Week 9: Lecture B
Networking 101
Thursday, October 26, 2023
Project 3: WebSec released

- **Deadline**: Thursday, November 9th by 11:59PM

---

**Project 3: Web Security**

**Deadline:** Thursday, November 9 by 11:59PM.

Before you start, review the course syllabus for the Lateness, Collaboration, and Ethical Use policies. You may optionally work alone, or in teams of **at most two** and submit **one project per team**. If you have difficulties forming a team, post on Piazza's Search for Teammates forum. Note that the final exam will cover project material, so you and your partner should collaborate on each part.

The code and other answers your group submits must be entirely your own work, and you are bound by the University's Student Code. You may consult with other students about the conceptualization of the project and the meaning of the questions, but you may not look at any part of someone else's solution or collaborate with anyone outside your group. You may consult published references, provided that you appropriately cite them (e.g., in your code comments). **Don't risk your grade and degree by cheating!**

Complete your work in the **CS 4440 VM**—we will use this same environment for grading. You may not use any **external dependencies**. Use only default Python 3 libraries and/or modules we provide you.
Project 3 progress

- Working on Part 1: 0%
- Working on Part 2: 0%
- Working on Part 3: 0%
- None of the above: 0%
Announcements

- **Project 2** grades are now available on **Canvas**

- **Statistics:**
  - Average score: **91%**

- **Fantastic job!**
Announcements

- **Project 2** grades are now available on **Canvas**

- Think we made an error? Request a regrade!
  - Valid regrade requests:
    - You have verified your solution is correct (i.e., we made an error in grading)

**Project 2 Regrade Requests** (see Piazza pinned link):
Submit by **11:59 PM** on **Monday 10/30** via **Google Form**
Announcements

See Discord for meeting info!

www.utahsec.com
Questions?
Last time on CS 4440...

Isolation-based Web Security
HTTPS, SSL, and TLS
Client-side web security should uphold...

- Confidentiality
  - ???
- Integrity
  - ???
- Privacy
  - ???
Client-side web security should uphold...

- **Confidentiality**
  - My sensitive information stays private

- **Integrity**
  - My computer and data aren’t tampered

- **Privacy**
  - My online activities are known only to me
Client-side Web Defenses

- Multi-process Browsing
  - ???
Multi-process Browsing
- Each tab, plugin, etc. gets its own unique process
- Leverage power of MMU to enforce process isolation
- Compromised process can’t read/write memory from other page processes

Caveat:
- ???

Client-side Web Defenses
Multi-process Browsing
- Each tab, plugin, etc. gets its own unique process
- Leverage power of MMU to enforce process isolation
- Compromised process can’t read/write memory from other page processes

Caveat:
- More tabs, more plugins, etc. creates more overhead
Client-side Web Defenses

- Remote Pixel Streaming
  - ???
Remote Pixel Streaming
- Browser lives in the cloud, not the client’s system
- Rendering done in cloud, not on client’s system
- Client only gets “streamed” version of rendered pages
- Thwarts client-side attacks

Caveat:
- ???
Remote Pixel Streaming
- Browser lives in the cloud, not the client’s system
- Rendering done in cloud, not on client’s system
- Client only gets “streamed” version of rendered pages
- Thwarts client-side attacks

Caveat:
- Consumes lots of bandwith
- Bulkier browsing experience
Client-side Web Defenses

- DOM Tree Mirroring
  - ???
**Client-side Web Defenses**

- **DOM Tree Mirroring**
  - Filters-out DOM elements deemed to be unsafe
  - User only gets “safe” DOM
  - List of undesired elements is defined ahead of time

- **Caveat:**
  - ???
Client-side Web Defenses

- **DOM Tree Mirroring**
  - Filters-out DOM elements deemed to be unsafe
  - User only gets “safe” DOM
  - List of undesired elements is defined ahead of time

- **Caveat:**
  - Need to **constantly update** list of unsafe elements
  - Must **retrofit** browsers
Client-side Web Defenses

- Tagged JS Sandboxing
  - ???
Client-side Web Defenses

- **Tagged JS Sandboxing**
  - Follow Same Origin Policy
  - Block JavaScript access based on site’s origin
  - Scripts from same origin can read/write/interact with others from origin
  - Scripts from different origin denied access

- **Caveat:**
  - ???
Tagged JS Sandboxing
- Follow Same Origin Policy
- Block JavaScript access based on site’s origin
- Scripts from same origin can read/write/interact with others from origin
- Scripts from different origin denied access

Caveat:
- Doesn’t stop XSS attacks
The Same-origin Policy

- Restricts access to content from the same origin (protocol + host)
The Same-origin Policy

- Restricts access to content from the same origin (protocol + host)
- Try the following, comparing to http://cs4440.eng.utah.edu/project1

<table>
<thead>
<tr>
<th>Candidate Request</th>
<th>SOP Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="https://cs4440.eng.utah.edu/project3">https://cs4440.eng.utah.edu/project3</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="http://cs4440.eng.utah.edu/project3/sqlinject0">http://cs4440.eng.utah.edu/project3/sqlinject0</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ftp://cs4440.eng.utah.edu</td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="http://www.cs4440.eng.utah.edu">http://www.cs4440.eng.utah.edu</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="https://eng.utah.edu/">https://eng.utah.edu/</a></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The Same-origin Policy

- Restricts access to content from the same origin (protocol + host)
- Try the following, comparing to http://cs4440.eng.utah.edu/project1

<table>
<thead>
<tr>
<th>Candidate Request</th>
<th>SOP Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="https://cs4440.eng.utah.edu/project3">https://cs4440.eng.utah.edu/project3</a></td>
<td>FAIL</td>
<td>Different protocol (https)</td>
</tr>
<tr>
<td><a href="http://cs4440.eng.utah.edu/project3/sqlinfect0">http://cs4440.eng.utah.edu/project3/sqlinfect0</a></td>
<td>PASS</td>
<td>Same protocol and host</td>
</tr>
<tr>
<td>ftp://cs4440.eng.utah.edu</td>
<td>FAIL</td>
<td>Different protocol (ftp)</td>
</tr>
<tr>
<td><a href="http://www.cs4440.eng.utah.edu">http://www.cs4440.eng.utah.edu</a></td>
<td>FAIL</td>
<td>Different host (www)</td>
</tr>
<tr>
<td><a href="https://eng.utah.edu/">https://eng.utah.edu/</a></td>
<td>FAIL</td>
<td>Different protocol and host</td>
</tr>
</tbody>
</table>

Candidate Request

- Try the following, comparing to http://cs4440.eng.utah.edu/project1

<table>
<thead>
<tr>
<th>Candidate Request</th>
<th>SOP Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="https://cs4440.eng.utah.edu/project3">https://cs4440.eng.utah.edu/project3</a></td>
<td>FAIL</td>
<td>Different protocol (https)</td>
</tr>
<tr>
<td><a href="http://cs4440.eng.utah.edu/project3/sqlinfect0">http://cs4440.eng.utah.edu/project3/sqlinfect0</a></td>
<td>PASS</td>
<td>Same protocol and host</td>
</tr>
<tr>
<td>ftp://cs4440.eng.utah.edu</td>
<td>FAIL</td>
<td>Different protocol (ftp)</td>
</tr>
<tr>
<td><a href="http://www.cs4440.eng.utah.edu">http://www.cs4440.eng.utah.edu</a></td>
<td>FAIL</td>
<td>Different host (www)</td>
</tr>
<tr>
<td><a href="https://eng.utah.edu/">https://eng.utah.edu/</a></td>
<td>FAIL</td>
<td>Different protocol and host</td>
</tr>
</tbody>
</table>
Secure web communication should uphold...

- Integrity
  - ???
- Confidentiality
  - ???
- Authenticity
  - ???
Secure web communication should uphold...

- **Integrity**
  - Messages I send should not be tampered

- **Confidentiality**
  - Messages private to only involved parties

- **Authenticity**
  - I should know exactly who I’m talking to
The TLS Handshake

Client Hello: Here’s Ciphers I support, and a random
The TLS Handshake

**Client Hello:** Here’s **Ciphers** I support, and a **random**

**Server Hello:** **Chosen Cipher**

**Certificate:** Here is my **Certificate** with my **PubKey**

Here’s your **random** back encrypted with my **PrivKey**
The TLS Handshake

**Client Hello:** Here’s **Ciphers** I support, and a **random**

**Server Hello:** Chosen **Cipher**

**Certificate:** Here is my **Certificate** with my **PubKey**

Here’s your **random** back encrypted with my **PrivKey**

**Key Exchange:** Our **SymKey** encrypted with your **PubKey**
The TLS Handshake

**Client Hello:** Here’s **Ciphers** I support, and a **random**

**Server Hello:** Chosen **Cipher**

**Certificate:** Here is my **Certificate** with my **PubKey**

Here’s your **random** back encrypted with my **PrivKey**

**Key Exchange:** Our **SymKey** encrypted with your **PubKey**

Switch to a **Symmetric Cipher**

Switch to a **Symmetric Cipher**
Higher-level TLS Handshake

**Client says:** “Howdy! Here is what cipher suites I support.”
“Here is a random number for you to encrypt.”
Client says:  “Howdy! Here is what cipher suites I support.”
“Here is a random number for you to encrypt.”

Server says:  “Howdy! Let’s go with this specific cipher.”
“Here is my signed certificate containing my public key.”
“Here is your random encrypted with my private key.”
**Higher-level TLS Handshake**

**Client says:** “Howdy! Here is what cipher suites I support.”  
“Here is a **random** number for you to encrypt.”

**Server says:** “Howdy! Let’s go with *this* specific cipher.”  
“Here is my **signed certificate** containing my **public key**.”  
“Here is your **random** encrypted with my **private key**.”

Client verifies Server’s authenticity from its **certificate**; and by decrypting the  
**Server-encrypted random** via Server’s **public key** and checking it to the original.
Higher-level TLS Handshake

**Client says:**  “Howdy! Here is what cipher suites I support.”
“Here is a **random** number for you to encrypt.”

**Server says:**  “Howdy! Let’s go with **this** specific cipher.”
“Here is my **signed certificate** containing my **public key**.”
“Here is your **random** encrypted with my **private key**.”

Client verifies Server’s authenticity from its **certificate**; and by decrypting the **Server-encrypted random** via Server’s **public key** and checking it to the original.

**Client says:**  “Great! You are who you say you are. Here’s our **symmetric key**.”
Higher-level TLS Handshake

Client says: “Howdy! Here is what cipher suites I support.”
“Here is a random number for you to encrypt.”

Server says: “Howdy! Let’s go with this specific cipher.”
“Here is my signed certificate containing my public key.”
“Here is your random encrypted with my private key.”

Client verifies Server’s authenticity from its certificate; and by decrypting the Server-encrypted random via Server’s public key and checking it to the original.

Client says: “Great! You are who you say you are. Here’s our symmetric key.”

We do not expect you to memorize the hairy details about SSL/TLS!
Why does the server send back the client's random nonce encrypted?

- If client can decrypt with their own private key, the server is verified! 0%
- If client can decrypt with server's private key, the server is verified! 0%
- If client can decrypt with server's public key, the server is verified! 0%
- None of the above 0%
Our HTTPS Ecosystem

- Certificate: ???
Our HTTPS Ecosystem

- **Certificate**: the verifiable “proof” of the server’s **authenticity**
  - The client (i.e., you) wants to know it is talking to **who it believes it is**
  - Also contains the server’s public key, issuer information, expiration, etc.
  - Your browser does **lots of checks** to ensure it's dealing with a valid certificate!

```
Subject: C=US/O=Google Inc/CN=www.google.com
Issuer: C=US/O=Google Inc/CN=Google Internet Authority
Expiration Period: Jul 12 2010 - Jul 19 2012
Public Key Algorithm: rsaEncryption
Signature Algorithm: sha1WithRSAEncryption
```
Our HTTPS Ecosystem

- **Certificate**: the verifiable "proof" of the server's authenticity
- The client (i.e., you) wants to know it is talking to who it believes it is
- Also contains the server's public key, issuer information, expiration, etc.
- Your browser does lots of checks to ensure it's dealing with a valid certificate!

**Subject:** C=US/O=Google Inc/CN=www.google.com

**Issuer:** C=US/O=Google Inc/CN=Google Internet Authority


**Expiration Period:** Jul 12 2010 - Jul 19 2012

**Public Key Algorithm:** rsaEncryption

**Public Key:**

```
```

**Signature Algorithm:** sha1WithRSAEncryption

**Signature:**

```
```

"THEY ARE WHO WE THOUGHT THEY WERE"

- the client
Our HTTPS Ecosystem

- Certificate Authority: ???
Our HTTPS Ecosystem

- **Certificate Authority**: the trusted entity that *vouches for* certificate
  - Acts as a notary for server’s certificate
Our HTTPS Ecosystem

- **Certificate Authority**: the trusted entity that **vouches for** certificate
  - Acts as a notary for server’s certificate
- Certificates are chained together
**Our HTTPS Ecosystem**

- **Certificate Authority:** the trusted entity that **vouches for** certificate
  - Acts as a notary for server’s certificate

- Certificates are chained together
  - Links are **intermediate certificates**
  - Ultimately begins in a **root certificate**
    - Your browser “walks” chain to locate certificate that it trusts
  - Anyone can sign a certificate...
    - But if not chained to a **root** certificate, it is **not valid!**
Our HTTPS Ecosystem

- **Certificate Authority:** the trusted entity that **vouches for** certificate
  - Acts as a notary for server’s certificate

- **Certificates:**
  - Links are **intermediate certificates**
  - Ultimately begins in a **root certificate**
  - Your browser “walks” chain to locate certificate that it trusts
  - Anyone can sign a certificate...
  - But if not chained to a **root** certificate, it is **not valid**!

**What could go wrong?**
Attacking HTTPS: via Key Theft
Attacking HTTPS: via Key Theft

- **What can happen if...**
  - Only **server’s** private key stolen:
    - ???
  - Only **client’s** private key stolen:
    - ???
  - **Both** private keys are stolen:
    - ???
Attacking HTTPS: via Key Theft

What can happen if...
- Only server’s private key stolen:
  - Fake comms to the client!
- Only client’s private key stolen:
  - Fake comms to the server!
- Both private keys are stolen:
  - Full man-in-the-middle!

Don’t leave your private keys lying around on public web!
Attacking HTTPS: via HTTP

- Forcing HTTPS instead to **HTTP**
  - The HTTP content is **not** encrypted
    - Attacker can intercept and read
  - User would need to add an exception
    - Possible with some trickery?
Attacking HTTPS: via HTTP

- Forcing HTTPS instead to HTTP
  - The HTTP content is **not** encrypted
    - Attacker can intercept and read
  - User would need to add an exception
    - Possible with some trickery?

- Attacking “mixed content” sites
  - An HTTPS page that retrieves some material (e.g., scripts) over HTTP
  - Can access HTTPS page’s **DOM**
    - Access data that attacker otherwise wouldn’t be able to
Attacking HTTPS: via HTTP

- **Forcing HTTPS instead to HTTP**
  - The HTTP content is not encrypted
  - Attacker can intercept and read
  - User would need to add an exception
  - Possible with some trickery?

- **Attacking “mixed content” sites**
  - An HTTPS page that retrieves some material (e.g., scripts) over HTTP
  - Can access HTTPS page’s DOM
  - Access info that attacker otherwise wouldn’t be able to

Modern web browsers eliminate such attacks.
Attacking HTTPS: via Certificate Authorities

- **Certificate Authorities** are what the security of HTTPS depends on
  - If an attacker manages to breach a CA, they can sign any certificate they want
Attacking HTTPS: via Certificate Authorities

- **Certificate Authorities** are what the security of HTTPS depends on
  - If an attacker manages to breach a CA, they can sign any certificate they want

**Result:** attacker can **impersonate** websites that you use—your browser will **accept their certs** as legitimate!
Attacking HTTPS: via Certificate Authorities

- **Real-world example:** DigiNotar
  - DigiNotar **was** a Dutch Certificate Authority
  - On June 10, 2011, *.google.com cert was issued to an attacker and subsequently used to perform **man-in-the-middle attacks** in Iran
  - Nobody noticed until someone found the cert in the wild... and posted it to pastebin
Attacking HTTPS: via Certificate Authorities

- **Real-world example:** DigiNotar
  - DigiNotar was a Dutch Certificate Authority
  - On June 10, 2011, *.google.com cert was issued to an attacker and subsequently used to perform **man-in-the-middle attacks** in Iran
  - Nobody noticed until someone found the cert in the wild... and posted it to pastebin

- DigiNotar later admitted that **dozens of fraudulent certificates** were created
  - Google, Microsoft, Apple and Mozilla all **revoked** the root Diginotar certificate
  - Dutch Government took over Diginotar
  - Diginotar went bankrupt and died

---

**DigiNotar Internet Trust Services**

**Google Security Blog**

The latest news and insights from Google on security and safety on the Internet

---

**An update on attempted man-in-the-middle attacks**
August 29, 2011

Posted by Heather Adams, Information Security Manager

Today we received reports of attempted SSL man-in-the-middle (MITM) attacks against Google users, whereby someone tried to get between them and encrypted Google services. The people affected were primarily located in Iran. The attacker used a fraudulent SSL certificate issued by DigiNotar, a root certificate authority that should not issue certificates for Google (and has since revoked it).

Google Chrome users were protected from this attack because Chrome was able to detect the fraudulent certificate.
The Google webmail of as many as 300,000 Iranians may have been intercepted using fraudulently issued security certificates made after a hack against Dutch certificate authority outfit DigiNotar, according to the preliminary findings of an official report into the megahack.

Between 10 July and 20 July hackers used compromised access to DigiNotar's systems to issue rogue 531 SSL certificate for Google and other domains, including Skype, Mozilla add-ons, Microsoft update and others. DigiNotar only began revoking rogue certificates on 19 July and waited more than a month after this to go public. The fake *.google.com certificate – which was valid for code-signing – wasn't revoked until 29 July.
Questions?
This time on CS 4440...

Introduction to Networking
The Physical, Link, Network, Transport, and Application Layers
What is the Internet?

- What is it?
What is the Internet?

- **What is it?**
  - How you trash-talk players in COD game lobbies
  - How Wall Street trades shares faster than you
  - How the CS 4440 website is distributed to you
What really is the Internet?

- **Connections**
  - HTTP, HTTPS, FTP, VOIP

- **The Web**
  - Content viewed in a web browser

- **How many internets?**
  - U.S.A. vs. China
  - TOR vs. non-TOR

- **What separates them?**
Analogy: Air Travel

- Each **layer** implements a service

**Departure Airport**
- Ticket (*purchase*)
- Baggage (*check*)
- Gate (*load*)
- Runway (*takeoff*)

**Destination Airport**
- Ticket (*complain*)
- Baggage (*claim*)
- Gate (*unload*)
- Runway (*land*)

**Routing** (*flying*)
The 5-layer Internet

Physical Layer

Ethernet
Fiber
WiFi
The 5-layer Internet

- **Applications initiating connections**
  - Application
  - Transport
  - Network
  - Link

- Physical Layer
  - Ethernet
  - Fiber
  - WiFi
The 5-layer Internet

Establishes connections
The 5-layer Internet

Network Layer

Transport Layer

Application Layer

Physical Layer:
- Ethernet
- Fiber
- WiFi

Forms/sends packets between IP addresses
The 5-layer Internet

- **Application**
- **Transport**
- **Network**
- **Link**

Physical Layer:
- **Ethernet**
- **Fiber**
- **WiFi**

Creates and sends frames
The 5-layer Internet

Physical Layer

Sends the physical bits

Application
Transport
Network
Link

Ethernet
Fiber
WiFi

Application
Transport
Network
Link

Application
Transport
Network
Link
Networking Devices

- **Network layer:**
  - **Router**
    - Connects different networks

- **Data Link layer:**
  - **Switch**
    - Connects multiple devices on the same network
  - **Modem**
    - Aka modulator/demodulator
    - Interface between 0/1 bits and cable/telephone wire
Internet Packet Encapsulation

- How packets are generated and sent
Internet Packet Encapsulation

- How packets are generated and sent

App Layer

Transport Layer

Application Message

Segment Header

Segment Data
Internet Packet Encapsulation

- How packets are generated and sent
Internet Packet Encapsulation

- How packets are generated and sent

![Diagram of packet encapsulation with layers and components]
Internet Packet Encapsulation

- How packets are generated and sent

What you care about

Application Message

Segment Header  Segment Data

Packet Header  Packet Data

Frame Header  Frame Data  Frame Footer

App Layer
Transport Layer
Network Layer
Link Layer
Internet Packet Encapsulation

- How packets are generated and sent

What really gets sent

App Layer
Transport Layer
Network Layer
Link Layer
Internet Packet Encapsulation

- How packets are generated and sent

What really gets sent

Creates a direct-connection abstraction

App Layer
Transport Layer
Network Layer
Link Layer
Layering of Protocols

Application Layer
- DNS
- SSH
- FTP
- SMTP
- NTP
- HTTP

Transport layer
- UDP
- TCP

Network layer
- IP

Link layer
- ATM
- SONET
- Ethernet
Layering of Protocols

Why do we rely on layering?
Why do we rely on layering?

Transparency, modularization
Questions?
The Physical Layer
Layer 5: The Physical Layer

- **Last layer** in the 5-layer network model
  - The physical means of sending/receiving data

- **Examples** of physical layers?
  - ???
Layer 5: The Physical Layer

- **Last layer** in the 5-layer network model
  - The physical means of sending/receiving data
- **Examples** of physical layers?
  - Radio waves
  - Telephone lines
  - Fiber optic cables
  - Undersea submarine cables
Evolution of the Physical Layer

- **ARPANET**: precursor to today’s Internet
  - **University of Utah** was one of its four nodes!
- Each member **physically linked** by cables
Evolution of the Physical Layer

- **ARPANET**: precursor to today’s Internet
  - University of Utah was one of its four nodes!

- Each member **physically linked** by cables

- By the 1990s: connected by **Telephone lines**
Evolution of the Physical Layer

- **ARPANET**: precursor to today’s Internet
  - University of Utah was one of its four nodes!
- Each member **physically linked** by cables
- By the 1990s: connected by **Telephone lines**
- Today: continents linked via **undersea cables**
The Link Layer
Layer 4: Link / Data-Link

- Hosts and switches: **nodes**
  - **Switches** interface with **hosts**

- Channels connecting adjacent nodes along a path: **links**
  - Wired links
  - Wireless links
  - LANs

- Layer-2 packet: **frame**
  - Encapsulates datagram of the previous three TCP/IP layers
Most network interfaces come with a predefined **MAC address**
- 48-bit number usually represented in hex
- E.g., 00-1A-92-D4-BF-86

The First three octets of any MAC address are IEEE-assigned Organizationally Unique Identifiers
- Cisco: 00-1A-A1
- D-Link: 00-1B-11
- ASUSTek: 00-1A-92
MAC Addresses

- Most network interfaces come with a predefined **MAC address**
  - 48-bit number usually represented in hex
  - E.g., 00-1A-92-D4-BF-86

- The First three octets of any MAC address are IEEE-assigned Organizationally Unique Identifiers
  - Cisco: 00-1A-A1
  - D-Link: 00-1B-11
  - ASUSTek: 00-1A-92

- MACs **can be reconfigured** by network interface driver software
  - This makes MAC address filtering insecure—they can easily be spoofed!
The “dominant” wired LAN technology:
- First widely used LAN technology
- Simpler, cheaper than token LANs and ATM
- Kept up with speed race: 10 Mbps – 100 Gbps

Ethernet Frame
- How the data is packaged up, sent/received
- Destination and source MACs, payload, and checksum
Where is the link layer implemented?

- **In each and every host!**
  - “Adaptor” (aka network interface card)
    - Ethernet card
    - 802.11 card
    - Ethernet chipset

- **Implements link and physical layer**
  - Attaches into host’s system buses
  - Combination of hardware and firmware
The Network Layer
Layer 3: Network

- Deliver **segment** from sending to receiving hosts
  - **Sender** encapsulates segments into IP datagrams
  - **Receiver** delivers segments to transport layer
  - Delivery based on logical addressing (i.e., IP addresses)

- Network layer protocols in every host, router
  - Router checks headers of IP datagrams passing through

![Diagram of packets creation, transport, and assembly](image)
Network Layer Functions

- **Routing**: determine *route taken* by packets from source to dest
  - Works based on IP addresses
  - Ideally aims to find **shortest path** for the packet to its destination
Network Layer Functions

- **Routing**: determine **route taken** by packets from source to dest
  - Works based on IP addresses
  - Ideally aims to find **shortest path** for the packet to its destination

- **Forwarding**: move packets from router’s **input** to router **output**
  - Can’t store full IP addrs—too huge!
  - Instead, a table based on IP **prefixes**
    - Get prefix from input packet
    - Choose its corresponding **link**
Network Layer Functions

- **Routing**: determine **route taken** by packets from source to dest
  - Works based on IP addresses
  - Ideally aims to find **shortest path** for the packet to its destination

- **Forwarding**: move packets from router’s **input** to router **output**
  - Can’t store full IP addr—to too huge!
  - Instead, a table based on IP **prefixes**
    - Get prefix from input packet
    - Choose its corresponding **link**
Internet Protocol

- **IP addresses**: routes datagrams in Internet
  - **IPv4**: 32 bit address
  - **IPv6**: 128 bit address

- **Two parts: network and host**
  - **Network**: used to route packets (ZIP code)
  - **Host**: identifies an individual host (house number)
  - Split between network/host based on address class
  - Usually in dotted decimal notation: \(141.211.144.212\)
    - Each number represents 8 bits: 0–255
IP Packets

- **Header:**
  - Source IP address
  - Destination IP address
  - Lots of other information
    - Version, length, checksum
    - Selected transport protocol

- **Data:**
  - The message!
    - E.g., string of letters
    - E.g., web page characters
The Transport Layer
Layer 2: Transport

- Provides **logical communication** between application processes running on **different hosts**

- Transport protocols in end systems
  - **Send side**: breaks app messages into segments, passes to network layer
  - **Receive side**: reassembles segments into messages, passes to app layer

- Nowadays, **multiple** transport protocols available
  - Internet: TCP and UDP
Transport Services

- **TCP: Transmission Control Protocol**
  - **Flow control:** sender won’t overwhelm receiver with packets
  - **Congestion control:** throttle sender when network overloaded
**TCP: Transmission Control Protocol**

- **Flow control:** sender won't overwhelm receiver with packets
- **Congestion control:** throttle sender when network overloaded
- **Doesn't provide:** timing, minimum throughput guarantee, security
- **Connection-oriented:** setup required between client and server
Transport Services

- **TCP: Transmission Control Protocol**
  - **Flow control**: sender won’t overwhelm receiver with packets
  - **Congestion control**: throttle sender when network overloaded
  - **Doesn’t provide**: timing, minimum throughput guarantee, security
  - **Connection-oriented**: setup required between client and server

- **UDP: User Datagram Protocol**
  - **Simpler protocol** for transmission without any error-checking
Transport Services

- **TCP: Transmission Control Protocol**
  - **Flow control:** sender won’t overwhelm receiver with packets
  - **Congestion control:** throttle sender when network overloaded
  - **Doesn’t provide:** timing, minimum throughput guarantee, security
  - **Connection-oriented:** setup required between client and server

- **UDP: User Datagram Protocol**
  - **Simpler protocol** for transmission without any error-checking
  - **Does not provide:** reliability, flow or congestion control, timing, throughput guarantee, security, or connection setup
Transport Services

- **TCP: Transmission Control Protocol**
  - **Flow control:** sender won’t overwhelm receiver with packets
  - **Congestion control:** throttle sender when network overloaded
  - **Doesn’t provide:** timing, minimum throughput guarantee, security
  - **Connection-oriented:** setup required between client and server

- **UDP: User Datagram Protocol**
  - **Simpler protocol** for transmission without any error-checking
  - **Does not provide:** reliability, flow or congestion control, timing, throughput guarantee, security, or connection setup
The Application Layer
Layer 1: Application

- Defines the following:
  - **Types** of messages exchanged
    - E.g., requests, responses
  - Message syntax:
    - Message **fields**, how they are delineated
  - Message semantics:
    - The **meaning** of information in each field
  - Rules for **when/how** processes **send/respond** to messages

![Application Layer Diagram]

- **APPLICATION_LAYER**
- **Request content**
- **Return content in required format**
- **Website**
Example: HTTP Requests

Request line (E.g., GET, POST, HEAD commands)

GET /index.html HTTP/1.1\r\nHost: www.cs.utah.edu\r\nUser-Agent: Firefox/3.6.10\r\nAccept: text/html,application/xhtml+xml\r\nAccept-Language: en-us,en;q=0.5\r\nAccept-Encoding: gzip,deflate\r\nAccept-Charset: ISO-8859-1,utf-8;q=0.7\r\nKeep-Alive: 115\r\nConnection: keep-alive\n\r\nCarriage return character
Line-feed character

Carriage return, line feed at start of line indicates end of header lines
Example: HTTP Requests

- What actually gets transmitted:

```
method  sp  URL  sp  version  cr  lf
header field name  value  cr  lf
header field name  value  cr  lf
                                                                                       entity body
```

- request line
- header lines
- body
Many open-source protocols we use daily

Examples:
- HTTP: Hypertext Transfer Protocol
- SMTP: Simple Mail Transfer Protocol
- FTP: File Transfer Protocol
Protocols Galore

- Many open-source protocols we use daily
  - Examples:
    - HTTP: Hypertext Transfer Protocol
    - SMTP: Simple Mail Transfer Protocol
    - FTP: File Transfer Protocol
  - Allows for:
    - Interoperability
    - Third-party security vetting
Protocols Galore

- Many **open-source** protocols we use daily
  - Examples:
    - HTTP: Hypertext Transfer Protocol
    - SMTP: Simple Mail Transfer Protocol
    - FTP: File Transfer Protocol
  - Allows for:
    - Interoperability
    - Third-party security vetting

- **Closed-source** proprietary protocols:
  - Examples: Skype, Zoom
Many **open-source** protocols we use daily

- **Examples:**
  - HTTP: Hypertext Transfer Protocol
  - SMTP: Simple Mail Transfer Protocol
  - FTP: File Transfer Protocol
- **Allows for:**
  - Interoperability
  - Third-party security vetting

**Closed-source** proprietary protocols:

- **Examples:** Skype, Zoom
- **Makes security vetting really difficult!**
Food for Thought

- Are any of the five network layers susceptible to attacks? If so, which ones?

```
<table>
<thead>
<tr>
<th>Frame</th>
<th>Frame Data</th>
<th>Frame Footer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet</td>
<td>Packet Data</td>
<td>Packet Header</td>
</tr>
<tr>
<td>Segment</td>
<td>Segment Data</td>
<td>Segment Header</td>
</tr>
<tr>
<td>Application</td>
<td>Application Message</td>
<td></td>
</tr>
<tr>
<td>Layer</td>
<td>Transport Layer</td>
<td>Network Layer</td>
</tr>
<tr>
<td>Link Layer</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Next time on CS 4440...

Application-layer Network Attacks