MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

Scripting for Cybersecurity

Network Attacks 2 – Port Scanning

Dylan Smyth

- Switch Statements
- TCP
- Port Scanning
- Idle Host Scan

Switch Statements

- Switch Statements are similar to “if else” statements and are often much neater to use when dealing with many possibilities

- For example, a script that changes the number of a month to the name of that month (i.e. 1 = January) might looks like this…

Switch Statements

```
month = 1
month_text = ""

if(month == 1):
  month_text = "January"
elif(month == 2):
  month_text = "February"
elif(month == 3):
  month_text = "March"
elif(month == 4):
  month_text = "April"
elif(month == 5):
  month_text = "May"
elif(month == 6):
  month_text = "June"
elif(month == 7):
  month_text = "July"
elif(month == 8):
  month_text = "August"
elif(month == 9):
  month_text = "September"
elif(month == 10):
  month_text = "October"
elif(month == 11):
  month_text = "November"
elif(month == 12):
  month_text = "December"
print(month_text)
```

Switch Statements

```python
month = 1
month_text = ""

if(month == 1):
  month_text = "January"
elif(month == 2):
  month_text = "February"
elif(month == 3):
  month_text = "March"
elif(month == 4):
  month_text = "April"
elif(month == 5):
  month_text = "May"
elif(month == 6):
  month_text = "June"
elif(month == 7):
  month_text = "July"
elif(month == 8):
  month_text = "August"
elif(month == 9):
  month_text = "September"
elif(month == 10):
  month_text = "October"
elif(month == 11):
  month_text = "November"
elif(month == 12):
  month_text = "December"
print(month_text)
```

```python
month = 1
month_text = ""

month_text = {

  1 : "January",
  2 : "February",
  3 : "March",
  4 : "April",
  5 : "May",
  6 : "June",
  7 : "July",
  8 : "August",
  9 : "September",
  10 : "October",
  11 : "November",
  12 : "December"

}[month]

print(month_text)
```

Switch Statements

- In Python, switch statements are dictionaries
- They're not given a name, so a variable does not hold the dictionary
- They're defined and used to return a value at the same time
- We can return values directly, like in the example, or we could call a function, or return a reference to a function

Switch Statements

```python
def return_value():
  return "some value"

def main():

  get_function = 1
  value = None

  value = {

    1 : return_value()

  }[get_function]

  print(value)
```

Switch Statements

```python
def return_value():
  return "some value"

def main():

  get_function = 1
  value = None

  value = {

    1 : return_value()

  }[get_function]

  print(value)
```

```python
def return_value():
  return "some value"

def main():

  get_function = 1
  value = None

  dict = {1:return_value()}
  value = dict[get_function]

  print(value)
```

Switch Statements

```
def return_value():
  return "some value"

def main():

  get_function = 1
  function = None
  value = None

  function = {

    1 : return_value

  }[get_function]

  value = function()
  print(value)
```

```
def return_value():
  return "some value"

def main():

  get_function = 1
  value = None

  value = {

    1 : return_value

  }[get_function]()

  print(value)
```

Switch Statements

```
def return_value():
  return "some value"

def main():

  get_function = 1
  function = None
  value = None

  function = {

    1 : return_value

  }[get_function]

  value = function()
  print(value)
```

```
def return_value():
  return "some value"

def main():

  get_function = 1
  value = None

  value = {

    1 : return_value

  }[get_function]()

  print(value)
```

Switch Statements

- The following example shows a practical example where we could use a switch statement to change the protocol type field value in the Ethernet header to the readable protocol name (e.g. 0x0800 is IP, 0x0806 is ARP, etc.)

- The main function, containing the call to the sniffing function is not shown in the image

Switch Statements

```
def get_proto_name(proto):
  proto_name = {
    0x0800 : "IP"
  }[proto]
  return proto_name

def count_proto_instances(count):
  def pkt_handler(pkt):
    proto_code = pkt[Ether].type

    try:
      proto_name = get_proto_name(proto_code)
    except KeyError:
      proto_name = "Other"

    if(proto_name in count):
      count[proto_name] += 1
    else:
      count[proto_name] = 1
    print(count)

  return pkt_handler
```

Switch Statements

- If we had a script which was going to perform complex actions for each protocol, we might use a switch statement like this:

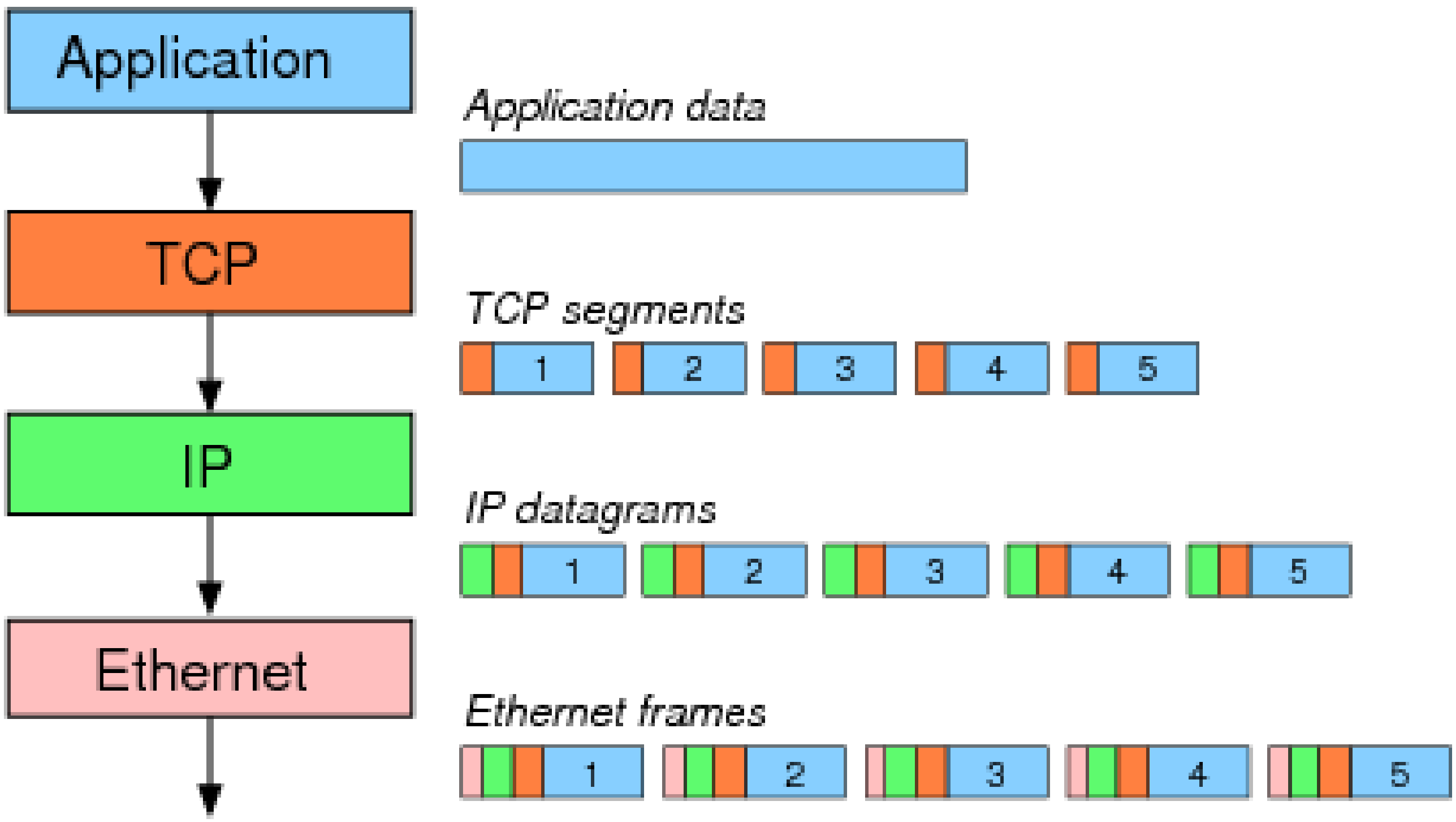
```
def handle_proto(proto, pkt):
  handler = {
    0x0800 : handle_ip
    0x0806 : handle_arp
  }[proto]
  handler(pkt)

def handle_packets():
  def pkt_handler(pkt):
    proto_code = pkt[Ether].type
    handle_proto(proto_code, pkt)
  return pkt_handler
```

TCP

- Transmission Control Protocol (TCP) is a Transport Layer (Layer 4) protocol used to help get data across the network
- Data comes down from the Application Layer (Application -> Presentation -> Session -> Transport)
- The Transport Layer breaks the data up into segments
- The Transport Layer will reconstruct the data at the receiving end and pass it to the Application Layer

TCP


Application
TCP
IP
Ethernet
Application data
TCP segments
1
2
3
4
5
IP datagrams
1
2
3
4
5
Ethernet frames
1
2
3
4
5

TCP

- The two main protocols used by the Transport Layer are TCP and User Datagram Protocol (UDP)

- TCP is reliable as it…
  - Reorders data segments at the receiving end
  - Retransmits lost data

- UDP on the other hand
  - Does not reorder data
  - Does not track or resend lost data

TCP

- The Transport Layer uses port numbers as an addressing mechanism
- Applications listen on ports and if you want to communication with an application over the network you connect to that port
- For example, when you connect to a web server (like google.ie) your browser will connect to the web server on port 80

TCP

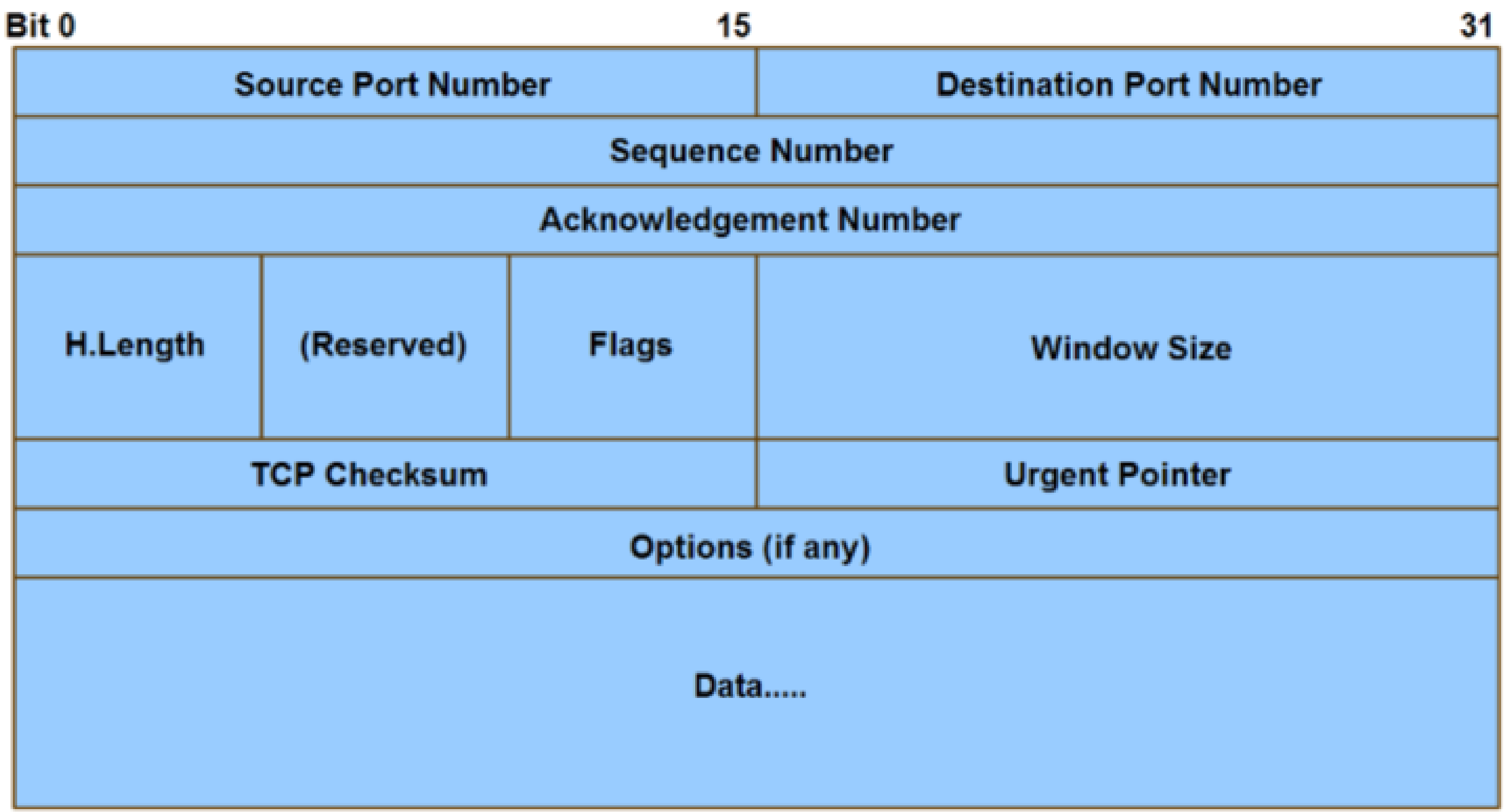
- Another example: Telnet connection between client and a server
- The server had a program running (the telnet server), which listens on port 23 for connections
- The client ran a program which connected to the server on port 23 to establish the Telnet connection
- The pairing of the IP an port is known as a Socket

TCP

- Telnet and HTTP are examples of applications that use TCP to manage data
- The reason being that they need all of the data to be delivered in order to work correctly
- Before sending data, TCP will establish a connection with the destination
- For example, a telnet client will establish a connection with a telnet server before

TCP

TCP Segment Header Fields



The fields of the TCP header enable TCP to provide connection-oriented, reliable data communications.

TCP

- The TCP header contains flag which indicate different types of messages

- These flags are also known as control bits

CWR	Congestion Window Reduced
ECE	ECN-Echo
URG	Urgent
ACK	Acknowledgement
PSH	Push
RST	Reset
SYN	Syn
FIN	Fin

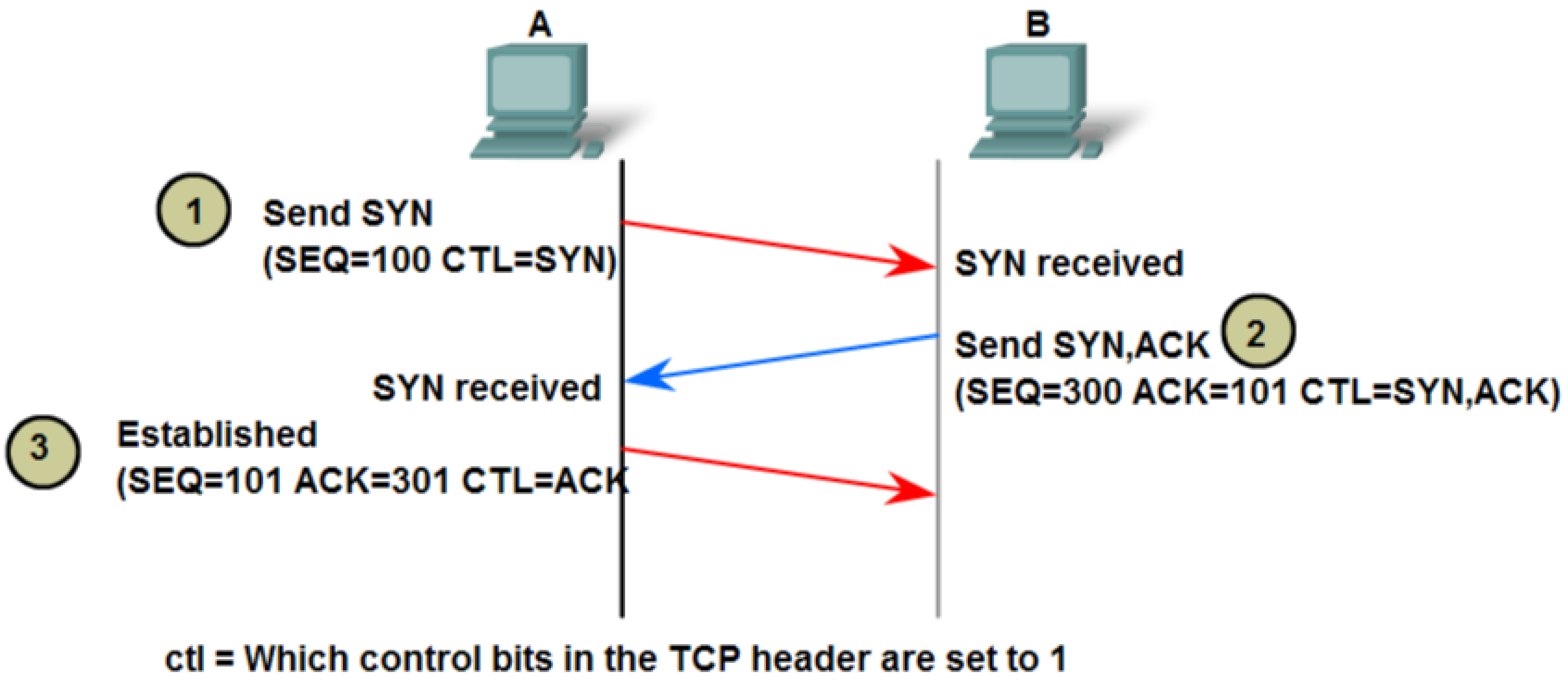
TCP

- TCP uses a handshake to establish a connection, where certain types of messages are sent in a sequence. Once the handshake is complete TCP will then start sending data

- When all of the data has been delivered, TCP will tear down the connection using a different sequence of messages

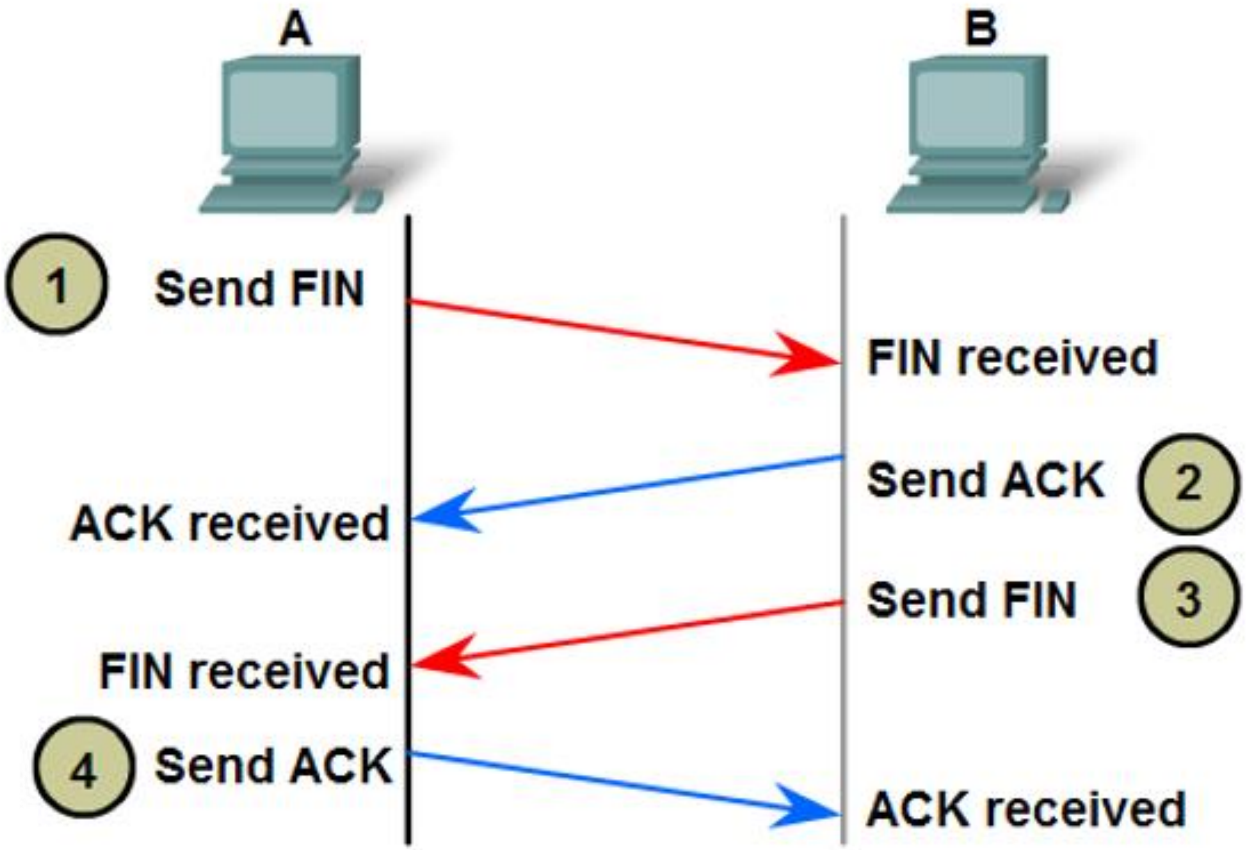
TCP

- The three-way handshake to establish the connection



TCP

- The sequence to tear down the communication session



TCP

- We can see the same in Wireshark

5	4.768883151	10.0.0.2	10.0.0.4	TCP	74	58302 → 23 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PER
6	4.769078195	10.0.0.4	10.0.0.2	TCP	74	23 → 58302 [SYN, ACK] Seq=0 Ack=1 Win=43440 Len=0 MSS=14
7	4.769097185	10.0.0.2	10.0.0.4	TCP	66	58302 → 23 [ACK] Seq=1 Ack=1 Win=42496 Len=0 TSval=22520
8	4.771248753	10.0.0.2	10.0.0.4	TELNET	93	Telnet Data ...
9	4.771265837	10.0.0.4	10.0.0.2	TCP	66	23 → 58302 [ACK] Seq=1 Ack=28 Win=43520 Len=0 TSval=2826
0	4.777309311	10.0.0.4	10.0.0.2	TELNET	78	Telnet Data ...
1	4.777321816	10.0.0.2	10.0.0.4	TCP	66	58302 → 23 [ACK] Seq=28 Ack=13 Win=42496 Len=0 TSval=225
2	4.777352733	10.0.0.4	10.0.0.2	TELNET	105	Telnet Data ...

141	8.070580429	10.0.0.2	10.0.0.4	TCP	66	58306 → 23 [ACK] Seq=152 Ack=842 Wi
142	8.078003225	10.0.0.4	10.0.0.2	TCP	66	23 → 58306 [FIN, ACK] Seq=842 Ack=1
143	8.078028544	10.0.0.2	10.0.0.4	TCP	66	58306 → 23 [FIN, ACK] Seq=152 Ack=8
144	8.078036629	10.0.0.4	10.0.0.2	TCP	66	23 → 58306 [ACK] Seq=843 Ack=153 Wi

TCP

- In the last example where the telnet client connected to the telnet server, port 23 was open on the server
- If TCP tries to connect to a port which is **not** open (i.e. there is no application listing on that port) then a communication session can not be established
- Instead of sending back a SYN-ACK, the server will respond with a RST message

1	0.000000000	10.0.0.2	10.0.0.4	TCP	74 55088 → 888 [SYN] Seq=0 Win=42340
2	0.000203341	10.0.0.4	10.0.0.2	TCP	54 888 → 55088 [RST, ACK] Seq=1 Ack=1

TCP – Port Scanning

- So, if we went a TCP SYN to a port on a server…
  - If the port is open we receive a SYN-ACK
  - If the port is closed we receive a RST

- It's therefore possible to "scan" the ports on a server to detect which ports are open

- By discovering which ports are open we an understand what applications are running on the server

TCP – Port Scanning

- nmap is a well-known tool used for port scanning

```
root@ubuntu-VirtualBox:~# nmap -p 22,23,80,8000 10.0.0.4
Starting Nmap 7.80 ( https://nmap.org ) at 2020-11-15 17:09 GMT
Nmap scan report for 10.0.0.4
Host is up (0.00026s latency).

PORT     STATE  SERVICE
22/tcp   closed ssh
23/tcp   open   telnet
80/tcp   closed http
8000/tcp open   http-alt
MAC Address: 96:A1:0F:98:51:95 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 13.06 seconds
```

TCP – Port Scanning

- We can perform port scanning using Scapy by building and sending TCP messages

```
>>> ls(TCP)
sport      : ShortEnumField                      = (20)
dport      : ShortEnumField                      = (80)
seq        : IntField                            = (0)
ack        : IntField                            = (0)
dataofs    : BitField (4 bits)                   = (None)
reserved   : BitField (3 bits)                   = (0)
flags      : FlagsField (9 bits)                 = (<Flag 2 (S)>)
window     : ShortField                          = (8192)
chksum     : XShortField                         = (None)
urgptr     : ShortField                          = (0)
options    : TCPOptionsField                     = (b'')
```

TCP – Port Scanning

```
>>> ip_hdr = IP(dst="10.0.0.4")
>>> tcp_hdr = TCP(dport=23, flags="S")
>>> pkt = ip_hdr/tcp_hdr
```

```
>>> pkt[IP].show()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= tcp
  chksum= None
  src= 10.0.0.2
  dst= 10.0.0.4
  \options\
```

```
>>> pkt[TCP].show()
###[ TCP ]###
  sport= ftp_data
  dport= telnet
  seq= 0
  ack= 0
  dataofs= None
  reserved= 0
  flags= S
  window= 8192
  chksum= None
  urgptr= 0
  options= []
```

TCP – Port Scanning

- We can see the SA flags in the response

```
>>> resp, unans = sr(pkt, iface="h2-eth0")
Begin emission:
*Finished sending 1 packets.

Received 1 packets, got 1 answers, remaining 0 packets
>>> resp[0][1][TCP].show()
###[ TCP ]###
  sport= telnet
  dport= ftp_data
  seq= 2203525317
  ack= 1
  dataofs= 6
  reserved= 0
  flags= SA
  window= 42340
  chksum= 0x3a64
  urgptr= 0
  options= [('MSS', 1460)]
```

TCP – Port Scanning

- We can see the RST flag if we try a port that is not open

```
>>> resp, unans = sr(IP(dst="10.0.0.4")/TCP(dport=80, flags="S"), iface="h2-eth0")
Begin emission:
.*Finished sending 1 packets.

Received 2 packets, got 1 answers, remaining 0 packets
>>> resp[0][1][TCP].show()
###[ TCP ]###
  sport= http
  dport= ftp_data
  seq= 0
  ack= 1
  dataofs= 5
  reserved= 0
  flags= RA
  window= 0
  chksum= 0x9b66
  urgptr= 0
  options= []
```

TCP – Port Scanning

- The scan being performed here is a SYN scan

- Some other types:
  - Connect Scan -> The full TCP three-way handshake is performed
  - FIN Scan -> Send TCP FIN. No response = port open, RST = closed
  - NULL Scan-> No flag. No response = port open, RST = closed
  - XMAS -> URG, PUSH, FIN flags. No resp = port open, RST = closed

TCP – Idle Host Scan

- The Idle Host Scan (or Zombie host scan) is a stealthy method of port scanning
- It requires an attacker, a target host, and an idle “zombie” host
- The idle host should be a host which isn’t communicating on the network often

TCP – Idle Host Scan

- The attack leverages the workings of TCP, and relies on the IPID field in the IP header to detect an open port

- Without going into too much details regarding the IP protocol, the IP header has an IP ID field. Everytime a device sends an IP packet, the number in this field is incremented

- We can use Scapy to observe the IP IDs and see this…

TCP – Idle Host Scan

- Create a listener on the server

```
>>> def handle_pkt(pkt):
...:     if(IP in pkt):
...:         print(pkt[IP].src + " " + str(pkt[IP].id))
...:
...:
...:
>>> sniff(iface="h4-eth0", prn=handle_pkt)
```

- We can see the following output after a telnet connection from the client

```
>>> sniff(iface="h4-eth0", prn=handle_pkt)
10.0.0.2 39559
10.0.0.4 0
10.0.0.2 39560
10.0.0.2 39561
10.0.0.4 8893
10.0.0.4 8894
10.0.0.2 39562
10.0.0.4 8895
```

TCP – Idle Host Scan

- The attacker chooses a host on the network who is not sending traffic (idle host).
- The attacker sends the idle host a message a logs the IP ID in the response.
- The attacker sends a TCP SYN to a target machine and spoofs the idle host’s address.
- The target responds to the idle host with a SYN ACK or RST
  - SYN ACK -> Will cause the idle host to respond with a RST
  - RST -> Idle host does nothing

TCP – Idle Host Scan

- The target responds to the idle host with a SYN ACK or RST
  - SYN ACK -> Will cause the idle host to respond with a RST
  - RST -> Idle host does nothing

- The attacker sends another message to the idle host and logs the IP ID

- If the idle host responded to the targets SYN ACK then the IP ID will have increased between attack messages (port open)

- If the IP ID has not increased between messages then the port must be closed

- Port scan performed without any trace of communication between the attacker and the target

TCP – Idle Host Scan

- Some operating systems make it difficult to perform an Idle host scan
- Some operating systems will set the IP ID to zero until a connection has been established (as you saw in the image in the previous slide)
- Others will also randomize the IP ID
- IP IDs will be sequential for specific conversations only

MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

Thank you