Automated Honeynet Deployment for Dynamic Network Environment

Christopher Hecker
University of Alaska Fairbanks
crhecker@alaska.edu

Brian Hay
University of Alaska Fairbanks
brian.hay@alaska.edu

Abstract

One of the challenges facing information technology (IT) security professionals is the laborious task of sifting through numerous log files in an attempt to identify malicious traffic and conduct a forensics analysis to determine an appropriate course of action. This process is complicated significantly by the volume of traffic that can be associated with a production device environment. A honeynet can provide a mechanism to identify much of the forensically interesting traffic by creating a representative system to collect traffic data. However, it is challenging to maintain an accurate representation of a dynamic system in order to consistently collect the appropriate data of interest. This research effort addresses a current challenge identified by researchers at the Honeynet Project by describing a methodology for automatically creating and dynamically updating a honeynet in order to facilitate IDS support.

1. Introduction

Interconnected networks are the reality of our everyday world, with millions of small to large LANs connected via the Internet. It is the responsibility of IT security professionals to secure the perimeter of their networks to keep out adversaries who might be interested in collecting personal data or proprietary information. The US federal government, for example, has approximately a hundred agencies, well over a million employees, and millions of IT resources which contain data that must be appropriately secured, all while ensuring that it is available to all authorized users to perform the work of government. At the personal level the government manages names, addresses, social security numbers, birth, death and marriage records, financial records, medical records, and immigration records, all of which are excellent targets for those seeking to perform identity theft, for example. Government also manages law enforcement, national security, critical infrastructure, and military data which are also highly valuable assets that must be protected from compromise. While it is necessary for information to be accessed by systems around the world, the undertaking of detecting intrusions or anomalies on such interconnected devices is a non-trivial task. Many compromised devices in small or large organizations are not discovered for an extended period of time due to the painstaking task of monitoring network traffic, maintaining IDS systems, and creating and implementing security policies which secure the environment but do not disrupt the flow of business. In addition, legacy hardware or software often must be supported.

The complexity of the Internet and other interconnected networks is a certainty which is inescapable. It is essential that the lag time between compromise and detection be minimized to protect the organization’s data and resources. As in other IT security realms there are technologies that can aid in this effort, but while a security officer (or security team) at a small to medium sized business can hope to have a reasonably clear understanding of their environment, it is certainly beyond our current human ability to fully grasp the extent of the IT resources in an organization at the scale of a modern government, and as such we need to augment our existing technologies with tools which help the human analyst focus on the important issues in the overwhelming deluge of data they are likely to encounter when searching for signs of compromise, investigating an incident, or planning changes to the current infrastructure or configuration. The deployment of honeypots (or honeynets) in an environment provides the opportunity to view interaction with systems for which there is no legitimate activity, thus allowing an analyst to focus more clearly on malicious activity without the noise of legitimate traffic to mask it. A honeypot or honeynet system is designed to have no production value, meaning that it should receive very little legitimate traffic, and as such the signal (malicious traffic) to noise (legitimate traffic) ratio is extremely high.
Honeypots as an IT security tool were first proposed in the 1990s, and the honeynet term was later introduced as groups of honeypots were deployed in a networked fashion that was more representative of production devices. For honeynets to be most effective, they should be indistinguishable from production hosts, at least from the attacker’s perspective. In practice this can be challenging to achieve, in part because each honeypot will include data collection functionality that is not part of a production host, and hiding such functionality from the attacker, who can be expected to gain full control of a compromised honeypot host, is a challenging and constantly escalating battle. Another challenge, which is the focus of this work, lies in providing context appropriate honeynets, which is a problem that is increasingly relevant as networks become less static. Particularly with the proliferation of wireless networks it is common to see devices connect and disconnect from networks, and a wide range of devices, from smart phones to servers, may be present on a given network. The characteristics of the network (e.g., scale, device types, operating systems, services, and activity) on which the honeynet is to be deployed is a vital consideration in the design of the honeynet, and due to the dynamic nature of many network environments this design cannot be assumed to be static, but must rather be adapted in step with changes in the network. In addition, the deployment of honeynets in a reactive manner (i.e., in response to some intrusion event in an effort to fully understand the scope or mechanism of the intrusion) may have to be conducted on a network for which the characteristics are partially or fully unknown to the investigator, and having an automated approach to quickly determining these characteristics and deploying a suitable honeynet may save valuable time during a fast moving intrusion.

In 2003, Lance Spitzner outlined the need for a “plug-n-play solution. You simply plug it in and the honeypot does all the work for you. It automatically determines how many honeypots to deploy, how to deploy them, and what they should look like to blend in with your environment. Even better, the deployed honeypots change and adapt to your environment [1].” In 2008, Spitzner reiterated this need, and identified future work that remains to be addressed in the honeypot community. He emphasized that automation, specifically in the form of an embedded system, is still needed to help combat today’s cybercrimes [2]. This call for a system which has the ability to monitor network traffic and deploy honeypots that are representative of the network environment in which they will be deployed motivates this research effort.

2. Background

Honeypots and honeynets have been tasked with gathering crucial information on computer attackers since their inception. These devices, whether low or high interaction, have traditionally been deployed manually by a network administrator. A fixed number of devices with fixed characteristics (such as which ports are accessible or which services are started) must be manually configured and deployed. All the while the administrator’s network is constantly changing as a result of the addition or removal of machines, updating or adding new operating systems and the addition of services. So for a majority of the time, the predetermined honeynet might look nothing like the actual network environment in which they are deployed. The data gathered by this predetermined honeynet might be interesting but the data does not accurately represent the administrator’s network environment, and is therefore not representative of the types of attacks that the production devices are likely to encounter. For example, consider a honeynet that is deployed on a university campus during the summer months, when the IT staff have time to configure the system to mimic the hosts and services found on the network. At the start of the Fall semester the sudden influx of students, with their newly purchased and independently configured computers, is likely to dramatically alter the network appearance, and the carefully designed honeynet which mimicked the types of devices and services found during the summer is now woefully inadequate. Less dramatic, but equally important, examples of changes to networks, hosts, and services can be seen elsewhere (e.g., the introduction of new hardware, such as the Apple iPhone and iTouch devices, updated operating systems, or new applications). If honeynets are to have real value to production devices they must be capable of providing a consistent and accurate representation of the networks that they seek to protect while providing relevant data after an intrusion has occurred.

3. Foundational Work

In the past several years there has been some progress in the area of honeypot systems that have allowed system administrators to automatically
scan their network and create a representative honeynet. These honeypot systems use either passive or active scanning to gain the required information about a network configuration, at which point a honeypot, or series of honeypots, can be configured that reflect the operating systems and services present on the target network. However, none of these systems has the capability to automatically respond to changes in the network, other than through periodic rescanning of the network initiated manually or by scheduling methods, such as cron.

The passive scanning model [3] listens to the network traffic with programs such as p0f [4] and tcpdump [5]. The acquired information is used to piece together a representation of the network configuration. By merely listening to the network traffic, the passive scanner is able to gather much of the information necessary to emulate the network without bringing attention to itself, as this technique does not introduce any network traffic. This also allows the system to continually gather information that can be used for network statistics and trends. However, due to the requirement for the honeypot control system to observe the traffic on the network, changes to the network configuration, such as creating span ports on switches, may be required to ensure that the scanner can observe all of the network traffic. In addition, a scanner assigned to a heavily utilized network may experience dropped packets as the volume of network traffic overwhelms the bandwidth limitation of the connection to span port, or even the ability of the scanner to process packets.

Once the scanner is positioned to monitor the network, p0f uses the characteristics of the acquired packets to deduce the operating system while tcpdump records the services on the hosts present in the network. Another issue with this approach is the fact that some communication from each of the services running on the target network must be present in the observed network traffic for the scanner to correctly identity all computer systems and the associated open ports. Any services which do not have network interactions during the scanning period will not be emulated by the honeypot system.

An alternative approach that addresses some of the limitations of passive scanning is an active scanning system [6], in which additional traffic is injected into the network to obtain a more complete assessment of the network configuration. The active scanning model uses programs such as Nmap [7] to find and fingerprint every host for its operating system, open ports, and service types and versions. The results are used to build a Honeyd [8] configuration file featuring emulated operating systems and services similar to those found on real hosts on the network.

The use of active scanning can address some of the limitations of the passive scanning, such as the requirement that the passive scanner be placed at a location(s) on the network from which it can observe all traffic. However, there are new limitations associated with this method, such as in environments that employ host-based firewalls or intrusion detection devices, which may reduce the accuracy of the active scanner’s results. Other issues with this approach are the bandwidth utilized for actively scanning the network and the fact that an active scan must be initiated for the system to be aware of any changes to the network, such as the addition of a new service or host. There is also the possibility that the act of actively scanning devices can result in unexpected failures in the scanned systems, particularly when older or unpatched devices are scanned, or when IDS devices are not configured to allow the scanning activity. The addition of a security technology which results in such system failures, no matter what the cause, is unlikely to be tolerated in a production environment.

To offset the negative aspects of only being able to use active or passive scanning, several authors suggested using either active or passive scanning based on the network configuration in their design proposal [9]. They proposed a design that would allow the user to choose which method of scanning to gather information. The information collected from the active or passive scanners are then recorded in a database. After gathering enough information to have a complete picture of the network, the dynamic honeypot server would generate a Honeyd configuration file. This proposal provided a good starting point to begin improving on the current methods of gathering a network model. As this paper was a design proposal and not an implementation the intricate details would still need to be contemplated.

Building on these implementations and the design proposal, this research effort helps address some of the concerns of the previous systems and bridge the gap between the current state of honeypots and the desired characteristics of a plug-and-play solution.

4. Proposed Solution

One solution to the problem of dynamically configuring a representative honeynet involves the
combination of both passive and active scanning of network traffic, and has been the subject of the research effort at the Advanced System Security Education, Research, and Training (ASSERT) Center. Progressing on the proposal by the design paper [9] the implemented system is able to be configurable in the various levels of employing the active and passive scanners. A current representation of the network is built from the scanning results, in which an initial honeynet configuration is created and used to instantiate a honeynet.

Though this first deployment of a honeynet might not be a complete picture of the network it still allows information to be gathered by the subset of deployed honeypots. Once the honeynet has been deployed, the honeynet scanner continues to monitor the network, and automatically creates an updated honeynet configuration when it detects that the current network differs from the initial layout by some predetermined amount. As the system maintains an updated honeynet configuration a more complete picture of the network is established and trends in the network are able to be recognized.

4.1 Architecture

This system is comprised of six modules, shown in Figure 1, which work together to collect information about the user’s network. The main module is honeypot_scanner, which monitors and manages the other main modules, and uses the collected information to create the honeypot configuration files. The user’s configuration input is stored in a database table which allows honeypot_scanner to identify under what circumstances it should create a new honeynet, what bandwidth limitations should be placed on the scanning process, the IP range of the target network, the file system location where the resulting honeynet configurations files will be output, and the location of some of the dependent subprograms. Honeypot_scanner continually monitors several modules lest these components unexpectedly shutdown.

4.2 Scanning Methodology

The honeypot scanner has the ability to utilize both passive scanning, in which it simply listens to traffic on the network, and active scanning, which involves the injection of new network traffic.

Passive Scanning. The passive scanning components are based on p0f and tcpdump. The passive scanning components listen to the network from one or several points and collects necessary information to construct a honeynet. The tcpdump_mysql module captures packet header information and records the ports that are communicating while p0f_mysql attempts to identify the operating system (OS). In order for this scanning technique to be effective the scanner must be placed at a network location, or multiple locations, that allow it to observe all traffic. Typically this is accomplished through the use of a span or mirror port. P0f and tcpdump were selected for use in the passive scanning modules in part due to their open source license, ensuring that they could be modified if necessary to provide output suitable for the honeynet active scanner.

Active Scanning. Active scanning involves the generation of network activity to probe the target devices; we refer to this increased activity as noise. Xprobe2 [10] and Nmap are the active scanning subprograms utilized in this system. Nmap and Xprobe2 emphasize different protocols and methods for determining the OS of the target. Xprobe2’s OS fingerprinting technique stresses the ICMP protocol, while Nmap uses the TCP/IP stack. Due to the subtle differences, Xprobe2_mysql was chosen to be deployed during the lower noise settings because of its ability to correctly determine the OS during a short scan. Nmap_mysql takes a more involved approach to identify the OS and ports hence more bandwidth is utilized, thus the higher noise settings engage Nmap to be used as the active scanner.

Scanning Noise Factor. The honeynet scanner has a configurable scanning noise level, which determines to what degree it will add traffic

![Figure 1, System Overview](image-url)
to the network during scanning. At this time there are five noise levels, as follows:

- **Passive Level:** At this level information is collected from the network using only the passive scanning modules, `p0f_mysql` and `tcpdump_mysql`. Although this results in a stealthy scanner that is likely to go unnoticed, it does place limits on the quantity and quality of information gathered, and as such the honeynet generated while in this mode will probably be no more than a gross approximation of the target network. The approximation becomes more accurate as the time progresses. This mode is also not as effective or quick at detecting the negative changes in the network. A negative change is when a host, port, or service is disabled, leaving the network in a net loss of host/services being offered. Conversely a positive change is where a port or services has been changed or enabled, thus providing more information to be gathered by the program. Since passive scanning slowly gathers information and the timeframe is generally large between redeployments of the honeypots, the deployed honeynet is not a particularly accurate representation of the current network, although it may be sufficient to detect large scale trends in malicious traffic.

- **Low Level:** In this mode, passive scanning is used to constantly monitor the network, and trigger a scan from the `xprobe2_mysql` module with port scanning disabled when new systems are observed by the `tcpdump_mysql` module. By using both active and passive scanning a more accurate picture of the network configuration can be obtained, versus passive scanning alone, albeit at the cost of reduced stealth and increased network utilization. `p0f_mysql` and `xprobe2_mysql` are both making a recommendation about the OS of the identified machine and `tcpdump_mysql` is recording the open port on the individual device. While this level does not affect the ability to better detect the negative changes on the port level, `xprobe2_mysql` offers the user a quicker response on determining the OS of a new target machine. This will allow the deployed honeypots to more accurately represent the target environment.

- **Medium Level:** This is similar to the low level setting, with the exception that the `xprobe2_mysql` scan includes the port scanning option. Port scanning allows the user to quickly identify open ports that might leave their target machines vulnerable to attack. Since `xprobe2_mysql` is able to detect both the positive and negative changes at the port level this also allows the resulting honeypots to accurately describe the network by having the same configurations as the production devices. While `tcpdump_mysql` is capturing and monitoring each packet for any new open port, the only ports being actively scanned during the medium mode are 1-1024 and 3306 for both TCP and UDP protocols.

- **Medium-high Level:** As at the other levels the passive scanners are used to trigger the active scanning module, which in this case is `nmap_mysql`. `Nmap_mysql` is invoked using the OS detection and port/service version options across a large number of ports, and reports additional information such as MAC address and vendor, host uptime (sec), last reboot, and distance in hops. The larger port scanning range allows the user to quickly identify any open port on the target device, without having to wait for `tcpdump_mysql` to capture a packet from the communicating port. `Nmap_mysql` is able to more effectively recognize whether a device’s state is up or down when a firewall is in place. The more aggressive scanning and broader range of port scanning comes at a higher price in terms of bandwidth used and time spent scanning the devices, but gathers the largest amount of data. This allows for a quick deployment of an accurate honeynet which represents the entire network.

- **High Level:** Passive scanning continues to monitor the network and record a detected device’s IP address into the active scanning queue. `Nmap_mysql` and `xprobe2_mysql` are used in combination to discover information about a device. A significant amount of noise will be generated from this noise level. However, as with the other active scanning noise levels, the devices are only actively scanned once in a predetermined amount of time. This allows the user to determine how often the devices are actively scanned which has a direct impact on how much traffic is generated from the system.

An appropriate scanning noise level is determined by the user to address the limitations of the target network and the requirement for accuracy in the resulting honeynet. The selected noise level is used during both the startup and monitoring phases.

### 4.3 Startup Phase
During the startup phase, the scanner knows nothing about the target network other than its IP range. The goal of the scanner during this stage is to build an initial honeynet configuration that is representative of the target network. The scanning process begins, regardless of the scanning noise mode, by listening to the network traffic using the passive scanners. Tcpdump_mysql is listening for packets with specific flags set which indicate that a port is responding to an initiating packet. The IP address, MAC address, port number, protocol, and time are logged in a database table, assuming that a record for this IP/MAC/port combination is not already present. If such a record is already present, the time field is updated in the existing record. In addition, if the system is running in anything expect passive mode, the IP and MAC addresses are also placed in the active scanning queue at this time. The mechanism used to process each packet ensures that all the relevant information is stored or updated so the deployed honeypots will represent the production environment.

P0f_mysql will record the devices for which it has identified the OS and it will keep track of the last time it saw traffic from that device. This will allow only recently seen devices to be used in the honeypot configuration files. P0f_mysql identifies each device by its IP and MAC address, thus if the same IP/MAC pair is identified with a different OS then a separate entry will be introduced into a database table. This allows for the system to identify dual boot computers and collect more accurate OS identifications from the P0f_mysql module. If a device’s OS is not identified but tcpdump_mysql records ports which are responding then this information will not be used in a honeypot configuration since the resulting honeypots must have a “personality”, but the information is still stored in a database table for later reference.

If the system is configured to use active scanning, the active_scanner module reads the IP address from the active scanning queue, and initiates the appropriate active scanning module based on the scanning noise level configured by the user. The results of the active scan, which may include open filtered ports, OS versions, and service names, are added to a database table. The time between rescanning a device and the aggressiveness of the scans are also configurable by the user.

Once the system has identified more than a predetermined number of devices within the target network, the information stored in the database is used to create an initial honeynet configuration, which can be used to deploy an initial honeynet.

At this point the system enters the monitoring phase.

4.4 Monitoring Phase

During the monitoring phase, a honeynet configuration has been deployed, and the system continues to gather packets from the network in passive mode. If a noise level other than passive is chosen, an active scan is initiated when new devices are detected. In addition, previously identified systems are rescanned at a user configurable frequency to detect configuration changes, and also to confirm that the device still exists on the target network. Changes observed in the target device are recorded in the database, and when the current state of the network is “sufficiently different” from the currently deployed honeynet configuration a new honeynet configuration is created and deployed. The threshold that determines when the honeynet should be reconfigured is user configurable and is based on percent change. Once the number of identified devices and recognized ports has changed enough to overcome the threshold then redeployment takes place.

5. Honeynet Deployment

The honeynet deployment involves the creation of a Honeyd configuration and XML file based on the current information in the system’s database. A Honeyd instance is then invoked based on the configuration, which can be tailored to the needs of the deployment environment.

Duplication of the target environment: In this case, the honeynet is configured with exactly the same characteristics as were observed in the target network. This includes using the same IP addresses for the honeynet systems, and creating one honeynet system for each target host identified. This approach has value for offline network modeling and analysis, such as creating a representative honeynet for use in a lab environment, but the honeynet can obviously not be deployed in the same network as the target devices while the original devices with the same IP addresses are active.

Partial duplication of the target environment: In this case IP addresses for the honeynet systems are selected from a different, user defined subnet, but all other characteristics of the target network remain unchanged. This option allows the honeynet to be easily deployed alongside the target network.
Integration into the target environment: In this case the IP addresses for the honeynet systems are chosen from unused IP addresses in the target network, using DHCP. In the next version the user will have the option of selecting the percentage of each class of operating system that will be deployed in the honeynet. For example, the user may specify that honeypots should be created at a 30% rate, which would result in three honeypot systems for every ten hosts found in the target network. This option allows the honeynet to be deployed within the target network, although care must be taken to ensure that enough address space exists to handle both the honeypots and production machines.

In each of these cases the Honeyd configuration file can also be manually configured by the user as necessary, allowing for further customization by experienced users. Additional honeypots can be deployed using the XML file that is also created during the honeynet deployment. The generated XML file contains all the necessary information to construct high interaction honeypots utilizing a script to enable services and configure the honeypots to look similar to the production machines. The purpose of creating both Honeyd and XML files is to be able to deploy both a low and high interaction honeypot based on the machines configuration. An example use would be, an attacker probes the network then interacts with a deployed low interaction honeypot. When a predetermined event is triggered during the attack, the attacker is redirected to the corresponding high interaction machine to gather more information about the attack. There are several examples of this type of system called honeyfarms. While current honeyfarms utilize low interaction, high interaction, or both types of honeypots, none of these systems presently possess the ability to mimic a production network and dynamically change to address new machines or services being added. The ability to generate configuration files for both high interaction and low interaction honeypots has not been demonstrated previously.

6. Testing

In order to test the system’s functionality and performance, a series of tests were conducted in a lab environment consisting of a VMware ESX server. Deploying the network in a virtual environment allows the rapid configuration of the network topology and hosts, and the ability to easily reset an experiment to a known state when conducting multiple runs in the same configuration. The VMs were connected to a single virtual switch with spanning port acting in promiscuous mode. The network topology in these cases was static throughout the duration of the test (i.e., no VMs were added to or removed from the network during the experiment). The time at which the system produced the initial honeynet configuration was recorded, as were the times at which subsequent configurations were produced as the system determined new information about the network topology. As seen in Figure 2, the virtual machines (VMs) deployed for the testing environment included two Linux distributions with several versions of the Ubuntu OS and several versions of the Windows OS.
It is obviously vital that such experiments include network traffic, and in the absence of real users in this scenario we generated traffic through web browsing. Two of the VMs were configured as http servers; Ubuntu running Apache and Windows XP SP3 running IIS. The website that was being hosted contained a flash based image that would loop continuously. This method allowed a continual stream of web content to be generated by the server and received by the workstations. One of the Windows XP SP3 systems had Back Officer Friendly (BOF), a simple honeypot, installed. 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).

### 6.1 Experimental Results

The tests were performed at each of the five noise levels; however two tests were done at the medium-high level with different Nmap scan types being executed. Each test was run for an extended period of time to ensure that the system settled on a stable honeynet configuration. The tests were configured with a 30 percent change required to deploy a new honeynet configuration. The percent change threshold included devices identified and ports recognized, so if ten devices or ports were identified it would require at least three new devices or ports to be seen to cause a new honeynet configuration to be deployed. The results are shown in Table 1.

![Table 1, Testing Results](data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAoAAAAHgCAIAAgAFA7SAAAABGdBTUExMSB3I3gsSDQAAAABlBMV0AAAI3ZGMwToyfX27uTaHh9QbtsAAAAAElFTkSuQmCC)

The low noise level test which uses Xprobe2 to actively scan the network, without using the settings to determine ports and services, was able to generate three honeypot configuration files within a four minute time span. The system ceased to detect any additional devices after that 4 minute interval though the test continued for almost 22 hours. The honeypot scanner using the low noise setting was able to accurately identify the operating systems on nine of the eleven devices. The system did a great job in this environment of identifying an operating system in a very short time, five devices in one minute. Though Xprobe2 was able to correctly identify the kernel version on the CentOS machine, it had a little trouble identifying the versions with the Ubuntu operating systems. This would have minimal impact on the deployment of the honeypots, because an organization would most likely not be running several kernel versions of an operating system in a production environment. Also to get an exact rollout of honeypots during the deployment process, the high interaction honeypots would be configured and loaded into the virtual infrastructure and mapped to a low interaction honeypot during the setup process. Thus, when the honeypots are deployed, only the operating systems configured and loaded would be started. The ability of p0f to identify the Windows OS was slightly degraded while Xprobe2 was running.
The next test conducted was using the medium noise level, which produced its first honeypot configuration file within four minutes. Since seven devices and eight ports were recognized on the first deployment, additional configuration files were not produced due to the 30 percent threshold. Xprobe2 configured to detect open UDP and TCP ports contributed to the scans taking a little longer, due to the additional time needed to actively scan each machine for open ports. The port scanning was able to identify several more open ports in the environment though there was no communication. A rather odd anomaly occurred between the low and medium noise level, while identifying the version of the Linux kernel. While Xprobe2 was able to correctly identify the kernel version for CentOS on the low noise level, it deduced an earlier version of the kernel while scanning with the medium noise level. As previously seen, the ability of p0f to identify the kernel version for CentOS on the low noise level, while identifying the version of the Linux kernel. While Xprobe2 was able to correctly identify the kernel version for CentOS on the low noise level, it deduced an earlier version of the kernel while scanning with the medium noise level. As previously seen, the ability of p0f to recognize the Windows OS was even more degraded with Xprobe2 and port scanning enabled.

The medium-high noise level uses Nmap as the active scanner. SYN and TCP-connect scans were used during different tests to determine which type of scan would be most effective. Though Table 1 shows a slightly different view, the results were nearly identical. The SYN scan was able to identify the devices and open ports a little more quickly which is demonstrated in the table and resulted in only two honeynet configuration files be deployed. The TCP-connect scan took a little longer which resulted in three deployments and more devices being deployed. While Nmap was able to correctly determine the status on all eleven devices, it was unable to identify the operating system or kernel version on the Linux VMs. Though Nmap was only able to identify two Windows devices, it was able to detect 21 services running for a more complete picture. The length of each Nmap scan had a negative impact on the rapid deployment ability of the honeypot configuration file. The increased time was most likely caused from the duration spent on Nmap trying to determine the operating system of the Linux and Windows VMs. Perhaps additional tweaking in the program call to Nmap is necessary to correctly identify the Linux machines and decrease the amount of time required to perform each scan.

The final test was the high noise level that utilizes both Nmap and Xprobe2. By using Xprobe2 again, the system was able to recognize the Linux devices in the environment. However, p0f’s ability to detect any OS was severely degraded by active scanning. Despite this detriment, the system was able to recognize the state of all eleven devices and correctly discover almost all of the open ports. The time to deploy the honeynets was the most extensive due to both Nmap and Xprobe2 needing to scan each device.

As demonstrated by the results, the use of active and passive scanning enhances the accuracy of the honeynet representation of the network. Each of the programs played a vital role in process. P0f was able to identify the Windows OS, Xprobe2 recognized the Linux OS and Nmap discovered the open ports.

7. Future Considerations

There are many future areas of research that will further increase the value of honeynets including: simplifying the process, improving data analysis, and expanding the datasets used for proactive and reactive IDS analysis.

Simplifying the process could include a plug-and-play solution with default settings and a user friendly interface which allows the administrators to easily configure the system to meet their needs and networks guidelines. Since a system such as this requires minimal input from the system administrator during the deployment phase, an embedded system or appliance could be developed, coupled with a web-based user interface. This would provide the administrator with a drop-in solution to monitor devices connected to the network and simulate their presence with a honeynet, all based on the addition of a single appliance to the network. If multiple devices are needed, depending on the network configuration, a control sensor can be installed to monitor and connect the embedded devices. This interface can be used to update the administrator of the current network trends and deployed honeypots. The control sensor could also be configured to sense whether additional honeypots need to be deployed in a certain area due to the increase rate of compromised honeypots.

Another area for future improvement would be to refine the subprogram commands at each noise levels to increase efficiency or allow the user to specify the subprogram commands to ensure that it is appropriate for their environment and data gathering requirements. Also calculating the devices identified with more weight versus the honeynet representation of the network. The length of each Nmap scan had a negative impact on the rapid deployment ability of the honeypot configuration file. The increased time was most likely caused from the duration spent on Nmap trying to determine the operating system of the Linux and Windows VMs. Perhaps additional tweaking in the program call to Nmap is necessary to correctly identify the Linux machines and decrease the amount of time required to perform each scan.

The final test was the high noise level that utilizes both Nmap and Xprobe2. By using Xprobe2 again, the system was able to recognize the Linux devices in the environment. However, p0f’s ability to detect any OS was severely degraded by active scanning. Despite this...
p0f during the active scanning would enhance the system’s ability to identify the Windows devices.

There are also issues associated with data analysis. Digital forensics and IDS systems are traditionally data-intensive, and this has become a very pressing issue with the increase in data collected from traditional honeypots. While the honeypot scanner produces configuration files which can be used by an administrator to observe the network and computer trends, additional features can be applied to reduce the work of data analysis. Many tools have been developed to quickly analyze a system for current running processes and system changes. Difficulties arise in determining which processes are legitimate and which have occurred during a compromise. Since we are dynamically deploying a honeypot with pre-established ports and system configurations, the difference between the compromised hosts and the honeypot template should be relatively easy to determine. This kind of integrity scanning function can be added to this process to help reduce the time needed to determine the method and source of compromise. A final observation is that the packet payloads captured by tcpdump during the network analysis phase are dropped, but interesting payloads could be stored to allow the user to analyze them for further insight. The overall potential for the system to contribute to forensic data collection is a positive step in evolution from reactive to proactive defenses.

This approach to automated honeynet deployment provides a method through which system administrators can easily deploy honeynets appropriate to their environment. Once deployed, the captured data can then be used to augment current IDS solutions. With data sanitization, the data can be combined with other honeynet datasets to provide an organization with current and emerging malware and attack trends.

8. References