Evaluation of Fingerprinting Techniques and a Windows-based Dynamic Honeypot

Hamid Mohammadzadeh1, Masood Mansoori2, Ian Welch3

1 Faculty of Computer Science and Information Technology, University of Malaya, Malaysia
2 School of Engineering and Computer Science, Victoria University of Wellington, New Zealand

Abstract

Server honeypots are static systems, setup to monitor attacks on research and production networks. Static honeypots are unable to represent the dynamic nature of today’s networks where different numbers of hardware devices and hosts running various operating systems are online at a particular time and frequently join and leave a network. A single static server honeypot presents a particular operating system, open ports and hardware type which are associated with a unique address (i.e. IP/MAC address). A static honeypot system is also always present on the network while other hosts leave and join frequently. These properties of static systems can be a revealing indication of presence of a honeypot within a network. Dynamic honeypots overcome the static nature of server honeypots by automatically adjusting the number of hosts, operating systems and running services of honeypots deployed in a network environment, based on the topology of the production network. In this paper a dynamic honeypot design with self-configuring capabilities based on Windows platform is presented with a focus on usability and simplicity in installation, configuration and management. The honeypot can be deployed within production networks without requiring prior knowledge of network topology, hardware, operating systems and associated services and open ports on the network. Dynamic honeypots can lead to popularity and increased adaption of server honeypots with end-users and within production networks. Active and passive fingerprinting techniques utilized in the process of mapping a network and its systems for dynamic honeypot deployments are also evaluated and their accuracy and speed in detection are measured and discussed.

Keywords: Honeypots, Fingerprinting, Dynamic Honeypots, Honeynet

1 Introduction

Honeypots are decoy systems setup to gather information regarding an attacker’s activity to gather information, penetrate and access a system. The ultimate goal of an attacker is to compromise the victim’s machine and obtain full control. The compromised machine can then be used to perform malicious activities such as accessing private resources, carrying out Denial of Service (DoS) attacks and sending large volumes of Spam emails.

Honeypots are one of the security tools that help administrators identify and combat these attacks. Honeypot’s simplicity in design concept, easier deployment and attack analysis makes it a valuable tool in security structure of an organization, enabling administrators to also monitor propagation of worms, detect insider attacks or an outsider who has bypassed the firewall and intrusion detection system.

Honeypots use deception by mimicking operational hosts of a production network. They can be deployed in large numbers within an internal network to increase the probability of capturing an attack. Unlike network intrusion detection systems, honeypots can only detect attacks targeted at them, therefore a higher number of honeypots in a network drastically increases the probability of detection. Deployment of large number of honeypots requires resources and extensive amount of time to configure, deploy and manage.

The first step in deployment of a honeypot within a network requires administrators to determine how many honeypots to set up and what network hardware, operating system and services they should represent based on the current topology of the network. The process of remotely obtaining network and operating system information is called fingerprinting.

Today’s networks comprise of diverse group of operating systems and hardware from various vendors. Mobile users contribute to such diversity by introducing mobile operating systems into the design of these networks. Statically placing honeypots in a network environment is not an ideal approach to deployment of honeypots in a dynamic environment where users frequently join and leave a network. Static honeypots do not represent the diversity of operating systems and hardware within the internal network at any given time. An attacker can monitor the traffic within a production network for a certain period of time and identify the honeypots from the production systems using a combination of factors such as a host’s online state, IP/MAC address, operating system and incoming/outgoing traffic. Server honeypots by definition are not a part of the production network, therefore they do not have inbound or outbound traffic. Any packet in and out of a honeypot is automatically categorized as malicious (Spitzner, 2003). A detected honeypot has no production or research value and could pose a threat to the security of the production hosts and validity of the analysis captured by the honeypots, as attackers simply bypass them or introduce false information. Dynamic honeypots were developed to overcome some of the challenges of installation, management and static nature of honeypots in a dynamic network environment.
The contributions of this work are as following:

- Design and development of a dynamic honeypot which performs network mapping, configuration, deployment and redeployment of low interaction server honeypots periodically based on the state of a network at a particular time.
- Evaluating the effectiveness and accuracy of operating system fingerprinting techniques and tools in the mapping process of a network, required by dynamic honeypots.

2 Dynamic Honeypots

One of the key factors in adaption of any new technology is its simplicity in deployment and management. Most security tools require a human to deploy, manage and monitor them directly. Constant monitoring of security systems is costly in terms of time and administration overhead. Minimizing these costs can help a security tool in its adaption within the security structure of production networks.

Honeypots, like all network security technologies need to be configured and monitored. Traditionally, low, medium or high-interaction honeypots are configured manually and deployed based on the purpose of deployment and the state of the network at a particular time. These honeypots are then monitored constantly, reconfigured and redeployed by an administrator to represent the topology of a network. Dynamic honeypots tend to lessen the required skill and facilitate the process of configuration and deployment of honeypots within a network. They strive to be a plug-n-play solution where they can simply be plugged into the network, and the honeypot does all the work.

Dynamic honeypots automatically determine the number of honeypots to be deployed, how they are deployed, and what they should look like initially so that they can blend with the organization’s network environment. Moreover, the deployed honeypots can change and adapt to be based on the operating system or the hardware added to the topology of the network. For instance, once Windows XP is added to the network, Windows XP honeypots are established or if a Windows 2008 Webserver server is removed from an organization’s network, the Windows Webserver honeypots automatically disappear.

3 Related Work

Dynamic honeypot research has focused on three main challenges of network mapping, automation of installation and deployment/redemption and level of interaction. Nmap and p0f are the mapping tools in the design of available dynamic honeypot systems (Kuwatly et al., 2004; Anuar et al., 2006; Hecker et al., 2006).

Honeyd is the popular honeypot system used in all implementation. Snort is also used in conjunction with Honeyd to capture and analyze attacks at network layer. Kuwatly et al. (2004) implementation uses high interaction and Sebek to monitor and capture attacks at host and network layer. Their design allows redirection of attacks from Honeyd to a high interaction honeypot for further logging capabilities. (Anuar et al., 2006) design focuses on web management aspect of the dynamic honeypot and incorporates p0f and Honeyd running on a Linux system to provide backend of the system while web interface allows users to remotely configure and deploy and manage honeypots.

Available dynamic honeypot systems are entirely developed for UNIX and Linux based systems. This is due to the flexibility those platforms provide in terms of development, available APIs and tools. However this imposes a restriction on those with less technical knowledge of the UNIX or Linux environments. The motivation of this research is to develop a dynamic honeypot system for Windows platforms with usability in mind, providing an interface that allows automatic yet extensive configuration of desired honeypots based on network topology in near real time.

4 Proposed System

Our proposed design is comprised of the following components:

- **Honeypot system**: To accumulate simplicity and minimize the knowledge required to maintain and secure honeypot systems and underlying operating system, low interaction honeypot WinHoneyd (NetVigilance, 2009) was selected. Low interaction honeypots run at application layer and provide low interaction for an attacker. The simulated environment of a low interaction honeypot ensures that an attacker will not be able to take control of the honeypot and use it to launch attacks against production systems in the production network environment. WinHoneyd is a Windows based version of the de facto low interaction honeypot Honeyd (Provos, 2003). WinHoneyd provides all the functionalities of the Honeyd and includes all the required libraries to work as a stand-alone application on Windows machines.

- **Winpcap library**: Winpcap library provides packet capture capabilities for many network packet sniffers (e.g. Wireshark, p0f)

- **Network Analyzer**: As mentioned earlier, p0f tool was chosen as the passive fingerprinting tool that enables mapping of the network systems, unused and used IP addresses and operating systems.

- **P0f Log parser**: P0f log parser processes log file created by p0f tool after an initial sniffing period. It parses the log file and extracts required information to create configuration file for honeypot creation. The required information is: The used IP address, Unused IP address, number of different operating systems available on the network and mapping an IP with an operating system.

- **Honeypot configuration file generator**: This module is responsible for automatic generation of configuration files required by WinHoneyd. The honeypot engine is responsible for deciding the number and placement of honeypots to deploy. In our implementation, two honeypot systems were deployed for each individual active host. In a small network consisting of four Windows and three Linux machines, assuming
unused IP addresses are available, eight Windows and six Linux honeypots will be deployed. The twofold number of honeypots was selected to increase exposure and maximize the probability of attack detection in the network environment.

- **Analysis component:** This module is responsible for collecting log files from multiple deployed honeypots and representing it to the administrator for further analysis.

Figure 1 provides an overview of the proposed design.

**Figure 1: Proposed dynamic honeypot modules**

### 4.1 System Flow and Interface

The initial step in the system is to map the network, running systems and corresponding operating systems. Once a network has been mapped, the system allows an administrator to create templates that can be applied to produce honeypots on a single machine or an entire subnet, either automatically or specified by the user. In VMware virtual environment, MAC addresses are assigned in formats of “00-05-69-xx-yy-zz”, “00-0C-29-xx-yy-zz” or “00-50-56-xx-yy-zz”. These range of MAC addresses are only allocated to VMware virtual machine (VMware). An attacker will be able to potentially detect the presence of a honeypot through identifying its MAC address. To avoid this, the system allows users to select interface type for a particular honeypot which assigns a unique MAC address to the honeypot, mimicking a unique physical device from a specific vendor (Figure 2).

**Figure 2: Unique MAC addresses can be assigned to each interface, avoiding VM detections through MAC address**

During template creation, a user is allowed to specify protocols and ports to monitor on the deployed honeypots and indicate the actions to be taken when a specific protocol packet is received. The engine configures the default protocol number based on the default ports (Figure 3).

**Figure 3: Protocols and ports can also be manually set for a template or specific honeypot**

There are two modes by which honeypots are arranged in the network; i) Manual configuration by the user or ii) Automatic configuration by the honeypot engine. If a user is familiar with the network topology and is interested to use a specific IP address or range of IP addresses, manual configuration overrides any automatic scripts created by the honeypot. Manual configuration is also suitable for networks with specific number of hosts and static IPs. In order to avoid any IP address conflicts in dynamic networks, honeypots can be deployed by the honeypot automatic engine. The system also allows an admin to enable or disable a particular honeypot once the clients are configured by honeypot through the automated process.

Honeypot engine uses the first available IP address in a network to deploy the honeypot. Initially it sorts the used IP addresses and maps them to an operating system. The decided number of honeypots is then deployed starting from the first available IP address. Below is an example of a simple network layout and corresponding honeypot deployment:

**Operational Network:**

Windows1, Linux1, Unused, Unused, Unused, Linux2, Unused, Unused, Unused

Would produce the following order of honeypots:

**Windows, Linux1, H-Windows1, H-Linux1, H-Linux2, H-Linux2, unused**

Consequent redeployment of honeypots based on updated network topology is performed by the honeypot engine and user defined intervals. The redeployment involves re-scanning and re-mapping the network, generating new configuration files based on the changes to the network and redeployment of the honeypot to represent the changes. The network mapping engine updates/gathers LAN network mapping data every 10 minutes by default. This time interval is chosen so that it can accurately record any changes that take place in the network. The newly generated output file will be different when compared with the previous file, and is used by the honeypot engine to reflect the updated scenario. These
The proposed system also allows honeypot to automatically manage command line script files that can be executed to map, and deploy honeypots accordingly. The automatic command line generation utilizes the default template for each honeypot (Figure 4).

![Annotate “template0” fragment old](image)

Create Provedinfo0
set Provedinfo0 ethernet “D-link”
set Provedinfo0 personality “XP SP4”
set Provedinfo0 default icmp action open
set Provedinfo0 default tcp action open
set Provedinfo0 default udp action open
add TCP port 80 open
Create template0
set template0 ethernet “JComi”
set template0 personality “Windows 2008 Server”
set template0 default icmp action open
set template0 default tcp action open
set template0 default udp action open
Bind 202.186.109.1 template0
Bind 202.186.109.2 Provedinfo0
Bind 202.180.109.3 Provedinfo0

![An excerpt for a default dynamic honeypot template configuration file](image)

5 Fingerprinting Tools and Techniques

Dynamic honeypots are designed to represent a production network or blend in with the surrounding network environment to avoid detection; therefore determining the state of the production network is an essential part of a dynamic honeypot design. The first step in deciding the number and type of honeypots in a network is to accurately determine the topology and detect the hardware, operating systems and services present in the network at any given time. There are two main techniques by which fingerprinting is achieved in a network environment; active and passive fingerprinting. The hypothesis is that passive fingerprinting can achieve similar detection accuracy to more popular active fingerprinting technique while avoiding fundamental issues that arise from sending active probes to systems on a production network.

In the following section fingerprinting techniques, utilized in the design of dynamic honeypots are discussed and evaluated. Their effectiveness in accurate detection of operating systems, the time to scan a network subnet and the amount of traffic introduced into the network environment are measured. These fingerprinting techniques are also evaluated in terms of compatibility with existing security systems available on a production network and if they can be embedded without causing incompatibility with available hardware and software present on the network. Although this paper focuses on two of the most popular tools, the results represent their respective fingerprinting techniques’ capabilities.

5.1 Active Probing

Active fingerprinting is achieved using signature detection technique by which unique messages are sent as probes to the target system and responses are analyzed. The difference in the responses reflects the differences in implementation. By analyzing the header fields of the response message, the operating systems can be determined. Active fingerprinting techniques are based on TCP and ICMP (Internet Control Message Protocol) protocol.

Nmap (Lyon, 2012) is one of the most widely used active fingerprinting tools utilized in deployment of dynamic honeypots. Active fingerprinting however has several drawbacks:

- Takes between 5 to 60 seconds for individual system and may generate large volume of data that can cause systems to shutdown (Kuwatly et al., 2004).
- Nmap needs to keep requesting answers from the clients if it wants to work in real-time. Therefore, if a local system has a firewall with high security, the targeted system cannot respond.
- Active fingerprinting can be blocked by the network protection tools, and can lead to setting off IDS (Intrusion Detection System) alarms and in worst case scenario cause legal proceedings.

5.2 Passive Probing

Passive fingerprinting detections works on a similar principle as the active probing, based on a database of known signatures. However, in this technique, instead of sending probes and requesting replies from a target host, the data is obtained passively by sniffing the network for packets transmitted by that particular system. This data will then be compared with the OS signatures database to identify the remote host and its services. Each operating system uses different implementation of the TCP/IP stack. By analyzing the sniffed data, it is possible to get sufficient information to detect a particular OS. Passive fingerprinting is not limited to TCP/IP; it can also be used with other protocols (e.g. ICMP) (Arkin, June 2001).

Passive fingerprinting offers the following advantages:

- It is not intrusive. It reduces the network traffic and network bandwidth since it collects data passively.
- It can detect network topology behind Network Address Translations and packet filters.
- It is not detectable by the remote user agents, but is less robust than active fingerprinting if the protocol headers are spoofed.

Passive mapping works effectively in Local Area Networks but may not work well across routed networks. In a Switch network environment, the passive fingerprinting tool is blind to the packets traversing on other ports and is only able to receive Broadcast traffic, Multicast with CGMP (Cisco Group Management Protocol), IGMP (Internet Group Management Protocol) snooping disabled and unknown Unicast traffic. In order for any packet to be analyzed by passive fingerprinting tools, packets transmitted through other ports must be replicated to a monitoring port where passive fingerprinting sensors resides. SPAN (Switch Port Analyzer) or port mirroring (monitoring) provides such a functionality with no or negligible performance impact (Cisco, 2007). Packets might be dropped in high-speed switches where a large volume of data needs to be replicated or transmitted to a single monitoring port, causing congestion and packet drop in output queue.
However despite possible packet drops in fast overloaded networks, passive fingerprinting is able to correctly identify operating systems as it requires a small number of packets to perform identification. P0f (Zalewski, 2012) is one of the most common Passive OS fingerprinting tools used in the area of network mapping.

5.3 Setups, Experiments and Evaluations

Prior to the implementation of the proposed dynamic honeypot design, network mapping tools were tested in two different sets of network environments, using the tools’ various settings. Various tests were performed on the hosts of two network environments to measure their properties in three aspects which are crucial to an optimum implementation of network mapping and deployment of dynamic honeypots. These properties are, Time to map a network and generated traffic, Accuracy in detection and IDS/IPS (Intrusion Prevention System) evasion.

Setup: Network Environment 1 (NE1)

The network comprised of 32 Windows XP and 32 Windows 7 machines with default firewall Disabled. This is a production network where random machines are in active or idle state. Nmap and p0f tools were selected and ran on 64 nodes to measure the time it takes to map a production network.

Experiment: Mapping Time and Generated Traffic

Network mapping can be performed in real time or user defined intervals. The defined intervals however must be set to allow the fingerprinting tool to accurately determine the active hosts and associated operating systems. Traffic generated by such tools are also an aspect that must be considered due to several reasons such as network congestion issues during peak times, intrusion detection systems false positive alerts and the probability of denial of service to users if services are affected by the generated traffic.

Results: Figure 5 shows the difference in scanning time using active and passive fingerprinting tools for various numbers of hosts in network environment 1 (NE1). As it can be seen, the scanning time increases at a higher rate in p0f as the number of hosts increase. This is due to the fact that p0f requires more time to process and analyze sniffed packets and to determine the total number of operating systems as traffic load increases. Nmap on the other hand sends a specific number of probes to each single target host, regardless of the network traffic volume or number of available systems in a network environment.

Setup: Network Environment 2 (NE2)

Ten different operating systems were set up on a /25 subnet - with no other active hosts present – to provide a controlled environment where various aspects of the fingerprinting tools can be tested and measured. The installed operating systems were:

1 - Windows Server 2003 R2 Standard 64bit
2 - Windows Vista Enterprise SP1 64bit
3 - Windows Vista Business 32bit
4 - Windows Server 2003 SP1 Standard 32bit
5 - Windows 2008 Server R2 Standard SP1 64bit
6 - Windows 7 Home Premium SP1 32bit
7 - Windows Webserver SP1 32bit
8 - Windows XP Professional SP2 32bit
9 - Mac OS X 10.6.8
10 - Linux Mint 12 64bit (Kernel 3.0)

Experiment 1: Mapping Time and Generated Traffic

Multiple settings of Nmap were tried to determine the time to scan a particular subnet (i.e. N.E2) and the amount of traffic generated in the process. Since a dynamic honeypot attempts to monitor a network in real time, scanning the hosts in pre-defined intervals is a requisite of the process. The best solution must be chosen to minimize the effects of the scanning on the performance of the network particularly in networks with large number of hosts during peak times, to avoid congestion.

Results: Based on our tests, the largest amount of traffic generated by Nmap in Network environment 2 was the fingerprinting using intense scan script which fingerprints a host’s operating system and all 65535 TCP ports. The generated traffic reaches 55MB over a period of 490 seconds and a peak download/upload rate of 2.23 MBps. In a large network of hundreds of potential hosts, this number can increase significantly. However today’s fast networks can handle such amount of data. A 100mbps and gigabit networks have specified maximum throughput of 12.5 MBps and 125 MBps respectively. Wireless routers and access points running 802.11n have a theoretical maximum throughput of 300Mbps which is higher than the required throughput by any of the active scanning techniques. Moreover this setting is the highest scanning setting provided by Nmap and is not required by the dynamic honeypot. A less intrusive and traffic generating setting such as “quick scan plus” can achieve the same accuracy while significantly minimizing the generated traffic (Table 2). Table 1 provides an overview of the scanning with default setting of Nmap against NE2.

Figure 5: Graph representing the time to detect hosts using active and passive fingerprinting tools on NE1
Table 1: Scanning time and generated traffic using various Nmap settings

<table>
<thead>
<tr>
<th>Nmap</th>
<th>Included Detection Switches</th>
<th>Time to Scan (Seconds)</th>
<th>Amount of Traffic Generated/Received</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intense Scan</td>
<td>Includes: -sV Version detection, SMB OS Discovery script Added: -O (OS Detection)</td>
<td>242</td>
<td>(562.47KB)</td>
</tr>
<tr>
<td>Intense Scan plus All UDP ports</td>
<td>Includes: -sV Version detection, SMB OS Detection Added: -O (OS Detection)</td>
<td>2504</td>
<td>(847.76KB)</td>
</tr>
<tr>
<td>Intense Scan plus All TCP ports</td>
<td>Includes: -sV Version detection, SMB OS Discovery script Added: -O (OS Detection)</td>
<td>529</td>
<td>(31.969MB)</td>
</tr>
<tr>
<td>Quick Scan</td>
<td>Added: -O (OS Detection)</td>
<td>157</td>
<td>Rcvd: (125.34KB)</td>
</tr>
<tr>
<td>Quick Scan Plus</td>
<td>Includes: -O (OS Detection) Added: -sV Version detection, SMB OS Discovery script</td>
<td>267</td>
<td>(256KB)</td>
</tr>
<tr>
<td>Comprehensive Scan</td>
<td>Added: -O (OS Detection) Includes: -sV Version detection, SMB OS Discovery script</td>
<td>2322</td>
<td>(927KB)</td>
</tr>
<tr>
<td>smb-os-discovery Script</td>
<td>Scan, ports 139/445 None</td>
<td>7.2</td>
<td>(13KB)</td>
</tr>
</tbody>
</table>

**Experiment 2: Detection Accuracy**

Detection accuracy of active fingerprinting tools is an uncertain aspect of the available technique. OS signatures can be replicated and spoofed to mimic different operating systems. Packet headers are also modified in transmission by intermediary devices which will result in inaccurate detection once matched against the OS signature database. NAT (Network Address Translation) gateways, firewalls, and especially port forwarding can confuse OS detection. Active fingerprinting tools rely heavily on port scanning for operating system detection; For instance OS discovery cannot be performed on Nmap without specifying a port scanning technique. Host based firewalls drop incoming active probes and any traffic which is not initiated from the host system, and prevent it from replying to the probe. Operating system detection test performed by Nmap on hosts of NE2 with enabled firewall resulted in no detection at all on any hosts despite various settings:

- "-O" Switch: Fingerprint based on scanning 1000 common ports for each protocol.
- OS Guessing - Nmap’s "--os-guess" switch attempts to guess the best match if it fails to detect the operating system through the default -O switch.
- Additional, Advanced and Aggressive "-A" mode: -A switch incorporates the version detection of services running on open ports for more intrusive but accurate detection (-sV). It includes the default -O detection.
- "smb-os-discovery” Script: This script attempts to determine the operating system over the SMB protocol on Windows and Samba on Linux systems (ports 139 and 445). This is achieved by starting a session with the target system; in response to a session initiation, the server will send information (e.g. operating system, computer name, workgroup, system time, NetBIOS computer name).

Disabling host firewall can resolve this inability of OS detection through active fingerprinting packets but exposes the host system to a significantly higher number of attacks.

**Result:** With default firewall off, Nmap manages to detect most of the operating systems reliably. Nmap works effectively if one open and one close port are available on target system. Detection accuracy however depends on the type of the scan used. In our test, default OS detection (-O) managed to detect nearly all operating systems, albeit with few extra guesses. Version detection (-sV) while taking longer and generating more traffic, did not result in improved detection accuracy. The smb-os-discovery script however managed to accurately determine the operating system and any individual versions and service packs (e.g. Windows Vista Home Premium SP1). SMB (Server Message Block) is mainly used to provide shared access to files, printers, serial ports, and miscellaneous communications between nodes on a network. Several operating system details (e.g. name, service pack, workgroup, domain name) are shared for easier resource discovery on SMB protocol implemented on Windows systems.

Table 2 provides a comparison between various operating system detection techniques in Nmap and passive p0f tool. Default OS detection and Version detection produced similar results in each scan setting regardless of any other switches used in the scan process (e.g. all UDP/TCP port scanning, fast port scanning mode -F). SMB OS detection script detected all operating systems accurately apart from MAC and Linux which returned the version of Samba running on the Linux host. On Windows based hosts, it required a single open port (139,445) to perform scanning. If there were no open ports present, SMB detection simply failed and did not provide any information.
regarding the target’s operating system.

P0f passive fingerprinting tool does not rely on the status of the host firewall or the availability of any open/closed port on the host machine and manages detection through analyzing the intercepted packets off the network. P0f also relies heavily on network traffic and will fail to detect the host OS if no packets are generated. In our test, P0f failed to recognize the operating system through ICMP only packets, therefore pinging a range of IP addresses will not result in detection of host operating system. P0f failed to detect a host’s operating system despite packets being transmitted in SNMP (Simple Network Management Protocol), NBNS (NetBIOS Name Service), LLNR (Link Local Multicast Name Resolution), SSDP (Simple Service Discovery Protocol), IGMP (Internet Group Management Protocol) and DNS (Domain Name System) protocols. TCP based protocols (e.g. HTTP, Telnet, SSH), NBSS (NetBIOS Session Service) and SMB (Server Message Block) packets resulted in quick detection of host operating system.

It must be noted that detecting Operating system name, release candidate [if available] and service packs are the measurement of accuracy in all scans. Release candidate and service packs represent major upgrade on the latter.

Exploits designed for the former OS to be ineffective which have been patched in Service Pack 2 resulting in a large number of vulnerabilities in Windows XP security of the operating system. For instance there are many vulnerabilities that can involve significant changes to the overall network. P0f also relies heavily on network traffic detection through analyzing the intercepted packets off the network. P0f passive fingerprinting tool does not rely on the respective characteristics in Nmap active fingerprinting tool.

Network Intrusion Detection System (NIDS) and Sax2 Windows based NIDS (Ax3-Soft, 2012). These alerts were raised despite the low sensitivity settings of port scanning in Snort. Nmap provides a general timing policy which allows a user to control how fast each probe is sent to the target host. Timing options range from the nearly invisible “T0 - paranoid” option to the overly-aggressive “T5 - insane” category. Figure 7 provides an overview of the timing options and their respective characteristics in Nmap active fingerprinting tool.

Figure 5 Timing policy properties in Nmap

Messer, 2005

Nmap documentation states that there are several different techniques to avoid raising intrusion detection system alerts. Two of the main and reliable techniques are, delayed and fragmented probes. “T0” and “T1” switches are designed to avoid IDS port scan alerts as probes are sent long apart while “–f” switch uses tiny fragmented IP packets. Packet fragmentation splits up the TCP header over several packets to make it more difficult for Intrusion detection systems and packet filters to detect the probes.

Result: Our tests showed that setting timing between each probe not only did not avoid raising alerts but raised a higher number of alerts in both Snort and Sax2. The timing between the probes does not seem to be the cause of the NIDS alerts but rather the malformed packets captured by NIDS sensors. On the other hand, fragmented packets and limiting the number of port scans to 100 ports down from 1000 for the default setting had absolutely no effect and produced the same number of alerts on each NIDS. Snort and Sax2 also raised “SCAN nmap XMAS” and “ICMP Unusual Length ping” respectively. The following table demonstrates the result of Nmap tests with various IDS evasion techniques and focuses only on port scan alerts raised by intrusion detection systems.

*Default Scanning will scan 1000 most common ports for each protocol. –F switch reduces the scanned port numbers to 100. **Not an IDS evasion Technique

<table>
<thead>
<tr>
<th>Category</th>
<th>Total</th>
<th>Rapid</th>
<th>Default</th>
<th>Slow</th>
<th>Default</th>
<th>Fast</th>
<th>Slow</th>
<th>Default</th>
<th>Fast</th>
<th>Slow</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0 - Paranoid</td>
<td>5 min</td>
<td>Default</td>
<td>Default</td>
<td>Default</td>
<td>15 sec</td>
<td>Default</td>
<td>Default</td>
<td>5 min</td>
<td>Default</td>
<td>15 sec</td>
<td>Default</td>
</tr>
<tr>
<td>T1 - Fingerprint</td>
<td>15 sec</td>
<td>Default</td>
<td>Default</td>
<td>Default</td>
<td>15 sec</td>
<td>Default</td>
<td>Default</td>
<td>15 sec</td>
<td>Default</td>
<td>15 sec</td>
<td>Default</td>
</tr>
<tr>
<td>T2 - Port</td>
<td>Default</td>
<td>5 sec</td>
<td>Default</td>
<td>Default</td>
<td>5 sec</td>
<td>Default</td>
<td>Default</td>
<td>5 sec</td>
<td>Default</td>
<td>5 sec</td>
<td>Default</td>
</tr>
<tr>
<td>T3 - Normal</td>
<td>Default</td>
<td>5 sec</td>
<td>Default</td>
<td>Default</td>
<td>5 sec</td>
<td>Default</td>
<td>Default</td>
<td>5 sec</td>
<td>Default</td>
<td>5 sec</td>
<td>Default</td>
</tr>
<tr>
<td>T4 - Aggressive</td>
<td>5 sec</td>
<td>5 sec</td>
<td>Parallel</td>
<td>3.30 sec</td>
<td>5 sec</td>
<td>Parallel</td>
<td>3.30 sec</td>
<td>5 sec</td>
<td>Parallel</td>
<td>3.30 sec</td>
<td>5 sec</td>
</tr>
<tr>
<td>T5 - Fast</td>
<td>25 sec</td>
<td>5 min</td>
<td>30 min</td>
<td>5 sec</td>
<td>25 sec</td>
<td>5 min</td>
<td>30 min</td>
<td>5 sec</td>
<td>25 sec</td>
<td>5 min</td>
<td>30 min</td>
</tr>
</tbody>
</table>

Table 2: Detection accuracy of p0f and Nmap against various operating systems

Overall Intense and Quick Plus with SMB-OS-Discovery (-A switch) perform the best in terms of OS detection, least amount of generated traffic and scanning time.

Experiment 3: Intrusion Detection/Prevention and Deception System Evasion

Test performed on the network environment 2 using Nmap, resulted in raising port scanning alerts in Snort

<table>
<thead>
<tr>
<th>Category</th>
<th>Total</th>
<th>Rapid</th>
<th>Default</th>
<th>Slow</th>
<th>Default</th>
<th>Fast</th>
<th>Slow</th>
<th>Default</th>
<th>Fast</th>
<th>Slow</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0 - Paranoid</td>
<td>5 min</td>
<td>Default</td>
<td>Default</td>
<td>Default</td>
<td>15 sec</td>
<td>Default</td>
<td>Default</td>
<td>5 min</td>
<td>Default</td>
<td>15 sec</td>
<td>Default</td>
</tr>
<tr>
<td>T1 - Fingerprint</td>
<td>15 sec</td>
<td>Default</td>
<td>Default</td>
<td>Default</td>
<td>15 sec</td>
<td>Default</td>
<td>Default</td>
<td>15 sec</td>
<td>Default</td>
<td>15 sec</td>
<td>Default</td>
</tr>
<tr>
<td>T2 - Port</td>
<td>Default</td>
<td>5 sec</td>
<td>Default</td>
<td>Default</td>
<td>5 sec</td>
<td>Default</td>
<td>Default</td>
<td>5 sec</td>
<td>Default</td>
<td>5 sec</td>
<td>Default</td>
</tr>
<tr>
<td>T3 - Normal</td>
<td>Default</td>
<td>5 sec</td>
<td>Default</td>
<td>Default</td>
<td>5 sec</td>
<td>Default</td>
<td>Default</td>
<td>5 sec</td>
<td>Default</td>
<td>5 sec</td>
<td>Default</td>
</tr>
<tr>
<td>T4 - Aggressive</td>
<td>5 sec</td>
<td>5 sec</td>
<td>Parallel</td>
<td>3.30 sec</td>
<td>5 sec</td>
<td>Parallel</td>
<td>3.30 sec</td>
<td>5 sec</td>
<td>Parallel</td>
<td>3.30 sec</td>
<td>5 sec</td>
</tr>
<tr>
<td>T5 - Fast</td>
<td>25 sec</td>
<td>5 min</td>
<td>30 min</td>
<td>5 sec</td>
<td>25 sec</td>
<td>5 min</td>
<td>30 min</td>
<td>5 sec</td>
<td>25 sec</td>
<td>5 min</td>
<td>30 min</td>
</tr>
</tbody>
</table>
Table 3: Comparison between Nmap's IDS evasion techniques and corresponding IDS generated alerts

Active probes sent by Nmap and such tools are logged in any server honeypots running within the production network environment, manipulating the logged data, resulting in incorrect analysis. Server honeypots by definition are not supposed to have any receiving or transmitting data, therefore any probe packet received by a server honeypot and corresponding responses are simply categorized and recorded as an initiation of an attack. The issue of probes sent by Nmap is increased in a honeynet setup where every transmitted packet sent to every honeypot within the honeynet is recorded by honeynet network sniffers (e.g. Sebek, TCPDump), resulting in large amounts of captured packets to be analyzed.

Passive fingerprinting tools on the other hand do not generate or transmit a single packet in the fingerprinting process therefore evading the detection by Intrusion detection and Deception systems. Their sensors simply sniff the traffic and provide OS detection accuracy nearly in par with active fingerprinting tools. They also reduce the risk of detection by an attacker as no repeated packets are sent down the network in preconfigured intervals – To map a network in near real time - and do not possess an interface with an IP address to be probed and detected as a scanning tool.

Although network scanning is achieved faster through active fingerprinting tools, passive fingerprinting was chosen in the development of dynamic honeypot due to its less intrusive nature, lack of traffic generation and avoiding consequent issues with firewalls, IDS systems and honeypot data capture tools.

6 Conclusion

In this paper a dynamic honeypot design focusing on simplicity and usability aspects was presented targeting users familiar with Windows based hosts. The system allows for automatic deployment and management of honeypots based on characteristics of a production network at a particular time. The system scans a specific network, maps present operating systems and hardware and configures, deploys and redeploys honeypots accordingly, in specific intervals set by a user. Active and passive fingerprinting techniques were also put to test against two network environments and their network mapping effectiveness and efficiency in terms of speed, traffic, accuracy and IDS/IPS evasions were discussed. Passive fingerprinting proved to be non-intrusive with no introduced traffic to the network and bypassed intrusion detection/prevention system while providing subpar accuracy in OS detection.

7 Future Work

There are a few areas in which the system can be improved. The current system disconnects the honeypot regardless of whether an attacker is interacting with the honeypot or the honeypot system is idle. The future system should be able to detect if honeypot is logging an activity and postpone the disconnection and redeployment accordingly.

The system should also disconnect, redeploy and integrate honeypots within the production network in a random manner to increase deception.

8 References


