Network III
DoS, DNS, TLS
CS 1660: Introduction to Computer Systems Security
DNS Domain Name System
www.example.com

208.77.188.166
Domain Name System

- The **domain name system** (DNS) is an application-layer protocol.
- **Basic function of DNS**
  - Map domain names to IP addresses.
  - The mapping is many to many.
- **Examples:**
  - `www.cs.brown.edu` and `cs.brown.edu` map to 128.148.32.12.
  - `google.com` maps to 198.7.237.251, 198.7.237.249, and other addresses.
- More generally, DNS is a distributed database that stores **resource records**:
  - **Address (A) record**: IP address associated with a host name.
  - **Mail exchange (MX) record**: mail server of a domain.
  - **Name server (NS) record**: authoritative server for a domain.
Domains

- **FQDN (Fully Qualified Domain Name)**
  - [Host name].[Domain].[TLD].[Root]
  - Two or more labels, separated by dots (e.g., cs.brown.edu)
- **Root name server**
  - It is a “.” at the end of the FQDN
- **Top-level domain (TLD)**
  - Generic (gTLD), e.g., .com, .org, .net
  - Country-code (ccTLD), e.g., .ca, .it
- **ICANN (Internet Corporation for Assigned Names and Numbers)**
  - "One World. One Internet."
    - Keeps database of registered gTLDs (InterNIC)
    - Accredits registrars for gTLDs
- **gTLDs**
  - Managed by ICANN
- **ccTLDs**
  - Managed by government organizations
Name Servers

• Name server
  • Keeps local database of DNS records
  • Answers DNS queries
  • Can ask other name servers if record not in local database
• Authoritative name server
  • Stores reference version of DNS records for a zone (partial tree)

• Examples
  • `dns.cs.brown.edu` is authoritative for `cs.brown.edu`
  • `bru-ns2.brown.edu` is authoritative for `brown.edu`

• Root servers
  • Authoritative for the root zone (TLDs)
  • `[a-m].root-servers.net` (ICANN)

• Command tools for checking DNS
  • `dig`, `nslookup`, `host` (similar results with some differences)
Name Resolution
Name Resolution

- **Resolver**
  - Program that retrieves DNS records
  - Caches records received
  - Connects to a name server (default, root, or given)

- **Iterative resolution**
  - Name server refers client to authoritative server (e.g., a TLD server) via an NS record
  - Repeat

- **Recursive resolution**
  - Name server queries another server and forwards the final answer (e.g., A record) to client
Iterative Name Resolution

Local Machine
- Application
- Resolver

Event:
- User queries www.google.com
- Local resolver queries (root) NS f.root-servers.net
- f.root-servers.net returns com NS d.gtld-servers.net
- Resolver queries d.gtld-servers.net
- d.gtld-servers.net returns google.com NS ns2.google.com
- Resolver queries ns2.google.com
- ns2.google.com returns A 74.125.226.116

Result:
- www.google.com A 74.125.226.116
Recursive Name Resolution

local machine

Application

Resolver

google.com server

Resolver

query

answer

A 74.125.226.176

other name server

Resolver

google.com

query

answer

A 74.125.226.176

...
Glue Records

• Circular references
  • The authoritative name server for a domain may be within the same domain
  • E.g., dns.cs.brown.edu is authoritative for cs.brown.edu

• Glue record
  • Record of type A (IP address) for a name server referred to NS record
  • Essential to break circular references

• Example
  • brown.edu. NS bru-ns1.brown.edu.
  • bru-ns1.brown.edu. A 128.148.248.11 [glue record]
DNS Caching
DNS Caching

• There would be too much network traffic if a path in the DNS tree would be traversed for each query
  – Root servers and TLD servers would be rapidly overloaded

• DNS servers cache records that are results of queries for a specified amount of time
  – Time-to-live field

• DNS queries with caching
  – First, resolver looks in cache for A record of query domain
  – Next, resolver looks in cache for NS record of longest suffix of query domain
Iterative Name Resolution with Caching

- Local Machine
  - Application
  - Resolver
  - DNS Cache
    - com NS d.gtld-servers.net
    - ...

- Resolver
  - (root)
    - Resolver
      - f.root-servers.net
  - Resolver
    - com
      - Resolver
        - d.gtld-servers.net
      - Resolver
        - google.com
          - Resolver
            - ns2.google.com

- google.com NS ns2.google.com
- www.google.com A 74.125.226.116
- www.google.com
- Application
- Resolver
- DNS Cache
Recursive Name Resolution with Caching

- Application
- Resolver
- Local name server
- Resolver
- DNS Cache
  - google.com A
  - 74.125.226.176
  - ...

Local machine

- Resolver
- Application

DoS, DNS, TLS
Local DNS Cache

• Operating system and some applications (e.g., browsers) maintain local DNS cache

• Operating system DNS cache
  – On some versions, DNS cache is shared among all running processes
  – Can be displayed by all users
  – View DNS cache in Windows with command `ipconfig /displaydns`
  – Clear DNS cache in Windows with command `ipconfig /flushdns`

• Privacy issues with shared operating system DNS cache
  • Browsing by other users can be monitored
  • Note that private/incognito browsing does not clear DNS cache
Imagine the following name resolution protocol: the resolver looks in the cache of the local name server, finds a record of the query domain, and returns it to the client. What category does this fall under?

A. Iterative Name Resolution
B. Iterative Name Resolution with caching
C. Recursive Name Resolution
D. Recursive Name Resolution with caching
E. Both B and D
Imagine the following name resolution protocol: the resolver looks in the cache of the local name server, finds a record of the query domain, and returns it to the client. What category does this fall under?

A. Iterative Name Resolution
B. Iterative Name Resolution with caching
C. Recursive Name Resolution
D. Recursive Name Resolution with caching
E. Both B and D
DNS Attacks
DNS Cache Poisoning

- Basic idea
  - Give a DNS server a false address record and get it cached
- DNS query mechanism
  - Queries issued over UDP on port 53
  - 16-bit request identifier in payload to match answers with queries
  - No authentication
  - No encryption

- Cache may be poisoned when a resolver
  - Disregards identifiers
  - Has predictable identifiers and return ports
  - Accepts unsolicited DNS records
ISP DNS Server

evil

www.example.com ID = 42 IP = 2.2.2.2

www.example.com

client

www.example.com

www.example.com ID = 42 IP = 1.1.1.1

www.example.com ID = 42 IP = 2.2.2.2

www.example.com IP = 2.2.2.2

DNS Lookup
DNS Cache Poisoning Defenses

• Query randomization
  • Random request identifier (16 bits)
• Probability of guessing request ID or return port
  \[ \frac{1}{2^{16}} = 0.0015\% \]

• Check request identifier
• Use signed records
  • DNSSEC
An attacker with a rogue machine on your local area network is sniffing traffic and wants to poison your DNS cache. Your DNS resolver uses both query ID and return port randomization. Is the poisoning attack going to succeed?

A. Not at all
B. Only with small probability $1 / 2^{16}$
C. Only with very small probability $1 / 2^{32}$
D. Likely
An attacker with a rogue machine on your local area network is sniffing traffic and wants to poison your DNS cache. Your DNS resolver uses both query ID and return port randomization. Is the poisoning attack going to succeed?

A. Not at all
B. Only with small probability $1 / 2^{16}$
C. Only with very small probability $1 / 2^{32}$
D. Likely
Kaminsky’s Attack

• Attacker causes victim to send
  • Many DNS requests for nonexistent subdomains of target domain
• Attacker sends victim
  • Forged NS responses for the requests
• Format of forged response
  • Random ID
  • Correct NS record
  • Spoofed glue record pointing to the attacker’s name server IP
  • http://unixwiz.net/techtips/iguide-kaminsky-dns-vuln.html

Requests
- 000.brown.edu
- 001.brown.edu
- ...999.brown.edu

Spoofed responses
- 000.brown.edu
- NS ns2.brown.edu
- ns2.brown.edu
- A 66.66.66.66
- ...
DNS Cache Poisoning Defenses after Kaminsky’s attack

- Query randomization
  - Random request identifier (16 bits)
  - Random return port (16 bits)
- Probability of guessing request ID or return port
  \[ \frac{1}{2^{16}} = 0.0015\% \]
- Probability of guessing request ID and return port is
  \[ \frac{1}{2^{32}} \text{ (less than one in four billion)} \]

- Check request identifier
- Use signed records
  - DNSSEC
DNS Hijacking/Redirecting

• Subvert the resolution of DNS queries

• **Malicious:** you type "bank.com" and attacker directs you to incorrect IP address

• **Censorship:** e.g. Great Firewall of China

• DoS: hacktivist can use to block dns resolution

• **ISPs:**
  • Display ads (instead of or in addition to existing ads),
  • Collect statistics about user traffic.
  • Block access to websites.
Denial-of-Service (DOS)
Denial-of-Service (DoS)

Idea: Disrupt operation of a host or service by making it unable to fulfill requests

• Attack on availability
Denial-of-Service (DoS)

Idea: Disrupt operation of a host or service by making it unable to fulfill requests

• Attack on availability

How?

• Overwhelm the target with messages
• Disrupt some resource the target requires for operation (ie, power, network connectivity, OS, DNS, ...)
How it works

Network-based: exhaust target's available network bandwidth for handling legitimate traffic
How it works

Network-based: exhaust target's available bandwidth for handling legitimate traffic

If attacker can generate traffic at a rate that exceeds B's bottleneck link capacity, legitimate packets will be dropped!
How it works

Application/OS-based: exhaust some resource used by target OS or application (eg. using all available memory)

Eg. What if request R forces server to call malloc()? 

Attacker sends lots of requests of type R => server uses all memory, can't respond to actual requests
How it works

Application/OS-based: exhaust some resource used by target OS or application (eg. using all available memory)

Eg. What if request R forces server to call malloc()?

Attacker sends lots of requests of type R
=> Server uses all memory, can't respond to actual requests
Distributed Denial-of-Service (DDoS)

- Single attacker: limited by attacker's bandwidth, ability to make requests, ...
- Can have much higher impact if attack is distributed across many hosts
Distributed Denial-of-Service (DDoS)

- Single attacker: limited by attacker's bandwidth, ability to make requests, ...
- Can have much higher impact if attack is distributed across many hosts

How?
- Usually: attacker controls a botnet: a huge group of small, infected machines that send packets on its behalf
- Modern botnets are complex and highly-distributed systems—as complex as the cloud services they attack!
DDoS: Targets

• Small scale:
DNS as a Target

• Oct. 21 2016: Spotify, Reddit, NYT, Wired, and many more became partially unavailable from the East Coast.

• **Dyn** provides DNS services to these companies, and was targeted with a massive DDoS attack.

  => Caused by Mirai botnet: >500K infected consumer devices
Example: Mirai Botnet

• Responsible for some of the largest DDoS attacks observed and studied
• Composed of cheap, insecure Internet of Things (IoT) devices
  • Consumer products: cameras, DVRs, home routers, ...
  • Primary vulnerability: weak login credentials
• Infected ~65K devices in first hour, ~200-300K steady state

Actually a complex distributed system!
Really good writeup/talk here
<table>
<thead>
<tr>
<th>Password</th>
<th>Device Type</th>
<th>Password</th>
<th>Device Type</th>
<th>Password</th>
<th>Device Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456</td>
<td>ACTi IP Camera</td>
<td>kv1234</td>
<td>HiSilicon IP Camera</td>
<td>1111</td>
<td>Xerox Printer</td>
</tr>
<tr>
<td>anko</td>
<td>ANKO Products DVR</td>
<td>jvbdz</td>
<td>HiSilicon IP Camera</td>
<td>Zte521</td>
<td>ZTE Router</td>
</tr>
<tr>
<td>pass</td>
<td>Axis IP Camera</td>
<td>admin</td>
<td>IPX-DDK Network Camera</td>
<td>1234</td>
<td>Unknown</td>
</tr>
<tr>
<td>888888</td>
<td>Dahua DVR</td>
<td>system</td>
<td>IQinVision Cameras</td>
<td>12345</td>
<td>Unknown</td>
</tr>
<tr>
<td>666666</td>
<td>Dahua DVR</td>
<td>meinsm</td>
<td>Mobotix Network Camera</td>
<td>admin1234</td>
<td>Unknown</td>
</tr>
<tr>
<td>vizxv</td>
<td>Dahua IP Camera</td>
<td>54321</td>
<td>Packet8 VOIP Phone</td>
<td>default</td>
<td>Unknown</td>
</tr>
<tr>
<td>7ujMko0vizxv</td>
<td>Dahua IP Camera</td>
<td>00000000</td>
<td>Panasonic Printer</td>
<td>fucker</td>
<td>Unknown</td>
</tr>
<tr>
<td>7ujMko0admin</td>
<td>Dahua IP Camera</td>
<td>realtek</td>
<td>RealTek Routers</td>
<td>guest</td>
<td>Unknown</td>
</tr>
<tr>
<td>666666</td>
<td>Dahua IP Camera</td>
<td>1111111</td>
<td>Samsung IP Camera</td>
<td>password</td>
<td>Unknown</td>
</tr>
<tr>
<td>dreambox</td>
<td>Dreambox TV Receiver</td>
<td>xmhdpcc</td>
<td>Shenzhen Anran Camera</td>
<td>root</td>
<td>Unknown</td>
</tr>
<tr>
<td>juantech</td>
<td>Guangzhou Juan Optical</td>
<td>smcadmin</td>
<td>SMC Routers</td>
<td>service</td>
<td>Unknown</td>
</tr>
<tr>
<td>xc3511</td>
<td>H.264 Chinese DVR</td>
<td>ikwb</td>
<td>Toshiba Network Camera</td>
<td>support</td>
<td>Unknown</td>
</tr>
<tr>
<td>OxhlwSG8</td>
<td>HiSilicon IP Camera</td>
<td>ubnt</td>
<td>Ubiquiti AirOS Router</td>
<td>tech</td>
<td>Unknown</td>
</tr>
<tr>
<td>cat1029</td>
<td>HiSilicon IP Camera</td>
<td>supervisor</td>
<td>VideoIQ</td>
<td>user</td>
<td>Unknown</td>
</tr>
<tr>
<td>hi3518</td>
<td>HiSilicon IP Camera</td>
<td>&lt;none&gt;</td>
<td>Vivotek IP Camera</td>
<td>zlxz.</td>
<td>Unknown</td>
</tr>
</tbody>
</table>

Table 5: **Default Passwords** — The 09/30/2016 Mirai source release included 46 unique passwords, some of which were traceable to a device vendor and device type. Mirai primarily targeted IP cameras, DVRs, and consumer routers.
Mirai: Capabilities

Mirai was a DDoS-for-hire service

- Received commands from control network with DDoS targets
- Various types of DDoS attacks supported
- Up to 600Gbps total bandwidth at its peak
Mirai: Capabilities

Mirai was a DDoS-for-hire service
• Received commands from control network with DDoS targets
• Various types of DDoS attacks supported
• Up to 600Gbps total bandwidth at its peak

Original version shut down (and author arrested), but code was open-sourced
• Lots of evolution since then
• See link for details
How to perform a DDoS?

Idea: flood target with lots of packets

What types of packets? Need...
• Small packets (easy to send, low-bandwidth)
• Causes resource usage on target

A few tricks to maximize attacker's capabilities...
Examples of attack types

- HTTP flood: request generates resources on host
- SYN/ACK: TCP resource exhaustion (more on this later)
- NTP: application vulnerability would cause crashes
- DNS flood

<table>
<thead>
<tr>
<th>Attack Type</th>
<th>Attacks</th>
<th>Targets</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP flood</td>
<td>2,736</td>
<td>1,035</td>
<td>A</td>
</tr>
<tr>
<td>UDP-PLAIN flood</td>
<td>2,542</td>
<td>1,278</td>
<td>V</td>
</tr>
<tr>
<td>UDP flood</td>
<td>2,440</td>
<td>1,479</td>
<td>V</td>
</tr>
<tr>
<td>ACK flood</td>
<td>2,173</td>
<td>875</td>
<td>S</td>
</tr>
<tr>
<td>SYN flood</td>
<td>1,935</td>
<td>764</td>
<td>S</td>
</tr>
<tr>
<td>GRE-IP flood</td>
<td>994</td>
<td>587</td>
<td>A</td>
</tr>
<tr>
<td>ACK-STOMP flood</td>
<td>830</td>
<td>359</td>
<td>S</td>
</tr>
<tr>
<td>VSE flood</td>
<td>809</td>
<td>550</td>
<td>A</td>
</tr>
<tr>
<td>DNS flood</td>
<td>417</td>
<td>173</td>
<td>A</td>
</tr>
<tr>
<td>GRE-ETH flood</td>
<td>318</td>
<td>210</td>
<td>A</td>
</tr>
</tbody>
</table>

Table 9: C2 Attack Commands—Mirai launched 15,194 attacks between September 27, 2016–February 28, 2017. These include [A]pplication-layer attacks, [V]olumetric attacks, and TCP [S]tate exhaustion, all of which are equally prevalent.
Tricks: IP address spoofing

- Many networks don't actually check the IP source field
- Attacker can send packets with a spoofed address
  - Harder to detect source, can help with attack efficiency...
IP spoofing

A 1.2.3.4

DNS server 8.8.8.8

Target 9.9.9.9

Src 9.9.9.9 8.8.8.8

IP A?

Dst somewhere.com
IP spoofing

A
1.2.3.4

DNS server
8.8.8.8

Target
9.9.9.9

IP
Src
9.9.9.9
8.8.8.8
somewhere.com

A?
Dst
8.8.8.8

Src
IP
9.9.9.9
Dst
8.8.8.8

somewhere.com is at ...

=> Actual attack packets sent by another server responding to attacker’s query!
IP spoofing with amplification

- **A**: 1.2.3.4
- **DNS server**: 8.8.8.8
- **Target**: 9.9.9.9

**Request**: small <50B

**Response**: could be much larger!! (hundreds of bytes or more!)

**IP A?**
- **Src**: 9.9.9.9
- **Dst**: 8.8.8.8
- **somewhere.com**

**IP A?**
- **Src**: 9.9.9.9
- **Dst**: 8.8.8.8
- **somewhere.com is at ...**

4/18/23

DoS, DNS, TLS
IP spoofing with amplification

**Request:** small <50B

**Response:** could be much larger!! (hundreds of bytes or more!)

Larger responses => more bandwidth to target, with low expense to attacker

---

A 1.2.3.4

DNS server 8.8.8.8

Target 9.9.9.9

IP spoofing with amplification

**Request:** small <50B

**Response:** could be much larger!! (hundreds of bytes or more!)

Larger responses => more bandwidth to target, with low expense to attacker

---

4/18/23

DoS, DNS, TLS
Defenses: Single host

• **Attempt #1: Make sure you have enough memory**
  • How much is “enough”?  
  • Depends on your threat model (how many resources do you think the attacker has?); might be hard to know  
  • ...and highly motivated adversary will just find (your limit + 1) resource  

• **Attempt #2: Firewall**
  • Identify evil IP addresses; refuse service to them  
  • Users might not use the same IP address  
    Can’t authenticate a user (i.e. via password) because we need an established connection to do that!  
  • Attacker can spoof addresses
Defenses?

• **Attempt #3: Outsource it**
  • Someone with lots of memory
  • Someone with lots of network bandwidth

Who has this?
- Large cloud companies
- Content Distribution Networks (CDNs)
  => Often provide DDoS mitigation services
Content Distribution Networks (high level)

- Cloud services with widely-distributed networks
- Can act as caches or proxies for other applications or services
Example: Cloudflare

The Cloudflare global network

Our vast global network, which is one of the fastest on the planet, is trusted by millions of web properties.

With direct connections to nearly every service provider and cloud provider, the Cloudflare network can reach 95% of the world's population within approximately 50 ms.

285
cities in 100+ countries, including mainland China

11,500
networks directly connect to Cloudflare, including every major

192 Tbps
global network edge capacity, consisting of transit connections,

~50 ms
from 95% of the world's Internet-connected population

4/18/23
DoS, DNS, TLS
Demo: How Akamai works

- Akamai (another CDN) has cache servers deployed close to clients
  - Co-located with many ISPs
- Challenge: make same domain name resolve to a proxy close to the client
- Lots of DNS tricks. BestBuy is a customer
  - Delegate name resolution to Akamai (via a CNAME)

From Brown:

dig www.bestbuy.com

;; ANSWER SECTION:

www.bestbuy.com.edgesuite.net. 21600 IN CNAME a1105.b.akamai.net.
a1105.b.akamai.net. 20 IN A 198.7.236.235
a1105.b.akamai.net. 20 IN A 198.7.236.240

- Ping time: 2.53ms

From Berkeley, CA:

a1105.b.akamai.net. 20 IN A 198.189.255.200
a1105.b.akamai.net. 20 IN A 198.189.255.207

- Ping time: 3.20ms
**DNS Resolution**

```
dig www.bestbuy.com

;; ANSWER SECTION:
www.bestbuy.com.edgesuite.net. 21600 IN CNAME a1105.b.akamai.net.
a1105.b.akamai.net. 20 IN A 198.7.236.235
a1105.b.akamai.net. 20 IN A 198.7.236.240

;; AUTHORITY SECTION:
b.akamai.net. 1101 IN NS n1b.akamai.net.
b.akamai.net. 1101 IN NS n0b.akamai.net.

;; ADDITIONAL SECTION:
n0b.akamai.net. 1267 IN A 24.143.194.45
n1b.akamai.net. 2196 IN A 198.7.236.236
```

- **n1b.akamai.net** finds an edge server close to the client’s local resolver
- Uses knowledge of network: BGP feeds, traceroutes. *Their secret sauce…*
dig www.bestbuy.com
; QUESTION SECTION:
@ IN A 104.88.86.223

; ANSWER SECTION:
www.bestbuy.com.edgekey.net. 85 IN CNAME e1382.x.akamaiedge.net.
e1382.x.akamaiedge.net. 58 IN CNAME www.bestbuy.com.

; Query time: 6 msec
; SERVER: 192.168.1.1#53(192.168.1.1)
; WHEN: Thu Nov 16 09:43:11 2017
; MSG SIZE rcvd: 123

traceroute to 104.88.86.223 (104.88.86.223), 64 hops max, 52 byte packets
1 192.168.1.1 2.461 ms 1.647 ms 1.178 ms
2 138.16.160.253 1.845 ms 1.599 ms 1.462 ms
3 10.1.18.5 (10.1.18.5) 1.886 ms 1.785 ms 1.707 ms
4 10.1.80.5 (10.1.80.5) 4.276 ms 6.444 ms 2.307 ms
5 labinet-r-235.net.brown.edu (128.148.235.6) 1.004 ms 1.878 ms 1.727 ms
6 131.109.200.1 (131.109.200.1) 2.841 ms 2.587 ms 2.538 ms
7 host-198-7-224-105.oshean.org (198.7.224.105) 4.421 ms 4.523 ms 4.496 ms
8 5-1-4.bearl.boston1.level3.net (4.53.54.21) 4.099 ms 3.974 ms 4.290 ms
9 128.148.230.6 (128.148.230.6) 4.689 ms 4.109 ms
10 ae-6.r24.nycmny81.us.bb.gin.ntt.net (129.250.4.114) 8.863 ms 10.285 ms
11 ae-1.r40.bb.gin.ntt.net (129.250.6.52) 9.298 ms
12 ae-0.r40.bb.gin.ntt.net (129.250.3.181) 10.006 ms 8.677 ms
13 a23-68-221-144.deploy.static.akamaitechnologies.com (104.88.86.223) 9.470 ms

Other DNS servers to try:
77.88.8.8 (St Petersburg), 89.233.43.71 (Copenhagen), 202.46.32.22 (Beijing)
How CDNs prevent DDoS

- DDoS => Widely distributed attack, large bandwidth
  
  vs.

- CDN => Widely distributed network, significant bandwidth
  - Distribute attack load across global scale
  - Also outsourced monitoring, analysis, etc.
Example: Cloudflare architecture
Demo: Anycast

• CDNs don't just work with DNS!
Transport Layer (Ports, TCP, UDP)
The Transport Layer

Network layer: moving data between hosts
Transport layer: Abstraction for getting data data to different applications on a host
The Transport Layer

Network layer: moving data between hosts
Transport layer: Abstraction for getting data data to different applications on a host

- Multiplexing multiple connections at same IP with port numbers
- Series of packets => stream of data/messages
- May provide: reliable data delivery

Two key protocols: TCP, UDP
From earlier: OSI Model

- End host
  - Application
  - Presentation
  - Session
  - Transport
  - Network
  - Data link
  - Physical

Application Protocol

Transport Protocol

Network Protocol

Link-Layer Protocol

One or more nodes within the network
What’s a port number?

• 16-bit unsigned number, 0-65535
• Ports define a communication *endpoint*, usually a process/service on a host
• OS keeps track of which ports map to which applications
What’s a port number?

• 16-bit unsigned number, 0-65535
• Ports define a communication *endpoint*, usually a process/service on a host
• OS keeps track of which ports map to which applications

Port numbering
• port < 1024: “Well known port numbers”
• port >= 20000: “ephemeral ports”, for general app. use
### Some common ports

<table>
<thead>
<tr>
<th>Port</th>
<th>Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>20, 21</td>
<td>File Transfer Protocol (FTP)</td>
</tr>
<tr>
<td>22</td>
<td>Secure Shell (SSH)</td>
</tr>
<tr>
<td>23</td>
<td>Telnet (pre-SSH remote login)</td>
</tr>
<tr>
<td>25</td>
<td>SMTP (Email)</td>
</tr>
<tr>
<td>53</td>
<td>Domain Name System (DNS)</td>
</tr>
<tr>
<td>67, 68</td>
<td>DHCP</td>
</tr>
<tr>
<td>80</td>
<td>HTTP (Web traffic)</td>
</tr>
<tr>
<td>443</td>
<td>HTTPS (Secure HTTP over TLS)</td>
</tr>
</tbody>
</table>
How ports work

Two modes:

• Applications "listen on" or "bind to" a port to wait for new connections

• Hosts make connections to a particular IP and port
How ports work

Two modes:

• Applications "listen on" or "bind to" a port to wait for new connections
  => Example:  webserver listens on port 80

• Hosts make connections to a particular IP and port
  => Example:  client connects to <webserver IP>, port 80
  (eg. 1.2.3.4:80)
A must know B is listening on port 80 ➞ "well known numbers"!

When connecting, A's OS picks random source port (eg. 12345), used for its side of connection
A
1.2.3.4

connect(1.2.3.4, 80)

B
5.6.7.8

listen(80)

Src Dst
IP 1.1.1.1 5.6.7.8
Port 12345 80

B responds to A using this port

Src Dst
IP 5.6.7.8 1.2.3.4
Port 80 12345
Sockets

OS keeps track of which application uses which port

Two types:

- Listening ports
- Connections between two hosts (src/dst port)

Socket: OS abstraction for a network connection, like a file descriptor

Table maps: port => socket

Want to know more? Take CS1680!
### Netstat

<table>
<thead>
<tr>
<th>Proto</th>
<th>Recv-Q</th>
<th>Send-Q</th>
<th>Local Address</th>
<th>Foreign Address</th>
<th>(state)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcp4</td>
<td>0</td>
<td>0</td>
<td>10.3.146.161.51094</td>
<td>104.16.248.249.443</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>tcp4</td>
<td>0</td>
<td>0</td>
<td>10.3.146.161.51076</td>
<td>172.66.43.67.443</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>tcp6</td>
<td>0</td>
<td>0</td>
<td>2620:6e:6000:900.51074</td>
<td>2606:4700:3108::443</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>tcp4</td>
<td>0</td>
<td>0</td>
<td>10.3.146.161.51065</td>
<td>35.82.230.35.443</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>tcp4</td>
<td>0</td>
<td>0</td>
<td>10.3.146.161.51055</td>
<td>162.159.136.234.443</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>tcp4</td>
<td>0</td>
<td>0</td>
<td>10.3.146.161.51038</td>
<td>17.57.147.5.5223</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>tcp6</td>
<td>0</td>
<td>0</td>
<td><em>.</em>.22</td>
<td><em>.</em></td>
<td>LISTEN</td>
</tr>
<tr>
<td>tcp4</td>
<td>0</td>
<td>0</td>
<td><em>.</em>.51036</td>
<td><em>.</em></td>
<td>LISTEN</td>
</tr>
<tr>
<td>tcp4</td>
<td>0</td>
<td>0</td>
<td>127.0.0.1.9999</td>
<td><em>.</em></td>
<td>LISTEN</td>
</tr>
</tbody>
</table>

- **netstat -an**: Show all connections
- **netstat -1np**: Show listening ports + applications using them (as root)
Why do we care?

Ports define what services are exposed to the network

- Open port: can send data to application (reconnaissance, attacks, ...)
- Surface for DDoS
- OS and network hardware can monitor port numbers
  - Make decisions on how to filter/monitor traffic
Demo: netcat
What can we learn if we just start connecting to well-known ports?

• Can discover things about the network
• Can learn about vulnerabilities
How to defend ports?

Single system or organization

- **Stateless**: block specific ports, IP ranges
- **Stateful**: track connection state, block certain types of connection state

Large scale

- Distribute load/filtering in the network => cloud provider
Firewall policy example
What We Have Learned

• DoS in Networks
  – Attacks and defenses
• IP address Spoofing

• How DNS operates
  – Distributed database
  – Resolvers and name servers
  – Iterative vs. recursive resolution
  – Caching
• DNS cache poisoning
• DNS Hijacking
References

- **Logjam** attack (2015)
More on transport layer
Recall: TCP Establishment Handshake

- SYN: Seq = x
- SYN-ACK: Seq = y, Ack = x + 1
- ACK: Seq = x + 1, Ack = y + 1

What happens on the server?
Recall: TCP Establishment Handshake

Server creates state associated with connection (client IP, port, counters...)

What happens if client doesn’t send this ACK?

- SYN
  Seq = x

- SYN-ACK
  Seq = y, Ack = x + 1

- ACK
  Seq = x + 1, Ack = y + 1
Recall: TCP Establishment Handshake

- SYN
  Seq = x

- SYN-ACK
  Seq = y, Ack = x + 1

Server creates state associated with connection (client IP, port, counters...)

- ACK
  Seq = x + 1, Ack = y + 1

What happens if adversary doesn’t send this ACK?
Recall: TCP Establishment Handshake

**Sequence: SYN**
- Seq = x

**Sequence: SYN-ACK**
- Seq = y, Ack = x + 1

**Sequence: ACK**
- Seq = x + 1, Ack = y + 1

The adversary does not send the ACK, so the server will create a state associated with the connection (client IP, port, counters...). A single SYN from the adversary forces the server to spend some memory.
Recall: TCP Establishment Handshake

What happens if an adversary doesn’t send this ACK?

A single SYN from adversary forces the server to spend some memory...
Recall: TCP Establishment Handshake

What happens if adversary doesn’t send this ACK?

A single SYN from adversary forces the server to spend some memory...

Resource exhaustion
Recall: TCP Establishment Handshake

<table>
<thead>
<tr>
<th>SYN</th>
<th>SYN</th>
<th>SYN</th>
<th>SYN</th>
<th>SYN</th>
<th>SYN</th>
<th>SYN</th>
<th>SYN</th>
<th>SYN</th>
</tr>
</thead>
</table>

Server creates state associated with connection (client IP, port, counters...)

What happens if adversary doesn’t send this ACK?

A single **SYN** from adversary forces the server to spend some memory...

**resource exhaustion**
SYN Flooding

- Attacker targets server memory rather than network capacity
- Every (unique) SYN forces the server to spend memory
  - Server can’t necessarily clear up the memory (at least, not right away)
- What happens when the server runs out of memory?
  - Refuse new connection?
    - Legitimate new users can’t access service
  - Evict old connections?
    - Legitimate old users get kicked out
Defenses?

• **Attempt #1: Make sure you have enough memory**
  • How much is “enough”? 
  • Depends on your threat model (how many resources do you think the attacker has?); might be hard to know 
  • …and highly motivated adversary will just find (your limit + 1) resource

• **Attempt #2: Firewall**
  • Identify evil IP addresses; refuse service to them 
  • Users might not use the same IP address
    Can’t authenticate a user (i.e. via password) because we need an established connection to do that!
  • Attacker can spoof addresses
Defenses?

• **Attempt #3: Outsource it**
  • Someone with lots of memory
  • Someone with lots of network bandwidth
Idea: Outsource Your State

Client sends back the state

SYN
Seq = x

SYN-ACK
Seq = y, Ack = x + 1

ACK
Seq = x + 1, Ack = y + 1

Don’t create state here; instead, give it to the client

Server only saves state here
SYN Cookies

Idea: Encode state entirely within the SYN-ACK sequence number

- SYN
  Seq = x

- SYN-ACK
  Seq = y, Ack = x + 1

  \[ \text{Seq} = y, \text{Ack} = x + 1 \]

  = HMAC of (IP / port / seq) w/ secret key

- ACK
  Seq = x + 1, Ack = y + 1

  = HMAC of (IP / port / seq) w/ secret key

Server validates, then saves state here
SYN Cookies

• General security strategy: rather than holding state, encode it so that it is still “trustable” when it’s returned
  • Attacker now needs to complete 3-way handshake in order to burden server (why is this okay?)
Last Remarks

• Some remarks:
  • You need enough bits to encode all the state (just barely enough for HMAC) is really great for this though—use a truncated HMAC.
  Using $b$ bits of HMAC ensures $\Pr[H_k(x) = H_k(y)] = 0.5^b$, so hash collisions are rare (in practice $b = 24$; remaining 8 bits are TCP boilerplate).
  • If it’s expensive to create or validate cookie, then it’s not good. Digital signatures would be expensive—more resource exhaustion.
    Once again, HMAC is really great for this.
  • Key idea: you can force others to store all the data you want, as long as you make sure to verify it later.
    • Make sure to remember this! You’ll see this later in the course...

• **TCP Cookie Transactions** (TCPCT) and **TCP Fast Open** are other approaches to mitigate syn flooding.
More on scanning
Large-scale port scanning

• Can reveal lots of open/insecure systems!
• Examples:
  • shodan.io
  • VNC roulette
  • Open webcam viewers..
  • …
• Also: penetration testing/vulnerability scanning (more on this later)
Disclaimer

• Network scanning is easy to detect

• Unless you are the owner of the network, it’s seen as malicious activity

• If you scan the whole Internet, the whole Internet will get mad at you (unless done very politely)

• Do NOT try this on the Brown network. I warned you.
Scanning I have done

- Scanned IPv4 space for ROS (Robot Operating System)
- Found ~200 “things” using ROS (some robots, some other stuff)