Lecture Network Security
Chapter 2 – Attack Overview

University of Bonn, Institute of Computer Science IV, Summer 2016

Agenda

Protocol Attacks

- TCP Refresher
- Session Hijacking
- (TCP) DoS Attacks
- The RST Attack
- DNS Spoofing

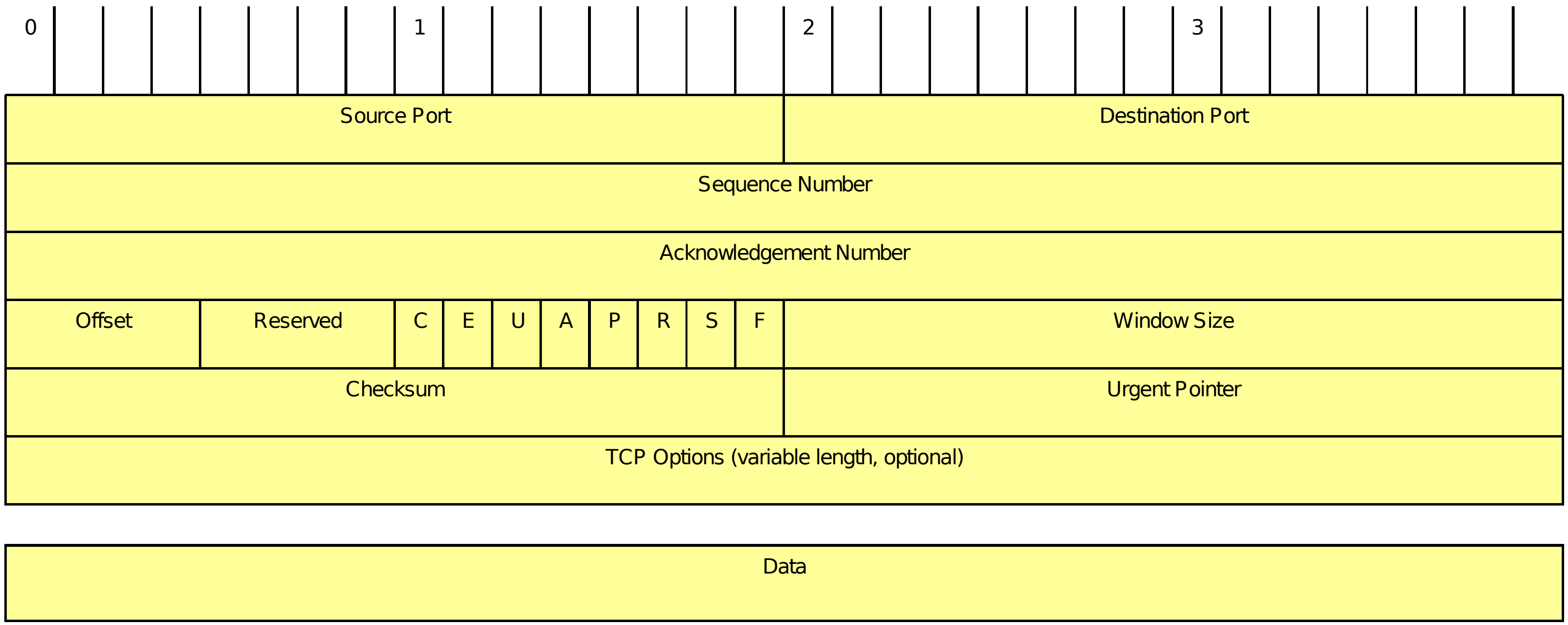
Refresher: Transmission Control Protocol

The Transmission Control Protocol provides **reliable**, **ordered** and **full-duplex byte streams**.

- **Segment-based**: data can be split up into segments
- **Connection-oriented**: special segments for session control
- **Reliable**: segments are acknowledged by the recipient
- **Ordered**: sequence numbers denote a segment’s offset in a stream

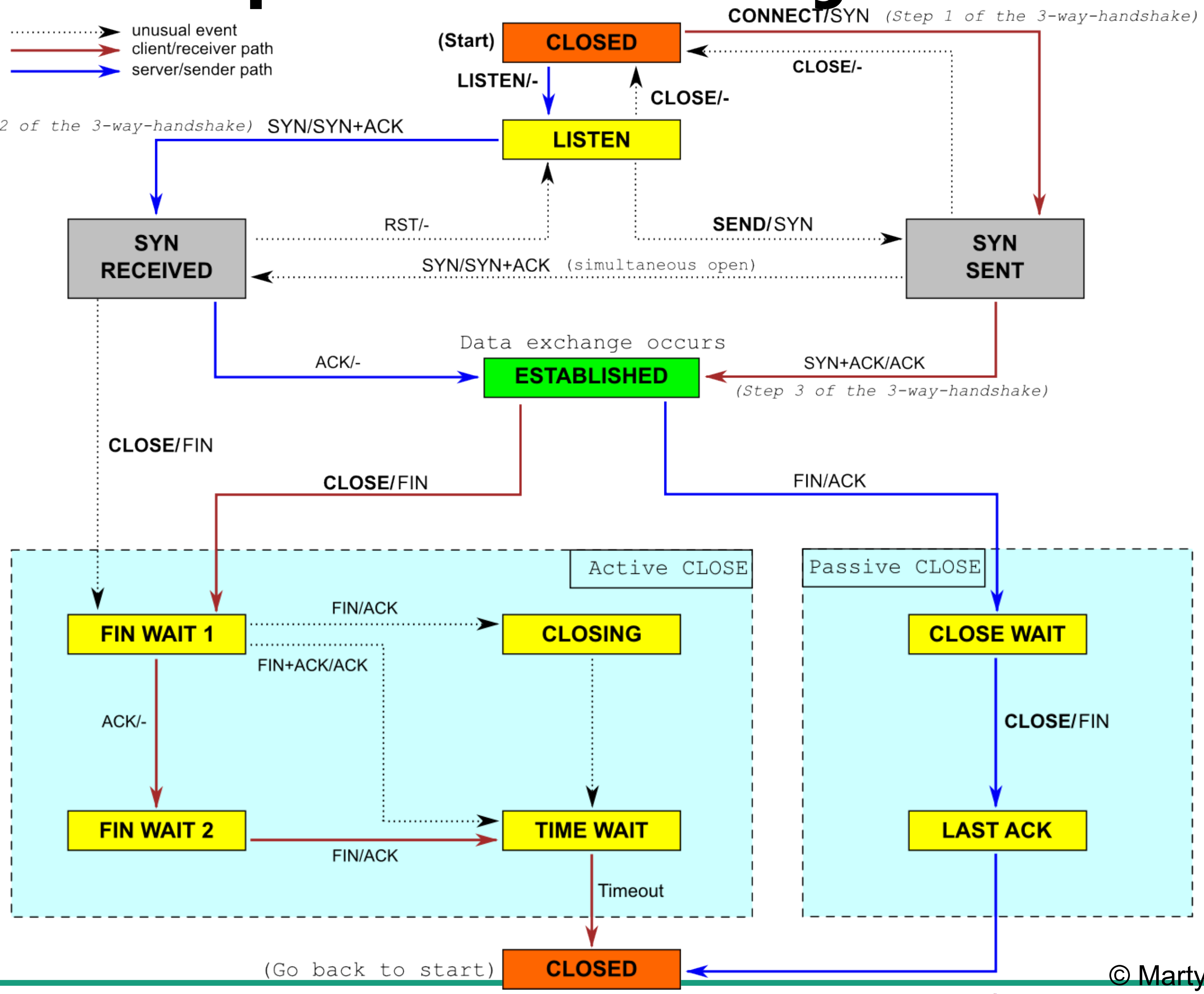
→ **A Session**

Refresher: TCP Header



- At least **20 bytes** in size
- Plus variable-length **options** which are part of the header

Refresher: Simplified TCP State Diagram



Refresher: A TCP Session

```
14:19:12 IP 131.220.6.51.33352 > 131.220.6.26.80: S 4105089174:4105089174(0) win 5840
14:19:12 IP 131.220.6.26.80 > 131.220.6.51.33352: S 930721722:930721722(0) ack 4105089175 win 5792
14:19:12 IP 131.220.6.51.33352 > 131.220.6.26.80: . ack 1 win 46
14:19:12 IP 131.220.6.51.33352 > 131.220.6.26.80: P 1:464(463) ack 1 win 46
14:19:12 IP 131.220.6.26.80 > 131.220.6.51.33352: . ack 464 win 54
14:19:12 IP 131.220.6.26.80 > 131.220.6.51.33352: . 1:1449(1448) ack 464 win 54
14:19:12 IP 131.220.6.51.33352 > 131.220.6.26.80: . ack 1449 win 69
14:19:12 IP 131.220.6.26.80 > 131.220.6.51.33352: . 1449:2897(1448) ack 464 win 54
14:19:12 IP 131.220.6.51.33352 > 131.220.6.26.80: . ack 2897 win 91
14:19:12 IP 131.220.6.26.80 > 131.220.6.51.33352: P 2897:4345(1448) ack 464 win 54
14:19:12 IP 131.220.6.51.33352 > 131.220.6.26.80: . ack 4345 win 114
14:19:12 IP 131.220.6.26.80 > 131.220.6.51.33352: . 4345:5793(1448) ack 464 win 54
14:19:12 IP 131.220.6.51.33352 > 131.220.6.26.80: . ack 5793 win 137
14:19:12 IP 131.220.6.26.80 > 131.220.6.51.33352: . 5793:7241(1448) ack 464 win 54
14:19:12 IP 131.220.6.51.33352 > 131.220.6.26.80: . ack 7241 win 159
14:19:12 IP 131.220.6.26.80 > 131.220.6.51.33352: P 7241:8689(1448) ack 464 win 54
14:19:12 IP 131.220.6.51.33352 > 131.220.6.26.80: . ack 8689 win 182
14:19:12 IP 131.220.6.26.80 > 131.220.6.51.33352: . 8689:10137(1448) ack 464 win 54
14:19:12 IP 131.220.6.51.33352 > 131.220.6.26.80: . ack 10137 win 204
14:19:12 IP 131.220.6.26.80 > 131.220.6.51.33352: . 10137:11585(1448) ack 464 win 54
14:19:12 IP 131.220.6.51.33352 > 131.220.6.26.80: . ack 11585 win 227
14:19:12 IP 131.220.6.26.80 > 131.220.6.51.33352: P 11585:13033(1448) ack 464 win 54
14:19:12 IP 131.220.6.51.33352 > 131.220.6.26.80: . ack 13033 win 250
14:19:12 IP 131.220.6.26.80 > 131.220.6.51.33352: . 13033:14481(1448) ack 464 win 54
14:19:12 IP 131.220.6.51.33352 > 131.220.6.26.80: . ack 14481 win 272
14:19:12 IP 131.220.6.26.80 > 131.220.6.51.33352: P 14481:15292(811) ack 464 win 54
14:19:12 IP 131.220.6.51.33352 > 131.220.6.26.80: . ack 15292 win 295
14:19:12 IP 131.220.6.26.80 > 131.220.6.51.33352: P 15292:15297(5) ack 464 win 54
14:19:12 IP 131.220.6.51.33352 > 131.220.6.26.80: . ack 15297 win 295
14:19:22 IP 131.220.6.26.80 > 131.220.6.51.33352: F 15297:15297(0) ack 464 win 54
14:19:22 IP 131.220.6.51.33352 > 131.220.6.26.80: . ack 15298 win 295
14:19:36 IP 131.220.6.51.33352 > 131.220.6.26.80: F 464:464(0) ack 15298 win 295
14:19:36 IP 131.220.6.26.80 > 131.220.6.51.33352: . ack 465 win 54
```

3-way handshake

full-duplex data exchange

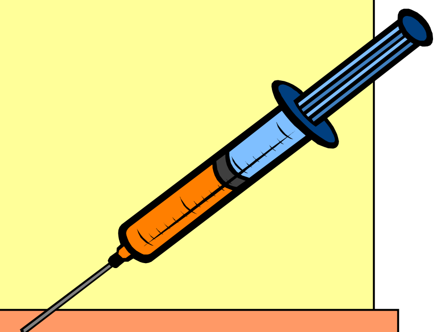
termination

Agenda

1. TCP Refresher
2. Session Hijacking
3. TCP DoS Attacks
4. The RST Attack
5. DNS Spoofing

Session Hijacking: Packet Injection

```
14:19:12 IP 131.220.6.51.33352 > 131.220.6.26.80: S 4105089174:4105089174(0) win 5840
14:19:12 IP 131.220.6.26.80 > 131.220.6.51.33352: S 930721722:930721722(0) ack 4105089175 win 5792
14:19:12 IP 131.220.6.51.33352 > 131.220.6.26.80: . ack 1 win 46
14:19:12 IP 131.220.6.51.33352 > 131.220.6.26.80: P 1:464(463) ack 1 win 46
14:19:12 IP 131.220.6.26.80 > 131.220.6.51.33352: . ack 464 win 54
14:19:12 IP 131.220.6.26.80 > 131.220.6.51.33352: . 1:1449(1448) ack 464 win 54
14:19:12 IP 131.220.6.51.33352 > 131.220.6.26.80: . ack 1449 win 69
14:19:12 IP 131.220.6.26.80 > 131.220.6.51.33352: . 1449:2897(1448) ack 464 win 54
14:19:12 IP 131.220.6.51.33352 > 131.220.6.26.80: . ack 2897 win 91
14:19:12 IP 131.220.6.26.80 > 131.220.6.51.33352: P 2897:4345(1448) ack 464 win 54
14:19:12 IP 131.220.6.51.33352 > 131.220.6.26.80: . ack 4345 win 114
14:19:12 IP 131.220.6.26.80 > 131.220.6.51.33352: . 4345:5793(1448) ack 464 win 54
14:19:12 IP 131.220.6.51.33352 > 131.220.6.26.80: . ack 5793 win 137
14:19:12 IP 131.220.6.26.80 > 131.220.6.51.33352: . 5793:7241(1448) ack 464 win 54
14:19:12 IP 131.220.6.51.33352 > 131.220.6.26.80: . ack 7241 win 159
14:19:12 IP 131.220.6.26.80 > 131.220.6.51.33352: P 7241:8689(1448) ack 464 win 54
14:19:12 IP 131.220.6.26.80 > 131.220.6.51.33352: P 7241:8689(1448) ack 464 win 54
...
```



Can an attacker **inject packets** into a TCP conversation?

- Easiest way: **Snooping** on the session to learn addresses and ports
- Manually craft header fields – source address has to be **spoofed**
- Appropriate **SEQ/ACK numbers**, otherwise the segment will be rejected

Is that really true? What are the exact requirements? More on that later…

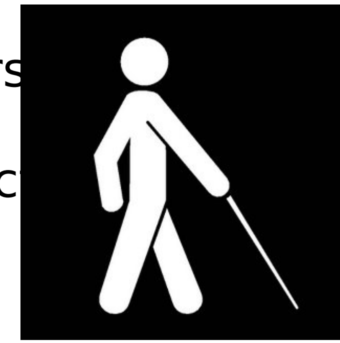
universitätbonn

Fraunhofer FKIE

Session Hijacking: Packet Forgery

It is highly unlikely that an attacker gains a man-in-the-middle position, so snooping on the session is not possible in most cases.

Blind Spoofing: Attacker can guess the relevant parameters



- … port numbers are steadily increased for new connec
- … consecutive ISNs differ by a fixed value,
- … the start values can be reliably determined.

What happens with **replies** to spoofed packets?

- Replies are sent to the original machine (the one owning the spoofed address).
- This machine might send RST packets if there is no related established connection and terminate the session.
- “Solution”: Make that machine unresponsive first (see next section).

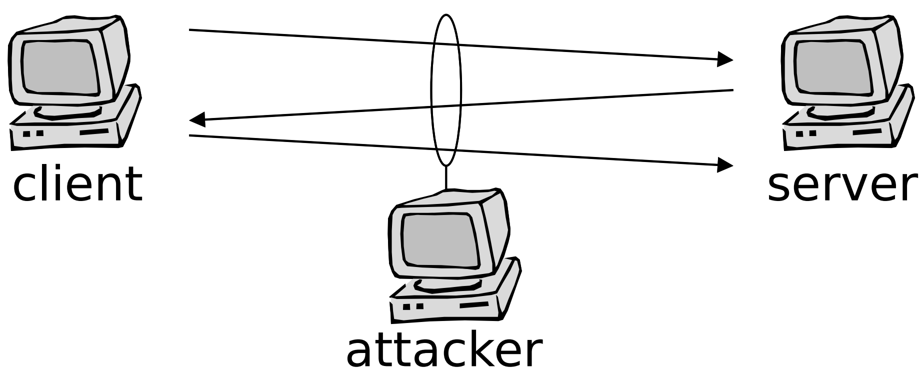
Session Hijacking

acking of TCP connections can be done by guessing TCP sequence numbers d injection of segments containing **commands** in an established stream

It is even possible to **initiate** TCP connections between two remote machines. This idea was used during the *Mitnick attack* in 1994.

This can lead to a full system compromise especially when there are **trust relationships** among those machines: The remote host might have full access rights.

The machine owning the spoofed address must be **made unresponsive** first (see next section).



Session Hijacking: Some History & Countermeasures

Randomized TCP Initial Sequence Numbers

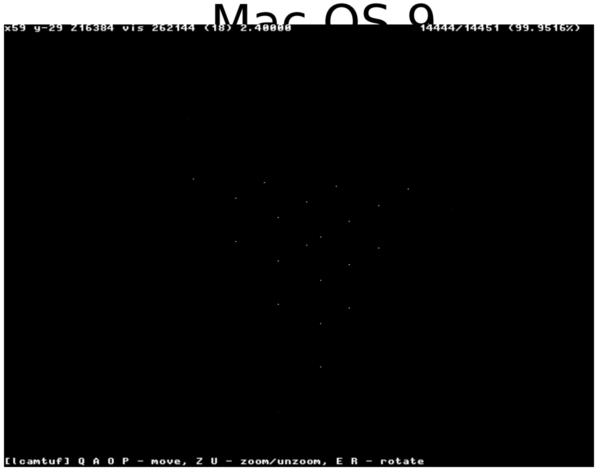
As good as the pseudo-random number generator (PRNG).

RFC1948: Defending Against Sequence Number Attacks, May 1996:

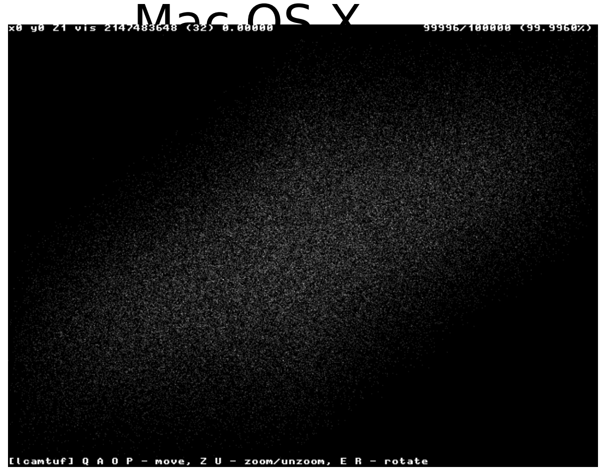
$$ISN = M + F(localhost, localport, remotehost, remoteport)$$

with M a timer and F a **secret hash function**.

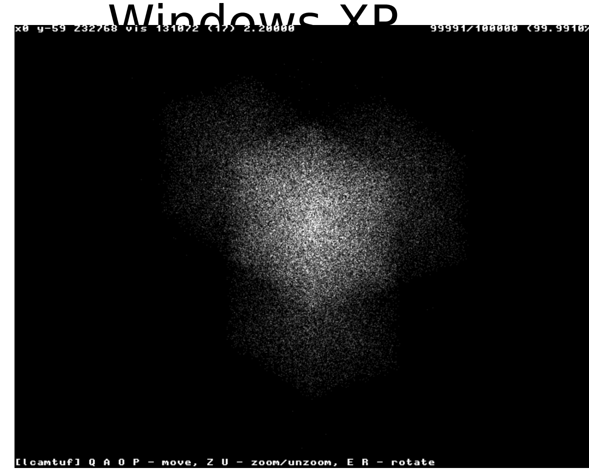
Mac OS 9



Mac OS X



Windows XP



© Michal Zalewski

Agenda

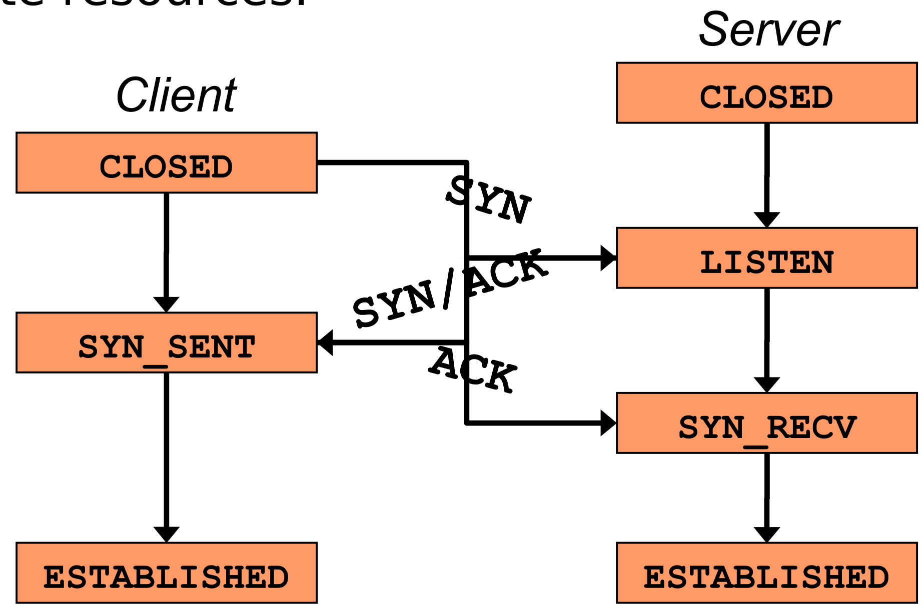
1. TCP Refresher
2. Session Hijacking
3. (TCP) DoS Attacks
4. The RST Attack
5. DNS Spoofing

TCP Denial-of-Service: Observation

SYN Flooding is a so-called **Denial of Service (DoS) attack**

The TCP SYN flooding attack uses the well-known TCP three-way handshake to attack TCP server.

The basic idea is quite simple, an attacker can execute this attack with moderate resources.

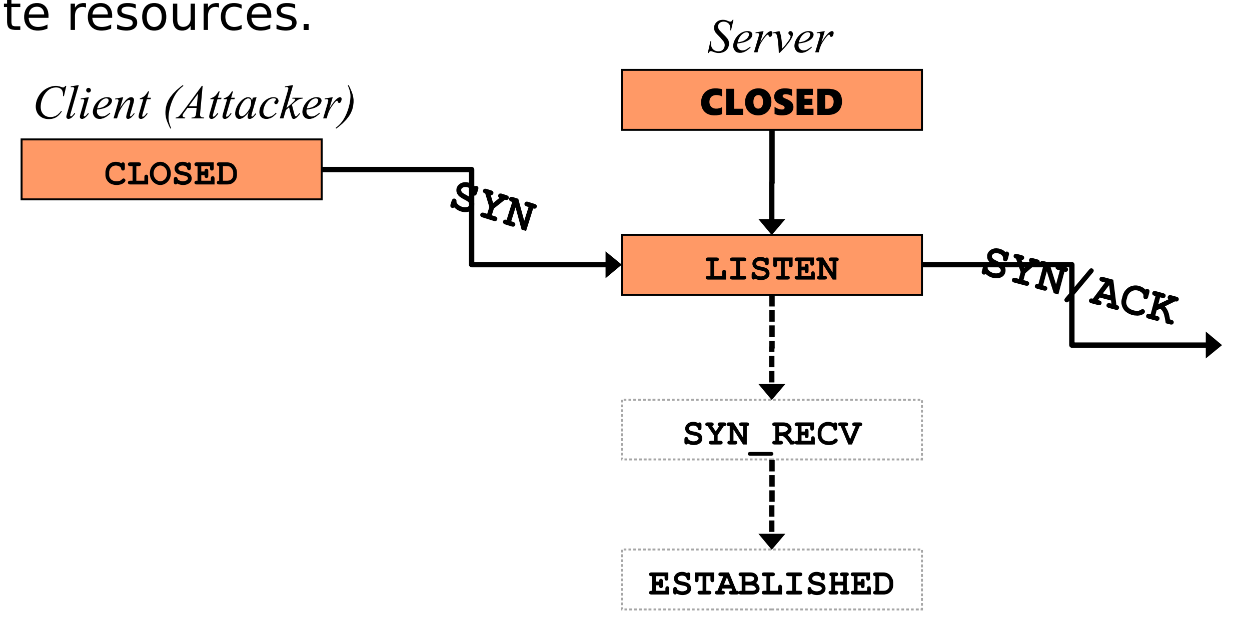


TCP Denial-of-Service: Observation (cont'd)

SYN Flooding is a so-called **Denial of Service (DoS) attack**

The TCP SYN flooding attack uses the well-known TCP three-way handshake to attack TCP server.

The basic idea is quite simple, an attacker can execute this attack with moderate resources.



TCP Denial-of-Service: Observation (cont'd)

```
14:31:40.051840 IP 10.0.0.2.1024 > 10.0.0.1.80: S 916607390:916607390(0)
14:31:40.105032 IP 10.0.0.1.80 > 10.0.0.2:1024: S 3260818460:3260818460(0) ack 916607391
14:31:43.902606 IP 10.0.0.1.80 > 10.0.0.2:1024: S 3260818460:3260818460(0) ack 916607391
14:31:49.902792 IP 10.0.0.1.80 > 10.0.0.2:1024: S 3260818460:3260818460(0) ack 916607391
14:32:01.903792 IP 10.0.0.1.80 > 10.0.0.2:1024: S 3260818460:3260818460(0) ack 916607391
14:32:26.104174 IP 10.0.0.1.80 > 10.0.0.2:1024: S 3260818460:3260818460(0) ack 916607391
14:33:14.305806 IP 10.0.0.1.80 > 10.0.0.2:1024: S 3260818460:3260818460(0) ack 916607391
```

Retransmissions of an unacknowledged SYN/ACK segment occur after 3, 6, 12, 24, and 48 seconds (on a standard Linux system).

The last retransmission has a timeout value of 96 seconds.

Until then, the socket remains in state `SYN_RECV`, which means it uses resources on the server system.

Resources are blocked until the session times out after 3 minutes and 9 seconds!

TCP Denial-of-Service: Idea

An example for a known vulnerability - TCP SYN flooding DoS
The TCP connection establishment:

Connection establishment
(Three-Way Handshake):

Client | Server

(Application needs connection)
Client sends **SYN seq=x** → Server receives **SYN**

The server reserves resources for administration of TCP connection

Client receives **SYN+ACK** ← Server sends **SYN seq=y, ACK x+1**

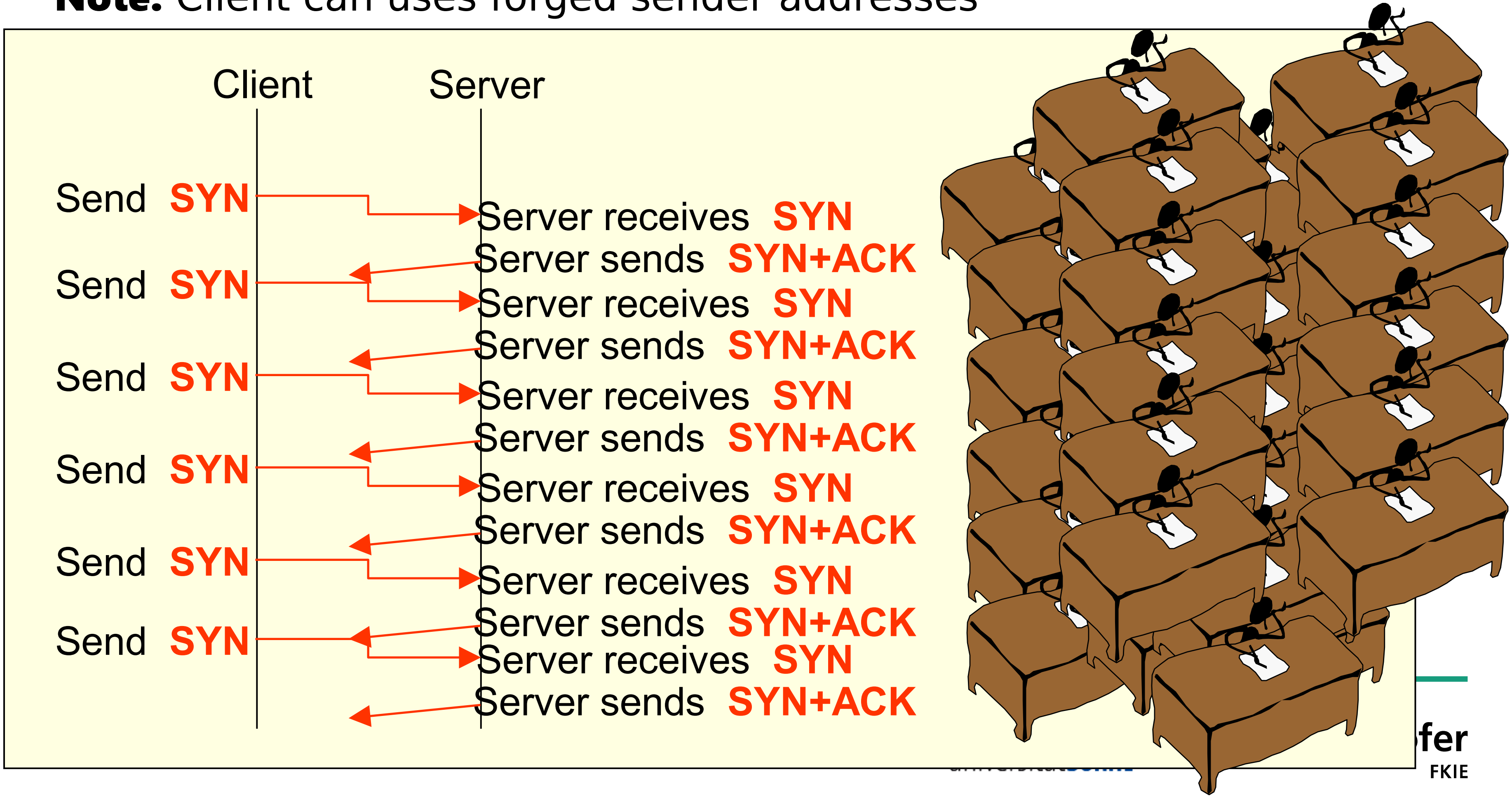
Client sends **ACK y+1** → Server receives **ACK**

TCP Denial-of-Service: The Attack

The initiating system does not need resources for opening connections. Therefore, it can request huge amounts of new connections within a short period of time.

Result: Server is blocked by half-open connections

Note: Client can uses forged sender addresses

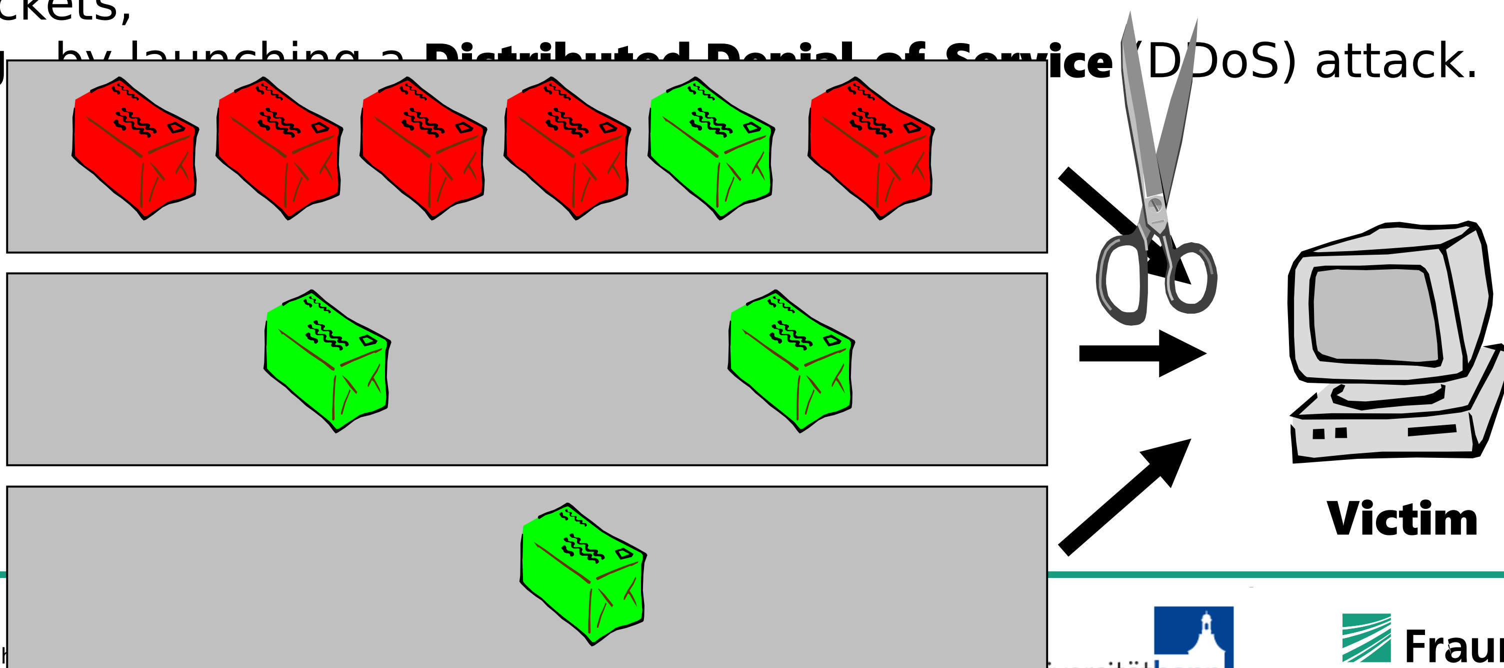


TCP Denial-of-Service: Defense 1 – Filtering

Defending against DoS is **hard** because each data packet belongs to a **request that looks legitimate** to the server.

Early filtering of the packets is necessary **before they reach the target system**.
But: Filtering is only possible if malicious and regular traffic can be distinguished.

Fortunately, to date such **filter criteria** can be identified in most cases.
Still, link congestion can always be achieved by sending even more packets,
e.g., by launching a **Distributed Denial-of-Service** (DDoS) attack.



TCP Denial-of-Service: Defense 2 – SYN Cookies

SYN Cookies are a method to proactively protect against SYN floods.

The problem was:
In case of attack, the backlog queue (bookkeeping of half-open connections) of the server fills at a rapid rate. Once the queue is full, all subsequent SYN packets are dropped.

Solution:
Move session state tracking into the session itself:
The server calculates a hash value from

- source and destination port,
- client IP address, and
- a secret.

This hash value is taken as the Initial Sequence Number (ISN) in a SYN packet

Resource allocation on the server for keeping track of half-open connections (a backlog queue) is not necessary anymore.

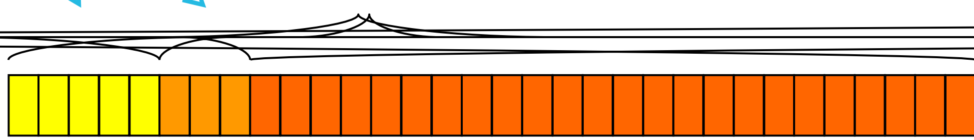
TCP Denial-of-Service: Defense 2 – SYN Cookies (cont'd)

Exact calculation of initial sequence number is

t mod 32, t is a 32-bit time counter that increases every 64 seconds

an encoding of a MSS selected by the server in response to the client's MSS

a server-selected secret function of the IP addresses and port numbers, and t



When the server receives an ACK packet (3rd packet of three-way handshake), it is able to verify whether this is a packet answering an own SYN/ACK packet.

In this case, the TCP connection is established. The MSS value can be reconstructed from the corresponding bits in the ACK number.

Implementations for several operating systems (including Linux, BSD, …) are available.

Reference: Dan Bernstein, http://cr.yp.to/syncookies.html

TCP Denial-of-Service: Going Further

What if an attacker completes the three-way handshake?

New attack: **Establish as many connections as possible**. Once the maximum number of open connections is reached, further connection attempts will be dropped.

The attacker can even use SYN cookies himself. This would allow for session establishment without binding local resources.

Keeping sessions alive: Many servers reset connections that are idle for some time. But sending and receiving data on a really slow rate and retransmissions keep the session up.

In addition, there are 1001 ways to trick a server into **allocating even more resources** for a session.

E.g., the attacker advertises a rather large receive buffer (window size), requests lots of data from the server, but does not acknowledge it.