Fingerprinting for Dynamic Honeypots

Brandon N. Stock
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA 56267
stock400@morris.umn.edu

ABSTRACT

Honeypots are intentionally vulnerable hosts that can be used to lure attackers to easily detect them and thus, improve the security of the system they are being used in. A honeypot can be either static or dynamic. A static honeypot must be completely configured by an administrator. On the other hand, a dynamic honeypot is able to configure itself with the assistance of fingerprinting tools such as Ettercap or Nmap. Fingerprinting tools scan a network to determine the operating systems of the computers in a network. This is important because in order for a honeypot to best emulate another another system, it needs to know the operating system of what it is emulating. A honeypot being dynamic provides it some advantages over a static honeypot. This paper will discuss two systems where tools like Nmap were used to configure honeypots and were tested for their effectiveness in networks.

Keywords

cyber-physical system, intrusion detection, honeypot, honeynet

1. INTRODUCTION

Honeypots are a deceptive intrusion detection tool that is meant to lure intruders in. Honeypots emulate real machines and are intentionally vulnerable systems. A honeypot should not have any traffic inside of it and because of that, they are always able to detect intrusions inside of them. This makes honeypots an effective tool for intrusion detection.

Honeypots can be static or dynamic. A static honeypot must be completely managed and configured by an administrator. A dynamic honeypot can do some of the managing and configuring on its own with the assistance of fingerprinting tools such as Ettercap or Nmap. Fingerprinting tools scan a network to determine the operating systems of the computers in a network. This is important because in order for a honeypot to best emulate another another system, it needs to know the operating system of what it is emulating. A dynamic honeypot can also be manually configured. This will overwrite any configuration choices that the dynamic honeypot may have chosen to do on its own.

Section 2 will give a brief background on intrusion detection, cyber-physical system, and honeypots and honeynets. In section 3 three different techniques of intrusion detection are covered. This is followed by an experiment using dynamic honeypots in a cyber-physical system in section 4 and a experiment using a dynamic honeypots in section 5. Finally, the conclusion is in section 6.

This work is licensed under the Creative Commons Attribution-Noncommercial-Share Alike 3.0 United States License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/3.0/us/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

.

2. BACKGROUND

In this section I will briefly explain intrusion detection, cyber-physical systems, honeypots and honeynets, what a dynamic honeypot is, and introduce networks and their elements.

2.1 Networking Terminology

There are several networking terms that need to be understood for this paper. Those terms will be defined here. A MAC address is a unique identifier that is assigned to network adapters. An IP address is a numerical label assigned to computers in a network. A host is a computer connected to a network. A port is an endpoint of communication in an operating system. ICMP is a protocol used by network devices such as routers to send error messages. A packet is a collection of data that can be used by computers to communicate. Finally, a packet sniffer is a program that targets packets transmitted over the Internet.

2.2 Intrusion Detection

Intrusion attacks are classified according to several categories such as how they affect the system, the threat they pose, if they are an active or passive attack, etc. To evaluate the effectiveness of an intrusion detection system (IDS), researchers usually use three metrics to measure the system's performance, false positive rate, false negative rate, and true positive rate. A false negative would occur if an IDS mistakenly identifies a malicious node as one that is well-behaved. A true positive is the opposite of a false negative and occurs when an IDS correctly identifies a malicious node. Similar to a false negative, a false positive occurs when an IDS mistakenly identifies a node that is well-behaved as malicious. [3]

2.3 Cyber-physical System

Cyber-physical systems (CPS) are large-scale, geographically dispersed, federated, heterogeneous, life-critical systems that comprise sensors, actuators, and control and networking components [3]. Examples of cyber-physical systems would be smart grids, hydro-electric dam controls, and flight control systems. Essentially, a cyber-physical system is a system where its control affects human lives or interacts

with the environment [8].

2.4 Honeypots and Honeynets

A honeypot is an intentionally vulnerable system that is set up to lure in attackers and to gather data on the attacker's activities and methods. Honeypots deceive attackers to infiltrate their systems by mimicking operational hosts. Honeypots can only detect attacks on themselves and not on other systems like some other intrusion detection systems can. To improve the chance of attacks being detected with honeypots, a large number of them can be deployed. A large network of honeypots is called a honeynet.

A honeypot can be static or dynamic. A static honeypot must be completely deployed and managed by an administrator. On the other hand, a dynamic honeypot is able to determine how many of them are needed and deploy, shutdown, and redeploy when it deems it to be necessary. A dynamic honeypot is able to scan the network based on a set time interval and change based on the current state of the network it is running in [4]. Section 3 discusses methods for this process.

3. TECHNIQUES

This section covers the different types of techniques that are used in intrusion detection systems.

3.1 Intrusion Detection in CPSs

An IDS for a cyber-physical system implements two core functions, which are collecting data about suspects and analyzing the data about those suspects. Intrusion detection in a CPS addresses the embedded physical components and physical environment in a CPS. When under attack, the physical components and physical environment manifest physical properties and usually require a closed control loop to react to those physical instances of attacks. A few of the existing intrusion detection techniques used in a CPS are knowledge-based, behavior-based, and behavior-specification-based intrusion detection methods. [8]

3.1.1 Knowledge-Based Intrusion Detection

In knowledge-based intrusion detection, run-time features that match a specific pattern of misbehavior are searched for. This approach is sometimes also referred to as misuse detection. A big advantage of using knowledge-based intrusion detection is that it has a low false-positive rate since this technique only searches for behavior that is known to be malicious. A good node will not exhibit the behavior that is being searched for, so it will not be falsely detected as malicious. [8]

A disadvantage to this technique, though, is that it will only search for a specific pattern. This means that the IDS must have a dictionary of attacks and stay current. So in order for this method to be effective, the dictionary of attacks needs to be constantly updated.

3.1.2 Behavior-Based Intrusion Detection

In behavior-based intrusion detection, run-time features that are not normal are searched for. What is normal is defined based on the history of the test signal or with respect to a collection of training data. For an IDS, behavior that is categorized as good or malicious would be *labeled* data and behavior that is not categorized would be *unlabeled*. If an approach of this technique is unsupervised, meaning that an IDS is trying to find hidden structure in data that is unlabeled, then researches use machine learning and train the IDS with live data. An example of this is called clustering, which takes a set of objects, in this case intrusion attacks, is grouped with other objects that are similar to each other than objects in another group [6].

A semi-supervised approach can also be used to train a behavior-based IDS. Semi-supervised machine learning takes a small amount of labeled data and a large amount of unlabeled data to train an IDS. This could involve training an IDS with a set of truth data, which is data that is provided by direct observation. The approach that researchers take for training the IDS depends on the data that is being protected. [8]

One of the biggest advantages to a behavior-based IDS is that they do not look for something specifically, which rids the need of the IDS knowing all the information about current attacks and keeping an updated and current attack dictionary. One big disadvantage, however, is behavior-based intrusion detection is more susceptible to false positives. Also, if the IDS is semisupervised, the IDS is vulnerable during its training phase.

3.1.3 Behavior-Specification-Based Intrusion Detection

Behavior-specification-based intrusion detection is a variation of behavior-based intrusion detection. This technique formally defines how a good node behaves and will detect an intrusion when the system behaves differently from that definition of good behavior. For behavior-specification-based intrusion detection, one of its major advantages is that it has a low false-negative rate since only situations where the behavior of a node is different from what is defined as good behavior are detected [8]. A malicious node should behave differently than one that is well-behaved and therefore, should be detected.

A second advantage to a behavior-specification-based IDS is that the system does not need any amount of training to be effective and will not have any period of vulnerability like a behavior-based IDS. The main disadvantage is the effort required to define a formal specification of what a well-behaved node will act like. A behavior-based-specification IDS does not use any user, group, or data profiling and instead, an expert defines what is legitimate behavior [8]. The IDS will then detect malicious behavior based on if a node deviates from the behavior that is specified as good.

4. FINGERPRINTING HONEYPOTS

Fingerprinting a network is incredibly important for the use of dynamic honeypots. In order to most effectively emulate a network, a dynamic honeypot needs to know what is in the network. Fingerprinting tools are used exactly for this. Fingerprinting tools determine the operating system of hosts in a network and the fingerprinting tools can either be active or passive.

4.1 Active Fingerprinting

Active fingerprinting is done using signature detection technique, which has unique messages sent as probes to the target system and the responses are analyzed. The responses received will vary based on the implementation of the fingerprinting tool [4]. To determine the OS of the target system the header fields are analyzed and compared to a database of known signatures.

4.2 Passive Fingerprinting

Passive fingerprinting tools also uses a signature detection technique that is based on a database of known signatures. Instead of sending probes and requesting replies from a host, however, the data is obtained by sniffing the network for packets transmitted by the target system [4]. This data is

Nmap	Time to Scan (Seconds)	Amount of Traffic Generated/Received
Intense Scan	242	(562.47KB) | Rcvd: (474KB)
Intense Scan plus all TCP ports	529	(31.969MB) | Rcvd: (28.866MB)
Quick Scan	157	Rcvd: (125.34KB)
Quick Scan Plus	267	(256KB) | Rcvd: (897KB)
smb-os-discovery Script Scan, ports 139/445	7.2	(13KB) | Rcvd: (19KB)

Table 1: Scanning time and generated traffic [4]

then compared to the database of signatures to determine the OS of the system.

4.3 Setup and goals

In Mohammadzadeh et al.'s research [4], the system they tested used the following tools: WinHoneyd, Winpcap library, Nmap, p0f, a p0f log parser, a honeypot configuration file generator, and an analysis component that collects log files from deployed honeypots and then sends those log files to an administrator for analysis. WinHoneyd is a low interaction honeypot. A low interaction honeypot limits the amount of interaction an attacker can have with a system, which secure the system and prevents an attacker from using the honeypot to attack other systems [2]. However, little information is received from an attack because of this. P0f is a passive fingerprinting tool and the Winpcap library provides packet capture capability for packet sniffers like p0f. With this honeynet system, Mohammadzadhe et al. Deployed two honeypots for every individual host on their test network.

They performed four total experiments that tested mapping time for a passive fingerprinting tool and compared it to the time it took for a active fingerprinting tool, generated traffic, operating system detection accuracy, and then intrusion detection/prevention and deception system evasion. The first goal of their experiments were to design and develop a dynamic honeypot that performed network mapping, configuration, deployment, and redeployment. The other goal was to evaluate the effectiveness and accuracy of OS fingerprinting techniques.

4.4 System Evaluation

The first experiment discussed in [4] involved a network consisting of 32 Windows XP machines and 32 Windows 7 machines with their firewalls disabled. The goal of this experiment was to measure the mapping time and generated traffic by fingerprinting in the network. Both Nmap and p0f were used on all 64 of the machines to measure the time that it would take to scan and map a network. Network mapping can be performed in real-time or it can be done in intervals set by the user, but if the intervals are set by the user, then the time must allow for the fingerprinting tools used to accurately determine the network's layout.

Figure 1 shows the difference in scanning time between Nmap and p0f. Nmap was the active fingerprinting tool and p0f was the passive fingerprinting tool. In all of the cases, Nmap was able to scan the network faster. The scanning time for p0f increased at a higher rate as the number of scanned hosts increased. This is because p0f needs more time to process and analyze sniffed packets and to determine the total number of operating systems as traffic increases in a network. Nmap, however, always sends a specific number of probes to each host, no matter the network traffic or the number of systems in a network environment [4]. The accuracy of the OS detection was not reported for these tests.

In the second experiment, mapping time and generated traffic were tested again. This time the network consisted of ten machines, all having different operating systems, and

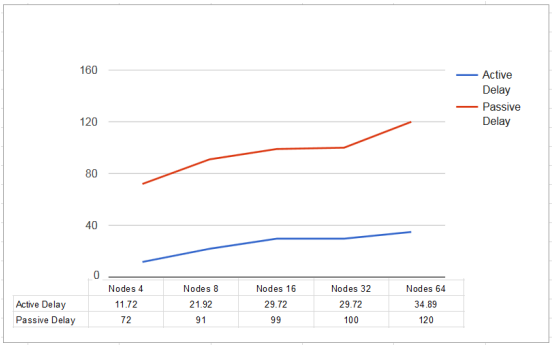

Figure 1: Graph representing the time to detect hosts on active and passive fingerprinting tools [8]

set up on a /25 subnet, a subdivision of a network, with no other active hosts present. This provided a controlled environment for the experiment.

This time only Nmap was used to scan the network. Multiple setting of Namp were used to determine the time to scan a particular subnet and the amount of traffic that was generated during the process. As mentioned earlier, dynamic honeypots attempt to monitor a network in real-time and because of this, scanning the hosts in a specified interval is required. For Mohammadzadeh et al.'s experiment, they needed to choose an interval that would minimize the effects of the scanning of the network to avoid congestion [4].

All of the results from this experiment are shown in Table 1. The two most important scans to look at are the intense scan and quick scan. Intense scan scans the most common TCP ports and tries to determine the OS and services and versions that a host is running. Quick scan only scans the top 100 most common TCP ports to try and determine the OS a host is running. A TCP port is used to accept information from other hosts. A scan like quick scan plus, which adds in version detection to quick scan, is able to produce the same accuracy as a scan such as intense scan while generating less traffic on a network [4]. The quickest scan used a smb-os-discovery script that took 7.2 seconds.

The third experiment tested the operating system detection accuracy of Nmap. In order for dynamic honeypots to work effectively and deceive intruders, the fingerprinting tool used along side the honeypot needs to be able to accurately detect the operating system of machines in the network. The same network was used from the second experiment.

With firewall off, Nmap was able to detect most of the operating systems reliably. Nmap, however, works most effectively if the target machine has one open port and one closed port. Detection accuracy also depends on the type of scan that Nmap uses. In Mohammadzadeh et al.'s experiment, the default operating system scan of Nmap was able to detect nearly all of the operating systems in the network. When a version detection scan of Nmap was used, the results were the same, but a bit slower. A final scan using a script

Host	Default	Version Detection	smb-OS-Discovery
Windows Server 2003 R2 Standard	Win XP SP2	Win. Server 2003 SP1 or SP2	Win Server 2003 R2 SP2
Windows Vista Enterprise SP1	Win 7	Win Server 2008 SP1 m	Win Vista Enterprise SP1
Windows Vista Business	Win 7	Win Server 2003 SP1 or SP2	Win Vista Business
Windows Server 2003 SP1 Standard	Win XP SP2	Win server 2003 SP1 or SP2	Win Server 2003 SP1
Windows 2008 Server R2 Standard SP1	Win 7	Win Server 2008 SP1	Win 7 Home Premium SP1
Windows 7 Home Premium SP1	Win 7	Win Server 2008 SP1	Win 7 Home Premium SP1
Windows Web Server SP1	Win 7	Win Server 2008 SP1	Win Web Server 2008 SP1
Windows XP Professional SP2	Win XP SP2 or SP3	Win Server 2003	Win XP
Mac OS X 10.6.8	Mac OS X 10.6.X	iOS 4.X	None
Linux Mint 12 64bit (Kernel 3.0)	Linux 2.6.38	Linux 3.2	Unix (Samba 3.5.11)

Table 2: Detection accuracy of Nmap [4]

called smb-os-discovery was used and this scan was able to accurately determine all of the operating sytems, versions, and service packs of the machines in the network. However, when the tests were performed with hosts having their firewall enabled, Nmap could not determine any of the hosts' operating systems regardless of the type of scan that was used by Nmap.

Table 2 compares the operating system detection techniques in Nmap. The default operating system detection and version detection were able to produce similar results for each scan. The SMB operating system detection script detected all the operating systems accurately, except for the MAC and Linux computers. The passive fingerprinting tool was unable to recognize any of the operating systems accurately [4].

5. DYNAMIC HONEYPOTS IN A CPS

In this section I will provide an overview of how Vollmer and Manic [5] performed their experiment with dynamic honeypots in a cyber-physical system. The goal of their experiments was to emulate hosts in a cyber-physical system and then test how well they emulated their hosts by scanning for them in their test networking using fingerprinting tools.

5.1 Setup

In Vollmer and Manic's research [5], Ettercap was used as their fingerprinting tool. Fingerprinting is the process of scanning and mapping hosts in a network to determine their operating systems. Ettercap is an open source passive fingerprinting tool that detects a network's configuration by the examination of packet headers [1].

For the dynamic honeypots, Vollmer and Manic chose to use Honeyd. Honeyd is a framework for virtual honeypots that simulates computer systems and their network behavior [7]. Honeyd provides flexible configuration capability and that is one of its advantages [5]. Honeyd is able to take the information gathered from Ettercap and automatically and correctly configure the necessary amount of honeypots. A honeypot needs its operating system chosen and mapped, MAC created, and a service port emulated. This is done based on information from Ettercap.

5.1.1 Operating System Selection

It is possible that Ettercap may not be able to detect the operating system of a host in a network. An operating system still needs to be chosen and it is best to still provide an exact match to best emulate the network. [5]

Pseudo code for Vollmer and Manic's algorithm is shown in Figure 2. The source of information to be used to determine a honeypot's operating system is *Read_Data*. Read_Data consists of extracting information from n host records h from the Ettercap entries and forming a set of host records O containing n hosts. These records are intended to be used to compare to a list of j IP addresses. [5]

```
Create and update virtual hosts with following:
   Network Entity Identification.
   Write entities to XML.

   Read_data; from input files and Ettercap
   For each IP create a Dynamic Virtual Host
      Find_closest representative OS.
      Map_OS values to Honeyd names
      Create_MAC address for new hosts
      Create_Features for device specific behaviors
      Create_Config for virtual hosts
   End
```

Figure 2: Vollmer and Manic's Algorithm [5]

If we have P_h, a set of port values for a host h, and a set of network ports, S_i, for a target i, the method *Find_closest* will examine the intersections of S_i and P_h for all h in the set O. The count of matching ports is stored for each intersection and the number of ports for each target is calculated. Based on these two values, a match percentage is calculated and a system with the highest matching percentage is chosen.

If an operating system cannot be determined by examining ports, then the MAC address is used. *Find_closest* will compare the vendor identification section of the candidate MAC address of i with the MAC addresses for each host h in the set O [5]. Once all the matches are found, the largest matching value is chosen as the OS, if a value exists [5].

If an OS still has not been determined, then a random number r is generated in the range 0 to N where N is the cardinality of O. If the host record h_N contains an OS field, then this is utilized [5]. If no operating system field exists for that host record, the random value is used and a field is possibly selected for inclusion proportional to the relative frequency of its presence in O. If no host records contain an OS, then a completely random OS value is required.

5.1.2 Operating System Name Mapping

To map an OS name, the Honeyd configuration value uses the database of Nmap. Ettercap, on the other hand, uses its own defined names that do not directly match up with Nmap [5]. In order to correctly configure the OS name mapping, Vollmer and Manic created a simple function *Map_OS* that associates Ettercap names with names from the Nmap database.

The first time the algorithm goes through the names of OSs from both the Ettercap and Nmap databases, it com-

pares the strings of the OS names and looks for case-insensitive matches. The number of words that match are summed and stored and after iterating through each OS combination, the combination with the largest sum is presented as the candidate [5]. Each OS name combination is then written to a file to be referenced during creation of the honeypots.

5.1.3 MAC Creation

Honeyd can specify MAC addressed in two different ways, either by vendor name or the six-octet string. Vollmer and Manic chose to use the six-octet representation for their algorithm in their *Create_MAC* function [5]. Ettercap captures all the MAC addresses for all the host records in O. The MAC protocol specifies that the first three octets should be unique to a vendor and should not overlap with any other vendor. Vollmer and Manic then used these first three octets to create a MAC address that would appear to come from a specific vendor [5].

The last three octets are randomly generated and appended to the end of the first three octets. The new MAC address is then compared to all the other MAC addresses that are in the host set O. If any of the addresses are the same, then another set of random values is generated for that address.

5.1.4 Network Service Emulation

All of the host records in O contain network ports, defined previously as P_h. In the host records, a port number and a port service is available. The port service name is a human readable string that is defined in an Ettercap configuration file called etter [5]. Based on the service names in this file, Vollmer and Manic created a new configuration file called serv.conf. This new file maps the port service names to a service emulation script path.

The function *Create_Features* examines any service ports that are found in the Ettercap output and load the serv.conf file. Service names that match entries in the file will result in the correct service script value placement in the Honeyd configuration [5]. This will enable the honeypots to emulate service specific behaviors. In addition to these ports, a variable number of ports that are associated with common services are randomly activated.

The file maps the vendor MAC to a list of common services that can be found on a device of the emulation target [5]. The services in the file are described by port number, protocol, service description, and action script. If used, the action script specifies which script Honeyd should use when it sees traffic on this port. This provides the ability to customize a response to the specific device type and still be able to retain generic service emulation functionality.

Each service description has an include "value", which is a floating-point value between 0 and 1. This value is compared to a randomly generated value in the same range. If the random value is less than the include value, the port is added to the honeypot's configuration. This varies port inclusion to represent the variability in device configurations [5].

5.1.5 Starting and Updating the Honeypots

As mentioned previously, the honeypots that Vollmer and Manic used were dynamic, which allows them to automatically shut down, deploy, or redeploy. The honeypots determine when any of these actions are necessary based on information from Ettercap. Candidate emulation hosts are provided at start up as a list of IP addresses and it is assumed that if one of the hosts disappears during a network scan, then the user still needs to have an emulated version of it [5]. The overhead to maintain any missing hosts' records is minimal. If the actual system appeared in the initial scan of the network, then an initial virtual host configuration will have been created for this host.

```
create vh1
set vh1 personality "Linux 2.4.xx"
set vh1 default tcp action reset
set vh1 default udp action reset
set vh1 default icmp action reset
set vh1 tcp port 23 "/script/router/telnet.pl"
set vh1 ethernet "00:00:BC:A1:00:23"
bind 192.168.1.125 vh1
```

Figure 3: Honeyd host configuration [5]

The initial configuration file is created by *Create_Config*. The configuration of the honeypots will change while the network is running. The time between configuration updates is configurable, but Vollmer and Manic chose 60 seconds. After the 60 seconds, etterlog is called on the ettercap daemon log file and the resulting output is saved and compared to an existing output file [5]. An example of a Honeyd host configuration is shown in Figure 3. The differences between the two files, if there are any, are noted and stored in a list. Based on this list, actions to change network services, updating operating system configuration, and changing MAC addresses is done [5].

5.2 Test System

The CPS Vollmer and Manic used was a small campus grid and a sensor network in the Center for Advance Energy Studies in Idaho Falls, Idaho to test their algorithm. The software for Vollmer and Manic's algorithm was implemented on a test platform that ran a 32-bit Ubuntu 12.04 operating system with a dual-core Intel Atom 330 processor, 2 GB of DDR2 RAM, a 250-GB hard drive and three Ethernet network ports [5]. One of the Ethernet ports was dedicated for use only by the honeypots. Honeyd is capable of running multiple virtual hosts on one physical network interface [5]. The second port was used to perform passive network scanning and the final port was connected to a second separate network used for the management of devices. The goal of their experiments were to emulate any identified hosts on a network as closely as possible [5].

5.3 Evaluation and Results

Vollmer and Manic performed nine tests with their algorithm. Eight of the nine tests were scans on the network to evaluate the effectiveness of various scans and the final test was evaluating the effectiveness of the anomaly detection of the honeypots.

Vollmer and Manic created three virtual honeypots and verified them by sending Internet control message protocol echo messages to them [5]. Then after 60 seconds, an updated input text message was sent to 12 IP addresses of the hosts in the test network. Their software automatically created configurations for all of the devices and assigned each one its own unique IP and MAC address. Out of the 13 devices Vollmer and Manic chose for emulation, 10 we emulated successfully, two were emulated with some degree of randomness, and one device could not be determined [5].

In Vollmer and Manic's first test, they used Nmap to perform a ping sweep on all 256 addresses in the range that contained the 12 successfully emulated devices. A ping sweep is used to establish a range of IP addresses that map to live hosts. Three requests were sent to the emulated devices. The requests were an ICMP echo request, a TCP "Three-way Handshake", and an ICMP timestamp request.

An ICMP echo request sends a message to a target and waits for a reply. In a TCP "Three-way Handshake" a host sends a request to a target, the target responds back in acknowledgment, and the original host acknowledges back again. Then, in a ICMP timestamp request a timestamp is set to the time a host last touched a timestamp. This timestamp is then sent to a target and the target stamps the time on the timestamp before sending it back to the original host. If any system responded to one of the requests, then it is considered to be available on the network. All 12 of the emulated devices were found.

In the second test, Nmap was again used to detect the operating system of the emulated devices. OS detection in Nmap is based on a database of signatures [5]. Each record in the database contains a field for vendor, OS family, OS generation, and device type. Output from the detection done by Nmap includes a list of the possible OSs and device classes with an accuracy score between 0 and 1 for the scanned target.

The OS detection produced multiple results for each device. For the 12 devices, 223 device types and 40 operating system matches were returned. For both of these, the accuracy ranged from 0.85 to 0.97 [5]. Since multiple results were returned for each device, any entry that matched the original device or its mapped operating system were considered a success. 7 out of the 10 successfully emulated devices were identified and the two random emulated devices were not identified.

For the third test, another operating system scan was done with Nmap. This time the scan that was done guessed operating systems more aggressively. The results from this scan were slightly improved to 80% correctly identified compared to 70%.

For the fourth test, Nmap was used send IP packets and iterate through the 8-bit IP protocol field. 256 different protocols were sent to the emulated hosts. The emulated hosts only responded to three of the 256 protocols [5].

For tests five to seven, ICMP echo requests, also called pings, were sent to the 12 emulated devices and 46 actual devices on the test network. ICMP packets are wrapped in an IP datagram and can contain IP option fields [5]. Only one of the 46 actual devices did not respond with varying levels of correctness and none of the 12 emulated devices were able to respond [5]. This test is different from the ping sweep test due to this test only sending an ICMP echo request, where the ping sweep sent three different requests. In the ping sweep test, if an emulated device responded to any of those three requests, then it was considered a success. It is very likely that the request the emulated devices responded to in the ping sweep test was not the ICMP echo request due to the poor results of these three tests.

For the eighth test, a framework called OpenVAS was used to perform more intensive network probes on the honeypots than Nmap is able to do [5]. Vollmer and Manic performed a large-scale discovery and vulnerability scan using OpenVAS on the 12 honeypots. All 12 honeypots were discovered during the scans the vulnerability report looked similar to how a report for actual hardware would look. All of the 12 devices had the same warning about a multi-cast address response flaw that could possibly lead to a denial of service attack.

The ninth test was anomaly detection. A message with the 12 emulated IP addresses was sent to an anomaly behavior component of the network. The anomaly behavior function was to passively monitor host traffic and send alert messages [5]. If the anomaly behavior component received an input IP that it has not been trained to, then it would consider all traffic to it to be abnormal. Honeypots should not have any traffic, therefore, if any traffic occurs in a honeypot, it should be alerted about. The anomaly behavior component alerted for 100% of the emulated hosts during test 1 to 7.

6. CONCLUSION

As we have seen, fingerprinting tools such as Nmap are not able to always identify every host in a network. Because of this Honeyd may not be able to emulate all hosts in a network. This can be a problem when changes occur in the network, but it is also possible to fix this problem by manually configuring honeypots where necessary. It is also possible that if an intruder were to scan the network and analyze information from the network that they are attempting to breach, then the honeypot could be discovered.

While there are these drawbacks with honeypots and honeynets, it is easy to detect attacks directed at them. As previously mentioned, honeypots are intentionally vulnerable and meant to deceive intruders. Since this is the case with honeypots, any traffic in a honeynet is considered malicious and will be detected.

Acknowledgments

Thanks to Elena Machkasova, Nic McPhee, and Melissa Helgeson for providing me with feedback.

7. REFERENCES

[1] W. Ghanem and B. Belaton. Improving accuracy of applications fingerprinting on local networks using nmap-amap-ettercap as a hybrid framework. In *Control System, Computing and Engineering (ICCSCE), 2013 IEEE International Conference on*, pages 403–407, Nov 2013.

[2] A. Mairh, D. Barik, K. Verma, and D. Jena. Honeypot in network security: A survey. In *Proceedings of the 2011 International Conference on Communication, Computing & Security*, ICCCS '11, pages 600–605, New York, NY, USA, 2011. ACM.

[3] R. Mitchell and I.-R. Chen. A survey of intrusion detection techniques for cyber-physical systems. *ACM Comput. Surv.*, 46(4):55:1–55:29, Mar. 2014.

[4] H. Mohammadzadeh, M. Mansoori, and I. Welch. Evaluation of fingerprinting techniques and a windows-based dynamic honeypot. In *Proceedings of the Eleventh Australasian Information Security Conference - Volume 138*, AISC '13, pages 59–66, Darlinghurst, Australia, Australia, 2013. Australian Computer Society, Inc.

[5] T. Vollmer and M. Manic. Cyber-physical system security with deceptive virtual hosts for industrial control networks. *Industrial Informatics, IEEE Transactions on*, 10(2):1337–1347, May 2014.

[6] Wikipedia. Cluster analysis — Wikipedia, the free encyclopedia, 2015. [Online; accessed 14-December-2015].

[7] X. Zhang and L. Zheng. Delude remote operating system (OS) scan by honeyd. In *Computer Science and Engineering, 2009. WCSE '09. Second International Workshop on*, volume 2, pages 503–506, Oct 2009.

[8] C. Zimmer, B. Bhat, F. Mueller, and S. Mohan. Time-based intrusion detection in cyber-physical systems. In *Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems*, ICCPS '10, pages 109–118, New York, NY, USA, 2010. ACM.