Lab 08 - Network Security

Objectives

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

Preparation

Please spawn a virtual machine on openstack.

You need to prepare the playground:

- To make it easier for others to identify your VM, please run this command:

```
echo "<h1>Hello, I am "<insert your 31337 name here>" </h1>" | sudo tee /var/www/html/index.html
# you may also append a slogan (choose one or make your own):
echo '<h3>All your base are belong to us!</h3>' | sudo tee -a /var/www/html/index.html
echo '<h3>We will rock you!</h3>' | sudo tee -a /var/www/html/index.html
echo '<h3>Am talent și nu mă las!</h3>' | sudo tee -a /var/www/html/index.html
```

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

```
sudo passwd student
# set an unbreakable password, but make sure you won't forget it until the end of the lab!
```

- After that, enable the hacker guest account for your ~~enemies~~ colleagues to use (*and don't cheat!*):

```
# let the h4x0rs in (enables the account):
sudo usermod -e -1 -U hacker
```

Leave hacker's password unchanged!

No trolling / spamming yet, please!

Tasks

[30p] 1. Port Scanning

Read here about port scanning techniques.

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

[10p] Task A: Network scan

- Scan the network for other hosts (you can use CIDR notation! Remember OpenStack's network prefix?).
- Either a ping scan (**nmap**) or an ARP scan (**arp-scan**) should be fine. Is there a difference?
- If you are doing this task with colleagues, try to find each other's hosts (note: since you all have the same VM images, try to find one with the same ports!).

[10p] Task B: TCP / UDP port scans

- Pick the VM of your nearest colleague. Scan it for ports.
- 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.
- If you don't have any willing colleague, just pick a host at random.

- Try TCP Connect Scan, SYN Scan, Xmas Scan and others. What do you notice?
- Hint: use -p <port range> for faster, directed scanning.
- Do the same for UDP: open a netcat -u server on your VM while the other scans for it.
- Note: OpenStack has its own firewall behind each VM; ~~the default rules allow traffic from 10000-40000, but others (e.g. 5001) may be blocked.~~ everything should be allowed now ;)

[10p] Task C: OS / Version scans

- Try running OS detection with **nmap** on some of the detected hosts. Can it correctly identify any of them? What about services running on open ports and their versions?

[30p] 2. 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 a iptables cheeatsheet to help you get started.

[5p] Intro: Best practices

Here are some best practices when writing firewall rules (read them):

- 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 (-i | -o), the source IP address (-s), the destination IP address (-d), the protocol and ports (-p <tcp|udp> --dport|sport <number>).
- First, place a rule at the very top of the INPUT chain, which includes the IP address of your workstation (in our case, the fep will originate your ssh packets to OpenStack), so that you won't get locked out. 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), because we want this rule to be the **first**.

```
iptables -I INPUT -s "<FEP IP address>" -j ACCEPT
```

- Add whitelist or blacklist rules for the types of traffic you use (or don't want to receive); you will do that for the next subtask! You should use the INPUT chain for that.
- You should use the connection tracking module(s) to bypass the firewall for already established (usually, outgoing) connections, otherwise replies cannot be received (if you drop them in INPUT).

Try to not get locked out of ssh-ing your virtual machine! Double-check the rule before you add it to a iptables chain (always ask yourself: does it match a broad range of packets / will it filter my SSH traffic from fep)!

[10p] Task B: Server Firewall

- For this task: work in pairs!
- Ask for your colleague's VM IP address. Using curl, read the message from his http server (check man / some examples if you never used it).
- See if you can ssh into it using the hacker:student credentials! When successful, send a broadcast message

```
hacker@<colleague-VM>$ wall "Wazzaap?"
```

- Run tcpdump to see where ssh packets come from (make sure to ignore the ones coming from fep):

```
tcpdump -i eth0 'host not <FEP's IP address here>'
# note the source IP of the other OpenStack VMs connecting to you!
```

- We want to stop the spam, so blacklist the ssh (TCP port 22) and http (TCP port 80) from any other OpenStack VMs addresses (recall the CIDR range?). **Be extra careful not to ban yourself!**
- You may still wish for your more civilised friends to be able to connect back to you. Whitelist them again by adding a ALLOW rule before the ones dropping the packets!

If you want to turn off wall messages being displayed on your tty, check out man mesg command!

But beware when nesting shells! (e.g., from student to root using su). In this case, you must exit them all and issue mesg on the first one (the tty spawned by ssh!).

[5p] Task C: 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.
- **Note:** this task is for reference only! But check it out: it 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 (iptables … -P DROP):

Workstation Firewall Script

[10p] Task D: DNS blocking

- Block access to Facebook by using iptables to block all DNS queries containing "facebook.com".
- **Hint**: The string iptables module can do packet contents matching (sudo iptables -m string -help). But what does a DNS query look like?
- **Note**: you can supply a hex string parameter with the syntax |HH HH HH …|<plaintext> (e.g., hello|20 57 6F|rld) to match binary contents of a packet! Also choose a string matching algorithm!

[0p] Task E: Port knocking (bonus)

- 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.

- Check out the recent iptables module!
- **Reference:** https://wiki.archlinux.org/index.php/Port_knocking

[30p] 3. Man in the Middle

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

[20p] Task A: ARP Cache Poisoning

- We will use two Docker containers to simulate a vulnerable local network (OpenStack filters ARP packets, so you won't be able to do this there).

```
# turn on IP Forwarding -- for the attacker (inherited from host)
# containers don't have permission for this and we don't want to bother with capabilities
sudo sysctl -w net.ipv4.ip_forward=1

# Open a "Victim" terminal (on your VM):
docker run --rm -ti --entrypoint /bin/bash --name victim ubuntu:22.04
# Open an "Attacker" terminal (also on the same VM):
docker run --rm -ti --entrypoint /bin/bash --name attacker --sysctl net.ipv4.ip_forward=1 ubuntu:22.04

# we need two terminals for the attacker (for the tcpdump later)
# so... in a third terminal, spawn a Docker exec shell:
docker exec -ti attacker /bin/bash
```

- **Victim terminal** (note: we're inside a container): install prerequisites & find out the IP address given by Docker:

```
apt update && apt install -y iproute2 iputils-ping netcat-openbsd
ip a sh
```

- **Attacker terminal:** install prerequisites & establish yourself as the Man in the Middle using ARP Spoofing:

```
# install prerequisites for this task
apt update && apt install -y dsniff tcpdump iproute2 iputils-ping

# start poisoning the host's ARP cache
arpspoof -i <INTERFACE> -t <VICTIM_IP> <GATEWAY_IP> -r
```

[10p] Task B: Test Implementation

- **Attacker terminal:** Listen for DNS traffic from the victim:

```
tcpdump udp port 53 -nvvX
```

- **Victim terminal:** Open a netcat server and listen for content

```
# check ARP table (your gateway's MAC should be the attacker's)
ip nei sh
# ping your favorite website
ping my.secretwebsite.com
# Unfortunately, IP forwarding inside container doesn't work :(
```

The tcpdump capture on the Attacker terminal should show the intercepted DNS requests ;)

[10p] 4. Feedback

Please take a minute to fill in the feedback form for this lab.

isc/labs/08.txt · Last modified: 2023/12/04 12:20 by florin.stancu

Lectures

- Lecture 01 - Introduction
- Lecture 02 - Cryptography
- Lecture 03 - Hardware Security
- Lecture 04 - Access Control
- Lecture 05 - Authentication and Key Establishment
- Lecture 06 - Application Security
- Lecture 07 - Operating System Security
- Lecture 08 - Network Security
- Lecture 09 - Web Security
- Lecture 10 - Privacy Preserving Technologies
- Lecture 11 - Forensics

Labs

- kernel
- Lab 01 - Introduction
- Lab 02 - Cryptography
- Lab 03 - Authentication and Key Establishment Lab
- Lab 03 - Hardware Security
- Lab 04 - Access control
- Lab 05 - Authentication in Linux
- Lab 06 - Application Security
- Lab 07 - Operating System Security
- Lab 08 - Network Security
- Lab 09 - Web Security
- Lab 10 - Forensics
- Lab 11 - Privacy Technologies
- Lab 12 - Security and Machine Learning

Support

- Useful resources
- Virtual Machine

Table of Contents