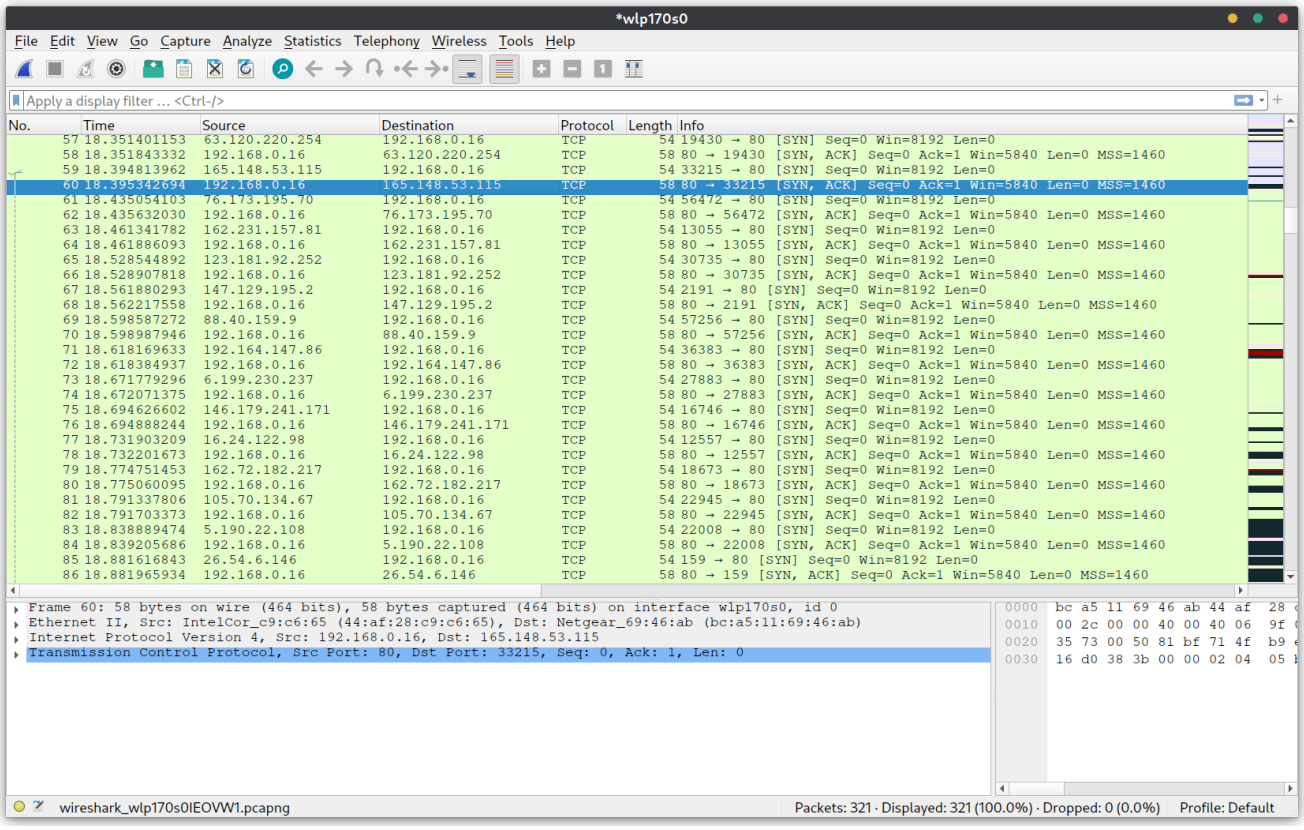
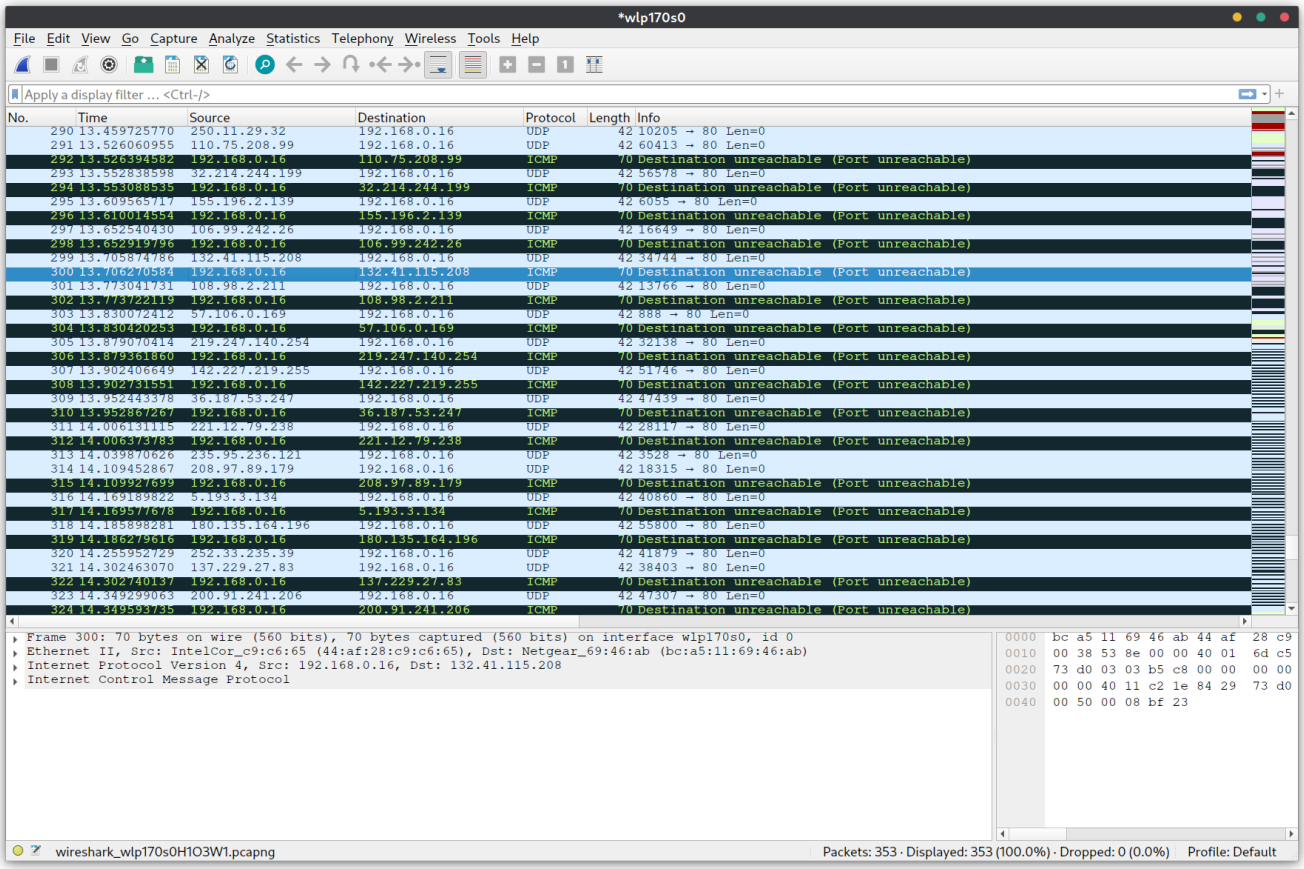
README.md

TCP SYN Flooding Attacks

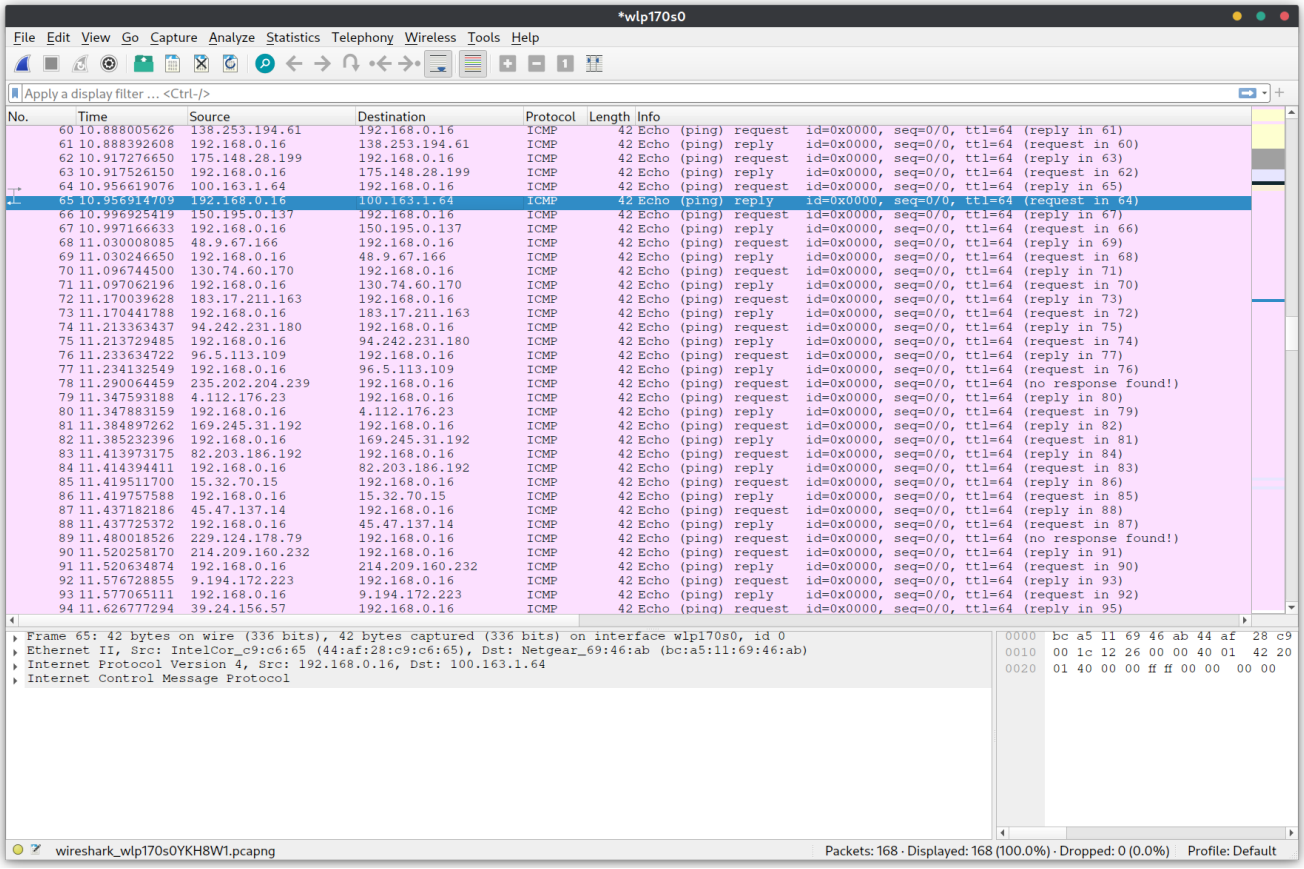
1. Explanation how the TCP SYN flood attack works.
    - TCP SYN flooding attacks take advantage of the TCP design that requires a 3 way handshake. By continually sending SYN packets, the server will keep sending back SYN ACK packets and waiting around to recieve an ACK. But the ACK will never come because either the sender isnt responding to them, or more likely, the sender has spoofed their IP anyways. This allows a single sender to spoof many different IP addresses, and make it look to the server as if there are many valid requests coming in. The server will use up all channels waiting for the ACKs and if it times out, another one will take its place. Thus the term *"flooding"* because all inputs are flooded with SYN packets.
2. Explanation how SYN cookies work to prevent denial-of-service effect from SYN flood attack.
    - SYN Cookies allow a TCP server not be flooded when TCP SYN flood attacks happen. Normally connections will be dropped when all available inputs are in use during a flood. SYN Cookies allow the server to essentially create the original SYN queue entry when an ACK is recieved because the SYN Cookie has helped encode the necessary information to do so.
3. Follow the lab document to launch TCP SYN flooding attacks by using Scapy.
    - See attached python code `syn_flood.py` or on github.
    - 

UPD Flooding Attacks

1. Explanation how the UPD attack works.
    - UDP flooding is much more rudimentary and relies on the complete overwhelming of a networks resources similar to a simple PING DDoS attack. It floods the network with UDP datagram packets, and as seen below in the wireshark screenshot, the server sends back a "Destination Unreachable" packet. The attacker needs to have enough resources to completely overwhelm the other network.
2. Follow the lab document to launch UDP flooding attacks by using Scapy.
    - See attached python code `udp_flood.py` or on github.
    - 

ICMP Flooding Attacks

1. Explanation how the ICMP attack works.
    - ICMP flood attacks are some of the first DDoS attacks that someone may learn. It relied on the *"ping"* protocol, and similar to the UDP flood attack, the attacker must have the network resources to completely overwhelm the other network. This is usually accomplished with botnets so that there can be a large expanse of computers all performing the attack at the same time. The attacker will repeatedly send an *"Echo (ping) request"* and the server will reply with an *"Echo (ping) response"*. This has to be done enough times at once to overload the server with too many messages, thus flooding its resources.
2. Follow the lab document to launch ICMP flooding attacks by using Scapy.
    - See attached python code `icmp_flood.py` or on github.


*wlp170s0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
No. Time Source Destination Protocol Length Info
60 10.888005626 138.253.194.61 192.168.0.16 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 61)
61 10.888392608 192.168.0.16 138.253.194.61 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 60)
62 10.917276650 175.148.28.199 192.168.0.16 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 63)
63 10.917526150 192.168.0.16 175.148.28.199 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 62)
64 10.956619076 100.163.1.64 192.168.0.16 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 65)
65 10.956914709 192.168.0.16 100.163.1.64 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 64)
66 10.996925419 150.195.0.137 192.168.0.16 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 67)
67 10.997166633 192.168.0.16 150.195.0.137 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 66)
68 11.030008085 48.9.67.166 192.168.0.16 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 69)
69 11.030246650 192.168.0.16 48.9.67.166 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 68)
70 11.096744500 130.74.60.170 192.168.0.16 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 71)
71 11.097062196 192.168.0.16 130.74.60.170 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 70)
72 11.170039628 183.17.211.163 192.168.0.16 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 73)
73 11.170441788 192.168.0.16 183.17.211.163 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 72)
74 11.213363437 94.242.231.180 192.168.0.16 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 75)
75 11.213729485 192.168.0.16 94.242.231.180 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 74)
76 11.233634722 96.5.113.109 192.168.0.16 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 77)
77 11.234132549 192.168.0.16 96.5.113.109 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 76)
78 11.290064459 235.202.204.239 192.168.0.16 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
79 11.347593188 4.112.176.23 192.168.0.16 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 80)
80 11.347883159 192.168.0.16 4.112.176.23 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 79)
81 11.384897262 169.245.31.192 192.168.0.16 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 82)
82 11.385232396 192.168.0.16 169.245.31.192 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 81)
83 11.413973175 82.203.186.192 192.168.0.16 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 84)
84 11.414394411 192.168.0.16 82.203.186.192 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 83)
85 11.419511700 15.32.70.15 192.168.0.16 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 86)
86 11.419757588 192.168.0.16 15.32.70.15 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 85)
87 11.437182186 45.47.137.14 192.168.0.16 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 88)
88 11.437725372 192.168.0.16 45.47.137.14 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 87)
89 11.480018526 229.124.178.79 192.168.0.16 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
90 11.520258170 214.209.160.232 192.168.0.16 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 91)
91 11.520634874 192.168.0.16 214.209.160.232 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 90)
92 11.576728855 9.194.172.223 192.168.0.16 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 93)
93 11.577065111 192.168.0.16 9.194.172.223 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 92)
94 11.626777294 39.24.156.57 192.168.0.16 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 95)
Frame 65: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface wlp170s0, id 0
Ethernet II, Src: IntelCor_c9:c6:65 (44:af:28:c9:c6:65), Dst: Netgear_69:46:ab (bc:a5:11:69:46:ab)
Internet Protocol Version 4, Src: 192.168.0.16, Dst: 100.163.1.64
Internet Control Message Protocol
0000 bc a5 11 69 46 ab 44 af 28 c9
0010 00 1c 12 26 00 00 40 01 42 20
0020 01 40 00 00 ff ff 00 00 00 00
wireshark_wlp170s0YKH8W1.pcapng
Packets: 168 · Displayed: 168 (100.0%) · Dropped: 0 (0.0%)
Profile: Default