Transport Layer (UDP, TCP) – Attacks & Countermeasures

CYBR371: System and Network Security, (2024/T1)

Arman Khouzani, Mohammad Nekooei
Slides modified from "Masood Mansoori"

25 March, 2024

Victoria University of Wellington – School of Engineering and Computer Science

Transport layer

Delivers data from application to application.

To address the application on a machine, we use port numbers (just like we use IP addresses and MAC addresses for machines).

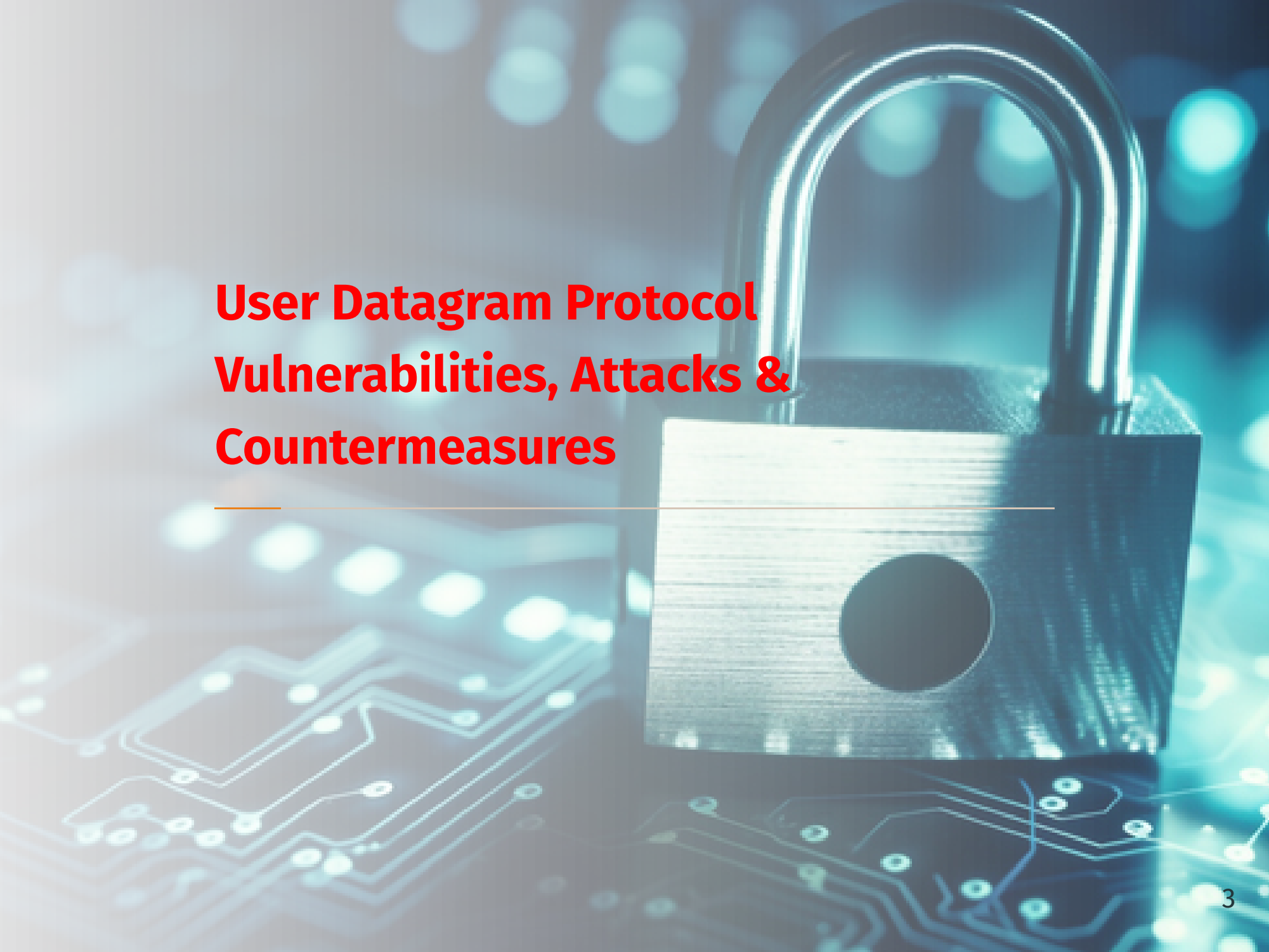
Each host (each IP!) has 65,536 ports

- Use of ports 1-1023 requires privileges
- Some ports are reserved for specific apps (20, 21: FTP, 23: Telnet, 80: HTTP)

Transport Layer

Transport layer has two main protocols:

- UDP (User Datagram protocol): Best effort protocol
- TCP (Transmission control protocol): reliable, byte-stream oriented, with capacity control, transmits segments.

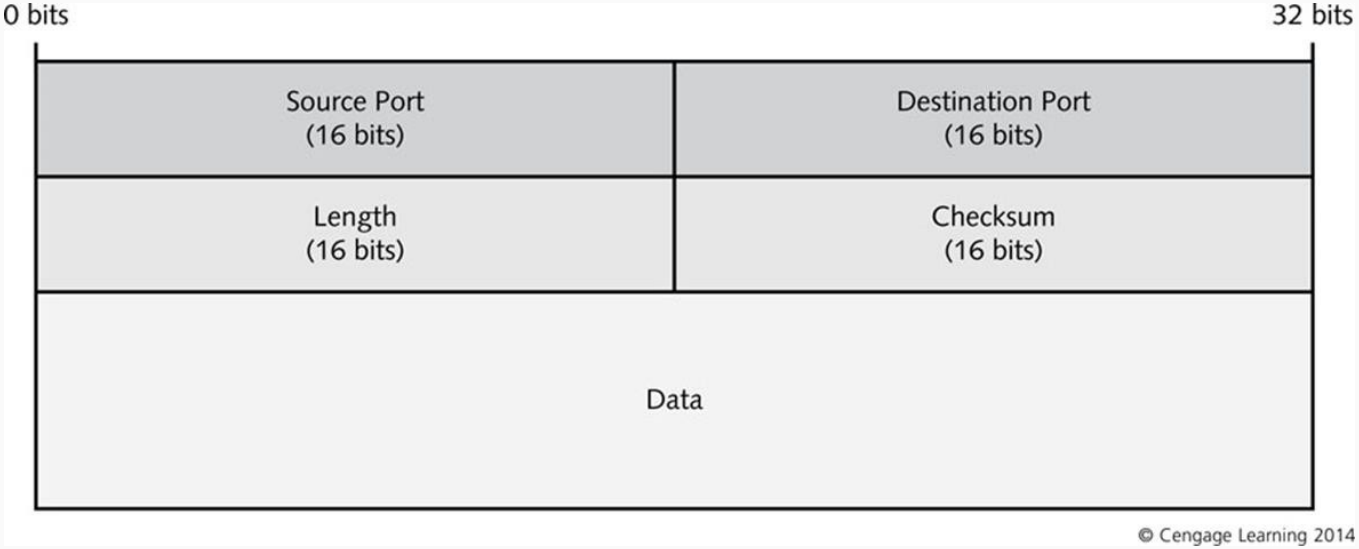


User Datagram Protocol Vulnerabilities, Attacks & Countermeasures

User Datagram Protocol (UDP)

User Datagram Protocol (UDP) provides a transport service for IP

- Connectionless, unreliable
- Much faster than TCP
- Used for broadcasting messages or for protocols that do not require the same level of service as TCP



UDP Client and Server Programs

```
import socket
IP = "127.0.0.1"
PORT = 9090
data = b'Hello World '!
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.sendto(data, (IP, PORT))
```

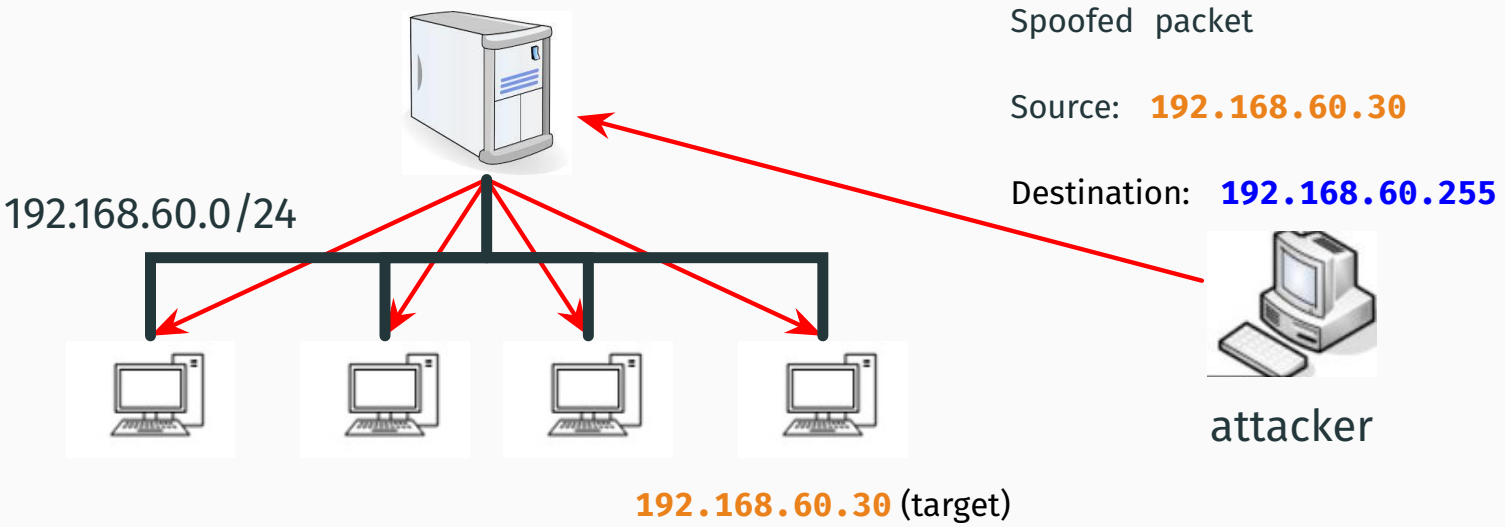
```
import socket IP = "0.0.0.0" PORT = 9090
sock = socket.socket(socket.AF_INET, socket_SOCK_DGRAM)
sock.bind(IP, PORT)
while True:
    data, (ip, port) = sock.recvfrom(1024)
    print("Sender: {} and Port: {}".format(ip, port))
    print("Received Message: {}".format(data))
```

Attacks - UDP Fraggle Attack

Fraggle Attack uses UDP echo packet instead of ICMP echo packets: Port 7 echoes whatever is sent to it.

1. Attacker sends a large UDP echo traffic (UDP echo request packet) to IP broadcast address with a spoofed source address.
2. All computers reply with UDP Echo reply packets.
3. Source IP was spoofed, victim is overwhelmed creating a DoS attack.

Attacks - UDP Fraggle Attack



Fraggle attacks, like Smurf attacks, are starting to become outdated and are commonly stopped by most firewalls or routers.

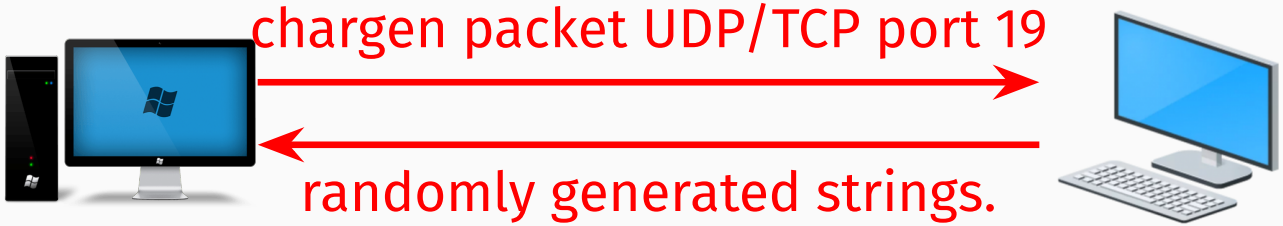
Attacks - UDP Fraggle Attack

Variations: Using the ports that generate some character string: echo (7), chargen (19), time (37), datetime (13)

Countermeasures:

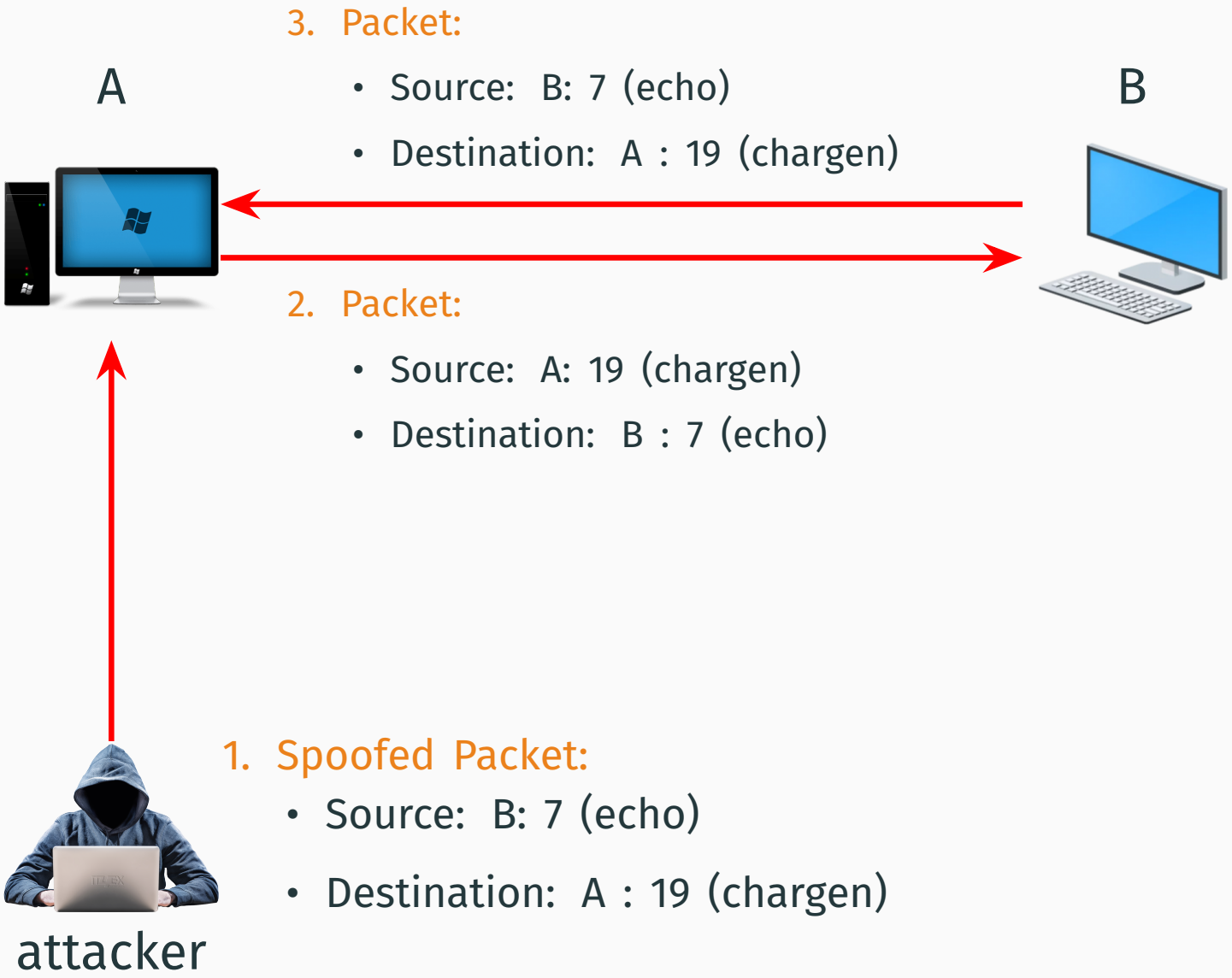
- Block packets with source address as broadcast address
- Block UDP service port if you're not using them

Attacks – UDP chargen




```
$ netstat xxx.xxx.xxx.xxx 19
0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvw
123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwx
23456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxy
3456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz
456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{
56789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|
6789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}
789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}
89:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|} !
9:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"
:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"#
;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"#$
<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"#$%
=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"#$%&'
>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"#$%&'
?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"#$%&'(
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"#$%&'()
ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"#$%&'()*
BCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"#$%&'()*+
CDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"#$%&'()*+,
DEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"#$%&'()*+,-
EFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"#$%&'()*+,-.
FGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"#$%&'()*+,-./
GHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"#$%&'()*+,-./0
HIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"#$%&'()*+,-./01
IJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"#$%&'()*+,-./012
JKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"#$%&'()*+,-./0123
KLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"#$%&'()*+,-./01234
LMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"#$%&'()*+,-./012345
MNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"#$%&'()*+,-./0123456
NOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"#$%&'()*+,-./01234567
OPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"#$%&'()*+,-./012345678
PQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"#$%&'()*+,-./0123456789
QRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"#$%&'()*+,-./0123456789:
RSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"#$%&'()*+,-./0123456789:;
STUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"#$%&'()*+,-./0123456789:;<
TUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"#$%&'()*+,-./0123456789:;<=
UVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"#$%&'()*+,-./0123456789:;<=>
VWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"#$%&'()*+,-./0123456789:;<=>?
WXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"#$%&'()*+,-./0123456789:;<=>?@
XYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"#$%&'()*+,-./0123456789:;<=>?@A
YZ[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"#$%&'()*+,-./0123456789:;<=>?@AB
Z[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"#$%&'()*+,-./0123456789:;<=>?@ABC
[\]^_`abcdefghijklmnopqrstuvwxyz{|} !"#$%&'()*+,-./0123456789:;<=>?@ABCD
```

Attacks - UDP Ping Pong Attack



Attacks - UDP Ping Pong Attack

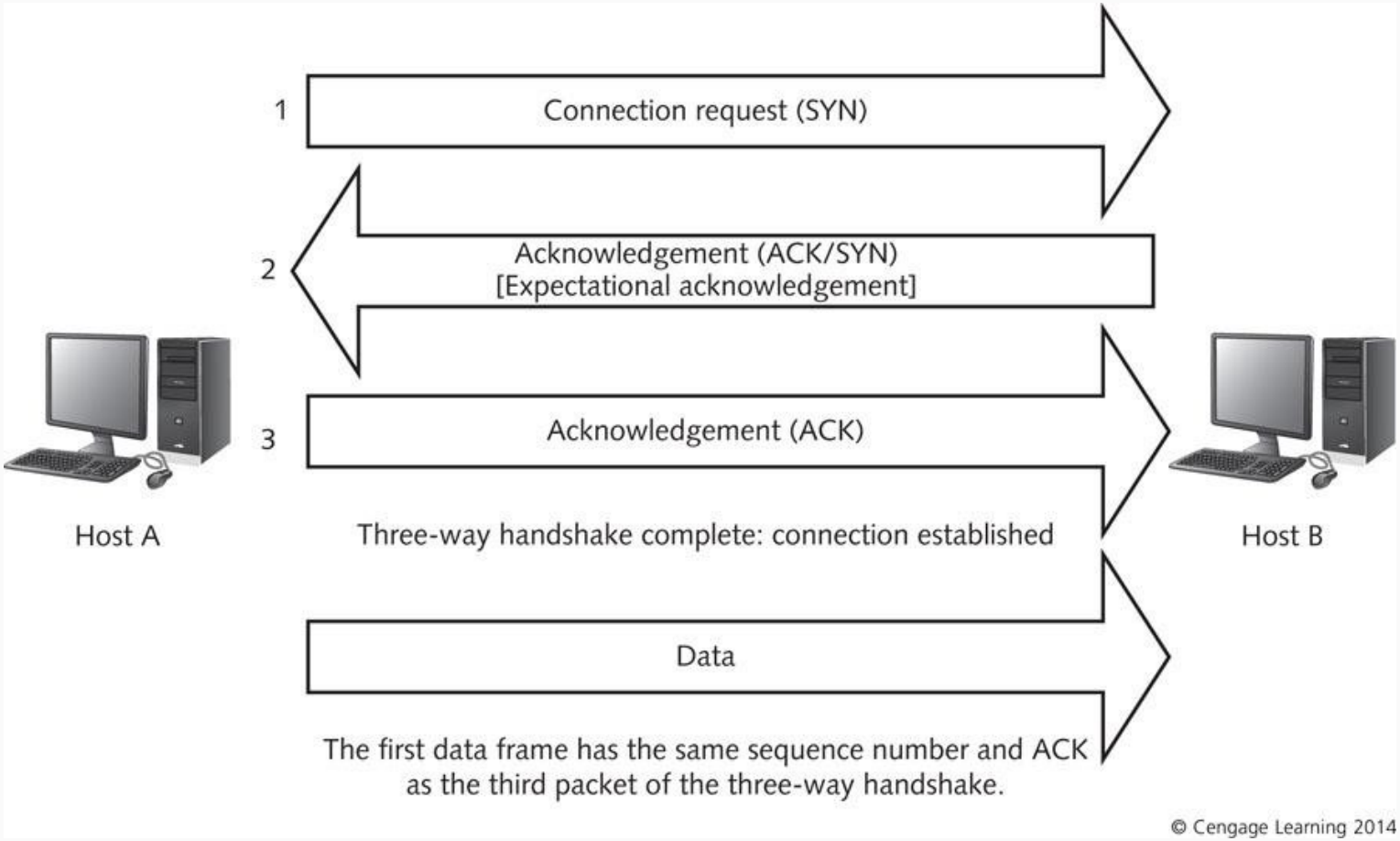
UDP ping pong attack takes advantage of the **chargen** and **echo** ports (used legitimately to test hosts and networks).

1. Attacker sends a malformed UDP packet to chargen port (19) of host A, with source address of host B (target) and source port as echo port (7).
2. Host A sends random characters to echo port of host B
3. Host B replies it back to chargen port of host A.
4. This sequence run infinitely between A and B.

Transmission Control Protocol (TCP) Attacks and Countermeasures

Transmission Control Protocol

Provides a reliable and ordered communication channel between networked applications. TCP maintains a logical connection (virtual) using a three-way handshake.



The TCP Three-Way Handshake

Establishing connection-oriented communication using a three-way handshake:

1. Host A sends an initial sequence number in its first packet to Host B.
    - The packet is called a **SYN** packet.
2. Host B receives **SYN** packet - responds with **SYN ACK** with an initial sequence number for Host B.
    - Includes an acknowledgement number that is one more than the initial sequence number.
3. Host A sends an **ACK** packet to Host B.
    - Increases Host B's sequence number by one.

TCP FIN

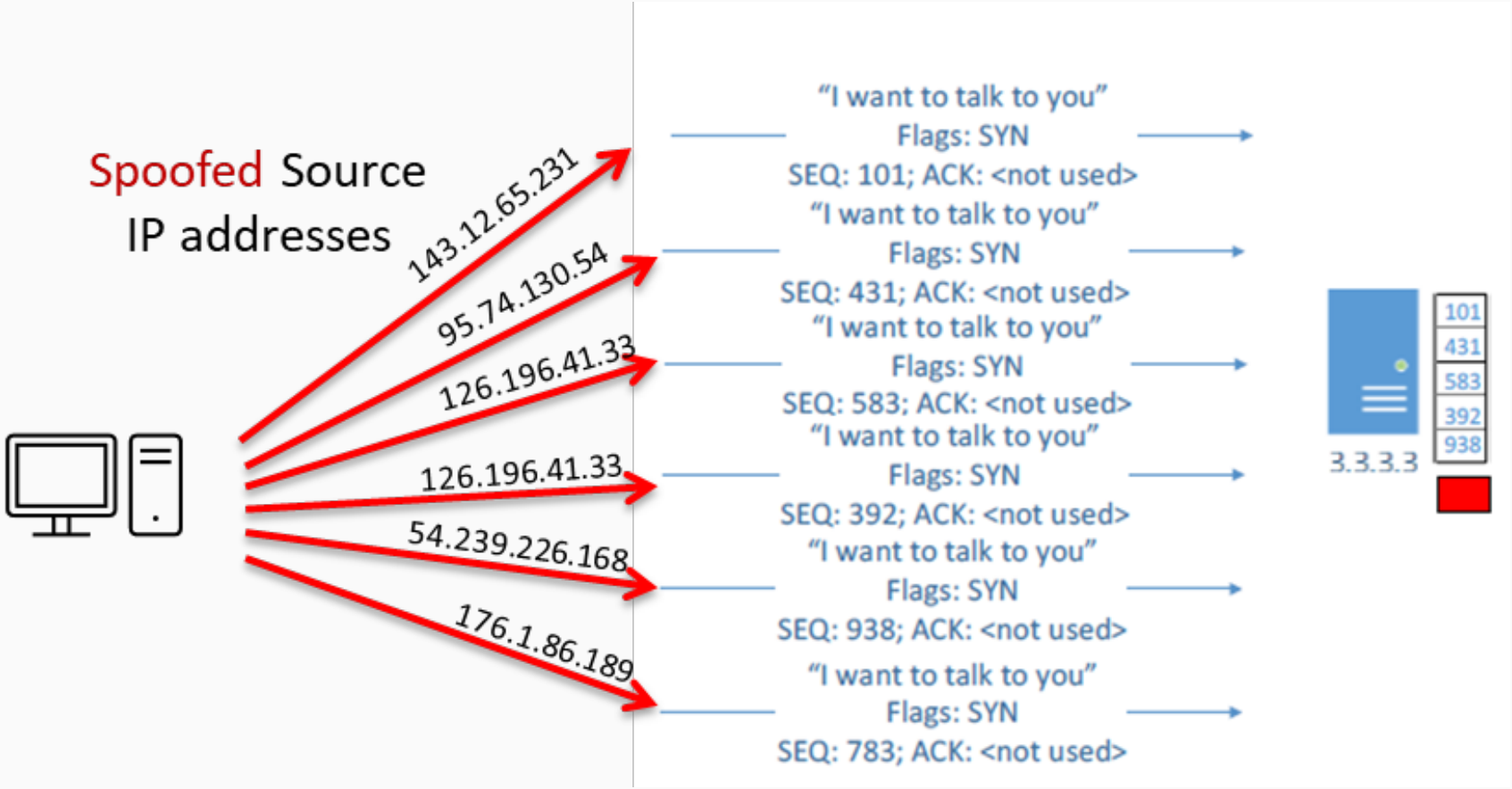
FIN flag is set when either side is ready to end the session.

Station that receives the initial flag sends a response packet with the ACK flag and its own FIN flag set to acknowledge receipt and to show it is ready to end the session.

SYN Flood Attack

Think of TCP 3-way handshake:

- Server has a number of slots for incoming connections.
- When slots are full no more connections are accepted.



SYN Flood Attack

`netstat -anp`

- Active Internet connections (servers and established)

```
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN      1339/rpcbind
tcp        0      0 0.0.0.0:33586           0.0.0.0:*               LISTEN      1395/rpc.statd
tcp        0      0 192.168.122.1:53        0.0.0.0:*               LISTEN      1962/dnsmasq
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN      1586/cupsd
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN      2703/sendmail: acce
tcp        0      0 0.0.0.0:1241            0.0.0.0:*               LISTEN      1851/nessusd: waiti
tcp        0      0 127.0.0.1:25            127.0.0.1:60365         TIME_WAIT   -
tcp        0      0 127.0.0.1:25            127.0.0.1:60240         TIME_WAIT   -
tcp        0      0 127.0.0.1:25            127.0.0.1:60861         TIME_WAIT   -
tcp        0      0 127.0.0.1:25            127.0.0.1:60483         TIME_WAIT   -
tcp        0      0 127.0.0.1:25            127.0.0.1:60265         TIME_WAIT   -
tcp        0      0 127.0.0.1:25            127.0.0.1:60618         TIME_WAIT   -
tcp        0      0 127.0.0.1:25            127.0.0.1:60407         TIME_WAIT   -
tcp        0      0 127.0.0.1:25            127.0.0.1:60423         TIME_WAIT   -
tcp        0      0 127.0.0.1:25            127.0.0.1:60211         TIME_WAIT   -
tcp        0      0 127.0.0.1:25            127.0.0.1:60467         TIME_WAIT   -
tcp        0      0 127.0.0.1:25            127.0.0.1:60213         TIME_WAIT   -
```

SYN/ACK Flood

The aim is to tie up resources attempting to match the responses (SYN-ACK) to non-existent SYN requests.

Is usually used in conjunction with IP spoofing

SYN Flood Mitigation

- SYN Cookies
  - Hiding SYN information in ISN (initial seq no) of the ACK packet
- Whitelists

What is a SYN cookie?

Rather than store connection data, send it to the host who is initiating the connection and have him return it to you

Check that f(data) is valid for this connection. Only at that point you allocate the state.

What is a SYN cookie?

The secure hash makes it difficult for the attacker to guess what f() will be, and therefore the attacker cannot guess a correct ACK if he spoofs

What is a SYN cookie?

SYN Cookie:

- Timestamp % 32 + MSS? + 24-bit hash
- Components of 24-bit hash:
  - Server IP address
  - Server port number
  - client IP address
  - client port
  - timestamp

What is a SYN cookie?

To enable SYN cookies (1):

- **`echo 1 > /proc/sys/net/ipv4/tcp_syncookies`**

To enable SYN cookies (2):

- Edit **`/etc/sysctl.conf`**
- Set **`sys.net.ipv4.tcp_syncooikes=0`**

Note: All TCP related settings can be seen from **`/proc/sys/net/ipv4/`**:

- **`tcp_max_syn_backlog`**
- **`tcp_synack_retries`**
- **`tcp_syn_retries`**

The TCP Three-Way Handshake – header

TCP three-way handshake: SYN

TCP three-way handshake: SYN ACK

TCP three-way handshake: ACK

TCP Reset Attack

Every packet sent over TCP has a sequence number, put by the sender (each).

The receiver uses it to (re-)order the packets it receives

TCP Reset Attack

Suppose attacker can guess the seq number for an existing connection

- Attacker can send RST packet to close the connection
- Results in DoS

TCP Reset Attack

Most TCP stacks now generate random SNs. Random generator should be unpredictable.

- but attacker can inject packets after eavesdropping to obtain current SN
- Predict next SN
- Attacker can now do TCP spoofing (create TCP session with forged source IP)

TCP Reset Attack

```
from scapy.all import *

def Spoof_RESET(pkt):
    tcp_r = pkt[TCP]
    a = IP(src="10.0.2.31", dst="10.0.2.32")
    b = TCP(sport=23, dport=tcp_r.sport, flags='R', seq=
        tcp_r.ack)
    pkt = a/b
    send(pkt, verbose=0)

f = 'tcp and src host 10.0.2.32 and dst host 10.0.2.31
    and dst port 23'
sniff(filter=f, prn=Spoof_RESET)
```

RST/FIN Flood

Uses spoofed RST or FIN packets to flood the target servers and consume their resources which attempt to match the packets to non-existent open TCP sessions.

Multiple ACK SYN-ACK Spoofed Session Flood

A variation of ACK flood and RST/FIN flood:

- Sends a large number of related SYN and ACK packets followed by RST or FIN packets.
- The goal is to mimic legitimate TCP traffic and consume resources on the target server which attempts to match the spoofed packets to legitimate traffic.

TCP Session Hijacking

Involves injecting packets into existing TCP connections.

- Predict the sequence number used to identify the packets in a TCP connection.
- Not necessarily the next sequence number (out of order packets are kept in buffer).

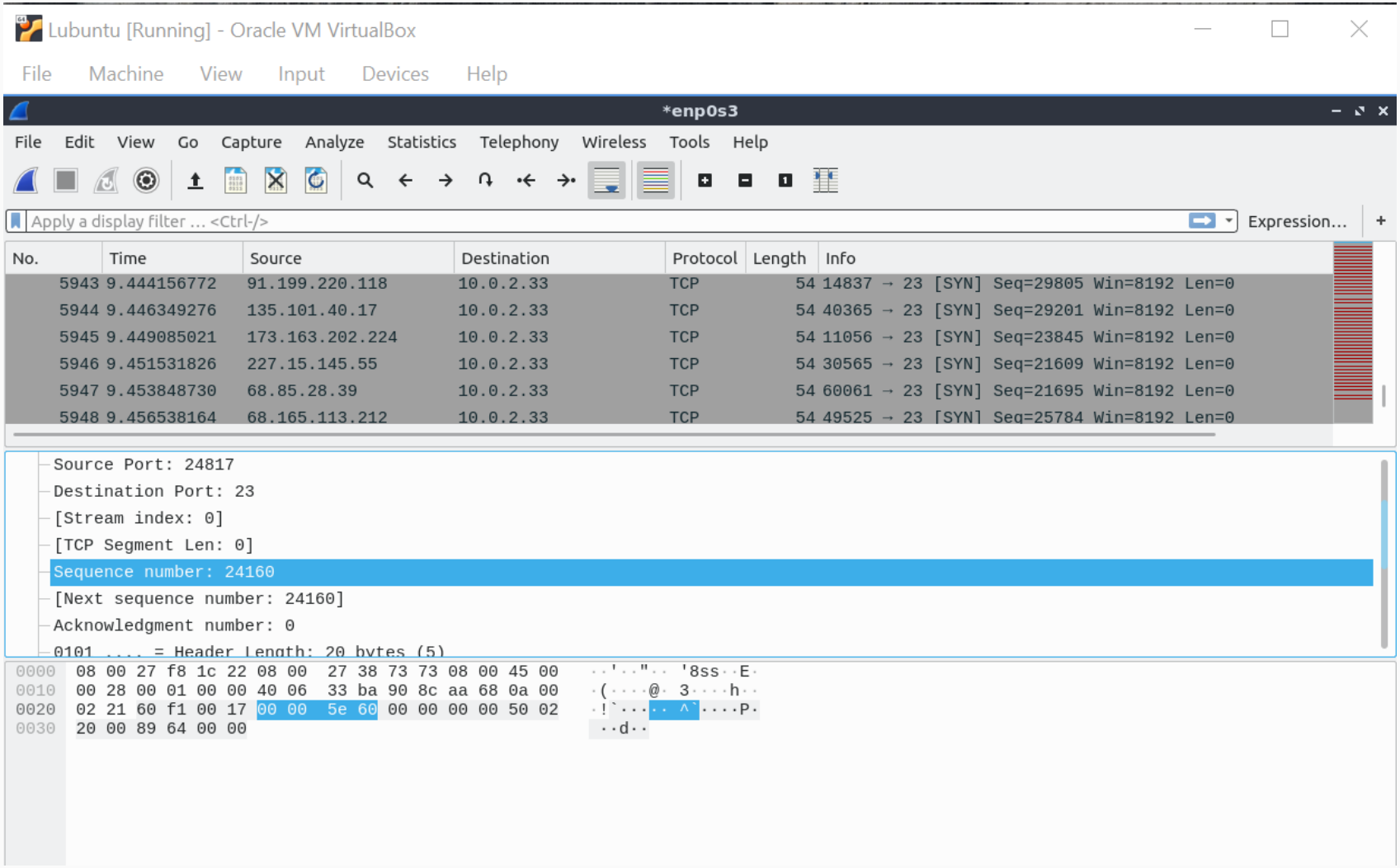
What does an attacker need?

- Source IP, Source port.
- Destination IP, Destination port.
- Sequence number (used for authenticating packets).

Initial seq number needs high degree of unpredictability.

- If the attacker knows the initial seq number and amount of traffic sent, they can estimate likely current values.

TCP Session Hijacking



Socket Exhaustion Attack: Socket

Socket is an abstraction allowing an application to bind to a transport layer address.

A socket is uniquely defined by:

- Internet address (e.g. **`135.211.34.8`**)
- Communication protocol (e.g. **TCP**, **UDP**)
- Port number (**22**, **23**, **80**, **37560**, **43484**)

Socket Example - Python

```python
import socket
HOST = '127.0.0.1'
PORT = 65432
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()
    conn, addr = s.accept()
    with conn:
        print('Connected by', addr)
        while True:
            data = conn.recv(1024)
            if not data:
                break
            conn.sendall(data)
```

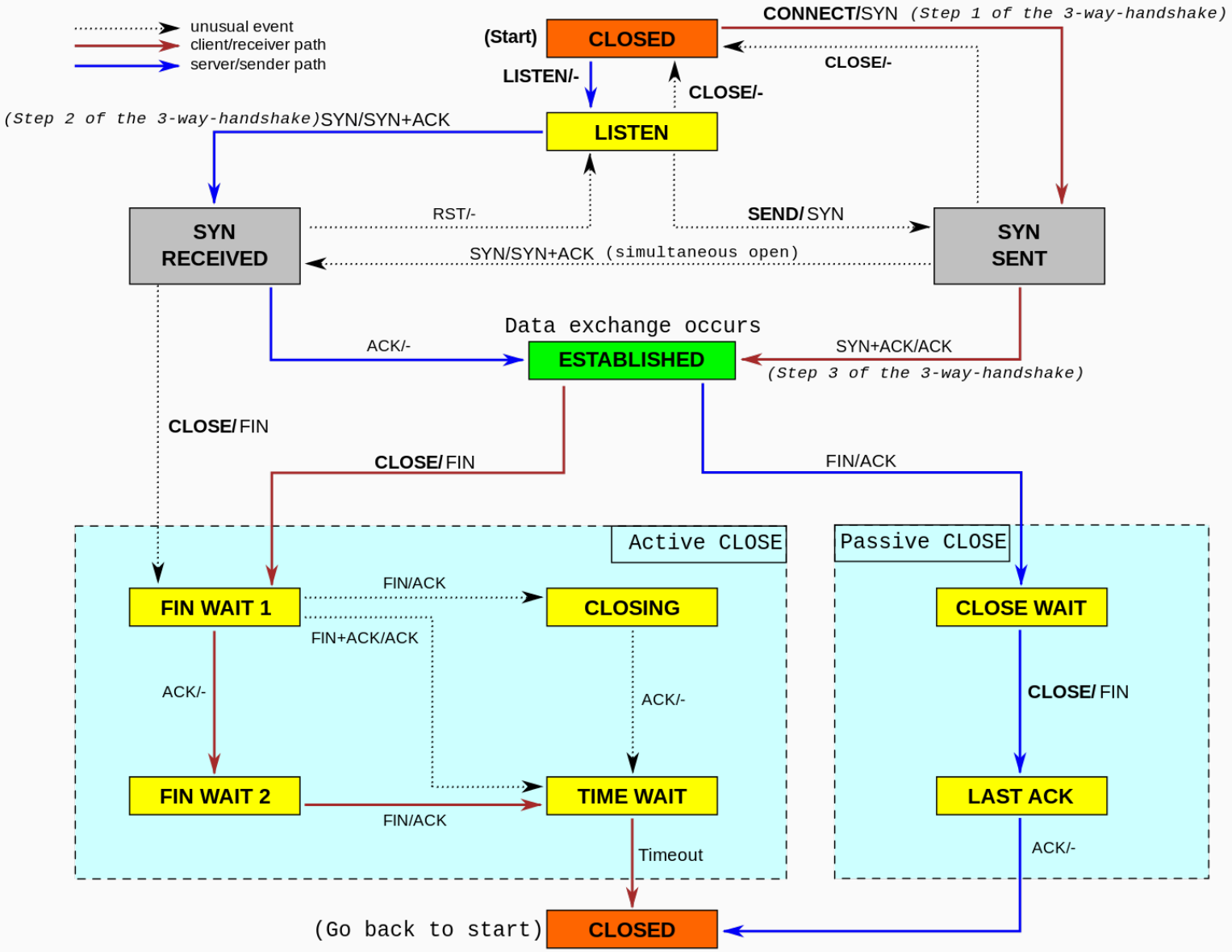
Socket States

Throughout a socket (object)'s lifetime, it goes through a number of states (described by a state machine).

Here are some of the socket states of importance:

- **`LISTEN`**: waiting for a connection request
- **`SYN_RECV`**: received request, still negotiating
- **`ESTABLISHED`**: connection working OK
- **`FIN_WAIT_1/2`**: my side closed the connection (waiting for the other side's ACK and then their `FIN`)
- **`CLOSE_WAIT`**: waiting for the connection to be closed (received `FIN` and sent back an acknowledgement, waiting for our application to OK the closing)
- **`TIME_WAIT`**: waiting for a while ...(2*Maximum Segment Lifetime, so about 4 minutes!)
- **`CLOSED`**: No connection state

Socket States



Socket exhaustion

Active Internet connections (servers and established):

```
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN      1339/rpcbind
tcp        0      0 0.0.0.0:33586           0.0.0.0:*               LISTEN      1395/rpc.statd
tcp        0      0 192.168.122.1:53        0.0.0.0:*               LISTEN      1962/dnsmasq
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN      1586/cupsd
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN      2703/sendmail: acce
tcp        0      0 0.0.0.0:1241            0.0.0.0:*               LISTEN      1851/nessusd: waiti
tcp        0      0 127.0.0.1:25            127.0.0.1:60365         TIME_WAIT   -
tcp        0      0 127.0.0.1:25            127.0.0.1:60240         TIME_WAIT   -
tcp        0      0 127.0.0.1:25            127.0.0.1:60861         TIME_WAIT   -
tcp        0      0 127.0.0.1:25            127.0.0.1:60483         TIME_WAIT   -
tcp        0      0 127.0.0.1:25            127.0.0.1:60265         TIME_WAIT   -
tcp        0      0 127.0.0.1:25            127.0.0.1:60618         TIME_WAIT   -
tcp        0      0 127.0.0.1:25            127.0.0.1:60407         TIME_WAIT   -
tcp        0      0 127.0.0.1:25            127.0.0.1:60423         TIME_WAIT   -
tcp        0      0 127.0.0.1:25            127.0.0.1:60211         TIME_WAIT   -
tcp        0      0 127.0.0.1:25            127.0.0.1:60467         TIME_WAIT   -
tcp        0      0 127.0.0.1:25            127.0.0.1:60213         TIME_WAIT   -
```

Socket exhaustion Mitigation

Enable socket reuse:

- Edit **/etc/sysctl.conf** and add the following line:

```
# Socket reuse
net.ipv4.tcp_tw_reuse=1
```

Increase local port range:

- Edit **/etc/sysctl.conf** and add the following line:

```
# Allowed local port range
net.ipv4.ip_local_port_range=1024 65535
```

Check and learn about the values in:

- **/proc/sys/net/ipv4/tcp_***

Next: DNS Security