Lab 07 - Network Security

Objectives

- Port Scanning (nmap)
- Traffic filtering / manipulation (iptables)
- Port knocking
- Man in the Middle Attack

Port Scanning

Read here [https://nmap.org/book/man-port-scanning-techniques.html] about port scanning techniques.

For the following exercises, you will be working on a OpenStack VM, scanning for the VMs of your colleagues.

Some exercises that require two VMs (attacker and victim VMs) will be solved in pairs.

00. Preparation

- To make it easier to identify this lab's networked virtual machines, please run this command:

```
sudo su
echo "Hello, I am <insert your 1337 nickname here>" > /var/www/html/index.html
```

- To prevent SSH-trolling, you might want to change the password of your account:

```
sudo passwd student
# just put something you won't forget for this lab, like the name of your crush ;)
```

01. [5p] Network scan

- Scan the virtual network for other online virtual machines (try your /24 prefix, for added speed).
- Try to identify the one of your nearest colleague ;)
- Note: **PLEASE** don't SSH into machines you do not own! Be civil!

02. [15p] TCP port scan

- Pick the VM of your nearest colleague. Scan it for ports.
- Try Connect Scan, SYN Scan, Xmas Scan and others. What do you notice?
- Now, work in pairs: one of you open a TCP netcat server on a non-standard port (e.g. 10002) on your VM while the other scans it.
- Note: OpenStack has its own firewall; the default rules allow traffic from 10000-40000, but some other (e.g. 5001) are blocked.
- How can you get nmap to detect this port?

03. [10p] UDP port scan

- Work in pairs: one of you open a UDP netcat server on your VM while the other scans it.
- What do you notice? Why is the port detected as closed?
- Hint: use "-p <port number>" for faster scanning.

04. [10p] OS / Version scans

- Pick a VM and scan it for the Operating System and service versions.

Iptables

Iptables is an interface to the Netfilter firewall that is built into the Linux kernel. It provides an administrator with an interface to add, remove, and modify packet rules.

Here's an iptables cheeatsheet [https://gist.github.com/davydany/0ad377f6de3c70056d2bd0f1549e1017] to help you get started.

00. Best practice

- Put specific rules at the top of the policy and generic rules at the bottom.
- The more criteria you specify in the rule, the less chance you will have of locking yourself out. There are plenty of ways in which you can be more specific, such as specifying the input or output interface, the source IP address, the destination IP address, the protocol and ports.
- You can also place a rule at the very top of the chain, which includes the IP address of your workstation, so that you won't get locked out. Everyone else might still do, but if that were the case you would be able to quickly do something about it.

A rule for whitelisting your IP address at the top of the policy looks like this. Notice that Insert (-I) was used instead of Append (-A).

```
iptables -I INPUT -s <your_IP> -j ACCEPT
```

- Place loopback rules as early as possible.
- Place forwarding rules as early as possible.
- Use the state and connection-tracking modules to bypass the firewall for established connections.
- Combine rules to standard TCP client-server connections into a single rule using port lists.
- Place rules for heavy traffic services as early as possible. So, where should you place stateful packet inspection?
- Oh, and the most important one: **try to not get locked out of your machine (if SSHing)**!

01. [10p] Server Firewall

- **Note:** This task must be done on the VM.
- Block all incoming HTTP connections to your server. Ask a colleague to test it.
- Whitelist your colleague s.t. [s]he can access your HTTP server again.

02. [20p] Workstation Firewall

- Any security-conscious user should secure his/her workstation. The recommended way is to block all traffic and whitelist only the required connections.

- Starting from this:

```
# Remove any existing rules from all chains
iptables --flush

# Set the default policy to drop
iptables --policy INPUT DROP
iptables --policy OUTPUT DROP
iptables --policy FORWARD DROP

# Allow traffic on the loopback interface
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT

# Accept any related or established connections
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

- Add the rules to allow: outbound DNS lookups, SSH, NTP, PING requests, HTTP and HTTPS and inbound SSH.

So far you have a basic workstation setup. Users can access the web, they can find addresses, and the administrator can obtain remote access to make changes if need be.

This is a very good policy if you had it on a laptop that you use with public wifi hotspots (such as the ones offered by coffee shops or hotels), and you would not want anybody to compromise it. The policy would do a good job because it allows very few things in and out. If it doesn't know what to do with a packet, the packet gets dropped automatically, because you set that up at the very beginning in the policy.

03. [10p] DNS blocking

- Block access to Facebook by using iptables to block all DNS queries containing "facebook.com".
- **Hint**: The "string" module can do packet contents matching. But what does a DNS query look like?

04. [0p] Port knocking

- **NOTE**: This task gets solved on the VM!
- **NOTE**: Make sure that the ports used are allowed by the Openstack firewall

- Start a netcat TCP server on 31337 and implement a port knocking scheme to open it!
- Ask a colleague to scan your port. It should be seen as closed / filtered.
- Then ask that colleague to knock the correct port sequence and connect to your netcat server.

- **Hint:** We're still on the iptables section ;) Check out the "recent" module!
- **Hint2:** https://wiki.archlinux.org/index.php/Port_knocking [https://wiki.archlinux.org/index.php/Port_knocking]

Man in the Middle

The ARP protocol is widely used for translating L3 (IP) addresses (IP) into L2 (link layer) addresses.

Unfortunately, it suffers from a critical security vulnerability: due to its unauthenticated nature, a destination machine has no way of determining whether a ARP reply is valid, so an attacker can forge ARP packets and tell a victim PC to associate the router's IP address to the attacker's MAC address, such that it will send all traffic through the victim's machine.

Note: Since OpenStack as built-in ARP Poisoning protection, you must do these tasks on your physical machines.

01. [20p] ARP Cache Poisoning

- Pair yourself with another colleague. Exchange the IP addresses of your PCs.
- One of you is the attacker, the other is the victim.

- **Attacker:** Establish yourself as the Man in the Middle using ARP Spoofing:

```
sudo apt-get install dsniff
# Turn on IP Forwarding
sysctl -w net.ipv4.ip_forward=1
# Start poisoning the ARP cache!
sudo arpspoof -i <interface> -t <victim_ip> <gateway_ip> -r
```

- **Victim:** check your ARP cache for the gateway's MAC address:

```
watch ip neigh show
```

- **Attacker**: in a separate terminal, start a tcpdump or Wireshark session and start spying the victim's network traffic :)

11. [10p] Feedback

Please take a minute to fill in the feedback form [https://forms.gle/5Lu1mFa63zptk2ox9] for this lab.

isc/labs/07.txt · Last modified: 2020/04/06 08:50 by mihai.chiroiu