https://brown-csci1660.github.io

CS1660: Intro to Computer Systems Security
Spring 2026

Lecture 11: Web Security II

Instructor: **Nikos Triandopoulos**

March 3, 2026

BROWN

CS1660: Announcements

- Course updates
  - Project 2 is due next week
  - Lecture notes to be consistently updated
  - Special materials to be posted for midterm preparation

Last class

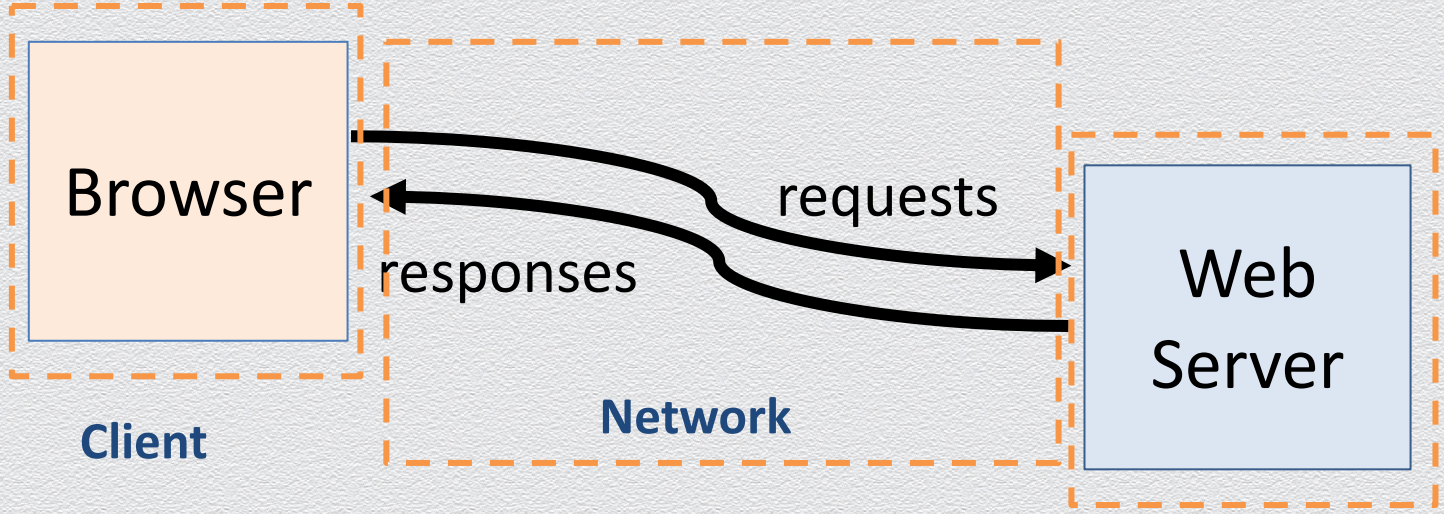
- Cryptography
- Authentication
  - User authentication: something you know, are, have
  - Data authentication: Merkle tree
  - System authentication: Challenge-response methodology, randomness revisited
- Web security
  - The Dyn DDOS attack
  - Web security model
    - Background, web-application security, browser security, cookies

Today

- Cryptography
- Authentication
  - User authentication: something you know, are, have
  - Data authentication: Merkle tree
  - System authentication: Challenge-response methodology, randomness revisited
- Web security
  - The Dyn DDOS attack
  - Web security model
    - Background, web-application security, browser security, cookies

11.0. Web security model

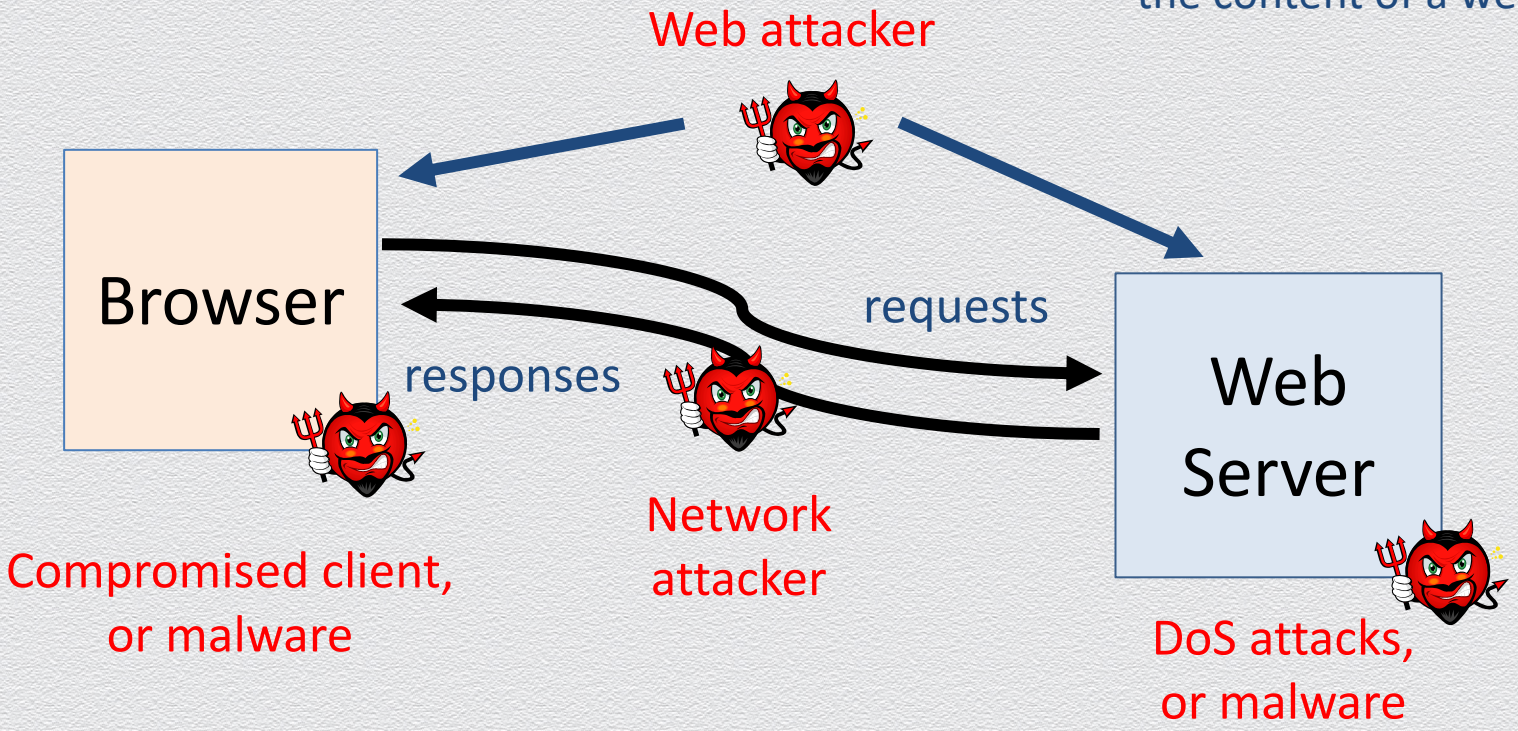
Web applications



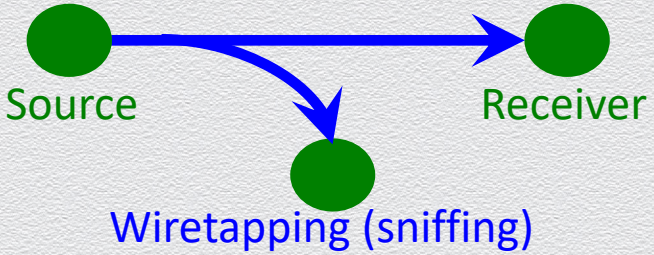
What are the dangers?

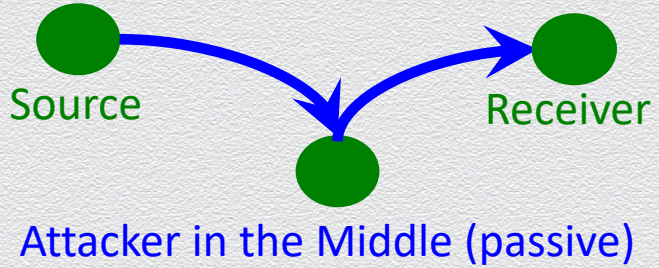
Threat models

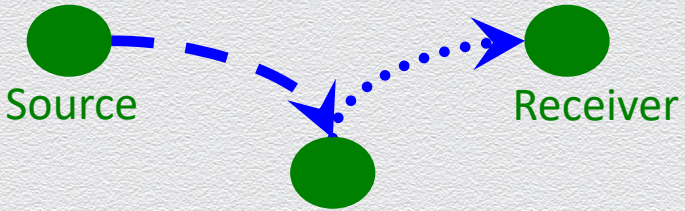
The main vector of attack is via the content of a website

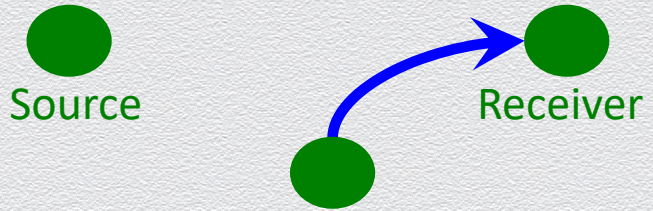


Network attacks









Web attacker capabilities

- Attacker controls a malicious website
  - website might look professional, legitimate, etc.
  - attacker can get users to visit website (how?)
- A benign 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
  - 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 (e.g., see 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, or accessing other sites

High stakes

- $40K 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 (Common Vulnerabilities and Exposures)
  - https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=sandbox
  - https://support.apple.com/en-us/HT213635

11.1 Domains, HTML, HTTP

URL and FQDN

URL: Uniform Resource Locator

https://cs.brown.edu/about/contacts.html

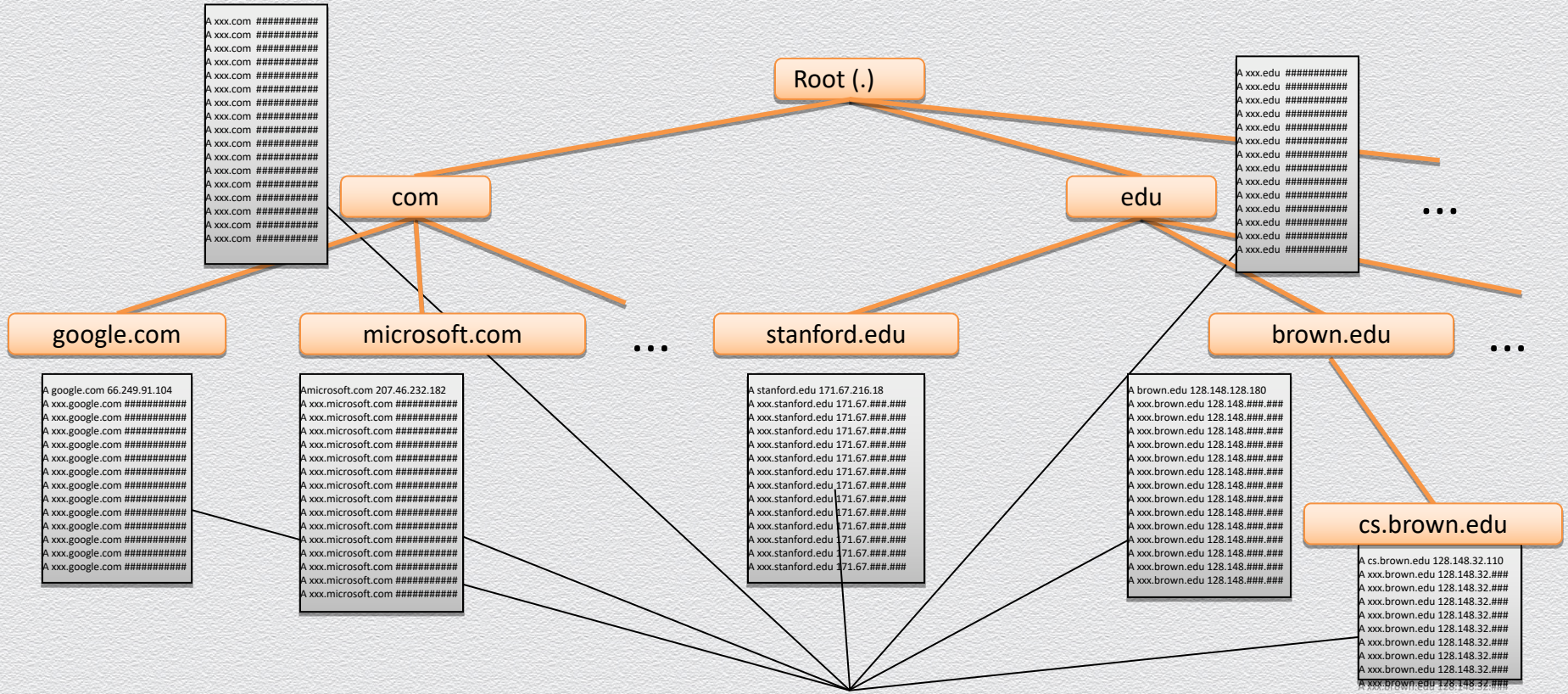
- a protocol
  - e.g. https
- a FQDN
  - e.g. cs.brown.edu
- a path and file name
  - e.g. /about/contacts.html

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
  - a "." at the end of the FQDN
- Top-level domain (TLD)
  - generic (gTLD): .com, .org, .net,
  - country-code (ccTLD): .ca, .it, , .gr …

Domain hierarchy



resource records

HTML

Hypertext markup language (HTML)

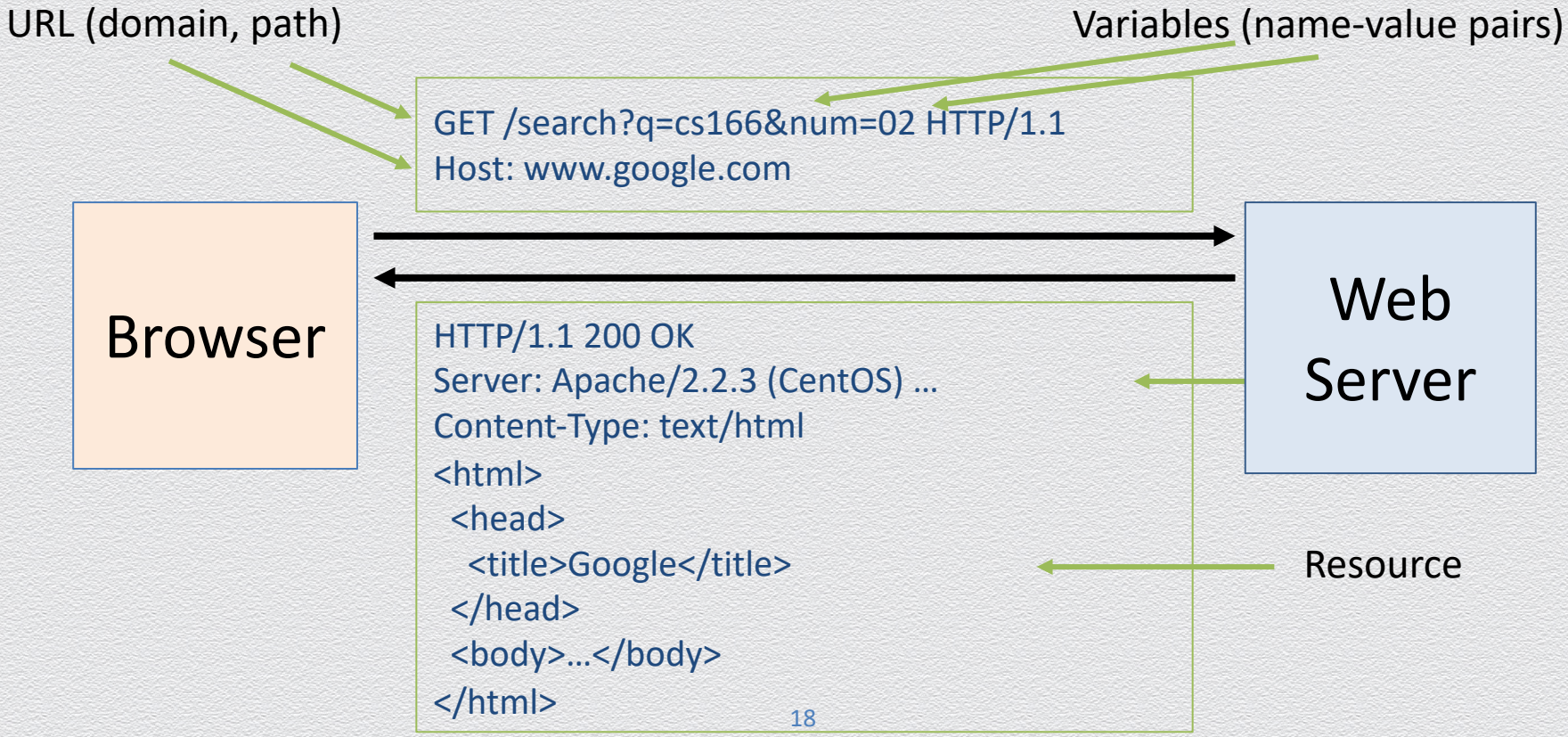
- allows linking to other pages (href)
- supports embedding of images, scripts, other pages (script, iframe)
- user input accepted in forms

```
<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 & submitted to server

- Submit variables in HTTP via GET or POST
- GET request: variables within HTTP URL
  - e.g., http://www.google.com/search?q=cs166&num=02
- POST request: variables within HTTP body
  - POST / HTTP/1.1
  - Host: example.com
  - Content-Type: application/x-www-form-urlencoded
  - Content-Length: 18
  - month=5&year=2024

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

11.2 Web-application security

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 (cont.)

A standard application may rely on client-side controls

- They 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

- In this threat model
  - Server does not trust the Client

Bypassing 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?
  - 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 multi-server applications, 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 it 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
- Referrer 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

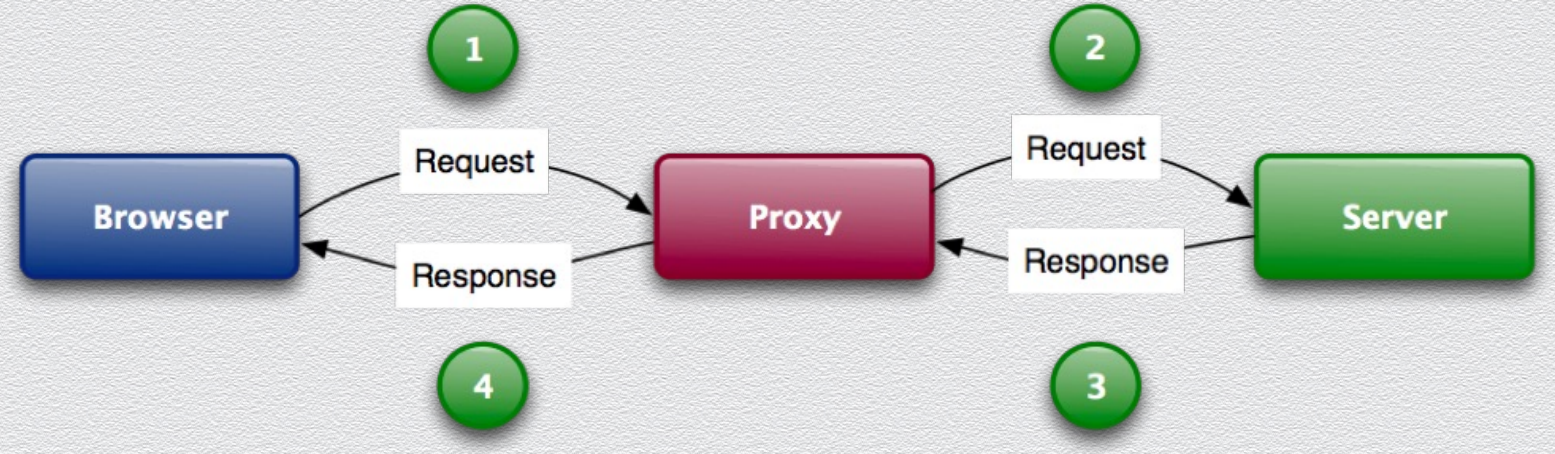
Web client tool

- 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.



11.3 Browser security

In BROWSER we trust…

- Most of our trust on web security relies on information stored in the Browser
  - a Browser should be updated since Bugs in the browser implementation can lead to various attacks
  - e.g., https://us-cert.cisa.gov/ncas/current-activity/2023/02/14/mozilla-releases-security-updates-firefox-110-and-firefox-esr
- Add-ons too are dangerous
  - Hacking Team flash exploits - goo.gl/syVwiD
  - github.com/greatsuspender/thegreatsuspender/issues/1263
- Executing a browser with low privileges helps

Browser Security: Same-Origin Policy (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…

Browser Security: Same-Origin Policy (SPO) (cont.)

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: http://store.company.com/dir/page.html

(protocol, domain, port)

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

11.4 Cookies

Cookies

- HTTP is a stateless protocol; cookies used to emulate state
- Servers can store cookies (name-value pairs) into browser
  - user preferences (e.g., language and page layout), user tracking, authentication
  - expiration date can be set
  - may contain sensitive information (e.g., for user authentication)

```
POST /login.php HTTP/1.1
Set-Cookie: Name: sessionid
        Value: 19daj3kdop8gx
        Domain: cs.brown.edu
        Expires: Wed, 21 Feb 2024 …
```

- Browser sends back cookies to server on the next connection

Cookies 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, included subdomains are denoted as .
    - e.g., .brown.edu
    - in fact, specified in HTTP with the "Domain:" attribute of a cookie

Same Origin Policy: Cookie Reads

Websites can only read cookies within their scope

- Browser has cookies with scope
  - brown.edu
  - .brown.edu
  - .math.brown.edu
  - cs.brown.edu
  - .cs.brown.edu
  - blog.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
  - …

Session Management

Session

- keep track of client over a series of requests
- server assigns clients a unique, unguessable ID
- clients send back ID to verify themselves

Session

- 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

Session Management (cont.)

- Goal
  - users should not have to authenticate for every single request
- Problem
  - HTTP is stateless
- Solution
  - user logs in once
  - server generates session ID and gives it to browser
    - temporary token that identifies and authenticates user
  - browser returns session ID to server in subsequent requests

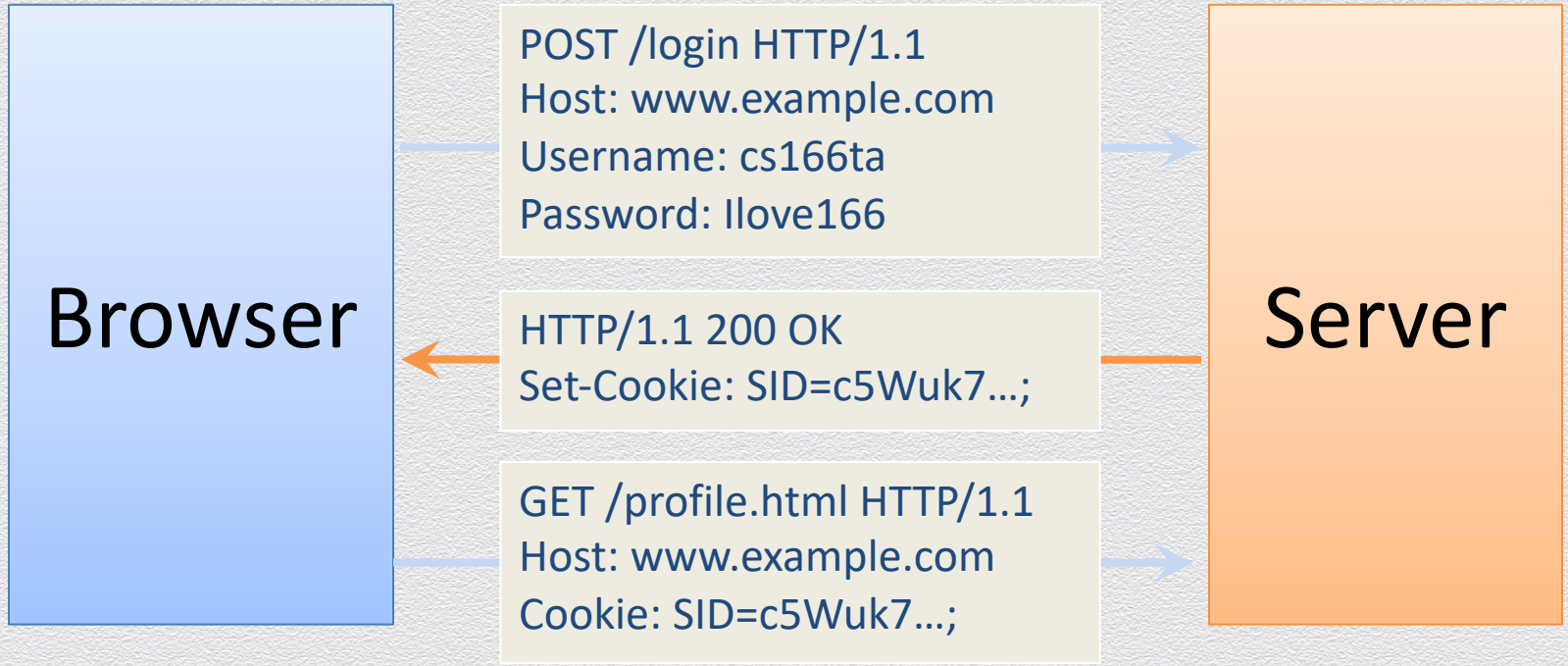
Specifications for a Session ID

- Created by server upon successful user authentication
  - generated as long random string
  - associated with scope (set of domains) and expiration
  - sent to browser
- Kept as secret shared by browser and server
- Transmitted by browser at each subsequent request to server
  - must use secure channel between browser and server
- Session ID becomes invalid after expiration
  - user asked to authenticate again

Implementation of Session ID

- Cookie
  - Transmitted in HTTP headers
  - Set-Cookie: SID=c5Wuk7…
  - Cookie: SID=c5Wuk7…
- GET variable
  - Added to URLs in links
  - https://www.example.com?SID=c5Wuk7…
- POST variable
  - Navigation via POST requests with hidden variable
  - <input type="hidden" name="SID" value="c5Wuk7…">

Session ID in Cookie



Session ID in Cookie (cont.)

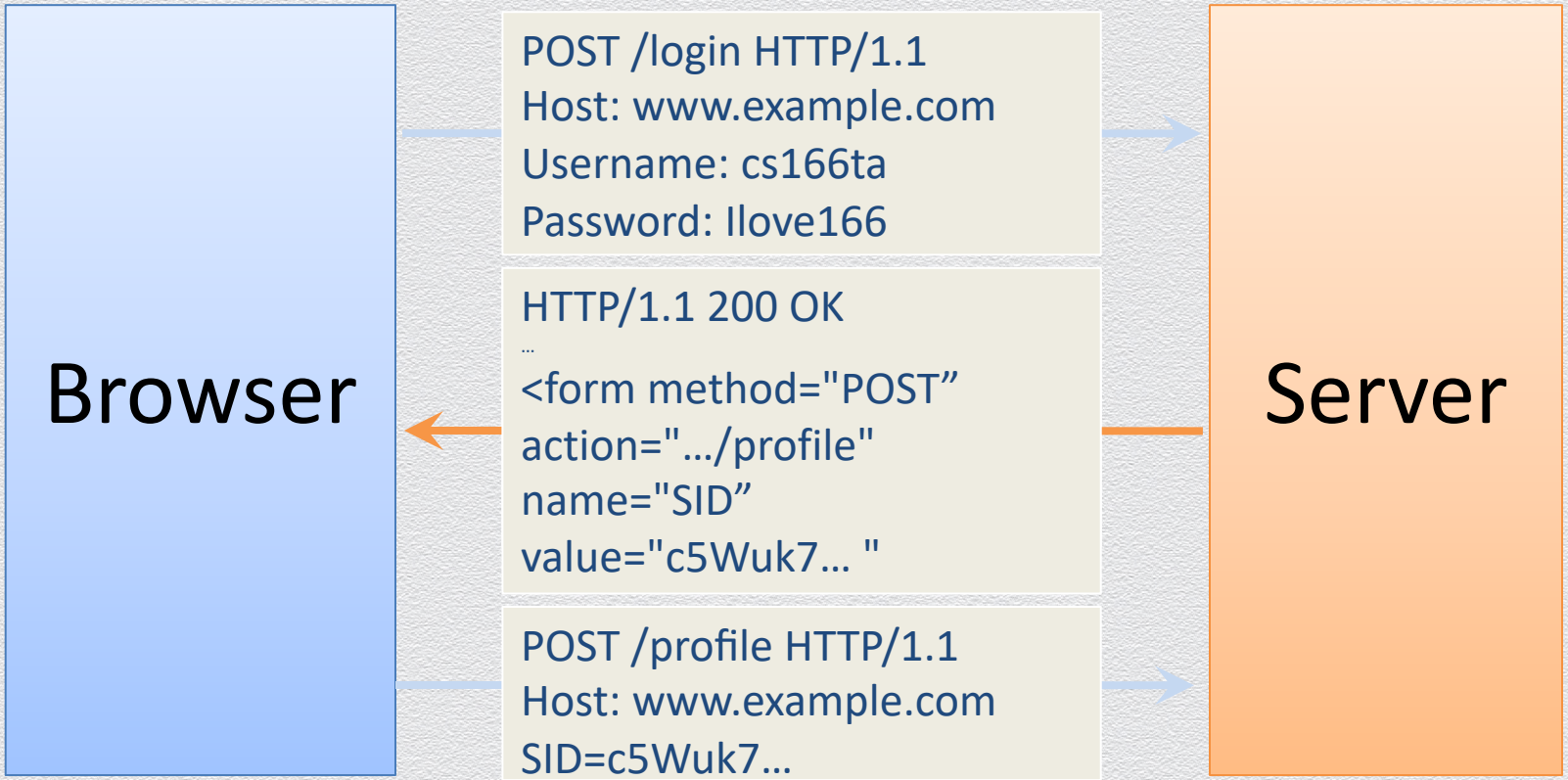
- Advantages
  - Cookies automatically returned by browser
  - Cookie attributes provide support for expiration, restriction to secure transmission (HTTPS), and blocking JavaScript access (httponly)
- Disadvantages
  - Cookies are shared among all browser tabs
    - (**not** other browsers or incognito)
  - Cookies are returned by browser even when request to server is made from element (e.g., image or form) within page from other server
  - This may cause browser to send cookies in context not intended by user

Session ID in GET Variable

Session ID in GET Variable (cont.)

- Advantages
  - Session ID transmitted to server only when intended by user
- Disadvantages
  - Session ID inadvertently transmitted when user shares URL
  - Session ID transmitted to third-party site within referrer
  - Session ID exposed by bookmarking and logging
  - Server needs to dynamically generate pages to customize site navigation links and POST actions for each user
  - Transmission of session ID needs to be restricted to HTTPS on every link and POST action

Session ID in POST Variable



Session ID in POST Variable

- Advantages
  - Session ID transmitted to server only when intended by user
  - Session ID not present in URL, hence not logged, bookmarked, or transmitted within referrer
- Disadvantages
  - Navigation must be made via POST requests
  - Server needs to dynamically generate pages to customize forms for each user
  - Transmission of session ID needs to be restricted to HTTPS on every link and POST action