advantage of a low false positive and increased efficiency. Despite the number of IDSs in place throughout our networks, the detection rate of new attacks is still relatively low. Distinguishing between real attacks and a rogue computer or program is quite difficult in today's network environment. With millions of packets on a large network, searching through the information to find new attacks and determine if a computer has been compromised is complex.

A comprehensive survey and classification of IDSs can be found in Joseph Sherif and Tommy Dearmond's paper *Intrusion Detection: Systems and Models* [Sherif & Dearmond, 2002].

2.1.3 Honeypots

In 1997, the deception toolkit was released which emulated through PERL scripts many UNIX security vulnerabilities [Talabis, 2006]. This toolkit allowed administrators to record the attackers through a deception scenario. From this concept many programs have been written in various implementations to watch the moves and tactics of attackers. This notion of using deceptive computer systems to attract attackers has become known as honeypots. Even though many people have used similar techniques, the wide scale implementation did not happen until the nineties. Honeypots are defined as computers with no production value except to be compromised and record the attacker's behavior and tools. Since there are no legitimate services or work being performed on a deployed honeypot, this creates a virtual mousetrap that can be observed, with all interaction with

the honeypot having a strong likelihood of being malicious. Previous to this time, the cost to usefulness ratio did not allow for "extra" computers to be used for ID research or capture devices other than by academic institutions and government agencies; but as the price of computers dropped dramatically in the late nineties and early 2000s, this opened the door for more research in this area. Virtual machines and emulating tools allow security conscious users to use and improve on this technology without the need for extra computer equipment.

Honeypots can either be active or passive, based on the way that they detect attackers on a network. Client-based honeypots actively seek attackers by visiting suspicious web servers or executing malicious malware [John, Yu, Xie, Krishnamurthy & Abadi, 2011]. Server-based honeypots passively wait for attackers to initiate attack. This paper will delve into the server-based subset of honeypots exclusively. Since many honeypots are put onto a network with running services that might allow the attackers to gain control of the computer or new application that might contain security holes, there is always the concern of the honeypots being compromised and initiating more attacks. The growing field of honeypots has since been sub-divided into two domains: low and high interaction. Though some experts [Barfar & Mohammadi, 2007; Mokube & Adams, 2007; Spitzner, 2002; Wicherski, 2006] differentiate between three types of honeypots, this paper will combine low and medium interaction honeypots. The amount of information able to be obtained from the honeypot, the risks and costs associated with deploying and operating the resources are the criteria which separate the low and high interaction honeypots.

Table 1, Types of honeypots [Danford, 2006]

Low Interaction Medium Interaction**	- Transport layer virtualization - Application layer virtualization
High Interaction	- Real, vulnerable systems
**Combined with low interaction honeypots throughout the paper	

2.1.4 Low Interaction Honeypots

The most prevalent honeypot category is low interaction systems. A low interaction honeypot (LIH) uses a program to emulate services, open ports, or applications. When an attacker scans a computer for open ports the LIH allows a limited interaction between the attacker and the emulated service for which the port is associated. The amount of interaction is dictated by the scripts behind the emulated service. Due to the limited interaction, the administrator is able to record the traffic that is visiting the system without worrying about the trouble of a compromise.

LIHs come in all shapes and sizes, the information gathered about the attack and attacker can also be quite varying. There are many focuses with regards to LIH such as web services (PHP.HoP [Oudot, Ropert & Riden, 2012], Google Hack Honeypot [GHH, 2012], Glastopf [Glastopf Project, 2012]), malware collection (Nepenthes [Nepenthes, 2012], Dionaea [Dionaea, 2012], Amun [Amun, 2012]), tarpitting (LaBrea [Liston, 2012]), single-host services (Deception Toolkit [Cohen, 2012], Back Officer Friendly [NFR, 2012]) and multi-host services (Honeyd [Provos, 2004], Tiny Honeypot [Bakos, 2012], Specter [Specter, 2012], KFSensor [KFSensor, 2012], Honeytrap [Honeytrap, 2012]).

A specific LIH which is relevant to this research effort is Honeyd, which is an open-source framework which allows network services to be simulated from thousands of internet protocol (IP) addresses. Honeyd is able to respond to thousands of requests for unused IP addresses on a network; this sets a large net to gather as much information on potential attacks. Honeyd is full of features and options which have been discussed in many books and in some it has at least a chapter devoted unto itself [Grimes, 2004; Provos & Holz, 2007; Spitzner, 2002]. Honeyd allows for scripts to be associated with a particular service to allow for more interaction with the attacker. Tools have been written which automate the process for script generation for Honeyd [Leita, Mermoud & Dacier, 2005].

One of the biggest challenges faced by the administrators of most security technologies, including honeypots, is that the configuration process can be quite challenging and time consuming [Spitzner, 2003]. Although these systems are available to a wide audience, they require constant modification and configuration to be a true representation of the current network environment, or to emulate the services or vulnerabilities of interest to the system administrator. Even if the administrator is putting forth the effort trying to mask the presence of a honeypot within the network, the honeypots still require monitoring and maintenance to ensure the host has not been compromised and that the latest patches to both the host and honeypots system have been applied to decrease the likelihood of detection. Research has also been conducted to discern between LIH and true services [Defibaugh-Chavez, Veeraghattam, Kannappa, Mukkamala & Sung, 2006; Mukkamala, Yendrapalli, Basnet, Shankarapani & Sung,

2007]. Their findings explain the ability of attackers to detect these low interaction honeypots unless they have been significantly modified from their known default configurations. Some of these detection methods are even distributed as commercial products, such as the Send-Safe Honeypot Hunter which detects a particular type of honeypot aimed at identifying SPAM sources by determining whether the open proxy on the honeypot can send email back to the spammer [Send-Safe Honeypot Hunter, 2006]. In other cases, past versions of Honeyd could be identified by sending a SYN/RST packet which was answered, incorrectly, by a SYN-ACK [Corey, 2004], or by sending fragmented SYN packets which normally would be dropped by the destination host, but which actually elicited a response from the Honeyd system [Oberheide & Karir, 2006]. To counter these anti-honeypot techniques, work has been done to camouflage Honeyd and defeat the Neyman-Pearson (NP) decision theory timing attacks by modifying Honeyd and the underlying operating system to allow for a higher-fidelity emulation of events [Fu et al., 2006]. Additional work has been done to create a deceptive system, honeyanole, which uses blacklists to avoid detection [Shiue & Kao, 2008]. The honeyanole uses a collection phase, redirection phase, and deception phase to collect information on a potential attack then performs blacklisting. If a potential attack occurs from a blacklisted IP address then the attacker will be redirected to a deception sever with TTL masquerading to hide the redirection.

Ease of installation and maintainability has made LIHs very popular and used in a wide variety of projects. The Brazilian Honeynet Project setup LIHs in various locations around Brazil and centralized the data for early warning and incident response [Hoepers,

Steding-Jessen, Cordeiro & Chaves, 2005]. The Leurre.com project used LIHs in a large distributed platform that spanned 11 countries, to watch attack frequencies, tools, and origins [Dacier, Pouget & Pham, 2012; Pouget, Dacier & Pham, March 2005].

While low interaction systems are prevalent due to their ability to emulate so many different systems and the relatively low risk of the underlying host system being compromised via the honeypot, the disadvantage of such systems is that they can generally not gather a great depth of information. For example, if a Honeyd system is configured to have an FTP and a telnet port open, these are the only ports the honeypot will gather information about. In addition, the "services" listening on the honeypot are typically just emulations of real services, and as such only allow limited interaction opportunities for an attacker. Through the use of scripts, some low interaction honeypots are able to gather more detailed information about a specific attack, but this is always limited by the extent to which the script is written to emulate realistic interaction with the attacker. Since these systems rely so heavily on the associated scripts for interaction beyond the most superficial levels, the system administrator is forced continually to change the scripts to match their current network configuration and associated services, or risk having their honeypots look significantly different than their production hosts, which not only limits the value of any information obtained, but also provides an easy mechanism to differentiate between honeypot and production systems.

Because low interaction honeypots do not fully emulate an entire operating system and associated services, the use of such honeypots generally does not allow the recovery of the complete methods by which the attacker would interact with a real target

to compromise it, cover their tracks, and either gather data from the compromised host, or use it as a launching point for further attacks.

2.1.5 High Interaction Honeypots

High interaction honeypots (HIH) are an actual full system with appropriate services and functionally with which the attacker can interact. The HIH provides a much more detailed look at the attacker's tools and interaction with the target host before, during, and after a system compromise operating system; although it remains likely that non-automated attackers will realize at some point that they are interacting with a honeypot system. However, the ability for the attacker to interact with the system at a much greater depth can allow observers to retrieve attack tools and find the security vulnerabilities in the operating system before a large scale attack is mounted. This ability of the HIH to gather large amounts of interesting information has compelled users to deploy HIH in extensive networks and use HIH in teaching environments [Azadegan & McKenna, 2005; Hoepers et al., 2005; Levine, LaBella, Owen, Contis & Culver, 2003; Levine, Grizzard & Owen, 2004].

Tools have been developed to aid in the logging and data capture of HIH. One common approach is to modify the honeypot host to include a service, such as SEBEK [Honeynet Project, 2003] can covertly monitor activity on the honeypot system. Versions of Sebek are currently available for Windows, Linux, and BSD Operating Systems, and involve a hidden client component which is installed on the honeypot system, and a remote server component. The client attempts to unobtrusively record a wide range of activities on the honeypot system (by capturing all system call read data from within the

operating system and monitoring encrypted network activity directly from the network card), which it then reports over the network the Sebek server, where the data can be reviewed and archived. Qebek is a QEMU [QEMU, 2012] emulator-based capture tool which monitors beneath the virtual machine [Song, Hay & Zhuge, 2010]. Qebek uses several modules and systems (breakpoint, interception, introspection, and output) to monitor the guest operating system. These tools aid in the monitoring of the HIH systems by recording varying information about the attacker's methods and tactics. After the HIH has been compromised additional prototype tools have been tested to assist the user in digital evidence collection [Carbone & de Geus, 2004].

Due to the nature of the HIH, the attacker is likely to have full access to the operating system and services which allows more information to be gathered about an attack yet also makes the HIH a liability. Attackers are able to leverage the HIH as an attack platform to initiate more attacks after they have successfully compromised the computer. But tools have been created to limit the harm done by an attacker after a system has been compromised. The Honeywall, Figure 4, monitors the inbound and outbound traffic from a honeypot, which should be minimal since the honeypot has no production value [Chamales, 2004]. The Honeywall can alert the administrator to an attack, block outbound traffic from a compromised system, scrub the attacking packets to reduce their effectiveness, and monitor the honeypots through a Sebek sever module.

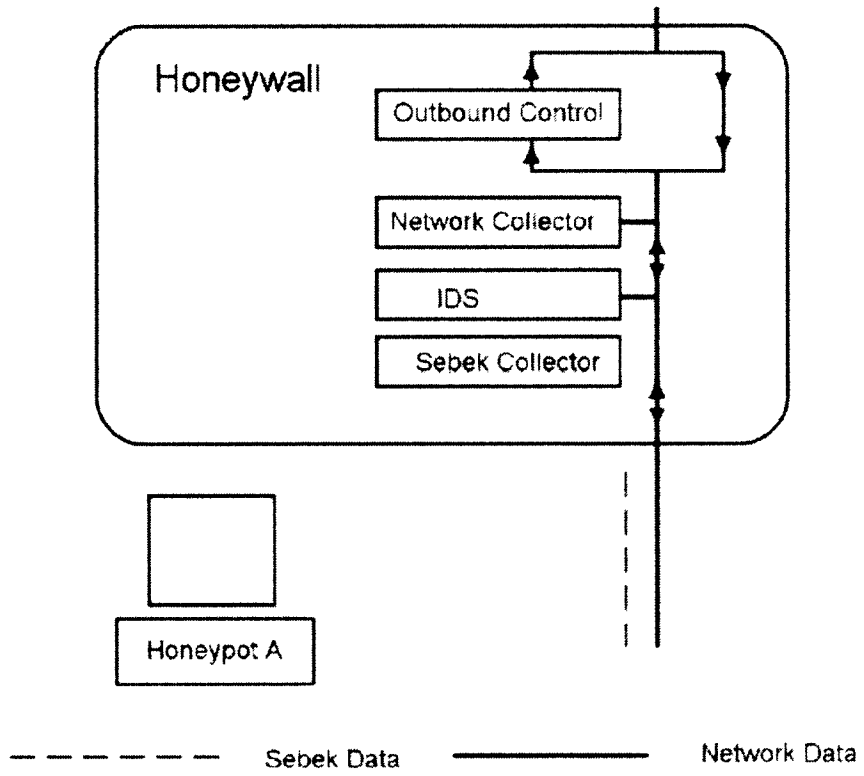


Figure 4, Honeywall Diagram [Balas & Viecco, 2005]

The hardware constraint of having one high interaction honeypot per computer limits their scalability. Virtualization has helped overcome the hardware limitations by being able to run multiple HIHs on one physical host, with associated cost reductions and scalability benefits. Beyond the savings in hardware costs, installation time can be considerably reduced for virtual machines versus physical systems because a template can be made of a virtual machine from which clones can be deployed. Additionally, when a compromise occurs the clone can be disabled for forensic analysis while a new clone is deployed from the previously uninfected state or snapshot. Even when virtualization is used, a high interaction honeypot requires an administrator to install a complete operating

system and all the services necessary to gather the desired information. With each change in the network configuration, operating system, and in services used on a workstation or server, the administrator is required to make changes to the honeypot so it continues to be representative of the environment in which it is to be deployed. Procedures and scripts have been developed by the Brazilian Honeynet Project to help streamline and create a uniform process in creating a HI honeynet [Chaves, Franco & Montes, 2005]. A freely distributable, bootable design was implemented to swiftly create an entire honeynet, including a Honeywall, utilizing QEMU technology on a single Honey-DVD [Dornseif, Freiling, Gedicke & Holz, 2006]. Alen Capalik devised an architecture which rapidly deployed HIH using virtualization for intrusion detection [Capalik, 2007]. The VMs run on top of a Kernel-based Virtual Machine (KVM) for Linux which monitors the honeynet through a "sentinel" module which perform low-level introspective memory analysis. In another study, several virtual machine monitors (VMM) were implemented, tested and evaluated on their ability to detect attacks [Asrigo, Litty & Lie, 2006]. These architectures show that VMs can be used to detect attacks on a small number of physical systems. Using some of the current tools, a third generation Honeywall architecture was designed for data capture [Balas & Viecco, 2005]. Two paths were suggested for the data that was captured from an intrusion to better help the administrator and the analyst. The fast path provided a high level comprehension of the data but resulted in some degradation of detail and is stored in a relational model. The slow path allows the analysts to look at all the nitty, gritty details and is stored in the canonical form. Argus [Argus, 2012] and Snort are used to monitor network flow, p0f [Zalewski, 2012] is used

to passively determine the OS of the intruder, Sebek is used to monitor socket activity; and then all this information is stored in a Hflow [Viecco, 2007] database. Pcap files are stored to restore all the captured data since they are in canonical form. Then an improved Walleye [Balas & Viecco, 2005] interface is used to display the high level data in a flow form. There are other systems which use VMs to rapidly deploy honeypots, and these will be discussed in the Hybrid section of this paper.

Despite all the advances in HIH technology, there are still many challenges that must be overcome. There have been papers written about how to detect Sebek utilizing various methods [Corey, 2003; Corey, 2004; Dornseif, Holz & Klein, 2004]. For an attacker to detect a honeypot is valuable information. Most attacks are conducted by exploiting a system then leveraging that system's place in the environment to compromise their next victim. If an attacker knows they are in a honeypot then they can use disinformation to thwart efforts to forensically analyze the compromise or use the information already collected by the honeypot for their own nefarious needs [Krawetz, 2004]. Since honeypots are also used for malware analysis, the detection of a honeypot could trigger the malware to sleep or act in a different manner than in a non-honeypot system which makes quick analysis more difficult. Detection techniques have also been researched in the area of VMs [Defibaugh-Chavez et al., 2006; Holz & Raynal, 2005; Mukkamala et al., 2007]. Since researchers are utilizing the benefits of VMs in honeypot deployment, attackers are scanning for a virtual environment through their malware and during their network penetration to deduce if they are being watched.

HIHs have been adapted to every field of network and computer security to provide security researchers with large amounts of information. Despite some of the challenges and legal questions, HIHs continue to be deployed in a large number of academic and production environments. One key issue addressed in the next section is how to scale and control these HIHs in a large network without wasting resources when aggregating the data so analysis can begin.

2.1.6 Hybrid Honeypots / Honeyfarms

Combining both low and high interaction systems into one cohesive unit has been achieved through the development of honeyfarms or bait and switch (B&S) systems. For B&S systems, the low interaction honeypots can be placed into any geographical locations and act as redirectors to the high interaction honeypots which are centrally located. The high interaction honeypots are physically centralized to facilitate maintenance and hardware consolidation. The routing or redirecting is done with programs like HoneyMole [Portuguese Honeynet Project, 2008]. HoneyMole provides relatively transparent communication in the form of an Ethernet bridge from a low interaction honeypot to a high interaction honeypot, which allows more information to be gathered about the attack. If an attacker probes one of the emulated low interaction honeypots it can cause a trigger sending the attacker to the high interaction honeypot via the secure communication channel. This process of combining both the low and high interaction honeypots allows the potential for thousands of low interaction honeypots to be deployed on a single host but also have a limited number of high interaction honeypots to gather detailed information. The identified open issues with this method include

creating enough high interaction honeypots that appear similar to the low interaction devices, and keeping the latency resulting from tunneling traffic to the remote high interaction honeypots to a minimum.

In 1998, an IDS architecture was designed and prototyped in which agents were looking for network or host anomalies in a distributed network [Balasubramaniyan, Garcia-Fernandez, Isacoff, Spafford & Zamboni, 1998]. The architecture, Figure 5, suggested is a tiered hierarchical structure and has three main components. The outlying stations are the "agents" which report to the middle tier "transceivers." The transceivers might have multiple agents to supervise and assimilate their findings. The "monitors" oversee multiple transceivers and are the ultimate decision makers based on the information received from one or more agents.

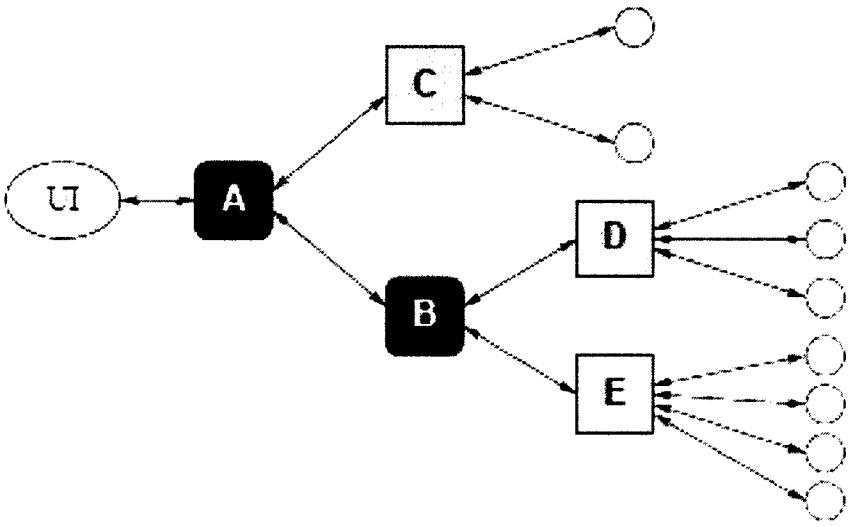


Figure 5, An Architecture for Intrusion Detection using Autonomous Agents [Balasubramaniyan et al., 1998]

Necessary characteristics of each component and the different methods that could be used to facilitate the implementation were discussed. Four of these characteristics that traditional honeypots do not currently possess include the following:

1. It must allow *dynamic-reconfiguration* this is the ability to reconfigure the honeypot or honeynet without having to restart it.
2. It must be able to *adapt* to changes in system and user behavior over time.
3. It must be able to *scale* to monitor a large number of hosts.
4. It must *run continually* with minimal human supervision.

A honeypot architecture, Figure 6, to detect internet threats and use different levels of interaction to gain information about these threats was recommended in a technical report from the University of Michigan [Bailey, Cooke, Watson, Jahanian & Provos, 2004]. Low interaction, high interaction and a filtering mechanism to monitor for new threats was proposed in the new design. The LIH would allow multiple contact points by which to monitor the internet for the threats and be the redirectors to allow new or older threats to pass to the HIH. The HIH would be the host which is infected by the threat then isolated and observed to gain information about the migration throughout the network and further infection of new hosts. The filtering mechanism allows most of the uninteresting traffic to be filtered as not to use cycles from the HIH which could be used to obtain information about the new threats. If a threat is encountered the LIH establishes contact with the attacker. After a brief handshake the filter either ends the connection or passes the conversation to a HIH to re-establish contact and gain further intelligence on the attack. Other papers have been published which present different ideas on the topic of

using LIHs to filter the attacks then using a redirecting mechanism to route the interesting traffic to HIHs [Hudak, 2008; Kyaw, 2008].

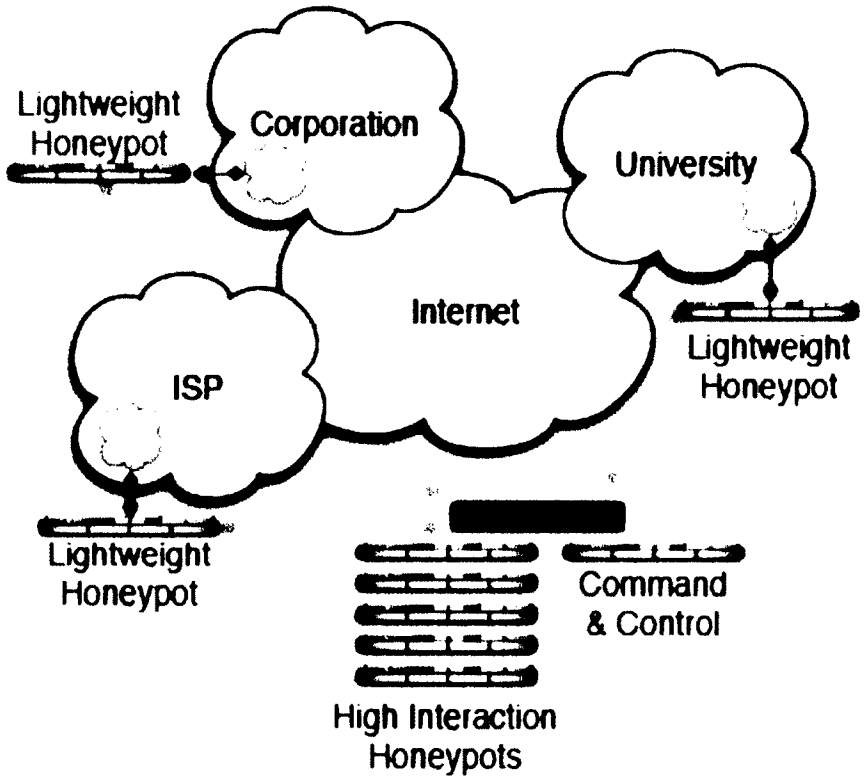


Figure 6, Hybrid honeypot architecture [Bailey et al., 2004]

The Collapsar honeyfarm, Figure 7, at Purdue University [Jiang & Xu, 2004; Jiang, Xu & Wang, 2006] was developed to engage attackers from a multitude of redirectors. The redirectors are servers that filter and direct traffic from a specific IP to the high interaction honeypot (HIH). A gateway is placed between the HIH and the redirectors to control the data flow and mitigate risk due to compromised devices. The HIH were configured using both VMware [VMware, 2012] and UML [Dike, 2006] to

increase the number of honeypots per device. Several experiments were conducted to test the benefits and tradeoffs between the two VM programs.

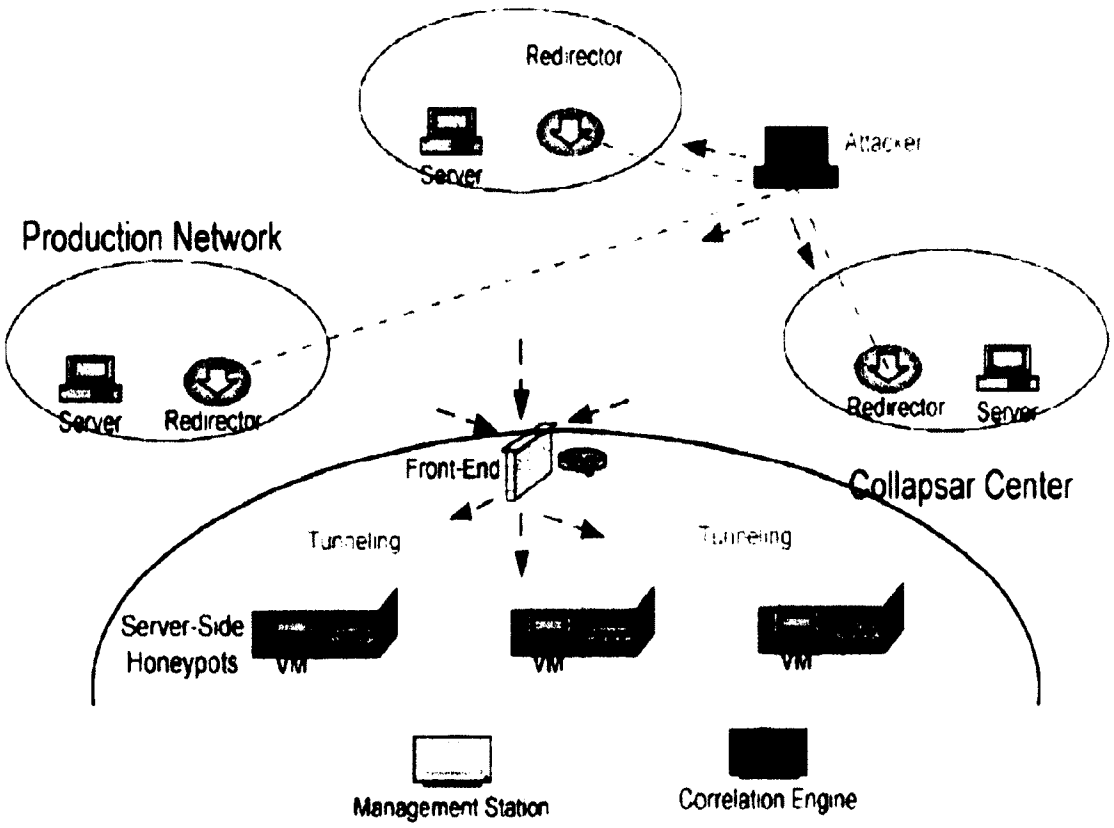


Figure 7, Collapsar honeyfarm architecture [Jiang et al., 2006]

There are several modules used in Collapsar to mitigate risk and facilitate the analysis of the attack. This system provides a centralized location to control the HIH and to investigate the attacks. Since real devices running UML are used for the redirectors, it is necessary to constantly update the host device as well as the VMs to prevent a compromise of the redirector. Due to the design, attackers near the redirector will notice the delay when comparing a real host on the same network. The second version of the Collapsar implementation, Figure 8, also integrated a reverse honeyfarm design which

allowed the redirectors to become client honeypots to increase the amount of information gathered from the system.

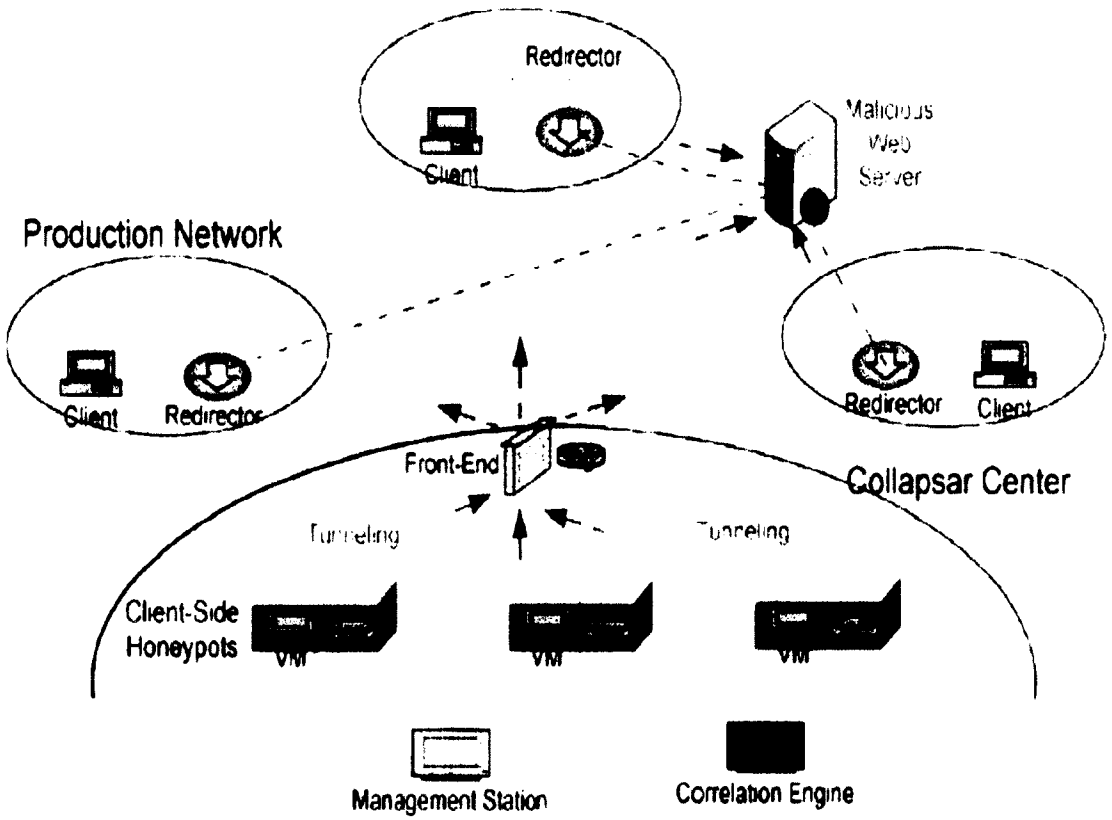


Figure 8, Collapsar reverse honeyfarm architecture [Jiang et al., 2006]

The Potemkin honeyfarm at UCSD [Vrable, Ma, Chen & Moore, 2005] optimized the amount of available honeypots on the server. This was accomplished by utilizing virtualization to effectively use their resources yet still benefiting from a high interaction honeypot system. The setup, Figure 9, utilized two main pieces of hardware: the gateway and virtual honeypot server which consisted of a cluster of 10 devices to implement a honeyfarm with a positive test.

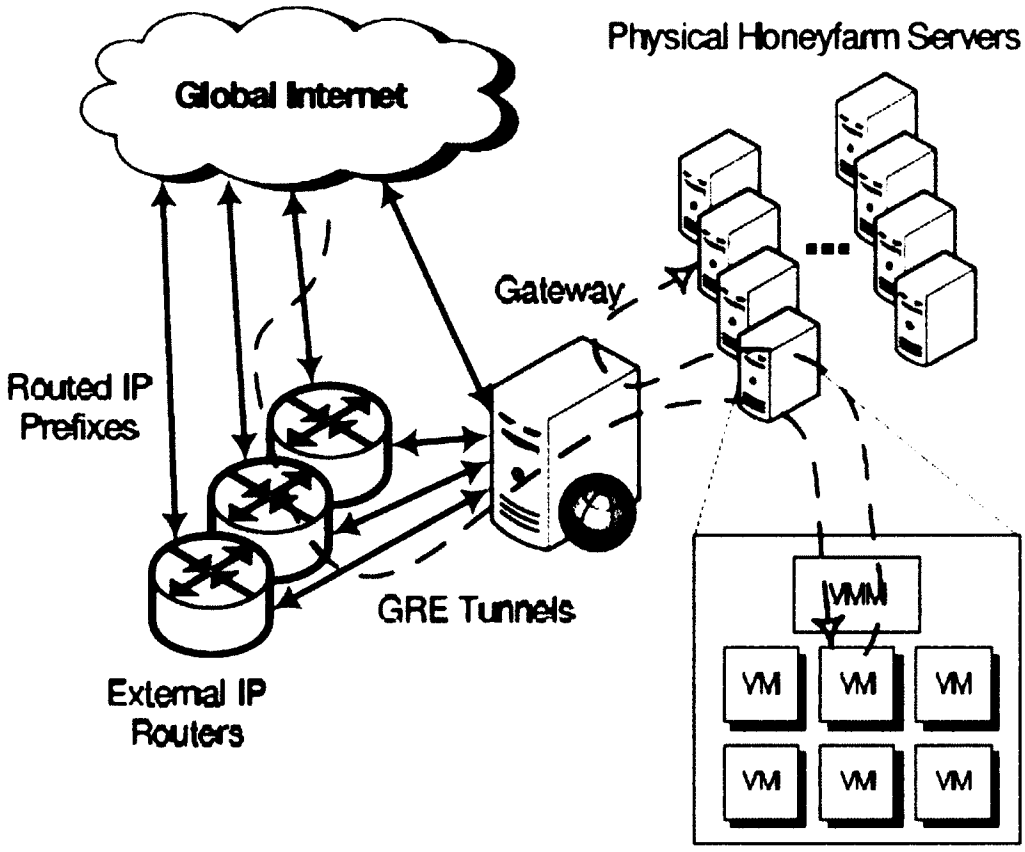


Figure 9, Potemkin virtual honeyfarm architecture [Vrable et al., 2005]

To direct traffic to the honeypots, GRE tunnels and configuring the gateway as a "last hop" router allows multiple IP addresses to converge at the honeyfarm. The external IP address is used in configuring the virtual honeypot with the same address for further engagement of the attacker. Xen [Xen, 2012] was used to construct the virtual honeypot system which allowed fast cloning of the honeypots and IP configuration to take place in just over a half of a second. Small VMs and low memory constraints were used to get hundreds of VMs to run on one server with the possibly of deploying over a thousand.

The outbound traffic is also configurable to protect other systems from further attack. Analysis on the number of VMs necessary to deal with all the traffic including scanning traffic would have been in the tens of thousands range. A scan filter was necessary if a low inactivity timeout period was used to deal with the traffic load.

NoAH, a European Network of Affined Honeypots, was a project funded by the European Community which lasted 42 months, April 2005 through September 2008 [NoAH, 2008]. The NoAH architecture, Figure 10, combined both LIH and HIH into an infrastructure which contained cyber-attacks and tested methods for attack detection and signature generation. The "NoAH core" is a distributable set of honeyfarms, both LIH and HIH, which can collaborate. The LIH is used as a filter to prevent uninteresting traffic from continuing any farther into the core. The LIH handles all communication with potential attackers then redirects "interesting" traffic to the HIH, which ensures the HIH are in a containment environment. Honeyd is used as the LIH which allows scripts to be written to interact with the attacker or act as proxies to specific services running on the HIH. The HIH are run as virtual machines on top of Argos, the containment environment. Argo uses QEMU to emulate the guest operating system and perform taint analysis on the code being attacked by the attacker [Portokalidis, Slowinska & Bos, 2006]. Argos generates a signature for the attacker's code which can be used to supply an IDS. Due to the emulation process and an inability to use QEMU's acceleration mode, Argos runs programs 15 times slower than native execution.

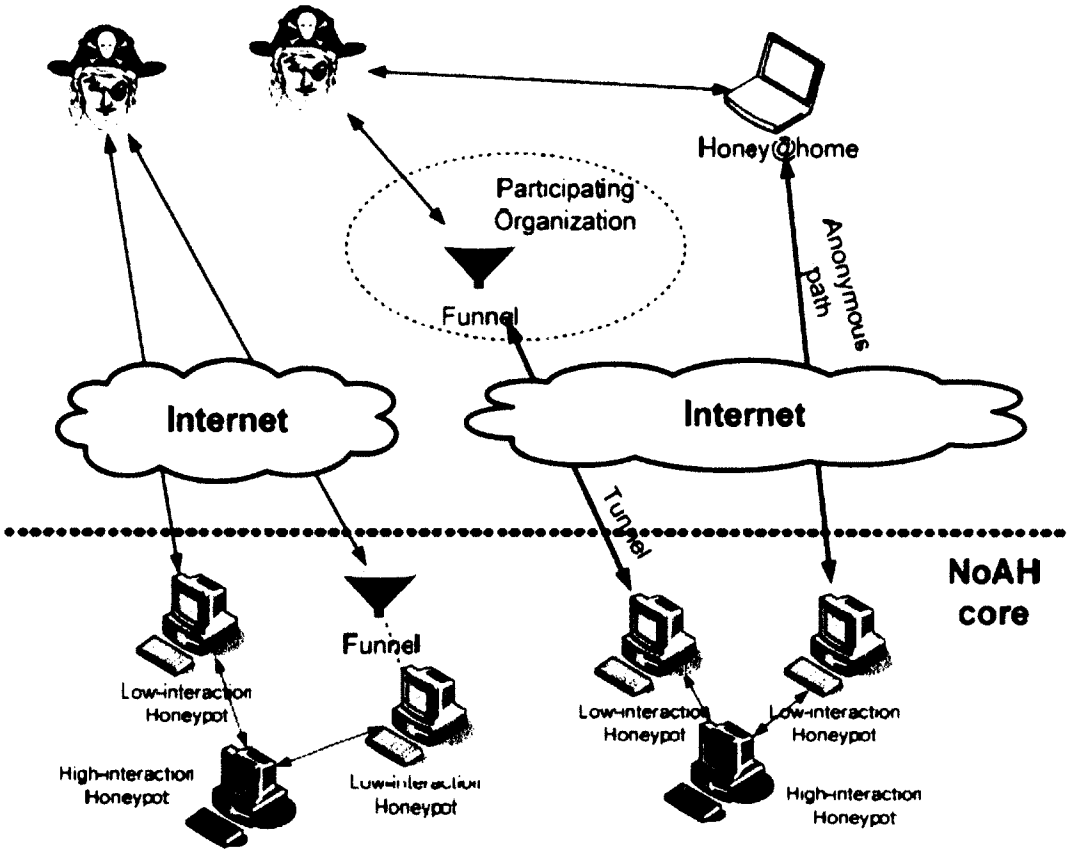


Figure 10, NoAH honeyfarm architecture [NoAH, 2008]

Xargos was developed to supplement Argos by using Xen to run the applications and dynamically switched to Argos to execute the code that interacts with the tainted data. Xargos sped up the process but still needs work to support Windows and new versions of Linux. Traffic is introduced into the "NoAH core" by several methods. Attackers could interact with the LIH directly, organizations could funnel or tunnel traffic from unused IP addresses to the LIH, or individuals could install "honey@home" (h@h) [Antonatos, Anagnostakis & Markatos, 2007] which would redirect traffic to the core. Funneling involves a LIH, like Honeyd, to assume the unused IP addresses and redirecting all traffic

directly to the core's LIH. Tunneling is the organization acting as a relay agent for all the traffic so the traffic appears to be coming from the organizations' network. The h@h software needs no configuration and is supported by Windows and Linux operating systems. The h@h clients are installed on a small business or personal network and use a SSL connection to redirect all traffic to the core. After the data has been processed, alerts are correlated and categorized with each sensor feeding three additional software components for analyzing the data statically, sensor monitoring, and geolocation.

Honeybrid was designed and constructed by Robin Berthier [Berthier, 2009]. This architecture is composed of three components; Honeybrid Gateway, a set of low-interaction honeypots and a set of high-interaction honeypots. The main component is the Honeybrid Gateway which is composed of two sub systems; decision engine and the redirection engine. The network traffic is routed to the LIH through the decision engine to establish a connection and allow the gateway to detect interesting traffic. If the traffic is determined to be worthy of further analysis (source IP address, destination port or payload) then the traffic is passed to the redirection engine which transparently passes the traffic from the LIH to the HIH for a more detailed analysis. Honeybrid uses an original way to redirect interesting traffic based upon a decision engine that is capable of handling large amounts of traffic.

HoneyLab was a design and proposal to deploying a distributed honeynet with the ability to monitor the honeypots [Chin, Markatos, Antonatos & Ioannidis, 2009]. Researchers and security experts would be allowed to deploy Honeypot sensors using the HoneyLab Central resources. IP address location or range would be requested then the

honeypot infrastructure would be allocated for the user to deploy their own honeypot services, instrumentation code and detection algorithms on the XenoServers.

To help detect and prevent network intrusions from both the internal and external environments, a Network Intrusion Collaboration System (NICS) design was put into operation [Prasad, Abraham, Abhinav, Gurlahosur & Srinivasa, 2011]. The NICS would be composed of subsystems called "System of Security Systems (SoSS)" which incorporated Network Intrusion Detection Systems (NIDS), Network Intrusion Prevention Systems (NIPS), and honeypots. This distributed design utilized Snort as the NIDS to detect known attacks based on signatures, IPTABLES provided by the GNU/Linux Debian Squeeze distribution as the NIPS firewall mechanism the prevented unauthorized activities and Honeyd as the honeypot system that helped with threat detection and assessment. Each of these intrusion systems would report information to a customized statistical classifier written using a shell programming language to extract information from the network data. This project demonstrated the ability to have a large system which composes smaller subsystems which centralize the data in one classifier system.

Additional work has been done to conceal honeypots in a distributed system using a Cooperative Deploy Honeynet (CDH) scheme [Wang & Chen, 2011]. The CDH uses a cooperative learning algorithm based on the multi-agent system [W. Wang, C. Wang & Shi-fu, 2006] to deploy the honeyfarm system (HFS). Wang and Chen created a method that quantifies the disguise capability of distributed honeynet system. The model used weighted factors to disregard poor honeypots and preserve good honeypots based on their

number of attacks. The model was then improved by proposing two algorithms, cooperative learning and evolution, to dynamically deploy the honeypots in a distributed honeynet environment.

There is much concern about the ability to detect bait and switch systems or honeyfarms [Corey, 2003]. Due to redirection, a network delay will be introduced into the process which can be noticeable to attackers that are close to their targets. Since a high interaction device must respond to redirected traffic, this implies that many honeypots must be available at all times or a mechanism must exist to create or clone them quickly. The ability to create a honeypot quickly is a necessity so as not to increase the network delay time and not arouse the attacker's suspicions. In addition, any inconsistency between the low interaction frontend systems and the high interaction system that an attacker is redirected to may provide an indication that the victim device is a honeypot system.

2.1.7 Dynamic Honeypots

The notion of dynamic honeypots was first conceived by Lance Spitzner [Spitzner, 2003]. He had a list of wishes that would make the "perfect honeypot."

1. A plug-and-play solution that learns the environment
2. Able to deploy the proper number and configuration of honeypots in such a way to blend into the network
3. One that adapts to changes in the network by adding and removing honeypots based on the environment

He presented the idea of actively scanning or passively listening to a network to determine the types of computers and services that should be implemented in the honeypots. The information gained from learning the network would be used to create a honeynet that represents the actual network or a smaller subset. Current problems with maintaining a honeynet were addressed and proposed that with the advent of a dynamic honeypot system that some of the problems would be alleviated.

A dynamic honeynet design, Figure 11, was proposed in 2004 that put more detail to the concept of Spitzner [Kuwatly, Sraj, Masri & Artail, 2004].

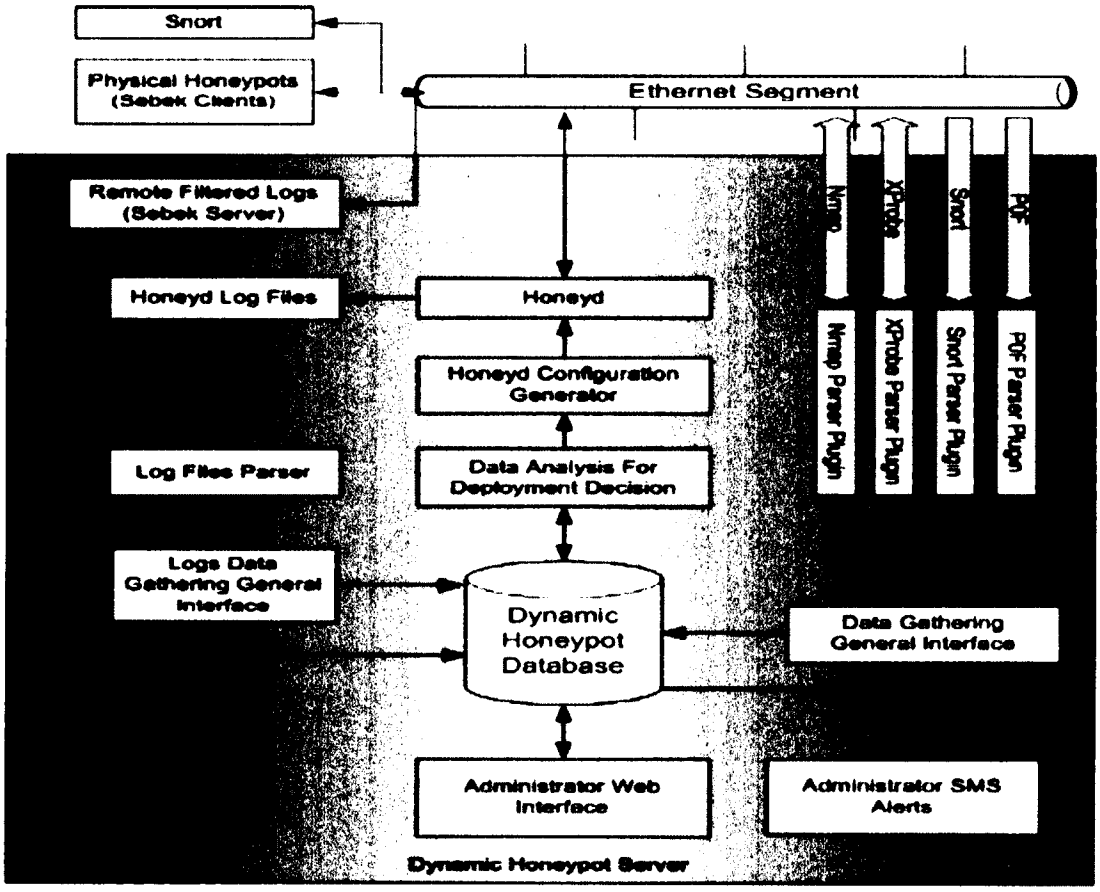


Figure 11, Dynamic honeypot server [Kuwatly et al., 2004]

The dynamic honeypot server (DHS) collects information about the environment by utilizing passive and active scanning based on the network architecture. Once the DHS has gained knowledge about the environment, a HIH configuration is suggested along with a suitable configuration for Honeyd. The LIH receives the traffic then passes it along to the HIH for a higher level of interaction. Each HIH is running a Sebek client which logs and routes the information to the DHS where the Sebek sever centralizes the logs from each system. The DHS is capable of alerting the administrator via short message service (SMS) texts. The LIH and HIH system information and log files are stored in a database which is accessible to the administrator through a web interface.

An experimental dynamic honeypot design and implementation, Figure 12, was presented in a technical report by the University of Louisville [Hieb & Graham, 2004]. The implementation followed Spitzner's design by exclusively using passive listening programs, tcpdump [Tcpdump, 2012] and p0f, to gather information about the network environment. IP addresses and open ports captured by tcpdump, along with operating systems fingerprinted by p0f are stored in the dynamic honeypot database. The dynamic honeypot configuration engine creates the Honeyd configuration file based on the database information. Honeyd is then deployed to mimic the systems already seen in the network. This system does not adjust instantly to a new system being introduced onto the network, but waits for a predetermined time to update the database and re-start Honeyd with the new configuration file. During testing of the system, the author also noticed the difficulty of identifying all of the open ports on the devices.

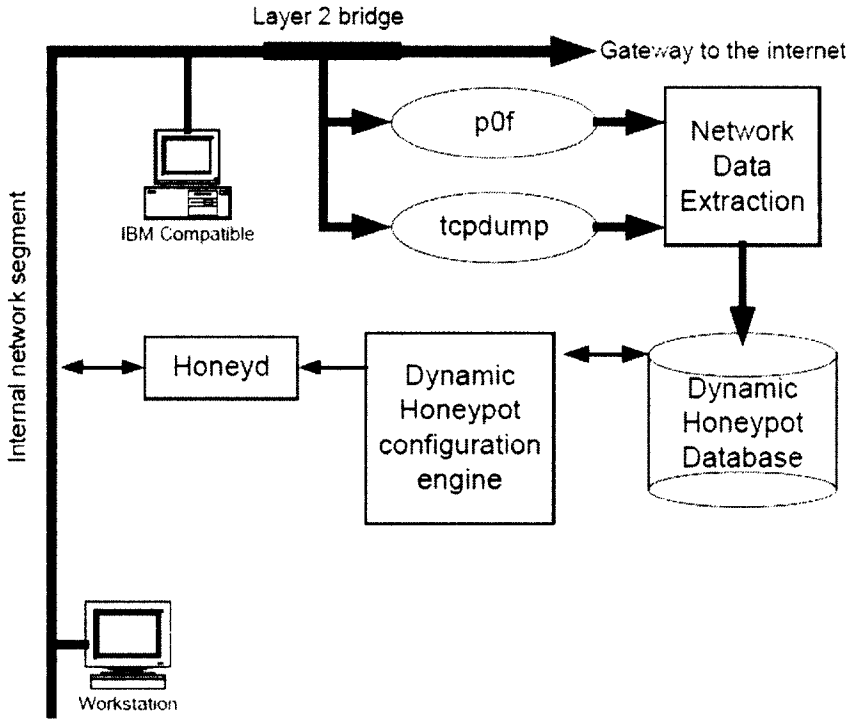


Figure 12, Dynamic passive scanning honeypot implementation [Hieb & Graham, 2004]

Communication through the layer 2 bridge was essential to recognize the open ports. External probing of all the ports with an active scanner was conducted; this allowed the passive scanner to gather the required information to better simulate the monitored network. Snort was used to capture the Honeyd traffic then generate alerts.

Active scanning was implemented in an alternate design shown in Figure 13, which dynamically adjusts to the environment on which system has gathered information [Hecker, Nance & Hay, 2006]. Nmap [Yarochkin, 2012] is utilized to scan a network to gain network and computer configurations. During initiation of the scan, four options could be selected which would determine the honeypot IP address configuration.

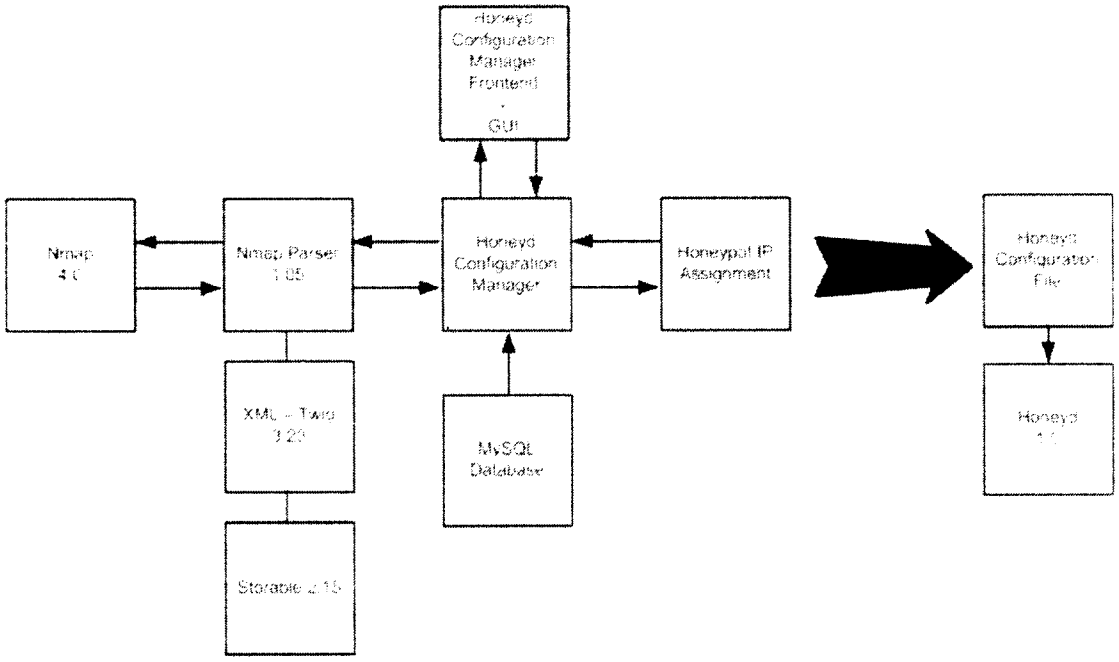


Figure 13, Dynamic, active scanning honeypot sever [Hecker et al., 2006]

The available flags were the following:

- -iS, same IP as scanned devices
- -iD, uses the same last octet of the IP with a different proceeding 24 bits
- -iR, selected a predetermined range will be used for the honeypot IP addresses
- -iI, interwove the honeypots IP addresses with the scanned production computers

The MySQL [MySQL, 2012] database is queried during creation of the Honeyd configuration file to combine the scan results with any port emulation scripts matching the scanned device's ports to enhance the resulting honeypots. Honeyd is started using the configuration file which creates the honeypots with the desired IP address, open ports and OS configurations. As with the previous dynamic honeypot system, the deployment process is self-initiated through the cron daemon or manually for the honeynet to remain current with the network. ARP packets were transmitted to find an open IP to interweave

the honeypots with the production devices. Requesting an IP from the DHCP server instead would ensure that an IP conflict did not occur. A graphical user interface (GUI) was also constructed to help the user through the process instead of relying solely on the command line.

Honeypots have been developed and advancing for the past 15 years with many different approaches. The preceding design ideas and architectures were considered in formulating this research project. Implementing a system which expands and incorporates the two dynamic honeypots designs was necessary to accurately build honeypots which resemble the network environment. Integrating passive and active scanning while storing the information in a database were key features to map the topology and record the flux of the network. Utilizing the benefits of low and high interaction honeypots is vital to collect attack statistics and detect new attacks. This research effort attempts to create a comprehensive system utilizing innovative network scanning tools to build configuration files for prevalent honeypot technology.

Chapter 3: System Development

3.1 Project Overview

Demonstrated by the literature search, there is a lack of a self-contained, dynamic honeynet system which is capable of deploying both low and high interaction honeypots. This honeynet system provides the user with the ability to scan a network passively or actively, stores the data from the scans to create a network depiction, and creates a Honeyd configuration file for deployment of low interaction honeypots and an extensible markup language (XML) file for the deployment of high interaction honeypots.

This system was designed around a core set of tools that aid in mapping a network environment. A module was built around each tool to process the resulting data gathered during operation of that tool and record the information into a database. Building in a modular fashion allows independent operation of each of the scanning modules. Commands, with a variety of flags, allow the user to initiate operation of each module. To reduce the burden on the user to constantly supervise the individual processes, two management programs were created to oversee the modules and control the creation of the honeypot configuration files. Figure 14 shows the different modules, their roles and functions, and the interactions between the modules.

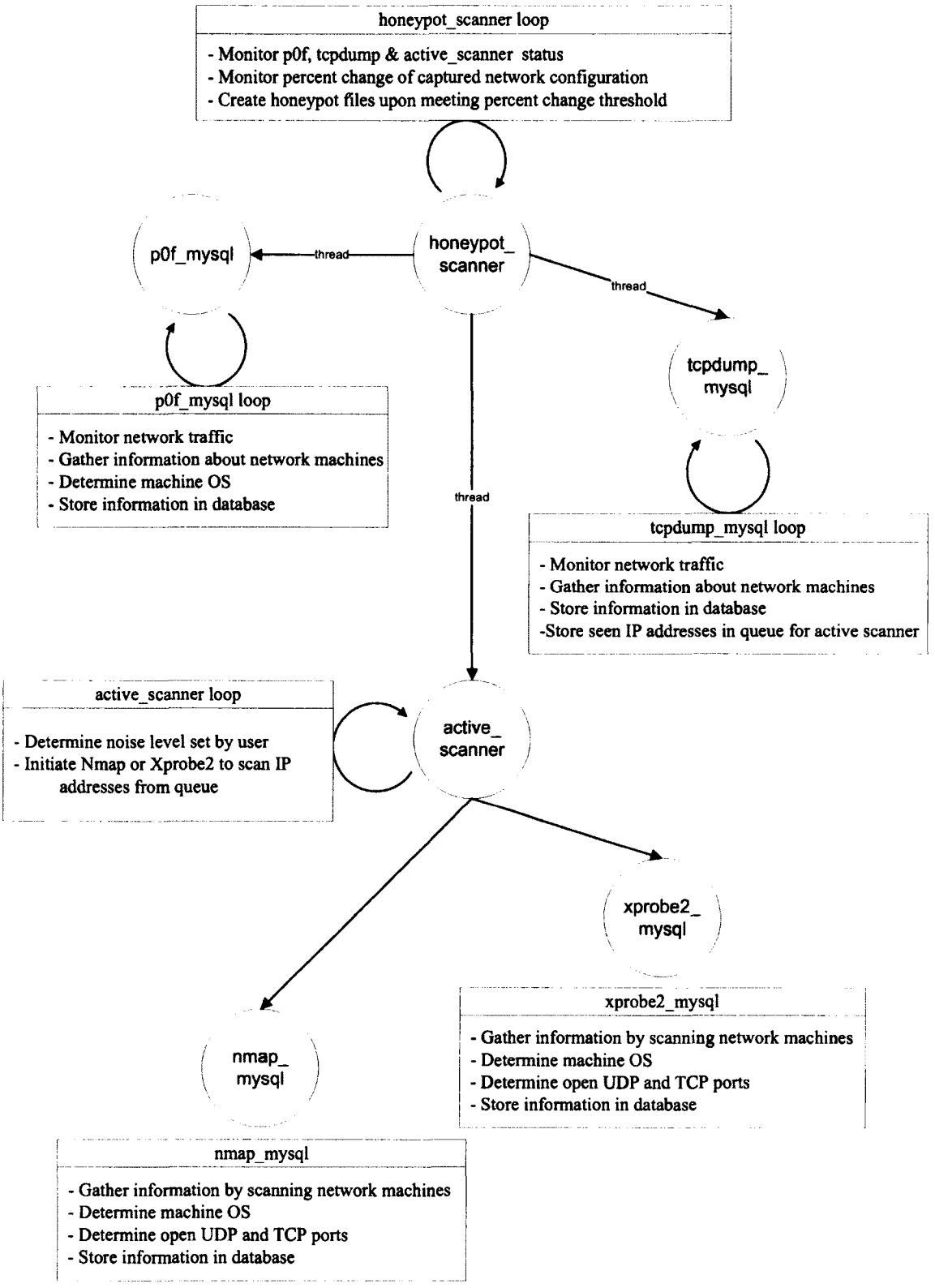


Figure 14, Project module overview [Hecker & Hay, 2010]

3.2 Management Programs

3.2.1 Objectives and Requirements

The intention of creating a management system is to alleviate the responsibility of the user to monitor the scanning process and modules, and create honeypot configuration files when the sensed network has met an identified threshold designating a "significant change." Being able to continually monitor the individual processes and redeploy if necessary was an essential feature of the management system.

3.2.2 Design and Implementation

honeypot_scanner

Honeypot_scanner is the primary management program which oversees both passive scanning modules and the secondary active scanning management program, *active_scanner*. *Honeypot_scanner* gathers the initial configuration information stored in the database and disseminates it to all the respective modules upon their deployment. As Figure 14 illustrates, the passive scanning modules and *active_scanner* are deployed as threads. The threads are monitored through the process identification (PID) to verify operation. The continual monitoring allows the re-deployment of any of these modules should they die unexpectedly. *Honeypot_scanner* also observes the results gathered from all the scanning modules and builds honeypot configuration files upon reaching a setpoint determined by the user. The initial setpoint is based on the number of devices required to be identified before deploying the honeypot configuration files for the first time.

Subsequently, the setpoint is based on the percent change of devices or services identified. So, if the initial set point required five devices to be identified and a percent change of 30%, then re-deployment of the configuration files would take place after two additional devices or services were identified or removed from the network. *Honeypot_scanner* is able to identify the removal of devices and active ports on an individual computer by updating the timestamp, associated with the device/ports, as the scanning modules gather information. If the timestamp is older than the predetermined number of seconds set by the user then the device or computer port is not added into the new iteration of honeypot configuration files. Figure 15 illustrates the decisions made by the *honeypot_scanner*.

This system is structured to create a Honeyd LIH configuration file with the same qualities as the XML HIH file. This expands the abilities of this system to create honeyfarms which allow for LIH redirectors to forward attackers to the similar HIHs. Programs have been discussed in Chapter 2 which tunnel traffic from LIHs to HIHs, and rapidly deploy VMs as HIHs. Database tables have already been created in this system to link a LIH with a HIH to facilitate the forwarding of an attacker. As previously stated, LIHs are easier to deploy and maintain in large quantities and can easily be distributed throughout an organization for alerting and redirection to a HIH if necessary.

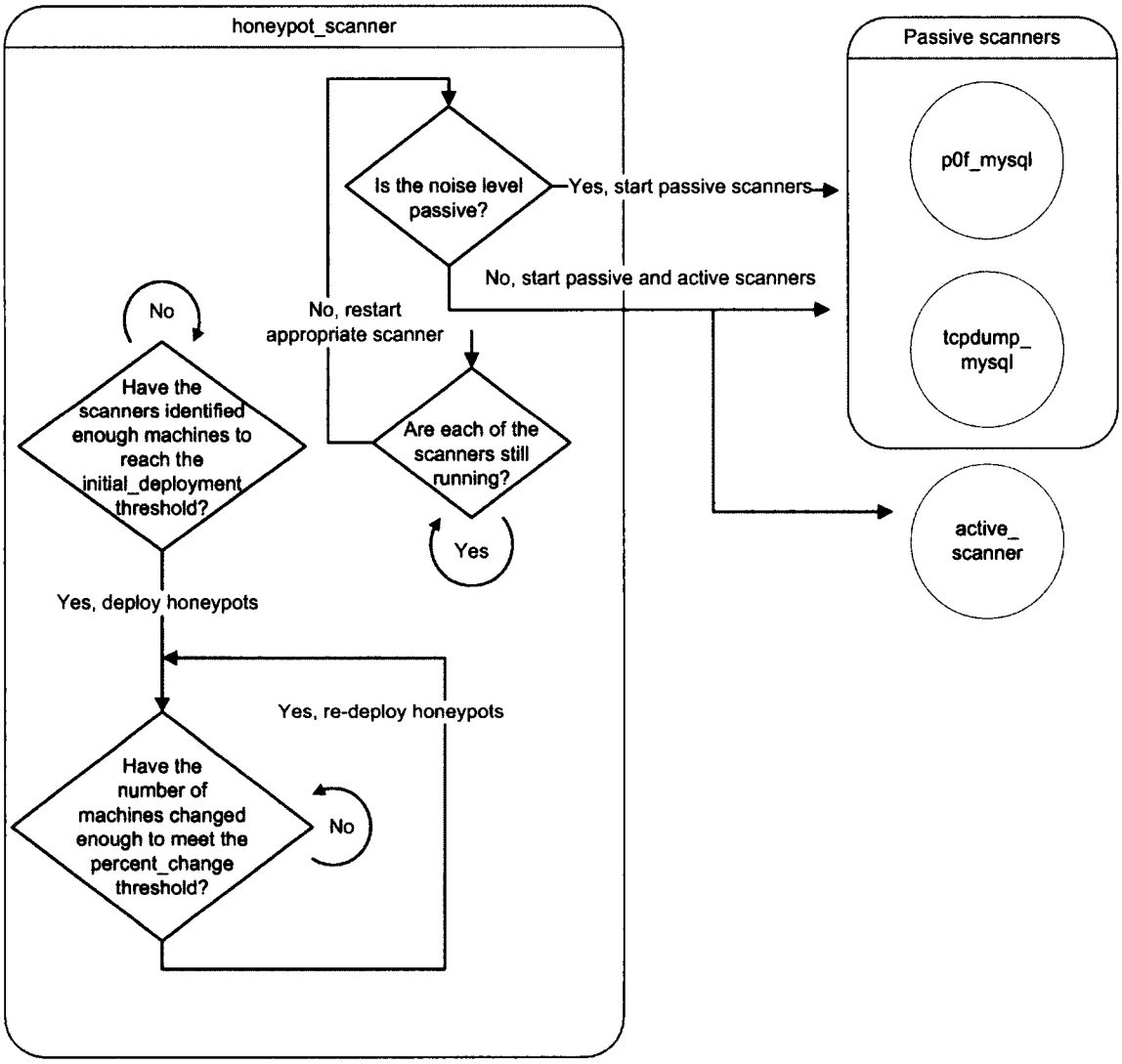


Figure 15, *honeypot_scanner* process

During the creation of the Honeyd configuration file, saved commands to execute existing port emulation scripts upon honeypot deployment are supplied. Honeyd scripts provide additional functionality to the LIHs. The XML file contains the information to configure and deploy a HIH virtual machine with similar features as the Honeyd LIH. *Honeypot_scanner* also handles the collecting IP addresses through DHCP. As the Honeyd configuration file is being created *honeypot_scanner* requests IP addresses from

the DHCP server. This allows the LIHs to be interwoven into the production network without manually assigning IP addresses or worrying about duplicate IP addresses on the network. *Honeypot_scanner* continually monitors when the IP address' lease expires and renews it for the LIH. If a production device is removed from the network, the LIH will not be redeployed and the DHCP IP address will be released.

To create a record of the network environment, the Honeyd configuration files and the XML files are backed up to a folder as new files are generated.

active_scanner

Active_scanner was developed to specially monitor the active scanning modules. Actively scanning a device can induce significant noise into the network. *Active_scanner* controls the noise by determining which scanning module is deployed, and the type of scan initiated by the sub-program. The determination of the module and type of scan is based on the noise level set by the user. Five noise levels are available to the user: passive, low, medium, med-high, and high. Though the passive scanning modules are capturing data throughout every noise level, the user can choose to have the program silently collect data by choosing the "passive" noise level. **Error! Reference source not found.** depicts the active scanning noise levels available to the user, the sub-programs used to scan the network and the commands used to initiate the scan.

As the passive scanners capture header information from the network traffic, the MAC and IP addresses are stripped from the data and input into a database table. The IP/MAC address combinations to be actively scanned are retrieved from the database by *active_scanner* and sent to the respective active scanning module

Table 2, Active scanning noise levels

Noise level	Sub-programs	Active scanning commands
low	xprobe2	xprobe2 -r -m 2 -o {file} -X {ip}
medium	xprobe2	xprobe2 -r -m 2 -o {file} -X -T 1-1024,3306 -U 1-1024 {ip}
medium-high	nmap	nmap -sT -sU -T3 -sV -O -oX {file} --host-timeout 180 {ip}
high	xprobe2 + nmap	xprobe2 -r -m 2 -o {file} -X -T 1-1024,3306 -U 1-1024 {ip} nmap -sT -sU -T3 -sV -O -oX {file} --host-timeout 180 {ip}

3.3 Network Scanning (Passive and Active)

3.3.1 Objectives and Requirements

The objective was to create as complementary set of completely independent modules that could be used separately or cooperatively. Each module would be initiated through the command line and accepts flags to perform a set of operations. Each module would store the information in unique MySQL database table(s).

3.3.2 Design and Implementation

Four modules were created to perform the active and passive scanning for the program. *P0f_mysql* and *tcpdump_mysql* passively gather information about devices from the network traffic. *Nmap_mysql* and *xprobe2_mysql* actively scan the identified devices on the network. As previously explained, the passive scanning modules are constantly gathering data about the devices on the network. *Active_scanner* takes the identified devices and initiates the respective active scanning module. Figure 16 demonstrates the decision process of each of the scanning modules and their associated database tables.

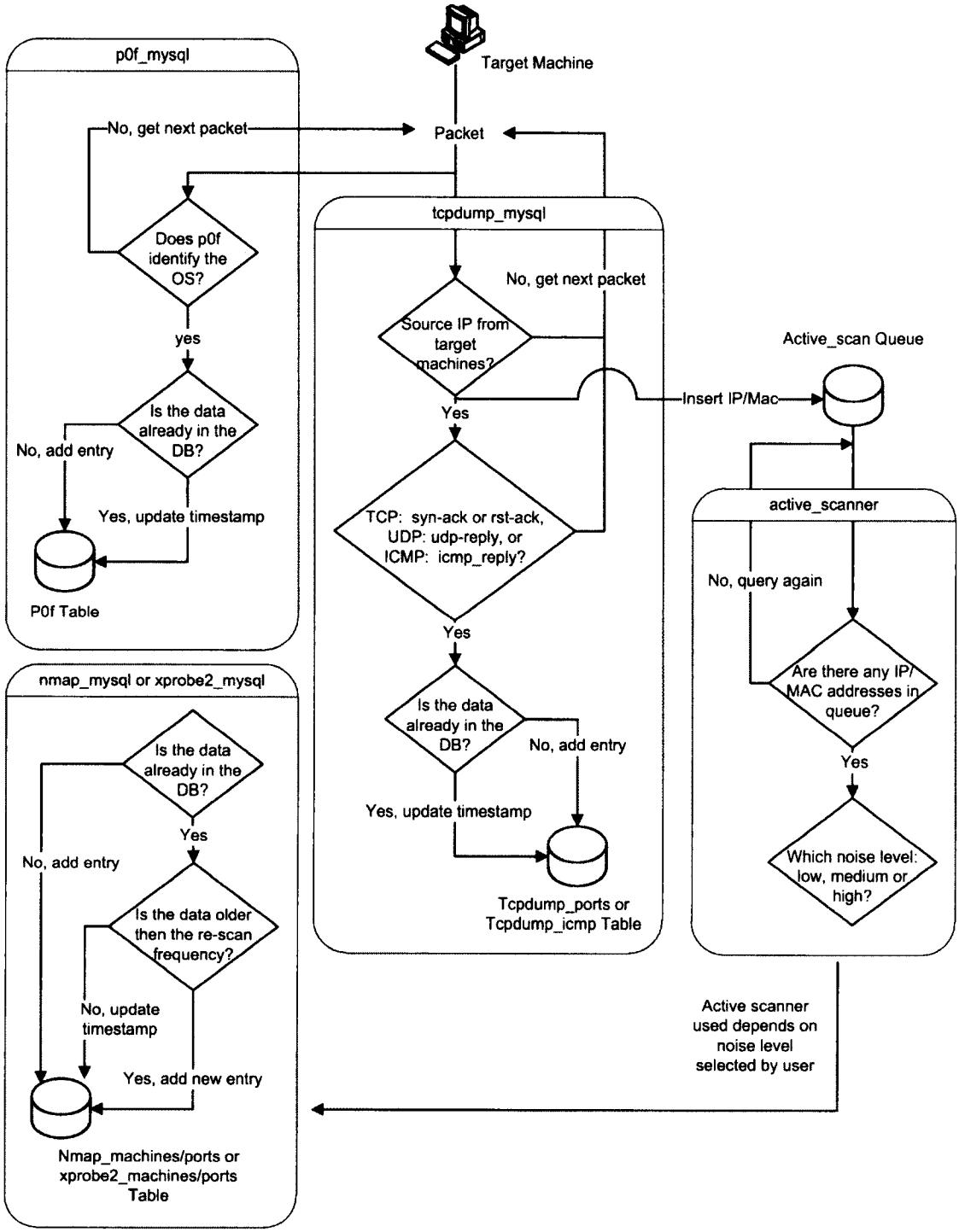


Figure 16, Scanning modules decision process

To reduce redundant data, all the scanning modules (*tcpdump_mysql, p0f_mysql, nmap_mysql* and *xprobe2_mysql*) query the database before inserting gathered data about a device or port to determine if it has been previously observed. If the data is not present, then it is stored into the database but if data is present, then the timestamp on the previous observation is updated.

3.3.2.1 Passive Scanning Modules

Two passive scanning modules are used to gather information about the devices on the network. Since they gather different data about the devices, it was necessary to use two separate sub-programs to obtain the operating system information and which ports are communicating.

p0f_mysql

P0f is passive scanning tool which uses sophisticated fingerprinting mechanisms to identify the operating system of a device by examining its TCP/IP packets [Zalewski, 2012]. P0f is also able to measure system uptime, distance in hops, and whether ports are firewalled. The command used to initiate p0f is:

```
p0f -i {interface} -p –l
```

P0f will begin listening on the interface specified after placing the interface into promiscuous mode. *P0f_mysql* parses the gathered information then stores it into the *p0f* database table.

Many users can exchange the same IP address over a relatively short period of time on a highly dynamic network plus operating systems may be upgraded; p0f queries

the database for the IP, MAC, and OS of an identified device to determine whether it has been previously stored in the database. If any of the parameters have changed or if the device is unknown, then the device information is stored in the table; otherwise the timestamp is updated.

tcpdump_mysql

Tcpdump is a packet analyzer which is used to grab packet headers to identify devices on the network and computer ports that are responding to requests [Tcpdump, 2012]. The following command is used to initiate tcpdump to begin gathering information:

```
tcpdump -nne -i {interface} src net {ip}
```

Tcpdump collects the link-level header information and does not convert the IP addresses or ports to names. Only packets with the source IP on a particular subnet are captured. This captures information about devices on the network for which a honeynet would like to be created. This subnet can range from a class A (1.0.0.0) to a specific device (1.2.3.4).

Tcpdump_mysql further filters the network traffic to ensure the resulting honeypots closely resemble the actual production device. *Tcpdump_mysql* gathers the header information from the following packets: ICMP echo reply, UDP reply, TCP SYN-ACK, and TCP RST-ACK. These packets are likely to be coming from responding ports/services. The IP and MAC addresses from the ICMP packets are stored in the *tcpdump_icmp* database table. The source IP, MAC and port number contained in the TCP and UDP packets are stored in the *tcpdump_ports* database table.

Tcpdump does not provide any details about the services that are communicating on a particular port and protocol in the packet. Nmap.org provides a file, nmap-services, which contains a list of ports and protocols along with the corresponding services. The file can be downloaded from http://nmap.org/svn/nmap-services. After removing the unknown services, the information was entered into the *nmap_services* database table via the *nmap_services_mysql* script. Once a packet has been observed, the port number is queried in the *nmap_services* database table and the service information is placed with the packet information in the *tcpdump_ports* table. Though several services may be using the same port or services may be using non-standard ports, this additional information tries to speculate which service might be using a particular port to provide better information to the user.

3.3.2.2 Active Scanning Modules

As with passive scanning, two modules are used for the active scanning portion. Each module's sub-program utilizes different network protocols to determine the operating system of the identified device. Based on initial testing which is discussed further in Chapter 5, each sub-program is able to correctly identify a different operating system with better proficiency. The speed and efficiency to correctly identify devices and network traffic generated to gather information were the factors that required the use of different active scanning modules.

nmap_mysql

Nmap is a network discovery tool which allows administrators to identify open ports and the OS which is running on a device [Yarochkin, 2012]. Nmap is highly configurable and allows the user to choose from a wide variety of scans. The command used to initiate nmap is as follows:

```
nmap -sS -sU -T3 -sV -O -oX {file} --host-timeout 180 {ip}
```

Nmap executes a TCP connect scan and UDP scan on the IP address with the following options: timing template is set to 3 to scan moderately, probes the open ports to determine service and version information, OS detection is enabled, output the results to file specified in a XML format and terminates the scan if it is not able to gather information on a host after 180 seconds.

After nmap scans the device, *nmap_mysql* parses the XML file and stores the data into two database tables. Information about the scanned device is stored in the *nmap_machines* table while information regarding open ports is stored in the *nmap_ports* table. If nmap was not able to determine information regarding a certain area of the device being scanned then "unknown" is stored in the respective field of the database table.

Regulating the duration between scans is accomplished by *active_scanner* passing a flag with a time in seconds. *Nmap_mysql* will query the database concerning that device with the IP and MAC addresses supplied in other flags before running a scan to determine if the device has formerly been scanned. If the query finds a match and returns the requested information, the time since the last scan is compared to the duration between scans to determine if a re-scan should be initiated.

xprobe2_mysql

Xprobe2 is active OS fingerprinting tool which applies different methods then nmap to gather information about a device [Ofir, Yarochkin & Kydyraliev, 2003]. Xprobe2 utilizes ICMP and UDP packets to determine the state of a device and the operating system. Xprobe2 also has the ability to scan a device for open TCP and UDP ports. As discussed earlier, two separate commands are used to initiate xprobe2 depending on the noise level. The first command is as follows:

```
xprobe2 -r -m 2 -o {file} -X {ip}
```

Xprobe2 probes the device with the IP address specified and generates primary and secondary OS deductions based on signature matching. The OS and route to target is recorded in a XML file. The second command used is the following:

```
xprobe2 -r -m 2 -o {file} -X -T 1-1024,3306 -U 1-1024 {ip}
```

This command is similar to the first but probes the TCP and UDP ports which are indicated. *Xprobe2_mysql* also regulates the duration between scanning by comparing the time designated with the –t flag that is passed from *active_scanner* and the time since the last timestamp. *Xprobe2_mysql* gathers the information about the device and ports scanned then stores the data in the *xprobe2_machines* and *xprobe2_ports* database tables.

3.4 Low Interaction Honeypot Configuration

3.4.1 Objectives and Requirements

Honeyd was chosen to be the low interaction honeypot program due to its prevalence, support and additional scripts to supplement the interaction with an attacker.

Figure 17 is an example of a Honeyd configuration file generated by *honeypot_scanner*. The objective was to combine the results from the scanning modules to create a Honeyd configuration file that is the most accurate representation of the production network.

3.4.2 Design and Implementation

Once the initial deployment or re-deployment threshold is reached, honeypot_scanner will create a configuration file with the data gathered by the scanning modules. Each device or open port that has been identified or updated, since the time between scan intervals, is added to the configuration file. Since several of the modules collect the same information, a ranking was instituted to determine the OS of the low interaction honeypot. Active scanning tools were ranked highest due to their ability to gather more information in a quicker timeframe. Nmap is the most current and readily updated tool so it was given the highest precedence. The ranking order for determining which OS is placed in the configuration file is nmap → xprobe2 → p0f. The configuration file is updated with all of the known data sets for each device that was identified and collected by the scanning modules.

```
##### Honeyd Configuration File ######
##### Sun Apr 16 20:04:09 2006 #####

########################################

create Windows1
set Windows1 personality "Microsoft Windows 2003
Server or XP SP2"
set Windows1 default tcp action reset
set Windows1 default udp action reset
set Windows1 default icmp action open
add Windows1 tcp port 21 open
add Windows1 tcp port 23 "perl /telnet/faketelnet.pl"
add Windows1 tcp port 25 open
add Windows1 tcp port 80 open
add Windows1 tcp port 110 open
add Windows1 tcp port 143 open
add Windows1 tcp port 143 open
add Windows1 udp port 123 open
add Windows1 udp port 135 open
add Windows1 udp port 137 open
add Windows1 udp port 138 open
add Windows1 udp port 139 open
add Windows1 udp port 445 open
add Windows1 udp port 500 open
add Windows1 udp port 1900 open
add Windows1 udp port 4500 open
add Windows1 udp port 31337 open
set Windows1 ethernet "Intel Corporate"
bind 192.168.192.103 Windows1

########################################
```

Figure 17, Honeyd configuration file [Hecker et al., 2006]

The IP address for each low interaction honeypot is decided by a setting chosen by the user in the *config* database table. There are four different settings available to the user, and each setting has a different purpose depending on the honeynet application of the user:

- -iS, the same IP addresses as the scanned computers
- -iD, the same host value, but a different subnet
- -iR, assigned in a subnet selected by the user

- -il, interwoven utilizing DHCP betwixt the existing IP addresses of the scanned host

As the configuration file is being created, the *lih_hih_link* table is being populated simultaneously. This table allows the low interaction honeypot and high interaction honeypot pairs to be associated. This facilitates future modules for the purpose of redirecting interesting traffic from the low interaction honeypots to the high interaction honeypots.

3.5 High Interaction Honeypot Configuration

3.5.1 Objectives and Requirements

The XML file is generated by *honeypot_scanner* to allow the user to generate a high interaction honeypot with any type of virtual machine application or to use as a formula to create a standalone high interaction honeypot. Though the devices included in the XML file are similar to the ones found in the Honeyd configuration file, they do not have to be used in conjunction with one another. Figure 18 is an example of a XML file generated by *honeypot_scanner*.

```
<?xml version="1.0"?>
<honeypot>
  <target name="Linux1" ip="192.168.192.150" mac="00:04:00:6D:0D:47" lih_hih_id="1">
    <os_guess>
      <primary> "Linux Kernel 2.4.0" </primary>
    </os_guess>
    <system_information>
      <icmp_reply state="closed"/>
      <firewall state="no/unknown"/>
      <lookup_link state="unknown"/>
      <last_reboot seconds="unknown"/>
      <real_time_target_seconds seconds="0.00332"/>
      <uptime_seconds time="unknown"/>
      <distance_hops hops="unknown"/>
    </system_information>
    <port_scan>
      <port number="21" protocol="tcp" service="ftp" version="unknown" extra_info="unknown"/>
      <port number="79" protocol="tcp" service="finger" version="unknown" extra_info="unknown"/>
      <port number="80" protocol="tcp" service="www" version="unknown" extra_info="unknown"/>
      <port number="515" protocol="tcp" service="printer" version="unknown" extra_info="unknown"/>
      <port number="631" protocol="tcp" service="ipp" version="unknown" extra_info="unknown"/>
    </port_scan>
  </target>
  <target name="Windows2" ip="192.168.192.171" mac="00:0C:29:20:E3:5E" lih_hih_id="2">
    <os_guess>
      <primary> "Microsoft Windows XP SP2" </primary>
    </os_guess>
    <system_information>
      <icmp_reply state="closed"/>
      <firewall state="no/unknown"/>
      <lookup_link state="unknown"/>
      <last_reboot seconds="unknown"/>
      <real_time_target_seconds seconds="0.00624"/>
      <uptime_seconds time="unknown"/>
      <distance_hops hops="unknown"/>
    </system_information>
    <port_scan>
      <port number="80" protocol="tcp" service="www" version="unknown" extra_info="unknown"/>
      <port number="135" protocol="tcp" service="loc-srv" version="unknown" extra_info="unknown"/>
      <port number="445" protocol="tcp" service="microsoft-ds" version="unknown" extra_info="unknown"/>
      <port number="3306" protocol="tcp" service="mysql" version="unknown" extra_info="unknown"/>
    </port_scan>
  </target>
</honeypot>
```

Figure 18, XML file

3.5.2 Design and Implementation

The XML file is not created with any libraries; the output file is generated using specific print commands. The XML file is created with the idea that it will be used to configure a real system from a blank VM template. An administrator would be able to create a plain VM template utilizing the OS that is used for a specific set of production devices. As HIHs are needed, based on those templates, a module would be able to clone the VM and configure the high interaction honeypot based on the XML configuration file. This type of rapid deployment of VM systems is already being utilized in a higher education environment. The Remote Access Virtual Environment (RAVE) project through the ASSERT Lab at the University of Alaska, Fairbanks, rapidly deploys and configures VMs from an XML based file [B. Hay, personal communication, May 30, 2012].

Chapter 4: Database System Design

4.1 Database System Overview

A single database design has been implemented to store the information necessary for the system's operation and data gathered during scanning. The database design has been broken into four components: 1) Network Scanning – Operation, 2) Network Scanning (Passive and Active) – Data, 3) Low Interaction Honeypots, and 4) High Interaction Honeypot. Multiple tables may be associated with each of these components in the database implementation. The database schema and the statements responsible for creating and inserting information data into the tables are located in Appendix E. In the following chapter, each component and table will be discussed along with the table fields to explain their relevance to the system.

4.2 Network Scanning – Operation

4.2.1 Objectives and Requirements

The purpose of the tables connected with the operation of the system is to provide the user a convenient place to store the desired traits and functionality and monitor the various modules without additional input from the user. While gathering the user input could have been accomplished through a complex command line statement or a configuration file, database tables provide the user a view of the required information and

the necessary format. The database also allows the user to change the functionality of the system on-the-fly by updating the operational tables.

4.2.2 Design and Implementation

There are three tables specifically dedicated with the operational duties of the system: *config*, *threads* and *honeypot_updates*.

config

The *config* table contains all the necessary information for the startup and operations of the system and its modules. Some of the database entries are for files which Honeyd uses to emulate different OSs, determine how to react to ICMP fingerprinting, and create dynamic templates. Though Honeyd is not currently deployed from this system, the database entries are in place to add in a small section of code to make it possible.

- eth_interface
  - Ethernet interface that the active and passive will utilize to gather information
- mac_addr
  - MAC address of the Ethernet interface which will allow the scanning modules to filter out any communication with the system
- dhcp_server
  - Allows the honeypot_scanner to request, renew or release IP addresses for the low interaction honeypots
- p0f_os
  - File location for ps.of for deploying Honeyd
- nmap_exe

  - Application for active_scanner to pass to *nmap_mysql*
- nmap_prints
  - File location for nmap.prints for deploying Honeyd
- nmap_assoc
  - File location for nmap.assoc for deploying Honeyd
- nmap_xml
  - Output location for nmap XML file
- xprobe2_conf
  - File location for xprobe2.conf for deploying Honeyd
- xprobe2_xml
  - Output location for xprobe2 XML file
- honeypot_xml
  - Output location for honeypot XML configuration file
- honeyd_config
  - Output location for Honeyd configuration file
- honeyd_ip_binding
  - Enables the user to choose which IP addresses are used in the Honeyd configuration file, discussed in section 3.4.2 Design and Implementation
- honeyd_ip_range
  - IP range that the user determines to use for the low interaction honeypot, used in conjunction with iD and iR of the honeyd_ip_binding.
    - iD, the low interaction honeypot will retain the same last octet as the system scanned but will use the first three octets of the honeyd_ip_range IP address.
    - iR, will begin configuring the low interaction honeypots with the beginning IP address in the range and will terminate if the range has been filled

- scan_ip_range
  - IP subnet from which the system will be gathering information to create configuration files
- initial_deployment
  - Number of identified devices to generate initial configuration files
- percent_change
  - Change in percent value that needs to occur before subsequent generation of the configuration files
- noise
  - Amount of noise that user deems acceptable to introduce to the network with the passive and active scanning modules, discussed in section 3.2.2 Design and Implementation *active_scanner*
- active_scan_seconds
  - Number of seconds between active scanning sessions which aids in regulating the noise introduced to the network
  - Timeframe in which devices need to be identified or updated to be included in the configuration files which keeps the honeypots relevant to the current network
- date_created
  - Date that the table was created

threads

The *threads* table aids in monitoring the passive scanning modules and the secondary active scanning management program is. *Threads* acts as a lookup table for the process identification (PID) for *p0f_mysql*, *tcpdump_mysql* and *active_scanner*. Each of the modules updates the *threads* table upon being deployed. *Honeypot_scanner* can track

the modules to ensure they are still running, and take necessary actions if a program has terminated.

- thread_id
  - Unique ID for this thread
- thr_name
  - Name of the module (tcpdump_scan, p0f_scan, active_scan)
- thr_pid
  - Process identification for the thread, which is polled by *honeypot_scanner* to ensure the module has not terminated
- last_tstamp
  - Last timestamp of the thread's creation

honeypot_updates

The *Honeypot_updates* table contains a continuous record of all the deployments of honeypot configuration files. This allows the user to gauge the dynamic nature of the network environment and identify patterns which can be linked to certain periods of time or situations. Hours of network traffic from individual computers can be tracked and incorporated into an anomaly based IDS, cycles in a school year can be recorded, rogue devices on an organization's network can be traced, and intrusions can be observed by identifying new and unusual open ports on a device.

- update_id
  - Unique ID for this honeypot update
- num_machines
  - Number of total devices identified since predetermined number of seconds (active_scan_seconds)
- num_services

  - Number of total services gathered since predetermined number of seconds (active_scan_seconds)
- date_updated
  - Time and date of the last update

scan_queue

The *scan_queue* table acts as a queue between the passive scanning modules and *active_scanner*. *Scan_queue* provides a place for the passive scanning program to insert identified computers and allows the *active_scanner* to remove the information to initiate the correct active scanning module. The queue was designed to decrease the need for constant communication and interrupts between the two programs. The queue allows the modules to insert and remove the information as needed.

- scan_id
  - Unique ID for this device information
- ip_addr
  - IP address of the identified device
- mac_addr
  - MAC address of the identified device
- date_created
  - Time and date of the devices identification
- last_tstamp_time
  - Last timestamp for seeing the identified device (UNIX time)

4.3 Network Scanning (Passive and Active) – Data

4.3.1 Objectives and Requirements

During operation scan results are queried, inserted and updated to eliminate duplicate data which ensures relevant information is utilized in the generation of the honeypot configuration files. Data collected from each module is stored in a separate table(s). Separate tables allow the user to review the information gathered by each of the modules and is specially tailored to each type of information.

4.3.2 Design and Implementation

There are seven tables used to store the gathered information from the scanning modules: *p0f, tcpdump_icmp, tcpdump_ports, nmap_machines, nmap_ports, xprobe2_machines* and *xprobe2_ports*. An additional table (*nmap_services*) is included in this section because it is used as a lookup table for *tcpdump_mysql*. The data collected by the sub-program within each module is well documented; likewise a thorough database schema for each table is also presented in Appendix E. The table layout of the data pertaining to each scanning module will be discussed along with any supplemental information. Though some of the sub-programs within a module might have the ability to collect more information, only the data necessary to build complete Honeyd and high interaction honeypot configuration files were accumulated.

P0f stores the information gathered by the p0f_mysql program. As p0f identifies a device, it deduces the OS and collects valuable information about the system's distance in hops, firewall status, system uptime and type of network connection.

Tcpdump_mysql stores its data into two different tables, *tcpdump_icmp* and *tcpdump_ports*. The types of data collected from the ICMP vs. UDP/TCP packets required that separate tables be used. The IP and MAC addresses from the ICMP reply packets are stored into the *tcpdump_icmp* table. The *tcpdump_ports* stores the IP and MAC addresses, port number, and records whether the packet was a UDP reply, TCP SYN-ACK, or RST-ACK. The additional lookup table (*nmap_services*) is used to supplement the information gathered by *tcpdump_mysql*. As *tcpdump_mysql* intercepts the specified UDP/TCP packets, *nmap_services* is queried to collect the service and extra information about the port number and protocol supplied.

Though *Nmap_mysql* and *xprobe2_mysql* gather slightly different information about the scanned devices, their data is broken apart in a similar fashion. Information that is specific to the device is stored in the *nmap_machines* and *xprobe2_machines* tables. The device's observed port information is stored in the *nmap_ports* and *xprobe2_ports* tables. Since one device can have multiple ports open (or services running), each device in the *nmap_machines* and *xprobe2_machines* tables has an identification number associated. The device's observed ports are linked to the device through the identification number. This schema helps reduce duplicate information.

4.4 Low Interaction Honeypots

4.4.1 Objectives and Requirements

While *honeypot_scanner* is creating the Honeyd configuration files, information is being stored into two tables and another table in being queried. These tables keep track

for information which enable the low interaction honeypot to be used to their fullest potential.

4.4.2 Design and Implementation

dhcp

As the configuration file for the low interaction honeypots are created, the Honeypot_scanner requests for IP addresses from the DHCP server. All the necessary information is stored in the *dhcp* database table to keep the low interaction honeypots operational without conflicts in the network.

- dhcp_id
  - Unique ID for DHCP interaction
- ip_addr
  - Client IP address from DHCP server
- mac_addr
  - Client MAC address
- lease_time_seconds
  - Time in seconds to renew DHCP lease
- renewal_tstamp_time
  - Timestamp of the next DHCP renewal (UNIX time)
- date_created
  - Time and date of the DHCP request
- last_tstamp
  - Timestamp of the last DHCP renewal
- last_tstamp_time
  - Timestamp of the last DHCP renewal (UNIX time)

honeyd_scripts

Honeyd_scripts contains a record of all the scripts which enhance Honeyd's interaction with an attacker. *Honeypot_scanner* will query *honeyd_scripts* while creating the configuration file and insert a script that matches an open port on the scanned device.

- script_id
  - Unique ID for the script
- primary_os
  - OS for which the script is designed to emulate
- protocol
  - Network protocol of the service
- port_num
  - Port number for the service being emulated
- script_lang
  - Language to interpret the script
- path_and_filename
  - Location and name of the script

lih_hih_link

The *lih_hih_link* database table is used to associate a LIH with a HIH. This table is populated with the LIH information as the Honeyd configuration file is being generated. The HIH portion is not currently populated but allows for future modules to insert the necessary information to facilitate honeyfarm type architecture.

- link_id
  - Unique ID for the LIH and HIH link
- lih_os_platform
  - OS platform for the LIH

- lih_ip_addr
  - IP address for the LIH
- lih_mac_addr
  - MAC address for the LIH
- hih_os_platform
  - OS platform for the HIH
- hih_ip_addr
  - IP address for the HIH
- hih_mac_addr
  - MAC address for the HIH
- hih_location
  - Location of the HIH template file; the statement starts with the [datastore] that is used for the VMware Server
- hih_state
  - State of the HIH VM (ON, OFF, SUSPENED)
- date_created
  - Time and date for creating LIH and HIH link
- last_tstamp
  - Last timestamp for deploying LIH

4.5 High Interaction Honeypots

4.5.1 Objectives and Requirements

Though the high interaction honeypot table is not currently used in this system, the tables allow for an expanded design which incorporates and deploys both low and high interaction honeypots.

4.5.2 Design and Implementation

vmware_template

Vmware-template was created to store information about VM templates. These templates will be used in future modules to deploy high interaction honeypots with the criteria specified in the XML configuration files. As the low interaction honeypots interaction with an attacker, the *vmware-template* table will be queried to find a suitable match to the OS platform of low interaction honeypot. As the new high interaction honeypot VM is dynamically deployed, the association is made in the lih_hih_link table.

- hih_id
  - Unique ID for the high interaction honeypot template
- os_platform
  - OS platform for the high interaction honeypot
- location
  - Location of the high interaction honeypot template file; the statement starts with the [datastore] that is used for the VMware Server
- date_created
  - Time and date for creating the high interaction honeypot template

Chapter 5: Testing

5.1 Testing Overview

Tests were conducted with the five different noise levels and two types of scanning with the nmap sub-program. The network was arranged similarly for each test to assess the ability of the sub-programs to identify the devices and measure the time needed to deploy the initial and subsequent honeypot configuration files. Different distributions and variations of Linux and Windows operating systems were used to evaluate the ability to identify the distinct variations and determine which sub-program worked best for each of the various operating systems. Figure 19 shows the topology of the network.

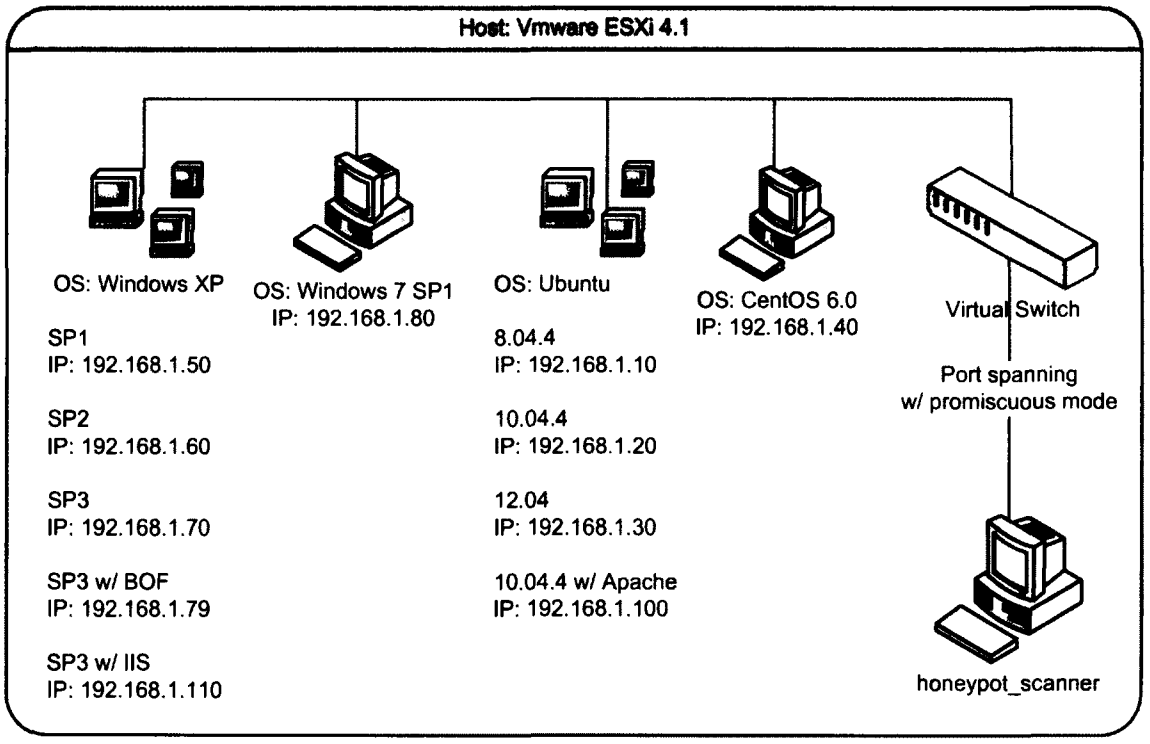


Figure 19, Testing - Network topology

The operating systems were installed as virtual machines (VMs) on a VMware ESXi 4.1 server. The VMs were connected via a virtual switch that was set to enable promiscuous mode so the *honeypot_scanner* could observe the network traffic. Traffic was generated by browsing with internet Explorer or Firefox from each of the different devices to one of the two webservers: Windows XP VM running the internet Information Services (IIS) application or Ubuntu 10.04.4 running the Apache webserver application. In each test, the *honeypot_scanner* program would be started on an Ubuntu 10.04.4 VM then the "production" VMs would browse to one of the webservers. In between each test the *honeypot_scanner* program would be stopped, the web browsers would be closed on each "production" VM, the data from the previous test was collected from the database, the database tables would be cleared of all information pertaining to the previous run, and the noise level in the *config* table on the database would be changed to reflect the next test.

The *config* table, shown in Figure 20 and discussed in previous chapters, allows the user to setup the system prior to scanning an organization's network. The eth0 Ethernet interface was set to observe the traffic for the testing phase. As the honeypot configuration files were being generated, the "iD" Honeyd IP binding would deploy the honeypots with the same IP address as the scanned computers. This was done for testing purposes to quickly determine the devices identified by the system. The initial honeypot deployment threshold was set to require five devices to be identified and a 30% change in the devices or ports observed for each subsequent deployment. The active scan or rescan

seconds were set to 86400, which would require the system to operate 24 hours before rescanning a device.

ethernet interface	eth0
MAC Address	aa:00:04:00:0a:04
DHCP Server	192.168.1.1
p0f OS file	/usr/local/share/honeyd/pf.os
Nmap executeable PATH	/usr/local/share/nmap
Nmap Prints PATH	/usr/local/share/honeyd/nmap.prints
Nmap Associates PATH	/usr/local/share/honeyd/nmap.assoc
Nmap Output PATH	/home/<user>/Output_files/nmap_output.xml
Xprobe2 Configuration PATH	/usr/local/share/honeyd/xprobe2.conf
Xprobe2 Output PATH	/home/<user>/Output_files/xprobe2_output.xml
Honeypot XML PATH	/home/<user>/Output_files/honeypot.xml
Honeyd Config PATH	/home/<user>/Output_files/honeyd.conf
Honeyd IP Binding	iD
Honeyd IP Range	192.168.1
Scan IP Range	192.168.1
Honeypot Initial Deployment	5
Percent change - redeployment	30
Noise Level	passive
Active Scan (sec) - rescan	86400
Date Created	06/16/12 10:05 AM

Figure 20, Testing - *config* table settings

Before the tests were initiated, "netstat –an" or "netstat –al" was executed using the command line on each VM to determine which ports were listening. "Uname –r" was executed on each Linux VM to ascertain the kernel version. To determine the IP and MAC address for each computer, the "ipconfig/all" or "ifconfig" commands were utilized. The configuration for each VM is found in Figure 21. Back Officer Friendly (BOF) was installed on a Windows XP service pack (SP) 3 VM. BOF was enabled to respond to TCP traffic on port 21 (FTP), port 25 (SMTP), port 80 (HTTP), port 110 (POP3), and port 143 (IMAP). BOF was installed on one of the computers to assess the ability of the scanners.

OS	IP address	MAC address	TCP ports	UDP ports
Ubuntu - *honeypot_scanner, 2.6.32-41-generic*	192.168.1.200	AA:00:04:00:0A:04		
Linux				
Ubuntu 8.04.4, 2.6.24-41-generic	192.168.1.10	00:50:56:B1:00:17		5353, 58021
Ubuntu 10.04.4, 2.6.32-41-generic	192.168.1.20	00:50:56:B1:00:15		5353, 43256
Ubuntu 12.04, 3.2.0-24-generic-pae	192.168.1.30	00:50:56:B1:00:11		5353, 42728
CentOS 6, 2.6.32-220.17.1.el6.i686	192.168.1.40	00:50:56:B1:00:1C	22, 111, 5672, 42800	111, 605, 631, 688, 5353, 37927, 41743
Ubuntu 10.04.4 - Apache, 2.6.32-41-generic	192.168.1.100	00:50:56:B1:00:13	22, 80	5353, 55189
Windows				
XP SP1	192.168.1.50	00:50:56:B1:00:0B	135, 139, 445, 1025, 5000	123, 137, 138, 445, 500, 1026, 1027, 1900
XP SP2	192.168.1.60	00:50:56:B1:00:0D	135, 139, 445	123, 137, 138, 445, 500, 1900, 4500
XP SP3	192.168.1.70	00:50:56:B1:00:0E	135, 139, 445	123, 137, 138, 445, 500, 1900, 4500
XP SP3 w/ Back Officer Friendly	192.168.1.79	00:50:56:B1:00:03	21, 23, 25, 80, 110, 135, 139, 143, 445	123, 137, 138, 445, 500, 1900, 4500
Windows 7 SP1	192.168.1.80	00:50:56:B1:00:18	135, 139, 445, 5357, 49152-6	123, 137, 138, 1900, 3702, 5355, 61582
Windows XP SP3 - IIS	192.168.1.110	00:50:56:B1:00:16	25, 80, 135, 139, 443, 445, 1025	123, 137, 138, 445, 500, 1900, 3456, 4500

Figure 21, Testing - Scanned network

5.1.1 Passive Scanning

During the passive scanning test, additional traffic was not generated from the *honeypot_scanner*. Due to the passive method of acquiring information, this test was allowed to run for the longest period of time, 41 hours and 42 minutes. As seen in Figure 22, the initial honeypot configuration file was deployed in 2 minutes and 47 seconds. Five computers were identified but none of the open ports were identified by that point in

time. The subsequent configuration file deployment would take an additional 24 minutes when another computer and the first port were observed. Then demonstrating the dynamic nature of the system, another configuration file was deployed after the 24 hour rescan threshold was reached. Devices which were not identified by p0f in the past 24 hours were removed from the third honeypot configuration file. Tcpdump was able to observe the HTTP ports on the webservers and a majority of the NetBIOS ports on the Windows computers. P0f correctly identify 5 of the 6 computers as having Windows operating systems but could only recognize one of the Linux computers.

Passive

IP address	MAC address	State	OS	TCP ports	UDP ports	ICMP reply
Linux						
192.168.1.10	00:50:56:B1:00:17		Linux 2.6			
192.168.1.20	00:50:56:B1:00:15					
192.168.1.30	00:50:56:B1:00:11		UNKNOWN [S10:64:1:60:M1460,S,T,N,W3:::?:?]			
192.168.1.40	00:50:56:B1:00:1C					
192.168.1.100	00:50:56:B1:00:13			80		
Windows						
192.168.1.50	00:50:56:B1:00:0B			139	137, 138	
192.168.1.60	00:50:56:B1:00:0D		Windows 2000 SP4, XP SP1+		137, 138	
192.168.1.70	00:50:56:B1:00:0E		Windows 2000 SP4, XP SP1+		137, 138	
192.168.1.79	00:50:56:B1:00:03		Windows 2000 SP4, XP SP1+		137, 138	
192.168.1.80	00:50:56:B1:00:18		Windows XP/2000		137	
192.168.1.110	00:50:56:B1:00:16		Windows 2000 SP4, XP SP1+	80	137, 138	

	Timestamp	Numeric_Time	Time to deploy honeypot files	Deployed Machines	Deployed Services
START	2012-6-16 11:46:55	41076.49091			
HP 1	2012-6-16 11:49:42	41076.49285	0:02:47	5	0
HP 2	2012-6-16 12:14:21	41076.50997	0:27:26	6	1
HP 3	2012-6-17 11:48:19	41077.49189	24:01:24	3	1
STOP	2012-6-18 5:29:30	41078.22882			
	TOTAL (hh:mm:ss)	41:42:35			

Figure 22, Testing - Passive scanning

5.1.2 Passive and Active Scanning

Noise Level – Low

The low noise level is the first test in which active scanning was utilized in determining the devices on the network. Xprobe2 was used to actively scan the IP addresses that were stored in the *scan_queue* database table by the *tcpdump_mysql* module. Since active scanning requires less time to discover the devices on the network, the test was allowed to run for 21 hours and 44 minutes and the results are shown in Figure 23.

Low

IP address	MAC address	State	OS	TCP ports	UDP ports	ICMP reply
Linux						
192.168.1.10	00:50:56:B1:00:17	U	Linux Kernel 2.4.30			Y
192.168.1.20	00:50:56:B1:00:15	U	Linux Kernel 2.4.30			Y
192.168.1.30	00:50:56:B1:00:11	U	Linux Kernel 2.4.30			
192.168.1.40	00:50:56:B1:00:1C	U	Linux Kernel 2.6.6			
192.168.1.100	00:50:56:B1:00:13	U	Linux Kernel 2.4.30	80		Y
Windows						
192.168.1.50	00:50:56:B1:00:08	U	Microsoft Windows XP	139	137, 138	Y
192.168.1.60	00:50:56:B1:00:0D	D	Windows 2000 SP4, XP SP1+		137, 138	
192.168.1.70	00:50:56:B1:00:0E	D	Windows 2000 SP4, XP SP1+		137, 138	
192.168.1.79	00:50:56:B1:00:03	D			137, 138	
192.168.1.80	00:50:56:B1:00:18	D	Windows XP/2000		137	
192.168.1.110	00:50:56:B1:00:16	D		80	137, 138	

	Timestamp	Numeric_Time	Time to deploy honeypot files	Deployed Machines	Deployed Services
START	2012-6-18 6:11:5	41078.2577			
HP 1	2012-6-18 06:12:09	41078.25844	0:01:04	5	5
HP 2	2012-6-18 06:12:39	41078.25878	0:01:34	7	5
HP 3	2012-6-18 06:14:40	41078.26019	0:03:35	10	5
STOP	2012-6-19 3:55:16	41079.16338			
	TOTAL (hh:mm:ss)	21:44:11			

Figure 23, Testing - Active scanning (Low noise level)

The time required to deploy three different honeypot configuration files was just three minutes and 35 seconds. Ten devices and five services were observed during that timeframe. Xprobe2 utilizes ICMP packets to determine the operating system of a device. This allowed the system to record that several computers replied to ICMP request packets. Xprobe2 was able to recognize one of the Windows computers and all of the Linux computers though the Linux kernel version was incorrect. P0f recognized several of the other Windows operating systems. Again, tcpdump was able to record the same UDP and TCP traffic as in the passive test plus the ICMP traffic. Though additional ports were observed, they did not meet the 30% threshold to deploy a fourth set of configuration files.

Noise Level – Medium

Xprobe2 with port scanning was used during the medium noise level test. The port scanning feature allowed the system to identify a few new open ports but hindered p0f from recognizing any of the Windows operating systems. The test was allowed to run for 9 hours and 18 minutes but only required 3 minutes and 50 seconds to deploy its one and only set of confirmation files. Figure 24 displays the devices and ports identified during the test. Due to the web browser caching the web pages and not sending a GET request to the webserver, the device with IP address 192.168.1.20 was not identified. By identifying this additional device, an additional deployment of configuration files may have been completed.

Medium

IP address	MAC address	State	OS	TCP ports	UDP ports	ICMP reply
Linux						
192.168.1.10	00:50:56:B1:00:17	U	Linux Kernel 2.4.30		62402	
192.168.1.20	00:50:56:B1:00:15					
192.168.1.30	00:50:56:B1:00:11	U	Linux Kernel 2.4.30			
192.168.1.40	00:50:56:B1:00:1C	U	Linux Kernel 2.4.22	22		
192.168.1.100	00:50:56:B1:00:13	U	Linux Kernel 2.4.30	22, 80		Y
Windows						
192.168.1.50	00:50:56:B1:00:08	U	Microsoft Windows XP	139, 445	137, 138	
192.168.1.60	00:50:56:B1:00:0D	D			137, 138	
192.168.1.70	00:50:56:B1:00:0E	D			137, 138	
192.168.1.79	00:50:56:B1:00:03	D			137, 138	
192.168.1.80	00:50:56:B1:00:18	D			137	
192.168.1.110	00:50:56:B1:00:16	D		80	137, 138	

	Timestamp	Numeric_Time	Time to deploy honeypot files	Deployed Machines	Deployed Services
START	2012-6-19 5:2:26	41079.21002			
HP 1	2012-6-19 5:6:16	41079.21269	0:03:50	7	8
STOP	2012-6-19 14:21:23	41079.59818			
	TOTAL (hh:mm:ss)	9:18:57			

Figure 24, Testing - Active scanning (Medium noise level)

Noise Level – Medium-High

Nmap: -sS (TCP SYN scan)

Two medium-high noise level tests were run to examine which nmap scan was more effective on this network environment. The –sS flag was set during the execution of the nmap sub-program which initiates a TCP SYN scan on each device. As seen in Figure 25, all of the devices were identified with a state of being UP but none of the Linux operating systems were recognized. P0f and nmap were only able to recognize one Windows operating system each. Nmap was able to observe a few more open ports than previous tests, especially on the Windows SP1 computer. Two deployments of

configurations files were completed though they did take longer to complete due to the extensive scanning that nmap performs.

Medium-High-sS

IP address	MAC address	State	OS	TCP ports	UDP ports	ICMP reply
Linux						
192.168.1.10	00:50:56:B1:00:17	U				
192.168.1.20	00:50:56:B1:00:15	U				
192.168.1.30	00:50:56:B1:00:11	U				
192.168.1.40	00:50:56:B1:00:1C	U		22		
192.168.1.100	00:50:56:B1:00:13	U		22		
Windows						
192.168.1.50	00:50:56:B1:00:0B	U	Microsoft Windows 2000 SP0/SP1/SP2 or Windows XP SP0/SP1	135, 139, 445, 1025, 5000	123, 137, 138, 445, 500, 1026, 1027, 1900	Y
192.168.1.60	00:50:56:B1:00:0D	U			137, 138	
192.168.1.70	00:50:56:B1:00:0E	U	Windows 2000 SP4, XP SP1+		137, 138	
192.168.1.79	00:50:56:B1:00:03	U		21, 23, 25, 80, 143	137, 138	
192.168.1.80	00:50:56:B1:00:18	U			137	
192.168.1.110	00:50:56:B1:00:16	U		80	137, 138	

	Timestamp	Numeric_Time	Time to deploy honeypot files	Deployed Machines	Deployed Services
START	2012-6-19 14:51:23	41079.61902			
HP 1	2012-6-19 15:8:13	41079.63071	0:16:50	5	7
HP 2	2012-6-19 15:9:43	41079.63175	0:18:20	6	21
STOP	2012-6-20 4:27:31	41080.18578			
	TOTAL (hh:mm:ss)	13:36:08			

Figure 25, Testing - Active scanning (Medium-High-sS noise level)

Nmap: -sT (TCP connect scan)

A TCP connect scan was utilized by nmap for this test. Similar results were observed even though a different type of nmap scan was used, though it appears that a few more ports were identified with the SYN scan. P0f was not able to recognize any of the operating systems during this test which might have been affected by the TCP connect scans. The deployment of configuration files was again a little slower than with

the xprobe2 scanning as seen in Figure 26. A third set of configuration files were deployed during this test due to the number of devices and the open ports that were initially identified. Configuration files with 10 devices and 20 open ports were produced during the final deployment.

Medium-High-sT

IP address	MAC address	State	OS	TCP ports	UDP ports	ICMP reply
Linux						
192.168.1.10	00:50:56:B1:00:17	U				
192.168.1.20	00:50:56:B1:00:15	U				
192.168.1.30	00:50:56:B1:00:11	U				
192.168.1.40	00:50:56:B1:00:1C	U				
192.168.1.100	00:50:56:B1:00:13	U		80		
Windows						
192.168.1.50	00:50:56:B1:00:0B	U	Microsoft Windows 2000 SP0/SP1/SP2 or Windows XP SP0/SP1	135, 139, 445, 1025, 5000	123, 137, 138, 445, 500, 1026, 1027, 1900	Y
192.168.1.60	00:50:56:B1:00:0D	U			137, 138	
192.168.1.70	00:50:56:B1:00:0E	U			137, 138	
192.168.1.79	00:50:56:B1:00:03	U		21, 23, 80, 143	137, 138	
192.168.1.80	00:50:56:B1:00:18	U			137	
192.168.1.110	00:50:56:B1:00:16	U		80	137, 138	

	Timestamp	Numeric_Time	Time to deploy honeypot files	Deployed Machines	Deployed Services
START	2012-6-20 4:59:22	41080.20789			
HP 1	2012-6-20 5:15:48	41080.21931	0:16:26	5	1
HP 2	2012-6-20 5:17:18	41080.22035	0:17:56	6	15
HP 3	2012-6-20 5:30:54	41080.22979	0:31:32	10	20
STOP	2012-6-20 13:22:16	41080.55713			
	TOTAL (hh:mm:ss)	8:22:54			

Figure 26, Testing - Active scanning (Medium-High-sT noise level)

Noise Level – High

The high noise level utilizes both nmap and xprobe2 to actively scan the network. The combination of the two scanners produced a more complete picture of the network

when compared with any other previous test. Xprobe2 was able to identify all the Linux operating systems, but p0f was able to ascertain a better match for one of the Linux systems. Nmap identified one of the Windows operating systems and was able to determine a majority of the open ports on the devices. As seen in Figure 27, the time to initially deploy the configurations files was again affected be the speed at which nmap scans each computer.

High

IP address	MAC address	State	OS	TCP ports	UDP ports	ICMP reply
Linux						
192.168.1.10	00:50:56:B1:00:17	U	Linux Kernel 2.4.30		62402	
192.168.1.20	00:50:56:B1:00:15	U	Linux 2.6			
192.168.1.30	00:50:56:B1:00:11	U	Linux Kernel 2.4.30			
192.168.1.40	00:50:56:B1:00:1C	U	Linux Kernel 2.4.22	22		
192.168.1.100	00:50:56:B1:00:13	U	Linux Kernel 2.4.30	22, 80		Y
Windows						
192.168.1.50	00:50:56:B1:00:0B	U	Microsoft Windows XP	135, 139, 445, 1025, 5000	123, 137, 138, 445, 500, 1026, 1027, 1900	
192.168.1.60	00:50:56:B1:00:0D	U			137, 138	
192.168.1.70	00:50:56:B1:00:0E	U			137, 138	
192.168.1.79	00:50:56:B1:00:03	U		21, 23, 80, 143	137, 138	
192.168.1.80	00:50:56:B1:00:18	U			137	
192.168.1.110	00:50:56:B1:00:16	U		80	137, 138	

	Timestamp	Numeric_Time	Time to deploy honeypot files	Deployed Machines	Deployed Services
START	2012-6-20 13:43:1	41080.57154			
HP 1	2012-6-20 14:3:26	41080.58572	0:20:25	5	4
HP 2	2012-6-20 14:8:58	41080.58956	0:25:57	7	18
HP 3	2012-6-20 14:20:2	41080.59725	0:37:01	10	23
STOP	2012-6-21 3:20:21	41081.13913			
	TOTAL (hh:mm:ss)	13:37:20			

Figure 27, Testing - Active scanning (High noise level)

5.2 Analysis

The results demonstrate that both passive and active scanning can be used simultaneously to gather an accurate picture of the network environment. Extensive information can be gathered to create a honeynet which is representative of the production environment. P0f was able to recognize the Windows operating systems although the ability was degraded when active scanning was included. Xprobe2 had great success at identifying the Linux operating systems. Nmap and tcpdump were able to observe a majority of the ports which were open and communicating. The combination of all the scanners allowed for a more complete picture to be obtained.

More tests may be completed to determine the optimal percent change needed to re-deploy a new honeynet while taking into consideration that stability is required to gather information about intruders. A larger network would require a smaller percent change to deploy additional honeypots as new machines are added to the network. Additional tweaking to the system could be made by adjusting the weight given to devices being identified compared to open ports being found. This would deploy more honeypots as devices are recognized but wait additional time when open ports are recorded.

Being able to deploy this research effort on a real production network with all the user generated traffic would aid in determining which noise level is necessary to map a network. With more traffic potentially generated from each device the aggressiveness of the active scanners could be scaled back to limit the traffic introduced by the system. Nmap has so many features and types of scans available that testing could be done to

determine the best configuration that works with the other system modules. As updates to the scanning programs are released, a fine-tuning period may be required to maximize the system if fine granularity in the honeynet is needed. Due to the nature of the system, organizations are hesitant to allow the installation of a device by a researcher which captures packets on their corporate network. However, when the system is deployed by the organization then tests can be conducted to determine the optimal noise level for their environment.

Chapter 6: Summary

6.1 Conclusions

Honeypot technology is continuing to expand and be updated as different areas of technology are explored for vulnerabilities by attackers. Honeypot research has expanded to client honeypots to gather malware from malicious websites [Seifert, Komisarczuk & Welch, 2009], monitoring for attackers in supervisory control and data acquisition (SCADA) systems which control our country's critical infrastructure [Krutz, 2006], anti-phishing frameworks to combat the exponential growth of phishing campaigns [Li & Schmitz, 2009], and collecting malware for analysis and reverse engineering [Dionaea, 2012].

Common misconceptions and not recognizing that a problem exists has delayed the spread of such a powerful technology in the corporate environment. *Richard Bejtlich, chief security officer at Mandiant, a computer-security company, said that in cases handled by his firm where intrusions were traced back to Chinese hackers, 94% of the targeted companies didn't realize they had been breached until someone else told them. The median number of days between the start of an intrusion and its detection was 416, or more than a year* [Barrett, 2012]. Organizations are not realizing that a problem exists until a breach has occurred and by the time they discover the compromise the organization's data has already been exfiltrated.

Honeypots can be deployed in a wide variety of shapes and sizes depending on the information that the individual is trying to collect. It would not be prudent for an

organization to deploy an unpatched system onto their network due to the legal liabilities and the unnecessary risk. But deploying honeypots that are relevant to the company would do no more harm than deploying a new production system. The benefit to an organization to collect valuable intelligence about how an attacker can compromise a system similar to their production environment is indispensable. Organizations can also shield themselves from more unnecessary risk by only deploying honeypots inside their network environments.

With the increasing volume of data that passes through an organization's network, it is becoming more burdensome to adequately monitor an organization's IDS logs and alerts. It also requires skilled individuals to create IDS signatures that maintain the balance between not alerting and alerting too much. And unfortunately, administrators are being asked to do more with fewer people, less time and resources. So normally it would be ridiculous to pile on more responsibly and duties to an already overworked administrator. But if the short-term burden of learning a new system results in securing the network with an increased efficiency and effectiveness, then sometimes it is a necessary evil. Yet, because the honeypot is a resource with no production value, "it's a great alarm system -- there are no false positives with honeypots. If a packet touches it, something is suspect" said Ralph Logan, principal with the Logan Group [Higgins, 2006]. While a honeypot might generate the same information that an IDS generates, it narrows down the data requiring analysis by eliminating all the additional noise that is collected. To help decrease the steep learning curve associated with using honeypot technology on the job, it is suggested that such skills should be taught in collaboration with other

information security material at the higher education level [Ahmad, Ali & Mustafa, 2011].

Knowing that someone is attempting to break into an organization's network isn't necessarily alarming but being alerted that someone is in the network might be valuable information. By properly placing a honeypot in vital areas of an organization's network, the alerts generated indicate that a potential intrusion has occurred. Many organizations are so consumed with watching their outer defenses that they are not watching for lateral movement inside their network by insiders or attackers that have already infiltrated their defenses. *FBI executive assistant director Shawn Henry added that companies need to do more than just react to intrusions. "In many cases, the skills of the adversaries are so substantial that they just leap right over the fence, and you don't ever hear an alarm go off," he said. Companies "need to be hunting inside the perimeter of their network," he added* [Barrett, 2012]. By deploying and monitoring a honeynet inside the organization's network, sustained access and reconnaissance by an attacker could be mitigated and studied.

This project creates a methodology for an organization to gather tremendous information about its network either passively or with the assistance of active scanners. Capturing data about the devices that are utilizing the network infrastructure can alert the administrators to rogue devices, unpatched or vulnerable systems, devices which are attempting to probe sensitive systems, and create a continuous network topology of the scanned network. By evaluating the benefits, organizations would realize that honeypots

would fulfill the compliance requirements in their risk and security framework that they already have in place [Nunes & Correia, 2010].

6.2 Future Work

There are a few areas of this system which could be expanded or explored to increase the ability to actively engage attackers and gather "valuable" intelligence. As the scanning sub-programs are updated, they need to be incorporated to increase the proficiency to gather data about the devices on the network. Deploying the low and high interaction honeypots from the *honeypot_scanner* module would decrease the need for user intervention. To establish the capabilities of this system at all noise levels, tests must be conducted in a live network which will allow fine tuning of the active scanning modules and demonstrate the ability to handle large volumes of network traffic.

Many honeypot technologies have become available over the past 15 years and even more security tools have been created since the 1970s. Many of these tools have not been kept up to date so their effectiveness is diminished as computer and network technology continues to race forward. Yet there are a handful of tools that are continuing to be updated and expanded as technology changes. Several of the sub-programs used by this system have been updated within the past six months. Nmap v6, p0f v3 and tcpdump v4.2.1 have recently become available which will offer an improved method of gathering information about the network devices. The enhancements made to these modules will allow more information to be gathered about devices with increased efficiency. More

information will be gathered about the devices which will allow a more complete topology to be obtained by the administrator.

Deploying the LIH high interaction honeypots with Honeyd through the management program, *honeypot_scanner*, requires very little customization. The more difficult task is deploying the high interaction honeypots with speed and efficiency. Work is currently being done to deploy the high interaction honeypot utilizing the VMware Virtual Infrastructure eXtension (VIX) API. The primary OS is read from the XML configuration file then compared against the *vmware_template* database table to see if a match is available. The information pertaining to the match is stored in the *lih_hih_link* table which associates the low and high interaction honeypot. The high interaction honeypot is deployed and a script is run on the VM to gather the IP and MAC addresses which is then stored in the *lih_hih_link* table. Work needs to be done to start services and open ports on the VM so they reflect the production devices accurately. Integrating a program such as HoneyMole, which bridges the low interaction honeypot to the high interaction honeypot, would allow for more actionable intelligence to be collected as the attacker probes the low interaction honeypot beyond its capabilities and is then redirected to the high interaction honeypot. As discussed in the hybrid honeypots / honeyfarms section in Chapter 2, systems are currently being deployed which redirect low interaction honeypot attacks to a high interaction honeypot. Yet these systems do not dynamically adjust to a production environment and the high interaction honeypots are deployed on a completely separate system from the low interaction honeypots. A fully integrated system would give the administrator the ability to deploy the honeypots in any network

environment with minimal network configuration, e.g. giving the honeypot scanning system access to a spanning port to gather information passively.

To determine the capabilities of the system to process large amounts of traffic, tests need to be done on varying size networks in different configurations. Creating closed loop networks utilizing VMware ESXi servers has allowed limited testing due to the complications of setting up dynamic networks and reproducing human activity. Testing the system on a live network would assess the responsiveness of the passive scanning modules and the ability to accurately map an organization's infrastructure. Fine tuning the active scanning modules to gather the greatest information without becoming detrimental to the bandwidth of the network could also be tested in a full scale environment.

This work represents an important step forward in honeynet technologies, but much work remains to be done to mitigate the threats facing our digital assets.

References

Acohido, B. (2011, July 4). New cyberattacks target small businesses. *USA TODAY*, Retrieved from http://www.usatoday.com/tech/news/2011-07-04-small-business-cyber-attackss_n.htm

Ahmad, A., Ali, M., & Mustafa, J. (2011, October). Benefits of Honeypots in Education Sector. *International Journal of Computer Science and Network 24 Security*, 11 (10).

Amun: Python Honeypot [computer software]. (2012). Available from http://amunhoney.sourceforge.net

Anderson, J. (1972a). Computer Security Technology Planning study, Volume I. *ESD-TR-73-51.* Hanscom Field, Bedford MA: ESD/AFSC.

Anderson, J. (1972b). Computer Security Technology Planning Study, Volume II. *ESD-TR-73-51.* Hanscom Field, Bedford MA: ESD/AFSC.

Anderson, J. (1980). Computer Security Threat Monitoring and Surveillance. *Technical Report*, James P. Anderson Corporation, Fort Washington, Pennsylvania.

Antonatos, S., Anagnostakis, K. & Markatos, E. (2007). Honey@home: A New Approach to Large-Scale Threat Monitor. *Proceedings of the 5th ACM Workshop on Recurring Malcode (WORM 2007,* Alexandria, VA.

Argus [computer software]. (2012). Available from http://www.qosient.com/argus

Asrigo, K., Litty, L., & Lie, D. (2006). Using VMM-based sensors to monitor honeypots. *Proceedings of the 2nd international conference on Virtual execution environments (VEE).* ACM, New York, NY, 13-23.

Azadegan, S. and McKenna, V. (2005). Use of Honeynets in Computer Security Education. *Proceedings of the Fourth Annual ACIS International Conference on Computer and Information Science* (ICIS). IEEE Computer Society, Washington, DC, 320-325.

Bailey, M., Cooke, E., Watson, D., Jahanian, F. & Provos, N. (2004). A hybrid honeypot architecture for scalable network monitoring. *CSE-TR-499-04.* Retrieved from http://www.eecs.umich.edu/techreports/cse/2004/CSE-TR-499-04.pdf

Bakos, G. Tiny Honeypot [computer software]. (2012). Available from http://web.archive.org/web/20090606073121/http://www.alpinista.org/files/thp

Balas, E. and Viecco, C. (2005). Towards a Third Generation Data Capture Architecture for Honeypots. *Proceedings from the Sixth Annual IEEE.* Systems, Man and Cybernetics (SMC) Information Assurance Workshop, 21-28.

Balasubramaniyan, J., Garcia-Fernandez, J., Isacoff, D., Spafford, E. & Zamboni, D. (1998). An Architecture for Intrusion Detection using Autonomous Agents. *14th Annual Computer Security Applications Conference*, 13–24.

Barfar, A., and Mohammadi, S. (2007, June). Honeypots: Intrusion deception. *Information Systems Security Association Journal*, 28-31.

Barrett, D. (2012, March 28). U.S. Outgunned in Hacker War. *The Wall Street Journal*, Retrieved from http://online.wsj.com/article/SB10001424052702304177104577307773326180032.html

Berthier, R. (2009). Advanced Honeypot Architecture for Network Threats Quantification. (Doctoral Dissertation). Retrieved from University of Maryland at College Park, College Park, MD. (AAI3359256).

Bruneau, G. (2003, October 13). The History and Evolution of Intrusion Detection. Retrieved from http://www.sans.org/reading_room/whitepapers/detection/344.php

Capalik, A. (2007). Driving Intrusion Intelligence into the Real-Time Realm. *NeuralIQ, Inc.*

Carbone, M. & de Geus, P. (2004, June). A Mechanism for Automatic Digital Evidence Collection on High-Interaction Honeypots. *Proceedings from the 5th IEEE Information Assurance Workshop.*

Chamales, G. (2004, March). The Honeywall CD-ROM. *IEEE Security and Privacy* 2, 2, 77-79.

Chaves, C., Franco, L. & Montes, A. (2005, June). Honeynet Maintenance Procedures and Tools. *Proceedings of the 2005 IEEE Workshop on Information Assurance.* United States Military Academy, West Point, NY.

Cheswick, W. (1990, January 20). An Evening with Berferd. *Proceedings of USENIX.*

Chin, W., Markatos, E., Antonatos, S. & Ioannidis, S. (2009, October). HoneyLab: Large-Scale Honeypot Deployment and Resource Sharing. *Third International Conference on Network and System Security (NSS),* 381-388.

Clayton, M. (2012, May 5). Alert: Major cyber attack aimed at natural gas pipeline companies. *The Christian Science Monitor.* Retrieved from http://www.csmonitor.com/USA/2012/0505/Alert-Major-cyber-attack-aimed-at-natural-gas-pipeline-companies

Cohen, F. Deception Toolkit [computer software]. (2012). Available from http://www.all.net/dtk/index.html

Corey, J. (2003, September). Local Honeypot Identification. Retrieved from http://www.ouah.org/ p62-0x07.txt

Corey, J. (2004, January). Advanced HoneyPot Identification. Retrieved from http://www.ouah.org/p63-0x09.txt

Dacier, M., Pouget, F. & Debar, H. (2004). Honeypots: A Practical Means to Validate Malicious Fault Assumptions. *Proceedings of the 10th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC),* 383-388.

Dacier, M., Pouget, F. & Pham V. (2012). LEURRE.COM Honeypot Project. Retrieved from http://www.leurrecom.org

Danford, R. (2006). 2nd Generation Honeyclients. *SANS internet Storm Center.* Retrieved from http://handlers.dshield.org/rdanford/pub/Honeyclients_Danford_SANSfire06.pdf

Defibaugh-Chavez, P., Veeraghattam, R., Kannappa, M., Mukkamala, S. & Sung, A. (2006, June). Network Based Detection of Virtual Environments and Low Interaction Honeypots. *Information Assurance Workshop, 2006 IEEE*, 283-289.

Denning, D., & Neumann, P. (1985). Requirements and Model for IDES -- a Real-Time Intrusion Detection System. *Technical Report*, Computer Science Lab, SRI International.

Denning, D. (1987). An Intrusion Detection Model. *IEEE Transactions on Software Engineering*, 13 (2), 222-232.

Dike, J. (2006). *User Mode Linux(R) (Bruce Perens Open Source)*. Prentice Hall PTR, Upper Saddle River, NJ.

Dionaea [computer software]. (2012). Available from http://dionaea.carnivore.it

Dornseif, M., Holz, T. & Klein, C. (2004, June). NoSEBrEaK - Attacking Honeynets. *Proceedings of the 5th Annual IEEE Information Assurance Workshop*. United States Military Academy, West Point, NY.

Dornseif, M., Freiling, F., Gedicke, N. & Holz, T. (2006, June). Design and Implementation of the Honey-DVD. *Proceedings of the 7th Annual IEEE Information Assurance Workshop*. United States Military Academy, West Point, NY.

Fink, G., O'Donoghue, K., Chappell, B., & Turner, T. (2002, April). A Metrics-Based Approach to Intrusion Detection System Evaluation for Distributed Real-Time Systems. *Proceedings of the 16th international Parallel and Distributed Processing Symposium.* IEEE Computer Society, Washington, DC, 17.

Freedom, P. (2011, October 25). Cyber Attacks on Small Businesses Increase. *Facebook.com*. Retrieved from http://www.facebook.com/note.php?note_id=301960729834086

Freri, M. (2011, September 28). Spear-phishing aiming at the big fish in corporate, Trustsphere says. *ITWire*. Retrieved from http://www.itwire.com/business-it-news/security/50056-spear-phishing-aiming-at-the-big-fish-in-corporate-trustsphere-says

Fu, X., Yu, W., Cheng, D., Tan, X., Streff, K. & Graham, S. (2006, September/October). On Recognizing Virtual Honeypots and Countermeasures. *2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing,* 211-218.

GHH: "Google Hack" Honeypot [computer software]. (2012). Available from http://ghh.sourceforge.net

Glastopf Project [computer software]. (2012). Available from http://glastopf.org

Grimes, Roger. (2004). *Honeypots for Windows*. APress.

Grizzard, J., Simpson, C., Krasser, S., Owen, H. & Riley, G. (2005, June). Flow Based Observations from NETI@home and Honeynet Data. *Proceedings from the Sixth Annual IEEE*. Systems, Man and Cybernetics (SMC) Information Assurance Workshop, 244-251.

Hauben, R. (2002). Commodifying Usenet and the Usenet Archive or Continuing the Online Cooperative Usenet Culture? *Science Studies,* 15 (1), 61-68.

Hecker, C., Nance, K. & Hay, B. (2006, June). Dynamic Honeypot Construction. *Proceedings of the 10th Colloquium for Information Systems Security Education.* University of Maryland, University College, Adelphi, MD.

Hecker, C., Hay, B. (2010, January). Securing E-Government Assets through Automating Deployment of Honeynets for IDS Support. *43rd Hawaii International Conference on System Sciences (HICSS)*, 1-10.

Hieb, J. & Graham, J. (2004, December). Anomaly-Based Intrusion Detection for Network Monitoring Using a Dynamic Honeypot. *Technical Report TR-ISRL-04-03.* Intelligent Systems Research Laboratory.

Higgins, K. (2006, August 23). Enterprises Still Not Sweet on Honeypots. *Darkreading,* Retrieved from http://www.darkreading.com/security/application-security/208804004/enterprises-still-not-sweet-on-honeypots.html

Hoepers, C., Steding-Jessen, K., Cordeiro, L. & Chaves, M. (2005, June). A National Early Warning Capability Based on a Network of Distributed Honeypots. *Proceedings of the 17th Annual FIRST Conference on Computer Security Incident Handling*, Singapore.

Holz, T. & Raynal, F. (2005). Detecting honeypots and other suspicious environments. *Proceedings of the 6th IEEE Information Assurance Workshop.* United States Military Academy, West Point, NY.

Honeynet Project, The. (2003, November 17). Know Your Tools: Sebek - A kernel based data capture tool. *Honeypot Project* Retrieved from http://old.honeynet.org/papers/sebek.pdf

Honeytrap [computer software]. (2012). Available from http://honeytrap.carnivore.it

Howard, J. & Longstaff, T. (1998). A Common Language for Computer Security Incidents. *Sandia Report: SAND98-8667.* Sandia National Laboratories.

Hudak, S. (2008). Automatic Honeypot Generation and Network Deception," 2008. Retrieved from CiteSeerX - Scientific Literature Digital Library and Search.

Jackson, T., Levine, J., Grizzard, J. & Owen, H. (2004, March). An Investigation of a Compromised Host on a Honeynet Being Used to Increase the Security of a Large Enterprise Network. *Proceedings of the 5th IEEE Information Assurance Workshop*, 9-14.

Jiang, X. & Xu, D. (2004). Collapsar: A vm-based architecture for network attack detention center. *Proceedings of the USENIX Security Symposium*, 15–28.

Jiang, X., Xu, D. & Wang, Y. (2006, September). Collapsar: a VM-based honeyfarm and reverse honeyfarm architecture for network attack capture and detention. *Journal of Parallel Distributed Computing,* 66 (9), 1165-1180.

John, J., Yu, F., Xie, Y., Krishnamurthy, A. & Abadi, M. (2011). Heat-seeking honeypots: design and experience. *Proceedings of the 20th International World Wide Web Conference (WWW).*

Kehoe, B. (1992, January). Zen and the Art of the internet: A Beginner's Guide to the internet. Retrieved from http://www.cs.indiana.edu/docproject/zen/zen-1.0_toc.html

KFSensor: Advanced Windows Honeypot System [computer software]. (2012). Available from http://www.keyfocus.net/kfsensor

Kemmerer, R. and Vigna, G. (2002). Intrusion Detection: A brief History and Overview. *Computer*, 35, 27-30.

Krawetz, N. (2004). Anti-Honeypot Technology. IEEE Security and Privacy, 2 (1), 76-79.

Krutz, R. (2006). *Securing SCADA Systems*. Wiley Publishing, Inc.

Kuwatly, I., Sraj, M., Masri, Z. & Artail, H. (2004, July). A Dynamic Honeypot Design for Intrusion Detection. *The IEEE/ACS International Conference on Pervasive Services (ICPS),* 95- 104.

Kyaw, K. (2008). Hybrid Honeypot System for Network Security. *World Academy of Science, Engineering and Technology*, 48.

Larose, C. (2012, May 3). Symantec: Malicious Cyber Attacks Increased by 81 Percent in 2011 and Data Breaches Up, *Mintz Levin - Privacy & Security*. Retrieved from http://www.jdsupra.com/post/documentViewer.aspx?fid=58c95d85-b966-4664-a4a1-15ed5f0d1580

Leiner, B., Cerf, V., Clark, D., Kahn, R., Kleinrock, L., Lynch, D., Postel, J., Roberts, L. & Wolff, S. (1997, February). The past and future history of the internet. *Communications of the ACM,* 40 (2), 102-108.

Leiner, B., Cerf, V., Clark, D., Kahn, R., Kleinrock, L., Lynch, D., Postel, J., Roberts, L. & Wolff, S. (2009, October). A brief history of the internet. *SIGCOMM Computer Communication Review,* 39 (5), 22-31.

Leita, C., Mermoud, K. & Dacier, M. (2005). ScriptGen: an automated script generation tool for honeyd. *Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC).* IEEE Computer Society, Washington, DC, 203-214.

Levine, J., LaBella, R., Owen, H., Contis, D. & Culver, B. (2003, June). The use of Honeynets to Detect Exploited Systems Across Large Enterprise Networks. *Proceedings of the IEEE Workshop on Information Assurance.* United States Military Academy, West Point, NY, 92 – 99.

Levine, J., Grizzard, J. & Owen, H. (2004b, November/December). Using Honeynets to protect large enterprise networks. *IEEE Security & Privacy*, 2 (6), 73-75.

Li, S. & Schmitz, R. (2009, October). A Novel Anti-Phishing Framework Based on Honeypots. *APWG eCrime Researchers Summit*, Tacoma, WA.

Liston, T. LaBrea: "Sticky" Honeypot and IDS [computer software]. (2012). Available from http://labrea.sourceforge.net

Lunt, T. & Jagannathan, R. (1988). A Prototype Real-Time Intrusion-Detection Expert System. *IEEE Symposium on Security and Privacy*, 59.

Mokube, I. & Adams, M. (2007). Honeypots: concepts, approaches, and challenges. *Proceedings of the 45th annual southeast regional conference (ACM-SE 45).* ACM, New York, NY, 321-326.

Mukkamala, S., Yendrapalli, K., Basnet, R., Shankarapani, M.K. & Sung, A. (2007, June). Detection of Virtual Environments and Low Interaction Honeypots. *Information Assurance and Security Workshop (IAW).* IEEE SMC, 92-98.

MySQL [computer software]. (2012). Available from http://www.mysql.com

Nepenthes [computer software]. (2012). Available from http://nepenthes.carnivore.it

NFR Security Inc. BackOfficer Friendly [computer software]. (2012). Available from http://web.archive.org/web/20020405044644/http://www.nfr.com/products/bof

NoAH (A European Network of Affined Honeypots), Project funded by the European Community, Retrieved from *http://www.fp6-noah.org*

Nunes, S. & Correia, M. (2010). From Risk Awareness to Security Controls: Benefits of Honeypots to Companies. *Proceedings of the 2nd OWASP Ibero-American Web Applications Security Conference (IBWAS).*

Oberheide, J. & Karir, M. (2006, January). Honeyd Detection via Packet Fragmentation. *Merit Technical Report.*

Ofir, A., Yarochkin, F. & Kydyraliev, M. (2003, July). The Present and Future of Xprobe2: The Next Generation of Active Operating System Fingerprinting. Retrieved from http://www.syssecurity.com/archive/papers/Present_and_Future_Xprobe2-v1.0.pdf

Oudot, L., Ropert, F. & Riden, J. PHP.Hop - PHP Honeypot Project [computer software]. (2012). Available from http://web.archive.org/web/20080509124834/http://www.rstack.org/phphop/

Portokalidis, G., Slowinska, A. & Bos, H. (2006, April). Argos: an Emulator for Fingerprinting Zero-Day Attacks. *Proceedings of ACM SIGOPS*, Eurosys.

Portuguese Honeynet Project. HoneyMole [computer software]. (2012). Available from http://www.honeynet.org.pt/index.php/HoneyMole

Pouget, F., Dacier, M. & Pham, V. (2005, March). Leurre.com: on the advantages of deploying a large scale distributed honeypot platform. *ECCE'05, E-Crime and Computer Conference*, Monaco.

Pouget, F. & Holz, T. (2005, July). A Pointillist Approach for Comparing Honeypots. *Proceedings of the Detection of Intrusions and Malware, and Vulnerability Assessment, Second International Conference (DIMVA)*, Vienna, Austria, Lecture Notes in Computer Science, 3548, 51–68.

Prasad, B., Abraham, A., Abhinav, A., Gurlahosur, S. & Srinivasa, Y. (2011, March). Design And Efficient Deployment Of Honeypot And Dynamic Rule Based Live Network Intrusion Collaborative System. *International Journal of Network Security & Its Applications (IJNSA)*, 3 (2).

Provos, N. (2004). A virtual honeypot framework. *Proceedings of the 13th conference on USENIX Security Symposium (SSYM),* 13. USENIX Association, Berkeley, CA.

Provos, N. & Holz, T. (2007). *Virtual Honeypots: From Botnet Tracking to Intrusion Detection* (First Edition). Addison-Wesley Professional.

QEMU [computer software]. (2012). Available from http://wiki.qemu.org

Seifert, C., Komisarczuk, P. & Welch, I. (2009, June). True Positive Cost Curve: A Cost-Based Evaluation Method for High-Interaction Client Honeypots. *Third*

International Conference on Emerging Security Information, Systems and Technologies (SECURWARE), 63-69.

Send-Safe Honeypot Hunter [computer software]. (2012). Available from http://www.sendsafe.com/honeypot-hunter.html

Sherif, J. & Dearmond, T. (2002). Intrusion Detection: Systems and Models. *Eleventh IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, 115.

Shiue, L. & Kao, S. (2008, May). Countermeasure for detection of honeypot deployment. *International Conference on Computer and Communication Engineering (ICCCE),* 595-599.

Snort [computer software]. (2012). Available from http://www.snort.org

Song, C., Hay, B., & Zhuge, J. (2010, October). *Know Your Tools: Qebek – Conceal the Monitoring.* Honeypot Project. Retrieved from http://honeynet.org/files/KYT-Qebek-final_v1.docx

Specter: Intrusion Detection System [computer software]. (2012). Available from http://www.specter.com

Spitzner, L. (2000, June). Learning the Tools and the Tactics of the Enemy with Honeynets. *Proceedings of the 12th Annual Computer Security Incident Handling Conference*, Chicago, Illinois.

Spitzner, L. (2002). *Honeypots: Tracking Hackers*. Addison-Wesley, Boston.

Spitzner, L. (2003, September). Dynamic Honeypots. Retrieved from http://www.symantec.com/connect/articles/dynamic-honeypots

Spitzner, L. (2004, January). Problems and Challenges with Honeypots. Retrieved from http://www.symantec.com/connect/articles/problems-and-challenges-honeypots

Srivastava, K. (2012, May 16). How Cyber-Hacks Are Hurting Small Businesses. *Mobiledia*. Retrieved from http://www.mobiledia.com/news/142976.html

Stoll, C. (1990). The Cuckoo's Egg: Tracking a Spy through the Maze of Computer Espionage. *Pocket Books*, New York.

Talabis, R. (2006). Honeypots 101: A Brief History of Honeypots. *A Student IT Security Awareness Initiative by Philippine Honeynet Project*. Retrieved from http://www.philippinehoneynet.org/docs/Honeypot101_history.pdf

Tammen, G. (2012, May 10). Now you see me, now you don't: Cybersecurity experts begin investigation on self-adapting computer network that defends itself against hackers. *Kansas State University, News and Editorial Services.* Retrieved from http://www.k-state.edu/media/newsreleases/may12/movingtarget51012.html

Tcpdump [computer software]. (2012). Available from http://www.tcpdump.org

Turk, R. (2005, October). Cyber Incidents Involving Control Systems. *Contract*, Idaho National Engineering and Environmental Laboratory. Retrieved from http://www.inl.gov/technicalpublications/Documents/3480144.pdf

Tzu, Sun. (1910). The Art of War, translated by Lionel Giles in 1910.

Viecco, C. (2007, June). Improving Honeynet Data Analysis. *Information Assurance and Security Workshop (IAW).* IEEE SMC, 99-106.

Vrable, M., Ma, J., Chen, J., Moore, D., Vandekieft, E., Snoeren, A., Voelker, G. & Savage, S. (2005, October). Scalability, fidelity, and containment in the potemkin virtual honeyfarm. *SIGOPS Operating System Review,* 39 (5), 148-162.

VMware [computer software]. (2012). Available from http://www.vmware.com

Wang, H. & Chen, Q. (2011, August). Modeling and Deploying of Dynamic Honeynet. *International Journal of Advancements in Computing Technology*, 3 (7).

Wang, W., Wang, C. & Shi-fu, C. (2006). Dynamic hierarchical distributed intrusion detection system based on multiagent system. *Proceedings of the 2006 IEEE/WIC/ACM international conference on web intelligence and intelligent agent technology*, 89-93.

Wicherski, G. (2006, April). Medium Interaction Honeypots. Retrieved from http://www.pixel-house.net/midinthp.pdf

Xen [computer software]. (2012). Available from http://www.xen.org

Yarochkin, F. Nmap - Network Mapper [computer software]. (2012). Available from http://nmap.org

Zalewski, M. Passive OS Fingerprinting Tool [computer software]. (2012). Available from http://lcamtuf.coredump.cx/p0f3

Appendices

Appendix A: Acronyms

LIH(s)	Low Interaction Honeypot(s) or Honeynet(s)
HIH(s)	High Interaction Honeypot(s) or Honeynet(s)
ID(S)	Intrusion Detection (System)
VM(s)	Virtual Machine(s)
TTL	Time to live

Appendix B: Glossary

Dynamic - Marked by continuous usually productive activity or change [1]; not necessarily with respect to time but adapting to the environment.

Honeypot - A security resource, whose value is in being probed, attacked or compromised. [2]

Noise (level) - Network activity generated by the research effort or any other device.

[1] "dynamic." *Merriam-Webster's Medical Dictionary*. Merriam-Webster, Inc. 22 Apr. 2007. <Dictionary.com http://dictionary.reference.com/browse/dynamic>.

[2] Spitzner, Lance. "Honeypots: Definitions and Value of Honeypots." 23 Apr. 2007. <http://www.spitzner.net/honeypots.html>.

Appendix C: Module Commands and Flags

Table 3, Module commands and flags

Module Command and Flags	
Modules	**Commands and Flags**
honeypot_scanner	perl honeypot_scanner.pl
tcpdump_mysql	perl tcpdump_mysql.pl -ip {ip} -i {interface}
	-ip # IP address(es) from which to gather information #
	-i # Ethernet interface utilized to capture packets #
p0f_mysql	perl p0f_mysql.pl -i {interface}
	-i # Ethernet interface utilized to capture packets #
nmap_mysql	perl nmap_mysql.pl -mip {ip} -mac {mac} -t {time} -o {file} -nmap {exe} -ip {ip}
	-mip # IP address passed from active scanner (scan_queue) #
	-mac # MAC address passed from active scanner (scan_queue) #
	-t # Time between active re-scans of known devices (seconds) #
	-o # Nmap output XML path #
	-nmap # Nmap.exe path #
	-ip # IP address that Nmap will be scanning #
xprobe2_mysql	perl xprobe2_mysql.pl -mip {ip} -mac {mac} -t {time} -o {file} -p {level} -ip {ip}
	-mip # IP address passed from active scanner (scan_queue) #
	-mac # MAC address passed from active scanner (scan_queue) #
	-t # Time between active re-scans of known devices (seconds) #
	-o # Nmap output XML path #
	-p # Increased scanning flag (1 = enabled) #
	-ip # IP address that Nmap will be scanning #

**active_scanner - Not able to be run independently because it needs the passive scanners to identify devices on the network and populate *scan_queue*

Appendix D: Source Code

Honeypot_scanner.pl

```
#!/usr/bin/perl
#honeypot_scanner.pl by Chris Hecker, 2007

use strict;
use threads;
use DBI;
use File::Copy;
use IO::Socket::INET;
use Net::DHCP::Packet;
use Net::DHCP::Constants;


# Config table variables
my $eth_interface, my $my_mac, my $dhcp_server, my $p0f_os, my $nmap_exe, my
$nmap_assoc, my $nmap_prints, my $nmap_xml, my$xprobe2_xml,
my $xprobe2_conf, my $honeypot_xml, my $honeyd_config, my $honeyd_ip_binding,
my $honeyd_ip_range, my $scan_ip_range, my $initial_deployment,
my $percent_change, my $noise, my $active_scan_seconds,

# Thread variables
my $tcp_pid, my $p0f_pid, my $act_pid, my $tcp_run, my $p0f_run, my $act_run,
my $tcp_name = "tcpdump_scan", my $p0f_name = "p0f_scan", my $act_name =
"active_scan",

# Global variables
my $pc_threshold_high, my $pc_threshold_low, my $honeypot_deploy = "N",
my $count = 0, my $honeyd_count, my $honeyd_max, my $honeyd_ip_temp;

system "killall p0f";                # Kill all instances of p0f running before our
program starts
system "killall tcpdump";            # Kill all instances of tcpdump running before our
program starts
system "/etc/init.d/dhcdbd stop";    # Stop DHCP client from running
system "killall dhclient";           # Kill all instances of dhclient running before our
program starts

# Calls connect_to_db() function to connect to honeypot_scanner MySQL database
my $dbh= connect_to_db();

# Calls db_query_config() function to retrieve all the config database variables
```

```
db_query_config($dbh);

# Inserts information returned from DHCP server into the database
# Prepare the insert statement just once, the actual values will replace the ? later
my $insert_dhcp_register = $dbh->prepare( "INSERT INTO dhcp (dhcp_id, ip_addr,
mac_addr, lease_time_seconds, renewal_tstamp_time, date_created, last_tstamp,
last_tstamp_time) VALUES(?, ?, ?, ?, ?, ?, ?, ?);");
my $insert_updates = $dbh->prepare( "INSERT INTO honeypot_updates
(num_machines, num_services, date_updated) VALUES (?, ?, ?);");
my $insert_lih_info = $dbh->prepare( "INSERT INTO lih_hih_link (lih_os_platform,
lih_ip_addr, lih_mac_addr, hih_os_platform, hih_ip_addr, hih_mac_addr, hih_location,
hih_state, date_created, last_tstamp) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?);");

# Creates tcpdump_mysql thread and detaches it
my $tcp_thr = threads->new(\&tcpdump_scan);
$tcp_thr->detach; # Now we officially don't care any more

# Creates p0f_mysql thread and detaches it
my $p0f_thr = threads->new(\&p0f_scan);
$p0f_thr->detach; # Now we officially don't care any more

# Checks the noise level for the config table and creates active_scan thread if necessary
then detaches it
my $act_thr;
if($noise eq 'low' || $noise eq 'medium' || $noise eq 'medium-high' || $noise eq 'high')
{       $act_thr = threads->new(\&active_scan);
        $act_thr->detach; # Now we officially don't care any more
}elsif ($noise eq "passive"){
}else {print "Please choose a correct noise level for this scanner in the config table!\n";
exit;}

# Sleep for 2 seconds to ensure threads are created before next step
sleep 2;

# Obtains the process id for each of the created threads to watch for early termination
$tcp_pid = db_query_threads($tcp_name, $dbh);
open (PSIN, "ps -p $tcp_pid -o comm= |");
$tcp_run=<PSIN>;
$p0f_pid = db_query_threads($p0f_name, $dbh);
open (PSIN, "ps -p $p0f_pid -o comm= |");
$p0f_run=<PSIN>;
$act_pid = db_query_threads($act_name, $dbh);
open (PSIN, "ps -p $act_pid -o comm= |");
$act_run=<PSIN>;
```

```
eval {
        # Install signal handlers
        $SIG{INT} = sub { die "Caught interrupt" };

        while (1){
                # Sleeps for 10 seconds
                sleep 10;

                # Increment count, mentioned below
                $count++;

                # Retrieve all the config database variables again to make sure use has
changed information
                db_query_config($dbh);

                # Checks to see if any honeyd IP addresses need to be renewed (only
necessary if -il honeyd_ip_binding used)
                db_query_renew_dhcp($dbh);

                # Looks to see if tcpdump_mysql is still running and spins off a new
thread if not
                open (PSIN, "ps -p $tcp_pid -o comm= |");
                my $tcp_still_run = <PSIN>;
                if ($tcp_still_run ne $tcp_run){
                        my $tcp_thr = threads->new(\&tcpdump_scan);
                        $tcp_thr->detach; # Now we officially don't care any more
                        sleep 2;
                        $tcp_pid = db_query_threads($tcp_name, $dbh);
                        open (PSIN, "ps -p $tcp_pid -o comm= |");
                        $tcp_run = <PSIN>;
                }

                # Looks to see if p0f_mysql is still running and spins off a new thread if
not
                open (PSIN, "ps -p $p0f_pid -o comm= |");
                my $p0f_still_run = <PSIN>;
                if ($p0f_still_run ne $p0f_run){
                        my $p0f_thr = threads->new(\&p0f_scan);
                        $p0f_thr->detach; # Now we officially don't care any more
                        sleep 2;
                        $p0f_pid = db_query_threads($p0f_name, $dbh);
                        open (PSIN, "ps -p $p0f_pid -o comm= |");
                        $p0f_run = <PSIN>;
```

```
                }

                # Looks to see if active_scan is still running and spins off a new thread if
not
                open (PSIN, "ps -p $act_pid -o comm= |");
                my $act_still_run = <PSIN>;
                if ($act_still_run ne $act_run){
                        if ($noise eq 'low' || $noise eq 'medium' || $noise eq 'medium-high'
|| $noise eq 'high')
                        {       $act_thr = threads->new(\&active_scan);
                                $act_thr->detach; # Now we officially don't care any more
                                sleep 2;
                                $act_pid = db_query_threads($act_name, $dbh);
                                open (PSIN, "ps -p $act_pid -o comm= |");
                                $act_run = <PSIN>;
                        }
                }
                # Since we don't need to check every 10 secondsfor honeypot
redeployment, I created this counter to extend time to 30 seconds
                if ($count >= 3){
                        # Check to see if threshold has been reached to create honeyd
config
                        db_query_honeypot_deployment($dbh);
                        $count = 0;
                }
        }
};

# Close down and clean up if ctl-c or interrupt occurred
if ($@) {
        # Disconnect from Database
        $dbh->disconnect();
        print "Honeypot_Scanner disconnected from Database.\n";

        # Print exit message
        print "\nExiting with: $@\n";

        system "killall p0f";
        system "killall tcpdump";
        system "killall nmap";
        system "killall xprobe2";
        system "kill $p0f_pid";
        system "kill $tcp_pid";
        system "kill $act_pid";
```

```
        sleep 1;
        exit;
}

# Connect to the database
sub connect_to_db
{       # Database Information
        my $db="honeypot_scanner";
        my $userid="root";
        my $passwd="rootpass";
        my $connectionInfo="dbi:mysql:$db";

        # Make Connection to Database
        my $dbh = DBI->connect($connectionInfo,$userid,$passwd, {
    RaiseError => 1,
    AutoCommit => 0
    } ) || die "Database connection not made: $DBI::errstr";
        return( $dbh);
}

# Retrieve information then check integrity of information from config table in database
sub db_query_config
{       my $dbh = shift;

        # Prepare and Execute DB Query
        my $query = "select * from config;";
        my $select_stmt = $dbh->prepare($query);
        $select_stmt->execute();
        $select_stmt->bind_columns(\my $ein, \my $mc, \my $dhcp, \my $p0f, \my $ne,
\my $np, \my $na, \my $nx, \my $xc, \my $xx,
        \my $hx, \my $hc, \my $hib, \my $hir, \my $sir, \my $id, \my $pc, \my $n, \my
$as, \my $dc);
        while ($select_stmt->fetch()) {
                $eth_interface = $ein;
                $my_mac = $mc;
                $dhcp_server = $dhcp;
                $p0f_os = $p0f;
                $nmap_exe = $ne;
                $nmap_prints = $np;
                $nmap_assoc = $na;
                $nmap_xml = $nx;
                $xprobe2_conf = $xc;
                $xprobe2_xml = $xx;
                $honeypot_xml = $hx;
```

```
            $honeyd_config = $hc;
            $honeyd_ip_binding = $hib;
            $honeyd_ip_range = $hir;
            $scan_ip_range = $sir;
            $initial_deployment = $id;
            $percent_change = $pc;
            $noise = $n;
            $active_scan_seconds = $as;
        }
        $select_stmt->finish;

        # Make sure there are no spaces or ; in file path names for security
        if($nmap_exe =~ /[^a-zA-Z0-9_\/.\-]/){print "nmap_exe in the config table
contains a bad character!\n"; exit;}
        elsif($p0f_os =~ /[^a-zA-Z0-9_\/.\-]/){print "p0f os database in the config table
contains a bad character!\n"; exit;}
        elsif($nmap_prints =~ /[^a-zA-Z0-9_\/.\-]/){print "nmap_prints in the config table
contains a bad character!\n"; exit;}
        elsif($nmap_assoc =~ /[^a-zA-Z0-9_\/.\-]/){print "nmap_assoc in the config table
contains a bad character!\n"; exit;}
        elsif($xprobe2_conf =~ /[^a-zA-Z0-9_\/.\-]/){print "xprobe2_conf in the config
table contains a bad character!\n"; exit;}
        elsif($xprobe2_xml =~ /[^a-zA-Z0-9_\/.\-]/){print "xprobe2_xml in the config
table database contains a bad character!\n"; exit;}
        elsif($honeyd_config =~ /[^a-zA-Z0-9_\/.\-]/){print "honeyd_config in the config
table contains a bad character!\n"; exit;}

        # Check to see if the correct honeyd_ip_binding was input into the config table
        if($honeyd_ip_binding =~ m/^(iS|iD|iR|iI)$/i){}else {print "Please choose a
correct honeyd_ip_binding for this scanner in the config table!\n"; exit;}

        # Check to see if the MAC address entered into the config table is in the correct
format
        my $my_mac_temp = $my_mac;
        my $mac_rgx = '^\w{2}\:\w{2}\:\w{2}\:\w{2}\:\w{2}\:\w{2}$';
        my $my_mac = mac_chkmac($my_mac_temp,$mac_rgx);
        if ($my_mac eq ""){print "A correct MAC address must be entered in the config
table!\n"; exit(1);}

        # Check to see if the dhcp server IP entered into the config table is in the correct
format
        my $dhcp_server_temp = $dhcp_server;
        my $ip_rgx = '^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}$';
        my $dhcp_server = ipv4_chkip($dhcp_server_temp,$ip_rgx);
```

```
        if ($dhcp_server == ""){print "Single & correct IP address must be entered for the
DHCP server in the config table!\n"; exit(1);}

        # Check to see if the correct honeyd_ip_range was input into the config table
        honeyd_ip_manipulation();
}

# Query the threads tables for PID information
sub db_query_threads
{       my $thr_name = shift;
        my $dbh = shift;
        my $thr_pid;

        # Prepare and Execute DB Query
        my $query = "select thread_id, thr_pid from threads where thr_name =
'$thr_name';";
        my $select_stmt = $dbh->prepare($query);
        $select_stmt->execute();
        $select_stmt->bind_columns(\my $t_id, \my $t_pid);
        while ($select_stmt->fetch()) {
                $thr_pid = $t_pid;
        }
        $select_stmt->finish;

        return $thr_pid;
}

# Insert information about new honeypot deployment
sub db_insert_updates
{       my $num_machines = shift;
        my $num_services = shift;
        my $dbh = shift;

        # Finalize the insert statement
        $insert_updates->bind_param( 1, $num_machines);  # refers to the first ? in the
query
        $insert_updates->bind_param( 2, $num_services);  # refers to the second ? in the
query
        my $date_updated = date_time();
        $insert_updates->bind_param( 3, $date_updated);    # refers to the third ? in the
query

        $insert_updates->execute();                    # excute the SQL statement
}
```

```
# Check to see if threshold has been reached to create honeyd config
sub db_query_honeypot_deployment
 {      my $dbh= shift;

        my $current_count = 0;
        my $num_machines = 0;
        my $num_services = 0;
        my $scan_time = time() - $active_scan_seconds;

        # Prepare and Execute DB Query
        my $query = "(select ip_addr, mac_addr, primary_os from p0f where
last_tstamp_time > '$scan_time' AND ip_addr NOT IN (select ip_addr from
nmap_machines where primary_os != 'unknown' AND last_tstamp_time > '$scan_time')
AND ip_addr NOT IN (select ip_addr from xprobe2_machines where primary_os !=
'unknown' AND last_tstamp_time > '$scan_time')) union (select ip_addr, mac_addr,
primary_os from xprobe2_machines where primary_os != 'unknown' AND
last_tstamp_time > '$scan_time' AND ip_addr NOT IN (select ip_addr from
nmap_machines where primary_os != 'unknown' AND last_tstamp_time > '$scan_time'))
union (select ip_addr, mac_addr, primary_os from nmap_machines where primary_os !=
'unknown' AND last_tstamp_time > '$scan_time') order by ip_addr;";
        my $select_stmt = $dbh->prepare($query);
        $num_machines = $select_stmt->execute();
        $select_stmt->bind_columns(\my $ip, \my $mac, \my $os);
        # LOOP THROUGH RESULTS
        while($select_stmt->fetch()) {

                # Prepare and Execute DB Query
                my $query2 = "(select ip_addr, mac_addr, port_num, protocol from
tcpdump_ports where syn_ack = 'Y' AND ip_addr = '$ip' AND mac_addr = '$mac' AND
last_tstamp_time > '$scan_time') union (select nmap_machines.ip_addr,
nmap_machines.mac_addr, nmap_ports.port_num, nmap_ports.protocol from
nmap_machines, nmap_ports where nmap_machines.machine_id =
nmap_ports.machine_id AND nmap_machines.ip_addr = '$ip' AND
nmap_machines.mac_addr = '$mac' AND last_tstamp_time > '$scan_time') union (select
xprobe2_machines.ip_addr, xprobe2_machines.mac_addr, xprobe2_ports.port_num,
xprobe2_ports.protocol from xprobe2_machines, xprobe2_ports where
xprobe2_machines.machine_id = xprobe2_ports.machine_id AND
xprobe2_machines.ip_addr = '$ip' AND xprobe2_machines.mac_addr = '$mac' AND
last_tstamp_time > '$scan_time') order by ip_addr, port_num, protocol;";
                my $select_stmt2 = $dbh->prepare($query2);
                $num_services = $select_stmt2->execute() + $num_services;
                $select_stmt2->finish;
        }
```

```
        $select_stmt->finish;

        $current_count = $num_machines + $num_services;

        if ($honeypot_deploy eq "N")
        {
                # Check to see if threshold has been reached to create honeyd config
(initial deployment)
                if ($num_machines >= $initial_deployment)
                {
                        db_insert_updates($num_machines, $num_services, $dbh);
                        honeyd_ip_manipulation();
                        output_honeyd_config_xml($dbh);
                        $honeypot_deploy = "Y";
                        my $temp = ($current_count * ($percent_change / 100));
                        $pc_threshold_low = $current_count - $temp;
                        $pc_threshold_high = $current_count + $temp;
                }
        } else {
                # Check to see if threshold has been reached to re-create honeyd config
(re-deployment)
                if ($current_count >= $pc_threshold_high || $current_count <=
$pc_threshold_low)
                {
                        db_insert_updates($num_machines, $num_services, $dbh);
                        honeyd_ip_manipulation();
                        output_honeyd_config_xml($dbh);
                        my $temp = ($current_count * ($percent_change / 100));
                        $pc_threshold_low = $current_count - $temp;
                        $pc_threshold_high = $current_count + $temp;
                }
        }
}

# Check for a valid MAC address.
sub mac_chkmac($$)
{       my ($mac) = $_[0] =~ m/($_[1])/;

        return undef unless $mac;

        # Check that bytes are in range
        for (split /\:/, $mac ) {
                return undef if $_ !~ m/(0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f)/i;
        }
```

```
        return $mac;
}

# Check for a valid IPv4 address.
sub ipv4_chkip($$)
 {      my ($ip) = $_[0] =~ m/($_[1])/;

        return undef unless $ip;

        # Check that bytes are in range
        for (split /\.|-/, $ip ) {
                return undef if $_ < 0 or $_ > 255;
        }
        return $ip;
}

# Subroutine to manipulate IPV4 address to the correct format for scan_ip_range
sub scan_ip_manipulation
 {      my $ip_rgx2 = '^\d{1,3}\.\d{1,3}$';
        my $ip_rgx3 = '^\d{1,3}\.\d{1,3}\.\d{1,3}$';
        my $ip_rgx4 = '^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}$';
        my $ip_range_rgx = '^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}-\d{1,3}$';
        my $pattern = '^\d{1,3}\.\d{1,3}\.\d{1,3}\.';
        my $pattern2 = '\d{1,3}$';
        my $pattern3 = '^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}-';
        my $pattern4 = '-\d{1,3}$';

        my $ip = ipv4_chkip($scan_ip_range,$ip_rgx2);
        if ($ip eq ""){
                my $ip = ipv4_chkip($scan_ip_range,$ip_rgx3);
                if ($ip eq ""){
                        my $ip = ipv4_chkip($scan_ip_range,$ip_rgx4);
                        if ($ip eq ""){
                                my $ip = ipv4_chkip($scan_ip_range,$ip_range_rgx);
                                if ($ip eq ""){
                                        print "The scan ip address format in the config table
is not correct.\n"; exit;
                                } else {
                                        my $ip_tcp = $scan_ip_range;
                                        $ip_tcp =~ s/$pattern4//s;
                                        $ip_tcp =~ s/$pattern2//s; chop $ip_tcp;
                                        return $ip_tcp;
                                }
                        } else { return my $ip_tcp = $scan_ip_range;}
```

```
                } else { return my $ip_tcp = $scan_ip_range;}
        } else { return my $ip_tcp = $scan_ip_range;}
}

# Subroutine to manipulate IPV4 address to the correct format for honeyd_ip_range
sub honeyd_ip_manipulation
{       my $ip_rgx = '^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}$';
        my $ip_range_rgx = '^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}-\d{1,3}$';
        my $pattern = '^\d{1,3}\.\d{1,3}\.\d{1,3}\.';
        my $pattern2 = '\d{1,3}$';
        my $pattern3 = '^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}-';
        my $pattern4 = '-\d{1,3}$';

        $honeyd_ip_temp = $honeyd_ip_range;

        if ($honeyd_ip_binding eq "iD")
        {
                my $honeyd_ip = ipv4_chkip($honeyd_ip_temp,$ip_rgx);
                if ($honeyd_ip == ""){print "Single IP address must be entered with the -
iD command.\n"; exit(1);}
        }
        elsif ($honeyd_ip_binding eq "iR")
        {
                my $honeyd_ip = ipv4_chkip($honeyd_ip_temp,$ip_range_rgx);
                if ($honeyd_ip == ""){print "IP range must be entered with the -iR
command.\n"; exit(1);}
                $honeyd_count = $honeyd_ip_temp;
                $honeyd_count =~ s/$pattern//s;
                $honeyd_count =~ s/$pattern4//s;
                $honeyd_max = $honeyd_ip_temp;
                $honeyd_max =~ s/$pattern3//s;
                $honeyd_ip_temp =~ s/$pattern4//s;
        }
}

# Start tcpdump_mysql.pl
sub tcpdump_scan
{       my $ip_tcp = scan_ip_manipulation();
        system "perl tcpdump_mysql.pl -ip $ip_tcp -i $eth_interface";
}

# Start p0f_mysql.pl
sub p0f_scan
{       system "perl p0f_mysql.pl -i $eth_interface";
```

```
}

# Start active_scan.pl
sub active_scan
{       system "perl active_scanner.pl -n $noise -nexe $nmap_exe -nxml $nmap_xml -
xxml $xprobe2_xml -t $active_scan_seconds";
}

# Write honeyd config & XML file
sub output_honeyd_config_xml
{       my $dbh= shift;
        my $num = 0, my $family, my $pattern6 = '^\w+',
        my $scan_time = time() - $active_scan_seconds;

        if ($honeyd_ip_binding eq "il") {db_query_release_dhcp($dbh);}

        # Looks for past_honeyd_files directory specified in the config table or creates the
directory
        my $past_honeyd_config = $honeyd_config;
        $past_honeyd_config =~ s/(honeyd.conf)$//s;
        if (-d
"".$past_honeyd_config."/past_honeyd_files"){}else{mkdir($past_honeyd_config."/past
_honeyd_files") || die "Unable to create past_honeyd directory <$!>\n";}
        $past_honeyd_config =~ s/$/past_honeyd_files\//s;

        # Looks for past_xml_files directory specified in the config table or creates the
directory
        my $past_xml_config = $honeypot_xml;
        $past_xml_config =~ s/(honeypot.xml)$//s;
        if (-d
"".$past_xml_config."/past_xml_files"){}else{mkdir($past_xml_config."/past_xml_files
") || die "Unable to create past_xml directory <$!>\n";}
        $past_xml_config =~ s/$/past_xml_files\//s;

        print "Starting to write the XML and honeyd config files.\n";

        # Open file to be overwritten using the single greater than character
        open(HONEYDFILE, ">$honeyd_config");
        print HONEYDFILE "\##### Honeyd Configuration File ######\n";
        print HONEYDFILE "\##### " .localtime(). " #####\n\n\n";
        print HONEYDFILE
"\###################################################################";

        # Open file to be overwritten using the single greater than character
```

```
        open(XMLFILE, ">$honeypot_xml");
        print XMLFILE "<?xml version=\"1.0\"?>\n";
        print XMLFILE "<honeypot>\n";

                # Prepare and Execute DB Query
        my $query = "(select ip_addr, mac_addr, primary_os from p0f where
last_tstamp_time > '$scan_time' AND ip_addr NOT IN (select ip_addr from
nmap_machines where primary_os != 'unknown' AND last_tstamp_time > '$scan_time')
AND ip_addr NOT IN (select ip_addr from xprobe2_machines where primary_os !=
'unknown' AND last_tstamp_time > '$scan_time')) union (select ip_addr, mac_addr,
primary_os from xprobe2_machines where primary_os != 'unknown' AND
last_tstamp_time > '$scan_time' AND ip_addr NOT IN (select ip_addr from
nmap_machines where primary_os != 'unknown' AND last_tstamp_time > '$scan_time'))
union (select ip_addr, mac_addr, primary_os from nmap_machines where primary_os !=
'unknown' AND last_tstamp_time > '$scan_time') order by ip_addr;";
        my $select_stmt = $dbh->prepare($query);
        $select_stmt->execute();
        $select_stmt->bind_columns(\my $ip, \my $mac, \my $os);
        # LOOP THROUGH RESULTS
        while($select_stmt->fetch()) {
                my @family_arguments = split (/\s+/, $os);
                if (@family_arguments[0] eq "Microsoft") {$family =
@family_arguments[1];
                }else{$family = @family_arguments[0];}
                print HONEYDFILE "\n\ncreate ",$family,++$num."\n";
                print HONEYDFILE "set ".$family."$num personality \"".$os."\"\n";

                db_insert_lih_hih_link($insert_lih_info, $os, $ip, $mac, $dbh);
                # Prepare and Execute DB Query
                my $query2 = "Select link_id, lih_ip_addr from lih_hih_link where
lih_ip_addr = '$ip' AND lih_mac_addr = '$mac' AND lih_os_platform = '$os' AND
hih_ip_addr = 'unknown';";
                my $select_stmt2 = $dbh->prepare($query2);
                $select_stmt2->execute();
                $select_stmt2->bind_columns(\my $tid, \my $tip);
                my $lih_hih_id = 0;
                while ($select_stmt2->fetch()) {
                        $lih_hih_id = $tid;
                }
                $select_stmt2->finish;

                print XMLFILE " <target name=\"".$family."$num\" ip=\"$ip\"
mac=\"$mac\" lih_hih_id=\"$lih_hih_id\">\n";
                print XMLFILE "  <os_guess>\n";
```

```
                print XMLFILE "    <primary> \"$os\" </primary>\n";
                print XMLFILE "  </os_guess>\n";
                print XMLFILE "  <system_information>\n";

                # Prepare and Execute DB Query
                my $query2 = "Select ip_addr, mac_addr, Count(*) AS Num_Rst_Ack
from tcpdump_ports where rst_ack = 'Y' AND ip_addr = '$ip' AND mac_addr = '$mac'
AND last_tstamp_time > '$scan_time' group by ip_addr, mac_addr;";
                my $select_stmt2 = $dbh->prepare($query2);
                $select_stmt2->execute();
                my $tcp_reset = 0;
                $select_stmt2->bind_columns(\my $tip, \my $tmac, \my $rsta);
                while ($select_stmt2->fetch()) {
                        $tcp_reset = $rsta;
                }
                $select_stmt2->finish;
                if($tcp_reset > 25){
                        print HONEYDFILE "set ".$family."$num default tcp action
reset\n";
                }else{
                        print HONEYDFILE "set ".$family."$num default tcp action
closed\n";
                }

                # Prepare and Execute DB Query
                my $query2 = "Select ip_addr, mac_addr, Count(*) AS Num_UDP_Reply
from tcpdump_ports where udp_reply = 'Y' AND ip_addr = '$ip' AND mac_addr = '$mac'
AND last_tstamp_time > '$scan_time' group by ip_addr, mac_addr;";
                my $select_stmt2 = $dbh->prepare($query2);
                $select_stmt2->execute();
                my $udp_reply = 0;
                $select_stmt2->bind_columns(\my $tip, \my $tmac, \my $udpr);
                while ($select_stmt2->fetch()) {
                        $udp_reply = $udpr;
                }
                $select_stmt2->finish;
                if($udp_reply > 25){
                        print HONEYDFILE "set ".$family."$num default udp action
reset\n";
                }else{
                        print HONEYDFILE "set ".$family."$num default udp action
closed\n";
                }
```

```
		# Prepare and Execute DB Query
		my $query2 = "Select ip_addr, mac_addr, icmp_reply from tcpdump_icmp
where ip_addr = '$ip' AND mac_addr = '$mac' AND last_tstamp_time > '$scan_time'
limit 1;";
		my $select_stmt2 = $dbh->prepare($query2);
		$select_stmt2->execute();
		my $icmp_reply = 0;
		$select_stmt2->bind_columns(\my $tip, \my $tmac, \my $icmp);
		while ($select_stmt2->fetch()) {
			$icmp_reply = $icmp;
		}
		$select_stmt2->finish;
		if($icmp_reply eq 'Y'){
			print HONEYDFILE "set ".$family."$num default icmp action
open\n";
			print XMLFILE "    <icmp_reply state=\"open\"/>\n";
		}else{
			print HONEYDFILE "set ".$family."$num default icmp action
closed\n";
			print XMLFILE "    <icmp_reply state=\"closed\"/>\n";
		}

		# Prepare and Execute DB Query
		my $query2 = "Select ip_addr, mac_addr, firewall, lookup_link from p0f
where ip_addr = '$ip' AND mac_addr = '$mac' AND last_tstamp_time > '$scan_time'
limit 1;";
		my $select_stmt2 = $dbh->prepare($query2);
		my $num_rows = $select_stmt2->execute();
		$select_stmt2->bind_columns(\my $tip, \my $tmac, \my $fire, \my $look);
		while ($select_stmt2->fetch()) {
			print XMLFILE "    <firewall state=\"".$fire."\"/>\n";
			print XMLFILE "    <lookup_link state=\"".$look."\"/>\n";
		}
		$select_stmt2->finish;
		if ($num_rows ==0){
			print XMLFILE "    <firewall state=\"no/unknown\"/>\n";
			print XMLFILE "    <lookup_link state=\"unknown\"/>\n";
		}

		# Prepare and Execute DB Query
		my $query2 = "Select ip_addr, mac_addr, real_time_target_sec from
xprobe2_machines where ip_addr = '$ip' AND mac_addr = '$mac' AND last_tstamp_time
> '$scan_time' limit 1;";
		my $select_stmt2 = $dbh->prepare($query2);
```

```
                my $num_rows = $select_stmt2->execute();
                $select_stmt2->bind_columns(\my $tip, \my $tmac, \my $rtts);
                while ($select_stmt2->fetch()) {
                        print XMLFILE "    <real_time_target_seconds
seconds=\"".$rtts."\"/>\n";
                }
                $select_stmt2->finish;
                if ($num_rows ==0){
                        print XMLFILE "    <real_time_target_seconds
seconds=\"unknown\"/>\n";
                }

                # Prepare and Execute DB Query
                my $query2 = "Select ip_addr, mac_addr, uptime_seconds from p0f where
uptime_seconds != 'unknown' AND ip_addr = '$ip' AND mac_addr = '$mac' AND
last_tstamp_time > '$scan_time' limit 1;";
                my $select_stmt2 = $dbh->prepare($query2);
                my $temp = $select_stmt2->execute();
                if ($temp == 1){
                        $select_stmt2->bind_columns(\my $tip, \my $tmac, \my $upt);
                        while ($select_stmt2->fetch()) {
                                print HONEYDFILE "set ".$family."$num uptime
".$upt."\n";
                                print XMLFILE "    <uptime_seconds
time=\"".$upt."\"/>\n";
                        }
                }else {
                        print XMLFILE "    <uptime_seconds time=\"unknown\"/>\n";
                }
                $select_stmt2->finish;

                # Prepare and Execute DB Query
                my $query2 = "(select ip_addr, mac_addr, distance_hops from p0f where
distance_hops != 'unknown' AND ip_addr = '$ip' AND mac_addr = '$mac' AND
last_tstamp_time > '$scan_time') union (select ip_addr, mac_addr, distance_hops from
nmap_machines where distance_hops != 'unknown' AND ip_addr = '$ip' AND mac_addr
= '$mac' AND last_tstamp_time > '$scan_time') limit 1;";
                my $select_stmt2 = $dbh->prepare($query2);
                my $temp = $select_stmt2->execute();
                if ($temp == 1){
                        $select_stmt2->bind_columns(\my $tip, \my $tmac, \my $dho);
                        while ($select_stmt2->fetch()) {
                                print XMLFILE "    <distance_hops
hops=\"".$dho."\"/>\n";
```

```
                }
        }else {
                print XMLFILE "    <distance_hops hops=\"unknown\"/>\n";
        }
        $select_stmt2->finish;

        print XMLFILE "  </system_information>\n";
        print XMLFILE "  <port_scan>\n";

        # Prepare and Execute DB Query
        my $query2 = "(select ip_addr, mac_addr, port_num, protocol, service
from tcpdump_ports where syn_ack = 'Y' AND ip_addr = '$ip' AND mac_addr = '$mac'
AND last_tstamp_time > '$scan_time' AND port_num NOT IN (select
nmap_ports.port_num from nmap_machines, nmap_ports where
nmap_machines.machine_id = nmap_ports.machine_id AND nmap_machines.ip_addr =
'$ip' AND nmap_machines.mac_addr = '$mac' AND nmap_machines.last_tstamp_time >
'$scan_time') AND port_num NOT IN (select xprobe2_ports.port_num from
xprobe2_machines, xprobe2_ports where xprobe2_machines.machine_id =
xprobe2_ports.machine_id AND xprobe2_machines.ip_addr = '$ip' AND
xprobe2_machines.mac_addr = '$mac' AND xprobe2_machines.last_tstamp_time >
'$scan_time')) union (select xprobe2_machines.ip_addr, xprobe2_machines.mac_addr,
xprobe2_ports.port_num, xprobe2_ports.protocol, xprobe2_ports.service from
xprobe2_machines, xprobe2_ports where xprobe2_machines.machine_id =
xprobe2_ports.machine_id AND xprobe2_machines.ip_addr = '$ip' AND
xprobe2_machines.mac_addr = '$mac' AND xprobe2_machines.last_tstamp_time >
'$scan_time' AND port_num NOT IN (select nmap_ports.port_num from
nmap_machines, nmap_ports where nmap_machines.machine_id =
nmap_ports.machine_id AND nmap_machines.ip_addr = '$ip' AND
nmap_machines.mac_addr = '$mac' AND nmap_machines.last_tstamp_time >
'$scan_time')) union (select nmap_machines.ip_addr, nmap_machines.mac_addr,
nmap_ports.port_num, nmap_ports.protocol, nmap_ports.service from nmap_machines,
nmap_ports where nmap_machines.machine_id = nmap_ports.machine_id AND
nmap_machines.ip_addr = '$ip' AND nmap_machines.mac_addr = '$mac' AND
nmap_machines.last_tstamp_time > '$scan_time') order by ip_addr, port_num, protocol,
service;";
        my $select_stmt2 = $dbh->prepare($query2);
        my $temp = $select_stmt2->execute();
        if ($temp >= 1){
                $select_stmt2->bind_columns(\my $ip, \my $mac, \my $pn, \my
$p, \my $ser);
                # LOOP THROUGH RESULTS
                while($select_stmt2->fetch()) {
                        # Prepare and Execute DB Query
```

```
                                my $query3 = "select script_lang, path_and_filename from
honeyd_scripts where primary_os LIKE '$family%' AND port_num = '$pn' AND
protocol = '$p';";
                                my $select_stmt3 = $dbh->prepare($query3);
                                $select_stmt3->execute();
                                # Assign fields to variables
                                $select_stmt3->bind_columns(\my $script_lang, \my
$path_and_filename);
                                while ($select_stmt3->fetch()) {
                                 if ($script_lang == $pattern6) {
                                        print HONEYDFILE "add ".$family."$num ".$p."
port ".$pn." \"$script_lang $path_and_filename\"\n";
                                        $script_lang = 1;
                                 }
                                }
                                if ($script_lang == 0) {
                                 print HONEYDFILE "add ".$family."$num ".$p." port
".$pn." open\n";
                                }
                                $select_stmt3->finish;

                                # Prepare and Execute DB Query
                                my $query3 = "Select ip_addr, mac_addr from
nmap_machines, nmap_ports where nmap_machines.machine_id =
nmap_ports.machine_id AND nmap_machines.ip_addr = '$ip' AND
nmap_machines.mac_addr = '$mac' AND nmap_ports.port_num = '$pn' AND
nmap_ports.protocol = '$p' AND nmap_machines.last_tstamp_time > '$scan_time';";
                                my $select_stmt3 = $dbh->prepare($query3);
                                my $num_rows = $select_stmt3->execute();
                                $select_stmt3->bind_columns(\my $tip, \my $tmac);
                                while ($select_stmt3->fetch()) {
                                        print XMLFILE "    <port number=\"".$pn."\"
protocol=\"".$p."\" service=\"".$ser."\"/>\n";
                                }
                                $select_stmt3->finish;

                        }
                }
                $select_stmt2->finish;

                my $pattern = '^\d{1,3}\.\d{1,3}\.\d{1,3}\.';
                my $pattern2 = '\d{1,3}$';

                if ($honeyd_ip_binding eq "iS") {
```

```
                print HONEYDFILE "bind $ip ".$family."$num";
        }
        elsif ($honeyd_ip_binding eq "iD") {
                $ip =~ s/$pattern//s;
                $honeyd_ip_temp =~ s/$pattern2/$ip/s;
                print HONEYDFILE "bind $honeyd_ip_temp ".$family."$num";
        }
        elsif ($honeyd_ip_binding eq "iR") {
                if ($honeyd_count >= 0 && $honeyd_count < $honeyd_max) {
                        $honeyd_ip_temp =~ s/$pattern2/$honeyd_count/s;
                }
                if ($honeyd_count >= $honeyd_max) {
                        $honeyd_ip_temp =~ s/$pattern2/$honeyd_count/s;
                        print HONEYDFILE "bind $honeyd_ip_temp
".$family."$num";
                        print HONEYDFILE
"\n\n#########################################################################
\n";
                        close(HONEYDFILE);
                        print "The honeyd config file is complete!\n";

                        print XMLFILE "  </port_scan>\n";
                        print XMLFILE " </target>\n";
                        print XMLFILE "</honeypot>\n";
                        close(XMLFILE);
                        print "The xml honeypot file is complete!\n";

                        #This copies the current honeyd.conf file and places the
copy in a past_honeyd_files folder
                        #located in the same folder as the config file. The copy is
named honeyd_<currenttime>.conf.
                        my $current_config = $past_honeyd_config;
                        $current_config =~ s/$/honeyd/s;
                        $current_config = join ('_', $current_config, time());
                        $current_config =~ s/$/\.conf/s;
                        copy("$honeyd_config","$current_config") or die
"Copying honeyd.conf failed: $!";

                        #This copies the current honeypot.xml file and places the
copy in a past_xml_files folder
                        #located in the same folder as the xml file. The copy is
named honeypot_<currenttime>.xml.
                        my $current_config = $past_xml_config;
                        $current_config =~ s/$/honeypot/s;
```

```
                        $current_config = join ('_', $current_config, time());
                        $current_config =~ s/$/\.xml/s;
                        copy("$honeypot_xml","$current_config") or die "Copying
honeypot.xml failed: $!";

                        #Return from output_honeyd_config since IP range is full
                        return;
                    }
                    $honeyd_count++;
                    print HONEYDFILE "bind $honeyd_ip_temp ".$family."$num";
                }
                elsif ($honeyd_ip_binding eq "iI") {
                    my $first_ip, my $second_mac;

                    ($first_ip, $second_mac) = register_dhcp($mac, $dbh);

                    print HONEYDFILE "set ".$family."$num ethernet
\"$second_mac\"\n";
                    print HONEYDFILE "bind $first_ip ".$family."$num";
                }
                print XMLFILE "  </port_scan>\n";
                print XMLFILE " </target>\n";
        }
        $select_stmt->finish;

        print HONEYDFILE
"\n\n#########################################################################
\n";
        close(HONEYDFILE);
        print "The honeyd config file is complete!\n";

        print XMLFILE "</honeypot>\n";
        close(XMLFILE);
        print "The xml honeypot file is complete!\n";

        #This copies the current honeyd.conf file and places the copy in a
past_honeyd_files folder
        #located in the same folder as the config file. The copy is named
honeyd_<currenttime>.conf.
        my $current_config = $past_honeyd_config;
        $current_config =~ s/$/honeyd/s;
        $current_config = join ('_', $current_config, time());
        $current_config =~ s/$/\.conf/s;
```

```
        copy("$honeyd_config","$current_config") or die "Copying honeyd.conf failed:
$!";

        #This copies the current honeypot.xml file and places the copy in a past_xml_files
folder
        #located in the same folder as the xml file. The copy is named
honeypot_<currenttime>.xml.
        my $current_config = $past_xml_config;
        $current_config =~ s/$/honeypot/s;
        $current_config = join ('_', $current_config, time());
        $current_config =~ s/$/\.xml/s;
        copy("$honeypot_xml","$current_config") or die "Copying honeypot.xml failed:
$!";
 }

sub db_query_dhcp_register
 {      my $mac_target = shift;
        my $dbh = shift;

        my $mac_pattern = '^\w{2}\:\w{2}\:\w{2}\:\w{2}\:\w{2}\:';
        my $mac_pattern2 = '\w{2}$';
        my $new_mac = $mac_target, my $last_mac, my $mac_check = 0, my
$ran_num2 = int(rand(0xFFF));

        while ($mac_check == 0)
        {
                $last_mac = $new_mac;
                $last_mac =~ s/$mac_pattern//s;
                my $ran_num2 = int(rand(0xFFF));
                if (hex($ran_num2) % 2) {
                        if ($last_mac eq '00'){
                                $last_mac = 'ff';
                        }else {
                                $last_mac = hex($last_mac) - 1;
                                $last_mac = sprintf("%x", $last_mac);
                                if (hex($last_mac) < 16){ $last_mac = '0'.$last_mac;}
                        }
                } else  {
                        if ($last_mac eq 'ff'){
                                $last_mac = '00';
                        }else {
                                $last_mac = hex($last_mac) + 1;
                                $last_mac = sprintf("%x", $last_mac);
                                if (hex($last_mac) < 16){ $last_mac = '0'.$last_mac;}
```

```
                }
            }
            $new_mac = $mac_target;
            $new_mac =~ s/$mac_pattern2/$last_mac/s;

            # Prepare and Execute DB Query
            my $query = "select dhcp_id, mac_addr from dhcp where mac_addr =
'$new_mac' limit 1;";
            my $select_stmt = $dbh->prepare($query);
            $select_stmt->execute();
            my $new_mac_temp = 0;
            $select_stmt->bind_columns(\my $did, \my $mac);
            while ($select_stmt->fetch()) {
                $new_mac_temp = $mac;
            }
            $select_stmt->finish;
            if ($new_mac_temp eq 0) {$mac_check = 1;}
        }
        return $new_mac;
}

# Register IP address through DHCP
sub register_dhcp
{       my $mac_target = shift;
        my $dbh = shift;

        my $ran_num = int(rand(0xFFFFFFFF));

        # Prepare and Execute DB Query
        my $query = "select dhcp_id, ip_addr from dhcp ORDER BY dhcp_id DESC
limit 1;";
        my $select_stmt = $dbh->prepare($query);
        $select_stmt->execute();
        my $dhcp_id = 0;
        $select_stmt->bind_columns(\my $did, \my $ip);
        while ($select_stmt->fetch()) {
            $dhcp_id = $did + 1;
        }
        $select_stmt->finish;

        my $mac_addr_dhcp = db_query_dhcp_register($mac_target, $dbh);
        my $mac_addr_short = $mac_addr_dhcp;
        # Remove : from MAC address
        for (split /\:/, $mac_addr_short) {
```

```
        $mac_addr_short =~ s/[\:]//;
}

# Change MAC address of ethernet interface to new MAC for honeypots
system `ifconfig $eth_interface hw ether $mac_addr_dhcp`;

#print "Opening socket\n";
my $handle = IO::Socket::INET->new(Proto => 'udp',
                        timeout => "5",
                        Broadcast => 0,
                        PeerAddr => $dhcp_server,
                        PeerPort => '67',
                        LocalPort => '68')
        or die "Socket creation error: $@\n"; # yes, it uses $@ here

# create DHCP Packet DISCOVER
my $discover = Net::DHCP::Packet->new(
                Chaddr => $mac_addr_short,
                Xid => $ran_num, # random xid
                DHO_DHCP_MESSAGE_TYPE() => DHCPDISCOVER());

# Send DISCOVER packet
#print "\nSending DISCOVER to ".$dhcp_server.":67\n";
#print $discover->toString();
$handle->send($discover->serialize())
        or die "Error sending broadcast inform:$!\n";

my $newmsg, my $packet;
eval {
        local $SIG{ALRM} = sub { die "alarm\n" }; # NB: \n required
        alarm 10;

        # Receive response
        #print "\nWaiting for response from server\n";
        $handle->recv($newmsg, 4096) or die("recv:$!");

        alarm 0;
};
if ($@) {
        print "The DHCP discover process could not finish.\n";
        print "Please check DHCP settings in config table and restart program.\n";
        system "killall p0f";
        system "killall tcpdump";
        system "killall nmap";
```

```
        system "killall xprobe2";
        system "kill $p0f_pid";
        system "kill $tcp_pid";
        system "kill $act_pid";
        sleep 1;
        exit;
        # timed out
}else {
        $packet = Net::DHCP::Packet->new($newmsg);
        #print "Got response\n";
        #print $packet->toString();
        # didn't
}

# Create DHCP Packet REQUEST
my $request = Net::DHCP::Packet->new(
          Chaddr => $mac_addr_short,
          Xid => $ran_num, # random xid
          DHO_DHCP_MESSAGE_TYPE() => DHCPREQUEST(),
                DHO_DHCP_SERVER_IDENTIFIER()=> $dhcp_server,
          DHO_DHCP_REQUESTED_ADDRESS() => $packet->yiaddr());

# Send REQUEST packet
#print "\nSending REQUEST to ".$dhcp_server.":67\n";
#print $request->toString();
$handle->send($request->serialize())
        or die "Error sending:$!\n";

eval {
        local $SIG{ALRM} = sub { die "alarm\n" }; # NB: \n required
        alarm 10;

        # Receive response
        #print "\nWaiting for response from server\n";
        $handle->recv($newmsg, 4096) or die("recv:$!");

        alarm 0;
};
if ($@) {
        print "The DHCP discover process could not finish.\n";
        print "Please check DHCP settings in config table and restart program.\n";
        system "killall p0f";
        system "killall tcpdump";
        system "killall nmap";
```

```
                system "killall xprobe2";
                system "kill $p0f_pid";
                system "kill $tcp_pid";
                system "kill $act_pid";
                sleep 1;
                exit;
                # timed out
        }else {
                $packet = Net::DHCP::Packet->new($newmsg);
                #print "Got response\n";
                #print $packet->toString();
                # didn't
        }

        my $ip_addr_dhcp = $packet->yiaddr();
        my $lease_time = $packet-
>getOptionValue(DHO_DHCP_RENEWAL_TIME());

        # Close IO::Socket::INET connection
        close $handle;

        # Change MAC address of ethernet interface to original MAC address
        system `ifconfig $eth_interface hw ether $my_mac`;

        db_insert_dhcp_register($insert_dhcp_register, $ip_addr_dhcp, $mac_addr_dhcp,
$lease_time, $dhcp_id, $dbh);

        my @n;
        $n[0] = $ip_addr_dhcp;
        $n[1] = $mac_addr_dhcp;
        return @n;
}

# Insert information from registering IP address through DHCP
sub db_insert_dhcp_register
{       my $insert_dhcp_register = shift;
        my $ip_addr_dhcp = shift;
        my $mac_addr_dhcp = shift;
        my $lease_time = shift;
        my $dhcp_id = shift;
        my $dbh = shift;

        my $time = time();
        my $renewal_time = $time + $lease_time;
```

```
        # finalize the insert statement
        $insert_dhcp_register->bind_param( 1, $dhcp_id);   # refers to the first ? in the
query
        $insert_dhcp_register->bind_param( 2, $ip_addr_dhcp);  # refers to the second ?
in the query
        $insert_dhcp_register->bind_param( 3, uc($mac_addr_dhcp)); # refers to the third
? in the query
        $insert_dhcp_register->bind_param( 4, $lease_time);         # refers to the fourth ?
in the query
        $insert_dhcp_register->bind_param( 5, $renewal_time);      # refers to the fifth ?
in the query
        my $date_created = date_time();
        $insert_dhcp_register->bind_param( 6, $date_created);      # refers to the sixth ?
in the query
        $insert_dhcp_register->bind_param( 7, $date_created);      # refers to the seventh
? in the query
        my $time = time();
        $insert_dhcp_register->bind_param( 8, $time);              # refers to the eighth ?
in the query

        $insert_dhcp_register->execute();     # excute the SQL statement
}

# Find IP address to renew through DHCP
sub db_query_renew_dhcp
{       my $dbh = shift;
        my $time = time();

        # Prepare and Execute DB Query
        my $query = "select ip_addr, mac_addr, lease_time_seconds,
renewal_tstamp_time from dhcp where renewal_tstamp_time < $time limit 1;";
        my $select_stmt = $dbh->prepare($query);
        $select_stmt->execute();
        $select_stmt->bind_columns(\my $dip, \my $dmac, \my $lts, \my $rett);
        while ($select_stmt->fetch()) {
                if (($lts + $rett) > $time){renew_dhcp($dip, $dmac, $dbh);}
        }
        $select_stmt->finish;
}

# Renew IP address through DHCP
sub renew_dhcp
{       my $ip_addr_dhcp = shift;
```

```
my $mac_addr_dhcp = shift;
my $dbh = shift;

my $ran_num = int(rand(0xFFFFFFFF));

my $mac_addr_short = $mac_addr_dhcp;
# Remove : from MAC address
for (split /\:/, $mac_addr_short) {
        $mac_addr_short =~ s/[\:]//;
}

# Change MAC address of ethernet interface to new MAC for honeypots
system `ifconfig $eth_interface hw ether $mac_addr_dhcp`;

#print "Opening socket\n";
my $handle = IO::Socket::INET->new(Proto => 'udp',
                        Broadcast => 0,
                        PeerAddr => $dhcp_server,
                        PeerPort => '67',
                        LocalPort => '68')
        or die "Socket creation error: $@\n"; # yes, it uses $@ here

# Create DHCP Packet RENEW
my $renew = Net::DHCP::Packet->new(
          Ciaddr => $ip_addr_dhcp,
          Chaddr => $mac_addr_short,
          Xid => $ran_num, # random xid
          DHO_DHCP_MESSAGE_TYPE() => DHCPREQUEST(),
                DHO_DHCP_SERVER_IDENTIFIER()=> $dhcp_server,
          DHO_DHCP_REQUESTED_ADDRESS() => $ip_addr_dhcp);

# Send RENEW packet
#print "\nSending RENEW to ".$dhcp_server.":67\n";
#print $renew->toString();
$handle->send($renew->serialize())
         or die "Error sending:$!\n";

my $newmsg, my $packet;
eval {
        local $SIG{ALRM} = sub { die "alarm\n" }; # NB: \n required
        alarm 10;

        # Receive response
        #print "\nWaiting for response from server\n";
```

```
                $handle->recv($newmsg, 4096) or die("recv:$!");

                alarm 0;
        };
        if ($@) {
                print "The DHCP discover process could not finish.\n";
                print "Please check DHCP settings in config table and restart program.\n";
                system "killall p0f";
                system "killall tcpdump";
                system "killall nmap";
                system "killall xprobe2";
                system "kill $p0f_pid";
                system "kill $tcp_pid";
                system "kill $act_pid";
                sleep 1;
                exit;
                # timed out
        }else {
                $packet = Net::DHCP::Packet->new($newmsg);
                #print "Got response\n";
                #print $packet->toString();
                # didn't
        }

        my $ip_addr = $packet->yiaddr();
        my $lease_time = $packet-
>getOptionValue(DHO_DHCP_RENEWAL_TIME());

        # Close IO::Socket::INET connection
        close $handle;

        # Change MAC address of ethernet interface to original MAC address
        system `ifconfig $eth_interface hw ether $my_mac`;

        db_update_dhcp_renew($ip_addr_dhcp, $mac_addr_dhcp, $lease_time, $dbh);
 }

# Update information from renewing IP address through DHCP
sub db_update_dhcp_renew
 {      my $ip_addr_dhcp = shift;
        my $mac_addr_dhcp = shift;
        my $lease_time = shift;
        my $dbh = shift;
```

```
        my $last_tstamp = date_time();
        my $time = time();
        my $renewal_time = $time + $lease_time;

        my $query = "update dhcp set lease_time_seconds = '$lease_time',
renewal_tstamp_time = '$renewal_time', last_tstamp = '$last_tstamp', last_tstamp_time =
'$time' where ip_addr = '$ip_addr_dhcp' AND mac_addr = '$mac_addr_dhcp';";
        my $update_stmt = $dbh->prepare($query);
        $update_stmt->execute();
}

# Find IP address to release through DHCP
sub db_query_release_dhcp
{       my $dbh = shift;
        my $num_rows;

        do
        {
                # Prepare and Execute DB Query
                my $query = "select * from dhcp;";
                my $select_stmt = $dbh->prepare($query);
                $num_rows = $select_stmt->execute();
                $select_stmt->finish;

                # Prepare and Execute DB Query
                my $query = "select ip_addr, mac_addr from dhcp limit 1;";
                my $select_stmt = $dbh->prepare($query);
                $select_stmt->execute();
                $select_stmt->bind_columns(\my $dip, \my $dmac);
                while ($select_stmt->fetch()) {
                        release_dhcp($dip, $dmac, $dbh);
                }
                $select_stmt->finish;
        }while($num_rows > 0);
}

# Release IP address through DHCP
sub release_dhcp
{       my $ip_addr_dhcp = shift;
        my $mac_addr_dhcp = shift;
        my $dbh = shift;

        my $ran_num = int(rand(0xFFFFFFFF));
```

```
        my $mac_addr_short = $mac_addr_dhcp;
        # Remove : from MAC address
        for (split /\:/, $mac_addr_short) {
                $mac_addr_short =~ s/[\:]//;
        }

        # Change MAC address of ethernet interface to new MAC for honeypots
        system `ifconfig $eth_interface hw ether $mac_addr_dhcp`;

        #print "Opening socket\n";
        my $handle = IO::Socket::INET->new(Proto => 'udp',
                                Broadcast => 0,
                                PeerAddr => $dhcp_server,
                                PeerPort => '67',
                                LocalPort => '68')
                or die "Socket creation error: $@\n"; # yes, it uses $@ here

        # Create DHCP Packet RELEASE
        my $release = Net::DHCP::Packet->new(
                Ciaddr => $ip_addr_dhcp,
                Chaddr => $mac_addr_short,
                Xid => $ran_num,                # random xid
                Flags => 0x8000,  # ask for broadcast answer
    DHO_DHCP_MESSAGE_TYPE() => DHCPRELEASE(),
                        DHO_DHCP_SERVER_IDENTIFIER()=> $dhcp_server);

        # Send RELEASE packet
        #print "\nSending RELEASE to ".$dhcp_server.":67\n";
        #print $release->toString();
        $handle->send($release->serialize())
                or die "Error sending:$!\n";

        # Close IO::Socket::INET connection
        close $handle;

        # Change MAC address of ethernet interface to original MAC address
        system `ifconfig $eth_interface hw ether $my_mac`;

        db_delete_dhcp_release($ip_addr_dhcp, $mac_addr_dhcp, $dbh);
}

# Delete information from releasing IP address through DHCP
sub db_delete_dhcp_release
{       my $ip_addr_dhcp = shift;
```

```
        my $mac_addr_dhcp = shift;
        my $dbh = shift;

        my $query = "delete from dhcp where ip_addr = '$ip_addr_dhcp' AND mac_addr
= '$mac_addr_dhcp';";
        my $delete_stmt = $dbh->prepare($query);
        $delete_stmt->execute();
}

# Insert information for the LIHoneypot
sub db_insert_lih_hih_link
{       my $insert_lih_info = shift;
        my $os_lih = shift;
        my $ip_addr_lih = shift;
        my $mac_addr_lih = shift;
        my $dbh = shift;

        my $os_hih = "unknown";
        my $ip_addr_hih = "unknown";
        my $mac_addr_hih = "unknown";
        my $location_hih = "unknown";
        my $state_hih = "unknown";

        # finalize the insert statement
        $insert_lih_info->bind_param( 1, $os_lih);   # refers to the first ? in the query
        $insert_lih_info->bind_param( 2, $ip_addr_lih);  # refers to the second ? in the
query
        $insert_lih_info->bind_param( 3, uc($mac_addr_lih)); # refers to the third ? in the
query

        $insert_lih_info->bind_param( 4, $os_hih);          # refers to the fourth ? in the
query
        $insert_lih_info->bind_param( 5, $ip_addr_hih);     # refers to the fifth ? in the
query
        $insert_lih_info->bind_param( 6, $mac_addr_hih);  # refers to the sixth ? in the
query
        $insert_lih_info->bind_param( 7, $location_hih);    # refers to the seventh ? in
the query
        $insert_lih_info->bind_param( 8, $state_hih);       # refers to the eighth ? in the
query
        my $date_created = date_time();
        $insert_lih_info->bind_param( 9, $date_created);    # refers to the ninth ? in the
query
```

```
        $insert_lih_info->bind_param( 10, $date_created);  # refers to the tenth ? in the
query

        $insert_lih_info->execute();            # excute the SQL statement
}

sub date_time
{       my @months = qw(Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec);
        my @weekDays = qw(Sun Mon Tue Wed Thu Fri Sat Sun);
        (my $second, my $minute, my $hour, my $dayOfMonth, my $month, my
$yearOffset, my $dayOfWeek, my $dayOfYear, my $daylightSavings) = localtime();
        my $year = 1900 + $yearOffset;
        my $fixed_month = 1 + $month;
        #my $theTime = "$weekDays[$dayOfWeek] $months[$month] $dayOfMonth
$hour:$minute:$second $year";
        my $theTime = "$year-$fixed_month-$dayOfMonth $hour:$minute:$second";
}
```

P0f_mysql.pl

```
#!/usr/bin/perl
#p0f_mysql.pl by Chris Hecker, 2007

#This version currently works with p0f v2.0.8

use strict;
use Getopt::Long;
use vars qw(%G);
use DBI;

#Global Variables
my $my_ip, my $machine_id, my $mac_addr, my $firewall, my $lookup_link,
my $distance_hops, my $uptime_hours, my $uptime_seconds, my $primary_os;

GetOptions( 'i=s'       => \$G{i});    #       Ethernet interface utilized to capture packets
                #

if($G{i} eq ''){print "Please use the following option \"-i ethernet interface\"\n"; exit(0);}

# Find ip address of ethernet interface
$my_ip = `/sbin/ifconfig $G{i} |grep inet |cut -d ' ' -f 12|cut -d ':' -f 2`; chomp $my_ip;
chomp $my_ip;

# Connect to Database
my $dbh= connect_to_db();

# prepare the insert statement just once, the actual values will replace the ? later
my $insert_p0f = $dbh->prepare( "INSERT INTO p0f (ip_addr, mac_addr, primary_os,
firewall, lookup_link, uptime_seconds, distance_hops, date_created, last_tstamp,
last_tstamp_time) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?);");

db_threads($dbh);

open(P0FIN, "p0f -i $G{i} -p -l |") || die "p0f could not be opened";
while (<P0FIN>) {
        my $machine_id = 0;
        my $input_p0f = <P0FIN>;
        my @arguments = split (/\s+/, $input_p0f);

        my $ip_addr = @arguments[0];
```

```
        my $pattern = '\:\d{1,5}$';
        $ip_addr =~ s/$pattern//s;
        my $mac_pattern = '(([0-9a-fA-F][0-9a-fA-F]:){5}([0-9a-fA-F][0-9a-fA-F]))';
        my @arping = `arping -f $ip_addr>&1`;
        my $line = @arping[1];
        $line =~ /$mac_pattern/; chop $line; my $mac_addr = $1;

        if($input_p0f =~ m/(firewall!)/i){$firewall = 'yes';} else {$firewall =
'no/unknown';}
        if($input_p0f =~ m/link: (.*)\)/i){$lookup_link = $1;} else {$lookup_link =
'unknown';}
        if($input_p0f =~ m/up: (.*)hrs/i && $firewall eq 'no/unknown')
        {
                $uptime_hours = $1; chop $uptime_hours;
                $uptime_seconds = $uptime_hours * 3600;
                $input_p0f =~ m/\- (.*) \(up:/i;
                $primary_os = $1;
        }elsif ($input_p0f =~ m/up: (.*)hrs/i)
        {
                $uptime_hours = $1; chop $uptime_hours;
                $uptime_seconds = $uptime_hours * 3600;
                $input_p0f =~ m/\- (.*) \(firewall/i;
                $primary_os = $1;
        }else{
                $uptime_seconds = 'unknown';
                $input_p0f =~ m/\- (.*) \-\>/i;
                $primary_os = $1;
        }
        if($input_p0f =~ m/distance (.*)\,/i){$distance_hops = $1;} else {$distance_hops
= 'unknown';}

        $machine_id = db_query_p0f($ip_addr, $mac_addr, $primary_os, $dbh);

        if ($ip_addr eq $my_ip){
        }else{
                if ($machine_id ==0){
                        db_insert_p0f($insert_p0f, $ip_addr, $mac_addr, $primary_os,
                        $firewall, $lookup_link, $uptime_seconds, $distance_hops, $dbh);
                }else{
                        my $last_tstamp = date_time();
                        my $time = time();
                        my $query = "update p0f set last_tstamp = '$last_tstamp',
last_tstamp_time = '$time' where machine_id = '$machine_id';";
                        my $update_stmt = $dbh->prepare($query);
```

```
                        $update_stmt->execute();
                }
        }
}

# Disconnect from Database
$dbh->disconnect();
print "P0f disconnected from Database.\n";

exit;

# connect to the database
sub connect_to_db
{       # Database Information
        my $db="honeypot_scanner";
        my $userid="root";
        my $passwd="rootpass";
        my $connectionInfo="dbi:mysql:$db";

        # Make Connection to Database
        my $dbh = DBI->connect($connectionInfo,$userid,$passwd, {
    RaiseError => 1,
    AutoCommit => 0
    } ) || die "Database connection not made: $DBI::errstr";
        return( $dbh);
}

sub db_threads
{       my $dbh = shift;

        my $thr_name = "p0f_scan";
        my $thr_pid = $$;
        my $last_tstamp = date_time();

        my $query = "update threads set thr_pid = '$thr_pid', last_tstamp = '$last_tstamp'
where thr_name = '$thr_name';";
        my $update_stmt = $dbh->prepare($query);
        $update_stmt->execute();
}

sub db_query_p0f
{       my $ip_addr = shift;
        my $mac_addr = shift;
        my $primary_os = shift;
```

```
        my $dbh = shift;

        # Prepare and Execute DB Query
        my $query = "select machine_id, last_tstamp from p0f where ip_addr = '$ip_addr'
AND mac_addr = '$mac_addr' AND primary_os = '$primary_os';";
        my $select_stmt = $dbh->prepare($query);
        $select_stmt->execute();
        $select_stmt->bind_columns(\my $m_id, \my $lts);
        while ($select_stmt->fetch()) {
         $machine_id = $m_id;
        }
        $select_stmt->finish;

        return $machine_id;
}

sub db_insert_p0f
{       my $insert_p0f = shift;
        my $ip_addr = shift;
        my $mac_addr = shift;
        my $primary_os = shift;
        my $firewall = shift;
        my $lookup_link = shift;
        my $uptime_seconds = shift;
        my $distance_hops = shift;
        my $dbh = shift;

        # finalize the insert statement
        $insert_p0f->bind_param( 1, $ip_addr);      # refers to the first ? in the query
        $insert_p0f->bind_param( 2, $mac_addr);     # refers to the second ? in the query
        $insert_p0f->bind_param( 3, $primary_os);  # refers to the third ? in the query
        $insert_p0f->bind_param( 4, $firewall);             # refers to the fourth ? in the
query
        $insert_p0f->bind_param( 5, $lookup_link);          # refers to the fifth ? in the
query
        $insert_p0f->bind_param( 6, $uptime_seconds); # refers to the sixth ? in the
query
        $insert_p0f->bind_param( 7, $distance_hops);        # refers to the seventh ? in
the query
        my $date_created = date_time();
        $insert_p0f->bind_param( 8, $date_created);         # refers to the eighth ? in the
query
        $insert_p0f->bind_param( 9, $date_created);         # refers to the ninth ? in the
query
```

```
        my $time = time();
        $insert_p0f->bind_param( 10, $time);                    # refers to the tenth ? in the
query

        $insert_p0f->execute();         # excute the SQL statement
}

sub date_time
{       my @months = qw(Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec);
        my @weekDays = qw(Sun Mon Tue Wed Thu Fri Sat Sun);
        (my $second, my $minute, my $hour, my $dayOfMonth, my $month, my
$yearOffset, my $dayOfWeek, my $dayOfYear, my $daylightSavings) = localtime();
        my $year = 1900 + $yearOffset;
        my $fixed_month = 1 + $month;
        #my $theTime = "$weekDays[$dayOfWeek] $months[$month] $dayOfMonth
$hour:$minute:$second $year";
        my $theTime = "$year-$fixed_month-$dayOfMonth $hour:$minute:$second";
}
```

Tcpdump_mysql.pl

```
#!/usr/bin/perl
#tcpdump_mysql.pl by Chris Hecker, 2007-2011

use strict;
use Getopt::Long;
use vars qw(%G);
use DBI;

#Global Variables
my $machine_id, my $port, my $reply, my $select_stmt, my $noise, my $nmap_exe, my
$xprobe2_xml,
my $protocol, my $port_num, my $syn_ack, my $rst_ack, my $udp_reply;

GetOptions( 'ip=s'      => \$G{ip},    #       IP address(es) from which to gather
infomration                     #
        'i=s'   => \$G{i});             #       Ethernet interface utilized to capture packets
                #

if($G{ip} eq '' || $G{i} eq ''){print "Please use the following option \"-ip
XXX.XXX.XXX.XXX (see \"man tcpdump\" for more information) -i ethernet
interface\"\n"; exit(0);}

# Find ip address of ethernet interface
my $my_ip = `/sbin/ifconfig $G{i} |grep inet |cut -d ' ' -f 12|cut -d ':' -f 2`; chomp
$my_ip; chomp $my_ip;

# Connect to Database
my $dbh= connect_to_db();

# prepare the insert statement just once, the actual values will replace the ? later
my $insert_tcpdump_ports= $dbh->prepare( "INSERT INTO tcpdump_ports (ip_addr,
mac_addr, port_num, protocol, service, extra_info, syn_ack, rst_ack, udp_reply,
date_created, last_tstamp, last_tstamp_time) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?);");
my $insert_tcpdump_icmp= $dbh->prepare( "INSERT INTO tcpdump_icmp (ip_addr,
mac_addr, icmp_reply, date_created, last_tstamp, last_tstamp_time) VALUES (?, ?, ?, ?,
?, ?);");
my $insert_queue= $dbh->prepare( "INSERT INTO scan_queue (ip_addr, mac_addr,
date_created, last_tstamp_time) VALUES (?, ?, ?, ?);");

db_threads($dbh);
```

```
select ((select (my $input_tcp), $|=1) [0]);
select ((select (my $tcpfile), $|=1) [0]);

open ($tcpfile, ">/home/user/Documents/phase1_hs/tcpdump_".time()."" || die "The
TCPDUMP file could not be opened");
print $tcpfile "\##### tcpdump File ######\n";
print $tcpfile "\##### " .localtime(). " #####\n\n\n";
print $tcpfile
"\##########################################################################\n"
;

open(TCPIN, "tcpdump -nne -i $G{i} src net $G{ip} and not src $my_ip |") || die
"TCPDUMP could not be opened";
while (<TCPIN>) {
        $machine_id = 0;
        $input_tcp = <TCPIN>;
        my @arguments = split (/\s+/, $input_tcp);
        if (@arguments[5] eq "IPv4"){
                my $mac_addr = @arguments[1];
                my @array = split (/\./, @arguments[9]);
                my $ip_addr = join ('.', @array[0], @array[1], @array[2], @array[3]);
                if ($ip_addr ne $my_ip){
                        db_active_scan_queue($insert_queue, $ip_addr, $mac_addr,
$dbh);

                        if($input_tcp =~ m/\s+(ICMP)\s+(echo)\s+(reply,)\s+/){
                                my $icmp_reply = "Y";
                                db_insert_tcpdump_icmp($insert_tcpdump_icmp,
$ip_addr, $mac_addr, $icmp_reply, $dbh, $tcpfile, $input_tcp);
                        }else{
                                if($input_tcp =~ m/\s+(UDP)\s+/){
                                        $protocol = 'udp';
                                        $port_num = @array[4];
                                        $syn_ack = "N"; $rst_ack = "N";$udp_reply = "Y";
                                        db_query_services($insert_tcpdump_ports,
$ip_addr, $mac_addr, $port_num, $protocol, $syn_ack, $rst_ack, $udp_reply, $dbh,
$tcpfile, $input_tcp);
                                }elsif($input_tcp =~ m/\[(S|R)\.\]/){
                                        if($input_tcp =~ m/\s+(ack)\s+/){
                                                $protocol = 'tcp';
                                                $port_num = @array[4];
                                                if($input_tcp =~ m/\[R\.\]/){
```

```
                                                $syn_ack = "N"; $rst_ack = "Y";
$udp_reply = "N";
                                        }else{
                                                $syn_ack = "Y"; $rst_ack = "N";
$udp_reply = "N";
                                        }
                                        db_query_services($insert_tcpdump_ports,
$ip_addr, $mac_addr, $port_num, $protocol, $syn_ack, $rst_ack, $udp_reply, $dbh,
$tcpfile, $input_tcp);
                                }
                        }
                }
        }
}

print $tcpfile
"\n\n#############################################################################
\n";
close($tcpfile);

# Disconnect from Database
$dbh->disconnect();
print "TCPdump disconnected from Database.\n";

exit;

# connect to the database
sub connect_to_db
 {      # Database Information
        my $db="honeypot_scanner";
        my $userid="root";
        my $passwd="rootpass";
        my $connectionInfo="dbi:mysql:$db";

        # Make Connection to Database
        my $dbh = DBI->connect($connectionInfo,$userid,$passwd, {
    RaiseError => 1,
    AutoCommit => 0
    } ) || die "Database connection not made: $DBI::errstr";
        return( $dbh);
 }

sub db_threads
```

```
{       my $dbh = shift;

        my $thr_name = "tcpdump_scan";
        my $thr_pid = $$;
        my $last_tstamp = date_time();

        my $query = "update threads set thr_pid = '$thr_pid', last_tstamp = '$last_tstamp'
where thr_name = '$thr_name';";
        my $update_stmt = $dbh->prepare($query);
        $update_stmt->execute();
}

sub db_insert_tcpdump_icmp
{       my $di_insert_tcpdump_icmp = shift;
        my $di_ip_addr = shift;
        my $di_mac_addr = shift;
        my $di_icmp_reply = shift;
        my $di_dbh = shift;
        my $di_tcpfile = shift;
        my $di_input_tcp = shift;

        # Prepare and Execute DB Query to see if information has already been added
        my $query = "select date_created, icmp_reply from tcpdump_icmp where ip_addr
= '$di_ip_addr' AND mac_addr = '$di_mac_addr' AND icmp_reply = '$di_icmp_reply';";
        $select_stmt = $di_dbh->prepare($query);
        $select_stmt->execute();
        $select_stmt->bind_columns(\my $dc, \my $r);
        while ($select_stmt->fetch()) {
         $reply = $r;
        }
        $select_stmt->finish;
        if ($reply eq $di_icmp_reply)
        {
                my $last_tstamp = date_time();
                my $time = time();
                my $query = "update tcpdump_icmp set last_tstamp = '$last_tstamp',
last_tstamp_time = '$time' where ip_addr = '$di_ip_addr' AND mac_addr =
'$di_mac_addr' AND icmp_reply = '$di_icmp_reply';";
                my $update_stmt = $di_dbh->prepare($query);
                $update_stmt->execute();
        }else{

print $di_input_tcp;
print $di_tcpfile "$di_input_tcp";
```

```
                # finalize the insert statement
                $di_insert_tcpdump_icmp->bind_param( 1, $di_ip_addr);  # refers to the
first ? in the query
                $di_insert_tcpdump_icmp->bind_param( 2, uc($di_mac_addr)); # refers to
the second ? in the query
                $di_insert_tcpdump_icmp->bind_param( 3, $di_icmp_reply); # refers to
the third ? in the query
                my $date_created = date_time();
                $di_insert_tcpdump_icmp->bind_param( 4, $date_created); # refers to the
fourth ? in the query
                $di_insert_tcpdump_icmp->bind_param( 5, $date_created); # refers to the
fifth ? in the query
                my $time = time();
                $di_insert_tcpdump_icmp->bind_param( 6, $time);          # refers to the
sixth ? in the query

                $di_insert_tcpdump_icmp->execute();        # excute the SQL statement
        }
}

sub db_query_services
{       my $di_insert_tcpdump_ports = shift;
        my $di_ip_addr = shift;
        my $di_mac_addr = shift;
        my $di_port_num = shift;
        my $di_protocol = shift;
        my $di_syn_ack = shift;
        my $di_rst_ack = shift;
        my $di_udp_reply = shift;
        my $di_dbh = shift;
        my $di_tcpfile = shift;
        my $di_input_tcp = shift;

        my $service='';
        my $extra_info='';

        # Prepare and Execute DB Query
        my $query = "select service, extra_info from nmap_services where port_num =
'$di_port_num' AND protocol = '$di_protocol';";
        $select_stmt = $di_dbh->prepare($query);
        $select_stmt->execute();
        $select_stmt->bind_columns(\my $s, \my $ei);
        while ($select_stmt->fetch()) {
```

```
        $service = $s;
        $extra_info = $ei;
        }
        $select_stmt->finish;

        if ($service ne ''){
                if ($extra_info eq ''){$extra_info = 'unknown';}
                # Prepare and Execute DB Query to see if information has already been
added
                my $query = "select date_created, port_num from tcpdump_ports where
ip_addr = '$di_ip_addr' AND mac_addr = '$di_mac_addr' AND port_num =
'$di_port_num' AND syn_ack = '$di_syn_ack' AND rst_ack = '$di_rst_ack' AND
udp_reply = '$di_udp_reply';";
                $select_stmt = $di_dbh->prepare($query);
                $select_stmt->execute();
                $select_stmt->bind_columns(\my $dc, \my $p);
                while ($select_stmt->fetch()) {
                 $port = $p;
                }
                $select_stmt->finish;
                if ($port == $di_port_num)
                {
                        my $last_tstamp = date_time();
                        my $time = time();
                        my $query = "update tcpdump_ports set last_tstamp =
'$last_tstamp', last_tstamp_time = '$time' where ip_addr = '$di_ip_addr' AND mac_addr
= '$di_mac_addr' AND port_num = '$di_port_num' AND syn_ack = '$di_syn_ack' AND
rst_ack = '$di_rst_ack' AND udp_reply = '$di_udp_reply';";
                        my $update_stmt = $di_dbh->prepare($query);
                        $update_stmt->execute();
                }else{

print $di_input_tcp;
print "The ip_addr: $di_ip_addr, protocol: $di_protocol, port_num: $di_port_num,
service: $service\n";
print $di_tcpfile $di_input_tcp;
print $di_tcpfile "The ip_addr: $di_ip_addr, protocol: $di_protocol, port_num:
$di_port_num, service: $service\n";

                        # finalize the insert statement
                        $di_insert_tcpdump_ports->bind_param( 1, $di_ip_addr); # refers
to the first ? in the query
                        $di_insert_tcpdump_ports->bind_param( 2, uc($di_mac_addr)); #
refers to the second ? in the query
```

```
                        $di_insert_tcpdump_ports->bind_param( 3, $di_port_num); #
refers to the third ? in the query
                        $di_insert_tcpdump_ports->bind_param( 4, $di_protocol); # refers
to the fourth ? in the query
                        $di_insert_tcpdump_ports->bind_param( 5, $service);      #
refers to the fifth ? in the query
                        $di_insert_tcpdump_ports->bind_param( 6, $extra_info);   #
refers to the sixth ? in the query
                        $di_insert_tcpdump_ports->bind_param( 7, $di_syn_ack);  #
refers to the seventh ? in the query
                        $di_insert_tcpdump_ports->bind_param( 8, $di_rst_ack);   #
refers to the eighth ? in the query
                        $di_insert_tcpdump_ports->bind_param( 9, $di_udp_reply); #
refers to the ninth ? in the query
                        my $date_created = date_time();
                        $di_insert_tcpdump_ports->bind_param( 10, $date_created); #
refers to the tenth ? in the query
                        $di_insert_tcpdump_ports->bind_param( 11, $date_created); #
refers to the eleventh ? in the query
                        my $time = time();
                        $di_insert_tcpdump_ports->bind_param( 12, $time);
        # refers to the twelfth ? in the query

                        $di_insert_tcpdump_ports->execute();        # excute the SQL
statement
                }
        }
}

sub db_active_scan_queue
{       my $da_insert_queue = shift;
        my $da_ip_addr = shift;
        my $da_mac_addr = shift;
        my $da_dbh = shift;

        my $scan_id;

        # Prepare and Execute DB Query
        my $query = "select scan_id, ip_addr from scan_queue where ip_addr =
'$da_ip_addr' AND mac_addr = '$da_mac_addr';";
        $select_stmt = $da_dbh->prepare($query);
        $select_stmt->execute();
        $select_stmt->bind_columns(\my $s_id, \my $ip);
        while ($select_stmt->fetch()) {
```

```
        $scan_id = $s_id;
        }
        $select_stmt->finish;

        if ($scan_id != '') {return;}
        else{
                # finalize the insert statement
                $da_insert_queue->bind_param( 1, $da_ip_addr);    # refers to the first ?
in the query
                $da_insert_queue->bind_param( 2, $da_mac_addr);          # refers to the
second ? in the query
                my $date_created = date_time();
                $da_insert_queue->bind_param( 3, $date_created);          # refers to the
third ? in the query
                my $time = time();
                $da_insert_queue->bind_param( 4, $time);                  # refers to the
fourth ? in the query

                $da_insert_queue->execute();                  # excute the SQL statement
        }
}

sub date_time
{       my @months = qw(Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec);
        my @weekDays = qw(Sun Mon Tue Wed Thu Fri Sat Sun);
        (my $second, my $minute, my $hour, my $dayOfMonth, my $month, my
$yearOffset, my $dayOfWeek, my $dayOfYear, my $daylightSavings) = localtime();
        my $year = 1900 + $yearOffset;
        my $fixed_month = 1 + $month;
        #my $theTime = "$weekDays[$dayOfWeek] $months[$month] $dayOfMonth
$hour:$minute:$second $year";
        my $theTime = "$year-$fixed_month-$dayOfMonth $hour:$minute:$second";
}
```

Active_scanner.pl

```
#!/usr/bin/perl
#active_scanner.pl by Chris Hecker, 2008

use strict;
use Getopt::Long;
use vars qw(%G);
use DBI;

#Global Variables
my $ip_addr, my $mac_addr, my $select_stmt, my $noise, my $hs_time, my $nmap_exe,
my $nmap_xml, my $xprobe2_xml, my $scan_id;

#n, nexe, xxml and t are coming from honeypot_scanner.pl
GetOptions( 'n=s'       => \$G{n},    #       Noise Level selector
                                      #
        'nexe=s'        => \$G{nexe},#        Nmap.exe path
                                      #
        'nxml=s'        => \$G{nxml},         #       Nmap output XML path
                                      #
        'xxml=s'        => \$G{xxml},         #       Xprobe2 output XML path
                                      #
        't=s'   => \$G{t});    #       Time between active re-scans of known machines
(seconds)       #

if($G{n} eq 'low' || $G{n} eq 'medium' || $G{n} eq 'medium-high' || $G{n} eq 'high')
{
        $noise = $G{n};
}else {print "A noise level must be selected, please use the following options \"-n (low ||
medium || medium-high || high)\"\n"; exit;}

if($G{nexe} eq '' || $G{nxml} eq '' || $G{xxml} eq '')
{       print "Please use the following options \"-nexe /nmap.exe path -nxml /nmap
output.xml path -xxml /xprobe2 output.xml path for active scanning programs\"\n";
        exit();
}else {$nmap_exe = $G{nexe}; $nmap_xml = $G{nxml}; $xprobe2_xml = $G{xxml};}

if($G{t} eq '')
{       print "Please use the following option \"-t (time in seconds between re-scanning
machines)\"\n";
        exit();
```

```
}else {$hs_time = $G{t};}

print "Active_Scanner - active scanning utility\n";

# Connect to Database
my $dbh= connect_to_db();

db_threads($dbh);

while (1){

        db_active_scan_queue($dbh);

        if ($ip_addr != '')
        {
                if ($noise eq 'low'){
                        xprobe2_scan1($ip_addr, $mac_addr);
                }
                elsif ($noise eq 'medium'){
                        xprobe2_scan2($ip_addr, $mac_addr);
                }
                elsif ($noise eq 'medium-high'){
                        nmap_scan($ip_addr, $mac_addr);
                }
                elsif ($noise eq 'high'){
                        nmap_scan($ip_addr, $mac_addr);
                        xprobe2_scan2($ip_addr, $mac_addr);
                }
        }
}

# Disconnect from Database
$dbh->disconnect();
print "Active_scan disconnected from Database.\n";

exit;

# connect to the database
sub connect_to_db
 {      # Database Information
        my $db="honeypot_scanner";
        my $userid="root";
        my $passwd="rootpass";
        my $connectionInfo="dbi:mysql:$db";
```

```
        # Make Connection to Database
        my $dbh = DBI->connect($connectionInfo,$userid,$passwd, {
    RaiseError => 1,
    AutoCommit => 0
    } ) || die "Database connection not made: $DBI::errstr";
        return( $dbh);
}

sub db_threads
{       my $dbh = shift;

        my $thr_name = "active_scan";
        my $thr_pid = $$;
        my $last_tstamp = date_time();

        my $query = "update threads set thr_pid = '$thr_pid', last_tstamp = '$last_tstamp'
where thr_name = '$thr_name';";
        my $update_stmt = $dbh->prepare($query);
        $update_stmt->execute();
}

sub db_active_scan_queue
{       my $dbh = shift;

        # Prepare and Execute DB Query
        my $query = "select scan_id, ip_addr, mac_addr from scan_queue limit 1;";
        $select_stmt = $dbh->prepare($query);
        $select_stmt->execute();
        $select_stmt->bind_columns(\my $s_id, \my $ip, \my $mac);
        while ($select_stmt->fetch()) {
                $scan_id = $s_id;
                $ip_addr = $ip;
                $mac_addr = $mac;
        }
        $select_stmt->finish;

        my $query = "delete from scan_queue where scan_id = '$scan_id';";
        my $delete_stmt = $dbh->prepare($query);
        $delete_stmt->execute();
}

sub xprobe2_scan1
{       my $ip_addr = shift;
```

```
        my $mac_addr = shift;
        system "perl xprobe2_mysql.pl -mip $ip_addr -mac $mac_addr -t $hs_time -o
$xprobe2_xml -ip $ip_addr";
 }

sub xprobe2_scan2
 {      my $ip_addr = shift;
        my $mac_addr = shift;
        system "perl xprobe2_mysql.pl -mip $ip_addr -mac $mac_addr -t $hs_time -o
$xprobe2_xml -p 1 -ip $ip_addr";
 }

sub nmap_scan
 {      my $ip_addr = shift;
        my $mac_addr = shift;
        system "perl nmap_mysql.pl -mip $ip_addr -mac $mac_addr -t $hs_time -o
$nmap_xml -nmap $nmap_exe -ip $ip_addr";
 }

sub date_time
 {      my @months = qw(Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec);
        my @weekDays = qw(Sun Mon Tue Wed Thu Fri Sat Sun);
        (my $second, my $minute, my $hour, my $dayOfMonth, my $month, my
$yearOffset, my $dayOfWeek, my $dayOfYear, my $daylightSavings) = localtime();
        my $year = 1900 + $yearOffset;
        my $fixed_month = 1 + $month;
        #my $theTime = "$weekDays[$dayOfWeek] $months[$month] $dayOfMonth
$hour:$minute:$second $year";
        my $theTime = "$year-$fixed_month-$dayOfMonth $hour:$minute:$second";
 }
```

Nmap_mysql.pl

```
#!/usr/bin/perl
#nmap_mysql.pl by Chris Hecker, 2007
#modified scan.pl written by Anthony G. Persaud

#This version currently works with nmap v5.51

use strict;
use Getopt::Long;
use vars qw(%G);
use DBI;
use XML::Twig;

#Global Variables
my $select_stmt, my $state, my $machine_id=0, my $as_ip_addr, my $as_mac_addr, my
$hs_time;

#t, mip and mac are coming from active_scanner.pl but are not necessary
GetOptions(    'nmap=s' => \$G{nmap},      #       Nmap.exe path
                                                    #
                        'o=s'    => \$G{o},     #       Nmap output XML path
                                                    #
                        'ip=s'   => \$G{ip},    #       IP address that Nmap will be
scanning                                            #
                        't=s'    => \$G{t},     #       Time between active re-scans of
known machines (seconds)    #
                        'mip=s' => \$G{mip}, #       IP address passed from active
scanner (scan_queue)        #
                        'mac=s' => \$G{mac});       #       MAC address passed from
active scanner (scan_queue)         #

if($G{nmap} eq '' || $G{o} eq '' || $G{ip} eq '')
        {print "Please use the following options \"-nmap /nmap.exe path -o /nmap.xml
(path) -ip XXX.XXX.XXX.XXX\"\n"; exit(0);}

if($G{mip} eq '' || $G{mac} eq '' || $G{t} eq ''){$as_ip_addr = "unknown";}
else {$as_ip_addr = $G{mip}; $as_mac_addr = $G{mac}; $hs_time = $G{t};}

my $dbh= connect_to_db();                                   # Connect to
Database
```

```
if ($as_ip_addr ne 'unknown'){db_query_machines($as_ip_addr, $as_mac_addr,
$hs_time, $dbh);}

print "\nNmap_mysql.pl - (Nmap active scanner)\n",('-'x80),"\n\n";
print "Using Nmap exe (".$G{nmap}.") for TCP/UDP scans on ".$G{ip}." then inserts
data into mysql database.\n\n";

# prepare the insert statement just once, the actual values will replace the ? later
my $insert_machines= $dbh->prepare( "INSERT INTO nmap_machines (ip_addr, state,
mac_addr, mac_vendor, primary_os, distance_hops, date_created, last_tstamp,
last_tstamp_time) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?);");
my $insert_machines_down= $dbh->prepare( "INSERT INTO nmap_machines (ip_addr,
state, mac_addr, date_created, last_tstamp, last_tstamp_time) VALUES (?, ?, ?, ?, ?,
?);");
my $insert_ports= $dbh->prepare( "INSERT INTO nmap_ports (machine_id, port_num,
protocol, service, product) VALUES (?, ?, ?, ?, ?);");

system "nmap -sT -sU -T4 -sV -O -oX $G{o} --host-timeout 180 $G{ip}";
print "\n";

my $twig_0= new XML::Twig( twig_handlers => { nmaprun =>
\&insert_row_machines1});
$twig_0->parsefile($G{o});
$twig_0->purge;                                                    # delete the
twig so far

my $twig_1= new XML::Twig( twig_handlers => { host => \&insert_row_machines2});
$twig_1->parsefile($G{o});
$twig_1->purge;                                                    # delete the
twig so far

my $twig_2= new XML::Twig( twig_handlers => { port => \&insert_row_ports});
$twig_2->parsefile($G{o});
$twig_2->purge;                                                    # delete the
twig so far

$dbh->disconnect();                                                # Disconnect
from Database
exit;

# connect to the database
sub connect_to_db
 {      # Database Information
        my $db="honeypot_scanner";
```

```
        my $userid="root";
        my $passwd="rootpass";
        my $connectionInfo="dbi:mysql:$db";

        # Make Connection to Database
        my $dbh = DBI->connect($connectionInfo,$userid,$passwd, {
    RaiseError => 1,
    AutoCommit => 0
    } ) || die "Database connection not made: $DBI::errstr";
        return( $dbh);
}

sub insert_row_machines1
{       my ($twig_0, $ename)= @_;
        if ($ename->descendants('host') == 0)
        {
                my $ip_addr= $ename->first_child('address')->att('addr');
                $state= $ename->first_child('status')->att('state');
                my $mac_addr= $ename->first_child('address')->next_sibling('address')-
>att('addr');
                if ($mac_addr == 0){
                        my $mac_pattern = '(([0-9a-fA-F][0-9a-fA-F]:){5}([0-9a-fA-F][0-
9a-fA-F]))';
                        my @arping = `arping -f $ip_addr>&1`;
                        my $line = @arping[1];
                        $line =~ /$mac_pattern/; chop $line;
                        $mac_addr = $1;
                }
                my $date_created = date_time();
                my $time = time();

                if($state eq 'up'){
                        # finalize the insert statement
                        $insert_machines_down->bind_param( 1, $ip_addr);        #
refers to the first ? in the query
                        $insert_machines_down->bind_param( 2, $state);          #
refers to the second ? in the query
                        $insert_machines_down->bind_param( 3, $mac_addr);       #
refers to the third ? in the query
                        $insert_machines_down->bind_param( 4, $date_created);   #
refers to the fourth ? in the query
                        $insert_machines_down->bind_param( 5, $date_created);   #
refers to the fifth ? in the query
```

```
                $insert_machines_down->bind_param( 6, $time);          #
refers to the sixth ? in the query

                $insert_machines_down->execute();          # excute the SQL
statement

                $twig_0->purge;                            # will not delete the
parent
                $dbh->disconnect();
            # Disconnect from Database
                exit;
            }
        }
}

sub insert_row_machines2
{       my ($twig_1, $ename)= @_;
        my $ip_addr= $ename->first_child('status')->next_sibling('address')->att('addr');
        $state= $ename->first_child('status')->att('state');
        my $mac_addr= $ename->first_child('address')->next_sibling('address')-
>att('addr');
        if ($mac_addr == 0){
            my $mac_pattern = '(([0-9a-fA-F][0-9a-fA-F]:){5}([0-9a-fA-F][0-9a-fA-
F]))';
            my @arping = `arping -f $ip_addr>&1`;
            my $line = @arping[1];
            $line =~ /$mac_pattern/; chop $line;
            $mac_addr = $1;
        }
        my $date_created = date_time();
        my $time = time();
        my $mac_vendor, my $distance, my $primary_os;

        if($state eq 'up'){
            if ($ename->descendants('address') == 0){$primary_os = 'unknown';}
            else {$mac_vendor = $ename->first_child('address')-
>next_sibling('address')->att('vendor');}
            if ($ename->descendants('osmatch') == 0){$primary_os = 'unknown';}
            else {$primary_os = $ename->first_child('os')->first_child('osmatch')-
>att('name');}
            if ($ename->descendants('distance') == 0){$distance = 'unknown';}
            else {$distance = $ename->first_child('distance')->att('value');}

            $machine_id = 0;
```

```
                # finalize the insert statement
                $insert_machines->bind_param( 1, $ip_addr);             # refers to the
first ? in the query
                $insert_machines->bind_param( 2, $state);           # refers to the second
? in the query
                $insert_machines->bind_param( 3, $mac_addr);            # refers to the
third ? in the query
                $insert_machines->bind_param( 4, $mac_vendor);          # refers to the
fourth ? in the query
                $insert_machines->bind_param( 5, $primary_os);          # refers to the
fifth ? in the query
                $insert_machines->bind_param( 6, $distance);                    #
refers to the sixth ? in the query
                $insert_machines->bind_param( 7, $date_created);        # refers to the
seventh ? in the query
                $insert_machines->bind_param( 8, $date_created);        # refers to the
eigth ? in the query
                $insert_machines->bind_param( 9, $time);                # refers to the
ninth ? in the query

                $insert_machines->execute();                        # excute the SQL
statement

                # Prepare and Execute DB Query
                my $query = "select machine_id, state from nmap_machines where
ip_addr = '$ip_addr' AND mac_addr = '$mac_addr' AND last_tstamp_time = '$time';";
                $select_stmt = $dbh->prepare($query);
                $select_stmt->execute();
                $select_stmt->bind_columns(\my $m_id, \my $st);
                while ($select_stmt->fetch()) {
                        $machine_id = $m_id;
                }
                $select_stmt->finish;
        } else {

                # finalize the insert statement
                $insert_machines_down->bind_param( 1, $ip_addr);        # refers to the
first ? in the query
                $insert_machines_down->bind_param( 2, $state);          # refers to the
second ? in the query
                $insert_machines_down->bind_param( 3, $mac_addr);       # refers to the
third ? in the query
```

```
		$insert_machines_down->bind_param( 4, $date_created);	# refers to the
fourth ? in the query
		$insert_machines_down->bind_param( 5, $date_created);	# refers to the
fifth ? in the query
		$insert_machines_down->bind_param( 6, $time);		# refers to the
sixth ? in the query

		$insert_machines_down->execute();		# excute the SQL statement

		$twig_1->purge;					# will not delete the parent
		$dbh->disconnect();
	# Disconnect from Database
		exit;
	}
}

sub insert_row_ports
{	my( $twig_2, $ename)= @_;

	my $protocol= $ename->att('protocol');
	my $port_num= $ename->att('portid');
	my $service= $ename->first_child('service')->att('name');
	my $product= $ename->first_child('service')->att('product');
	if ($product eq ''){$product = 'unknown';}

	# Prepare and Execute DB Query
	my $query = "select port_num from nmap_ports where machine_id =
'$machine_id' AND port_num = '$port_num' AND protocol = '$protocol';";
	$select_stmt = $dbh->prepare($query);
	$select_stmt->execute();
	my $p_n = $select_stmt->fetch();
	if ($p_n == 0){
		# finalize the insert statement
		$insert_ports->bind_param( 1, $machine_id);			#
refers to the first ? in the query
		$insert_ports->bind_param( 2, $port_num);		# refers to the
second ? in the query
		$insert_ports->bind_param( 3, $protocol);		# refers to the
third ? in the query
		$insert_ports->bind_param( 4, $service);		# refers to the
fourth ? in the query
		$insert_ports->bind_param( 5, $product);		# refers to the
fifth ? in the query
```

```
                $insert_ports->execute();                    # excute the SQL statement

                my $last_tstamp = date_time();
                my $time = time();
                my $query = "update nmap_machines set last_tstamp = '$last_tstamp',
last_tstamp_time = '$time' where machine_id = '$machine_id';";
                my $update_stmt = $dbh->prepare($query);
                $update_stmt->execute();
        }
        $select_stmt->fmish;
}

sub db_query_machines
{       my $as_ip_addr = shift;
        my $as_mac_addr = shift;
        my $hs_time = shift;
        my $dbh = shift;
        my $last_tstamp_time;
        my $current_time = time();

        # Prepare and Execute DB Query
        my $query = "select machine_id, last_tstamp_time from nmap_machines where
ip_addr = '$as_ip_addr' AND mac_addr = '$as_mac_addr' order by last_tstamp_time desc
limit 1;";
        $select_stmt = $dbh->prepare($query);
        $select_stmt->execute();
        $select_stmt->bind_columns(\my $m_id, \my $ts);
        while ($select_stmt->fetch()) {
                $machine_id = $m_id;
                $last_tstamp_time = $ts;
        }
        $select_stmt->finish;

        my $difference = $current_time - $last_tstamp_time;

        if ($machine_id == 0){return;}
        else {
                if ($difference < $hs_time){exit;}
        }
}

sub date_time
{       my @months = qw(Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec);
        my @weekDays = qw(Sun Mon Tue Wed Thu Fri Sat Sun);
```

```
        (my $second, my $minute, my $hour, my $dayOfMonth, my $month, my
$yearOffset, my $dayOfWeek, my $dayOfYear, my $daylightSavings) = localtime();
        my $year = 1900 + $yearOffset;
        my $fixed_month = 1 + $month;
        #my $theTime = "$weekDays[$dayOfWeek] $months[$month] $dayOfMonth
$hour:$minute:$second $year";
        my $theTime = "$year-$fixed_month-$dayOfMonth $hour:$minute:$second";
}
```

Xprobe2_mysql.pl

```
#!/usr/bin/perl
#xprobe2_mysql.pl by Chris Hecker, 2007

#This version currently works with Xprobe2 v.0.3

use strict;
use Getopt::Long;
use vars qw(%G);
use DBI;
use XML::Twig;

#Global Variables
my $select_stmt, my $machine_id=0, my $state, my $as_ip_addr, my $as_mac_addr, my
$hs_time;

#p, t, mip and mac are coming from active_scanner.pl but are not necessary
GetOptions( 'ip=s'    => \$G{ip},    #       IP address that Xprobe2 will be scanning
                      #
                'o=s'         => \$G{o},    #       Xprobe2 output XML path
                                            #
        'p=s'         => \$G{p},    #       Increased scanning flag (1 = enabled)
                              #
        't=s'         => \$G{t},    #       Time between active re-scans of known
machines (seconds)    #
        'mip=s'       => \$G{mip}, #       IP address passed from active scanner
(scan_queue)          #
        'mac=s'       => \$G{mac});        #       MAC address passed from active
scanner (scan_queue)          #

if($G{ip} eq '' || $G{o} eq ''){print "Please use the following options \"-o /xprobe2.xml
(path) -ip XXX.XXX.XXX.XXX\"\n"; exit(0);}

if($G{mip} eq '' || $G{mac} eq '' || $G{t} eq ''){$as_ip_addr = "unknown";}
else {$as_ip_addr = $G{mip}; $as_mac_addr = $G{mac}; $hs_time = $G{t};}

my $dbh= connect_to_db();
                # Connect to Database

if ($as_ip_addr ne 'unknown'){db_query_machines($as_ip_addr, $as_mac_addr,
$hs_time, $dbh);}
```

```
if($G{p} == 1){
        system "xprobe2 -r -m 2 -o $G{o} -X -T 1-1024,3306 -U 1-1024 $G{ip}";
        # with port scannning
}else{
        system "xprobe2 -r -m 2 -o $G{o} -X $G{ip}";
        # without port scannning
}

print "\nXprobe2_mysql.pl - (Xprobe2 active scanner)\n",('-'x80),"\n\n";
print "Using Xprobe2 exe for actively scanning ".$G{ip}." then inserts data into mysql
database.\n\n";

# prepare the insert statement just once, the actual values will replace the ? later
my $insert_machines= $dbh->prepare( "INSERT INTO xprobe2_machines (ip_addr,
state, mac_addr, real_time_target_sec, primary_os, secondary_os, date_created,
last_tstamp, last_tstamp_time) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?);");
my $insert_machines_down= $dbh->prepare( "INSERT INTO xprobe2_machines
(ip_addr, state, mac_addr, date_created, last_tstamp, last_tstamp_time) VALUES (?, ?, ?,
?, ?, ?);");
my $insert_ports= $dbh->prepare( "INSERT INTO xprobe2_ports (machine_id,
port_num, protocol, service, port_state) VALUES (?, ?, ?, ?, ?);");

my $twig_1= new XML::Twig( twig_handlers => { Xprobe2 =>
\&insert_row_machines});
$twig_1->parsefile($G{o});
$twig_1->purge;
                        # delete the twig so far

if($G{p} == 1){
        my $twig_2= new XML::Twig( twig_handlers => { port =>
\&insert_row_ports});
        $twig_2->parsefile($G{o});
        $twig_2->purge;
                        # delete the twig so far
}

$dbh->disconnect();
                        # Disconnect from Database
exit;

# connect to the database
sub connect_to_db
 {      # Database Information
```

```
        my $db="honeypot_scanner";
        my $userid="root";
        my $passwd="rootpass";
        my $connectionInfo="dbi:mysql:$db";

        # Make Connection to Database
        my $dbh = DBI->connect($connectionInfo,$userid,$passwd, {
    RaiseError => 1,
    AutoCommit => 0
    } ) || die "Database connection not made: $DBI::errstr";
        return( $dbh);
}

sub insert_row_machines
{       my( $twig_1, $ename)= @_;
        my $ip_addr= $ename->first_child('target')->att('ip');

        $state= $ename->first_child('target')->first_child('reachability')-
>first_child('state')->att('state');
        my $mac_pattern = '(([0-9a-fA-F][0-9a-fA-F]:){5}([0-9a-fA-F][0-9a-fA-F]))';
        my @arping = `arping -f $ip_addr>&1`;
        my $line = @arping[1];
        $line =~ /$mac_pattern/; chop $line;
        my $mac_addr = $1;
        my $date_created = date_time();
        my $time = time();
        my $rtt, my $primary_os, my $secondary_os;
        if ($state eq "up")
        {
                $rtt= $ename->first_child('target')->first_child('reachability')-
>first_child('state')->next_sibling('rtt')->att('real');
          chop $rtt; $rtt= substr($rtt, 1);
          $primary_os= $ename->first_child('target')->first_child('reachability')-
>next_sibling('os_guess')->first_child('primary')->text;
          chop $primary_os; chop $primary_os; $primary_os= substr($primary_os, 2);
                if ($primary_os eq ''){$primary_os = 'unknown';}
                $secondary_os= $ename->first_child('target')->first_child('reachability')
                        ->next_sibling('os_guess')->first_child('primary')-
>next_sibling('secondary')->text;
          chop $secondary_os; chop $secondary_os; $secondary_os=
substr($secondary_os, 2);
                if ($secondary_os eq ''){$secondary_os = 'unknown';}

                $machine_id = 0;
```

```
                # finalize the insert statement
                $insert_machines->bind_param( 1, $ip_addr);             # refers to the
first ? in the query
                $insert_machines->bind_param( 2, $state);               # refers to the
second ? in the query`
                $insert_machines->bind_param( 3, $mac_addr);            # refers to the
third ? in the query
                $insert_machines->bind_param( 4, $rtt);                 # refers to the
fourth ? in the query
                $insert_machines->bind_param( 5, $primary_os);          # refers to the
fifth ? in the query
                $insert_machines->bind_param( 6, $secondary_os);                #
refers to the sixth ? in the query
                $insert_machines->bind_param( 7, $date_created);        # refers to the
seventh ? in the query
                $insert_machines->bind_param( 8, $date_created);        # refers to the
eighth ? in the query
                $insert_machines->bind_param( 9, $time);                # refers to the
ninth ? in the query

                $insert_machines->execute();            # excute the SQL statement

                $twig_1->purge;                         # will not delete the parent

                # Prepare and Execute DB Query
                my $query = "select machine_id, state from xprobe2_machines where
ip_addr = '$ip_addr' AND mac_addr = '$mac_addr' AND last_tstamp_time = '$time';";
                $select_stmt = $dbh->prepare($query);
                $select_stmt->execute();
                $select_stmt->bind_columns(\my $m_id, \my $s);
                while ($select_stmt->fetch()) {
                        $machine_id = $m_id;
                }
                $select_stmt->finish;

        } else {

                # finalize the insert statement
                $insert_machines_down->bind_param( 1, $ip_addr);        # refers to the
first ? in the query
                $insert_machines_down->bind_param( 2, $state);          # refers to the
second ? in the query`
```

```
                $insert_machines_down->bind_param( 3, $mac_addr);       # refers to the
third ? in the query
                $insert_machines_down->bind_param( 4, $date_created);   # refers to the
fourth ? in the query
                $insert_machines_down->bind_param( 5, $date_created);   # refers to the
fifth ? in the query
                $insert_machines_down->bind_param( 6, $time);           # refers to the
sixth ? in the query

                $insert_machines_down->execute();           # excute the SQL statement

                $twig_1->purge;                             # will not delete the parent
                $dbh->disconnect();
        # Disconnect from Database
                exit;
        }
}

sub insert_row_ports
{       my( $twig_2, $ename)= @_;
        my $port_num= $ename->att('number');
 my $protocol= $ename->att('proto');
        my $service= $ename->att('service');
        my $port_state= $ename->att('state');

        if ($port_state eq 'open')
        {
                # Prepare and Execute DB Query
                my $query = "select port_num, port_state from xprobe2_ports where
machine_id = '$machine_id' AND port_num = '$port_num' AND protocol = '$protocol';";
                $select_stmt = $dbh->prepare($query);
                $select_stmt->execute();
                $select_stmt->bind_columns(\my $p_n, \my $p_st);
                $select_stmt->fetch();
                if ($p_n == 0){
                        # finalize the insert statement
                        $insert_ports->bind_param( 1, $machine_id);         # refers to the
first ? in the query
                        $insert_ports->bind_param( 2, $port_num);  # refers to the second
? in the query
                        $insert_ports->bind_param( 3, $protocol);    # refers to the third ?
in the query
                        $insert_ports->bind_param( 4, $service);     # refers to the fourth ?
in the query
```

```
                        $insert_ports->bind_param( 5, $port_state);  # refers to the fifth ?
in the query

                        $insert_ports->execute();       # excute the SQL statement

                        $twig_2->purge;                 # will not delete the parent

                        my $last_tstamp = date_time();
                        my $time = time();
                        my $query = "update xprobe2_machines set last_tstamp =
'$last_tstamp', last_tstamp_time = '$time' where machine_id = '$machine_id';";
                        my $update_stmt = $dbh->prepare($query);
                        $update_stmt->execute();
                }
                $select_stmt->finish;
        }
}

sub db_query_machines
{       my $as_ip_addr = shift;
        my $as_mac_addr = shift;
        my $hs_time = shift;
        my $dbh = shift;
        my $last_tstamp_time;
        my $current_time = time();

        # Prepare and Execute DB Query
        my $query = "select machine_id, last_tstamp_time from xprobe2_machines where
ip_addr = '$as_ip_addr' AND mac_addr = '$as_mac_addr' order by last_tstamp_time desc
limit 1;";
        $select_stmt = $dbh->prepare($query);
        $select_stmt->execute();
        $select_stmt->bind_columns(\my $m_id, \my $ts);
        while ($select_stmt->fetch()) {
                $machine_id = $m_id;
                $last_tstamp_time = $ts;
        }
        $select_stmt->finish;

        my $difference = $current_time - $last_tstamp_time;

        if ($machine_id == 0){return;}
        else {
                if ($difference < $hs_time){exit;}
```

```
        }
}

sub date_time
{       my @months = qw(Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec);
        my @weekDays = qw(Sun Mon Tue Wed Thu Fri Sat Sun);
        (my $second, my $minute, my $hour, my $dayOfMonth, my $month, my
$yearOffset, my $dayOfWeek, my $dayOfYear, my $daylightSavings) = localtime();
        my $year = 1900 + $yearOffset;
        my $fixed_month = 1 + $month;
        #my $theTime = "$weekDays[$dayOfWeek] $months[$month] $dayOfMonth
$hour:$minute:$second $year";
        my $theTime = "$year-$fixed_month-$dayOfMonth $hour:$minute:$second";
}
```

Nmap-services_mysql.pl

```
#!/usr/bin/perl
#nmap-services_mysql.pl by Chris Hecker, 2007

use strict;
use DBI;

# Connect to Database
my $dbh= connect_to_db();

# prepare the insert statement just once, the actual values will replace the ? later
my $insert_nmap_services= $dbh->prepare( "INSERT INTO nmap_services (port_num,
protocol, service, extra_info) VALUES (?, ?, ?, ?);");

#open the employeesfile
open (my $file,"nmap-services.txt") || die "Can't find file\n";

#assigns lines to array
my @lines = <$file>;

#go through each line in file
foreach my $line (@lines){

        #remove the carriage return
        chomp $line;

        #split the line between whitespace and get the different elements
        my @info = split (/\s+/, $line, 4);

        my $service = @info[0];
        my $port_num = @info[1];
        my @temp = split (/\//, $port_num);
        $port_num = @temp[0];
        my $protocol = @temp[1];
        my $extra_info = @info[3];

        insert_row($port_num, $protocol, $service, $extra_info);
}
close ($file);
```

```
# Disconnect from Database
$dbh->disconnect();

exit;

# connect to the database
sub connect_to_db
 { # Database Information
        my $db="honeypot_scanner";
        my $userid="root";
        my $passwd="rootpass";
        my $connectionInfo="dbi:mysql:$db";

        # Make Connection to Database
        my $dbh = DBI->connect($connectionInfo,$userid,$passwd, {
    RaiseError => 1,
    AutoCommit => 0
    } ) || die "Database connection not made: $DBI::errstr";
 return( $dbh);
 }


sub insert_row
 {
        my $port_num = shift;
        my $protocol = shift;
        my $service = shift;
        my $extra_info = shift;

        # finalize the insert statement
        $insert_nmap_services->bind_param( 1, $port_num); # refers to the first ? in the
query
        $insert_nmap_services->bind_param( 2, $protocol);  # refers to the second ? in
the query
        $insert_nmap_services->bind_param( 3, $service);  # refers to the third ? in the
query
        $insert_nmap_services->bind_param( 4, $extra_info);        # refers to the fourth ?
in the query

        $insert_nmap_services->execute();              # excute the SQL statement
 }
```

Appendix E: Database Statements and Schema

Database Statements

Database

// Create Database

CREATE DATABASE honeypot_scanner;

// Database control Statement

USE honeypot_scanner;

Operational Tables

// Create Table: config

CREATE TABLE config (eth_interface blob NOT NULL, mac_addr varchar(17) NOT NULL, dhcp_server blob NOT NULL, p0f_os blob NOT NULL, nmap_exe blob NOT NULL, nmap_prints blob NOT NULL, nmap_assoc blob NOT NULL, xprobe2_conf blob NOT NULL, xprobe2_xml blob NOT NULL, honeypot_xml blob NOT NULL, honeyd_config blob NOT NULL, honeyd_ip_binding varchar(2) NOT NULL, honeyd_ip_range blob, scan_ip_range blob NOT NULL, initial_deployment int NOT NULL, percent_change int NOT NULL, noise blob NOT NULL, active_scan_seconds int NOT NULL, date_created blob NOT NULL);

// Insert Information: config

INSERT into config (eth_interface, mac_addr, dhcp_server, p0f_os, nmap_exe, nmap_prints, nmap_assoc, xprobe2_conf, xprobe2_xml, honeypot_xml, honeyd_config, honeyd_ip_binding, honeyd_ip_range, scan_ip_range, initial_deployment, percent_change, noise, active_scan_seconds, date_created) values ("eth0", "00:11:22:33:44:55", "1.2.3.4", "/usr/share/honeyd/pf.os", "/root/Downloads/Honeypot/nmap-4.60/nmap", "/usr/share/honeyd/nmap.prints", "/usr/share/honeyd/nmap.assoc", "/usr/share/honeyd/xprobe2.conf", "/root/Output_files/xprobe2_output.xml", "/root/Output_files/honeypot.xml", "/root/Output_files/honeyd.conf", "il", NULL, "1.2.3.4-5", 5, 30, "low", 86400, "2008-4-16 16:04:05");

// Create Table: threads

CREATE TABLE threads (thread_id int NOT NULL auto_increment primary key, thr_name varchar(20) NOT NULL, thr_pid int NOT NULL, last_tstamp blob NOT NULL);

// Insert Information: threads

INSERT into threads (thr_name, thr_pid, last_tstamp) values ("tcpdump_scan", "0", "2008-1-9 16:04:05");

INSERT into threads (thr_name, thr_pid, last_tstamp) values ("p0f_scan", "0", "2008-1-9 16:04:05");
INSERT into threads (thr_name, thr_pid, last_tstamp) values ("active_scan", "0", "2008-1-9 16:04:05");

// Create Table: honeypot_updates

CREATE TABLE honeypot_updates (update_id int NOT NULL auto_increment primary key, num_machines int NOT NULL, num_services int NOT NULL, date_updated blob NOT NULL);

// Create Table: scan_queue

CREATE TABLE scan_queue (scan_id int NOT NULL auto_increment primary key, ip_addr varchar(15) NOT NULL, mac_addr varchar(17), date_created blob NOT NULL, last_tstamp_time int NOT NULL);

Network Scanning Tables

// Create Table: p0f

CREATE TABLE p0f (machine_id int NOT NULL auto_increment primary key, ip_addr varchar(15) NOT NULL, mac_addr varchar(17), primary_os blob, firewall varchar(64) NOT NULL, lookup_link VARCHAR(128) NOT NULL, uptime_seconds blob NOT NULL, distance_hops blob NOT NULL, date_created blob NOT NULL, last_tstamp blob NOT NULL, last_tstamp_time int NOT NULL);

// Create Table: tcpdump_ports

CREATE TABLE tcpdump_ports (tcp_id int NOT NULL auto_increment primary key, ip_addr varchar(15) NOT NULL, mac_addr varchar(17), port_num int NOT NULL, protocol varchar(15) NOT NULL, service blob NOT NULL, extra_info blob, syn_ack varchar(1), rst_ack varchar(1), udp_reply varchar(1), date_created blob NOT NULL, last_tstamp blob NOT NULL, last_tstamp_time int NOT NULL)

// Create Table: tcpdump_icmp

CREATE TABLE tcpdump_icmp (tcp_id int NOT NULL auto_increment primary key, ip_addr varchar(15) NOT NULL, mac_addr varchar(17), icmp_reply varchar(1), date_created blob NOT NULL, last_tstamp blob NOT NULL, last_tstamp_time int NOT NULL);

// Create Table: nmap_services

CREATE TABLE nmap_services (s_id int NOT NULL auto_increment primary key, port_num int NOT NULL, protocol varchar(15) NOT NULL, service blob NOT NULL, extra_info blob);

// Create Table: nmap_machines

CREATE TABLE nmap_machines (machine_id int NOT NULL auto_increment primary key, ip_addr varchar(15) NOT NULL, state varchar(5) NOT NULL, mac_addr varchar(17), mac_vendor blob, primary_os blob NOT NULL, uptime_seconds blob NOT NULL, last_reboot blob NOT NULL,

distance_hops blob NOT NULL, date_created blob NOT NULL, last_tstamp blob NOT NULL, last_tstamp_time int NOT NULL);

// Create Table: nmap_ports

CREATE TABLE nmap_ports (machine_id int NOT NULL, port_num int NOT NULL, protocol varchar(15) NOT NULL, service blob NOT NULL, version blob, product blob, extra_info blob);

// Create Table: xprobe2_machines

CREATE TABLE xprobe2_machines (machine_id int NOT NULL auto_increment primary key, ip_addr varchar(15) NOT NULL, state varchar(5) NOT NULL, mac_addr varchar(17), real_time_target_sec blob NOT NULL, primary_os blob NOT NULL, secondary_os blob NOT NULL, date_created blob NOT NULL, last_tstamp blob NOT NULL, last_tstamp_time int NOT NULL);

// Create Table: xprobe2_ports

CREATE TABLE xprobe2_ports (machine_id int NOT NULL, port_num int NOT NULL, protocol varchar(15) NOT NULL, service blob NOT NULL, port_state blob NOT NULL);

Low Interaction Honeypot Configuration Tables

// Create Table: dhcp

CREATE TABLE dhcp (dhcp_id int NOT NULL, ip_addr varchar(15) NOT NULL, mac_addr varchar(17) NOT NULL, lease_time_seconds blob NOT NULL, renewal_tstamp_time blob NOT NULL, date_created blob NOT NULL, last_tstamp blob NOT NULL, last_tstamp_time int NOT NULL);

// Create Table: honeyd_scripts

CREATE TABLE honeyd_scripts (script_id int NOT NULL auto_increment primary key, primary_os varchar(20) NOT NULL, protocol varchar(5) NOT NULL, port_num int NOT NULL, script_lang varchar(20) NOT NULL, path_and_filename blob NOT NULL);

// Insert Information: honeyd_scripts

INSERT into honeyd_scripts (primary_os, protocol, port_num, script_lang, path_and_filename) values ("windows 2000", "tcp", 23, "perl", "/home/Honeyd Scripts/telnet-emul/telnet/faketelnet.pl");

//Create Table: lih_hih_link

CREATE TABLE lih_hih_link (link_id int NOT NULL auto_increment primary key, lih_os_platform blob NOT NULL, lih_ip_addr varchar(15) NOT NULL, lih_mac_addr varchar(17) NOT NULL, hih_os_platform blob NOT NULL, hih_ip_addr varchar(15) NOT NULL, hih_mac_addr varchar(17) NOT NULL, hih_location blob NOT NULL, hih_state blob NOT NULL, date_created blob NOT NULL, last_tstamp blob NOT NULL);

High Interaction Honeypot Configuration Tables

//Create Table: vmware_template

CREATE TABLE vmware_template (hih_id int NOT NULL auto_increment primary key, os_platform text NOT NULL, location blob NOT NULL, date_created blob NOT NULL, FULLTEXT (os_platform));

//Insert Information: vmware_template

//Location starts with the datastore that is used for the VMware Server; in this case a data store called [standard]

//Linux Example

INSERT into vmware_template (os_platform, location, date_created) values ("Linux Kernel 2.6.24-19-generic", "[standard] Ubuntu8/Ubuntu8.vmx", "2008-11-07 16:04:05");

//Windows Example

INSERT into vmware_template (os_platform, location, date_created) values ("Microsoft Windows XP SP3", "[standard] Windows_XP/Windows_XP.vmx", "2008-11-09 17:04:05");

Database Schema

Operational Tables

Table 4, Operational Tables – config

Table: config		
Field names	**Field Description**	**Example Data**
eth_interface	The name of the network interface that will be used to gather the information	eth0
mac_addr	MAC address of eth_interface	00:01:02:03:04:05
dhcp_server	IP address of the network DHCP server	1.2.3.4
p0f_os	Location of the P0f fingerprinting file	/honeyd/pf.os
nmap_exe	Location of the Nmap executable	/nmap-4.60/nmap
nmap_prints	Location of the Nmap fingerprinting file	/honeyd/nmap.prints
nmap_assoc	Location of the Nmap-Xprobe2 association file	/honeyd/nmap.assoc
xprobe2_conf	Location of the Xprobe2 fingerprinting file	/honeyd/xprobe2.conf
xprobe2_xml	Location for the output of Xprobe2	/Output_files/xprobe2.xml
honeypot_xml	Location for the resulting honey.conf file	/Output_files/honeypot.xml
honeyd_config	Location for the resulting honeypot.xml file	/Output_files/honeyd.conf
honeyd_ip_binding	Indicates the resulting IP address scheme of the honeyd.conf file	iS, iD, iR, or iI
honeyd_ip_range	Used if the honeyd_ip_binding is iD or iR and indicates alternate IP subnet or range	1.2.3.0 or 1.2.3.4-7
scan_ip_range	IP address subnet that the user wishes to scan	1.0.0.0, 1.2.0.0, 1.2.3.0, or 1.2.3.4
initial_deployment	Threshold for the initial deployment of the configuration files	5
percent_change	Threshold in percent change for the re-deployment of the configuration files	30
noise	Indicates the amount of noise introduced by the network scanner	passive, low, medium, medium-high or high
active_scan_seconds	Time delay before re-scanning an identified machine (seconds)	86400
date_created	Date created the config table (format: yyyy/mm/dd hh:mm:ss)	2008/04/16 16:04:05

Table 5, Operational Tables – threads

Table: threads		
Field names	**Field Description**	**Example Data**
thread_id	Unique ID for this thread	1
thr_name	Name of the thread	tcpdump_scan
thr_pid	Process ID (PID) for the thread	5517
last_tstamp	Last timestamp of the thread's creation (format: yyyy/mm/dd hh:mm:ss)	2008/04/16 16:04:05

Table 6, Operational Tables – honeypot_updates

Table: honeypot_updates		
Field names	**Field Description**	**Example Data**
update_id	Unique ID for the update	1
num_machines	Number of total machines identified	5
num_services	Number of total services gathered	5517
date_updated	Time and Date of the last update (format: yyyy/mm/dd hh:mm:ss)	2008/04/16 16:04:05

Table 7, Operational Tables – scan_queue

Table: scan_queue		
Field names	**Field Description**	**Example Data**
scan_id	Unique ID for this device information	1
ip_addr	IP address of the identified machine	1.2.3.4
mac_addr	MAC address of the identified machine	00:01:02:03:04:05
date_created	Time and date of the machines identification (format: yyyy/mm/dd hh:mm:ss)	2008/04/16 16:04:05
last_tstamp_time	Last timestamp for seeing the identified machine UNIX time (seconds since 1-1-1970)	1208361845

Network Scanning Tables

Table 8, Network Scanning Tables – p0f

Table: p0f		
Field names	**Field Description**	**Example Data**
machine_id	Unique ID for this identified machine	1
ip_addr	IP address of the identified machine	1.2.3.4
mac_addr	MAC address of the identified machine	00:01:02:03:04:05
primary_os	Operating system guess of the identified machine	Linux Kernel 2.4.0
firewall	Firewall on the identified machine	Yes or unknown
lookup_link	Type of network hookup	ethernet
uptime_seconds	Number of seconds that the identified machine has been running	12345
distance_hops	Number of hops to the identified machine	0
date_created	Time and date of the machines identification (format: yyyy/mm/dd hh:mm:ss)	2008/04/16 16:04:05
last_tstamp	Last timestamp for seeing the identified machine (format: yyyy/mm/dd hh:mm:ss)	2008/04/16 16:04:05
last_tstamp_time	Last timestamp for seeing the identified machine UNIX time (seconds since 1-1-1970)	1208361845

Table 9, Network Scanning Tables – tcpdump_icmp

Table: tcpdump_icmp		
Field names	**Field Description**	**Example Data**
tcp_id	Unique ID for the identified packet	1
ip_addr	Destination IP address	1.2.3.4
mac_addr	Destination MAC address	00:01:02:03:04:05
icmp_reply	Was the packet an icmp-reply packet?	Y
date_created	Time and date of the first ICMP reply packet (format: yyyy/mm/dd hh:mm:ss)	2008/04/16 16:04:05
last_tstamp	Last timestamp for seeing an ICMP reply packet (format: yyyy/mm/dd hh:mm:ss)	2008/04/16 16:04:05
last_tstamp_time	Last timestamp for seeing an ICMP reply packet UNIX time (seconds since 1-1-1970)	1208361845

Table 10, Network Scanning Tables – nmap_services

Table: nmap_services		
Field names	**Field Description**	**Example Data**
s_id	Unique ID for the service	1
port_num	Port number for the service	80
protocol	Network protocol of the service	TCP
service	Service of the port number and the protocol	HTTP
extra_info	Extra information for the service	World Wide Web HTTP

Table 11, Network Scanning Tables – tcpdump_ports

Table: tcpdump_ports		
Field names	**Field Description**	**Example Data**
tcp_id	Unique ID for the identified packet	1
ip_addr	Destination IP address	1.2.3.4
mac_addr	Destination MAC address	00:01:02:03:04:05
port_num	Destination port number	80
protocol	Network protocol	TCP
service	Service of the port number and the protocol	HTTP
extra_info	Extra information for the service	World Wide Web HTTP
syn_ack	Was the packet a syn-ack packet?	Y or N
rst_ack	Was the packet a rst-ack packet?	Y or N
udp_reply	Was the packet a udp-reply packet?	Y or N
date_created	Time and date of the first packet for a particular port (format: yyyy/mm/dd hh:mm:ss)	2008/04/16 16:04:05
last_tstamp	Last timestamp for a particular port (format: yyyy/mm/dd hh:mm:ss)	2008/04/16 16:04:05
last_tstamp_time	Last timestamp for a particular port UNIX time (seconds since 1-1-1970)	1208361845

Table 12, Network Scanning Tables – nmap_machines

Table: nmap_machines		
Field names	**Field Description**	**Example Data**
machine_id	Unique ID for this identified machine	1
ip_addr	IP address of the identified machine	1.2.3.4
state	State of the identified machine	up
mac_addr	MAC address of the identified machine	00:01:02:03:04:05
mac_vendor	Vendor for the identified MAC address	Intel
primary_os	Operating system guess of the identified machine	Linux Kernel 2.4.0
uptime_seconds	Number of seconds that the identified machine has been running	12345
distance_hops	Number of hops to the identified machine	0
date_created	Time and date of the machines identification (format: yyyy/mm/dd hh:mm:ss)	2008/04/16 16:04:05
last_tstamp	Last timestamp for seeing the identified machine (format: yyyy/mm/dd hh:mm:ss)	2008/04/16 16:04:05
last_tstamp_time	Last timestamp for seeing the identified machine UNIX time (seconds since 1-1-1970)	1208361845

Table 13, Network Scanning Tables – nmap_ports

Table: nmap_ports		
Field names	**Field Description**	**Example Data**
machine_id	Unique ID for this identified machine	1
port_num	Port number for the service	80
protocol	Network protocol of the service	TCP
service	Service of the port number and the protocol	HTTP
product	Product running the service	Apache Web Server

Table 14, Network Scanning Tables – xprobe2_machines

Table: xprobe2_machines		
Field names	**Field Description**	**Example Data**
machine_id	Unique ID for this identified machine	1
ip_addr	IP address of the identified machine	1.2.3.4
state	State of the identified machine	up
mac_addr	MAC address of the identified machine	00:01:02:03:04:05
real_time_target_sec	Time in seconds to identified machine	0.05
primary_os	First operating system guess of the identified machine	Linux Kernel 2.4.0
secondary_os	Second operating system guess of the identified machine	Linux Kernel 2.6.0
date_created	Time and date of the machines identification (format: yyyy/mm/dd hh:mm:ss)	2008/04/16 16:04:05
last_tstamp	Last timestamp for seeing the identified machine (format: yyyy/mm/dd hh:mm:ss)	2008/04/16 16:04:05
last_tstamp_time	Last timestamp for seeing the identified machine UNIX time (seconds since 1-1-1970)	1208361845

Table 15, Network Scanning Tables – xprobe2_ports

Table: xprobe2_ports		
Field names	**Field Description**	**Example Data**
machine_id	Unique ID for this identified machine	1
port_num	Port number for the service	80
protocol	Network protocol of the service	TCP
service	Service of the port number and the protocol	HTTP
port_state	Version of the product running the service	open

Low Interaction Honeypot Configuration Tables

Table 16, LIH Configuration Tables – dhcp

Table: dhcp		
Field names	**Field Description**	**Example Data**
dhcp_id	Unique ID for DHCP interaction	1
ip_addr	Client IP address from DHCP server	1.2.3.4
mac_addr	Client MAC address	00:01:02:03:04:05
lease_time_seconds	Time in seconds to renew DHCP lease	86400
renewal_tstamp_time	Timestamp of the next renewal UNIX time (seconds since 1-1-1970)	1208361845
date_created	Time and date of the DHCP request (format: yyyy/mm/dd hh:mm:ss)	2008/04/16 16:04:05
last_tstamp	Timestamp of the last renewal (format: yyyy/mm/dd hh:mm:ss)	2008/04/16 16:04:05
last_tstamp_time	Timestamp of the last renewal UNIX time (seconds since 1-1-1970)	1208361845

Table 17, LIH Configuration Tables – honeyd_scripts

Table: honeyd_scripts		
Field names	**Field Description**	**Example Data**
script_id	Unique ID for the script	1
primary_os	OS for which the script is designed to emulate	Linux
protocol	Network protocol of the service	TCP
port_num	Port number for the service	80
script_lang	Language to interpret the script	Perl
path_and_filename	Location of the script	/telnet-emul/faketelnet.pl

Table 18, LIH Configuration Tables - lih_hih_link

Table: lih_hih_link		
Field names	**Field Description**	**Example Data**
link_id	Unique ID for the LIH and HIH link	1
lih_os_platform	OS platform for the LIH	Microsoft Windows XP SP2
lih_ip_addr	IP address for LIH	1.2.3.4
lih_mac_addr	MAC address for LIH	00:11:22:33:44:55
hih_os_platform	OS platform for the HIH	Microsoft Windows XP SP3
hih_ip_addr	IP address for HIH	1.2.3.5
hih_mac_addr	MAC address for HIH	00:11:22:33:44:66
hih_location	Location of HIH template; starts with the datastore that is used for the VMware Server	[standard] Windows_XP.vmx
hih_state	State of the HIH VM	ON
date_created	Time and date for creating LIH and HIH link (format: yyyy/mm/dd hh:mm:ss)	2008/04/16 16:04:05
last_tstamp	Last timestamp for deploying LIH (format: yyyy/mm/dd hh:mm:ss)	2008/04/16 16:04:05

High Interaction Honeypot Configuration Tables

Table 19, HIH Configuration Tables - vmware_template

Table: vmware_template		
Field names	**Field Description**	**Example Data**
hih_id	Unique ID for the HIH template	1
os_platform	OS platform for the HIH	Linux Kernel 2.6.24-19-generic
location	Location starts with the datastore that is used for the VMware Server	[standard] Ubuntu8/Ubuntu8.vmx
date_created	Time and date of template creation (format: yyyy/mm/dd hh:mm:ss)	2008/04/16 16:04:05

Appendix F: Publications

Hecker, C., Nance, K. & Hay, B. Dynamic Honeypot Construction. *Proceedings of the 10th Colloquium for Information Systems Security Education.* University of Maryland, University College, Adelphi, MD. June 5-8, 2006.

Hecker, C. & Hay, B. Securing E-Government Assets through Automating Deployment of Honeynets for IDS Support. *43rd Hawaii International Conference on System Sciences (HICSS)*, pp.1-10, 5-8 Jan. 2010 (Best Paper Award)

Hecker, C. Automated Honeynet Deployment for Dynamic Network Environments. *46th Hawaii International Conference on System Sciences (HICSS)*, Jan 2013. (Submitted)