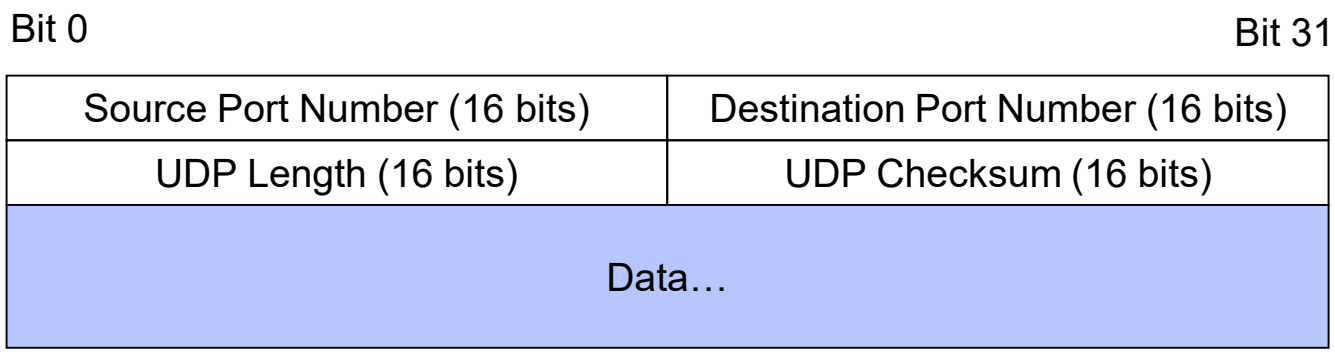
Transport Layer Security

Reading material:
Security assessment of the TCP Protocol by CPNI (see Canvas)
Read chapters 5 to 8, 12 and 14.

Or RFC draft: https://tools.ietf.org/html/draft-ietf-tcpm-tcp-security-03

User Datagram Protocol (UDP)



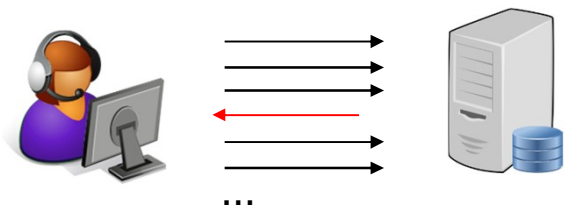
UDP

- Source IP address and port number spoofing possible
  - No security in UDP
  - Send data to victims from well-known port numbers
  - ntp/123, syslog/514, DNS queries and replies/53, media streaming, ...
  - May believe that data comes from a legitimate sender?
  - Never rely on IP address or source port number for security

- Easy to insert UDP datagrams into an ongoing stream
  - No sequence numbers = impossible to detect on UDP level
  - Application protocol may or may not detect inserted datagrams
  - Firewalls may not know how to inspect *application-level* protocol
  - Even if they do, is it possible to know what is faked?

Port scanning (TCP and UDP)

- Goal: find services (open ports) on selected hosts
- Noisy process: 65,535 ports for both TCP and UDP
  - Can be reduced to 2,000 if only well-known services are searched
  - Try likely existing port numbers (80, 443, 22, etc.)
- Testing all ports on a subnet with 254 machines → 16,500,000 packets
- This is a good reason for having an intrusion detection (IDS) system
  - Trivial to detect
  - No IDS = we won't even see this noisy attack



UDP port scanning

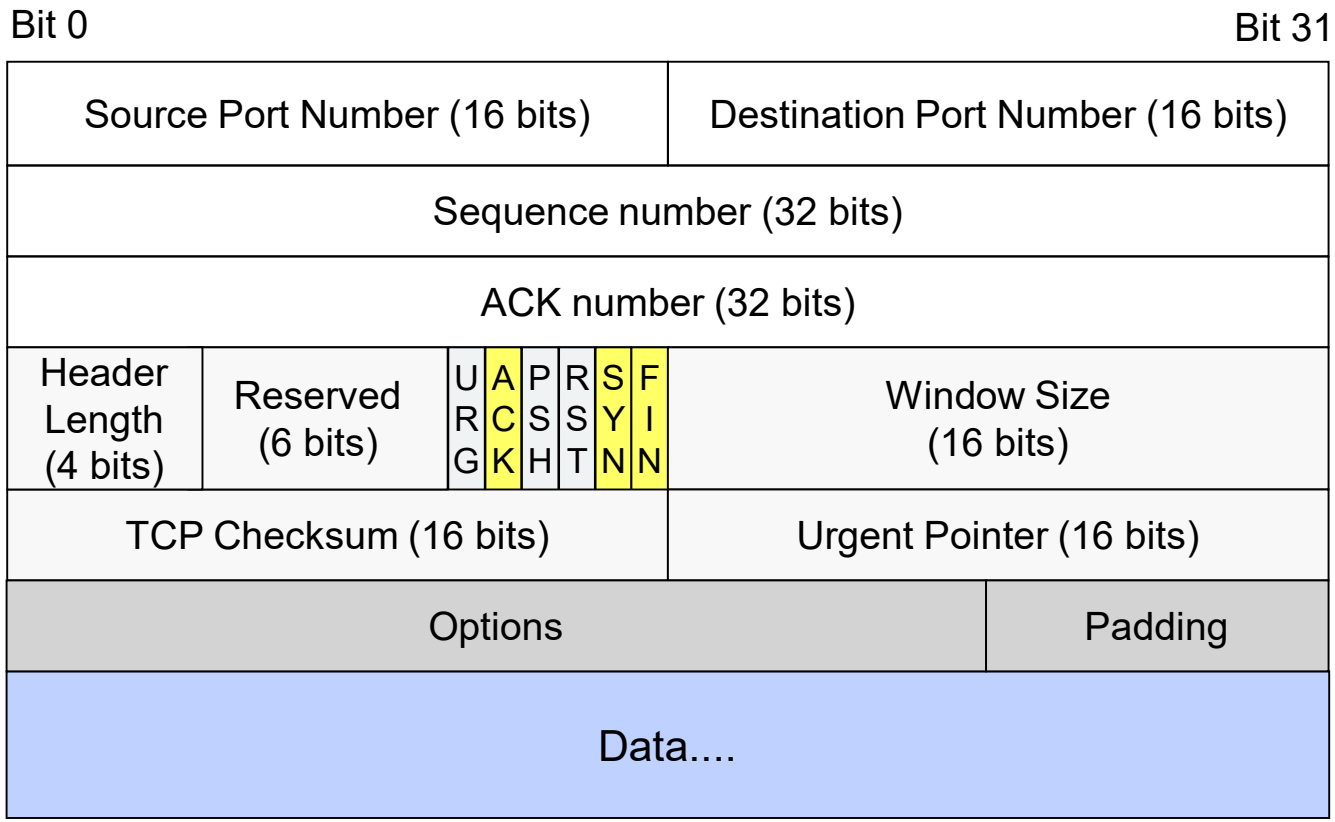
- Hard: from UDP itself, we get no reply (except possibly from ICMP)
  - But maybe the application-level protocol responds?
  - *TCP is easier: a SYN generates a SYN/ACK if port is open*
- Scan: Send a 0 byte datagram to each port
  - Does not interfere with application
  - If ICMP port unreachable is received, host exists but port is closed. If no reply, we don't know
  - Firewalls may filter out ICMP responses to make hosts invisible
- Many operating systems have defense against this (RFC 1812) by limiting number of ICMP messages sent out:
  - Windows and Linux 1 per second → 18 hours for a complete scan (configurable)

```
icmp_ratelimit (integer; default: 1000; since Linux 2.4.10)
       Limit the maximum rates for sending ICMP packets whose type
       matches icmp_ratemask (see below) to specific targets.  0 to
       disable any limiting, otherwise the minimum space between
       responses in milliseconds.

icmp_ratemask (integer; default: see below; since Linux 2.4.10)
       Mask made of ICMP types for which rates are being limited.
```

https://man7.org/linux/man-pages/man7/icmp.7.html

TCP Header


Bit 0
Bit 31
Source Port Number (16 bits)
Destination Port Number (16 bits)
Sequence number (32 bits)
ACK number (32 bits)
Header Length (4 bits)
Reserved (6 bits)
URG
ACK
PSH
RST
SYN
FIN
Window Size (16 bits)
TCP Checksum (16 bits)
Urgent Pointer (16 bits)
Options
Padding
Data....

TCP port scanning (SYN scan)

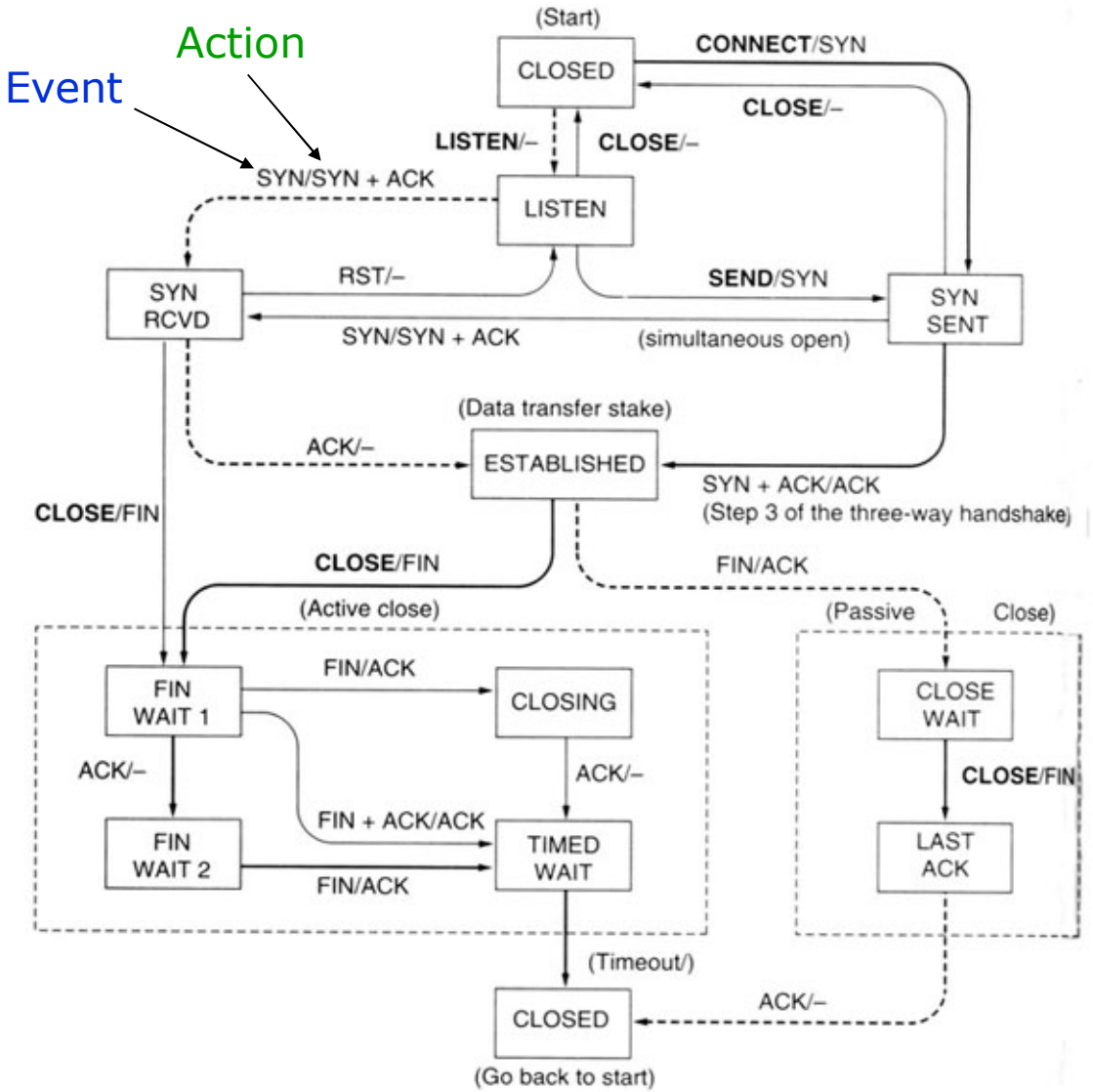
- The most popular scan type
  - RFC 793: Traffic to a **closed** port: always return **RST**
  - If port open, SYN/ACK is sent back
  - If we don't complete the handshake (no ACK) → the application will not be notified
  - Firewalls may drop the SYN or send ICMP port unreachable
    (nmap will show this port as *filtered* by a firewall)
- Partly established connections are visible on the target with the "netstat" command (next slide)
  - Therefore programs like nmap immediately sends RST after receiving SYN/ACK
  - Third party programs like tcplog, Synlogger and Courtney can detect and log these scans
- Noisy process to scan many ports
  - Scan fewer systems and ports
  - Scan more slowly to avoid detection
  - Scan one host but from different IP addresses
  - Scan many hosts instead of one if systems are equally configured

```
> netstat

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    127.0.0.1:52453        CSE-1089208:52454      ESTABLISHED
  TCP    127.0.0.1:52454        CSE-1089208:52453      ESTABLISHED
  TCP    127.0.0.1:52844        CSE-1089208:52845      ESTABLISHED
  TCP    127.0.0.1:52845        CSE-1089208:52844      ESTABLISHED
  TCP    129.16.23.8:52449      107.152.24.197:https   CLOSE_WAIT
  TCP    129.16.23.8:52450      74.112.184.85:https    CLOSE_WAIT
  TCP    129.16.23.8:52456      107.152.24.197:https   CLOSE_WAIT
  TCP    129.16.23.8:52835      file09:microsoft-ds    ESTABLISHED
  TCP    129.16.23.8:52847      webmail:imaps          ESTABLISHED
  TCP    129.16.23.8:54143      74.112.185.182:https   CLOSE_WAIT
  TCP    129.16.23.8:54144      74.112.185.182:https   CLOSE_WAIT
  TCP    129.16.23.8:54146      74.112.185.182:https   CLOSE_WAIT
  TCP    129.16.23.8:54170      162.125.17.3:https     ESTABLISHED
  TCP    129.16.23.8:55014      162.125.17.1:https     CLOSE_WAIT
  TCP    129.16.23.8:60004      sol:microsoft-ds       ESTABLISHED
  TCP    129.16.23.8:60011      soleil:microsoft-ds    ESTABLISHED
  TCP    129.16.23.8:62879      webmail:https          ESTABLISHED
  TCP    129.16.23.8:62880      webmail:https          ESTABLISHED
  TCP    129.16.23.8:62885      webmail:https          ESTABLISHED
  TCP    129.16.23.8:62930      webmail:https          ESTABLISHED
  TCP    129.16.23.8:62932      ec2-52-3-172-96:https  CLOSE_WAIT
  TCP    129.16.23.8:62935      webmail:https          ESTABLISHED
  TCP    129.16.23.8:62937      webmail:https          ESTABLISHED
  TCP    129.16.23.8:62975      74.112.184.86:https    ESTABLISHED
  TCP    129.16.23.8:62984      print:microsoft-ds     ESTABLISHED
  TCP    129.16.23.8:62987      green:epmap            TIME_WAIT
  TCP    129.16.23.8:62988      green:1026             TIME_WAIT
  TCP    129.16.23.8:62990      server-54-192-1-170:https  ESTABLISHED
  TCP    [::1]:65382            CSE-1089208:65384      ESTABLISHED
  TCP    [::1]:65384            CSE-1089208:65382      ESTABLISHED
```

The TCP state machine RFC-793 (setup and close states)

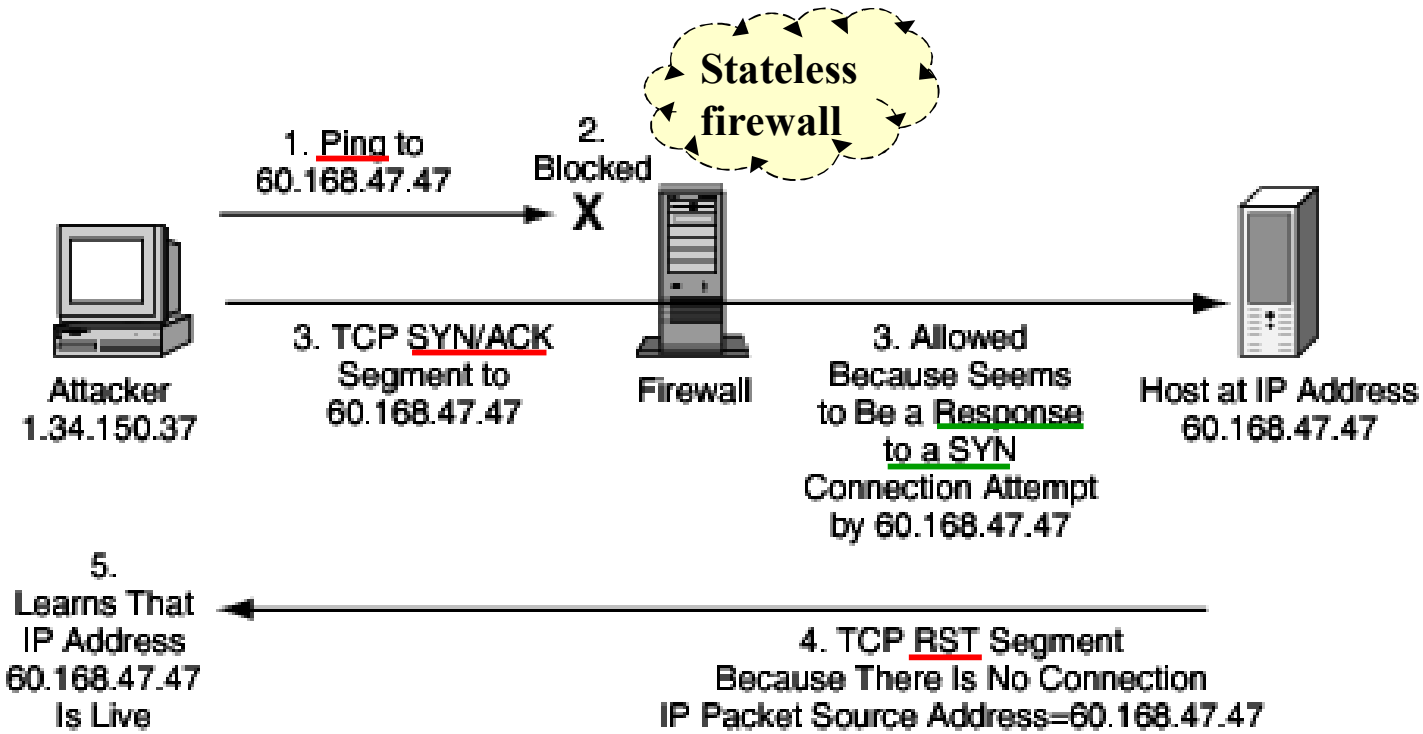


TCP port scanning (ACK scan)

- nmap option "-sA" performs an ACK scan
  - RFC 793: Traffic to a **closed** port: always return **RST**
  - **Incorrect ACKs** (addresses and port numbers don't match) send **RST**
- A firewall may (should) drop invalid ACK and RST
  - no response → nmap reports "port filtered"
- Not visible in the system as the SYN scan (e.g. using "netstat" command)
- ACKs can be passed through non-stateful firewalls
  - If it does not keep state, ACKs must be allowed – it may just block SYNs to prevent new connections
  - If reply is RST know someone is there at that IP address
  - Can also be used to see if a firewall is stateful
- The RST responses may leak more information
  - If window_size=0 port is closed, if > 0 port is open
  - Some systems decrement TTL by one for open ports, other do not

From the NMAP manual: "*Let your creative juices flow, while evading IDS systems whose vendors simply paged through the Nmap man page adding specific rules!*"

TCP port scanning (SYN/ACK scan)



TCP port scanning (FIN scan)

- Similar to SYN scan
  - RFC 793: Traffic to a **closed** port: always return **RST**
  - If port is **open** and segment does **not** contain SYN, RST, or ACK: RFC 793: *"you are unlikely to get here, but if you do, **drop** the segment"*
- A FIN scan can make use of this:
  - RST if the port is closed
  - And no response if the port is open as long as SYN, RST or ACK is not present
- Not visible in the system as the SYN scan (by "netstat" command)
- But some systems always return RST
  - Windows best example
  - If so, we don't know if port is open or not (just that the server exists)
  - But other systems follow the RFC …

TCP port scanning (NULL, XMAS, etc.)

- **NULL** scan
  - Invalid, no flags at all set – no SYN, ACK or RST present
  - Same responses as FIN scan (RST or dropped)
- **XMAS** tree scan
  - Try all invalid combinations
  - URG + PSH + FIN + … (all flags set)
  - SYN/FIN, SYN/ACK/FIN, SYN/FIN/RST, …
  - Result is implementation dependent – see TCP state machine

Firewalls detect scans

- Or use special software in hosts
- Example: Cisco firewall (ASA)

3040	400026	TCP NULL flags	Attack	Triggers when a single TCP packet with none of the SYN, FIN, ACK, or RST flags set has been sent to a specific host.
3041	400027	TCP SYN+FIN flags	Attack	Triggers when a single TCP packet with the SYN and FIN flags are set and is sent to a specific host.
3042	400028	TCP FIN only flags	Attack	Triggers when a single orphaned TCP FIN packet is sent to a privileged port (having port number less than 1024) on a specific host.

TCP Fingerprinting

- Goal: Identify version of an operating system and/or application
  - Windows 10 version 22H2, LINUX 6.2, …
  - Useful because most exploits are specific to particular programs or versions
- Passive fingerprinting
  - Listen to packets and look at parameters (TTL, window size, etc.)
- Active fingerprinting
  - Send TCP, IP or ICMP messages,
  - Send malformed or illegal messages, e.g. to port 0 where response is system dependent
- Administrators may change for example TTL in a server to confuse!
- Wikipedia lists fingerprinting tools:

Fingerprinting tools [edit]

A list of TCP/OS Fingerprinting Tools

- Zardax.py[8] – Passive open source TCP/IP Fingerprinting Tool.
- Ettercap – passive TCP/IP stack fingerprinting.
- Nmap – comprehensive active stack fingerprinting.
- p0f – comprehensive passive TCP/IP stack fingerprinting.
- NetSleuth – free passive fingerprinting and analysis tool
- PacketFence[9] – open source NAC with passive DHCP fingerprinting.
- Satori – passive CDP, DHCP, ICMP, HPSP, HTTP, TCP/IP and other stack fingerprinting.
- SinFP – single-port active/passive fingerprinting

Project Memoria stack identifier tool

The script identifies the TCP/IP stacks on a target device via four active fingerprinting methods:

- ICMP probing: the script performs a malformed ICMP echo request and checks for characteristics of the reply, including changes in the Time-to-live (TTL) value and specific payload content, which varies per stack.
- TCP options signatures: the script sends a TCP SYN packet and monitors the TCP SYN ACK response for the format of the TCP options field. Each stack replies with different values for the options, such as a Maximum Segment Size (MSS) and window scale.
- TCP Urgent flag handling: the script sends a TCP packet with the Urgent flag set and monitors the response. Each stack replies with a different set of TCP flags and a different TCP Window size value.
- HTTP banners and error messages: the script performs HTTP requests to a webserver hosted on a device and checks for specific HTTP headers and application-specific error messages.
- SSH banners and error messages: the script performs requests to an SSH server and checks for specific SSH banners.
- FTP banners and error messages: the script performs requests to an FTP server and checks for specific FTP banners.

```
$ sudo -E python project-memoria-detector.py -ip_dst 192.168.212.42 -p 80 -i eth1
    Host 192.168.212.42 runs uIP/Contiki TCP/IP stack (High level of confidence)

$ sudo -E python project-memoria-detector.py -ip_dst 192.168.212.42 -p 80 -i eth1 -v
Host IP: 192.168.212.42
        ICMP fingerprint => uIP/Contiki (High level of confidence)
        TCP fingerprint => failed to determine the TCP/IP stack (reason: No match)
        HTTP fingerprint => failed to determine the TCP/IP stack (reason: No match)
```

https://github.com/Forescout/project-memoria-detector

Stacks identified 2021:

- uIP, Contiki or Contiki-NG
- picoTCP or picoTCP-NG
- Nut/Net
- FNET
- Nucleus NET
- CycloneTCP
- NDKTCPIP
- uC/TCP-IP
- MPLAB Harmony Net
- NicheStack
- Nucleus Net
- FreeBSD
- Microsoft ThreadX
- CMX TCP/IP
- emNet
- Keil TCP/IP
- lwIP

Examples of TCP fingerprints

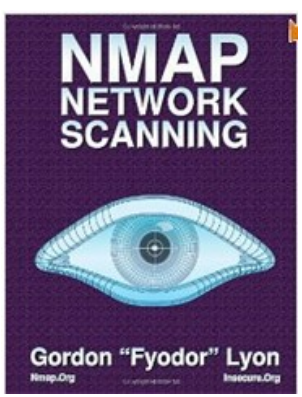
OS	Version	Window size	TTL*	DF	SYN size	Other characteristics
Windows	95b	8192	128	Y	64	MSS, ...
	XP sp1 & 2	32767 or 64512	128	Y	48	MSS, ...
	2003 AS	32768	32	Y	52	MSS=1460
	Vista/7/8/10	8192	128	Y	52	wscale=8
Solaris	10	32850	64	Y	64	
Linux	2.6.17	4*MSS	64	N	60	wsacle=5
Cisco	IOS 11.0	3800-5000	255	N	44	MSS=536

* A value of 64 is recommended in RFC 1700

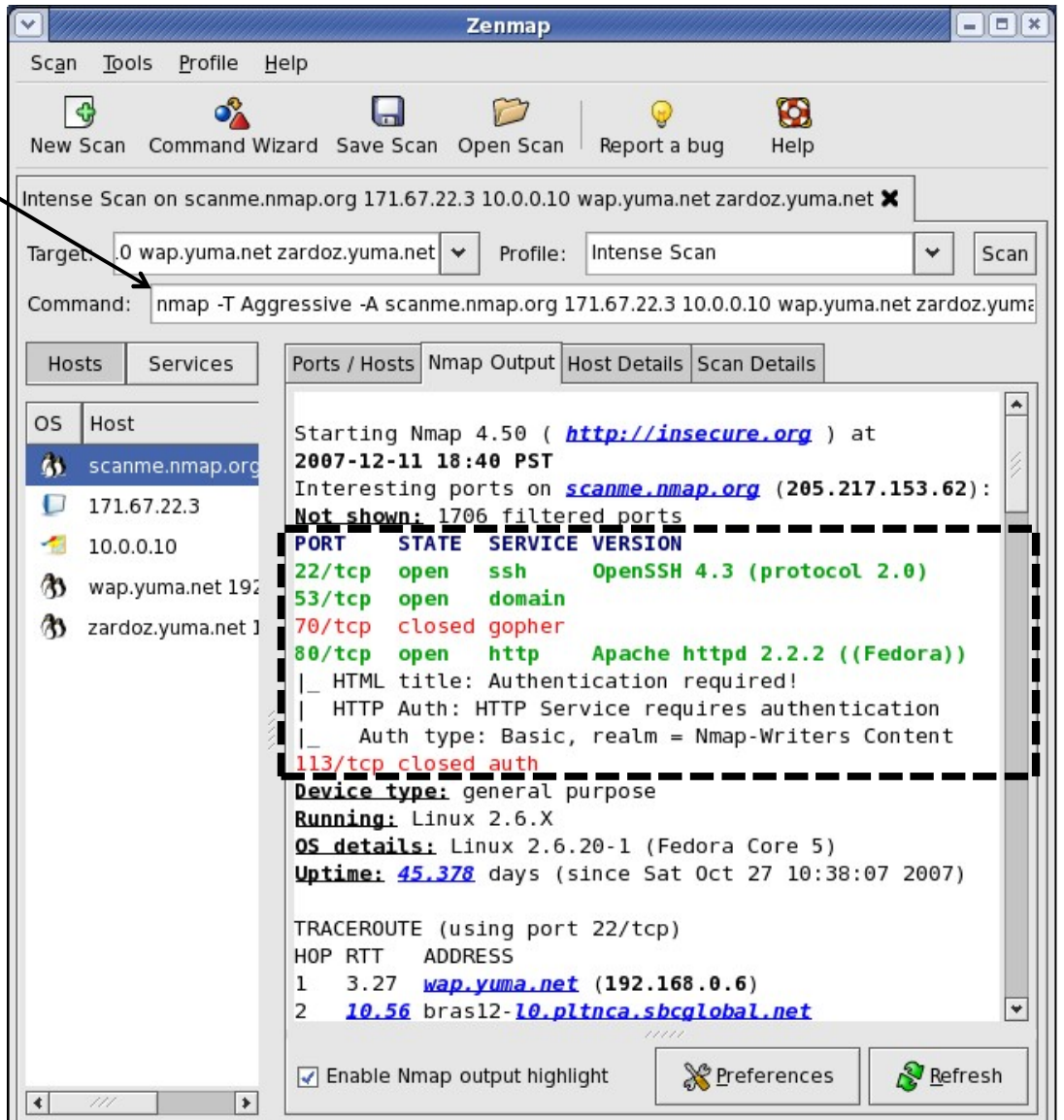
More info: p0f (passive fingerprint tool): https://lcamtuf.coredump.cx/p0f3/

Nmap Security Scanner

- Nmap – Network Mapper (used in the labs)
- Freeware tool
  - For systems administrators
  - Recommended by Linux Journal, Microsoft, etc.
  - Used in this course
- Available for most systems
  - Command-line interface, GUI as an add-on
- Programmer has written a book ---------->
- Many scan options:
  - All scans described here
  - And many more …



NMAP Port Scanning with OS Fingerprinting

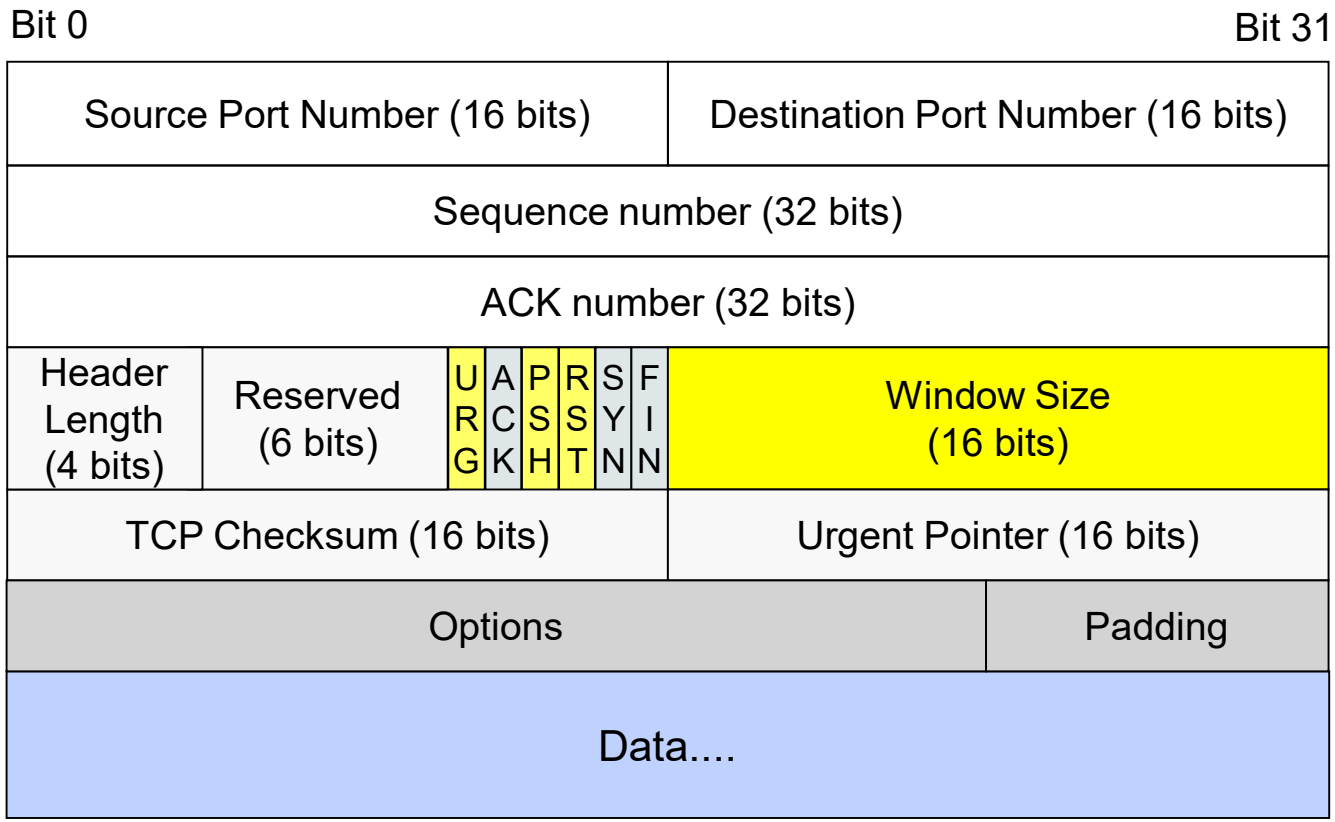


https://nmap.org/book/zenmap.html

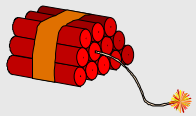
After scanning the targets…

- The operating systems is known
- Applications (services) running on these machines are known
  - In many cases both type and version
- Network structure may be known
  - The DMZ may host many services…
  - Start with the easiest to attack, then move on: A → B → C→ …
  - Trust between machines may be found with the IDLE/Dumb scan
- Known vulnerabilities are regularly published
  - Script kiddies
  - Try different attacks: guess passwords, try known web server bugs, SQL injection, …
- Attackers can test and plan attacks off-line
  - Can set up similar system at home to make sure attack works the first time – without traces
- Worst case scenario for the attacker: to have to find own vulnerabilities ☹
  - Hopefully they go somewhere else… which is what we want to achieve ☺

TCP Header

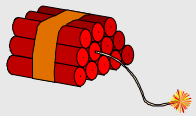

Bit 0
Bit 31
Source Port Number (16 bits)
Destination Port Number (16 bits)
Sequence number (32 bits)
ACK number (32 bits)
Header Length (4 bits)
Reserved (6 bits)
URG
ACK
PSH
RST
SYN
FIN
Window Size (16 bits)
TCP Checksum (16 bits)
Urgent Pointer (16 bits)
Options
Padding
Data....

Zero Window Size Vulnerability

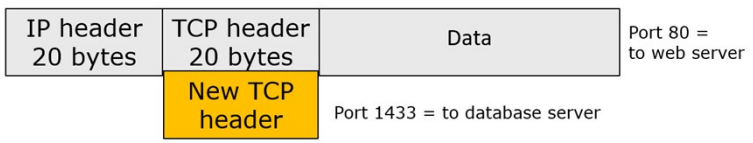


- This DoS attack discovered Sept 2008: CVE-2008-4609
- Problem: Attacker creates many connections to a server and sets TCP window size to 0 or a few bytes
  - Keeps connections alive but server can never send any data
  - Victims may become slow, unresponsive and not accept new connections or even crash
- Even if the application (e.g. web server) closes connection, TCP will hang in FIN-WAIT state
  - It wants to send the last data before sending its FIN
  - Impact (result) varies by implementation (DoS)

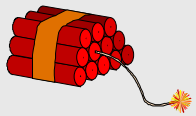
TCP and small IP fragments



- TCP header should be present only in the first **IP fragment**
  - IP header + TCP header = 20+20 bytes
  - No networks need to fragment datagrams < 40 bytes
  - In fact, 512 bytes should always work
- What if a fragment **overwrites** already inspected parts?
  - Overlaps may change port number for TCP/UDP
  - Fragmentation makes packet inspection by firewalls much harder
  - Firewalls may have to reassemble fragments before inspection – not good
- Firewalls should not allow short fragments
  - Always drop fragments overwriting packet headers
  - Good firewalls automatically discard ”short fragments”
- In most (?) environments, all fragmented datagrams can be dropped
  - TCP figures out MTU and does not need fragmentation
  - Fragmentation is very rare today



TCP flag misuse



- Packets with flag PSH + ACK
  - TCP stack is supposed to immediately deliver received data to application
  - It normally waits for more data to arrive before application is notified
  - Intended for protocols where the sender knows what it is doing
  - Victim may be overloaded with work
  - Example: web requests where each segment only contains one letter

- TCP urgent flag (URG) informs victim that some data should bypass the queue
  - E.g. interrupt (CTRL-C) in Telnet but otherwise rarely used
  - May cause confusion (next slide)

- TCP performance algorithms can be attacked
  - Inserting 3 ACKs for someone else may make the sender go back to “fast retransmit” or “slow start” mode
  - Sending false window size updates can decrease performance

Example: Bugs in industrial control systems

- … all ABB products that use the 3rd generation of the AC 800PEC controller are potentially affected
- Summary: WindRiver is the provider of the real time operating system … recently become aware of security vulnerabilities in the TCP/IP stack (IPnet) which is used in the AC 800PEC
- All Ethernet based protocols are affected. In certain scenarios this would lead to the controller becoming inaccessible

CVE	Title	CVSSv3 Score
CVE-2019-12256	Stack overflow in the parsing of IPv4 packets' IP options	9.8
CVE-2019-12257	Heap overflow in DHCP Offer/ACK parsing inside ipdhcpc	8.8
CVE-2019-12255	TCP Urgent Pointer = 0 leads to integer underflow	9.8
CVE-2019-12260	TCP Urgent Pointer state confusion caused by malformed TCP AO option	9.8
CVE-2019-12261	TCP Urgent Pointer state confusion during connect() to a remote host	8.8
CVE-2019-12263	TCP Urgent Pointer state confusion due to race condition	8.1
CVE-2019-12258	DoS of TCP connection via malformed TCP options	7.5
CVE-2019-12259	DoS via NULL dereference in IGMP parsing	6.3
CVE-2019-12262	Handling of unsolicited Reverse ARP replies (Logical Flaw)	7.1
CVE-2019-12264	Logical flaw in IPv4 assignment by the ipdhcpc DHCP client	7.1
CVE-2019-12265	IGMP Information leak via IGMPv3 specific membership report	5.4

2023

2023-03-27: Cyber Security Advisory - ABB RCCMD – Use of default password

2023-03-07: Cyber Security Advisory - ABB Substation management unit COM600 IEC-104 protocol stack vulnerability

2023-03-01: Cyber Security Advisory - Improper authentication vulnerability in S+ Operations

2023-02-27: Cyber Security Advisory - IEC 61850 Communication Stack vulnerability, impact on ABB AC 800PEC and AC 800PEC-based products

2023-02-27: Cyber Security Advisory - Vulnerable TigerVNC Version used in B&R Products

https://search.abb.com/library/Download.aspx?DocumentID=9AKK107492A6671
https://global.abb/group/en/technology/cyber-security/alerts-and-notifications

Attacks known by “one good firewall”

LAND attack and Smurf attacks

Winnuke attack (Netbios out-of-bound)

Unknown IP protocol

Reassembly attacks

Syndrop

Teardrop 2

Opentear

Tentacle

Ping of Death attack

Nestea

Big ping

Targa 3

Newtear

Bonk, Boink

IP fragment overlap and 1st length change

Too many IP fragments

Very small IP fragments

Empty fragment

SSPing

Flushot

IP Spoofing across network

Twinge

TCP SYN flood.

IP source route option detection

Jolt and Jolt2

Ascend attack

TCP XMAS scan

Octopus

Overdrop

Echo / chargen

Ascend Kill

Mime flood

Zero length IP option

IP unaligned time stamp

ICMP router advertisement

Snork attack

Fraggle attack

UDP short header

TCP header fragmentation

TCP short header

TCP null scan

TCP sequence out of range

TCP FIN (Stealth)

TCP postconnection SYN

TCP invalid urgent offset

RFProwl

Blind spoofing

W2K domain controller attack

FTP bounce attack

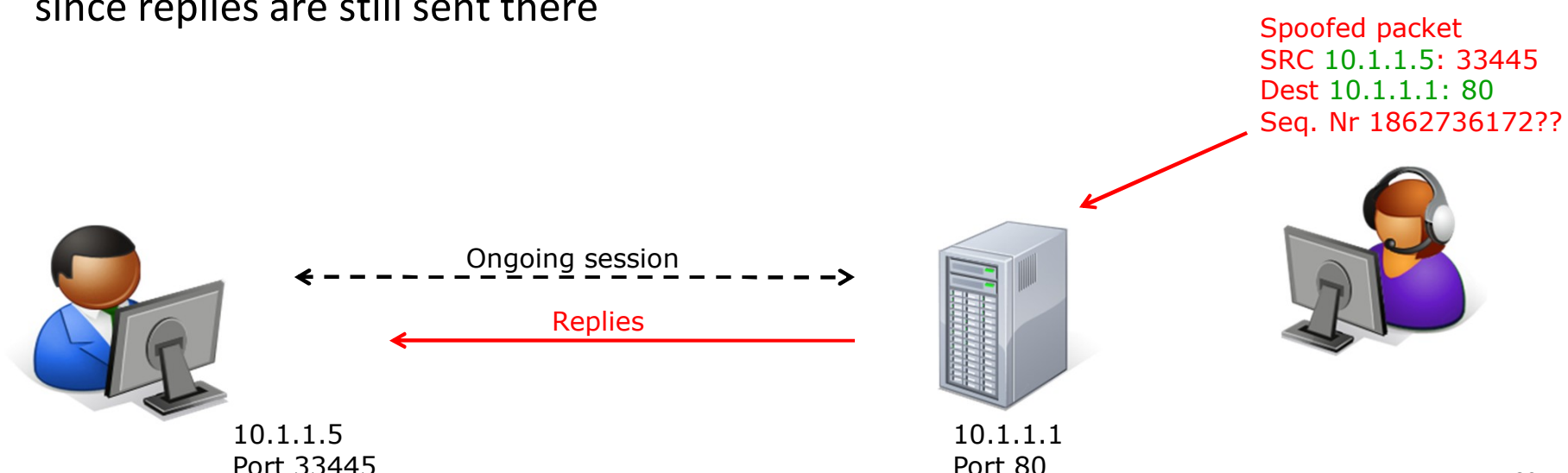
Sequence number prediction

Rose attack

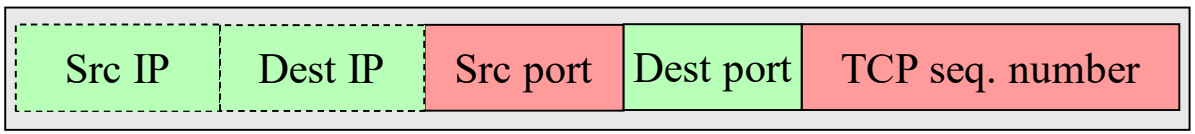
Session hijacking through TCP sequence number prediction

Blind session hijacking – how hard is it?

- Goal: Take over a complete session
  - As a man-on-the-side (MOTS) and not MITM
  - Attacker cannot see TCP sequence numbers
  - The original source must be silenced (DoS attack?) since replies are still sent there



Spoofing TCP segments



- Three fields more or less known (source IP address, server IP address, server port)
- Source IP address may be guessed or be a static address which can be found in a mail header?
  `Received: from [129.16.11.37] by targaryen.ita.chalmers.se`
- Source port (16 bits) must be guessed
  - Some applications use fixed (known) port numbers on both client and server side (e.g. BGP, …)
  - Most systems start using port 1024 after reboot for clients and increase it by 1 for each new connection
- TCP sequence number (32 bits) must be guessed
  - If numbers are predictable, it is possible to guess numbers in use
- Protection: TCP sequence numbers should be randomly selected at connection setup
  - Many (older?) implementations use predictable sequence numbers
  - However, it is enough if TCP sequence number is within receiver's window
  - 64k window => "only" 16 bits remains to be guessed (and 2^{16} = 65,536)
- 1,000,000 packets/sec (2^{20}) can be sent on a 1 Gbps link

TCP sequence numbers

Connecting several times to a service reveal how sequence numbers are selected

- The NMAP utility can estimate how hard they are to predict

Why use unique sequence numbers?

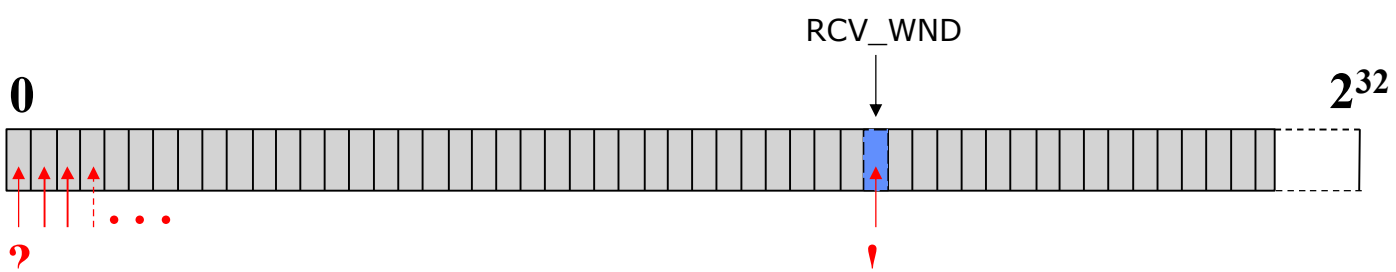
1. RFC 793: To avoid accepting old incarnations of segments (from old connections)
   - New numbers must differ from all older connections
   - A counter that constantly increment numbers is ok
2. And now today, also to protect against TCP sequence number guessing
   - They should be completely **random**
   - Satisfying (1.) may violate (2.)
   - Good random numbers require a good source for entropy
   - May be hard especially for an IoT device to generate good random numbers after reboot

Selecting the initial TCP sequence number, ISN

- **RFC 793:** ISNs should be unique per 4-tuple (addresses and ports)
  - A global 32-bit counter is increased every 4 µs – lasts 4.5 hours
  - Therefore easy to predict what numbers connections use
- Complication: full randomness requires that state of old connections are kept
- **RFC 6528** suggests a compromise:
  - Algorithm must be fast – no overhead in connection establishment
  - Calculate a new ISN as:
    - ISN = M + hash(source_IP, source_port, dest_IP, dest_port, secret_random_nr)
    - M = monotonically increasing counter increased every 4 µs
    - Connections reusing port numbers will have different ISNs
    - And that there is (some) randomness involved – although the increment of M predictable
    - secret_random_nr should change after reboot and after a pre-defined or random time
- Linux (not RFC 6528 compliant)
  - Recalculates secret_random_nr every five minutes
  - Collisions may occur, but rarely a real problem

Predicting sequence numbers – RST attacks (1)

- Attacker only needs to match a seq. number within the window for data to be inserted
- Large windows needed for high-speed connections since more data in flight
  - Larger window = fewer guesses needed (see next slide)
- Instead of inserting own data, a TCP RST → closes the connection immediately if inside window



Predicting sequence numbers – RST attacks (2)

- Discussions started when BGP router RST attacks became popular
  - BGP port numbers known and sessions extremely long
  - RST spoofing → other routers believe a router is down when connection is closed – affected the whole Internet
- Likelihood to insert a RST (RFC 4953):

BW	BW*delay	RSTs needed	Time needed
10 Gbps	125 MB	35	1 us (microsecond)
1 Gbps	12.5 MB	344	110 us
100 Mbps	1.25 MB	3,436	10 ms (millisecond)
10 Mbps	0.125 MB	34,360	1 second
1 Mbps	0.0125 MB	343,598	2 minutes
100 Kbps	0.00125 MB	3,435,974	3 hours

Figure 1: Time needed to kill a connection

Bandwidth-delay product: number of bits that fill up the network link (number to transmit before ACK comes)

Fast and long links can contain many packets in flight!

https://datatracker.ietf.org/doc/html/rfc4953 - RFC4953 Defending TCP Against Spoofing Attacks

Protection against RST attacks – RFC 5961

- Improving TCP's Robustness to Blind In-Window Attacks [RFC 5961]
  - Changes proposed due to security/DoS problems
- Only accept **RST** that exactly matches the next expected sequence number

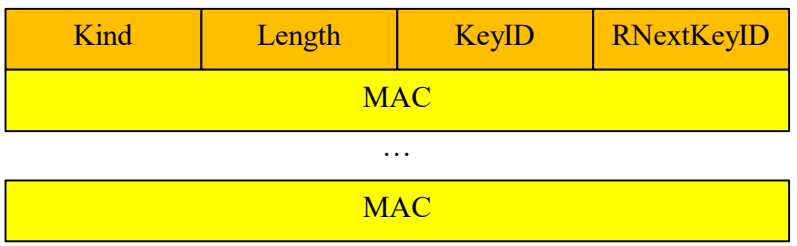
```
if within window but not expected seq_num:
  – Drop RST and send ACK with expected seq_num   ("challenge ACK")
  – Sender sends new RST with acceptable number
```

- A spoofed RST now only generates an extra ACK
- And to prevent data insertion, limit how far ahead segments can be accepted
- TCP Authentication Option (TCP-AO) is a possibility – used for BGP

TCP Authentication Option: TCP-AO

- RFC 5925 and 5926
- Suitable for all long-lived connections
  - Originally designed for BGP routing protocol protection
  - Not yet widely accepted and implemented outside BGP
- Authentication info (a MAC) is sent in each TCP segment
  - Done in a TCP Option header (type=29)
  - Negotiation and setup not done by TCP
  - Algorithm and key negotiation done by out-of-band protocol or manually
  - MAC algorithm example: HMAC-SHA1 (RFC 5926)
  - Different keys in each direction
- Also prevents port scanning
- Alternatives to TCP-AO can be IPsec or TLS

TCP option field:



Kind = 29 (i.e. TCP option 29)

Length = length of this option

KeyID = Master key used to generate message (i.e. session) keys

RNextKeyID = the desired “receive next” key ID (what the other end wants us to use when we send)

Summary

- Port scanning – to find computers and services. Tools like NMAP can be used.
  - SYN/ACK, ACK, FIN, …
- Small fragment attacks (IP attack really, but TCP headers affected)
  - Fragments may overwrite original TCP header to confuse firewalls
- Port number spoofing (TCP and UDP)
  - Don't rely on source port numbers (or IP addresses)
- UDP attacks
  - Possible to insert packets if IP addresses and port numbers can be guessed.
    Known for many services: DNS, p2p, games, …
- TCP packet injection
  - For example, to terminate connections send faked RST
  - Necessary to match IP addresses, port numbers and TCP sequence numbers
  - Trivial if we can see TCP sequence numbers on our network
  - Blind hijacking requires TCP sequence number prediction – much harder
- TCP session hijacking – take over a session
  - Need to keep one party busy (flood with data or send RST) while talking to the other