Lecture 15

Transport Layer Security/ Secure Socket Layer (TLS/SSL)

(Chapter 9 in KPS)

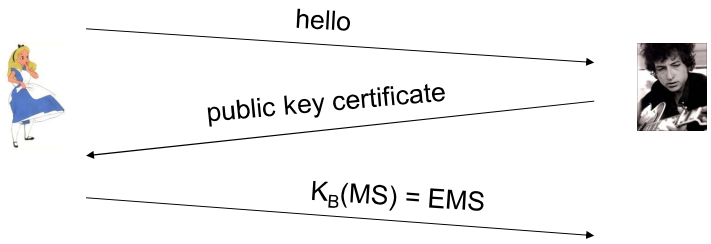
•1

Toy SSL/TLS: a Simple Secure Channel

- *handshake:* Alice and Bob use their certificates, private keys to authenticate each other and exchange a shared secret
- *key derivation:* Alice and Bob use shared secret to derive set of keys
- *data transfer:* data to be transferred is broken up into series of records
- *connection closure:* special messages to securely close connection

•2

Toy: a Simple Handshake



MS: master secret

EMS: encrypted master secret (using Bob's public key)

•3

Toy: Key Derivation

- considered bad to use same key for more than one cryptographic operation
  - use different keys for message authentication code (MAC) and encryption
- **four keys**:
  - K_c = encryption key for data sent from client to server
  - M_c = MAC key for data sent from client to server
  - K_s = encryption key for data sent from server to client
  - M_s = MAC key for data sent from server to client
- keys derived from key derivation function (KDF)
  - takes master secret and (possibly) some additional random data and creates the keys

•4

Toy: Data Records

- why not encrypt data in constant stream as we write it to TCP?
  - where would we put the MAC? If at end, no message integrity until all data processed.
  - e.g., with instant messaging, how can we do integrity check over all messages in a session before displaying?
- instead, break stream in series of records
  - each record carries a MAC
  - receiver can act on each record as it arrives
- issue: in record, receiver needs to distinguish MAC from data
  - want to use variable-length records



•5

Toy: Sequence Numbers

- *problem:* attacker can capture and replay or re-order records
- *solution:* put sequence number into MAC:
  - MAC = MAC(M_x, sequence||data)
  - note: no sequence number field, M_x = MAC key

•6

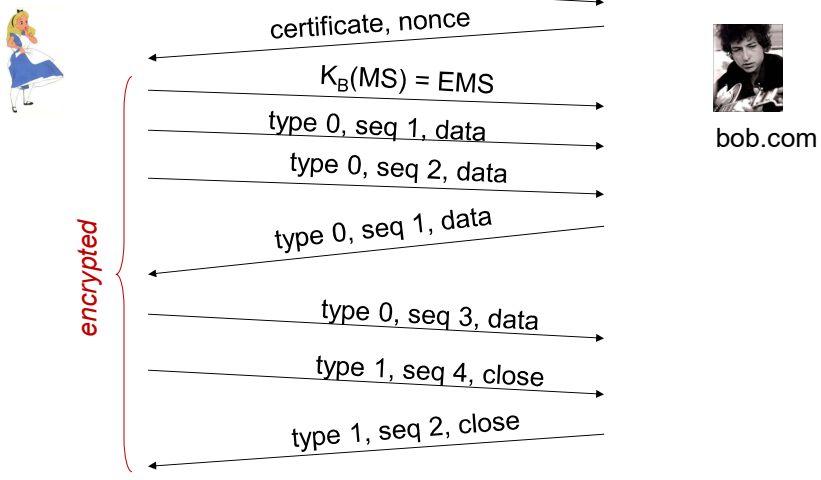
Toy: Control Information

- *problem:* truncation attack:
  - attacker forges TCP connection close segment
  - one sides thinks there is less data than there actually is
- *solution:* record types, with special type for closure
  - type 0 for data; type 1 for closure
- MAC = MAC(M_x, sequence||type||data)



•7

Toy SSL/TLS: Summary



•8

Toy SSL/TLS isn't complete

- how long are fields?
- which encryption algorithms to use?
- we may want parameter negotiation
  - allow client and server to support different encryption algorithms
  - allow client and server to choose together specific algorithm before data transfer

•9

SSL/TLS Cipher Suite

- cipher suite
  - public-key algorithm
  - symmetric encryption algorithm
  - MAC algorithm

•10

SSL/TLS Cipher Suite

- cipher suite
  - public-key algorithm
  - symmetric encryption algorithm
  - MAC algorithm
- SSL/TLS supports multiple cipher suites
- negotiation: client, server agree on a cipher suite
  - client offers choice
  - server picks one

Common SSL/TLS symmetric ciphers
- AES
- 3DES

SSL/TLS Public key encryption
- RSA
- DH
- EC-DH
- DSA

- MAC
  - SHA-256, SHA-128, etc.

•11

Real SSL/TLS: Handshake (1)

Purpose

1. server authentication
2. negotiation: agree on crypto algorithms
3. establish keys
4. client authentication (optional)

•12

Real SSL/TLS: Handshake (2)

1. client sends a list of algorithms it supports, along with a client nonce
2. server chooses algorithms from list; sends back: choice + own certificate + server nonce
3. client verifies certificate, extracts server's public key, generates pre_master_secret, encrypts with server's public key, sends to server
4. client and server independently compute encryption and MAC keys from pre_master_secret and both nonces
5. client sends a MAC of all handshake messages
6. server sends a MAC of all the handshake messages

•13

Real SSL/TLS: Handshake (3)

last 2 steps protect handshake from tampering

- client typically offers range of algorithms, some strong, some weak
- man-in-the middle could delete stronger algorithms from list
- last 2 steps prevent this
  - last two messages are encrypted

•14

Real SSL/TLS: Handshake (4)

- why two random nonces?

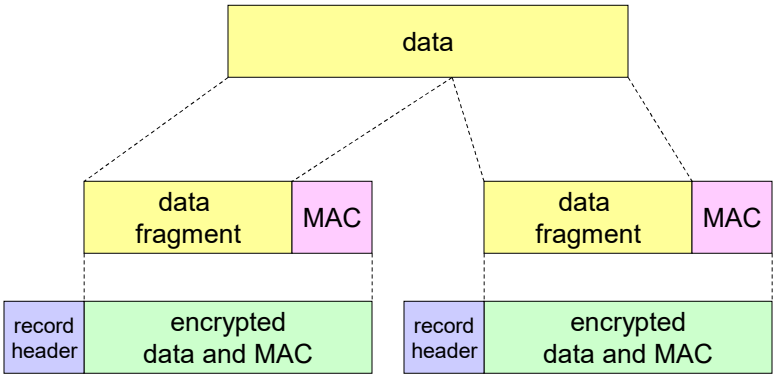
•15

Real SSL/TLS: Handshake (4)

- why two random nonces?
- suppose Eve sniffs all messages between Alice & Bob
- next day, Eve sets up TCP connection with Bob, sends exact same sequence of records
  - Bob (Amazon) thinks Alice made two separate orders for the same thing
  - solution: Bob sends different random nonce for each connection. This causes encryption keys to be different on the two days
  - Eve's messages will fail Bob' s integrity check

•16

SSL/TLS Record Protocol



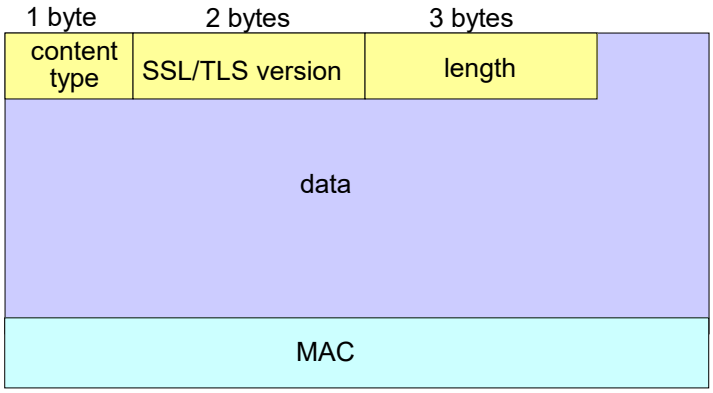
record header: content type; version; length

MAC: includes sequence number, compute with MAC key M_x

fragment: each SSL fragment 2^{14} bytes (~16 Kbytes)

•17

SSL/TLS Record Format



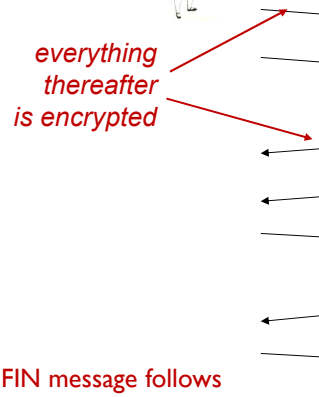
data and MAC encrypted (symmetric algorithm)

•18

Real SSL/TLS Connection







TCP FIN message follows

•19

Detailed explanations

1. **ClientHello**: The client initiates the handshake by sending a "hello" message to the server. The message will include which TLS version the client supports, the cipher suites supported, and a string of random bytes known as the "client random".
2. **ServeHello**: In reply to the client hello message, the server sends a message containing the server's SSL certificate, the server's chosen cipher suite, and the "server random," another random string of bytes that's generated by the server.
3. **ClientKeyExchange**: The client sends one more random string of bytes, the "premaster secret." The premaster secret is encrypted with the public key and can only be decrypted with the private key by the server. (The client gets the public key from the server's SSL certificate.)
4. **ChangeCipherSpec**: notify the receiving party that subsequent records will be protected under the just-negotiated CipherSpec and keys.
5. **Finished**: Sends a "finished" message that is encrypted with the session key.

•20

Key Derivation

- client nonce, server nonce, and pre-master secret input into pseudo random-number generator (PRG).
  - produces master secret
- master secret and new nonces input into another random-number generator: "key block"
- key block used to derive separate:
  - client MAC key
  - server MAC key
  - client encryption key
  - server encryption key
  - client initialization vector (IV)
  - server initialization vector (IV)

•21

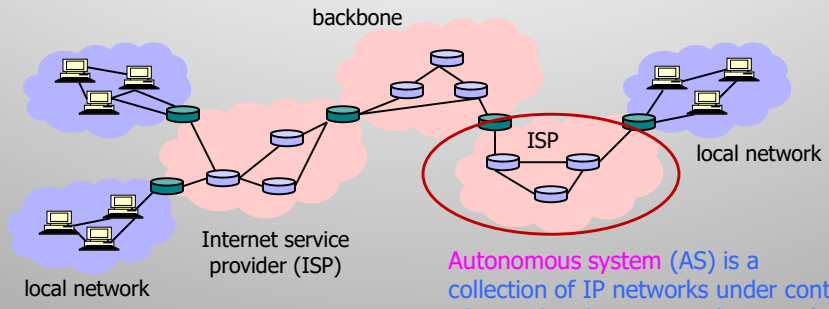
Lecture 16
CS 134

Network/Internet
Threats/Attacks

22

•22

Internet Structure

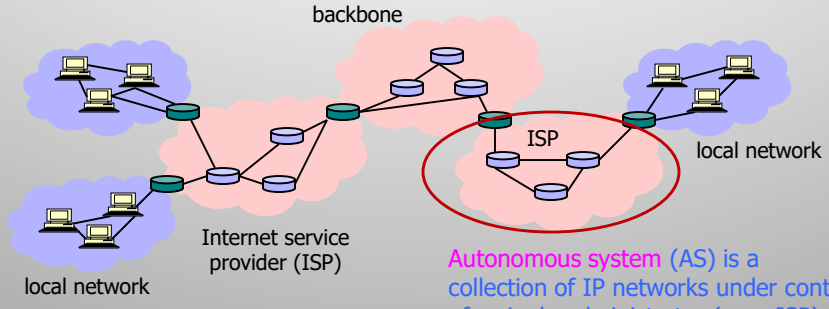


- TCP/IP for packet routing and connections
- Border Gateway Protocol (BGP) for external route discovery
- Domain Name System (DNS) for IP address discovery

23

•23

Internet Structure

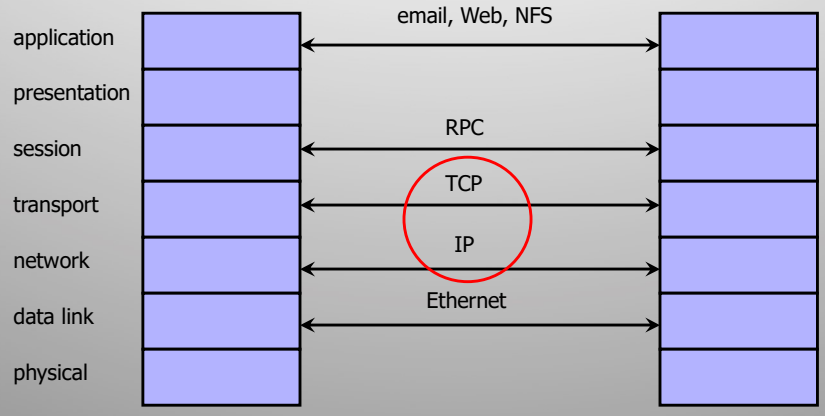


- **TCP/IP for packet routing and connections**
- Border Gateway Protocol (BGP) for external route discovery
- Domain Name System (DNS) for IP address discovery

24

•24

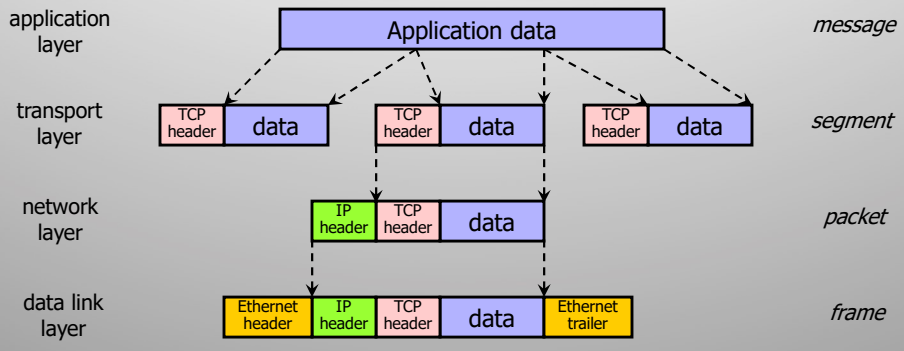
OSI Protocol Stack



25

• 25

Data Formats



26

• 26

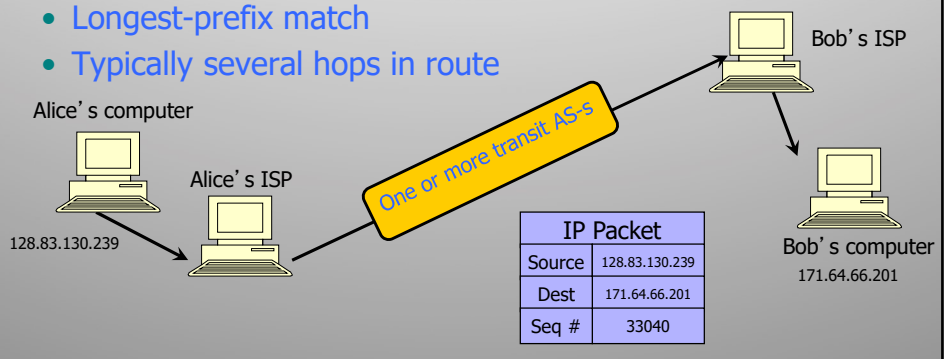
TCP (Transmission Control Protocol)

- Connection-oriented for reliable transmission
- Sender: break data into segments
  - Sequence number is attached to every packet
- Receiver: reassemble segments
  - Acknowledge receipt; lost packets are re-sent
- Connection state maintained by both sides

•27

IP (Internet Protocol)

- Connectionless
  - Unreliable, "best-effort" protocol
- Uses addresses (and prefixes thereof) used for routing
  - Longest-prefix match
  - Typically several hops in route



•28

ICMP (Control Message Protocol)

- Provides feedback about network operation
  - Out-of-band (control) messages carried in IP packets
  - Error reporting, congestion control, reachability, etc.
- Example messages:
  - Destination unreachable
  - Time exceeded
  - Parameter problem
  - Redirect to better gateway
  - Reachability test (echo / echo reply)
  - Timestamp request / reply

29

•29

Security Issues in TCP/IP

- Network packets pass by and thru untrusted hosts
  - Eavesdropping (packet sniffing)
- IP addresses are public
  - E.g., Smurf attacks, Ping-of-Death
- TCP connection requires state
  - SYN flooding
- TCP state easy to guess
  - TCP spoofing and connection hijacking

30

•30

Security Issues in TCP/IP

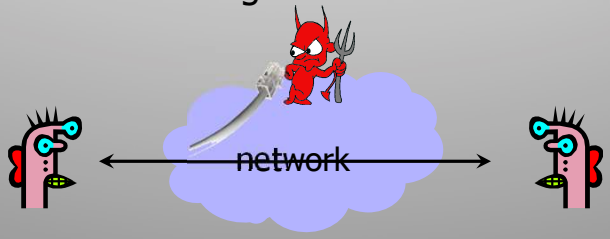
- **Network packets pass by and thru untrusted hosts**
  - Eavesdropping (packet sniffing)
- IP addresses are public
  - E.g., Smurf attacks, Ping-of-Death
- TCP connection requires state
  - SYN flooding
- TCP state easy to guess
  - TCP spoofing and connection hijacking

31

•31

Packet Sniffing

- Many old applications send data unencrypted
  - Plain ftp, telnet send passwords in the clear (as opposed to sftp and ssh)
- Network Interface Card (NIC), e.g., Ethernet device, in "promiscuous mode" can read all data on its broadcast segment



Solution: encryption (e.g., IPsec), improved routing

32

•32

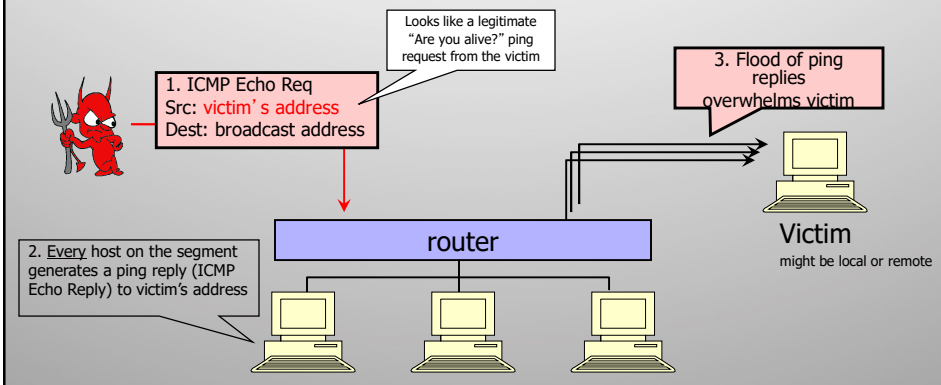
Security Issues in TCP/IP

- Network packets pass by and thru untrusted hosts
  - Eavesdropping (packet sniffing)
- **IP addresses are public**
  - E.g., Smurf attacks, Ping-of-Death
- TCP connection requires state
  - SYN flooding
- TCP state easy to guess
  - TCP spoofing and connection hijacking

33

•33

"Smurf" Attack



Solution: reject external packets to broadcast addresses

34

•34

"Ping of Death"

- When an old Windows machine receives an ICMP packet with payload over 64K, it crashes and/or reboots
  - Programming error in older versions of Windows
  - Packets of this length are illegal, so programmers of old Windows code did not account for them

Solution: patch OS, filter out ICMP packets

35

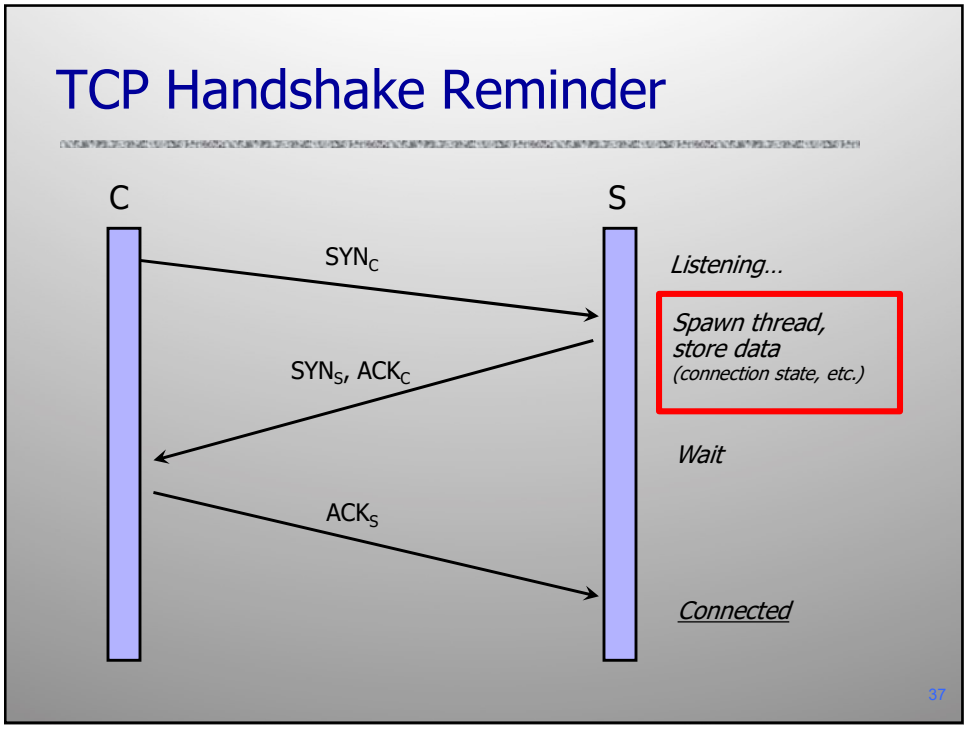
•35

Security Issues in TCP/IP

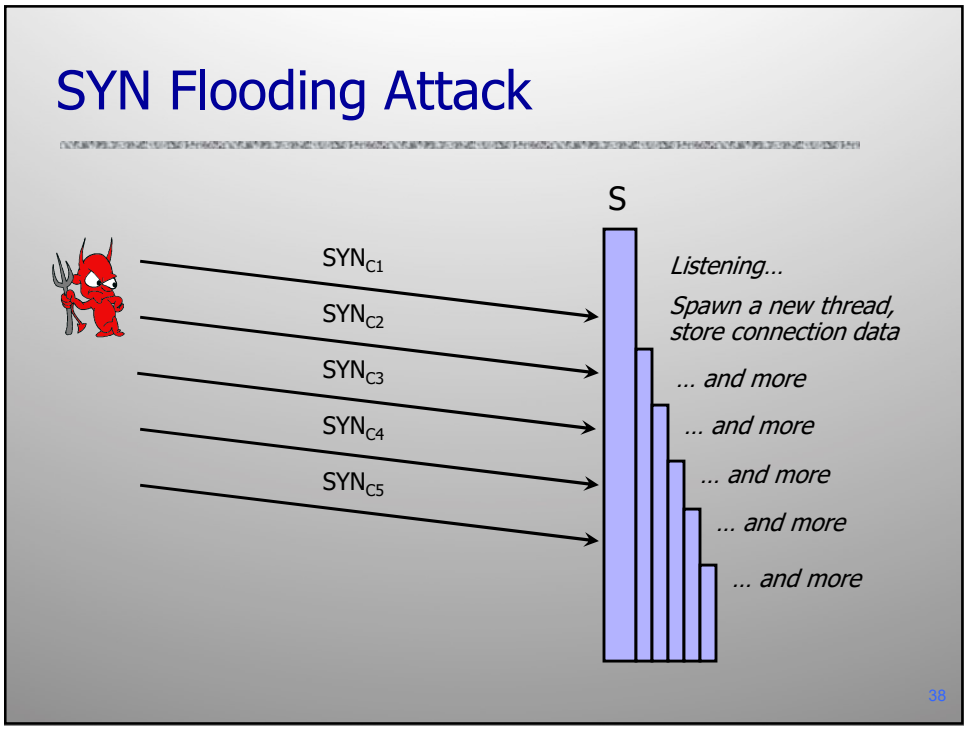
- Network packets pass by and thru untrusted hosts
  - Eavesdropping (packet sniffing)
- IP addresses are public
  - E.g., Smurf attacks, Ping-of-Death
- **TCP connection requires state**
  - SYN flooding
- TCP state easy to guess
  - TCP spoofing and connection hijacking

36

•36


TCP Handshake Reminder
C
S
SYN_C
Listening…
Spawn thread,
store data
(connection state, etc.)
SYN_S, ACK_C
Wait
ACK_S
Connected
37

•37


SYN Flooding Attack
S
SYN_C1
SYN_C2
SYN_C3
SYN_C4
SYN_C5
Listening…
Spawn a new thread,
store connection data
… and more
… and more
… and more
… and more
… and more
38

•38

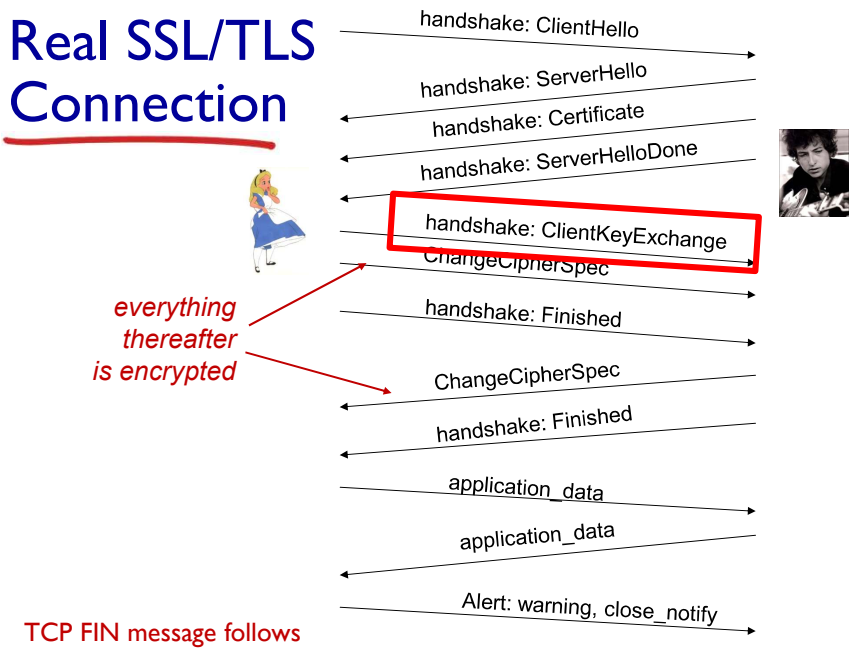
SYN Flooding Explained

- Attacker sends many connection requests (SYNs) with **spoofed source (IP) addresses**
- Victim allocates resources for each request
  - New thread, connection state maintained until timeout
  - Fixed bound on half-open connections
- Once server resources are exhausted, requests from legitimate clients are denied
- This is a classic DoS attack example: ASYMMETRY!!!
  - Common pattern: it costs nothing to TCP client to send a connection request, but server must **do sth really expensive** for each request
  - **Other examples of this behavior?**
    - **TLS/SSL server public key decryption**

39

•39

Real SSL/TLS Connection



•40

Preventing Denial of Service

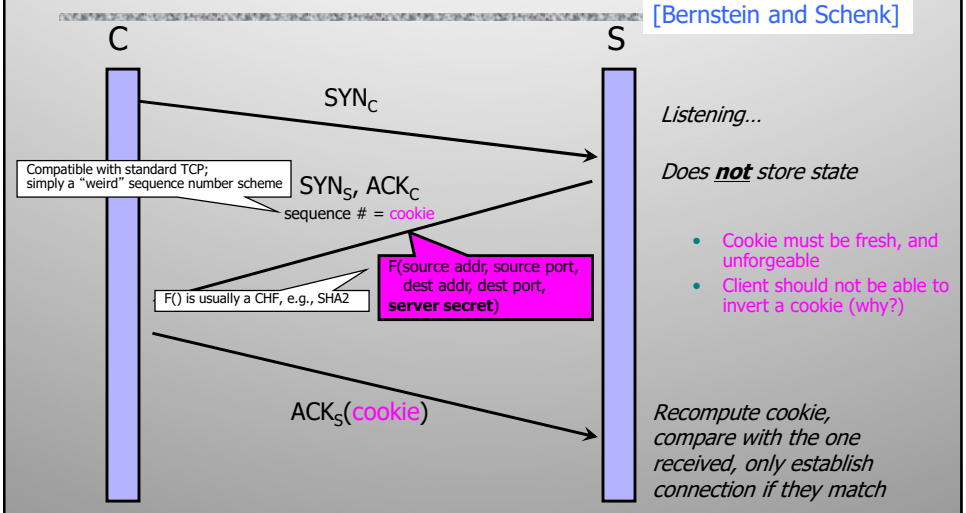
- DoS is caused by asymmetric state allocation
  - If server opens new state for each connection attempt, attacker can initiate many connections from bogus or forged IP addresses
- Cookies allow server to remain stateless until client produces:
  - Server state (IP addresses and ports) stored in a cookie and originally sent to client
- When client responds, cookie is verified

41

•41

SYN Cookies

[Bernstein and Schenk]



More info: http://cr.yp.to/syncookies.html

Note: each TCP packet carries a 32-bit seq numbers

42

•42

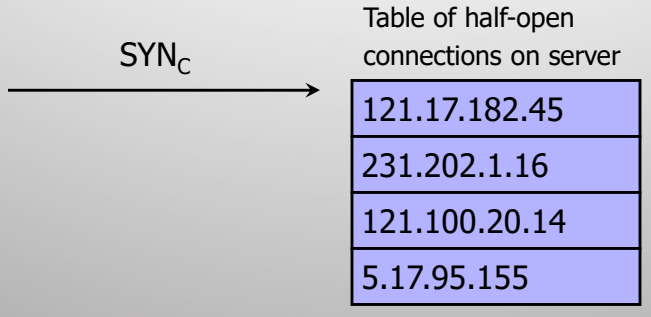
Anti-Spoofing Cookies: Basic Pattern

- Client sends request (message #1) to server
- Typical protocol:
  - Server sets up connection, responds with message #2
  - Client may complete session or not (potential DoS)
- Cookie version:
  - Server responds with hashed connection data instead of message #2
    - Does not spawn any threads, does not allocate resources!
  - Client confirms by returning cookie (with other fields)
    - If source IP address is bogus, attacker can't confirm
    - WHY?

43

•43

Passive Defense: Random Deletion



- If SYN queue is full, delete random entry
  - Legitimate connections have a chance to complete
  - Fake addresses will be eventually deleted. WHY?
- Easy to implement

44

•44

Security Issues in TCP/IP

- Network packets pass by and thru untrusted hosts
  - Eavesdropping (packet sniffing)
- IP addresses are public
  - E.g., Smurf attacks, Ping-of-Death
- TCP connection requires state
  - SYN flooding
- **TCP state easy to guess**
  - TCP spoofing and connection hijacking

45

•45

TCP Connection Spoofing

- Each TCP connection has associated "state" info:
  - Sequence number, port number, src IP, dst IP, etc.
- TCP state is easy to guess
  - Port numbers are standard, seq numbers are *predictable*
- Can inject packets into existing connections
  - If attacker knows initial sequence number and amount of traffic, can guess current number
  - Guessing a 32-bit seq number is not practical, BUT…
  - Most systems accept a *large window* of sequence numbers (to handle massive packet losses, e.g., in shaky wireless networks)
  - Send a flood of packets with likely sequence numbers

46

•46

DoS by Connection Reset

- If attacker can guess the current sequence number for an existing connection, can send a **reset** packet to close it (RST flag=1 in TCP header)
- Especially effective against long-lived connections
  - For example, background system services such as push notification

47

•47

Countermeasures

- Above transport layer: Kerberos
  - Provides authentication, protects against application-layer spoofing
  - Does not protect against connection hijacking
- Above network layer: SSL/TLS and SSH
  - Protects against connection hijacking and injected data
  - Does not protect against DoS by spoofed packets
- Network (IP) layer: IPsec
  - Protects against hijacking, injection, DoS using connection resets, IP address spoofing
  - But muddled/poor key management…

48

•48