WHY DOES THE INTERNET WORK?

1. PROTOCOLS  Agreements on how to communicate

   Publicly standardized, esp. via Requests for Comments (RFCs)

   RFC 826: ARP   RFC 103{4,5}: DNS   RFC 793: TCP

   Code to the protocol and your product will work with other products
### Why Does the Internet Work?

The payload is the “data” that IP is delivering:

May contain another protocol’s header & payload, and so on

<table>
<thead>
<tr>
<th>4-bit Version</th>
<th>4-bit Header len</th>
<th>8-bit Type of service (TOS)</th>
<th>16-bit Total length (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>16-bit Identification</td>
<td>3-bit Flags</td>
</tr>
<tr>
<td>8-bit Time-to-live (TTL)</td>
<td>8-bit Protocol</td>
<td></td>
<td>13-bit Fragment offset</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>16-bit Header checksum</td>
</tr>
<tr>
<td></td>
<td></td>
<td>32-bit Source IP address</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>32-bit Destination IP address</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Payload</td>
<td></td>
</tr>
</tbody>
</table>
WHY DOES THE INTERNET WORK?

2. THE NETWORK IS DUMB

**End-hosts** are the periphery (users, devices)

**Routers** and **switches** are interior nodes that

**Route** (figure out where to forward)

**Forward** (actually send)

- Principle: the routers have no knowledge of ongoing connections through them
  - They do "destination-based" routing and forwarding
    - Given the destination in the packet, send it to the "next hop" that is best suited to help ultimately get the packet there
WHY DOES THE INTERNET WORK?

2. THE NETWORK IS DUMB

**End-hosts** are the periphery (users, devices)

**Routers** and **switches** are interior nodes that

- **Route** (figure out where to forward)
- **Forward** (actually send)

- Principle: the routers have no knowledge of ongoing connections through them
- They do “destination-based” routing and forwarding
  - Given the destination in the packet, send it to the “next hop” that is best suited to help ultimately get the packet there

**Mental model: The postal system**
3. LAYERS

• The design of the Internet is strongly partitioned into layers
  • Each layer relies on the services provided by the layer immediately below it...
  • ... and provides service to the layer immediately above it
LAYERS OF THE INTERNET

PHYSICAL

Send / receive bit

Broadcasts on shared link
LAYERS OF THE INTERNET

PHYSICAL
- Send / receive bit
  - Broadcasts on shared link

LINK
- Local send/recv
  - Adds framing & destination;
    Still assumes shared link
# Layers of the Internet

<table>
<thead>
<tr>
<th>Layer</th>
<th>Function</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Physical</strong></td>
<td>Send / receive bit</td>
<td>Broadcasts on shared link</td>
</tr>
<tr>
<td><strong>Link</strong></td>
<td>Local send/recv</td>
<td>Adds framing &amp; destination; Still assumes shared link</td>
</tr>
<tr>
<td><strong>Network (IP)</strong></td>
<td>Global send/recv</td>
<td>Adds global addresses; Requires routing</td>
</tr>
</tbody>
</table>
## Layers of the Internet

<table>
<thead>
<tr>
<th>Layer</th>
<th>Function</th>
<th>Additional Features</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Physical</strong></td>
<td>Send / receive bit</td>
<td>Broadcasts on shared link</td>
</tr>
<tr>
<td><strong>Link</strong></td>
<td>Local send/recv</td>
<td>Adds framing &amp; destination; Still assumes shared link</td>
</tr>
<tr>
<td><strong>Network (IP)</strong></td>
<td>Global send/recv</td>
<td>Adds global addresses; Requires routing</td>
</tr>
<tr>
<td><strong>Transport (TCP,UDP)</strong></td>
<td>Process send/recv</td>
<td>E2E communication between processes; Adds ports/reliability</td>
</tr>
</tbody>
</table>
# Layers of the Internet

<table>
<thead>
<tr>
<th>Layer</th>
<th>Function</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application</strong></td>
<td>Arbitrary</td>
<td>Application-specific semantics</td>
</tr>
<tr>
<td><strong>Transport (TCP, UDP)</strong></td>
<td>Process send/recv</td>
<td>E2E communication between processes; Adds ports/reliability</td>
</tr>
<tr>
<td><strong>Network (IP)</strong></td>
<td>Global send/recv</td>
<td>Adds global addresses; Requires routing</td>
</tr>
<tr>
<td><strong>Link</strong></td>
<td>Local send/recv</td>
<td>Adds framing &amp; destination; Still assumes shared link</td>
</tr>
<tr>
<td><strong>Physical</strong></td>
<td>Send / receive bit</td>
<td>Broadcasts on shared link</td>
</tr>
</tbody>
</table>
Hop-by-hop vs. end-to-end layers

Host C communicates with host A
Hop-by-hop vs. end-to-end layers

Different physical & link layers

End-host A

Ethernet

Router 1

Router 3

WiFi

End-host C

End-host D

Router 2

Router 4

Router 5

Router 6

End-host B

End-host E
Hop-by-hop vs. end-to-end layers

Same network, transport, and application layers (3/4/7)
Routers *ignore* transport & application

E.g., HTTP over TCP over IP
IP packet “header”

<table>
<thead>
<tr>
<th>4-bit Version</th>
<th>4-bit Header len</th>
<th>8-bit Type of service (TOS)</th>
<th>16-bit Total length (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16-bit Identification</td>
<td>3-bit Flags</td>
<td>13-bit Fragment offset</td>
<td></td>
</tr>
<tr>
<td>8-bit Time-to-live (TTL)</td>
<td>8-bit Protocol</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16-bit Header checksum</td>
<td>32-bit</td>
<td>Source IP address</td>
<td></td>
</tr>
<tr>
<td>32-bit Destination IP address</td>
<td>16-bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Payload</td>
<td>32-bit</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

20-byte header
IP Packet Header Fields (1)

- **Version number** (4 bits)
  - Indicates the version of the IP protocol
  - Necessary for knowing what fields follow
  - “4” (for IPv4) or “6” (for IPv6)

- **Header length** (4 bits)
  - How many 32-bit words (rows) in the header
  - Typically 5
  - Can provide IP options, too

- **Type-of-service** (8 bits)
  - Allow packets to be treated differently based on different needs
  - Low delay for audio, high bandwidth for bulk transfer, etc.
IP Packet Header Fields (2)

- Two IP addresses
  - Source (32 bits)
  - Destination (32 bits)

- Destination address
  - *Unique* identifier/locator for the receiving host
  - Allows each node (end-host and router) to make forwarding decisions

- Source address
  - Unique identifier/locator for the sending host
  - Recipient can decide whether to accept the packet
  - Allows destination to *reply* to the source
IP: “Best effort” packet delivery

- Routers inspect destination address, determine “next hop” in the forwarding table

- Best effort = “I’ll give it a try”
  - Packets may be lost
  - Packets may be corrupted
  - Packets may be delivered out of order

Fixing these is the job of the transport layer!
## Attacks on IP

<table>
<thead>
<tr>
<th>4-bit Version</th>
<th>4-bit Header Len</th>
<th>8-bit Type of service (TOS)</th>
<th>16-bit Total length (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>16-bit Identification</td>
<td>3-bit Flags</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3-bit Flags</td>
<td>13-bit Fragment offset</td>
</tr>
<tr>
<td>8-bit Time-to-live (TTL)</td>
<td>8-bit Protocol</td>
<td></td>
<td>16-bit Header checksum</td>
</tr>
<tr>
<td>32-bit Source IP address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32-bit Destination IP address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Payload</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Attacks on IP

Source-spoof

There is nothing in IP that enforces that your source IP address is really “yours”
## Attacks on IP

There is nothing in IP that enforces that your source IP address is really “yours”

### Eavesdrop / Tamper

IP provides no protection of the *payload* or *header*
Source-spoofing

• Why source-spoof?
  • Consider spam: send many emails from one computer
  • Easy defense: block many emails from a given (source) IP address
  • Easy countermeasure: spoof the source IP address
  • Counter-countermeasure?

• How do you know if a packet you receive has a spoofed source?
Salient network features

• Recall: The Internet operates via destination-based routing

• attacker: pkt (spoofed source) -> destination
destination: pkt -> spoofed source

• In other words, the response goes to the spoofed source, not the attacker
Defending against source-spoofing

• How do you know if a packet you receive has a spoofed source?
  • Send a challenge packet to the (possibly spoofed) source (e.g., a difficult to guess, random nonce)
  • If the recipient can answer the challenge, then likely that the source was not spoofed

• So do you have to do this with every packet??
  • Every packet should have something that’s difficult to guess
  • Recall the query ID in the DNS queries! Easy to predict => Kaminsky attack
Source spoofing

• Why source-spoof?
  • Consider DoS attacks: generate as much traffic as possible to congest the victim’s network
  • Easy defense: block all traffic from a given source near the edge of your network
  • Easy countermeasure: spoof the source address

• Challenges won’t help here; the damage has been done by the time the packets reach the core of our network

• Ideally, detect such spoofing near the source
Egress filtering

- The point (router/switch) at which traffic *enters* your network is the *ingress point*.

- The point (router/switch) at which traffic *leaves* your network is the *egress point*.

- You don’t know who owns all IP addresses in the world, but you *do* know who in *your own network* gets what IP addresses.
  - If you see a packet with a source IP address that doesn’t belong to your network trying to cross your egress point, then *drop it*.

Egress filtering is not widely deployed.
Eavesdropping / Tampering

- No security built into IP
- $\Rightarrow$ Deploy secure IP over IP
Virtual Private Networks (VPNs)

Goal: Allow the client to connect to the trusted network from within an untrusted network

Example: Connect to your company’s network (for payroll, file access, etc.) while visiting a competitor’s office
Virtual Private Networks (VPNs)

Idea: A VPN “client” and “server” together create end-to-end encryption/authentication

Predominate way of doing this: IPSec
IPSec

• Operates in a few different modes
  • Transport mode: Simply encrypt the payload but not the headers
  • Tunnel mode: Encrypt the payload *and* the headers

• But how do you encrypt the headers? How does routing work?
  • Encrypt the entire IP packet and make that the payload of another IP packet
Tunnel mode

The VPN server decrypts and then sends the payload (itself a full IP packet) as if it had just received it from the network.

From the client/servers’ perspective:
Looks like the client is physically connected to the network!
Layer 4: Transport layer

- End-to-end communication between **processes**
- Different types of services provided:
  - UDP: unreliable *datagrams*
  - TCP: *reliable* byte stream
- “Reliable” = keeps track of what data were received properly and retransmits as necessary
TCP: reliability

• Given best-effort deliver, the goal is to ensure reliabilit
  • All packets are delivered to applications
  • … in order
  • … unmodified (with reasonably high probability)

• Must robustly detect and retransmit lost data
TCP’s bytestream service

- Process A on host 1:
  - Send byte 0, byte 1, byte 2, byte 3, ...

- Process B on host 2:
  - Receive byte 0, byte 1, byte 2, byte 3, ...

- The applications do **not** see:
  - packet boundaries (looks like a stream of bytes)
  - lost or corrupted packets (they’re all correct)
  - retransmissions (they all only appear once)
TCP bytestream service

Abstraction: Each byte reliably delivered in order

Process A on host H1

Process B on host H2
TCP bytestream service

Reality: *Packets* sometimes retransmitted, sometimes arrive out of order

Packet 1: Needs to be retransmitted
Packet 2: Needs to be buffered
Packet 3: Needs to be buffered
TCP bytestream service

Reality: *Packets* sometimes retransmitted, sometimes arrive out of order

TCP’s first job: achieve the abstraction while hiding the reality from the application
How does TCP achieve reliability?
How does TCP achieve reliability?

Waterfall diagram

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Expecting byte 1000</td>
</tr>
</tbody>
</table>
How does TCP achieve reliability?

Waterfall diagram

Time

Bytes 1000-1500

Expecting byte 1000
How does TCP achieve reliability?

Waterfall diagram

Time

Bytes 1000-1500

A

B

Expecting byte 1000

Expecting byte 1501
How does TCP achieve reliability?

Waterfall diagram

Time

Bytes 1000-1500

ACK 1501

Expecting byte 1000

Expecting byte 1501
How does TCP achieve reliability?

Reliability through acknowledgments to determine whether something was received.
How does TCP achieve reliability?
How does TCP achieve reliability?

Waterfall diagram

A

Expecting byte 1000

B

Time
How does TCP achieve reliability?

Waterfall diagram

Time

Bytes 1000-1500

Expecting byte 1000
How does TCP achieve reliability?

Waterfall diagram

Time

A

Bytes 1000-1500

Bytes 1501-2000

B

Expecting byte 1000
How does TCP achieve reliability?

Waterfall diagram

A

Bytes 1000-1500

Bytes 1501-2000

Bytes 2001-3000

B

Expecting byte 1000
How does TCP achieve reliability?

A

Bytes 1000-1500

Bytes 1501-2000

Bytes 2001-3000

Expecting byte 1000

Still expecting byte 1000

B

Waterfall diagram

Time
How does TCP achieve reliability?

A

Bytes 1000-1500

Bytes 1501-2000

Bytes 2001-3000

ACK 1000

B

Expecting byte 1000

Still expecting byte 1000

Waterfall diagram

Time
How does TCP achieve reliability?

A

Bytes 1000-1500

Bytes 1501-2000

Bytes 2001-3000

ACK 1000

B

Expecting byte 1000

Still expecting byte 1000

Still expecting byte 1000

Waterfall diagram

Time
How does TCP achieve reliability?

Waterfall diagram

Time

Bytes 1000-1500

Bytes 1501-2000

Bytes 2001-3000

ACK 1000

Still expecting byte 1000

Still expecting byte 1000

Expecting byte 1000
How does TCP achieve reliability?

A

- Bytes 1000-1500
- Bytes 1501-2000
- Bytes 2001-3000

B

- Expecting byte 1000
- Still expecting byte 1000

Waterfall diagram

Time
How does TCP achieve reliability?

A

<table>
<thead>
<tr>
<th>Bytes 1000-1500</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bytes 1501-2000</td>
</tr>
<tr>
<td>Bytes 2001-3000</td>
</tr>
</tbody>
</table>

Expected byte 1000

Still expecting byte 1000

Expecting packet 3001

Waterfall diagram
How does TCP achieve reliability?

A

Waterfall diagram

Bytes 1000-1500

Bytes 1501-2000

Bytes 2001-3000

ACK 1000

ACK 1000

ACK 1000

ACK 3001

Time

B

Expecting byte 1000

Still expecting byte 1000

Still expecting byte 1000

Expecting packet 3001
How does TCP achieve reliability?

A

Bytes 1000-1500

Still expecting byte 1000

Still expecting byte 1000

Expecting byte 1000

Bytes 1501-2000

Expecting packet 3001

Bytes 2001-3000

ACK 1000

ACK 1000

ACK 1000

Bytes 1000-1500

ACK 3001

Buffer these until

Waterfall diagram
TCP congestion control

TCP’s second job: don’t break the network!

- Try to use as much of the network as is safe (does not adversely affect others’ performance) and efficient (makes use of network capacity)

- Dynamically adapt how quickly you send based on the network path’s capacity

- When an ACK doesn’t come back, the network may be beyond capacity: slow down.
### TCP header

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>16-bit Source port</td>
<td>Source port number</td>
</tr>
<tr>
<td>16-bit Destination port</td>
<td>Destination port number</td>
</tr>
<tr>
<td>32-bit Sequence number</td>
<td>Sequence number</td>
</tr>
<tr>
<td>32-bit Acknowledgment</td>
<td>Acknowledgment number</td>
</tr>
<tr>
<td>4-bit Header Length</td>
<td>Header length</td>
</tr>
<tr>
<td>Reserved</td>
<td>Reserved bit(s)</td>
</tr>
<tr>
<td>6-bit Flags</td>
<td>Flags</td>
</tr>
<tr>
<td>16-bit Advertised window</td>
<td>Advertised window</td>
</tr>
<tr>
<td>16-bit Checksum</td>
<td>Checksum</td>
</tr>
<tr>
<td>16-bit Urgent pointer</td>
<td>Urgent pointer</td>
</tr>
<tr>
<td>Options (variable)</td>
<td>Optional data options</td>
</tr>
<tr>
<td>Padding</td>
<td>Padding</td>
</tr>
<tr>
<td>Data</td>
<td>Data</td>
</tr>
</tbody>
</table>
# TCP header

## IP Header

<table>
<thead>
<tr>
<th>16-bit Source port</th>
<th>16-bit Destination port</th>
</tr>
</thead>
<tbody>
<tr>
<td>32-bit Sequence number</td>
<td></td>
</tr>
<tr>
<td>32-bit Acknowledgment</td>
<td></td>
</tr>
<tr>
<td>4-bit Header Length</td>
<td>Reserved</td>
</tr>
<tr>
<td>16-bit Checksum</td>
<td>16-bit Advertised window</td>
</tr>
<tr>
<td></td>
<td>16-bit Urgent pointer</td>
</tr>
<tr>
<td>Options (variable)</td>
<td>Padding</td>
</tr>
<tr>
<td>Data</td>
<td></td>
</tr>
</tbody>
</table>
TCP ports

- Ports are associated with **OS processes**
- Sandwiched between IP header and the application data
- \{src IP/port, dst IP/port\} : this 4-tuple uniquely identifies a TCP connection
- Some port numbers are well-known
  - 80 = HTTP
  - 53 = DNS
## TCP header

<table>
<thead>
<tr>
<th>Field</th>
<th>Bits/Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source port</td>
<td>16-bit</td>
<td>Identification of the sender</td>
</tr>
<tr>
<td>Destination port</td>
<td>16-bit</td>
<td>Identification of the recipient</td>
</tr>
<tr>
<td>Sequence number</td>
<td>32-bit</td>
<td>Number identifying the start of the sequence of data exchanged between the source and destination hosts</td>
</tr>
<tr>
<td>Acknowledgment</td>
<td>32-bit</td>
<td>Number identifying the data that is to be acknowledged by the recipient</td>
</tr>
<tr>
<td>Header Length</td>
<td>4-bit</td>
<td>Length of the TCP header</td>
</tr>
<tr>
<td>Flags</td>
<td>6-bit</td>
<td>Flags for control of the connection</td>
</tr>
<tr>
<td>Advertised window</td>
<td>16-bit</td>
<td>Advertised window size</td>
</tr>
<tr>
<td>Checksum</td>
<td>16-bit</td>
<td>Used to detect errors in the transmission of the TCP header and data</td>
</tr>
<tr>
<td>Urgent pointer</td>
<td>16-bit</td>
<td>Pointer to urgent data within the data segment</td>
</tr>
<tr>
<td>Options (variable)</td>
<td>variable</td>
<td>Options to provide additional functionality</td>
</tr>
<tr>
<td>Padding</td>
<td></td>
<td>Padding bytes to ensure header length is a multiple of 32</td>
</tr>
<tr>
<td>Data</td>
<td></td>
<td>Data exchanged between hosts</td>
</tr>
</tbody>
</table>
TCP seqno

• Each byte in the byte stream has a unique “sequence number”
  • Unique for both directions

• “Sequence number” in the header = sequence number of the first byte in the packet’s data

• Next sequence number = previous seqno + previous packet’s data size

• “Acknowledgment” in the header = the next seqno you expect from the other end-host
### TCP header

<table>
<thead>
<tr>
<th>Field</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source port</td>
<td>16-bit</td>
</tr>
<tr>
<td>Destination port</td>
<td>16-bit</td>
</tr>
<tr>
<td>Sequence number</td>
<td>32-bit</td>
</tr>
<tr>
<td>Acknowledgment</td>
<td>32-bit</td>
</tr>
<tr>
<td>Header Length</td>
<td>4-bit</td>
</tr>
<tr>
<td>Reserved</td>
<td>4-bit</td>
</tr>
<tr>
<td>Flags</td>
<td>6-bit</td>
</tr>
<tr>
<td>Advertised window</td>
<td>16-bit</td>
</tr>
<tr>
<td>Urgent pointer</td>
<td>16-bit</td>
</tr>
<tr>
<td>Checksum</td>
<td>16-bit</td>
</tr>
<tr>
<td>Options (variable)</td>
<td>Variable</td>
</tr>
<tr>
<td>Padding</td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td></td>
</tr>
</tbody>
</table>
TCP flags

- SYN
  - Used for setting up a connection

- ACK
  - Acknowledgments, for data and “control” packets

- FIN

- RST
Setting up a connection

Three-way handshake

Waterfall diagram

Time
Setting up a connection

Three-way handshake

Waterfall diagram

Time
Setting up a connection

Three-way handshake

A

SYN

B

Let's SYNchronize sequence numbers

Waterfall diagram

Time
Setting up a connection

Three-way handshake

A

SYN

B

SYN + ACK

Let's SYNchronize sequence numbers

Waterfall diagram

Time
Setting up a connection

Three-way handshake

A

SYN

SYN + ACK

B

Let's SYNchronize sequence numbers

Got yours; here's mine

Waterfall diagram

Time
Setting up a connection

Three-way handshake

A

SYN

SYN + ACK

B

ACK

Let's SYNchronize sequence numbers

Got yours; here's mine

Waterfall diagram

Time
Setting up a connection

Three-way handshake

A

SYN

SYN + ACK

ACK

B

Let's SYNchronize sequence numbers

Got yours; here's mine

Got yours, too

Waterfall diagram

Time
Setting up a connection

Three-way handshake

Let’s SYNchronize sequence numbers

Got yours; here’s mine

Got yours, too

Time

Waterfall diagram

A

SYN

SYN + ACK

ACK

Data

B
Setting up a connection

Three-way handshake

A

SYN

SYN + ACK

ACK

Data

Data

B

Let's SYNchronize sequence numbers

Got yours; here's mine

Got yours, too

Waterfall diagram

Time
Setting up a connection

Three-way handshake

Waterfall diagram

Time

A

SYN

SYN + ACK

ACK

Data

Data

Data

B

Let's SYNchronize sequence numbers

Got yours; here's mine

Got yours, too
Setting up a connection

Three-way handshake

A

SYN seqno=x

SYN seqno=y

+ACK x+1

ACK y+1

Data

Data

Data

B

Let's SYNchronize sequence numbers

Got yours; here's mine

Got yours, too

Waterfall diagram
TCP flags

- SYN
- ACK
- FIN: Let’s shut this down (two-way)
  - FIN
  - FIN+ACK
- RST: I’m shutting you down
  - Says “delete all your local state, because I don’t know what you’re talking about
Attacks

- SYN flooding
- Injection attacks
- Opt-ack attack
SYN flooding
SYN flooding

Recall the three-way handshake:

A

B

Waterfall diagram

Time
SYN flooding

Recall the three-way handshake:

- A
- B

Waterfall diagram

Time
SYN flooding

Recall the three-way handshake:

At this point, B allocates state for this new connection (incl. IP, port, maximum segment size)
SYN flooding

Recall the three-way handshake:

At this point, B allocates state for this new connection (incl. IP, port, maximum segment size)
SYN flooding

Recall the three-way handshake:

At this point, B allocates state for this new connection (incl. IP, port, maximum segment size)
SYN flooding

Recall the three-way handshake:

A

SYN

B

SYN + ACK

ACK

IP/port, MSS,...

At this point, B allocates state for this new connection (incl. IP, port, maximum segment size)
SYN flooding

Recall the three-way handshake:

At this point, B allocates state for this new connection (incl. IP, port, maximum segment size)
SYN flooding

Recall the three-way handshake:

A

Waterfall diagram

Time

B

- SYN
- SYN + ACK
- ACK
- SYN + ACK

At this point, B allocates state for this new connection (incl. IP, port, maximum segment size)

B will hold onto this local state and retransmit SYN+ACK’s until it hears back or times out (up to 63 sec).
SYN flooding

The attack
SYN flooding

The attack
SYN flooding

The attack

IP/port, MSS,...
SYN flooding

The attack

A

SYN

SYN

B

IP/port, MSS,...

C
SYN flooding
The attack

A

B

C

SYN
SYN

IP/port, MSS,...

IP/port, MSS,...
SYN flooding
The attack

A

B

C

SYN

SYN

SYN

IP/port, MSS,…

IP/port, MSS,…
SYN flooding

The attack

A

B

C

SYN

IP/port, MSS,…

IP/port, MSS,…

IP/port, MSS,…
SYN flooding

The attack

A

SYN

SYN

SYN

SYN

SYN

SYN

SYN

B

IP/port, MSS,…

IP/port, MSS,…

IP/port, MSS,…

C
SYN flooding

The attack
SYN flooding

The attack

A

SYN

SYN

SYN

SYN

SYN

SYN

SYN

IP/port, MSS,…

IP/port, MSS,…

IP/port, MSS,…

IP/port, MSS,…

Exhaust memory at the victim B.
SYN flooding

The attack

Exhaust memory at the victim B.
SYN flooding

The attack

A

SYN
SYN
SYN
SYN
SYN
SYN
SYN
SYN

B

IP/port, MSS,…
IP/port, MSS,…
IP/port, MSS,…

Exhaust memory at the victim B.

C

New connections will fail (insufficient memory)

SYN
SYN flooding details

• Easy to detect many incomplete handshakes from a single IP address

• *Spoof* the source IP address
  • It’s just a field in a header: set it to whatever you like

• Problem: the host who really owns that spoofed IP address may respond to the SYN+ACK with a RST, deleting the local state at the victim

• Ideally, spoof an IP address of a host you know won’t respond
SYN cookies

The defense
SYN cookies

The defense
SYN cookies

The defense

A

B

SYN

IP/port, MSS, ...
SYN cookies

The defense

Rather than store this data, send it to the host who is initiating the connection and have him return it to you
SYN cookies

The defense

Rather than store this data, send it to the host who is initiating the connection and have him return it to you.

Store the necessary state in your seqno

seqno = f(data)
The defense

Rather than store this data, send it to the host who is initiating the connection and have him return it to you.

Store the necessary state in your seqno.
SYN cookies

The defense

Rather than store this data, send it to the host who is initiating the connection and have him return it to you.

A

SYN

B

SYN + ACK
seqno = f(data)

Store the necessary state in your seqno

ACK f(data)+1
SYN cookies

The defense

Rather than store this data, send it to the host who is initiating the connection and have him return it to you.

Store the necessary state in your seqno

Check that \( f(\text{data}) \) is valid for this connection. Only at that point do you allocate state.
SYN cookies

The defense

Rather than store this data, send it to the host who is initiating the connection and have him return it to you.

Store the necessary state in your seqno

Check that $f(data)$ is valid for this connection. Only at that point do you allocate state.
SYN cookie format

The secure hash makes it difficult for the attacker to guess what f() will be, and therefore the attacker cannot guess a correct ACK if he spoofs.
Injection attacks

• Suppose you are on the path between src and dst; what can you do?
  • Trivial to inject packets with the correct sequence number

• What if you are not on the path?
  • Need to guess the sequence number
  • Is this difficult to do?
Initial sequence numbers

• Initial sequence numbers used to be deterministic

• What havoc can we wreak?
  • Send RSTs
  • Inject data packets into an existing connection (TCP veto attacks)
  • *Initiate and use an entire connection without ever hearing the other end*
Mitnick attack

- X-terminal server
- Server that X-term trusts: Any connection initiated from this IP address is allowed access to the X-terminal server
- Attacker
Mitnick attack

X-terminal server

Server that X-term trusts

Any connection initiated from this IP address is allowed access to the X-terminal server

1. SYN flood the trusted server
Mitnick attack

1. SYN flood the trusted server

Any connection initiated from this IP address is allowed access to the X-terminal server
Mitnick attack

1. SYN flood the trusted server
2. Spoof trusted server’s IP addr in SYN to X-terminal

X-terminal server

Server that X-term trusts

Any connection initiated from this IP address is allowed access to the X-terminal server
Mitnick attack

1. SYN flood the trusted server
2. Spoof trusted server’s IP addr in SYN to X-terminal

Any connection initiated from this IP address is allowed access to the X-terminal server.
Mitnick attack

1. SYN flood the trusted server
2. Spoof trusted server’s IP addr in SYN to X-terminal

X-terminal server → SYN src: SYN+ACK

Server that X-term trusts

Any connection initiated from this IP address is allowed access to the X-terminal server

seqno
Mitnick attack

X-terminal server

Server that X-term trusts

Any connection initiated from this IP address is allowed access to the X-terminal server

1. SYN flood the trusted server
2. Spoof trusted server’s IP addr in SYN to X-terminal
3. Trusted server too busy to RST
Mitnick attack

1. SYN flood the trusted server
2. Spoof trusted server’s IP addr in SYN to X-terminal
3. Trusted server too busy to RST
4. ACK with the guessed seqno
Mitnick attack

1. SYN flood the trusted server
2. Spoof trusted server’s IP addr in SYN to X-terminal
3. Trusted server too busy to RST
4. ACK with the guessed seqno
Mitnick attack

Any connection initiated from this IP address is allowed access to the X-terminal server

1. SYN flood the trusted server
2. Spoof trusted server’s IP addr in SYN to X-terminal
3. Trusted server too busy to RST
4. ACK with the guessed seqno

"echo ++ >> ./rhosts"
Mitnick attack

X-terminal server

Server that X-term trusts

Any connection initiated from this IP address is allowed access to the X-terminal server

1. SYN flood the trusted server
2. Spoof trusted server’s IP addr in SYN to X-terminal
3. Trusted server too busy to RST
4. ACK with the guessed seqno
5. Grant access to all sources

“echo ++ >> ./rhosts”
Mitnick attack

1. SYN flood the trusted server
2. Spoof trusted server’s IP addr in SYN to X-terminal
3. Trusted server too busy to RST
4. ACK with the guessed seqno
5. Grant access to all sources

```
"echo ++ >> ./rhosts"
```
Mitnick attack

1. SYN flood the trusted server
2. Spoof trusted server’s IP addr in SYN to X-terminal
3. Trusted server too busy to RST
4. ACK with the guessed seqno
5. Grant access to all sources
6. RSTs to trusted server (cleanup)
Mitnick attack

1. SYN flood the trusted server
2. Spoof trusted server’s IP addr in SYN to X-terminal
3. Trusted server too busy to RST
4. ACK with the guessed seqno
5. Grant access to all sources
6. RSTs to trusted server (cleanup)
Defenses

- Initial sequence number must be difficult to predict!
TCP uses ACKs not only for reliability, but also for **congestion control**: the more ACKs come back, the faster I can send
Opt-ack attack

TCP uses ACKs not only for reliability, but also for congestion control: the more ACKs come back, the faster I can send
TCP uses ACKs not only for reliability, but also for **congestion control**: the more ACKs come back, the faster I can send
TCP uses ACKs not only for reliability, but also for congestion control: the more ACKs come back, the faster I can send.
TCP uses ACKs not only for reliability, but also for **congestion control**: the more ACKs come back, the faster I can send.
TCP uses ACKs not only for reliability, but also for congestion control: the more ACKs come back, the faster I can send
TCP uses ACKs not only for reliability, but also for **congestion control**: the more ACKs come back, the faster I can send
Opt-ack attack

Bytes 1000-1500

ACK 1501

Bytes 1501-2001

Bytes 2002-2502
Opt-ack attack

If I could convince you to send REALLY quickly, then you would effectively DoS your own network!
Opt-ack attack

A

Bytes 1000-1500

ACK 1501

Bytes 1501-2001

Bytes 2002-2502

B

But to get you to send faster, I need to get data in order to ACK, so I need to receive quickly

If I could convince you to send REALLY quickly, then you would effectively DoS your own network!
Opt-ack attack

But to get you to send faster, I need to get data in order to ACK, so I need to receive quickly ...

...or do I?

If I could convince you to send REALLY quickly, then you would effectively DoS your own network!
Opt-ack attack
Opt-ack attack

Bytes 1000-1500
Opt-ack attack

If I can predict what the last seqno will be *and* when A will send it
Opt-ack attack

If I can predict what the last seqno will be and when A will send it
Opt-ack attack

If I can predict what the last seqno will be and when A will send it

Then I could ACK early! ("optimistically")
Opt-ack attack

Then I could ACK early! ("optimistically")

If I can predict what the last seqno will be and when A will send it
Opt-ack attack

If I can predict what the last seqno will be and when A will send it,
Then I could ACK early! (“optimistically”)

Bytes 1000-1500

ACK 1501
ACK 2001
ACK 2502
Opt-ack attack

If I can predict what the last seqno will be and when A will send it, I could ACK early! ("optimistically")
Opt-ack attack

Bytes 1000-1500

If I can predict what the last seqno will be and when A will send it

Then I could ACK early! ("optimistically")

Bytes 1501-2001

Bytes 2002-2502

ACK 1501

ACK 2001

ACK 2502
Opt-ack attack

If I can predict what the last seqno will be and when A will send it

Then I could ACK early! (“optimistically”)

A will think “what a fast, legit connection!”
Opt-ack attack

A

Bytes 1000-1500

Bytes 1501-2001

Bytes 2002-2502

B

ACK 1501

ACK 2001

ACK 2502

Then I could ACK early! (“optimistically”)

If I can predict what the last seqno will be and when A will send it

A will think “what a fast, legit connection!”

Eventually, A’s outgoing packets will start to get dropped.
Then I could ACK early! (“optimistically”)

If I can predict what the last seqno will be and when A will send it

A will think “what a fast, legit connection!”

Eventually, A’s outgoing packets will start to get dropped.
Opt-ack attack

Then I could ACK early! ("optimistically")

If I can predict what the last seqno will be and when A will send it

A will think "what a fast, legit connection!"

Eventually, A's outgoing packets will start to get dropped.
Opt-ack attack

Then I could ACK early! (“optimistically”)

If I can predict what the last seqno will be and when A will send it

A will think “what a fast, legit connection!”

Eventually, A’s outgoing packets will start to get dropped.

But so long as I keep ACKing correctly, it doesn’t matter.
Opt-ack attack

Then I could ACK early! ("optimistically")

If I can predict what the last seqno will be and when A will send it

A will think "what a fast, legit connection!"

Eventually, A’s outgoing packets will start to get dropped.

But so long as I keep ACKing correctly, it doesn’t matter.
Amplification

- The big deal with this attack is its *Amplification Factor*
  - Attacker sends \(x\) bytes of data, causing the victim to send many more bytes of data in response
  - Recent examples: NTP, DNSSEC

- Amplified in TCP due to cumulative ACKs
  - “ACK \(x\)” says “I’ve seen all bytes up to but not including \(x\)”
Opt-ack’s amplification factor

- Max bytes sent by victim per ACK:

- Max ACKs attacker can send per second:
Opt-ack’s amplification factor

- Max bytes sent by victim per ACK:

\[
\text{Max window size} \times \frac{\text{MSS}}{\text{MSS}} \times (14 + 40 + \text{MSS})
\]

- Max ACKs attacker can send per second:
Opt-ack’s amplification factor

- Max bytes sent by victim per ACK:

\[
\text{Packets sent per ACK} \times \frac{\text{Max window size}}{\text{MSS}} \times (14 + 40 + \text{MSS})
\]

- Max ACKs attacker can send per second:

\[
\frac{\text{Attacker bandwidth (bytes/sec)}}{(14 + 40)}
\]
Opt-ack’s amplification factor

- Boils down to max window size and MSS
  - Default max window size: 65,536
  - Default MSS: 536

- Default amp factor: $65536 \times \left(\frac{1}{536} + \frac{1}{54}\right) \sim 1336\times$

- Window scaling lets you increase this by a factor of $2^{14}$

- Window scaling amp factor: $\sim 1336 \times 2^{14} \sim 22\text{M}$

- Using minimum MSS of 88: $\sim 32\text{M}$
Opt-ack defenses

- Is there a way we could defend against opt-ack in a way that is still compatible with existing implementations of TCP?

- An important goal in networking is *incremental deployment*: ideally, we should be able to benefit from a system/modification when even a subset of hosts deploy it.
NAMING

- IP addresses allow global connectivity

- But they’re pretty useless for humans!
  - Can’t be expected to pick their own IP address
  - Can’t be expected to remember another’s IP address

- **DHCP** : Setting IP addresses

- **DNS** : Mapping a memorable name to a routable IP address
DHCP

DYNAMIC HOST CONFIGURATION PROTOCOL

New host

DHCP server
### DHCP

**Dynamic Host Configuration Protocol**

<table>
<thead>
<tr>
<th>New host</th>
<th>DHCP server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doesn’t have an IP address yet (can’t set src addr)</td>
<td></td>
</tr>
<tr>
<td>New host</td>
<td>DHCP server</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Doesn’t have an IP address yet</td>
<td></td>
</tr>
<tr>
<td>(can’t set src addr)</td>
<td></td>
</tr>
<tr>
<td>Doesn’t know <em>who</em> to ask for one</td>
<td></td>
</tr>
</tbody>
</table>
**DHCP**

**DYNAMIC HOST CONFIGURATION PROTOCOL**

<table>
<thead>
<tr>
<th>New host</th>
<th>DHCP server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doesn’t have an IP address yet (can’t set src addr)</td>
<td></td>
</tr>
<tr>
<td>Doesn’t know who to ask for one</td>
<td></td>
</tr>
<tr>
<td>Solution: Discover one on the local subnet</td>
<td></td>
</tr>
</tbody>
</table>
DHCP

Dynamic Host Configuration Protocol

New host

- Doesn’t have an IP address yet (can’t set src addr)
- Doesn’t know who to ask for one
- Solution: Discover one on the local subnet

DHCP server

- DHCP discover (L2 broadcast)
DHCP

DYNAMIC HOST CONFIGURATION PROTOCOL

New host

Doesn’t have an IP address yet (can’t set src addr)

Doesn’t know who to ask for one

Solution: Discover one on the local subnet

DHCP server

DHCP discover (L2 broadcast)

DHCP offer
DHCP

DYNAMIC HOST CONFIGURATION PROTOCOL

New host

Doesn’t have an IP address yet (can’t set src addr)

 Doesn’t know who to ask for one

Solution: Discover one on the local subnet

DHCP server

offer includes: IP address, DNS server, gateway router, and duration of this offer (“lease” time)

DHCP discover (L2 broadcast)

DHCP offer
Doesn’t have an IP address yet (can’t set src addr)

Doesn’t know who to ask for one

Solution: Discover one on the local subnet

Doesn’t have an IP address yet (can’t set src addr)

Doesn’t know who to ask for one

Solution: Discover one on the local subnet

DHCP discover (L2 broadcast)

DHCP offer

DHCP request (L2 broadcast)

offer includes: IP address, DNS server, gateway router, and duration of this offer (“lease” time)
DHCP

Dynamic Host Configuration Protocol

New host

- Doesn’t have an IP address yet (can’t set src addr)
- Doesn’t know who to ask for one
- Solution: Discover one on the local subnet

DHCP server

- offer includes: IP address, DNS server, gateway router, and duration of this offer (”lease” time)
- request asks for the offered IP address

- DHCP discover (L2 broadcast)
- DHCP offer
- DHCP request (L2 broadcast)
**DHCP**

**DYNAMIC HOST CONFIGURATION PROTOCOL**

New host

- Doesn’t have an IP address yet (can’t set src addr)
- Doesn’t know who to ask for one
- Solution: Discover one on the local subnet

DHCP server

- Offer includes: IP address, DNS server, gateway router, and duration of this offer (“lease” time)
- Request asks for the offered IP address

---

DHCP discover (L2 broadcast)

DHCP offer

DHCP request (L2 broadcast)

DHCP ACK
DHCP ATTACKS

• Requests are broadcast: attackers on the same subnet can hear new host’s request

• Race the *actual* DHCP server to replace:
  • DNS server
    - Redirect any of a host’s lookups (“what IP address should I use when trying to connect to google.com?”) to a machine of the attacker’s choice
  • Gateway
    - The gateway is where the host sends all of its outgoing traffic (so that the host doesn’t have to figure out routes himself)
    - Modify the gateway to intercept all of a user’s traffic
    - Then relay it to the gateway (MITM)
    - How could the user detect this?
gold:~ dml$ ping google.com
PING google.com (74.125.228.65): 56 data bytes
64 bytes from 74.125.228.65: icmp_seq=0 ttl=52 time=22.330 ms
64 bytes from 74.125.228.65: icmp_seq=1 ttl=52 time=6.304 ms
64 bytes from 74.125.228.65: icmp_seq=2 ttl=52 time=5.186 ms
64 bytes from 74.125.228.65: icmp_seq=3 ttl=52 time=12.805 ms
HOSTNAMES AND IP ADDRESSES

gold:~ dml$ ping google.com
PING google.com (74.125.228.65): 56 data bytes
64 bytes from 74.125.228.65: icmp_seq=0 ttl=52 time=22.330 ms
64 bytes from 74.125.228.65: icmp_seq=1 ttl=52 time=6.304 ms
64 bytes from 74.125.228.65: icmp_seq=2 ttl=52 time=5.186 ms
64 bytes from 74.125.228.65: icmp_seq=3 ttl=52 time=12.805 ms
gold:~ dml$ ping google.com
PING google.com (74.125.228.65): 56 data bytes
64 bytes from 74.125.228.65: icmp_seq=0 ttl=52 time=22.330 ms
64 bytes from 74.125.228.65: icmp_seq=1 ttl=52 time=6.304 ms
64 bytes from 74.125.228.65: icmp_seq=2 ttl=52 time=5.186 ms
64 bytes from 74.125.228.65: icmp_seq=3 ttl=52 time=12.805 ms
google.com is easy to remember, but not routable

74.125.228.65 is routable

**Name resolution:**
The process of mapping from one to the other
TERMINOLOGY

- **www.cs.umd.edu** = “domain name”
- www.cs.umd.edu is a “subdomain” of cs.umd.edu

- Domain names can map to a set of IP addresses

```bash
gold:~ dml$ dig google.com

;; <<>> DiG 9.8.3-P1 <<>> google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 35815
;; flags: qr rd ra; QUERY: 1, ANSWER: 11, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;google.com. IN A

;; ANSWER SECTION:
google.com. 105 IN A 74.125.228.70
google.com. 105 IN A 74.125.228.66
google.com. 105 IN A 74.125.228.64
google.com. 105 IN A 74.125.228.69
google.com. 105 IN A 74.125.228.78
google.com. 105 IN A 74.125.228.73
google.com. 105 IN A 74.125.228.68
google.com. 105 IN A 74.125.228.65
google.com. 105 IN A 74.125.228.72
```

We’ll understand this more in a bit; for now, note that **google.com** is mapped to many IP addresses.
TERMINOLOGY

- **www.cs.umd.edu** = “domain name”
- **www.cs.umd.edu** is a “subdomain” of cs.umd.edu

- Domain names can map to a set of IP addresses

```
gold:~ dml$ dig google.com
; <<<〉 DiG 9.8.3-P1 <<<〉 google.com
;; global options: +cmd
;; Got answer:
;; -->HEADER<<- opcode: QUERY, status: NOERROR, id: 35815
;; flags: qr rd ra; QUERY: 1, ANSWER: 11, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;google.com. IN A

;; ANSWER SECTION:
google.com. 105 IN A 74.125.228.70
google.com. 105 IN A 74.125.228.66
google.com. 105 IN A 74.125.228.64
google.com. 105 IN A 74.125.228.69
google.com. 105 IN A 74.125.228.78
google.com. 105 IN A 74.125.228.73
google.com. 105 IN A 74.125.228.68
google.com. 105 IN A 74.125.228.65
google.com. 105 IN A 74.125.228.72
```

We’ll understand this more in a bit; for now, note that **google.com** is mapped to many IP addresses.
• “zone” = a portion of the DNS namespace, divided up for administrative reasons
  • Think of it like a collection of hostname/IP address pairs that happen to be lumped together
    - www.google.com, mail.google.com, dev.google.com, ...

• Subdomains do not need to be in the same zone
  • Allows the owner of one zone (umd.edu) to delegate responsibility to another (cs.umd.edu)
• “Nameserver” = A piece of code that answers queries of the form “What is the IP address for foo.bar.com?”
  • Every zone must run ≥2 nameservers
  • Several very common nameserver implementations: BIND, PowerDNS (more popular in Europe)

• “Authoritative nameserver”:
  • Every zone has to maintain a file that maps IP addresses and hostnames (“www.cs.umd.edu is 128.8.127.3”)
  • One of the name servers in the zone has the master copy of this file. It is the authority on the mapping.
TERMINOLOGY

• "Resolver" - while name servers answer queries, resolvers ask queries.

• Every OS has a resolver. Typically small and pretty dumb. All it typically does it forward the query to a local...

• "Recursive nameserver" - a nameserver which will do the heavy lifting, issuing queries on behalf of the client resolver until an authoritative answer returns.

• Prevalence
  • There is almost always a local (private) recursive name server
  • But very rare for name servers to support recursive queries otherwise
**TERMINOLOGY**

- "Record" (or "resource record") = usually think of it as a mapping between hostname and IP address

- But more generally, it can map virtually anything to virtually anything

- Many record types:
  - (A)ddress records (IP <-> hostname)
  - Mail server (MX, mail exchanger)
  - SOA (start of authority, to delineate different zones)
  - Others for DNSSEC to be able to share keys

- Records are the unit of information
Terminology

Nameservers within a zone must be able to give:

- **Authoritative answers (A)** for hostnames in that zone
  - The umd.edu zone’s nameservers must be able to tell us what the IP address for umd.edu is

"A" record: umd.edu = 54.84.241.99

54.84.241.99 is a valid IP address for umd.edu
TERMINOLOGY

Nameservers within a zone must be able to give:

- **Authoritative answers (A)** for hostnames in that zone
  - The umd.edu zone’s nameservers must be able to tell us what the IP address for umd.edu is

  "A" record: umd.edu = 54.84.241.99

  54.84.241.99 is a valid IP address for umd.edu

- **Pointers to name servers (NS)** who host zones in its subdomains
  - The umd.edu zone’s nameservers must be able to tell us what the name and IP address of the cs.umd.edu zone’s nameservers is

  "NS" record: cs.umd.edu = ipa01.cs.umd.edu

  Ask ipa01.cs.umd.edu for all cs.umd.edu subdomains
Domain Name Service at a very high level

What is an IP address for cs.umd.edu?
Domain Name Service at a very high level

What is an IP address for cs.umd.edu?
DNS

Domain Name Service at a very high level

What is an IP address for cs.umd.edu?
Domain Name Service at a very high level

What is an IP address for cs.umd.edu?
What is an IP address for cs.umd.edu?
What is an IP address for cs.umd.edu?
Domain Name Service at a very high level

What is an IP address for cs.umd.edu?
DNS

Domain Name Service at a very high level

What is an IP address for cs.umd.edu?
What is an IP address for cs.umd.edu?
Domain Name Service at a very high level

What is an IP address for cs.umd.edu?
What is an IP address for cs.umd.edu?
What is an IP address for cs.umd.edu?
What is an IP address for cs.umd.edu?
What is an IP address for cs.umd.edu?
What is an IP address for cs.umd.edu?
DNS

Domain Name Service at a very high level

What is an IP address for cs.umd.edu?

Caching responses is critical to DNS’s success.

Every response (3, 5, 7, 8) has a time-to-live (TTL). TTLs should be reasonably long (days), but some are minutes.
HOW DO THEY KNOW THESE IP ADDRESSES?

- Local DNS server: host learned this via DHCP
- A parent knows its children: part of the registration process
- Root nameserver: *hardcoded* into the local DNS server (and every DNS server)
  - 13 root servers (logically): A-root, B-root, …, M-root
  - These IP addresses change *very* infrequently
- **UMD runs D-root.**
  - IP address changed beginning of 2013!!
  - For the most part, the change-over went alright, but Lots of weird things happened — ask me some time.
CACHING

• Central to DNS’s success

• Also central to attacks

• “Cache poisoning”: filling a victim’s cache with false information
What is an IP address for cs.umd.edu?

Every query (2,4,6) has the same request in it ("what is the IP address for cs.umd.edu?")

But different:
- dst IP (port = 53)
- query ID
WHAT'S IN A RESPONSE?

• Many things, but for the attacks we’re concerned with…

• A record: gives “the authoritative response for the IP address of this hostname”

• NS record: describes “this is the name of the nameserver who should know more about how to answer this query than I do”
  • Often also contains “glue” records (IP addresses of those name servers to avoid chicken and egg problems)
  • Resolver will generally cache all of this information
The local resolver has a lot of incoming/outgoing queries at any point in time.

To determine which response maps to which queries, it uses a *query ID*

Query ID: 16-bit field in the DNS header

- Requester sets it to whatever it wants
- Responder must provide the same value in its response
• The local resolver has a lot of incoming/outgoing queries at any point in time.

• To determine which response maps to which queries, it uses a query ID.

• Query ID: 16-bit field in the DNS header.
  • Requester sets it to whatever it wants.
  • Responder must provide the same value in its response.
QUERY IDS USED TO INCREMENT

- Global query ID value
- Map outstanding query ID to local state of who to respond to (the client)
- Basically:
  new Packet(queryID++)
QUERY IDS USED TO INCREMENT

16322 16322
16323 16323
16328 16328

• Global query ID value
• Map outstanding query ID to local state of who to respond to (the client)
• Basically: new Packet(queryID++)

How would you attack this?
CACHE POISONING

Local nameserver

Bad guy 6.6.6.6
CACHE POISONING

Local nameserver

Bad guy 6.6.6.6

www.bank.com
CACHE POISONING

Local nameserver

Authoritative DNS server

Bad guy 6.6.6.6

www.bank.com
CACHE POISONING

Local nameserver

Authoritative DNS server

16322

Bad guy 6.6.6.6

www.bank.com
CACHE POISONING

Local nameserver

Authoritative DNS server

16322

Bad guy 6.6.6.6

16322:

www.bank.com
CACHE POISONING

Local nameserver 16322

Authoritative DNS server 6.6.6.6

Bad guy

www.bank.com
CACHE POISONING

Local nameserver 16322 -> Authoritative DNS server 16322:

Will cache www.bank.com = 6.6.6.6 and ignore authority’s answer

www.bank.com
CACHE POISONING

How do you guess this?

Will cache www.bank.com = 6.6.6.6 and ignore authority’s answer

www.bank.com
How do you guess this?

Will cache `www.bank.com = 6.6.6.6` and ignore authority’s answer.
How do you guess this?

Will cache www.bank.com = 6.6.6.6 and ignore authority’s answer
How do you guess this?

Will cache www.bank.com = 6.6.6.6 and ignore authority’s answer

Next is likely 16322

www.bank.com

www.bad.com
DETAILS OF GETTING THE ATTACK TO WORK

• Must guess query ID: ask for it, and go from there
  • Partial fix: randomize query IDs
  • Problem: small space
  • Attack: issue a Lot of query IDs

• Must guess source port number
  • Typically constant for a given server (often always 53)

• The answer must not already be in the cache
  • It will avoid issuing a query in the first place
CACHE POISONING

Can we do more harm than a single record?

Local nameserver

com. TLD

Bad guy 6.6.6.6
CACHÉ POISONING

Can we do more harm than a single record?

Local nameserver → com. TLD → www.bad.com → Bad guy 6.6.6.6
CACHE POISONING

Can we do more harm than a single record?

Local nameserver 16321 www.bad.com Bad guy 6.6.6.6

com. TLD
CACHE POISONING

Can we do more harm than a single record?

Local nameserver → com. TLD → www.bad.com

16321

Bad guy 6.6.6.6

Next is likely 16322
CACHE POISONING

Can we do more harm than a single record?

Local nameserver → com. TLD

www.bad.com

16321

Bad guy → 6.6.6.6

somethingnotcached.bank.com

Next is likely 16322
CACHE POISONING

Can we do more harm than a single record?

Local nameserver 16322

com. TLD

www.bad.com 16321

Bad guy 6.6.6.6

Next is likely somethingnotcached.bank.com 16322
CACHE POISONING

Can we do more harm than a single record?

Local nameserver

com. TLD

www.bad.com

Bad guy 6.6.6.6

Next is likely

somethingnotcached.bank.com

16322

NS bank.com = ns.bank.com

A ns.bank.com = 6.6.6.6

16321

16322:
CACHE POISONING

Can we do more harm than a single record?

Local nameserver

com. TLD

16322

16322

6.6.6.6

www.bad.com

Bad guy

16321

16322:

NS bank.com = ns.bank.com

A ns.bank.com = 6.6.6.6

Next is likely

somethingnotcached.bank.com

16322
CACHE POISONING

Can we do more harm than a single record?

Will cache “the person to ask for ALL bank.com queries is 6.6.6.6”
SOLUTIONS?

- Randomizing query ID?
  - Not sufficient alone: only 16 bits of entropy

- Randomize source port, as well
  - There’s no reason for it stay constant
  - Gets us another 16 bits of entropy

- DNSSEC?
DNSSEC

www.cs.umd.edu?

Root DNS server "."
www.cs.umd.edu?

Ask ".edu"
.edu’s public key = PK_{edu}
(Plus "."’s sig of this zone-key binding)
Ask "edu"
.edu's public key = PK_{edu}
(Plus "."'s sig of this zone-key binding)

www.cs.umd.edu?
www.cs.umd.edu?

Ask ".edu"
.edu's public key = PK_{edu}
(Plus "."'s sig of this zone-key binding)

www.cs.umd.edu?

Ask "umd.edu"
umd.edu's public key = PK_{umd}
(Plus "edu"'s sig of this zone-key binding)
www.cs.umd.edu?

Ask "edu"
.edu’s public key = PK_{edu} (Plus "."’s sig of this zone-key binding)

www.cs.umd.edu?

Ask "umd.edu"
umd.edu’s public key = PK_{umd} (Plus "edu”’s sig of this zone-key binding)

www.cs.umd.edu?
DNSSEC

www.cs.umd.edu?

Ask ".".edu"
.edu’s public key = PK_{edu}
(Plus "."’s sig of this zone-key binding)

Root DNS server ".".

TLD DNS server

www.cs.umd.edu?

Ask "umd.edu"
umd.edu’s public key = PK_{umd}
(Plus "edu”’s sig of this zone-key binding)

TLD DNS server

www.cs.umd.edu?

IN A www.cs.umd.edu 128.8.127.3
(Plus "umd.edu”’s signature of the answer)

Authoritative DNS server
DNSSEC

www.cs.umd.edu?

Ask ".edu"
.edu’s public key = PK_{edu}
(Plus "."’s sig of this zone-key binding)

www.cs.umd.edu?

Ask “umd.edu”
umd.edu’s public key = PK_{umd}
(Plus “edu”’s sig of this zone-key binding)

www.cs.umd.edu?

IN A www.cs.umd.edu 128.8.127.3
(Plus “umd.edu”’s signature of the answer)

Root DNS server "."

TLD DNS server

Authoritative DNS server

Only the authoritative answer is signed
PROPERTIES OF DNSSEC

• If everyone has deployed it, and if you know the root’s keys, then prevents spoofed responses
  • Very similar to PKIs in this sense

• But unlike PKIs, we still want authenticity despite the fact that not everyone has deployed DNSSEC
  • What if someone replies back without DNSSEC?
  • Ignore = secure but you can’t connect to a lot of hosts
  • Accept = can connect but insecure

• Back to our notion of incremental deployment
  • DNSSEC is not all that useful incrementally