Countdown

Class is starting now!

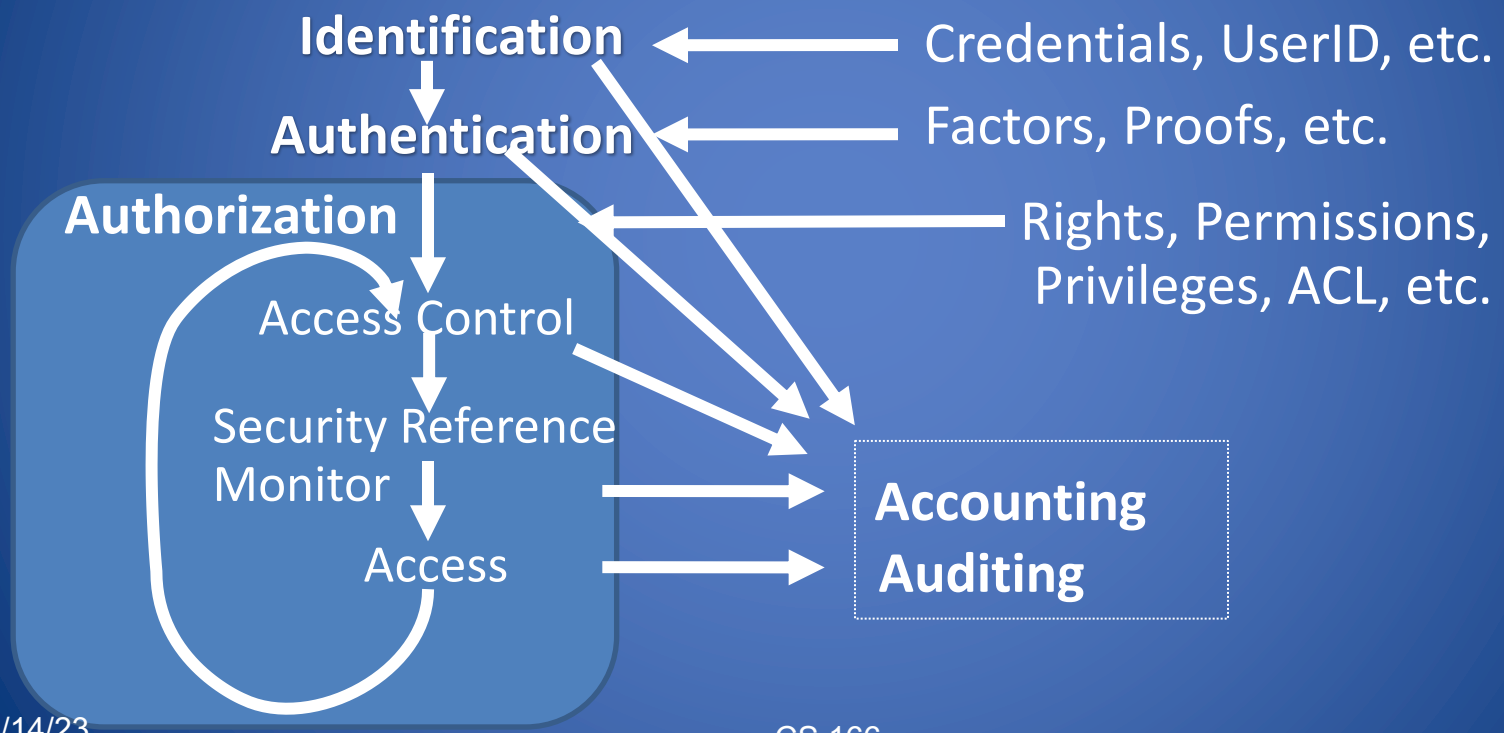
Web Security I

Web Security Models
Browser Security
Web Technologies and Protocols

AAA (recap)

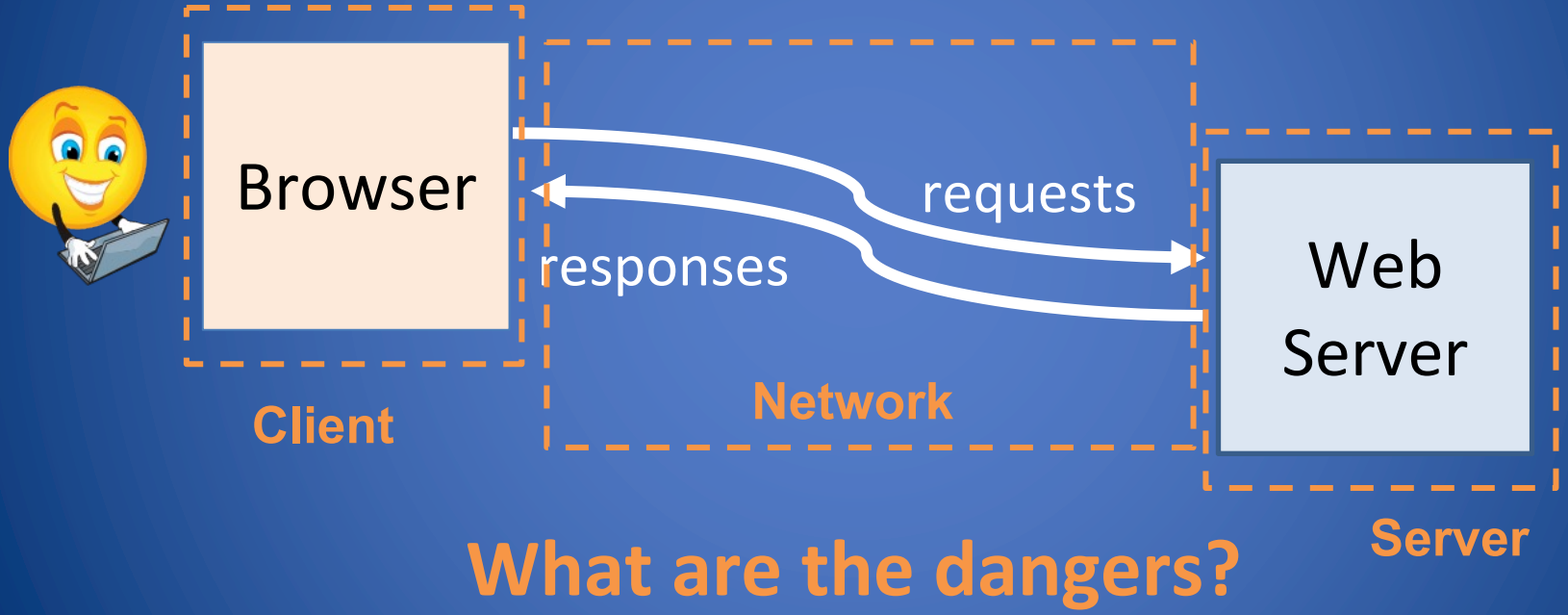
Identification, **Authentication**, **Authorization**, **Accounting**, Auditing

- AAA Working Group, IETF



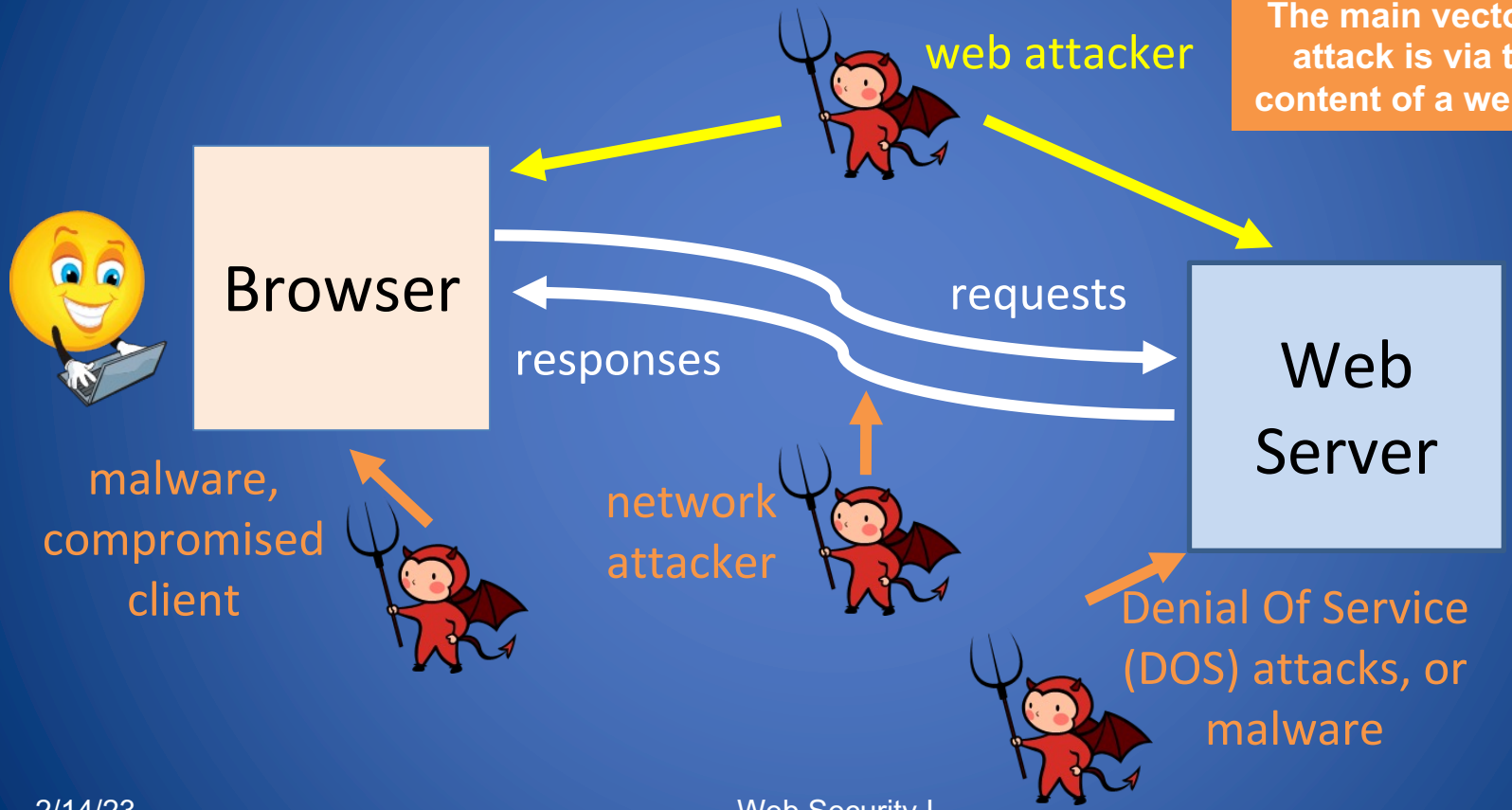
Web Security Model

Web Applications

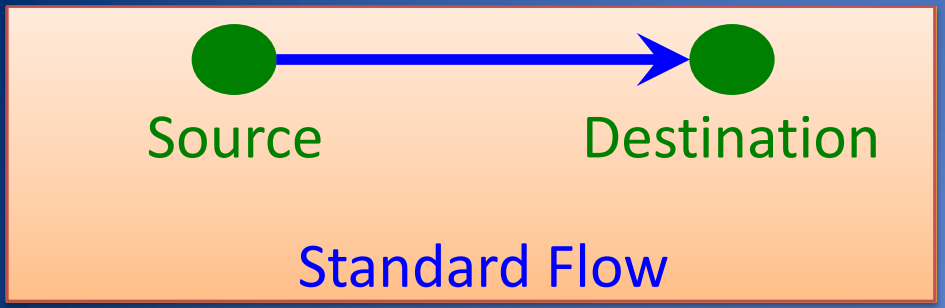


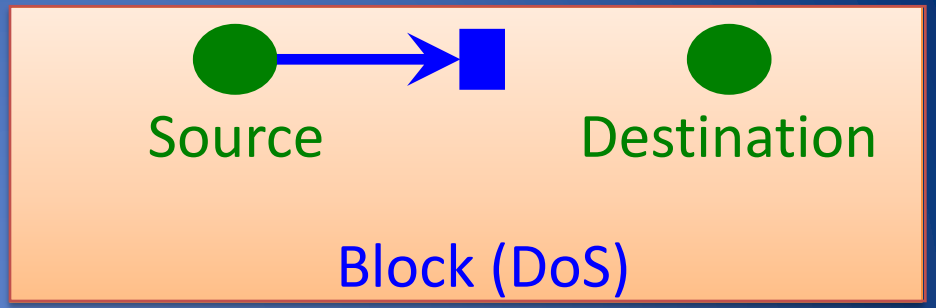
What are the dangers?

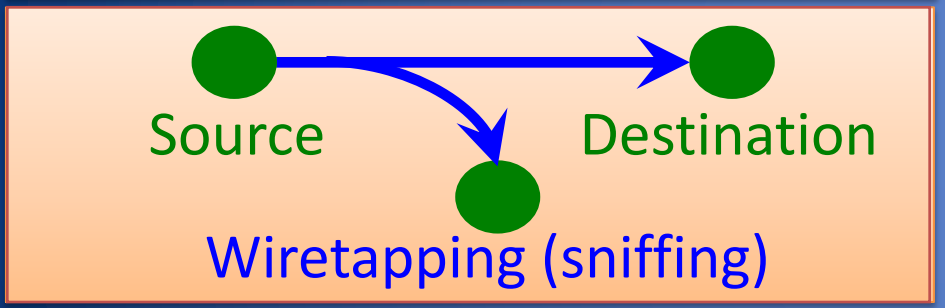
Threat Models

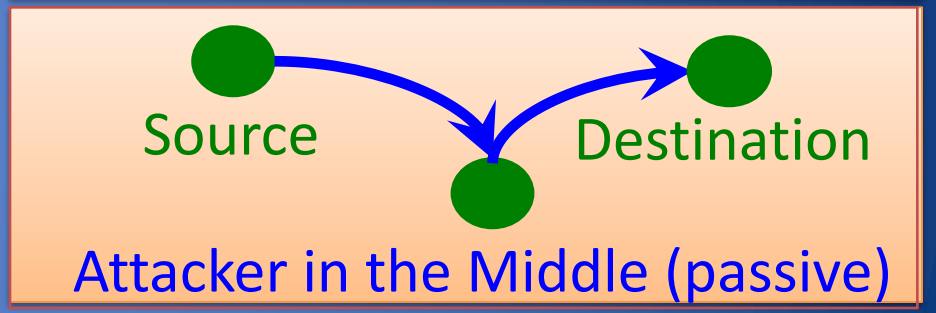


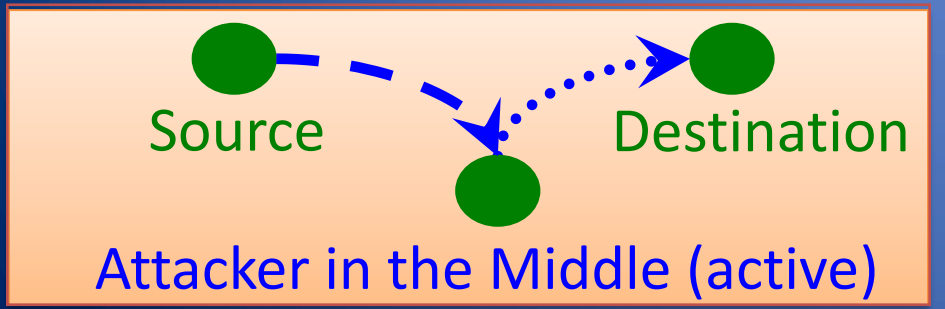
Network Attacks

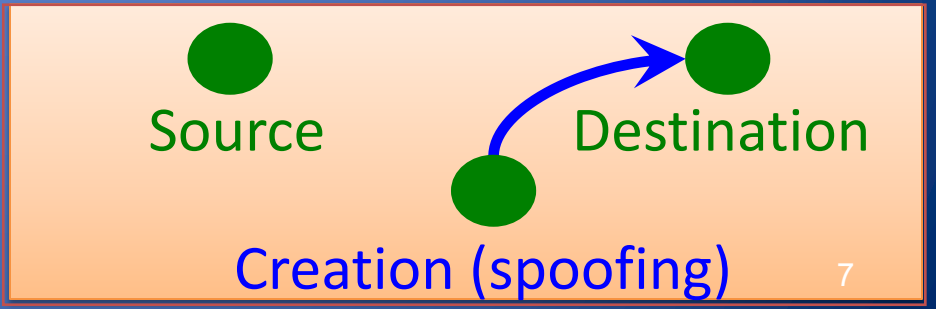












Web Attacker Capabilities

- Attacker controls malicious website
  - Website might look professional, legitimate, etc.
  - Attacker can get users to visit website (how?)
- Good website is compromised by attacker
  - Attacker inserts malicious content into website
  - Attacker steals sensitive data from website
  - …

Attacker does not have direct access to user's machine

Potential Damage

- An attacker gets you to visit a malicious website
  - Can they perform actions on other websites impersonating you?
  - Can they run evil code on your OS?
- Ideally, none of these exploits are possible ...

Attack Vectors

- Web browser (focus of this lecture)
  - Renders web content (HTML pages, scripts)
  - Responsible for confining web content
  - **Note:** Browser implementations dictate what websites can do
- Web applications
  - Server code (PHP, Ruby, Python, …)
  - Client-side code (JavaScript)
  - Many potential bugs (which you'll explore in Project 2 ☺)

Browser Security: Sandbox

- Goal: protect local computer from web attacker
  - Safely execute code on a website
  - … without the code accessing your files, tampering with your network, accessing other sites
- High stakes ($30K bounty for Google Chrome ; www.google.com/about/appsecurity/chrome-rewards/)
- We won't address attacks that break the sandbox
- But they **happen** check the **CVE list**
  - **https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=sandbox**
  - **https://support.apple.com/en-us/HT213635**

Domains, HTML and HTTP

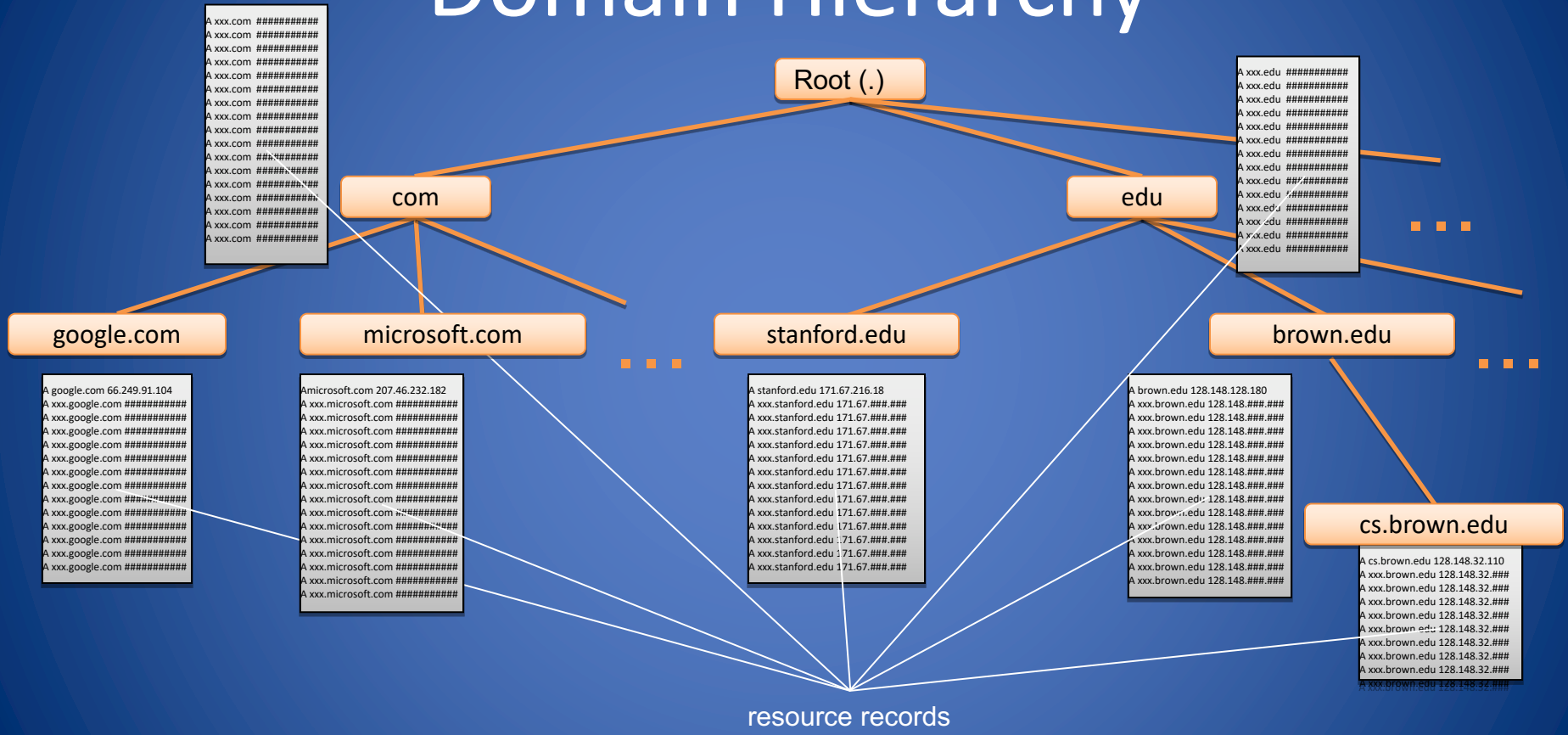
URL and FQDN

- URL Uniform Resource Locator
- https://cs.brown.edu/about/contacts.html
  - a protocol (e.g. https), a FQDN (e.g. cs.brown.edu)
  - a path and file name (e.g. /about/contacts.html).

- 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), .com, .org, .net, …
  - Country-code (ccTLD), .ca, .it, …

Domain Hierarchy



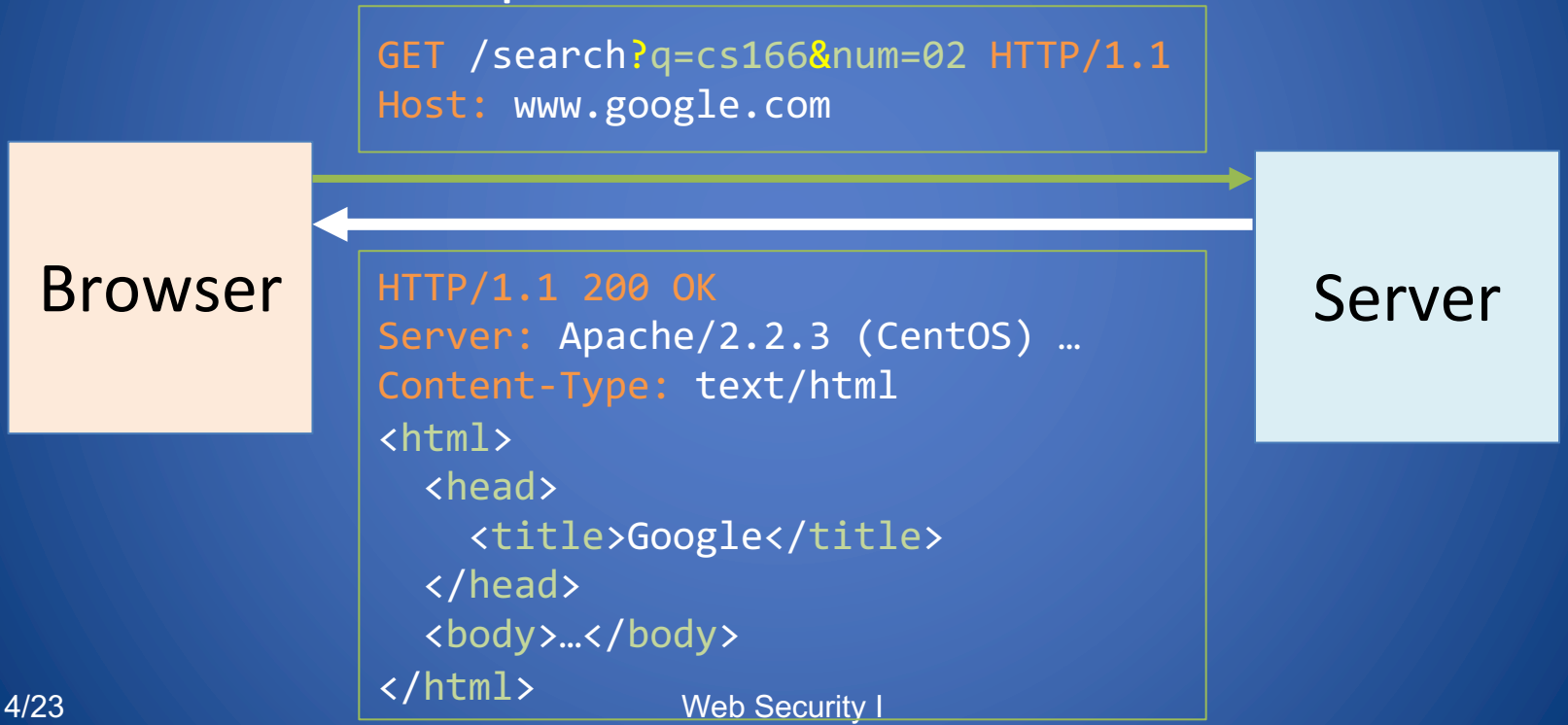
HTML

- Hypertext markup language (HTML)
  - Allows linking to other pages (href)
  - Supports embedding of images, scripts, other pages (script, iframe)
  - User input accepted in forms

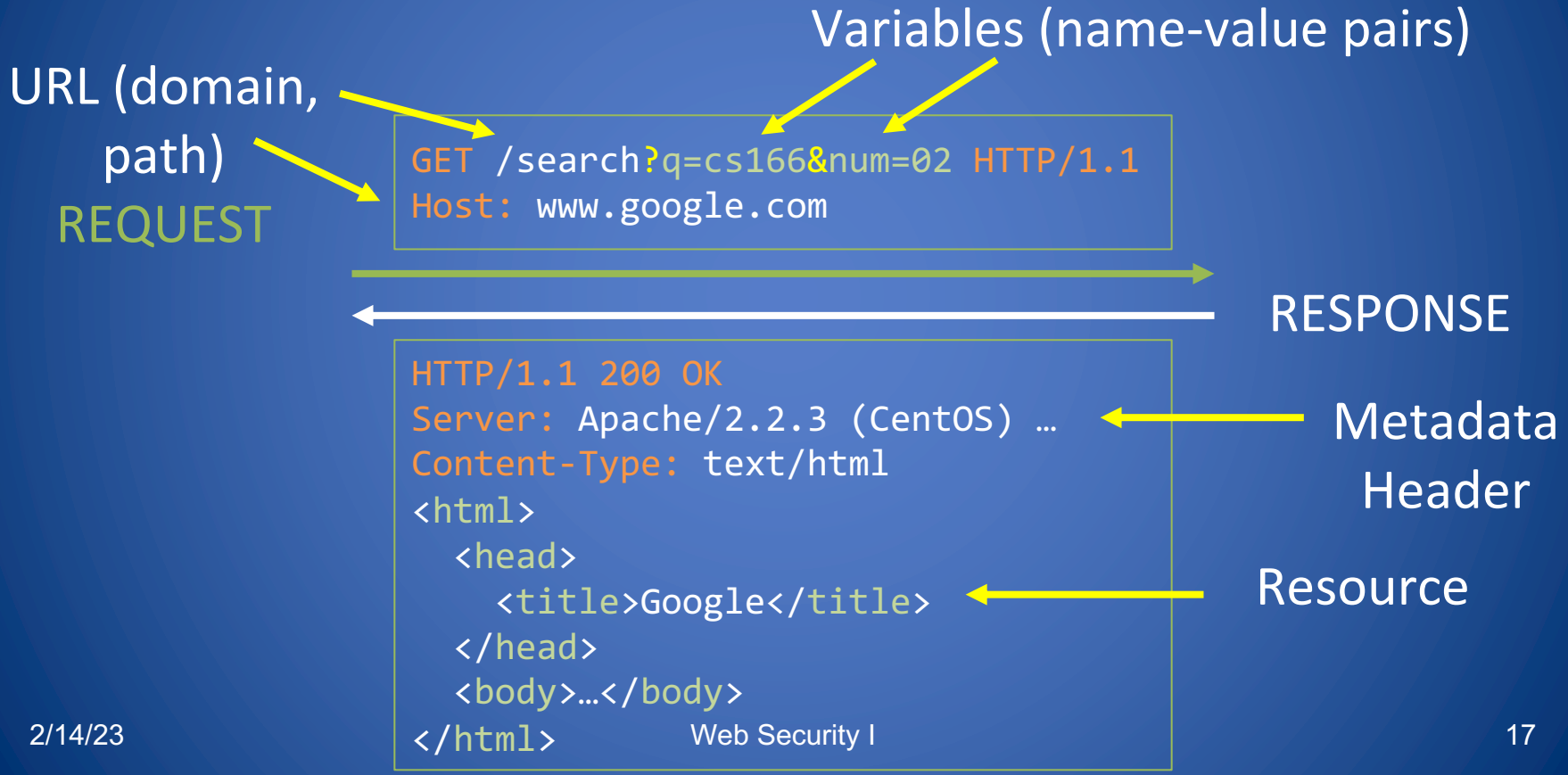
```
<html>
  <head>
    <title>Google</title>
  </head>
  <body>
    <p>Welcome to my page.</p>
    <script>alert(“Hello world”);
    </script>
    <iframe src=“http://example.com”>
    </iframe
  </body>
</html>
```

HTTP (Hypertext Transport Protocol)

- Communication protocol between client and server



What's in a request (or response)?



Variables

- Key-value pairs obtained from user input into forms and submitted to server
- Submit variables in HTTP via GET or PUT
- GET request: variables within HTTP URL, e.g.,

  http://www.google.com/search?q=cs166&num=02

- POST request: variables within HTTP body, e.g.,

```
POST / HTTP/1.1
Host: example.com
Content-Type:
application/x-www-form-urlencoded
Content-Length: 18

month=05&year=2021
```

Semantics: GET vs. POST

- GET
  - Request target resource
  - Read-only method
  - Submitted variables may specify target resource and/or its format
- POST
  - Request processing of target resource
  - Read/write/create method
  - Submitted variables may specify how resource is processed (e.g., content of resource to be created, updated, or executed)

GET vs. POST

	GET	POST
Browser history	✓	X
Browser bookmarking	✓	X
Browser caching	✓	X
Server logs	✓	X
Reloading page	immediate	warning
Variable values	Restricted	arbitrary

Moving from Browser Security to Web Application Security: Client-Side Controls

Client-Side Controls

- Web security problems arises because clients can submit arbitrary input
- **What about using client side controls to check the input?**
- **Which kind of controls?**

Client-Side Controls

- A standard application may rely on client-side controls to restrict user input in two general ways:
  - Transmitting data via the client component using a mechanism that should prevent the user from modifying that data
  - Implementing measures on the client side

Bypassing Web Client-Side Controls

- In general a security flaw because it is easy to bypass
- The user:
  - has a full control over the client and the data it submits
  - Can bypass any controls that are client-side and not replicated on the server
- Why these controls are still useful?
  - E.g. for load balancing or usability
  - Often we can suppose that the vast majority of users are honest

Transmitting Data Via the Client

- A common developer bad habit is passing data to the client in a form that the end user cannot directly see or modify
- Why is it so common?
  - It removes or reduces the amount of data to store server side per-session
  - In a multi-server application it removes the need to synchronize the session data among different servers
  - The use of third-party components on the server may be difficult or impossible to integrate
- Transmitting data via the client is often the easy solution but unfortunately is not secure.

Common Mechanisms

- HTML Hidden fields
  - A field flagged hidden is not displayed on-screen
- HTTP Cookies
  - Not displayed on-screen, and the user cannot modify directly
- Referer Header
  - An optional field in the http request that it indicates the URL of the page from which the current request originated
- If you use the proper tool you can tamper the data on the client-side

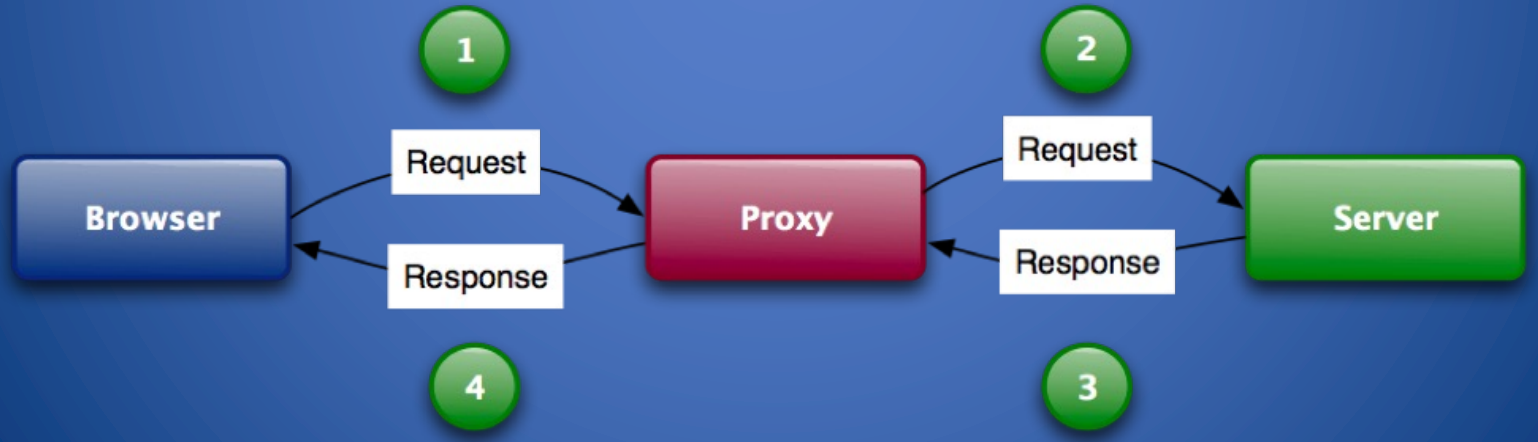
Web client tool

- Web inspection tool:
  - Firefox or Chrome web developer:

    - powerful tools that allow you to edit HTML, CSS and view the coding behind any website: CSS, HTML, DOM and JavaScript
- Web Proxy:
  - Burp, OWASP ZAP, etc.
    - Allow to modify GET or POST requests

HTTP Proxy

- An intercepting Proxy:
  - inspect and modify traffic between your browser and the target application
  - Burp Intruder, OWASP ZAP, etc.



Demos

- Owasp Webgoat https://tryhackme.com/room/webgoat
  - parameter injection
  - Bypass html field restrictions
  - Exploit hidden fields
  - Bypass client side java script validation

Break!!!!!

60 60 60 60 60

Class is starting now!

Browser Security: Same-Origin Policy

- Goal: Protect and isolate web content from other web content
  - Content from different origins should be isolated, e.g., mal.com should not interact with bank.com in unexpected ways
  - What about cs.brown.edu vs brown.edu or mail.google.com vs drive.google.com?
  - Lots of subtleties

SOP Example: (protocol, domain, port)
http://store.company.com/dir/page.html

URL	Outcome	Reason
`http://store.company.com/dir2/other.html`	Same origin	Only the path differs
`http://store.company.com/dir/inner/another.html`	Same origin	Only the path differs
`https://store.company.com/page.html`	Failure	Different protocol
`http://store.company.com:81/dir/page.html`	Failure	Different port (`http://` is port 80 by default)
`http://news.company.com/dir/page.html`	Failure	Different host

Source: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

Back to Browser Security: SOP

- Very simple idea: “Content from different origins should be isolated”
  - Website origin defined over tuple (protocol, domain, port)
- Very difficult to execute in practice…
  - Messy number of cases to worry about…
    - HTML elements?
    - Navigating Links?
    - Browser cookies?
    - JavaScript capabilities?
    - iframes?
    - etc.
  - Browsers didn’t always get this correct…

SOP: Cookies

Cookies

- HTTP is a stateless protocol; cookies used to emulate state
- Servers can store cookies (name-value pairs) into browser
  - Used for user preferences, user tracking, authentication
  - Expiration date can be set
  - May contain sensitive information (e.g., for user authentication)
- Browser sends back cookies to server on the next connection

```
POST /login.php HTTP/1.1
Set-Cookie: Name: sessionid
            Value: 19daj3kdop8gx
            Domain: cs.brown.edu
            Expires: Wed, 21 Oct 2021 …
```

Cookie Scope

- Each cookie has a scope
  - Base domain, which is a given host (e.g., brown.edu)
  - Plus, optionally, all its subdomains (cs.brown.edu, math.brown.edu, www.cs.brown.edu , etc.)
- For ease of notation, we denote with + the inclusion of subdomains (e.g., +brown.edu)
  - This isn't the real notation—it's actually specified in HTTP with the "Domain:" attribute of a cookie

Same Origin Policy: Cookie Reads

Websites can only read cookies within their scope

- Example: browser has cookies with scope
  brown.edu
  +brown.edu,
  +math.brown.edu
  cs.brown.edu
  +cs.brown.edu,
  help.cs.brown.edu
- Browser accesses cs.brown.edu
- Browser sends cookies with scope
  +brown.edu
  cs.brown.edu
  +cs.brown.edu

Same Origin Policy: Cookie Writes

A website can set cookies for (1) its base domain; or (2) a super domain (except TLDs) and its subdomains

- Browser accesses cs.brown.edu
- cs.brown.edu can set cookies for
  +brown.edu
  cs.brown.edu

- But not for
  google.com
  +com
  math.brown.edu
  brown.edu
  …

Application of Cookies: Sessions

- Sessions
  - Keep track of client over a series of requests
  - Server assigns clients a unique, unguessable ID
  - Clients send back ID to verify themselves
- Sessions
  - Necessary in sites with authentication (e.g., banking)
  - Useful in most other sites (e.g., remembering preferences)
- Various methods to implement them (mainly cookies), but also could be in HTTP variables

Third-Party Cookies

- Cookies are set and returned in each HTTP request and response
- Accessing a site can result in HTTP requests to various domains
  - E.g., embedded images can be loaded from other domains
- Third-party cookie
  - Set by server with domain different from that of original request (e.g., ad network)
- Example
  - Site brown.edu embeds YouTube videos
  - Accessing brown.edu results in third-party cookies set by youtube.com
- Browser can be configured not to store third-party cookies (recommended)

Clicker Question #1

If the browser accesses cs.brown.edu, the server can set cookies with which of the following scopes?

A. +brown.edu
B. only math.brown.edu
C. only help.cs.brown.edu
D. All of the above
E. None of the above

Answer

If the browser accesses cs.brown.edu, the server can set cookies with which of the following scopes?

A. +brown.edu
B. only math.brown.edu
C. only help.cs.brown.edu

…

The scope is cs.brown.edu by default

The server can optionally set cookies with scope +cs.brown.edu and +brown.edu, but nothing else

What We Have Learned

- Web Security Models
- Same-Origin Policy
- Basics of HTTP protocol
- GET and POST methods for HTTP variables
- Client-Side Controls
- Scope of cookies
- Session cookies
- Third-party cookies
- JavaScript