

# Week 8: Lecture A

## Introduction to The Web

Tuesday, October 15, 2024

# Announcements

- **Project 2: AppSec** released
  - **Deadline:** October 17th by 11:59PM (**this Thursday**)

## Project 2: Application Security

**Deadline: Thursday, October 17 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.

### Helpful Resources

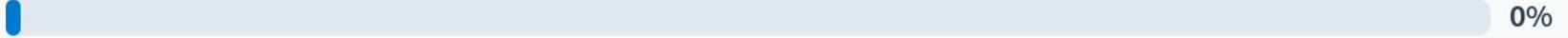
- [The CS 4440 Course Wiki](#)
- [VM Setup and Troubleshooting](#)
- [Terminal Cheat Sheet](#)
- [GDB Cheat Sheet](#)
- [x86 Cheat Sheet](#)
- [C Cheat Sheet](#)

### Table of Contents:

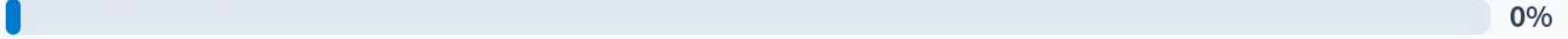
- [Helpful Resources](#)
- [Introduction](#)
- [Objectives](#)
- [Start by reading this!](#)
  - [Setup Instructions](#)
  - [Important Guidelines](#)
- [Part 1: Beginner Exploits](#)
  - [Target 0: Variable Overwrite](#)
  - [Target 1: Execution Redirect](#)
  - [What to Submit](#)
- [Part 2: Intermediate Exploits](#)
  - [Target 2: Shellcode Redirect](#)
  - [Target 3: Indirect Overwrite](#)
  - [Target 4: Beyond Strings](#)
  - [What to Submit](#)
- [Part 3: Advanced Exploits](#)
  - [Target 5: Bypassing DEP](#)
  - [Target 6: Bypassing ASLR](#)
  - [What to Submit](#)
- [Part 4: Super L33T Pwnage](#)
  - [Extra Credit: Target 7](#)
  - [Extra Credit: Target 8](#)
  - [What to Submit](#)
- [Submission Instructions](#)

## Project 2 Progress Update

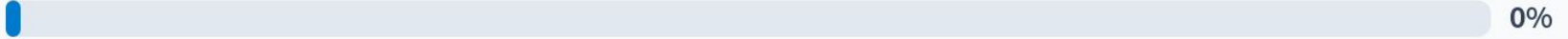
Working on Targets 0-2



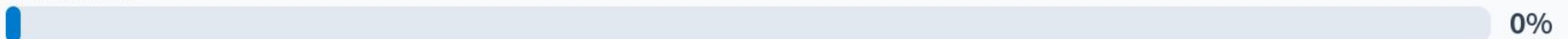
Working on Targets 3-4



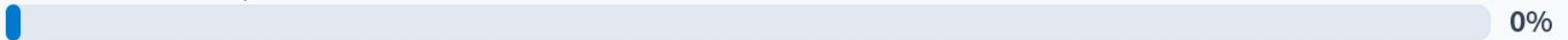
Working on Targets 5-6



Finished!



Haven't started :(



# Announcements

- **Project 3: WebSec** released
  - **Deadline:** Thursday, November 7th by 11:59PM

## Project 3: Web Security

**Deadline: Thursday, November 7 by 11:59PM.**

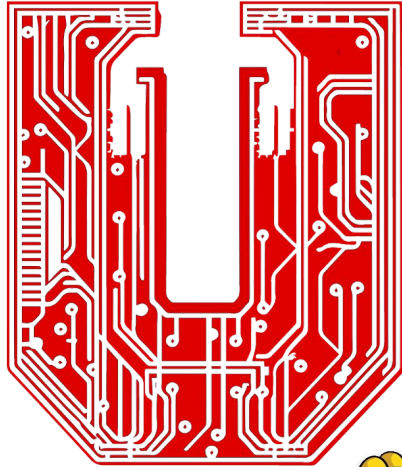
Before you start, review the [course syllabus](#) for the Lateness, Collaboration, and Ethical Use policies.

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.

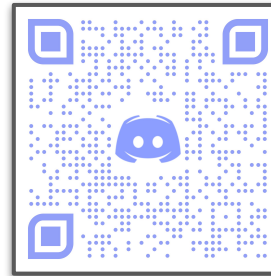
The code and other answers your group submits must be entirely your own work, and you are bound by the University's Student Code. You may consult with other students about the conceptualization of the project and the meaning of the questions, but you may not look at any part of someone else's solution 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.

# Announcements



utahsec



See Discord for  
meeting info!

[utahsec.cs.utah.edu](https://utahsec.cs.utah.edu)

# Announcements

## Resume Workshop!

Join ACM and U Career Success:

- Develop skills needed to build a resume as a student in computing
- Connect with others looking for industry opportunities and advice from career professionals



Thurs, Oct 17, 5pm

MEB 3147

Please RSVP  
for headcount



# Questions?



# Last time on CS 4440...

Malware  
Today's Malware "Zoo"  
Malware Detection and Prevention



# Malware: Malicious Software

- **Definition: ???**



# Malware: Malicious Software

- **Definition:** software (more generally, a set of instructions) that runs on a computer it **doesn't have access to** and/or does **something nefarious**
- **Goals of Malware: ???**



# Malware: Malicious Software

- **Definition:** software (more generally, a set of instructions) that runs on a computer it **doesn't have access to** and/or does **something nefarious**
- **Goals of Malware:**
  - Steal private data
  - Display ads, send spam
  - Damage local machine
  - Congest a network
  - Attack other systems on the network
  - Commit online fraud
  - Gain, then grant, unauthorized access
  - Up to the attacker(s) really...



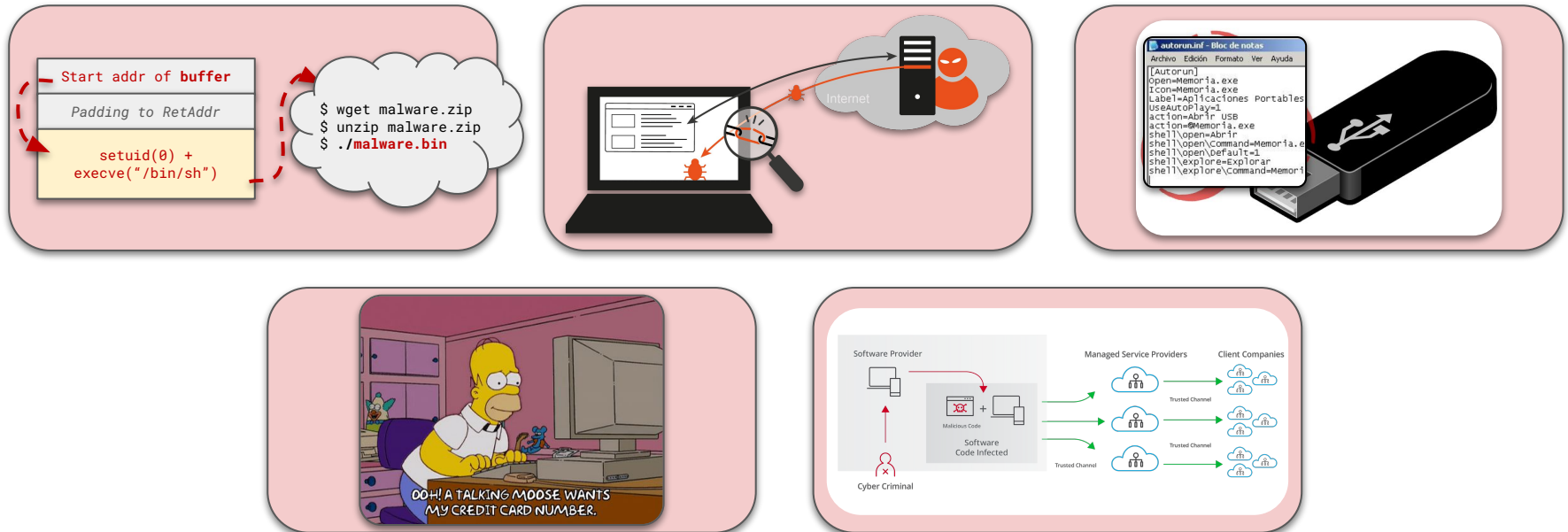
# Malware Infection

- **How** does malicious software get on victim computers in the first place?



# Malware Infection

- **How** does malicious software get on victim computers in the first place?



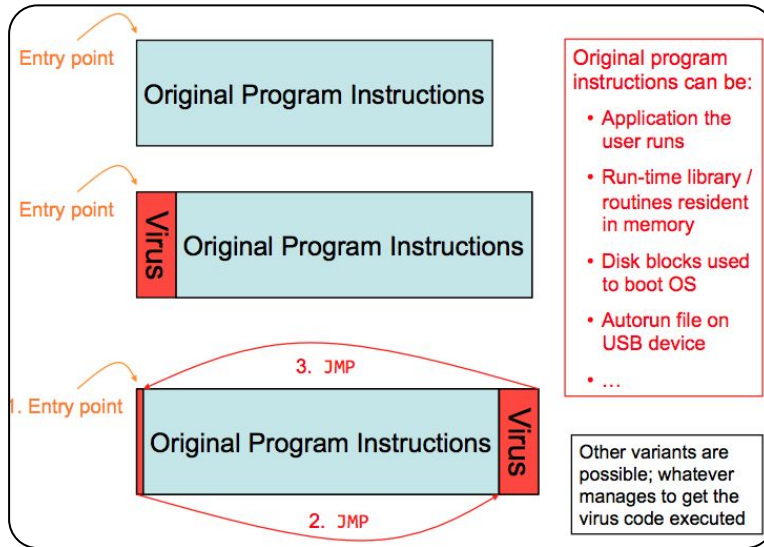
# Summary: Major Malware Types

- **Virus**
  - ???










# Summary: Major Malware Types

## ■ Virus

- Self-replicating software that **infects other programs**, mutates itself to avoid detection



### Different types of computer viruses

-  Polymorphic virus
-  Multipartite virus
-  Boot sector virus
-  File infector
-  Overwrite virus
-  Browser hijacker
-  Web scripting virus
-  Network virus
-  Macro virus

# Summary: Major Malware Types

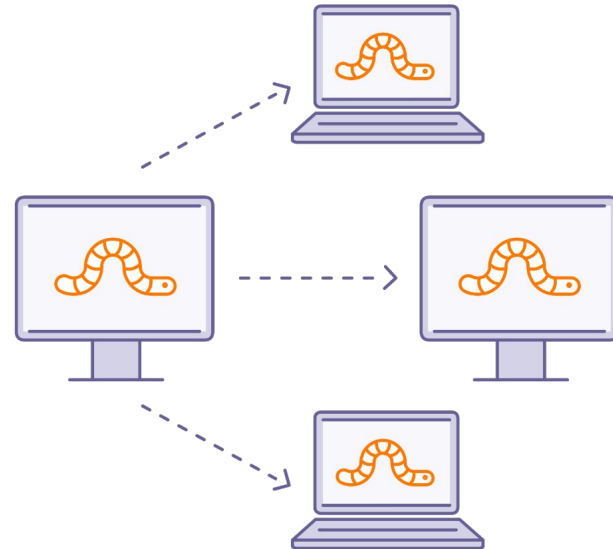
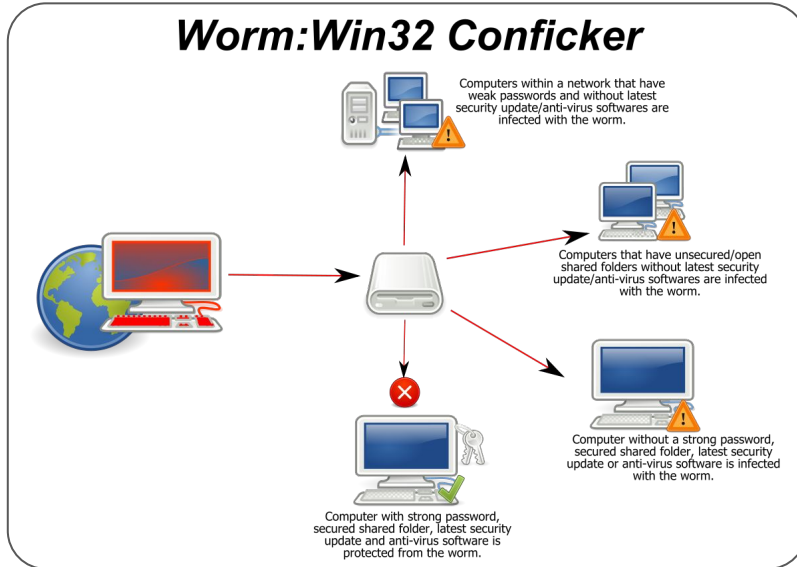
- **Worm**
  - ???



# Summary: Major Malware Types

## ■ Worm

- Self-replicating software that **spreads** over networks to infect programs on other systems



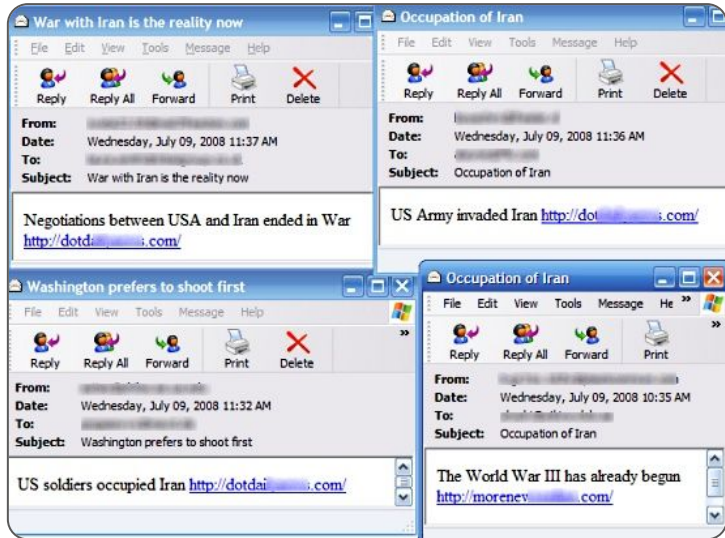
# Summary: Major Malware Types

- **Trojans**
  - ???

# Summary: Major Malware Types

## ■ Trojans

- Appears to perform desirable function, but does something **malicious behind the scenes**



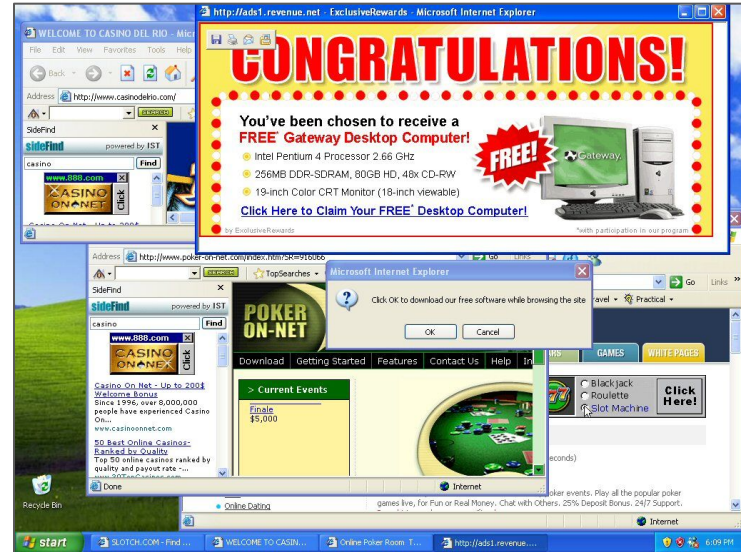
# Summary: Major Malware Types

- **Adware**
  - ???

# Summary: Major Malware Types

## ■ Adware

- Software that incessantly **displays advertisements**; often bundled with other malware



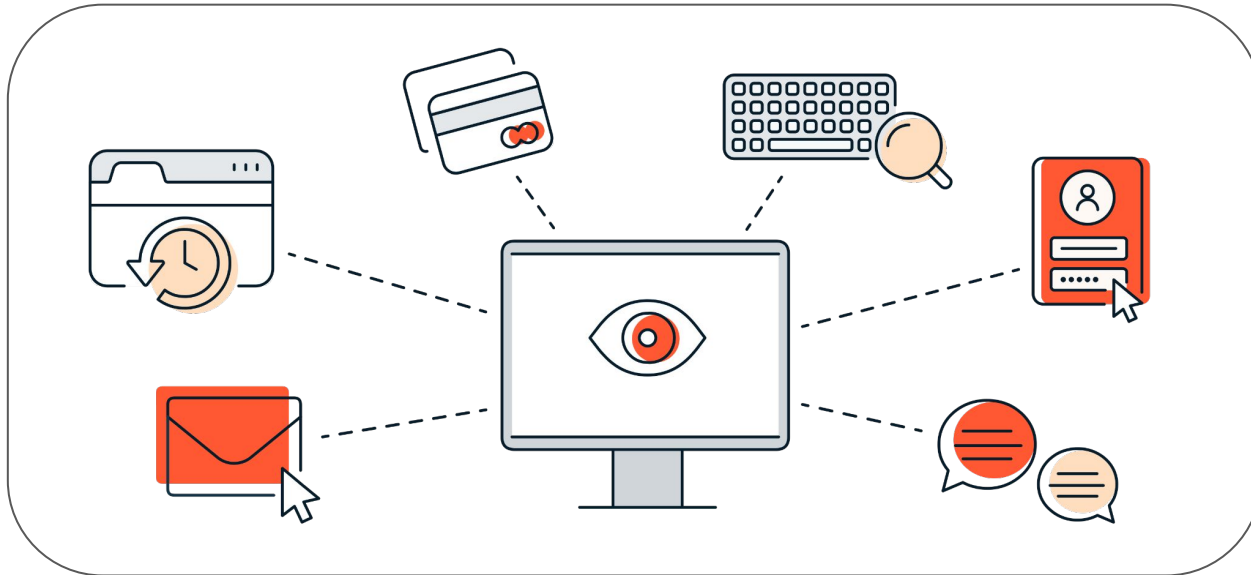
# Summary: Major Malware Types

- **Spyware**
  - ???

# Summary: Major Malware Types

## ■ Spyware

- Software that tracks, collects, and exfiltrates **sensitive user information**



# Summary: Major Malware Types

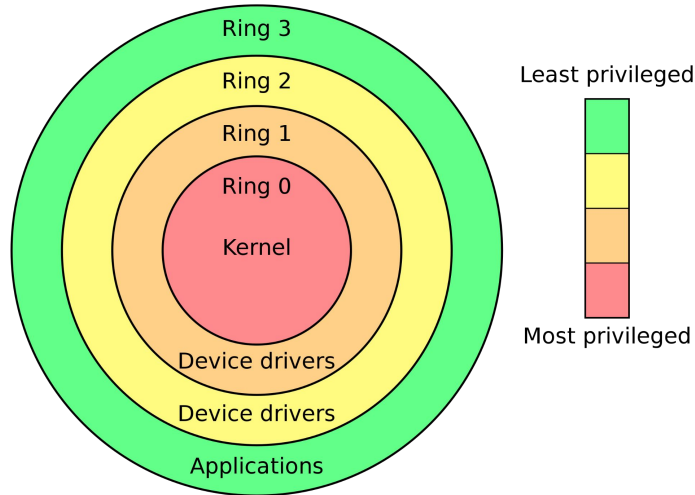
- **Rootkit**
  - ???



# Summary: Major Malware Types

## ■ Rootkit

- Malware that uses stealth to achieve **persistent, privileged control** over a victim machine



GMER 1.0.14.14536

Type	Name	Value
Device	\Driver\BTHUSB \Device\00000097	bthport.sys (Bluetooth Bus Driver/Micr...
Device	\Driver\BTHUSB \Device\00000099	bthport.sys (Bluetooth Bus Driver/Micr...
AttachedD...	\FileSystem\Fastfat \Fat	fltMgr.sys (Microsoft Filesystem Filter ...
Library	c:\windows\system32\z [*** hidden ***] @ C:\WINDOWS\system32\services.exe [680]	0x6A300000
Library	c:\windows\system32\z [*** hidden ***] @ C:\WINDOWS\Explorer.EXE [1568]	0x6A300000
Reg	HKLM\SYSTEM\CurrentControlSet\Services\BTHPORT\Parameters\Keys\005056fdd03	
Reg	HKLM\SYSTEM\CurrentControlSet\Services\BTHPORT\Parameters\Keys\040cce23b9f5	
Reg	HKLM\SYSTEM\CurrentControlSet\Services\BTHPORT\Parameters\Keys\14109fd4869e	
Reg	HKLM\SYSTEM\ControlSet003\Services\BTHPORT\Parameters\Keys\005056fdd03	
Reg	HKLM\SYSTEM\ControlSet003\Services\BTHPORT\Parameters\Keys\040cce23b9f5	
Reg	HKLM\SYSTEM\ControlSet003\Services\BTHPORT\Parameters\Keys\14109fd4869e	

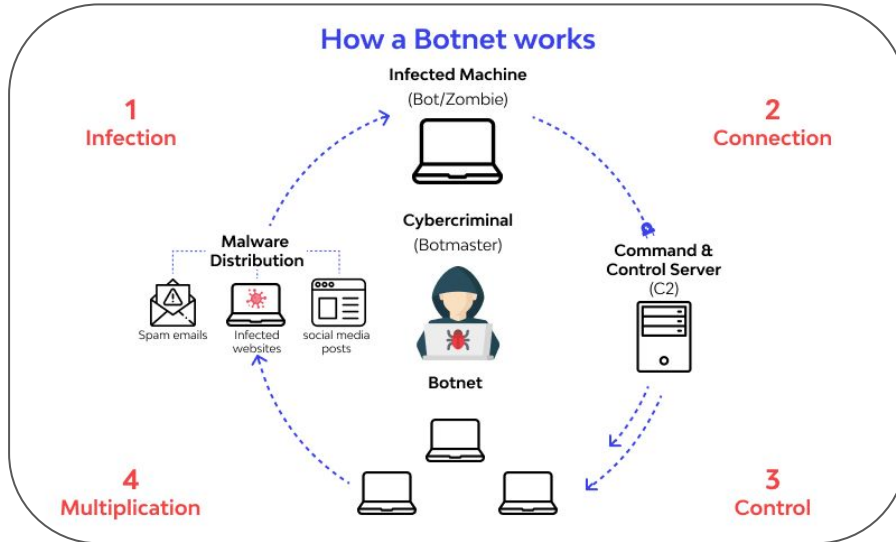
# Summary: Major Malware Types

- **Botnet**
  - ???

# Summary: Major Malware Types

## ■ Botnet

- A network of **compromised “zombie” or “bot” computers** that do a botmaster’s bidding



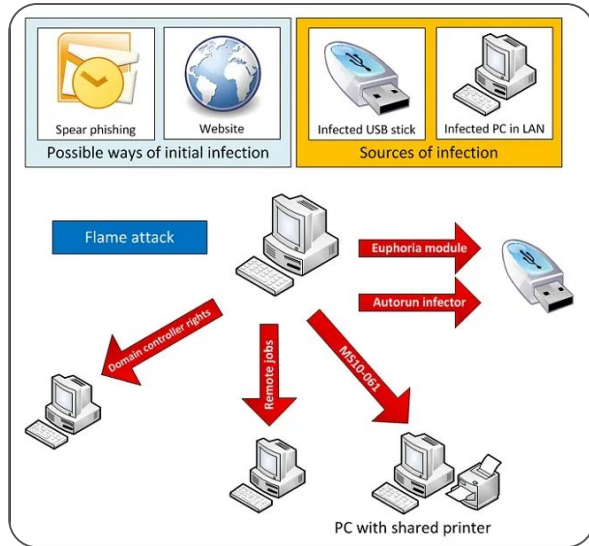
# Summary: Major Malware Types

- **Advanced Persistent Threat**
  - ???

# Summary: Major Malware Types

## ■ Advanced Persistent Threat

- Combined threats, usually targeting a specific entity; **extremely sophisticated and stealthy**



# Detection

- **Anti-virus software**
  - Software for detecting, eliminate malware
  - E.g., Malwarebytes, Avast, McAfee, Symantec
- **Signature-based anti-virus:**
  - ???
- **Heuristic-based anti-virus:**
  - ???



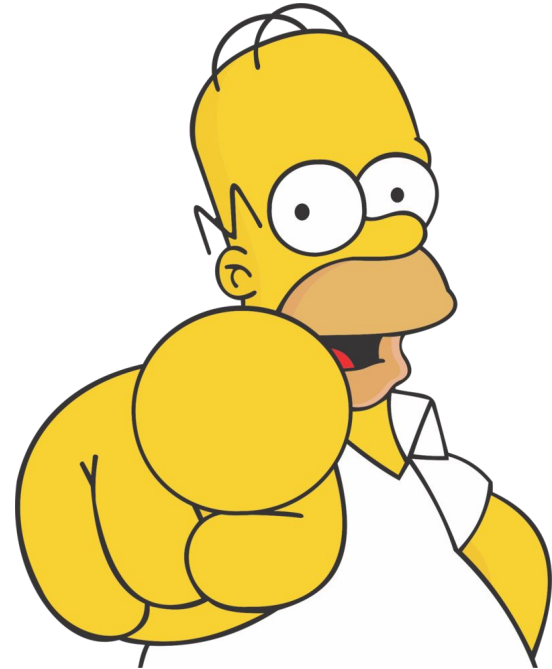
# Detection

- **Anti-virus software**
  - Software for detecting, eliminate malware
  - E.g., Malwarebytes, Avast, McAfee, Symantec
- **Signature-based anti-virus:**
  - Track identifying strings (like a fingerprint)
  - Difficult against mutating viruses
- **Heuristic-based anti-virus:**
  - Analyze program behavior, identify unusual patterns
  - E.g. network access, file deletion, modify boot sector



# Other Defenses

- **Tripwired Hashes**
  - Keep hash of known system files
  - Periodically re-hash and check
    - **If hash changes, file tampered**
- Be a **security-conscious** citizen
  - Strong passwords, 2-factor authentication
  - Do not access suspicious files or websites
    - **Use your intuition: if it seems too good to be true, it probably is!**
  - Keep software updated and use anti-virus
  - **Teach others!**





# Questions?



# This time on CS 4440...

The Web  
HTML & HTTP  
HTTP Cookies  
JavaScript  
SQL

# The Web

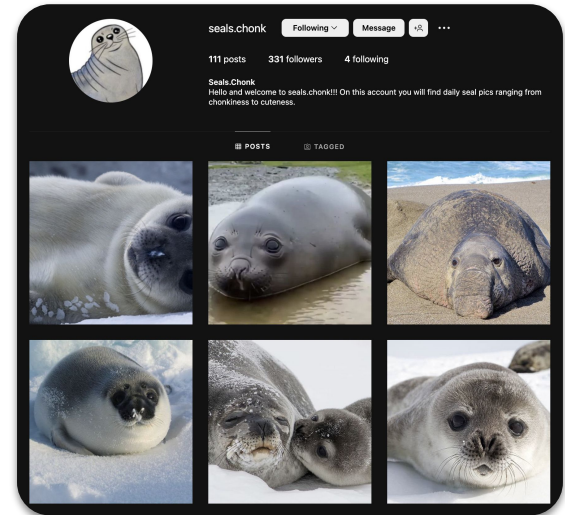
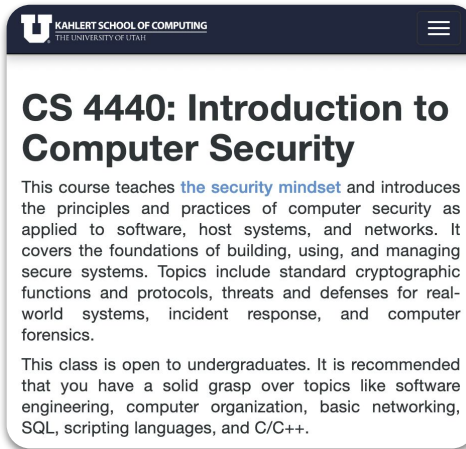
# What is the Web?

- **What is it?**

# What is the Web?

## ■ What is it?

- A venue for me to ridicule Broncos fans
- A place to view (and share) pictures of seals
- The location where I host the CS 4440 website

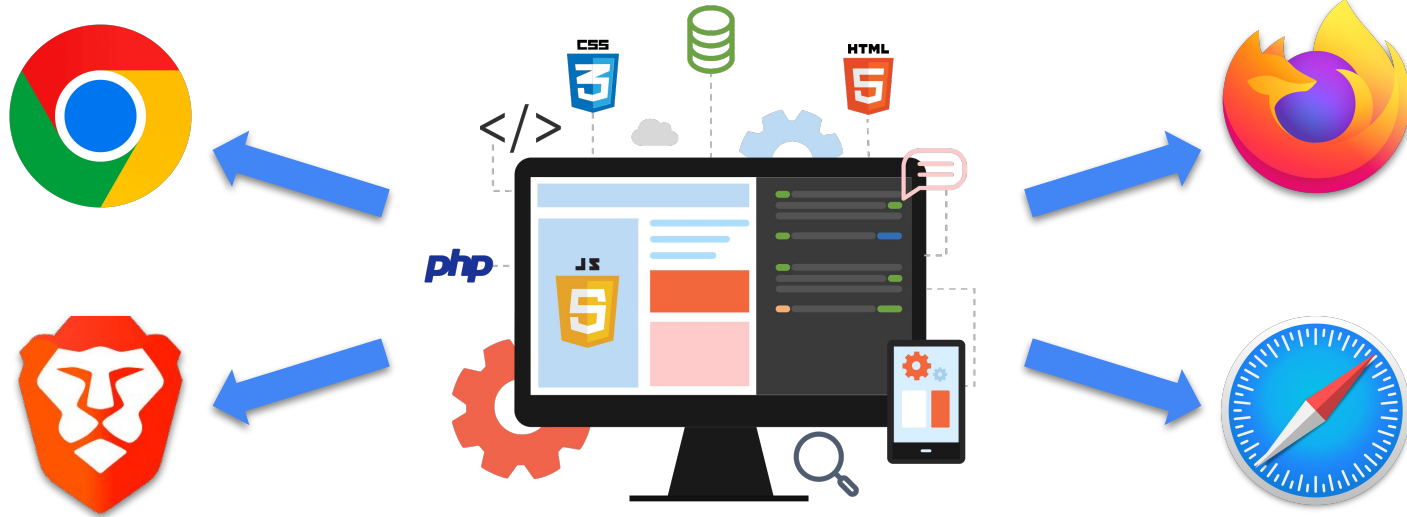


# What is the Web?

- What **really** is it?

# What is the Web?

- What **really** is it?
  - A platform for deploying applications, **portably** and **securely**



# A Historical Perspective

- The web is an example of **bolt-on security**
  - Originally invented to allow physicists to share their research papers
  - Only **textual web pages**, with links to other pages;  
**no security model**





# A Historical Perspective

- The web is an example of **bolt-on security**
  - Originally invented to allow physicists to share their research papers
  - Only **textual web pages**, with links to other pages;  
**no security model**
- Then we added **embedded media** (e.g., images)
  - **Crucial decision:** a page can embed images loaded from another web server
  - Then, **Javascript**, **dynamic HTML**, **AJAX**, **CSS**, **frames**, **audio**, **video**, and **others!**

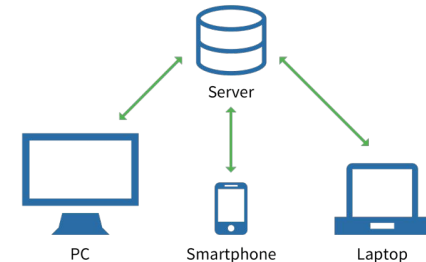


# A Historical Perspective

- The web is an example of **bolt-on security**
  - Originally invented to allow physicists to share their research papers
  - Only **textual web pages**, with links to other pages;  
**no security model**
- Then we added **embedded media** (e.g., images)
  - **Crucial decision:** a page can embed images loaded from another web server
  - Then, **Javascript**, **dynamic HTML**, **AJAX**, **CSS**, **frames**, **audio**, **video**, and **others!**
- Today, a website is a **distributed application**



Client-Server Model



# Web Security: Two Tales

- **Web Browser** (the **client** side)
  - Attacks targeting **browser security weaknesses** cause:
    - Malware installation (e.g., keyloggers, rootkits)
    - Theft of sensitive data (e.g., files, passwords)



# Web Security: Two Tales

- **Web Browser** (the **client** side)
  - Attacks targeting **browser security weaknesses** cause:
    - Malware installation (e.g., keyloggers, rootkits)
    - Theft of sensitive data (e.g., files, passwords)
- **Web Application** (the **server** side)
  - Runs on **the site** (e.g., e-commerce, blogs)
  - Written in PHP, ASP, JSP, Ruby, etc.
  - Many attacks:
    - Cross-site **Scripting**
    - Cross-site **Request Forgery**
    - **SQL Injection**



# Questions?



# HTML and HTTP

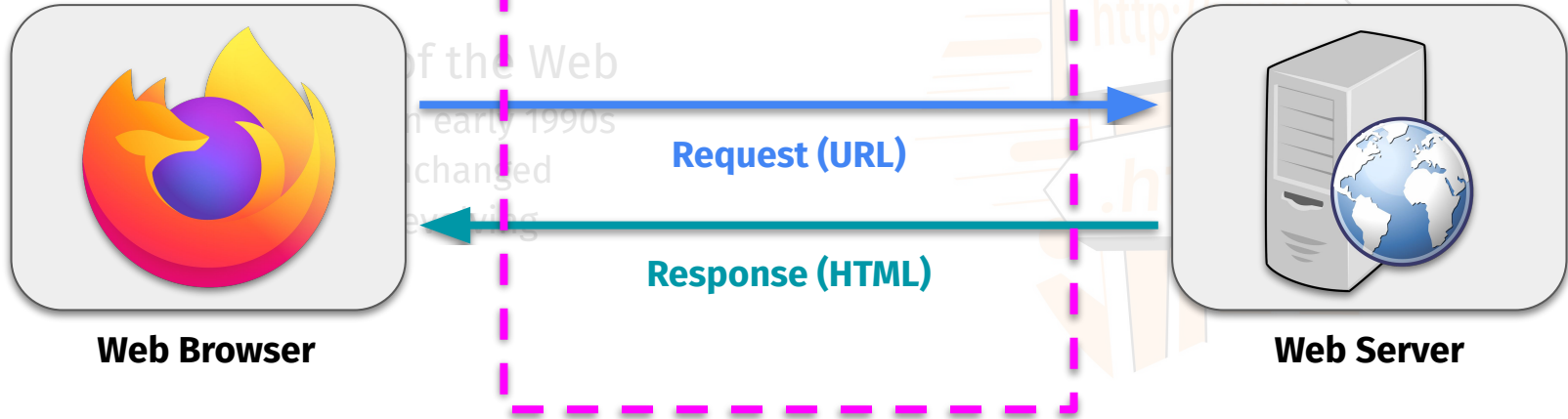
# HTML and HTTP

- What are **HTML** and **HTTP**?
  - **HTML** = how we **represent** content
  - **HTTP** = how we **transfer** content
- Key components of the Web
  - Both developed in early 1990s
  - HTTP is mostly unchanged
  - HTML still evolving (albeit slowly)



# HTML and HTTP

- What are **HTML** and **HTTP**?
  - HTML** = how we **represent** content
  - HTTP** = how we **transfer** content





# HyperText Markup Language (HTML)

- Describes **content** and **formatting** of web pages
  - Rendered within browser window
- **HTML features**
  - **Static** document description language
  - Links to external pages, images by **reference**
  - User input sent to server via **forms**

```
<form action="home.html">
  First Name:<br>
  <input type="text" name="first_name">
</br>
  Last Name:<br>
  <input type="text" name="last_name">
</br>
  Email:<br>
  <input type="text" name="email">
</br>
  <input type="submit" name="Submit">
</form>
```



First Name:

Last Name:

Email:

# HyperText Markup Language (HTML)

- Describes **content** and **formatting** of web pages
  - Rendered within browser window
- **HTML features**
  - **Static** document description language
  - Links to external pages, images by **reference**
  - User input sent to server via **forms**
- **HTML extensions**
  - Additional media (e.g., PDF, videos) via **plugins**
  - Embedding **programs** in other languages (e.g., **Java**) provides **dynamic content** that can:
    - Interacts with the user
    - Modify the browser user interface
    - Access the client computer environment

```
<form action="home.html">
  First Name:<br>
  <input type="text" name="first_name">
</br>
  Last Name:<br>
  <input type="text" name="last_name">
</br>
  Email:<br>
  <input type="text" name="email">
</br>
  <input type="submit" name="Submit">
</form>
```



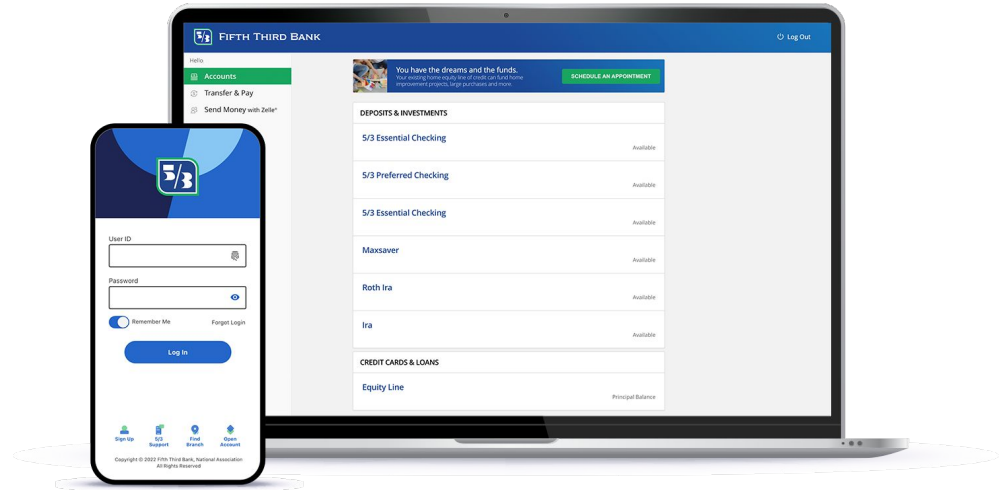
First Name:

Last Name:

Email:

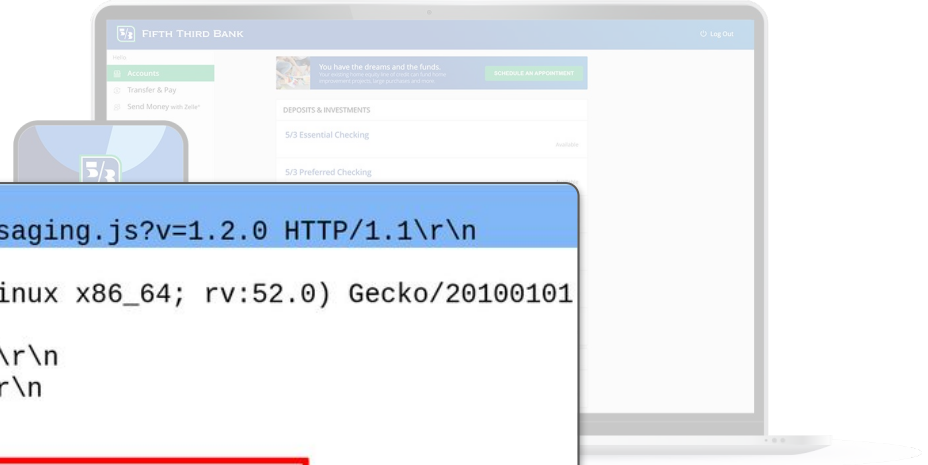
# HyperText Transfer Protocol (HTTP)

- **Protocol for transmitting hypermedia documents (e.g., web pages)**
  - Widely used
  - **Simple**
  - **Unencrypted**



# HyperText Transfer Protocol (HTTP)

- **Protocol for transmitting hypermedia documents** (e.g., web pages)
  - Widely used
  - **Simple**
  - **Unencrypted**

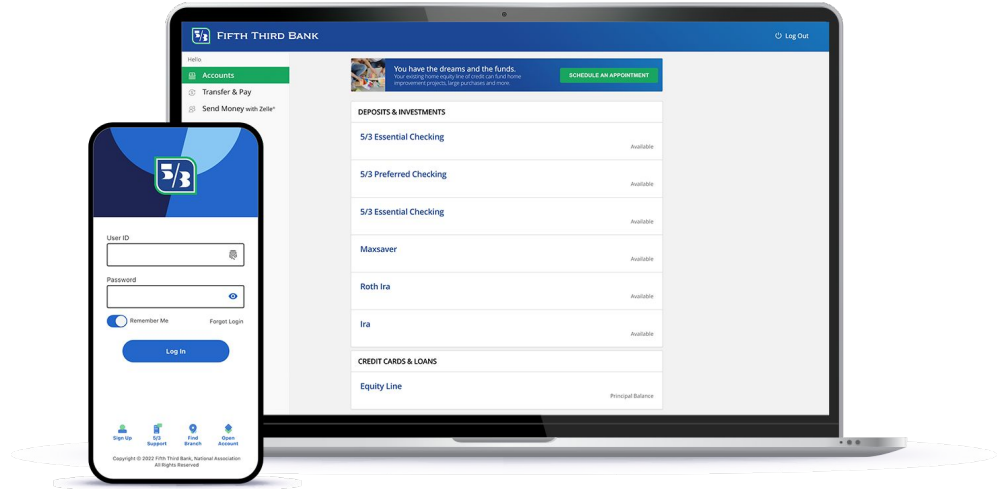


## Hypertext Transfer Protocol

```
GET /libs/qimessaging/1.0/qimessaging.js?v=1.2.0 HTTP/1.1\r\nHost: 10.0.0.6\r\nUser-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101\r\nAccept: */*\r\nAccept-Language: en-US,en;q=0.5\r\nAccept-Encoding: gzip, deflate\r\nReferer: http://10.0.0.6/\r\nConnection: keep-alive\r\nAuthorization: Basic bmFvOmNhcmVzZ2VzLTIwMDE=\r\nCredentials: nao: [REDACTED]\r\n\r\n
```

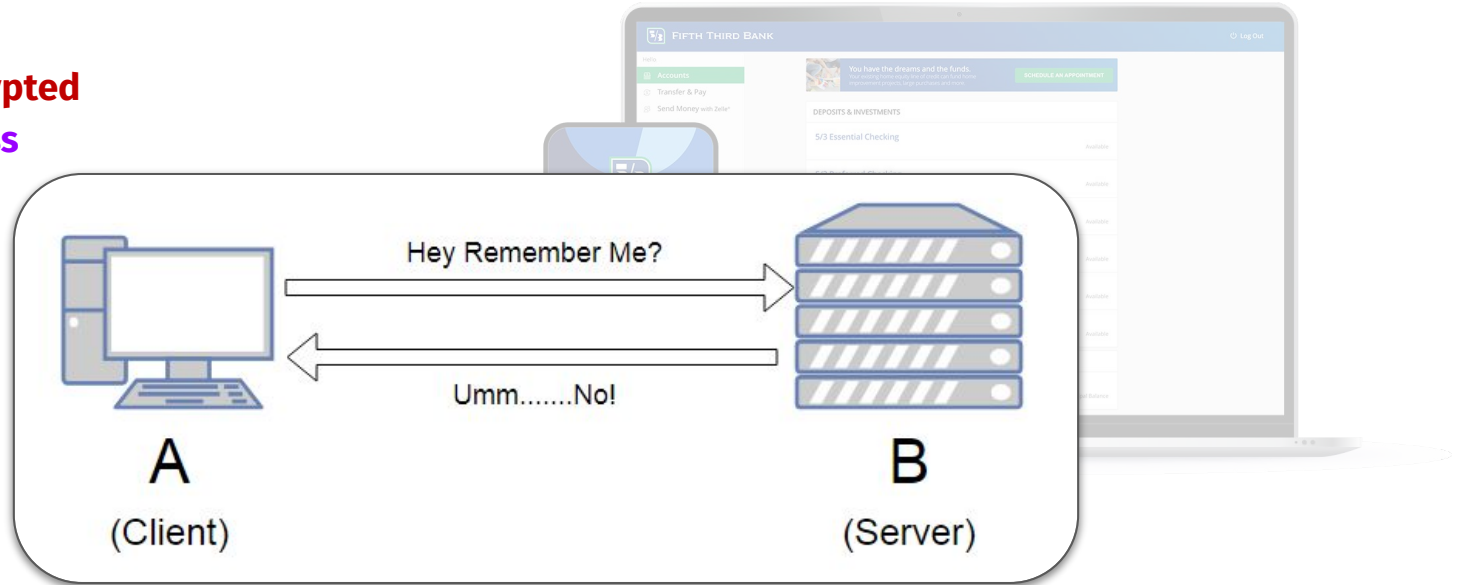
# HyperText Transfer Protocol (HTTP)

- **Protocol for transmitting hypermedia documents (e.g., web pages)**
  - Widely used
  - **Simple**
  - **Unencrypted**
  - **Stateless**



# HyperText Transfer Protocol (HTTP)

- **Protocol for transmitting hypermedia documents** (e.g., web pages)
  - Widely used
  - **Simple**
  - **Unencrypted**
  - **Stateless**



# Uniform Resource Locator (URL)

- **Reference to a web resource** (e.g., a website)
  - Specifies its **location** on a computer network
  - Specifies the mechanism for **retrieving it**
- **Example:** `http://www.cs.utah.edu/class?name=cs4440#homework`
  - **Protocol:** How to **retrieve** the web resource
  - **Path:** Identifies the **specific resource** to access (case **insensitive**)
  - **Query:** Assigns values to specified **parameters** (case **sensitive**)
  - **Fragment:** Location of a **resource subordinate** to another

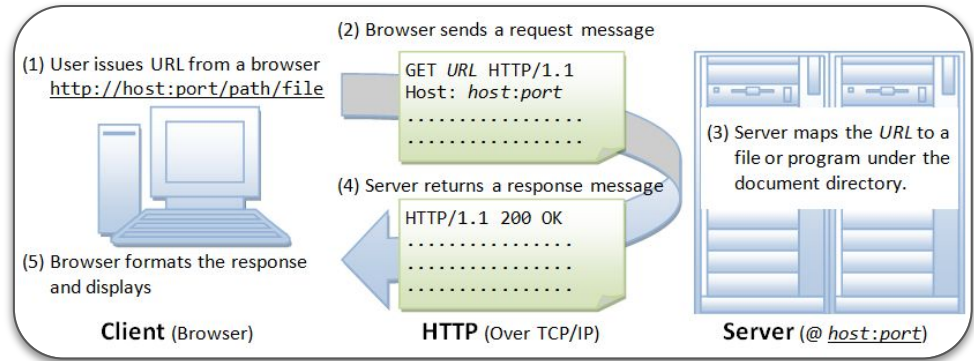
# Uniform Resource Locator (URL)

- **Reference to a web resource** (e.g., a website)
  - Specifies its **location** on a computer network
  - Specifies the mechanism for **retrieving it**
- **Example:** `http://www.cs.utah.edu/class?name=cs4440#homework`
  - **Protocol:** How to **retrieve** the web resource
    - HTTP
  - **Path:** Identifies the **specific resource** to access (case **insensitive**)
    - `www.cs.utah.edu/class`
  - **Query:** Assigns values to specified **parameters** (case **sensitive**)
    - `name=cs4440`
  - **Fragment:** Location of a **resource subordinate** to another
    - `#homework`



# HTTP Requests

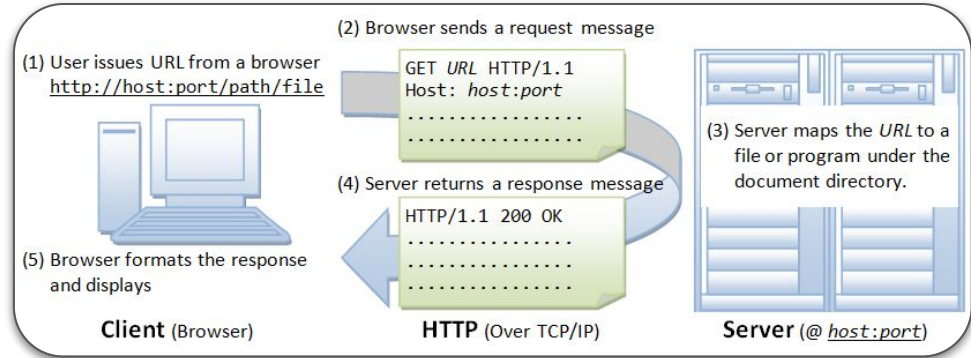
- **Browser (client):**
  1. **Open** connection
  2. Client sends **request**
  3. Server obtains **resource**
  4. Server **responds** (stateless!)
  5. Display and **close** connection



# HTTP Requests

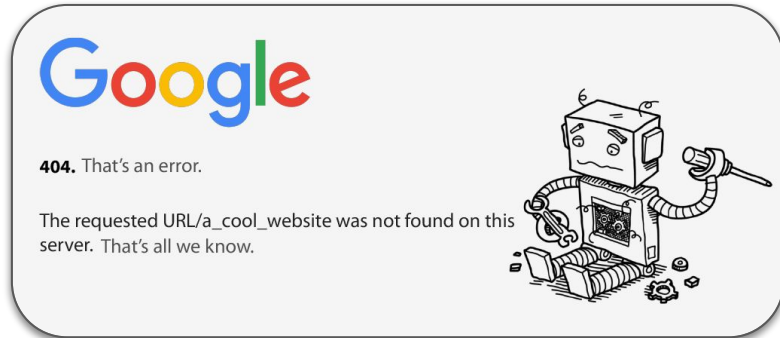
## ■ Browser (client):

1. **Open** connection
2. Client sends **request**
3. Server obtains **resource**
4. Server **responds** (stateless!)
5. Display and **close** connection



## ■ Server Responses:

- “200 OK”
- “304 Document moved”
- “404 Not found”
- “400 Bad request”



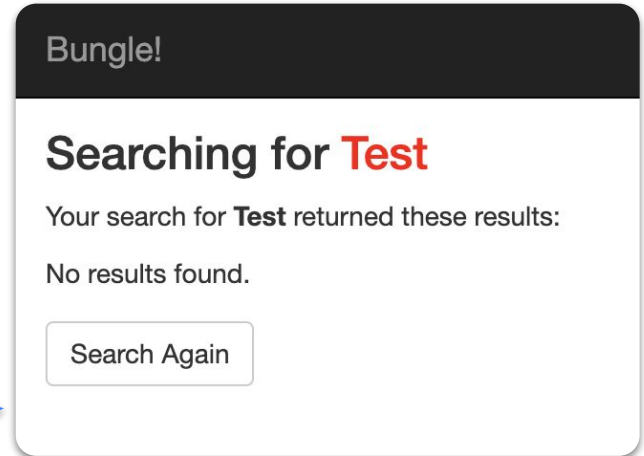
# HTTP Requests

- **Two types** of HTTP requests: **GET** and **POST**
  - **GET requests:** set within the request's **URL**
  
- What does this example request do?

`http://cs4440.eng.utah.edu/project3/search?q=Test`

# HTTP Requests

- **Two types** of HTTP requests: **GET** and **POST**
  - **GET requests:** set within the request's **URL**
- What does this example request do?
  - Sets parameter **q** to value **Test** for interface **search**



<http://cs4440.eng.utah.edu/project3/search?q=Test>

# HTTP Requests

- **Two types** of HTTP requests: **GET** and **POST**
  - **POST requests:** parameters within request **body**
  
- What does this example request do?

```
<form action="http://cs4440.eng.utah.edu/project3/login?" method="POST">  
  <input name="username" value="attacker" type="hidden"/>  
  <input name="password" value="l33th4x" type="hidden"/>  
</form>
```

# HTTP Requests

- Two types of HTTP requests: **GET** and **POST**
  - POST requests:** parameters within request **body**

Logged in as **attacker**. [Log out](#)



- What does this example request do?
  - Sets **username** to value **attacker** (and type **hidden**) for interface **login**
  - Sets **password** to value **133th4x** (and type **hidden**) for interface **login**

```
<form action="http://cs4440.eng.utah.edu/project3/login?" method="POST">  
  <input name="username" value="attacker" type="hidden"/>  
  <input name="password" value="133th4x" type="hidden"/>  
</form>
```

# Questions?

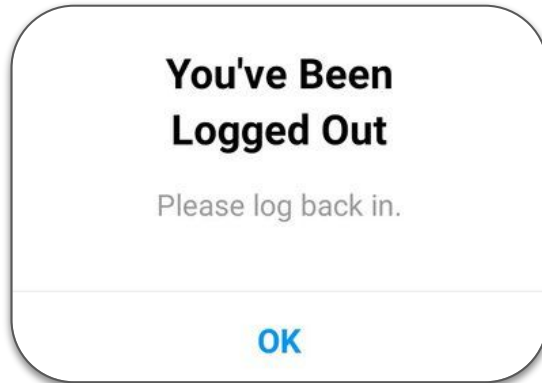


# Cookies



# Supporting Stateful Connections

- Stateless connection is impractical—why?



**Login**

uNID: *(e.g. u8675309)*

[Forgot your uNID?](#)

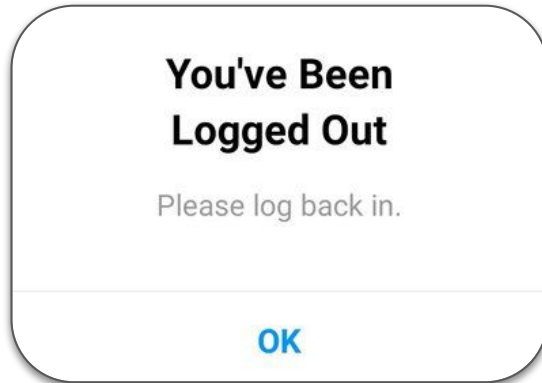
Password:

[Forgot your password?](#)

LOGIN

# Supporting Stateful Connections

- **Stateless connection is impractical—why?**
  - **Performance:** cost of **re-transmitting** redundant info
  - **Convenience:** user must perform same **redundant tasks**



**Login**

uNID: (e.g. u8675309)

[Forgot your uNID?](#)

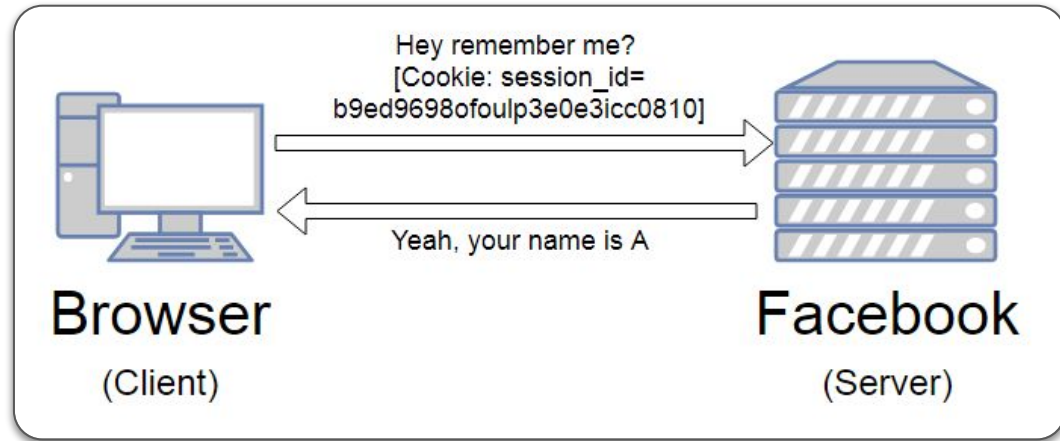
Password:

[Forgot your password?](#)

LOGIN

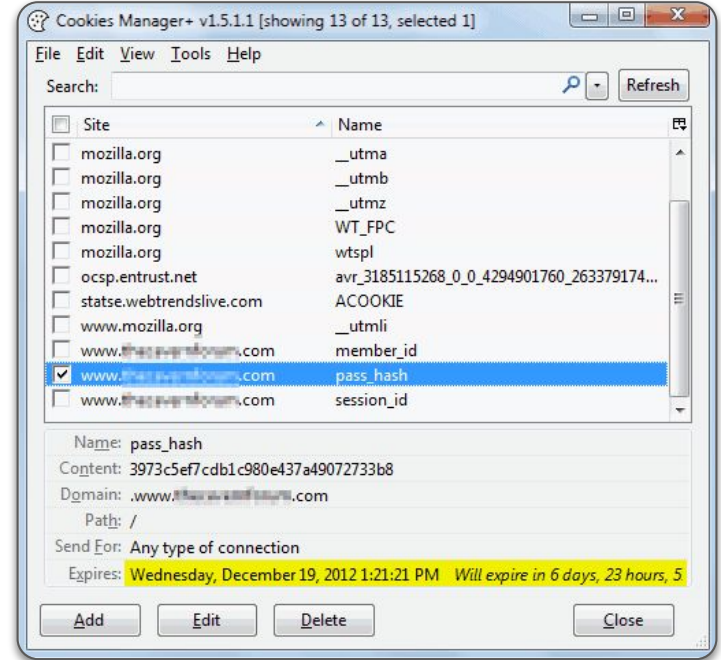
# HTTP Cookies

- **Small chunks of info stored on a computer associated with a specific server**
  - When you access a website, it might store information as a cookie
  - Every time you visit that server, the cookie is re-sent to the server
  - Effectively used to **hold state information over multiple sessions**



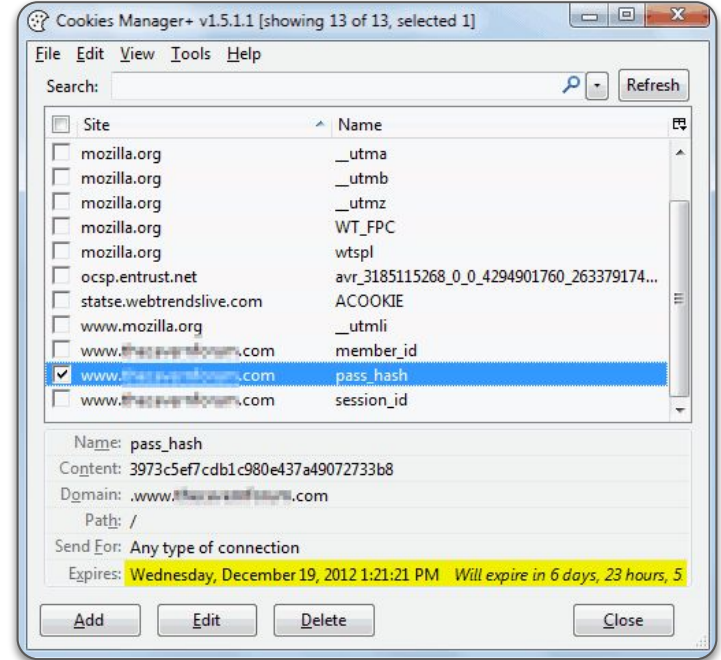
# HTTP Cookies

- **Cookies expire!**
  - Date is chosen by server (e.g., [January 1st, 2036](#))
  - Thus, any cookies will **stick around for a while!**



# HTTP Cookies

- **Cookies expire!**
  - Date is chosen by server (e.g., [January 1st, 2036](#))
  - Thus, any cookies will **stick around for a while!**
- **Every large website** that you use today makes use of cookies in some form
  - **“Necessary” cookies**
    - Core functionality like security, accessibility
  - **“Analytics” cookies**
    - Used to collect data about your browsing, or to display you targeted advertisements



# HTTP Cookies

- **Cookies can hold any type of information—including sensitive information**
  - Passwords, credit card information, social security numbers, etc.
  - Session cookies, non-persistent cookies, persistent cookies

## CWE-315: Cleartext Storage of Sensitive Information in a Cookie

**Weakness ID: 315**

**Abstraction:** Variant

**Structure:** Simple

*View customized information:*

Conceptual

Operational

Mapping-Friendly

Complete

### ▼ Description

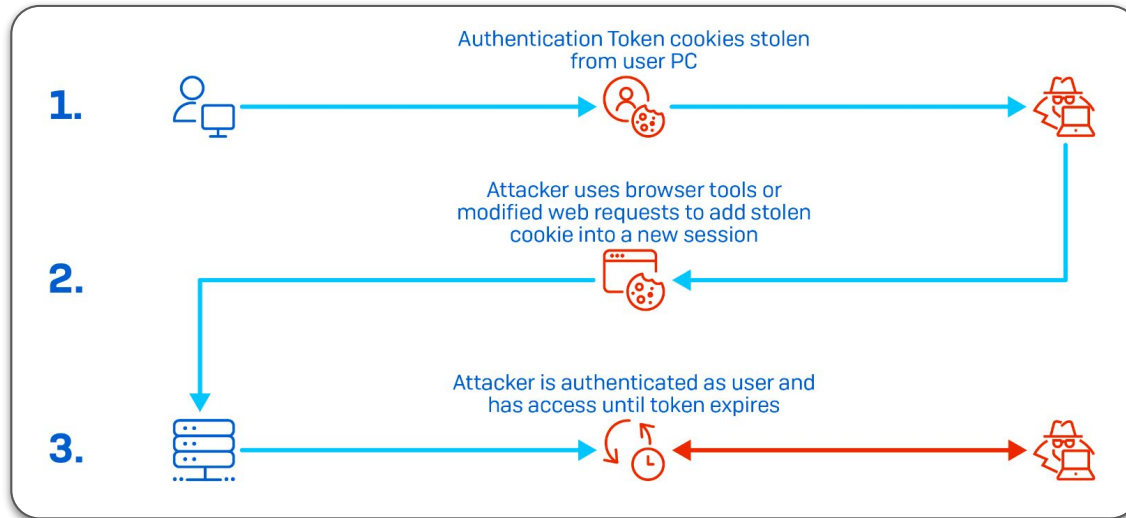
The product stores sensitive information in cleartext in a cookie.

### ▼ Extended Description

Attackers can use widely-available tools to view the cookie and read the sensitive information. Even if the information is encoded in a way that is not human-readable, certain techniques could determine which encoding is being used, then decode the information.

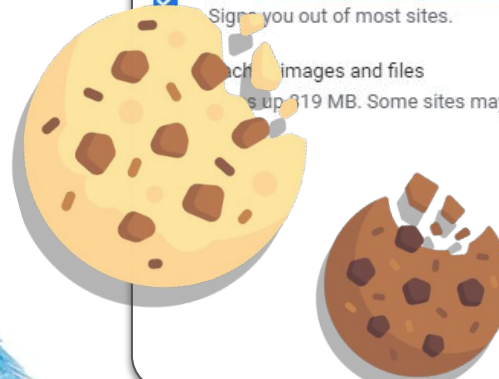
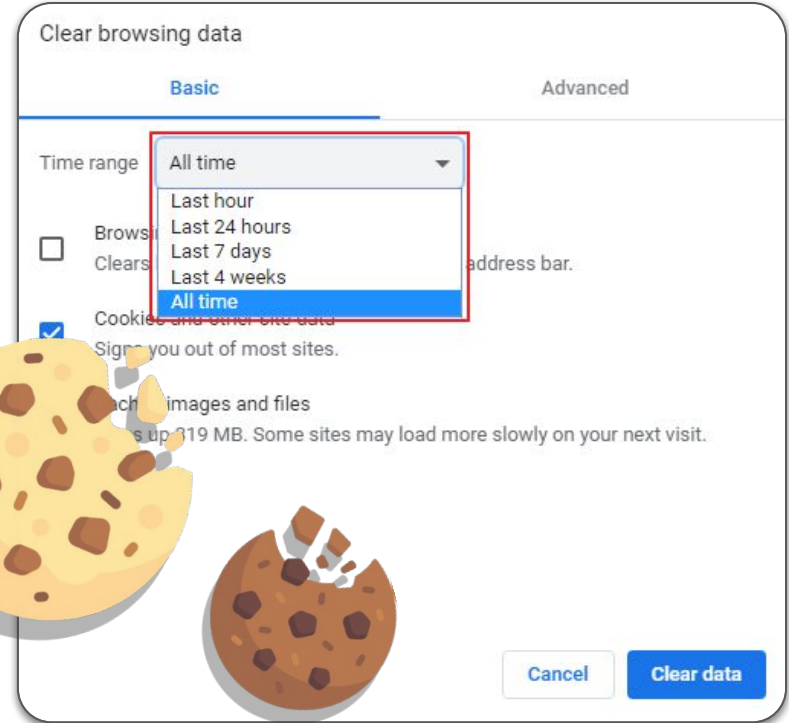
# HTTP Cookies

- **Cookies are stored on your computer and can be controlled or manipulated**
  - Many sites require that you enable cookies to access the site's full capabilities
  - Their storage on your computer naturally **lends itself to cookie exploitation**



# HTTP Cookies

- You can (and probably should) **clear your cookies regularly**
  - Most browsers nowadays have mechanisms to **disable cookies**
  - You can also choose to **accept** or **exclude** cookies from certain sites





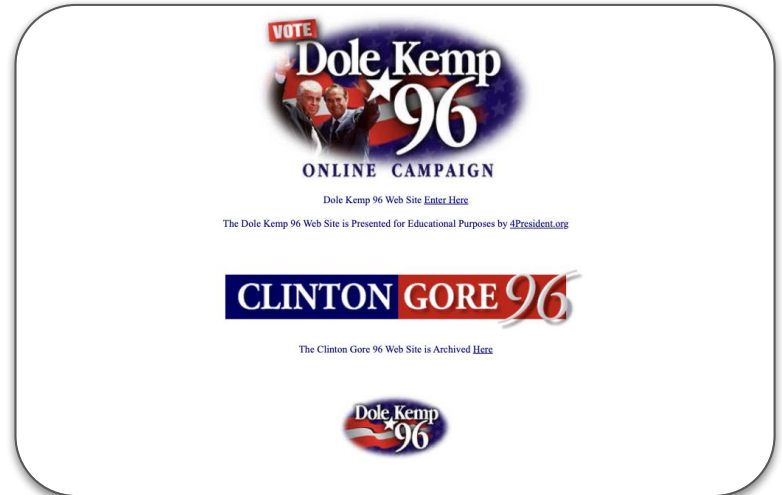
# Questions?



# JavaScript

# From Web 1.0 to Web 2.0

- **Recall that HTML is a static language**
  - Pages are rendered only **once**
  - Ideal for **non-interactive** content
    - E.g., “About Us”, “Contact Us”, etc.



# From Web 1.0 to Web 2.0

- Recall that HTML is a **static language**
  - Pages are rendered only **once**
  - Ideal for **non-interactive** content
    - E.g., “About Us”, “Contact Us”, etc.
- Since **Web 1.0**, we’ve evolved to now express web pages as **programs**
  - Enables **richer**, more **interactive** content



# From Web 1.0 to Web 2.0

- Recall that HTML is a **static language**
  - Pages are rendered only **once**
  - Ideal for **non-interactive** content
    - E.g., “About Us”, “Contact Us”, etc.
- Since **Web 1.0**, we’ve evolved to now express web pages as **programs**
  - Enables **richer**, more **interactive** content
  - E.g., the **JavaScript** language

```
1
2 <html>
3 <head>
4 <script>
5   function HelloWorld() {
6     alert("Hello World");
7   }
8 </script>
9 </head>
10 <body onload="HelloWorld()">
11
12 </body>
13 </html>
```

# JavaScript

- **A powerful, popular web programming language**
  - Scripts embedded in web pages returned by web server
  - Scripts **executed** by browser (client-side scripting). Can:
    - **Alter contents** of a web page
    - **Track events** (mouse clicks, motion, keystrokes)
    - **Read/set cookies**
    - **Issue web requests** and read replies
  
- **Note:** despite the name, has *nothing* to do with Java!

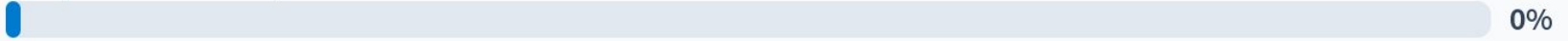


## Familiarity with HTML and JavaScript?

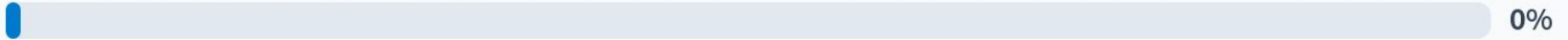
Only some HTML



Only some JavaScript



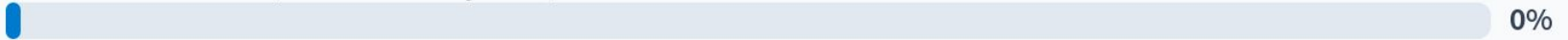
Some of both HTML and JavaScript



Lots of both HTML and JavaScript



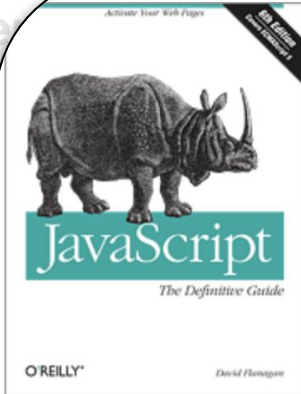
None of the above (which is totally fine!)



# JavaScript

## A power

- Sc
- Sc



## JavaScript: The Definitive Guide, 6th Edition

By [David Flanagan](#)

**Publisher:** [O'Reilly Media](#)

**Release Date:** May 2011

**Pages:** 1096

Since 1996, *JavaScript: The Definitive Guide* has been the bible for JavaScript programmers. It is a comprehensive reference to the core language and the client-side JavaScript API.

The 6th edition covers HTML5 and the latest JavaScript features. It has been completely rewritten and updated to include the latest developments in the language of the Web. The book is a best web development practice guide for experienced programmers who want to master it.

"A must-have reference for expert JavaScript programmers...well-organized and detailed."  
—Brendan Eich, creator of JavaScript, CTO of Mozilla

"I made a career of what I learned from *JavaScript: The Definitive Guide*."  
—Andrew Hedges, Tapulous



## Note: c



# JavaScript

## A power

- Sc
- Sc



## JavaScript: The Good Parts

By [Douglas Crockford](#)

**Publisher:** Yahoo Press

**Release Date:** December 2008

**Pages:** 172



Most programming languages contain good and bad parts, but JavaScript was developed and released as a subset of JavaScript that's truly extensible and efficient.

Considered *the* JavaScript explanation in the development of good ideas that make JavaScript an outstanding language with loose typing, dynamic objects, and an expressive object literal syntax. It also contains downright awful ideas, like a programming model based on

—Andrew Hedges, Tapulous

# JavaScript

- A power

- Sc
- Sc

For **Project 3**, you'll use just a **tiny subset** of this!



- Note: C

—Andrew Hedges, Tapulous

# CS 4440 Wiki: JavaScript Cheat Sheet

## CS 4440 Wiki: JavaScript Cheat Sheet

Below is an abridged cheat sheet of JavaScript fundamentals relevant to Project 3.

**This page is by no means comprehensive—we encourage you to bookmark and familiarize yourself with one of the many in-depth JavaScript tutorials on the web.** Some great examples are:

- [The Official JavaScript Docs](#)
- [HTMLCheetSheet's JS Cheat Sheet](#)
- [W3 Schools' JavaScript Introduction](#)

### Executing JavaScript Code

In Project 3, you'll work with three fundamental ways to execute JavaScript code: **on-page scripts** wrapped in HTML code, **functions**, and **event**-driven execution.

#### On-page scripts:

```
<script>
  /* Code to be executed as the parent HTML code is processed. */
</script>
```

#### Functions:

```
function foo(){
  /* Code to be executed when this function is called. */
}
```

#### Table of Contents:

- [Execution](#)
  - [Scripts](#)
  - [Functions](#)
  - [Events](#)
- [Debugging](#)
  - [Alerts](#)
  - [Console](#)
- [Variables](#)
  - [Initialization](#)
  - [Data Typing](#)
- [Strings](#)
  - [Length](#)
  - [Appending](#)
  - [Substrings](#)
  - [Splitting](#)
- [Arrays](#)
  - [Indexing](#)
- [Requests](#)
  - [GET requests](#)
  - [POST requests](#)
- [Access Elements](#)
  - [DOM tree](#)
  - [Cookies](#)

# Embedding JavaScript within HTML

- Code enclosed within `<script>` tags

# Embedding JavaScript within HTML

- Code enclosed within `<script>` tags
- **Defining functions**

```
<script type="text/javascript">  
    function hello() { alert("Hello world!"); }  
</script>
```

# Embedding JavaScript within HTML

- Code enclosed within **<script>** tags

- **Defining functions**

```
<script type="text/javascript">  
    function hello() { alert("Hello world!"); }  
</script>
```

- **Event handlers** embedded in HTML

```

```

# Embedding JavaScript within HTML

- Code enclosed within `<script>` tags

- **Defining functions**

```
<script type="text/javascript">  
    function hello() { alert("Hello world!"); }  
</script>
```

- **Event handlers** embedded in HTML

```

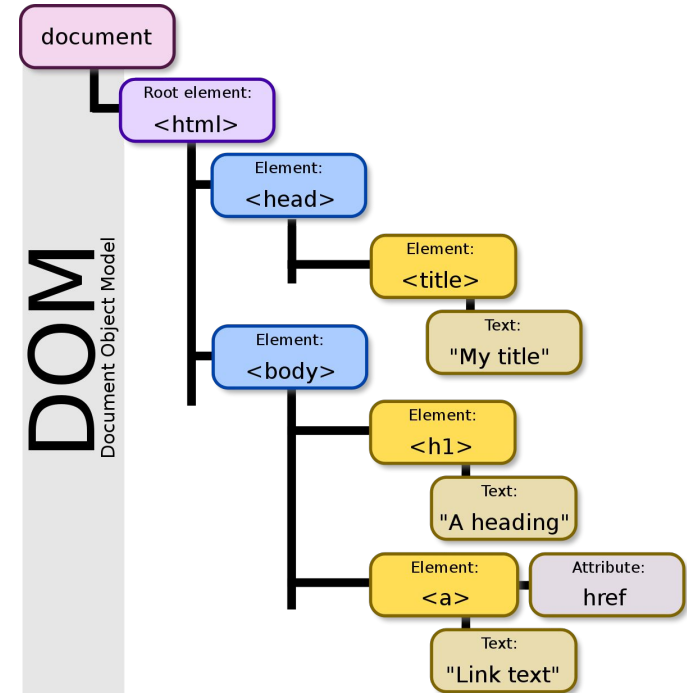
```

- **Built-in functions can change content** of a window: **click-jacking attack**

```
<a onMouseUp="window.open('http://www.evilsite.com')"  
href="http://www.trustedsite.com/">Trust me!?!</a>
```

# Document Object Model (DOM Tree)

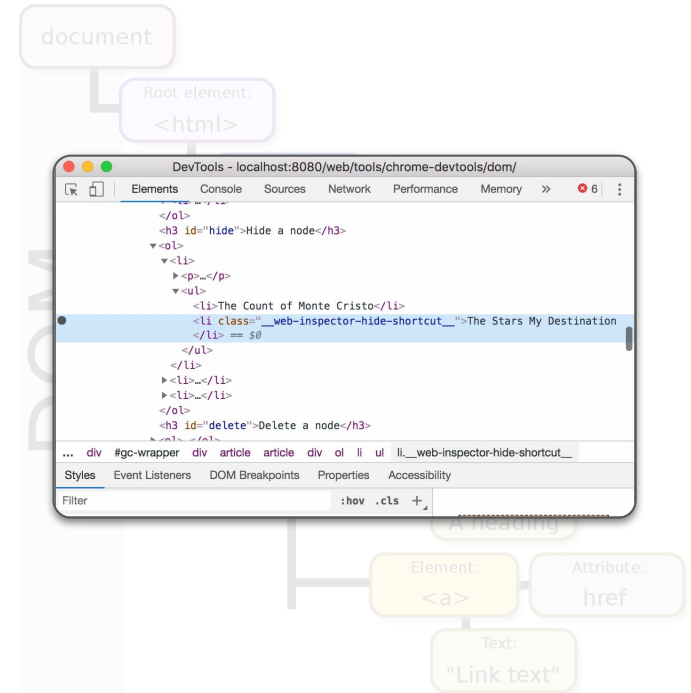
- **Platform- and language-neutral interface**
  - Allows programs and scripts to dynamically **access/update** document **content, structure, style**
- Backbone of modern web browser plugins





# Document Object Model (DOM Tree)

- **Platform- and language-neutral interface**
  - Allows programs and scripts to dynamically **access/update** document **content, structure, style**
- Backbone of modern web browser plugins
- You can access and update the DOM Tree yourself via browser's **web developer tools**
  - You will get familiar with this in **Project 3!**

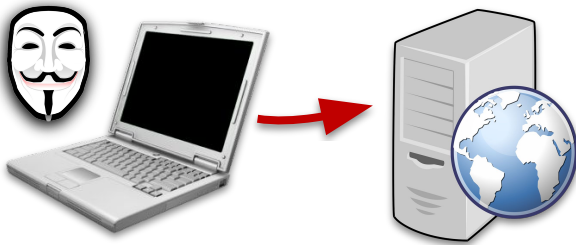


# Questions?

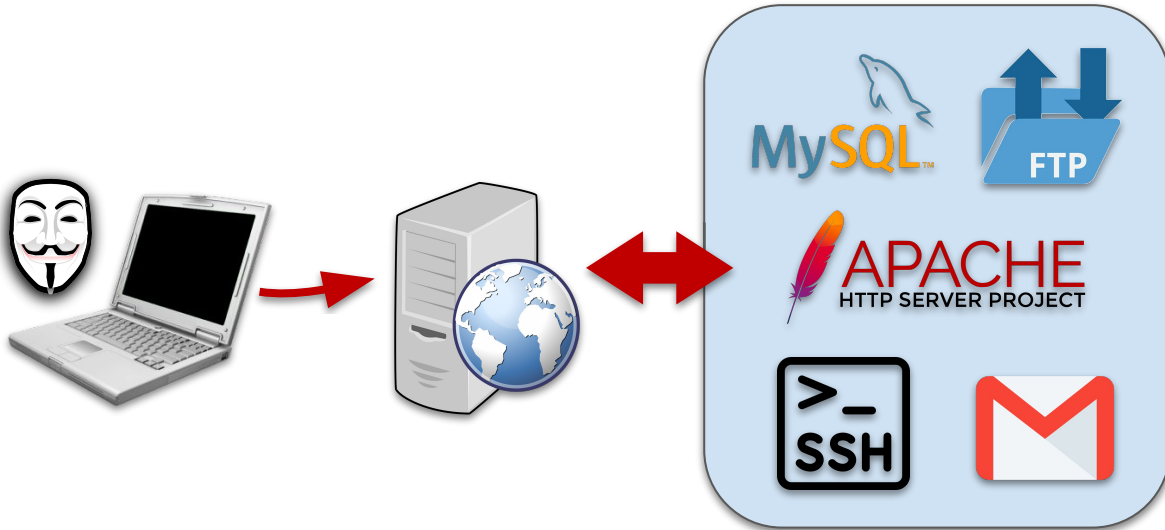


# SQL

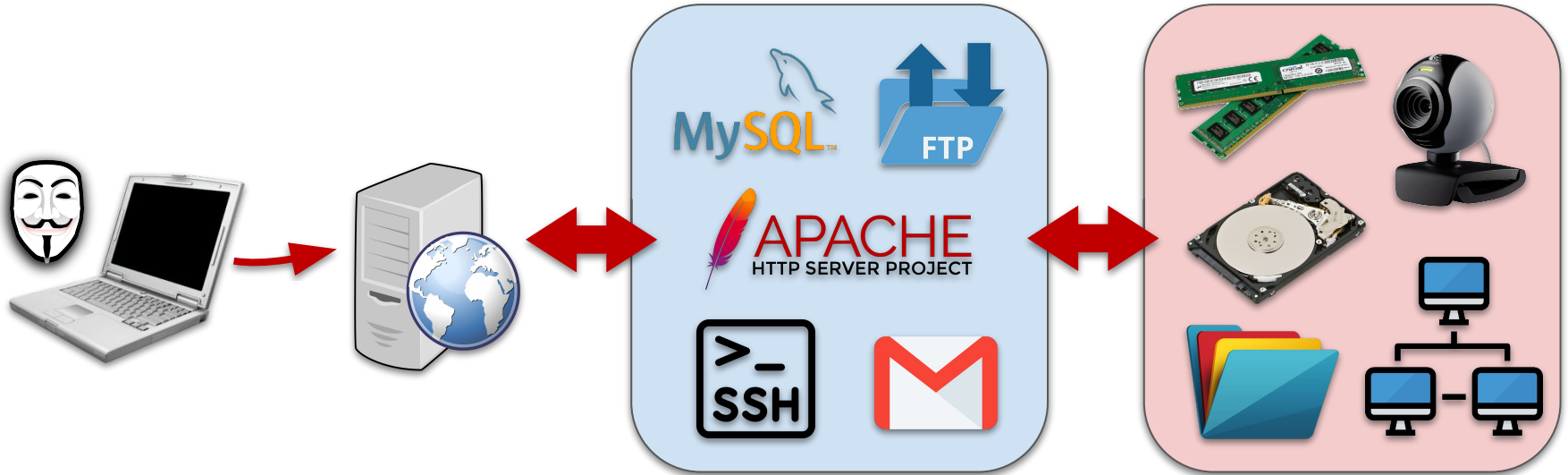
# Server-side vs. Client-side Scripting



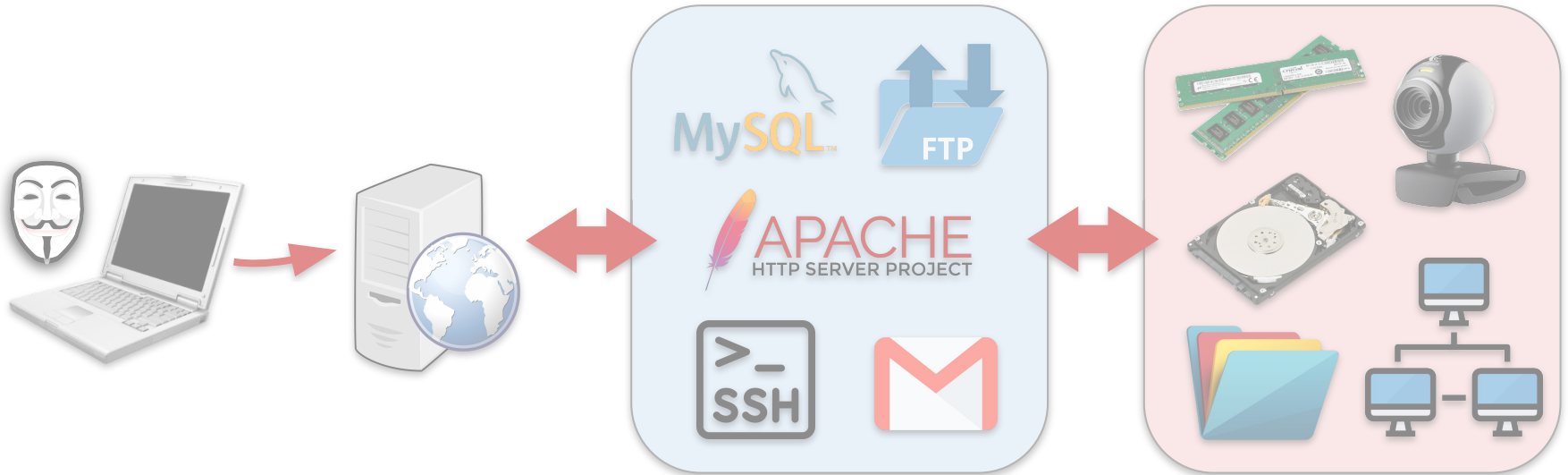
# Server-side vs. Client-side Scripting



# Server-side vs. Client-side Scripting



# Server-side vs. Client-side Scripting



**Servers are a *gateway* for attackers!**

# Server-side vs. Client-side Scripting

Can't we just **restrict all scripting** to be **exclusively on the client-side**?

Servers are a **gateway** for attackers!



# Server-side vs. Client-side Scripting

Can't we just **restrict all scripting** to be **exclusively on the client-side**?

The **client** would need to have **all server data stored locally...**

Servers are a **gateway** for attackers!

# Server-side vs. Client-side Scripting

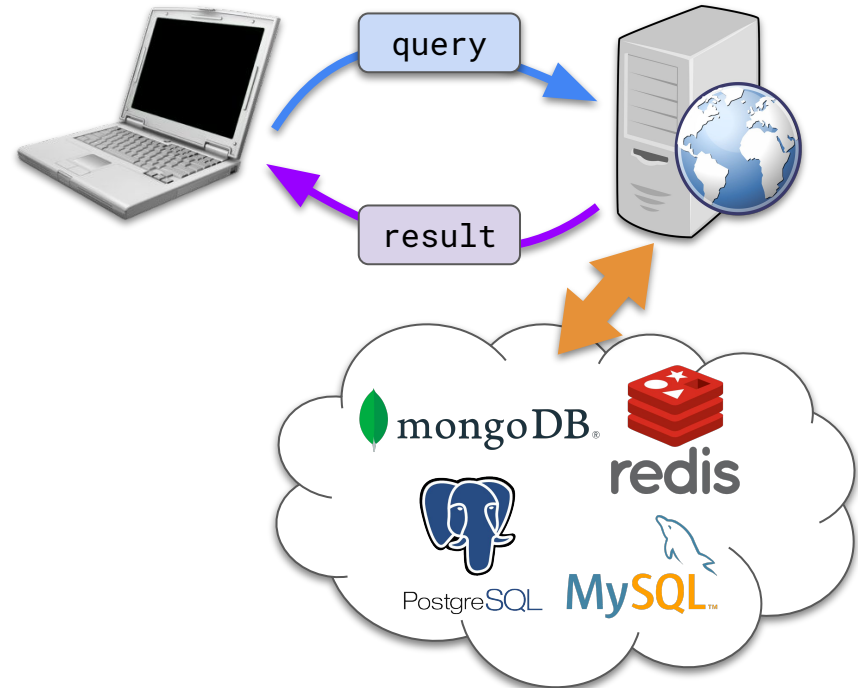
Can't we just **restrict all scripting** to be **exclusively on the client-side**?

The **client** would need to have **all server data stored locally...**

**Would be inefficient and insecure!**

# Web Databases

- **Databases:** how we store data on the server-side
  - Data **stored** by **server**
  - Data **queried** by **client**
  - Query **executed** by **server**
- A massive component of modern web applications
  - **Examples:** record keeping, user account management
- Popular DB Software:
  - MySQL, PostgreSQL
  - Redis, MongoDB



## Familiarity with SQL?

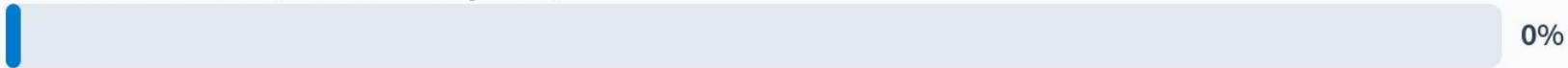
Some



Lots



None of the above (which is totally fine!)



# Structured Query Language (SQL)

- **A language to ask (“query”) databases questions**
  - Information stored in **tables**; columns = **attributes**, rows = **records**
- **Fundamental operations:**
  - **“SELECT”** : express queries
  - **“INSERT”** : create new records
  - **“UPDATE”** : modify existing data
  - **“DELETE”** : delete existing records
  - **“UNION”** : combine results of multiple queries
  - **“WHERE/AND/OR”** : conditional operations
- **Syntactical Tips:**
  - **“\*”** : all
  - **“NULL”** : nothing
  - **“-- ”** : comment-out the rest of the line (note the space at the end)

# Structured Query Language (SQL)

- A language to ask (“**query**”) databases questions
- E.g, How many users have the location **Salt Lake City**?
  - “**SELECT** **COUNT**(\*) **FROM** `users` **WHERE** **location**=‘Salt Lake City’”

# Structured Query Language (SQL)

- A language to ask (“**query**”) databases questions
- E.g., How many users have the location **Salt Lake City**?
  - “**SELECT** **COUNT**(**\***) **FROM** `users` **WHERE** **location**=‘Salt Lake City’”
- E.g., Is there a user with username “**bob**” and password “**abc123**”?
  - “**SELECT** **\*** **FROM** `users` **WHERE** **username**=‘bob’ **AND** **password**=‘abc123’”

# Structured Query Language (SQL)

- A language to ask (“**query**”) databases questions
- E.g, How many users have the location **Salt Lake City**?
  - “**SELECT** **COUNT**(**\***) **FROM** `users` **WHERE** **location**=‘Salt Lake City’”
- E.g., Is there a user with username “**bob**” and password “**abc123**”?
  - “**SELECT** **\*** **FROM** `users` **WHERE** **username**=‘bob’ **AND** **password**=‘abc123’”
- E.g., Completely delete this table!
  - “**DROP** **TABLE** `users`”



# Example DB and SQL Queries

- **Table name:** `users`

ID	username	password	passHash	location
1	Prof Nagy	c4ntgu3\$\$m3!	0x12345678	Salt Lake, UT
2	Average User	password123	0x87654321	Boulder, CO
3	Below Average	password	0x81726354	Denver, CO

- `SELECT * FROM users;`
  - `???`
- `SELECT * FROM users WHERE id = 2;`
  - `???`
- `SELECT password FROM users WHERE username = "Prof Nagy";`
  - `???`

# Example DB and SQL Queries

- **Table name: users**

ID	username	password	passHash	location
1	Prof Nagy	c4ntgu3\$\$m3!	0x12345678	Salt Lake, UT
2	Average User	password123	0x87654321	Boulder, CO
3	Below Average	password	0x81726354	Denver, CO

- `SELECT * FROM users;`
  - Will return **all users**
- `SELECT * FROM users WHERE id = 2;`
  - Will return just **Average User**
- `SELECT password FROM users WHERE username = "Prof Nagy";`
  - Will return **Prof Nagy's password**

# Questions?



# Food for Thought

- SQL databases and other web applications operate on **users' inputs**
  - E.g., SQL queries, HTTP GET and POST requests
  - That's how we interact with their **server-side applications!**
- **Question:** can we assume that all user input will only ever be **data**?



# Next time on CS 4440...

Web Exploitation, SQL Injection, CSRF, XSS