Validation of NMAP's Network Behavior using Wireshark

Daniel Cruz Ramírez
Master of Engineering in Computer Engineering
Dr. Jeffrey Duffany
Electrical and Computer Engineering and Computer Science Department
Polytechnic University of Puerto Rico

Abstract *— NMAP is used to actively scan networks using different ping techniques. There is not much information available on how NMAP works besides its website. Although the program states how it works, there is little validation of its functionality. Wireshark, a network protocol analyzer, was used to validate these features in a test system environment: ping scans, OS detection, including port scanning and version detection. Among NMAP's weaknesses, we find it relies on an OS Database that should be updated regularly to be able to detect new operating systems and that its scans produce a large number of packets, which might cause detection of the scan in a properly protected network environment. NMAP's OS Database can also be used to simulate operating systems for network scans, such as in a honeypot, using a program called honeyd. Any scan in a foreign network environment should be corroborated with other tools, passively if possible.*

Key Terms *— NMAP, Ping Scan, Remote OS Detection, Wireshark.*

NMAP

NMAP ("Network Mapper") is a free and open source (license) utility for network discovery and security auditing. Many systems and network administrators also find it useful for tasks such as network inventory, managing service upgrade schedules, and monitoring host or service uptime. NMAP uses raw IP packets in novel ways to determine what hosts are available on the network, what services (application name and version) those hosts are offering, what operating systems (and OS versions) they are running, what type of packet filters/firewalls are in use, and dozens of other characteristics. It was designed to rapidly scan large networks, but works fine against single hosts. NMAP runs on all major computer operating systems, and official binary packages are available for Linux, Windows, and Mac OS X. In addition to the classic command-line NMAP executable, the NMAP suite includes an advanced GUI and results viewer (ZeNMAP), a flexible data transfer, redirection, and debugging tool (Ncat), a utility for comparing scan results (Ndiff), and a packet generation and response analysis tool (Nping) [1].

NMAP possesses the following ping scanning techniques:

- TCP SYN
- TCP ACK
- UDP
- ICMP
- Ping sweep
- ARP
- Broadcast

WIRESHARK

Wireshark is the world's foremost and widely-used network protocol analyzer. It lets you see what's happening on your network at a microscopic level and is the de facto (and often de jure) standard across many commercial and non-profit enterprises, government agencies, and educational institutions. Wireshark development thrives thanks to the volunteer contributions of networking experts around the globe and is the continuation of a project started by Gerald Combs in 1998 [2].

Test Setup for Ping Scanning Techniques

There is little information regarding how NMAP works besides the documentation on NMAP's website. Although the program is open source, no invasive study exists about how the program performs and what information is sends with each scan attempt.

For this purpose, we setup a test network for this project which includes a router, a Dell Inspiron 5720 laptop running Ubuntu Mate 16.10 x64 with Linux kernel 4.8.0.30-generic, an Internet Router, and a Raspberry Pi 3 Model B, running Raspbian Jessie Lite and Linux kernel 4.4.34-v7+. Both operating systems updated as of December 12, 2016. There are no other network objects.

A Raspberry Pi is a credit card-sized computer originally designed for education, inspired by the 1981 BBC Micro. The Raspberry Pi 3 is the third generation Raspberry Pi. It has:

- A 1.2GHz 64-bit quad-core ARMv8 CPU
- 802.11n Wireless LAN (not used in the project)
- Bluetooth 4.1
- Bluetooth Low Energy (BLE)
- 1GB RAM
- 4 USB ports
- 40 GPIO pins
- Full HDMI port
- Ethernet port
  - MAC Address - B8:27:EB:CF:6C:77
  - IP Address – 10.0.0.100
- Combined 3.5mm audio jack and composite video
- Camera interface (CSI)
- Display interface (DSI)
- Micro SD card slot
- VideoCore IV 3D graphics core [3]

The Dell Laptop's specifications are:

- CPU: Intel Core i7-3632QM
- Graphics:
  - Intel 3rd Gen Core Processor Graphics Controller
  - NVIDIA GF117M [GeForce 610M/710M/810M/820M / GT 620M/625M/630M/720M]
- Audio: Intel 7 Series/C210 Series Family High Definition Audio Controller
- Network:
  - Intel Centrino Wireless-N 2230 (disabled)
  - Realtek RTL8101/2/6E PCI Express Fast/Gigabit Ethernet controller
    - MAC Address – 5C:F9:DD:4F:0C:56
    - IP Address – 10.0.0.11
- RAM: 12GB DDR3 10600

All ping scanning techniques are done from the Dell Laptop to the Raspberry Pi 3 Model B. To minimize communication, the Dell Laptop has no gateway or DNS information and, whenever possible, there are no other programs running. Both NMAP and Wireshark are run from the Dell laptop. Although NMAP has a GUI (ZENMAP), we have used the command line version only. Wireshark is used to analyze NMAP's output.

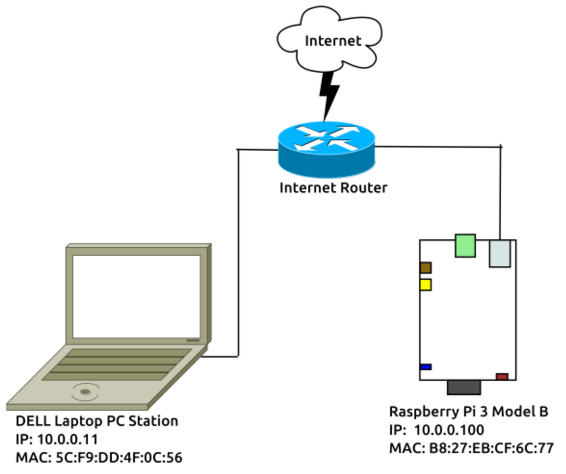


Figure 1
Test Network Setup

This project was started with NMAP version 6.40 and Wireshark version 2.2.3, distributed with Ubuntu Mate 16.10. As it progressed, newer NMAP versions came out and were tested. Finally, we settled on NMAP version 7.31, compiled from source.

For the OS detection scans, although we have primarily used the TEST SETUP, we have also tested several other systems, without any modification to their LAN structure, to determine NMAP's efficiency.

TCP SYN Ping Scan: *map -sP -PS <target>*

For this type of ping scan, NMAP sends a TCP SYN packet to port 80. If the port is closed, the host responds with an RST packet. If the port is open, the host responds with a TCP SYN/ACK packet indicating that a connection can be established. Afterwards, an RST packet is sent to reset this connection [1].

For this scan, we expect to find a SYN/ACK packet from the source ip to the destination ip.

TCP ACK Ping Scan: NMAP -sP -PA <target>

This type of scan can be used to detect hosts that block SYN packets or ICMP echo requests, but it will most likely be blocked by modern firewalls that track connection states. NMAP sends an empty TCP packet with the ACK flag set to port 80, if the host is offline, it should not respond to this request, if the host is online, it returns an RST packet, since the connection does not exist. TCP ACK ping scans need to run as a privileged user, otherwise a system call connect() is used to send an empty TCP SYN packet [1].

For this scan, we expect to find an ACK packet to port 80 from the source IP to the destination IP.

UDP Ping Scans: NMAP -sP -PU <target>

UDP is a minimal message-oriented transport layer protocol that is documented in RFC 768. UDP provides no guarantees to the upper layer protocol for message delivery and the UDP layer retains no state of UDP messages once sent [4]. UDP ping scans have the advantage of being capable of detecting systems behind firewalls with strict TCP filtering leaving the UDP traffic forgotten. NMAP is one of a handful of programs capable of doing a UDP ping scan. NMAP sends an empty UDP packet to ports 31 and 338. If the host is responding, it should return an ICMP port unreachable error. If the host is offline, various ICMP error messages could be returned [1].

For this type of scan, we expect to find a UDP broadcast packet from the source machine, expecting a response from the target machine.

ICMP Ping Scans: NMAP -sP -PE <target>

ICMP messages are typically used for diagnostic or control purposes or generated in response to errors in IP operations (as specified in RFC 1122). ICMP errors are directed to the source IP address of the originating packet. ICMP ping scans use these types of packets to determine if a host is active or not [1].

For this type of scan, we expect to find ICMP ping responses from the target to the source of the scan, along with the time for the ping to return.

Ping Sweeps: NMAP -sP -PO <target>

This technique tries sending different packets using different IP protocols, hoping to get a response indicating that a host is online. By default, this ping scan will use the protocols IGMP, IP-in-IP, and ICMP to try to obtain a response that will indicate that the host is online. Using *--packet-trace* will show more details of what happened behind the curtains [1].

For this type of scan, we expect to find a ping responses from the target to the source of the scan, along with the time for the ping to return.

ARP Ping Scans: NMAP -sP -PR <target>

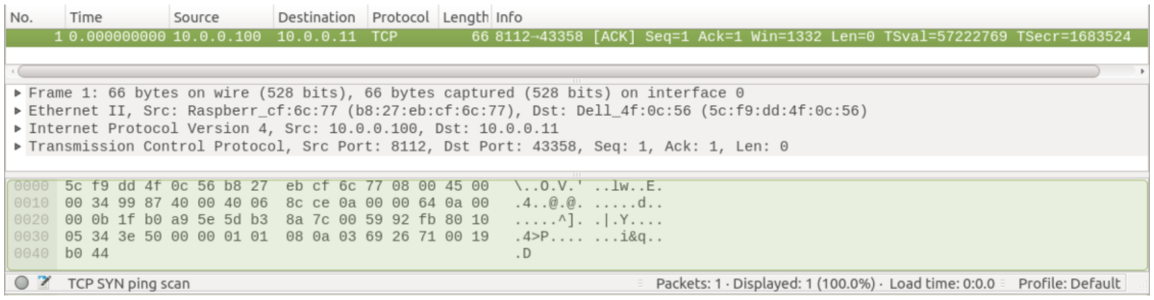
IPv4 devices must respond to ARP packets even if the targeted device uses firewalls or other stealthy methods to hide from ICMP or UDP packet based ping tools. ARP, a non-routable protocol, operates at OSI Layer 2. The IPv4 address of the device must be known. In this scan, ARP requests are sent to the target, if the host responds with an ARP reply, it's online [1].

For this type of scan, we expect to find an arp responses from the target to the source of the scan.

Broadcast Pings: NMAP -script broadcast-ping <target>

Broadcast pings send ICMP echo requests to the local broadcast address, and then waiting for hosts to reply with an ICMP echo reply. This a nice way of discovering hosts in a network without sending probes to the other hosts [1].

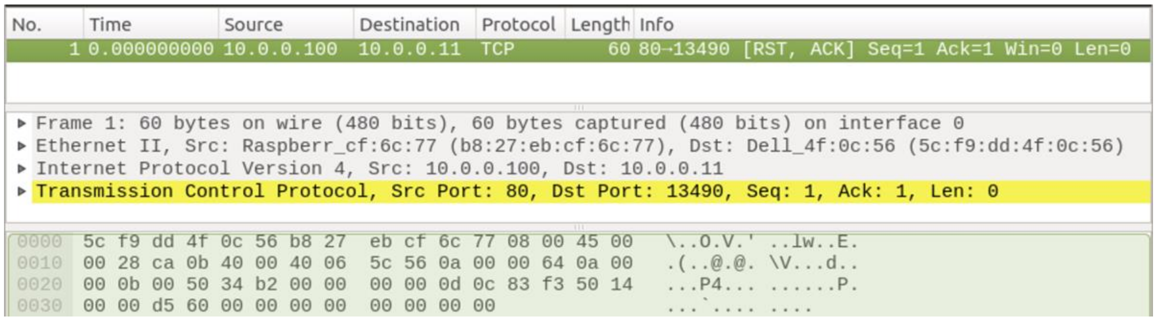
For this type of scan, we expect to find ICMP echo requests throughout the local broadcast address, along with ICMP echo replies.



```
No.  Time        Source      Destination Protocol Length Info
   1 0.000000000 10.0.0.100  10.0.0.11   TCP          66 8112→43358 [ACK] Seq=1 Ack=1 Win=1332 Len=0 TSval=57222769 TSecr=1683524

▸ Frame 1: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
▸ Ethernet II, Src: Raspberr_cf:6c:77 (b8:27:eb:cf:6c:77), Dst: Dell_4f:0c:56 (5c:f9:dd:4f:0c:56)
▸ Internet Protocol Version 4, Src: 10.0.0.100, Dst: 10.0.0.11
▸ Transmission Control Protocol, Src Port: 8112, Dst Port: 43358, Seq: 1, Ack: 1, Len: 0

0000  5c f9 dd 4f 0c 56 b8 27  eb cf 6c 77 08 00 45 00   \..O.V.' ..lw..E.
0010  00 34 99 87 40 00 40 06  8c ce 0a 00 00 64 0a 00   .4..@.@. .....d..
0020  00 0b 1f b0 a9 5e 5d b3  8a 7c 00 59 92 fb 80 10   .....^]. .|.Y....
0030  05 34 3e 50 00 00 01 01  08 0a 03 69 26 71 00 19   .4>P.... ...i&q..
0040  b0 44                                              .D

TCP SYN ping scan                    Packets: 1 · Displayed: 1 (100.0%) · Load time: 0:0.0   Profile: Default
```

Figure 2

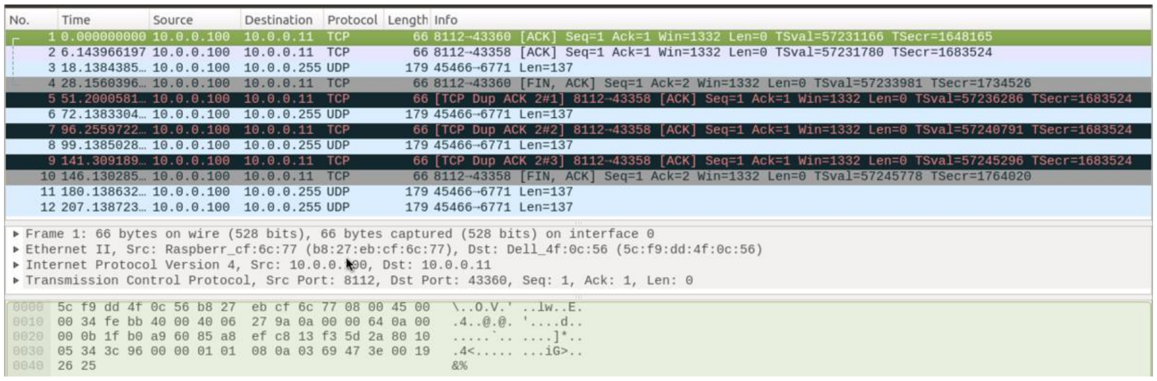
Wireshark TCP SYN Ping Scan Results



```
No.  Time        Source      Destination Protocol Length Info
   1 0.000000000 10.0.0.100  10.0.0.11   TCP          60 80→13490 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

▸ Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▸ Ethernet II, Src: Raspberr_cf:6c:77 (b8:27:eb:cf:6c:77), Dst: Dell_4f:0c:56 (5c:f9:dd:4f:0c:56)
▸ Internet Protocol Version 4, Src: 10.0.0.100, Dst: 10.0.0.11
▸ Transmission Control Protocol, Src Port: 80, Dst Port: 13490, Seq: 1, Ack: 1, Len: 0

0000  5c f9 dd 4f 0c 56 b8 27  eb cf 6c 77 08 00 45 00   \..O.V.' ..lw..E.
0010  00 28 ca 0b 40 00 40 06  5c 56 0a 00 00 64 0a 00   .(..@.@. \V...d..
0020  00 0b 00 50 34 b2 00 00  00 00 0d 0c 83 f3 50 14   ...P4... ......P.
0030  00 00 d5 60 00 00 00 00  00 00 00 00               ...`.... ....
```

Figure 3

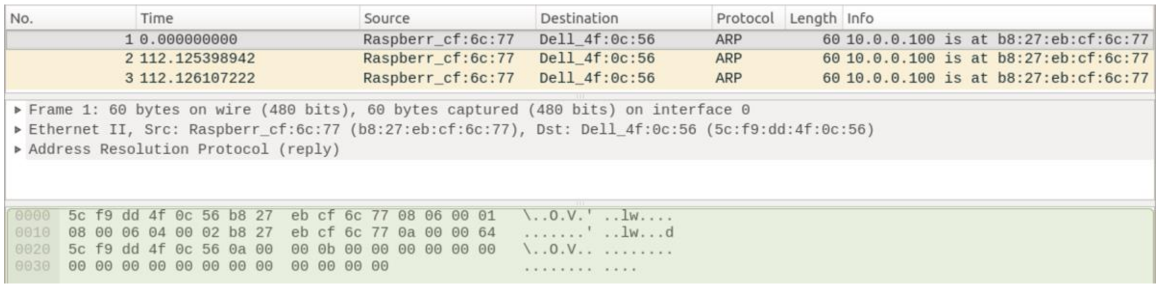
Wireshark TCP ACK Ping Scan Results



```
No.  Time          Source      Destination Protocol Length Info
   1 0.000000000   10.0.0.100  10.0.0.11   TCP          66 8112→43360 [ACK] Seq=1 Ack=1 Win=1332 Len=0 TSval=57231166 TSecr=1648165
   2 6.143966197   10.0.0.100  10.0.0.11   TCP          66 8112→43358 [ACK] Seq=1 Ack=1 Win=1332 Len=0 TSval=57231780 TSecr=1683524
   3 18.1384385…   10.0.0.100  10.0.0.255  UDP         179 45466→6771 Len=137
   4 28.1560396…   10.0.0.100  10.0.0.11   TCP          66 8112→43360 [FIN, ACK] Seq=1 Ack=2 Win=1332 Len=0 TSval=57233981 TSecr=1734526
   5 51.2000581…   10.0.0.100  10.0.0.11   TCP          66 [TCP Dup ACK 2#1] 8112→43358 [ACK] Seq=1 Ack=1 Win=1332 Len=0 TSval=57236286 TSecr=1683524
   6 72.1383304…   10.0.0.100  10.0.0.255  UDP         179 45466→6771 Len=137
   7 96.2559722…   10.0.0.100  10.0.0.11   TCP          66 [TCP Dup ACK 2#2] 8112→43358 [ACK] Seq=1 Ack=1 Win=1332 Len=0 TSval=57240791 TSecr=1683524
   8 99.1385028…   10.0.0.100  10.0.0.255  UDP         179 45466→6771 Len=137
   9 141.309189…   10.0.0.100  10.0.0.11   TCP          66 [TCP Dup ACK 2#3] 8112→43358 [ACK] Seq=1 Ack=1 Win=1332 Len=0 TSval=57245296 TSecr=1683524
  10 146.130285…   10.0.0.100  10.0.0.11   TCP          66 8112→43358 [FIN, ACK] Seq=1 Ack=2 Win=1332 Len=0 TSval=57245778 TSecr=1764020
  11 180.138632…   10.0.0.100  10.0.0.255  UDP         179 45466→6771 Len=137
  12 207.138723…   10.0.0.100  10.0.0.255  UDP         179 45466→6771 Len=137

▸ Frame 1: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
▸ Ethernet II, Src: Raspberr_cf:6c:77 (b8:27:eb:cf:6c:77), Dst: Dell_4f:0c:56 (5c:f9:dd:4f:0c:56)
▸ Internet Protocol Version 4, Src: 10.0.0.100, Dst: 10.0.0.11
▸ Transmission Control Protocol, Src Port: 8112, Dst Port: 43360, Seq: 1, Ack: 1, Len: 0

0000  5c f9 dd 4f 0c 56 b8 27  eb cf 6c 77 08 00 45 00   \..O.V.' ..lw..E.
0010  00 34 fe bb 40 00 40 06  27 9a 0a 00 00 64 0a 00   .4..@.@. '....d..
0020  00 0b 1f b0 a9 60 85 a8  ef c8 13 f3 5d 2a 80 10   .....`.. ....]*..
0030  05 34 3c 96 00 00 01 01  08 0a 03 69 47 3e 00 19   .4<..... ...iG>..
0040  26 25                                              &%
```

Figure 4

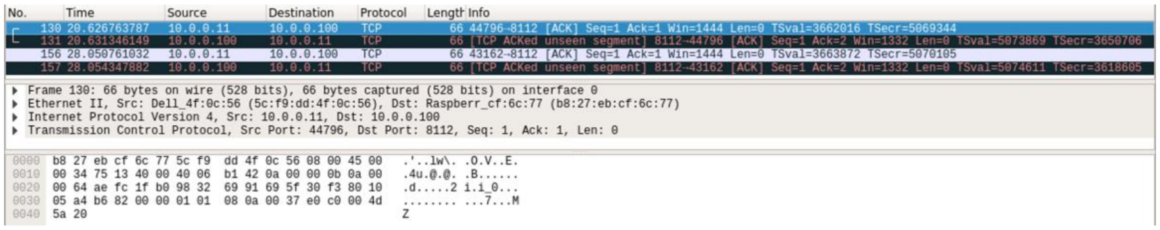
Wireshark UDP Ping Scan Results



```
No.  Time          Source             Destination       Protocol Length Info
   1 0.000000000   Raspberr_cf:6c:77  Dell_4f:0c:56     ARP          60 10.0.0.100 is at b8:27:eb:cf:6c:77
   2 112.125398942 Raspberr_cf:6c:77  Dell_4f:0c:56     ARP          60 10.0.0.100 is at b8:27:eb:cf:6c:77
   3 112.126107222 Raspberr_cf:6c:77  Dell_4f:0c:56     ARP          60 10.0.0.100 is at b8:27:eb:cf:6c:77

▸ Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▸ Ethernet II, Src: Raspberr_cf:6c:77 (b8:27:eb:cf:6c:77), Dst: Dell_4f:0c:56 (5c:f9:dd:4f:0c:56)
▸ Address Resolution Protocol (reply)

0000  5c f9 dd 4f 0c 56 b8 27  eb cf 6c 77 08 06 00 01   \..O.V.' ..lw....
0010  08 00 06 04 00 02 b8 27  eb cf 6c 77 0a 00 00 64   .......' ..lw...d
0020  5c f9 dd 4f 0c 56 0a 00  00 0b 00 00 00 00 00 00   \..O.V.. ........
0030  00 00 00 00 00 00 00 00  00 00 00 00               ........ ....
```

Figure 5

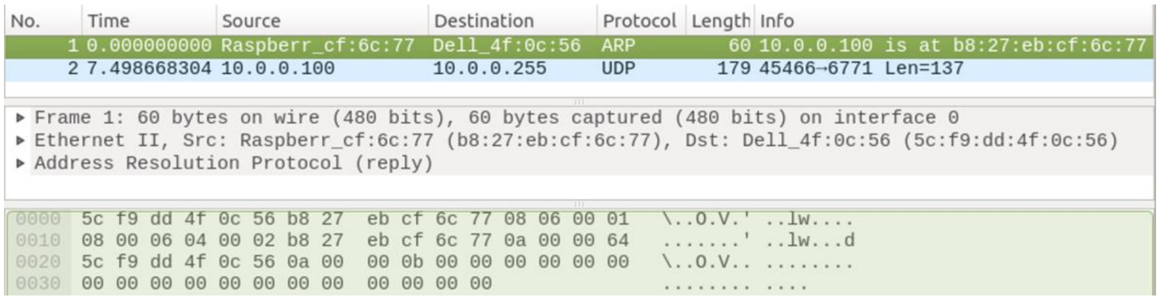
Wireshark ICMP Ping Scan Results



```
No.    Time          Source       Destination   Protocol  Length Info
  130 20.626763787  10.0.0.11    10.0.0.100    TCP         66 44796→8112 [ACK] Seq=1 Ack=1 Win=1444 Len=0 TSval=3662016 TSecr=5069344
  131 20.631346149  10.0.0.100   10.0.0.11     TCP         66 [TCP ACKed unseen segment] 8112→44796 [ACK] Seq=1 Ack=2 Win=1332 Len=0 TSval=5073869 TSecr=3650706
  156 28.050761032  10.0.0.11    10.0.0.100    TCP         66 43162→8112 [ACK] Seq=1 Ack=1 Win=1444 Len=0 TSval=3663872 TSecr=5070105
  157 28.054347882  10.0.0.100   10.0.0.11     TCP         66 [TCP ACKed unseen segment] 8112→43162 [ACK] Seq=1 Ack=2 Win=1332 Len=0 TSval=5074611 TSecr=3618605

▸ Frame 130: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
▸ Ethernet II, Src: Dell_4f:0c:56 (5c:f9:dd:4f:0c:56), Dst: Raspberr_cf:6c:77 (b8:27:eb:cf:6c:77)
▸ Internet Protocol Version 4, Src: 10.0.0.11, Dst: 10.0.0.100
▸ Transmission Control Protocol, Src Port: 44796, Dst Port: 8112, Seq: 1, Ack: 1, Len: 0

0000  b8 27 eb cf 6c 77 5c f9  dd 4f 0c 56 08 00 45 00   .'..lw\. .O.V..E.
0010  00 34 75 13 40 00 40 06  b1 42 0a 00 00 0b 0a 00   .4u.@.@. .B......
0020  00 64 ae fc 1f b0 98 32  69 91 69 5f 30 f3 80 10   .d.....2 i.i_0...
0030  05 a4 b6 82 00 00 01 01  08 0a 00 37 e0 c0 00 4d   ........ ...7...M
0040  5a 20                                              Z
```

Figure 6

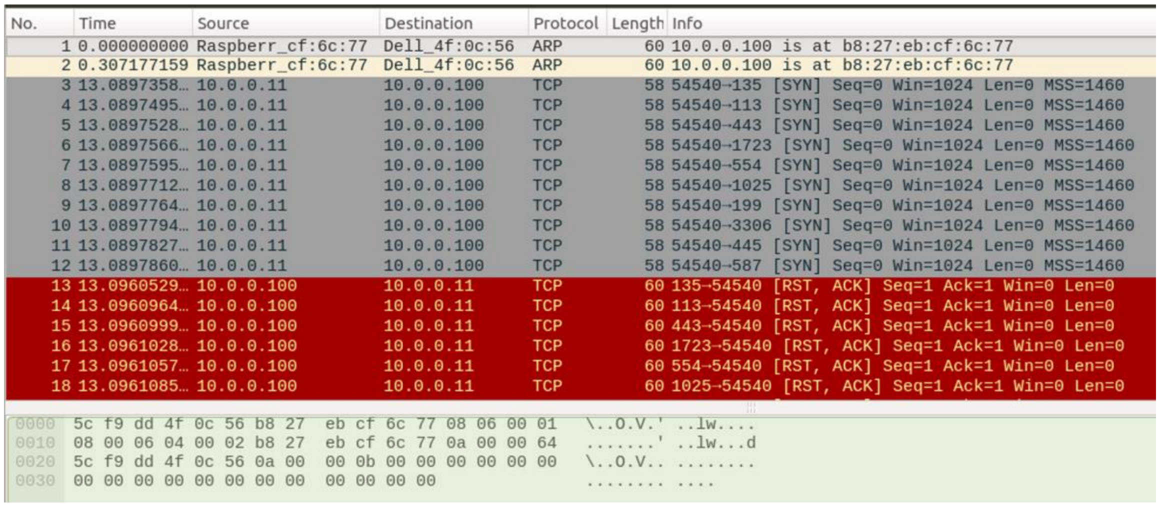
Wireshark Ping Sweep Scan Results



```
No.   Time          Source              Destination     Protocol  Length Info
    1 0.000000000   Raspberr_cf:6c:77   Dell_4f:0c:56   ARP           60 10.0.0.100 is at b8:27:eb:cf:6c:77
    2 7.498668304   10.0.0.100          10.0.0.255      UDP          179 45466→6771 Len=137

▸ Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▸ Ethernet II, Src: Raspberr_cf:6c:77 (b8:27:eb:cf:6c:77), Dst: Dell_4f:0c:56 (5c:f9:dd:4f:0c:56)
▸ Address Resolution Protocol (reply)

0000  5c f9 dd 4f 0c 56 b8 27  eb cf 6c 77 08 06 00 01   \..O.V.' ..lw....
0010  08 00 06 04 00 02 b8 27  eb cf 6c 77 0a 00 00 64   .......' ..lw...d
0020  5c f9 dd 4f 0c 56 0a 00  00 0b 00 00 00 00 00 00   \..O.V.. ........
0030  00 00 00 00 00 00 00 00  00 00 00 00               ........ ....
```

Figure 7

Wireshark ARP Ping Scan Results



```
No.  Time          Source             Destination    Protocol Length Info
   1 0.000000000   Raspberr_cf:6c:77  Dell_4f:0c:56  ARP          60 10.0.0.100 is at b8:27:eb:cf:6c:77
   2 0.307177159   Raspberr_cf:6c:77  Dell_4f:0c:56  ARP          60 10.0.0.100 is at b8:27:eb:cf:6c:77
   3 13.0897358…   10.0.0.11          10.0.0.100     TCP          58 54540→135 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
   4 13.0897495…   10.0.0.11          10.0.0.100     TCP          58 54540→113 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
   5 13.0897528…   10.0.0.11          10.0.0.100     TCP          58 54540→443 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
   6 13.0897566…   10.0.0.11          10.0.0.100     TCP          58 54540→1723 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
   7 13.0897595…   10.0.0.11          10.0.0.100     TCP          58 54540→554 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
   8 13.0897712…   10.0.0.11          10.0.0.100     TCP          58 54540→1025 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
   9 13.0897764…   10.0.0.11          10.0.0.100     TCP          58 54540→199 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
  10 13.0897794…   10.0.0.11          10.0.0.100     TCP          58 54540→3306 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
  11 13.0897827…   10.0.0.11          10.0.0.100     TCP          58 54540→445 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
  12 13.0897860…   10.0.0.11          10.0.0.100     TCP          58 54540→587 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
  13 13.0960529…   10.0.0.100         10.0.0.11      TCP          60 135→54540 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
  14 13.0960964…   10.0.0.100         10.0.0.11      TCP          60 113→54540 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
  15 13.0960999…   10.0.0.100         10.0.0.11      TCP          60 443→54540 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
  16 13.0961028…   10.0.0.100         10.0.0.11      TCP          60 1723→54540 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
  17 13.0961057…   10.0.0.100         10.0.0.11      TCP          60 554→54540 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
  18 13.0961085…   10.0.0.100         10.0.0.11      TCP          60 1025→54540 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

0000  5c f9 dd 4f 0c 56 b8 27  eb cf 6c 77 08 06 00 01   \..O.V.' ..lw....
0010  08 00 06 04 00 02 b8 27  eb cf 6c 77 0a 00 00 64   .......' ..lw...d
0020  5c f9 dd 4f 0c 56 0a 00  00 0b 00 00 00 00 00 00   \..O.V.. ........
0030  00 00 00 00 00 00 00 00  00 00 00 00               ........ ....
```

Figure 8

Broadcast Ping Scan Results

OS Detection using NMAP: NMAP -O -v <target>

NMAP OS fingerprinting works by sending up to 16 TCP, UDP, and ICMP probes to known open and closed ports of the target machine. These probes are specially designed to exploit various ambiguities in the standard protocol RFCs. Then NMAP listens for responses. Dozens of attributes in those responses are analyzed and combined to generate a fingerprint. Every probe packet is tracked and resent at least once if there is no response. All of the packets are IPv4 with a random IP ID value. Probes to an open TCP port are skipped if no such port has been found. For closed TCP or UDP ports, NMAP will first check if such a port has been found. If not, NMAP will just pick a port at random and hope for the best [5].

NMAP then compares the results to its OS database and prints out the OS details if there is a match. Each fingerprint includes a freeform textual description of the OS, and a classification which provides the vendor name, underlying OS, OS generation, and device type [6].

OS detection enables some other tests which make use of information that is gathered anyway during the process:

- Device type – if several device types are shown, they will be separated with the pipe symbol.
- Running - shows the OS Family and OS generation if available.
- OS CPE - shows a Common Platform Enumeration (CPE) representation of the operating system when available.
- OS details - gives the detailed description for each fingerprint that matches.

  Uptime guess.
- Network Distance – how many routers are between it and a target host? The distance is zero when you are scanning localhost, and one for a machine on the same network segment. Each additional router on the path adds one to the hop count.
- TCP Sequence Prediction - Systems with poor TCP initial sequence number generation are vulnerable to blind TCP spoofing attacks. In other words, you can make a full connection to those systems and send (but not receive) data while spoofing a different IP address.
- IP ID sequence generation - This field describes the ID generation algorithm that NMAP was able to discern [6].

Interrogating open ports for clues is another effective approach for OS fingerprinting. Some applications, such as Microsoft IIS, only run on a single platform, while many other apps divulge their platform in overly verbose banner messages. Adding the *-sV* option enables NMAP version detection, which is trained to look for these clues, among others [6].

TCP/IP fingerprinting will identify the proxy, while version scanning will generally detect the server running the proxied application. Even when no proxying or port forwarding is involved, using both techniques is beneficial. If they come out the same, that makes the results more credible. If they come out wildly different, further investigation must determine what is going on before relying on either. Since OS and version detection go together so well, the *-A* option enables them both [6].

When NMAP performs OS detection against a target and fails to find a perfect match, it usually repeats the attempt: by default, it tries five times if conditions are favorable for OS fingerprint submission, and twice when conditions aren't so good [6].

Besides the OS Database, NMAP also has a MAC Address database which maps MAC address prefixes to vendor names. Ethernet devices are each programmed with a unique 48-bit identifier known as a MAC address. This address is placed in Ethernet headers to identify which machine on a local network sent a packet, and which machine the packet is destined for. To assure that MAC addresses are unique in a world with thousands of vendors, the IEEE assigns an Organizationally Unique Identifier (OUI) to each company manufacturing Ethernet devices. The company must use its own OUI for the first three bytes of MAC addresses for equipment it produces. It can choose the remaining three bytes however it wishes, as long as they are unique. NMAP can determine the MAC address of hosts on a local Ethernet LAN by reading the headers off the wire. It uses this database to look up and report the manufacturer name based on the OUI. This can be useful for roughly identifying the type of machine being dealt with [1].

Version Detection using NMAP: NMAP -O -sV <target>

Version detection is one of the most popular features of NMAP. Knowing the exact version of a service is highly valuable for penetration testers who use this service to look for security vulnerabilities, and for system administrators who wish to monitor their networks for any unauthorized changes. Fingerprinting a service may also reveal additional information about a target, such as available modules and specific protocol information [1].

NMAP has a special flag to activate aggressive detection, *-A*. Aggressive mode enables OS detection (*-O*), version detection (*-sV*), script scanning (*-sC*), and traceroute (*--traceroute*). This mode sends a lot more probes and it is more likely to be detected, but provides a lot of valuable host information [1].

Results ultimately come from the target machine itself. Numerous reconnaissance methods to explore a network should be used to confirm the information needed, you should not trust only one of them [1].

OS Detection using a Simple Ping

Below are some typical initial TTL values and window sizes of common operating systems:

Table 1
TTL and TCP Window Size for Different Operating Systems [7]

Operating System (OS)	IP Initial TTL	TCP window size
Linux (kernel 2.4 and 2.6)	64	5840
Google's Android and Chrome OS	64	5720
FreeBSD	64	65535
Windows XP	128	65535
Windows 7, Vista and Server 2008	128	8192
Cisco Router (IOS 12.4)	255	4128
MacOS/MacTCP X (10.5.6)	64	

In Windows Vista and 2008 server, Microsoft introduced a new TCP/IP stack with a number of improvements. It also includes a concept called TCP Window "Auto-Tuning" that's been used in Linux for years [8].

One reason for why the TTL and window size values varies between different OS's is because the RFC's for TCP and IP do not require implementations to use any particular default value for these fields. There is, however, a recommendation in RFC 1700 saying: “The current recommended default time to live (TTL) for the Internet Protocol (IP) is 64.” This recommendation is obviously not followed in many IP implementations [7].

Basically, from a simple ping, if ttl=64, the system pinged is Linux/Unix-based; else if ttl=128, the system pinged is Microsoft based; else if ttl=255, the system pinged is CISCO based. Current Apple Operating Systems are based on Unix. [9]

OS Detection on Project System

OS Detection of the Raspberry Pi 3 failed with NMAP version 6.40. I believe this to be because the shipping version of NMAP in Ubuntu Mate 16.10 was released on July 30, 2013 [1], before the introduction of the Raspberry Pi 3 (February 29, 2016) [3].

With version 7.30 (released on September 29, 2016) [1], however, OS Detection of the Raspberry Pi 3 takes about 15 seconds. It correctly identifies the operating system as a Linux 3.2 – 4.4. It does this with a combination of several methods: port scanning, arp ping response time, and MAC Address identification. In this case, since Raspberry Pis only run either Linux or Windows 10 IoT Core [3], the program uses the open ports and the ping response to correctly identify the operating system.

NMAP Output:

```
Starting NMAP 7.31 (https://NMAP.org) at 2017-01-16 13:43 AST
NMAP scan report for 10.0.0.100
Host is up (0.0044s latency).
Not shown: 998 closed ports
PORT     STATE SERVICE VERSION
53/tcp   open domain
MAC Address: B8:27:EB:CF:6C:77 (Raspberry Pi Foundation)
```

Device type: general purpose
Running: Linux 3.X|4.X
OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/o:linux:linux_kernel:4
OS details: Linux 3.2 - 4.4
Network Distance: 1 hop

OS Detection on Other Systems

NMAP version 7.31 was tested on several systems, identifying them correctly: Windows 2003 R2, Windows 2008 R2, Windows 10 Professional, Windows 7 Professional, Linux kernel XXX, among others. The only time NMAP failed was against a Windows 2016 Data Center Edition Virtual Machine, in which it guesses as being: Microsoft Windows Server 2012 or Windows Server 2012 R2 (93%), Microsoft Windows Server 2012 R2 (88%), Microsoft Windows 10 build 10586 - 14393 (87%), Microsoft Windows 7 Professional (87%), Microsoft Windows Phone 7.5 or 8.0 (86%), Microsoft Windows 10 build 10586 (86%), Microsoft Windows Server 2008 R2 or Windows 8.1 (86%), Microsoft Windows 7 Professional or Windows 8 (86%), Microsoft Windows Vista SP0 or SP1, Windows Server 2008 SP1, or Windows 7 (86%), Microsoft Windows Vista SP2, Windows 7 SP1, or Windows Server 2008 (86%). Since it detects it as both Windows 10 and Windows Server 2012, you could extrapolate that the system is the Server version of Windows 10. Windows Server 2016 was released on September 26, 2016.

If the conditions are not ideal, NMAP will not detect correctly the operating system. We added a Windows 10 system into out test setup. If we setup the LAN connection as private, NMAP version 7.31 detected the correct operating system. If the LAN connection was setup as Public (affecting the firewall), NMAP version 7.31 was unable to detect the operating system correctly and gave us the following message and the end of the scan: "Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port.

Aggressive OS guesses: Microsoft Windows 8.1 R1 (92%), Microsoft Windows Phone 7.5 or 8.0 (92%), Microsoft Windows Server 2008 or 2008 Beta 3 (92%), Microsoft Windows Server 2008 R2 or Windows 8.1 (92%), Microsoft Windows 7 Professional or Windows 8 (92%), Microsoft Windows Vista SP0 or SP1, Windows Server 2008 SP1, or Windows 7 (92%), Microsoft Windows Vista SP2, Windows 7 SP1, or Windows Server 2008 (92%), Microsoft Windows Embedded Standard 7 (92%), Microsoft Windows 7 (90%), Microsoft Windows Server 2008 SP1 (89%)

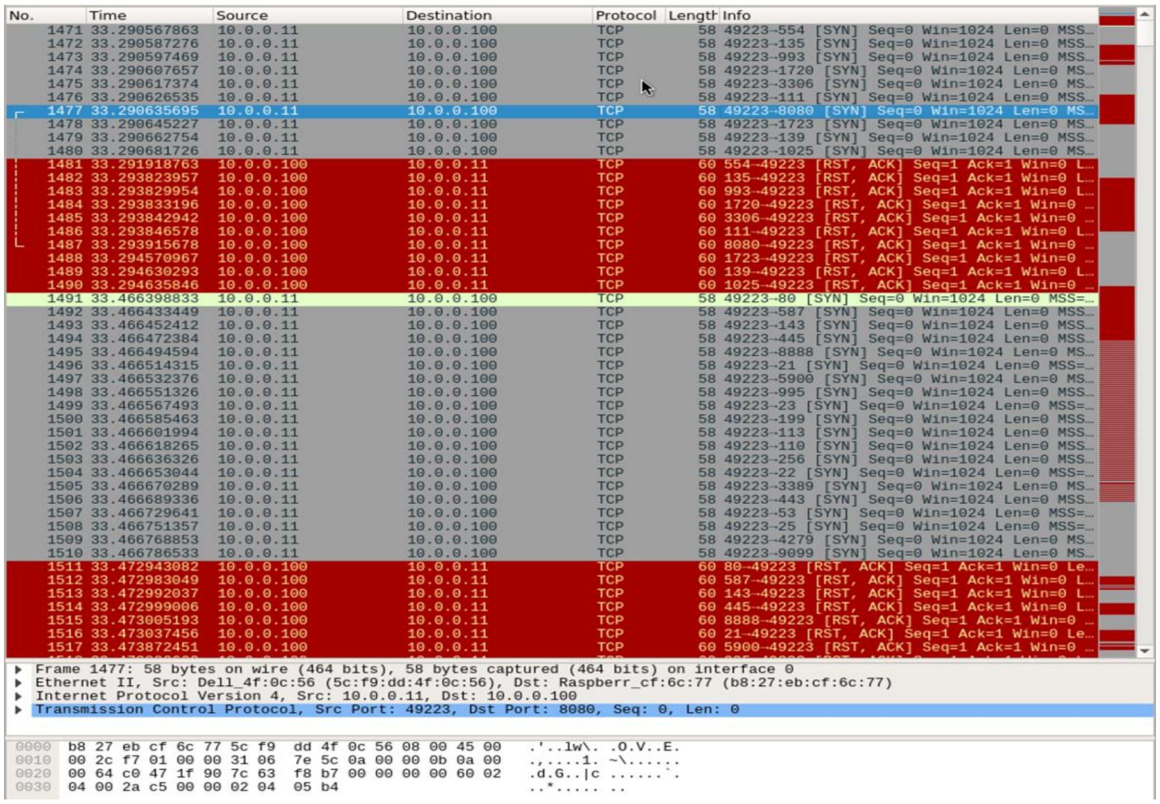
No exact OS matches for host (test conditions non-ideal)."

NMAP's OS Database must be continually updated to be able to detect new systems released. This can be done by downloading the database directly or recompiling from the sources from the project's Github Page [10].

Honeyd & Honeypots

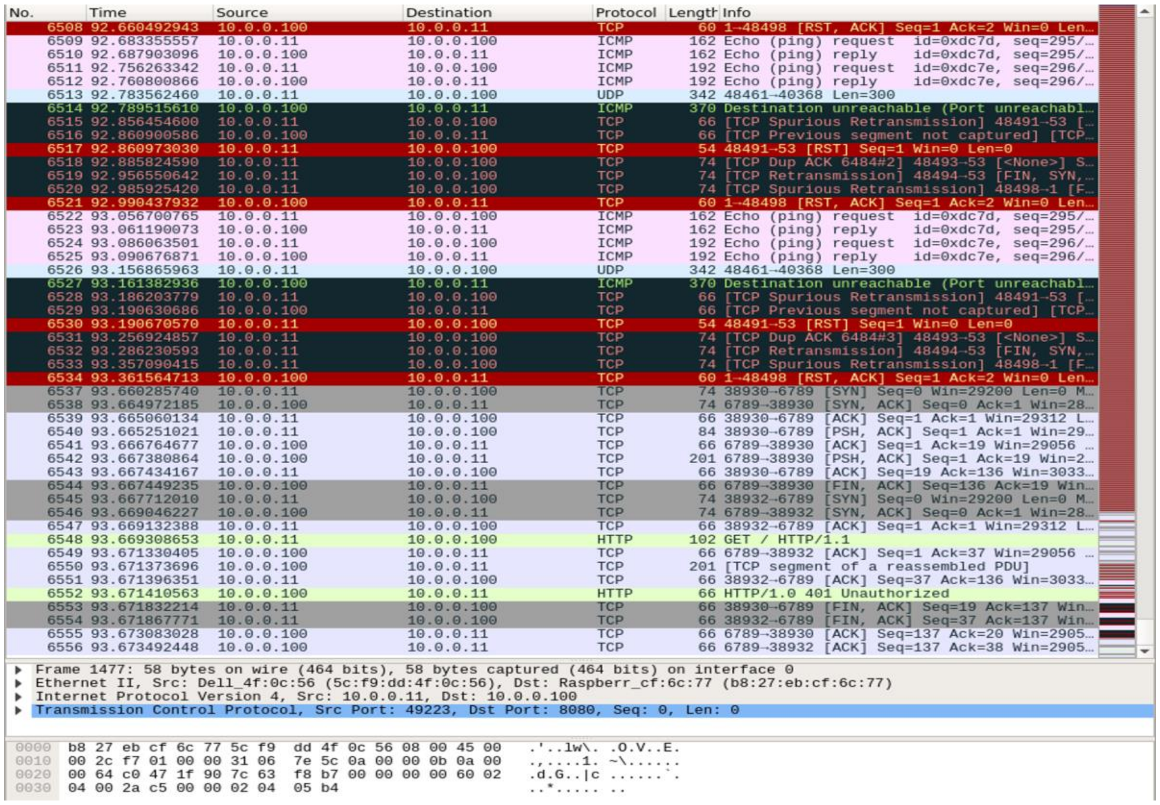
NMAP's OS database can be used with other purposes, such as creating virtual hosts on a network with other programs (a honeypot), such as HONEYD.

Honeyd is a small daemon that creates virtual hosts on a network. The hosts can be configured to run arbitrary services, and their personality can be adapted so that they appear to be running certain operating systems. Honeyd enables a single host to claim multiple addresses. Honeyd improves cyber security by providing mechanisms for threat detection and assessment. It also deters adversaries by hiding real systems in the middle of virtual systems. Honeyd can mimic several operating systems within a LAN from reading a NMAP fingerprint file. The configured personality is the operating system that NMAP will return. Personalities can be annotated to determine if they allow FIN-scans for open ports or to select the preference in which they reassemble fragmented IP packets [11].



```
No.   Time          Source      Destination  Protocol Length Info
 1471 33.290567863  10.0.0.11   10.0.0.100   TCP      58 49223→554 [SYN] Seq=0 Win=1024 Len=0 MSS…
 1472 33.290587276  10.0.0.11   10.0.0.100   TCP      58 49223→135 [SYN] Seq=0 Win=1024 Len=0 MSS…
 1473 33.290597469  10.0.0.11   10.0.0.100   TCP      58 49223→993 [SYN] Seq=0 Win=1024 Len=0 MSS…
 1474 33.290607657  10.0.0.11   10.0.0.100   TCP      58 49223→1720 [SYN] Seq=0 Win=1024 Len=0 MS…
 1475 33.290617374  10.0.0.11   10.0.0.100   TCP      58 49223→3306 [SYN] Seq=0 Win=1024 Len=0 MS…
 1476 33.290626535  10.0.0.11   10.0.0.100   TCP      58 49223→111 [SYN] Seq=0 Win=1024 Len=0 MSS…
 1477 33.290635695  10.0.0.11   10.0.0.100   TCP      58 49223→8080 [SYN] Seq=0 Win=1024 Len=0 MS…
 1478 33.290645227  10.0.0.11   10.0.0.100   TCP      58 49223→1723 [SYN] Seq=0 Win=1024 Len=0 MS…
 1479 33.290662754  10.0.0.11   10.0.0.100   TCP      58 49223→139 [SYN] Seq=0 Win=1024 Len=0 MSS…
 1480 33.290681726  10.0.0.11   10.0.0.100   TCP      58 49223→1025 [SYN] Seq=0 Win=1024 Len=0 MS…
 1481 33.291918763  10.0.0.100  10.0.0.11    TCP      60 554→49223 [RST, ACK] Seq=1 Ack=1 Win=0 L…
 1482 33.293823957  10.0.0.100  10.0.0.11    TCP      60 135→49223 [RST, ACK] Seq=1 Ack=1 Win=0 L…
 1483 33.293829954  10.0.0.100  10.0.0.11    TCP      60 993→49223 [RST, ACK] Seq=1 Ack=1 Win=0 L…
 1484 33.293833196  10.0.0.100  10.0.0.11    TCP      60 1720→49223 [RST, ACK] Seq=1 Ack=1 Win=0 …
 1485 33.293842942  10.0.0.100  10.0.0.11    TCP      60 3306→49223 [RST, ACK] Seq=1 Ack=1 Win=0 …
 1486 33.293846578  10.0.0.100  10.0.0.11    TCP      60 111→49223 [RST, ACK] Seq=1 Ack=1 Win=0 L…
 1487 33.293915678  10.0.0.100  10.0.0.11    TCP      60 8080→49223 [RST, ACK] Seq=1 Ack=1 Win=0 …
 1488 33.294570967  10.0.0.100  10.0.0.11    TCP      60 1723→49223 [RST, ACK] Seq=1 Ack=1 Win=0 …
 1489 33.294630293  10.0.0.100  10.0.0.11    TCP      60 139→49223 [RST, ACK] Seq=1 Ack=1 Win=0 L…
 1490 33.294635846  10.0.0.100  10.0.0.11    TCP      60 1025→49223 [RST, ACK] Seq=1 Ack=1 Win=0 …
 1491 33.466398833  10.0.0.11   10.0.0.100   TCP      58 49223→80 [SYN] Seq=0 Win=1024 Len=0 MSS=…
 1492 33.466433449  10.0.0.11   10.0.0.100   TCP      58 49223→587 [SYN] Seq=0 Win=1024 Len=0 MSS…
 1493 33.466452412  10.0.0.11   10.0.0.100   TCP      58 49223→143 [SYN] Seq=0 Win=1024 Len=0 MSS…
 1494 33.466472384  10.0.0.11   10.0.0.100   TCP      58 49223→445 [SYN] Seq=0 Win=1024 Len=0 MSS…
 1495 33.466494594  10.0.0.11   10.0.0.100   TCP      58 49223→8888 [SYN] Seq=0 Win=1024 Len=0 MS…
 1496 33.466514315  10.0.0.11   10.0.0.100   TCP      58 49223→21 [SYN] Seq=0 Win=1024 Len=0 MSS=…
 1497 33.466532376  10.0.0.11   10.0.0.100   TCP      58 49223→5900 [SYN] Seq=0 Win=1024 Len=0 MS…
 1498 33.466551326  10.0.0.11   10.0.0.100   TCP      58 49223→995 [SYN] Seq=0 Win=1024 Len=0 MSS…
 1499 33.466567493  10.0.0.11   10.0.0.100   TCP      58 49223→23 [SYN] Seq=0 Win=1024 Len=0 MSS=…
 1500 33.466585463  10.0.0.11   10.0.0.100   TCP      58 49223→199 [SYN] Seq=0 Win=1024 Len=0 MSS…
 1501 33.466601994  10.0.0.11   10.0.0.100   TCP      58 49223→113 [SYN] Seq=0 Win=1024 Len=0 MSS…
 1502 33.466618265  10.0.0.11   10.0.0.100   TCP      58 49223→110 [SYN] Seq=0 Win=1024 Len=0 MSS…
 1503 33.466636326  10.0.0.11   10.0.0.100   TCP      58 49223→256 [SYN] Seq=0 Win=1024 Len=0 MSS…
 1504 33.466653044  10.0.0.11   10.0.0.100   TCP      58 49223→22 [SYN] Seq=0 Win=1024 Len=0 MSS=…
 1505 33.466670289  10.0.0.11   10.0.0.100   TCP      58 49223→3389 [SYN] Seq=0 Win=1024 Len=0 MS…
 1506 33.466689336  10.0.0.11   10.0.0.100   TCP      58 49223→443 [SYN] Seq=0 Win=1024 Len=0 MSS…
 1507 33.466729641  10.0.0.11   10.0.0.100   TCP      58 49223→53 [SYN] Seq=0 Win=1024 Len=0 MSS=…
 1508 33.466751357  10.0.0.11   10.0.0.100   TCP      58 49223→25 [SYN] Seq=0 Win=1024 Len=0 MSS=…
 1509 33.466768853  10.0.0.11   10.0.0.100   TCP      58 49223→4279 [SYN] Seq=0 Win=1024 Len=0 MS…
 1510 33.466786533  10.0.0.11   10.0.0.100   TCP      58 49223→9099 [SYN] Seq=0 Win=1024 Len=0 MS…
 1511 33.472943082  10.0.0.100  10.0.0.11    TCP      60 80→49223 [RST, ACK] Seq=1 Ack=1 Win=0 Le…
 1512 33.472983049  10.0.0.100  10.0.0.11    TCP      60 587→49223 [RST, ACK] Seq=1 Ack=1 Win=0 L…
 1513 33.472992037  10.0.0.100  10.0.0.11    TCP      60 143→49223 [RST, ACK] Seq=1 Ack=1 Win=0 L…
 1514 33.472999006  10.0.0.100  10.0.0.11    TCP      60 445→49223 [RST, ACK] Seq=1 Ack=1 Win=0 L…
 1515 33.473005193  10.0.0.100  10.0.0.11    TCP      60 8888→49223 [RST, ACK] Seq=1 Ack=1 Win=0 …
 1516 33.473037456  10.0.0.100  10.0.0.11    TCP      60 21→49223 [RST, ACK] Seq=1 Ack=1 Win=0 Le…
 1517 33.473872451  10.0.0.100  10.0.0.11    TCP      60 5900→49223 [RST, ACK] Seq=1 Ack=1 Win=0 …

▸ Frame 1477: 58 bytes on wire (464 bits), 58 bytes captured (464 bits) on interface 0
▸ Ethernet II, Src: Dell_4f:0c:56 (5c:f9:dd:4f:0c:56), Dst: Raspberr_cf:6c:77 (b8:27:eb:cf:6c:77)
▸ Internet Protocol Version 4, Src: 10.0.0.11, Dst: 10.0.0.100
▸ Transmission Control Protocol, Src Port: 49223, Dst Port: 8080, Seq: 0, Len: 0

0000  b8 27 eb cf 6c 77 5c f9  dd 4f 0c 56 08 00 45 00   .'..lw\. .O.V..E.
0010  00 2c f7 01 00 00 31 06  7e 5c 0a 00 00 0b 0a 00   .,....1. ~\......
0020  00 64 c0 47 1f 90 7c 63  f8 b7 00 00 00 00 60 02   .d.G..|c ......`.
0030  04 00 2a c5 00 00 02 04  05 b4                     ..*..... ..
```

Figure 9
Test System Detection Start



```
No.   Time          Source      Destination  Protocol Length Info
 6508 92.660492943  10.0.0.100  10.0.0.11    TCP       60 1→48498 [RST, ACK] Seq=1 Ack=2 Win=0 Len…
 6509 92.683355557  10.0.0.11   10.0.0.100   ICMP     162 Echo (ping) request  id=0xdc7d, seq=295/…
 6510 92.687903096  10.0.0.100  10.0.0.11    ICMP     162 Echo (ping) reply    id=0xdc7d, seq=295/…
 6511 92.756263342  10.0.0.11   10.0.0.100   ICMP     192 Echo (ping) request  id=0xdc7e, seq=296/…
 6512 92.760800866  10.0.0.100  10.0.0.11    ICMP     192 Echo (ping) reply    id=0xdc7e, seq=296/…
 6513 92.783562460  10.0.0.11   10.0.0.100   UDP      342 48461→40368 Len=300
 6514 92.789515610  10.0.0.100  10.0.0.11    ICMP     370 Destination unreachable (Port unreachabl…
 6515 92.856454600  10.0.0.11   10.0.0.100   TCP       66 [TCP Spurious Retransmission] 48491→53 […
 6516 92.860900586  10.0.0.100  10.0.0.11    TCP       66 [TCP Previous segment not captured] [TCP…
 6517 92.860973030  10.0.0.11   10.0.0.100   TCP       54 48491→53 [RST] Seq=1 Win=0 Len=0
 6518 92.885824590  10.0.0.11   10.0.0.100   TCP       74 [TCP Dup ACK 6484#2] 48493→53 [<None>] S…
 6519 92.956550642  10.0.0.11   10.0.0.100   TCP       74 [TCP Retransmission] 48494→53 [FIN, SYN,…
 6520 92.985925420  10.0.0.11   10.0.0.100   TCP       74 [TCP Spurious Retransmission] 48498→1 [F…
 6521 92.990437932  10.0.0.100  10.0.0.11    TCP       60 1→48498 [RST, ACK] Seq=1 Ack=2 Win=0 Len…
 6522 93.056700765  10.0.0.11   10.0.0.100   ICMP     162 Echo (ping) request  id=0xdc7d, seq=295/…
 6523 93.061190073  10.0.0.100  10.0.0.11    ICMP     162 Echo (ping) reply    id=0xdc7d, seq=295/…
 6524 93.086063501  10.0.0.11   10.0.0.100   ICMP     192 Echo (ping) request  id=0xdc7e, seq=296/…
 6525 93.090676871  10.0.0.100  10.0.0.11    ICMP     192 Echo (ping) reply    id=0xdc7e, seq=296/…
 6526 93.156865963  10.0.0.11   10.0.0.100   UDP      342 48461→40368 Len=300
 6527 93.161382936  10.0.0.100  10.0.0.11    ICMP     370 Destination unreachable (Port unreachabl…
 6528 93.186203779  10.0.0.11   10.0.0.100   TCP       66 [TCP Spurious Retransmission] 48491→53 […
 6529 93.190630686  10.0.0.100  10.0.0.11    TCP       66 [TCP Previous segment not captured] [TCP…
 6530 93.190670570  10.0.0.11   10.0.0.100   TCP       54 48491→53 [RST] Seq=1 Win=0 Len=0
 6531 93.256924857  10.0.0.11   10.0.0.100   TCP       74 [TCP Dup ACK 6484#3] 48493→53 [<None>] S…
 6532 93.286230593  10.0.0.11   10.0.0.100   TCP       74 [TCP Retransmission] 48494→53 [FIN, SYN,…
 6533 93.357090415  10.0.0.11   10.0.0.100   TCP       74 [TCP Spurious Retransmission] 48498→1 [F…
 6534 93.361564713  10.0.0.100  10.0.0.11    TCP       60 1→48498 [RST, ACK] Seq=1 Ack=2 Win=0 Len…
 6537 93.660285740  10.0.0.11   10.0.0.100   TCP       74 38930→6789 [SYN] Seq=0 Win=29200 Len=0 M…
 6538 93.664972185  10.0.0.100  10.0.0.11    TCP       74 6789→38930 [SYN, ACK] Seq=0 Ack=1 Win=28…
 6539 93.665060134  10.0.0.11   10.0.0.100   TCP       66 38930→6789 [ACK] Seq=1 Ack=1 Win=29312 L…
 6540 93.665251021  10.0.0.11   10.0.0.100   TCP       84 38930→6789 [PSH, ACK] Seq=1 Ack=1 Win=29…
 6541 93.666764677  10.0.0.100  10.0.0.11    TCP       66 6789→38930 [ACK] Seq=1 Ack=19 Win=29056 …
 6542 93.667380864  10.0.0.100  10.0.0.11    TCP      201 6789→38930 [PSH, ACK] Seq=1 Ack=19 Win=2…
 6543 93.667434167  10.0.0.11   10.0.0.100   TCP       66 38930→6789 [ACK] Seq=19 Ack=136 Win=3033…
 6544 93.667449235  10.0.0.100  10.0.0.11    TCP       66 6789→38930 [FIN, ACK] Seq=136 Ack=19 Win…
 6545 93.667712010  10.0.0.11   10.0.0.100   TCP       74 38932→6789 [SYN] Seq=0 Win=29200 Len=0 M…
 6546 93.669046227  10.0.0.100  10.0.0.11    TCP       74 6789→38932 [SYN, ACK] Seq=0 Ack=1 Win=28…
 6547 93.669132388  10.0.0.11   10.0.0.100   TCP       66 38932→6789 [ACK] Seq=1 Ack=1 Win=29312 L…
 6548 93.669308653  10.0.0.11   10.0.0.100   HTTP     102 GET / HTTP/1.1
 6549 93.671330405  10.0.0.100  10.0.0.11    TCP       66 6789→38932 [ACK] Seq=1 Ack=37 Win=29056 …
 6550 93.671373696  10.0.0.100  10.0.0.11    TCP      201 [TCP segment of a reassembled PDU]
 6551 93.671396351  10.0.0.11   10.0.0.100   TCP       66 38932→6789 [ACK] Seq=37 Ack=136 Win=3033…
 6552 93.671410563  10.0.0.100  10.0.0.11    HTTP      66 HTTP/1.0 401 Unauthorized
 6553 93.671832214  10.0.0.11   10.0.0.100   TCP       66 38930→6789 [FIN, ACK] Seq=19 Ack=137 Win…
 6554 93.671867771  10.0.0.11   10.0.0.100   TCP       66 38932→6789 [FIN, ACK] Seq=37 Ack=137 Win…
 6555 93.673083028  10.0.0.100  10.0.0.11    TCP       66 6789→38930 [ACK] Seq=137 Ack=20 Win=2905…
 6556 93.673492448  10.0.0.100  10.0.0.11    TCP       66 6789→38932 [ACK] Seq=137 Ack=38 Win=2905…

▸ Frame 1477: 58 bytes on wire (464 bits), 58 bytes captured (464 bits) on interface 0
▸ Ethernet II, Src: Dell_4f:0c:56 (5c:f9:dd:4f:0c:56), Dst: Raspberr_cf:6c:77 (b8:27:eb:cf:6c:77)
▸ Internet Protocol Version 4, Src: 10.0.0.11, Dst: 10.0.0.100
▸ Transmission Control Protocol, Src Port: 49223, Dst Port: 8080, Seq: 0, Len: 0

0000  b8 27 eb cf 6c 77 5c f9  dd 4f 0c 56 08 00 45 00   .'..lw\. .O.V..E.
0010  00 2c f7 01 00 00 31 06  7e 5c 0a 00 00 0b 0a 00   .,....1. ~\......
0020  00 64 c0 47 1f 90 7c 63  f8 b7 00 00 00 00 60 02   .d.G..|c ......`.
0030  04 00 2a c5 00 00 02 04  05 b4                     ..*..... ..
```

Figure 10
Test System Detection Stop

A honeypot is a closely monitored computing resource intended to be probed, attacked, or compromised. The value of a honeypot is determined by the information that we can obtain from it. Monitoring the data that enters and leaves a honeypot allows us to gather information that is not available to Network Intrusion Detection Systems [11].

We downloaded the honeyd version 1.6d source from its github repository [12] and compiled it in the Raspberry Pi 3 of our test system setup. We were able to successfully create virtual hosts with services on our test network as follows:

- An HP Microsoft Windows 10 with IP address 10.0.0.14 and ports 139, 445 (File sharing), 137 (netbios), and 112 open
- An Intel Apple macOS 10.12 (Sierra) (Darwin 16.0.0) with IP address 10.0.0.15 and ports 21 (FTP) and 23 (TELNET) open
- A Dell Linux Computer with IP address 10.0.0.16 and ports 22 (SSH), 23 (TELNET), 56 (DNS), and 80 (HTTP) open
- A CISCO OpenWRT Router with IP address 10.0.0.1 and ports 22 (SSH) and 23 (TELNET) open.

These hosts were created by adding the corresponding signatures from the NMAP database into the database used by HONEYD. We were able to fool NMAP into identifying these virtual hosts systems as real systems on a network, since it's using the same information that NMAP uses to identify them.

CONCLUSIONS

NMAP does its job as stated. It produces a lot of packets on some scans, so it may be detectable in a network setting if the looking for it. Some of its scan methods are unique and, if it is not being run by an authorized party, may be attributed to a hacking attempt. Its various ping methods go beyond the standard ping in an attempt to get a response from the target independent of network conditions.

Its OS detection database is extremely useful for scans and also for other applications such as honeyd, which might be used to confuse would-be infiltrators by using it to create the illusion of other computer systems in an internal LAN. However, newer systems are not identified until NMAP updates its database. You might be able to extrapolate the result from a new system if you have some knowledge it is being used and from discrepancies in the NMAP results.

Also, conditions must be ideal for OS detection to work correctly, or else the program might guess with whatever information it received from the target.

Either way, if running NMAP within an unknown environment, it is always good practice to run similar tools to corroborate the information NMAP presented.

NMAP has to send packets at the potential target to get a read on their attributes, risking detection. Whenever packets are sent to a target, your IP address is attached (unless you spoof your IP address, but then you wouldn't get a response from the target). In the interest of remaining as stealthy as possible, you want to be able to determine the operating system of a potential target without touching it. This can be accomplished with other programs like p0f, which uses attributes of the packets on the wire to determine the operating system that sent the packet. Packets are captured with a tool like Wireshark and then analyzed with a passive tool, such as P0F. P0F passively listens to the network traffic without creating any extra packets. It determines the operating system of the remote host by analyzing certain fields in the captured packets [13]. Due to this passive analysis, the remote system will not be able to detect the packet capture, such as p0f for example [14]. However, the reliability of passive recon is lower than active recon [15].

Passive fingerprinting could also be used to sniff TCP/IP ports, rather than generating network traffic by sending packets to them. Hence, it's a more effective way of avoiding detection or being stopped by a firewall [16].

References

[1] NMAP Website [Online]. Available: http://www.nmap.org. [Accessed: Feb. 1, 2017].

[2] Wireshark Website. Available: http://www.wireshark.org. [Online, accessed: Feb. 1, 2017].

[3] Raspberry Pi Website. [Online]. Available: https://www.raspberrypi.org. [Accessed: Feb. 1, 2016].

[4] Wikipedia. 2017, January 13, *User Datagram Protocol*. [Online]. Available: https://en.wikipedia.org/wiki/User_Datagram_Protocol. [Accessed: Feb. 1, 2016].

[5] Gordon "Fyodor" Lyon. (2011). "NMAP OS Fingerprinting" in *The Official Nmap Project Guide to Network Discovery and Security Scanning*, Free Edition [Online]. Available: https://nmap.org/book/osdetect-methods.html. [Accessed: Feb. 1, 2016].

[6] Gordon "Fyodor" Lyon. (2011). "NMAP OS Detect" in *The Official Nmap Project Guide to Network Discovery and Security Scanning*, Free Edition [Online]. Available: https://nmap.org/book/osdetect.html. [Accessed: Feb. 1, 2016].

[7] Erick Hjelmvik. (2011, November 5). *Passive OS Fingerprinting*. [Online]. Available: http://www.netresec.com/?page=Blog&month=2011-11&post=Passive-OS-Fingerprinting. [Accessed: Feb. 1, 2016].

[8] Philip. (2009, December 13). *The TCP Windows, Latency, and the Bandwidth Delay product*. [Online]. Available: http://www.speedguide.net/articles/the-tcp-window-latency-and-the-bandwidth-delay-2678. [Accessed: Feb. 1, 2016].

[9] Wikipedia. (2017, February 1). *Macinstosh Operating Systems*. [Online]. Available: https://en.wikipedia.org/wiki/Macintosh_operating_systems. [Accessed: Feb. 1, 2016].

[10] NMAP Github Repository. [Online]. Available: https://github.com/nmap/nmap. [Accessed: Feb. 1, 2016].

[11] Honeyd.org Webpage. [Online]. Available: http://www.honeyd.org. [Accessed: Feb. 1, 2016].

[12] Honeyd Github Page. [Online]. Available: https://github.com/DataSoft/Honeyd/. [Accessed: Feb. 1, 2016].

[13] M. Zalewski. (2012-2014). Pof Website, V3 [Online]. Available: http://lcamtuf.coredump.cx/p0f3/. [Accessed: Feb. 1, 2016].

[14] S. Pillai. (2012, December 29). *Fingerprinting-Detect Remote Operating Systems* [Online]. Available: http://www.slashroot.in/fingerprinting-detect-remote-operating-system. [Accessed: Feb. 1, 2016].

[15] OccupytheWeb. (2014, February 27). *How to conduct Passive OS Fingerprinting with Pof* [Online]. Available: http://null-byte.wonderhowto.com/how-to/hack-like-pro-conduct-passive-os-fingerprinting-with-p0f-0151191/. [Accessed: Feb. 1, 2016].

[16] The InfoSec Institute. (2014, June 19). *What You Must Know About OS Fingerptinting* [Online]. Available: http://resources.infosecinstitute.com/must-know-os-fingerprinting/. [Accessed: Feb. 1, 2016].