Week 9: Lecture A
Client-side Web Security and HTTPS

Tuesday, October 21, 2025

Announcements

- **Project 3: WebSec** released
  - **Deadline:** Thursday, November 6th by 11:59PM

Project 3: Web Security

Deadline: Thursday, November 6 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 across all teams: **93.39%**
  - Three solved one of the extra credit targets

- **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 10/27** via **Google Form**

Reminders: Extra Credit Opportunities

- Extra credit challenges **in all projects**
  - Many points to earn **in Projects 3 & 4**

- **Online Piazza Participation:**
  - Top-10 answerers = **5pts extra credit**

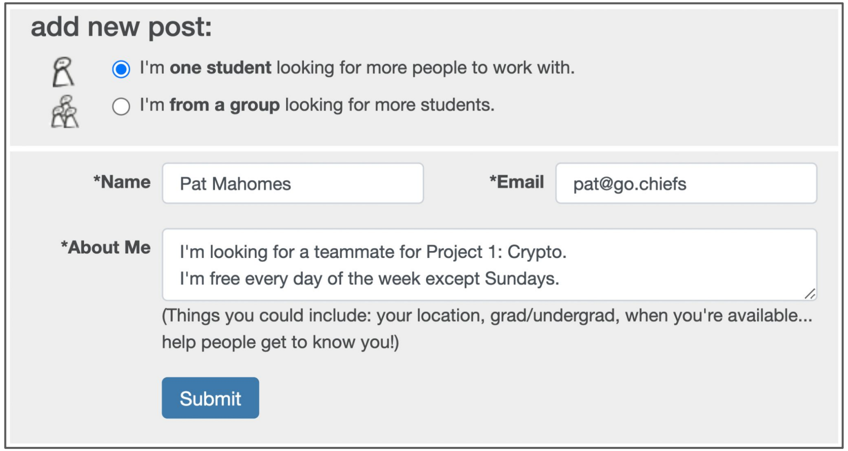
- **Course Wiki Contributions:**
  - Valuable additions = **1pt extra credit**
  - Requires my pre-approval

Reminders: Find a Teammate!

- Can work in **teams of up to two**
  - Find teammates on **Piazza**
  - Post on 

- Why work with someone else?
  - Pair programming
  - Divide and conquer
  - Two sets of eyes to solve problems
  - Teaching others helps you learn more

- Yes, you are free to work solo…
  - But we encourage you to team up!



Announcements

INAUGURAL

STUDENT AI SYMPOSIUM

Student Perspectives: AI and Society

DATE: Friday, November 21, 2025

TIME: 8:00 AM–4:00 PM

LOCATION: Marriott Library - Gould Auditorium

- A platform for students to lead conversations about AI in society.
- Invites faculty to listen and learn from student perspectives.
- Sparks meaningful discussions on AI's impact today and in the future.

LIGHTNING TALK

Share your most impactful use of AI with a 5–10 minute presentation.

RESEARCH PRESENTATION

Share your research or project in a 15–20 minute presentation.

DISCUSS THE IMPLICATIONS FOR

ETHICS

TECHNOLOGY

EDUCATION

BUSINESS

SPONSORED BY

TEKCLUB

AI4U AT THE U

RAi

J. Willard Marriott Library, THE UNIVERSITY OF UTAH

DIGITAL LEARNING TECHNOLOGIES, THE UNIVERSITY OF UTAH

Office of RESEARCH INTEGRITY & COMPLIANCE, THE UNIVERSITY OF UTAH

SCAN THE QR CODE TO APPLY ON CANVAS

SUBMISSION DEADLINE OCTOBER 31, 2025

Announcements

WHAT

STUDENT TEAMS OF ALL SKILL LEVELS
WITH INDUSTRY MENTORS

WHEN

Friday, October 24
4:30 PM - 12:00 AM

WHERE

WEB 1230
72 S Central Campus Dr
Salt Lake City, UT 84112

REGISTER

SCAN THE QR CODE
FOR MORE DETAILS
AND TO REGISTER

Announcements

utahsec

See Discord for meeting info!

utahsec.cs.utah.edu

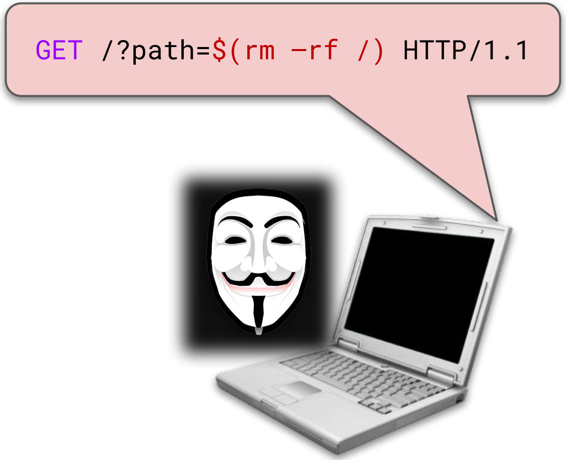
Questions?

Last time on CS 4440…

Web Attacks
SQL Injection
Cross-site Scripting
Cross-site Request Forgery

Code Injection in Web Apps

- **A common and dangerous class of attacks**
  - Shell Injection
  - SQL Injection
  - Cross-Site Scripting
  - Control-flow Hijacking (buffer overflows)



Code Injection in Web Apps

- **A common and dangerous class of attacks**
  - Shell Injection
  - SQL Injection
  - Cross-Sit
  - Control-f

```
GET /?path=$(rm -rf /) HTTP/1.1
```

What is the universal flaw here?

Code Injection in Web Apps

- A common and dangerous class of attacks
  - Shell Injection
  - SQL Injection
  - Cross-Sit
  - Control-f

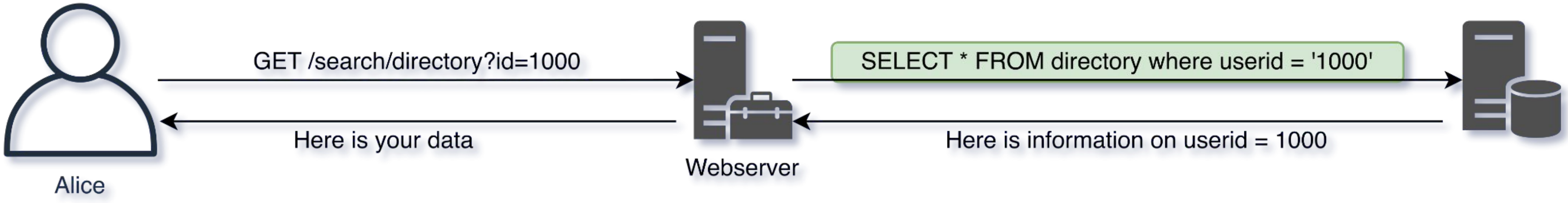
```
GET /?path=$(rm -rf /) HTTP/1.1
```

What is the universal flaw here?

Confusing input data with code!

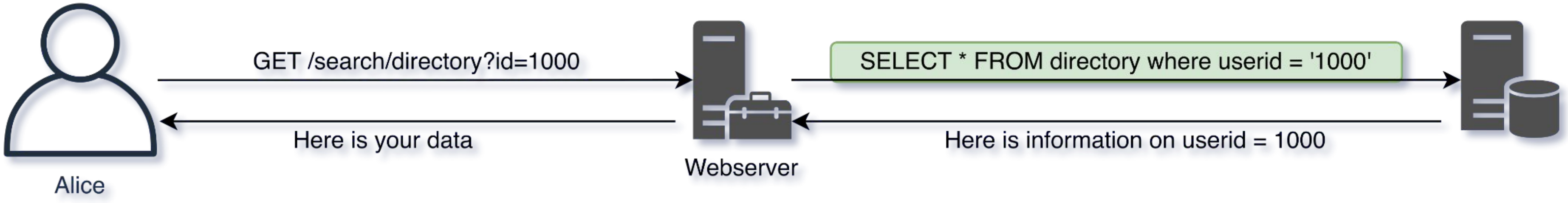
SQL Injection Attacks

- **Attacker goal: ???**



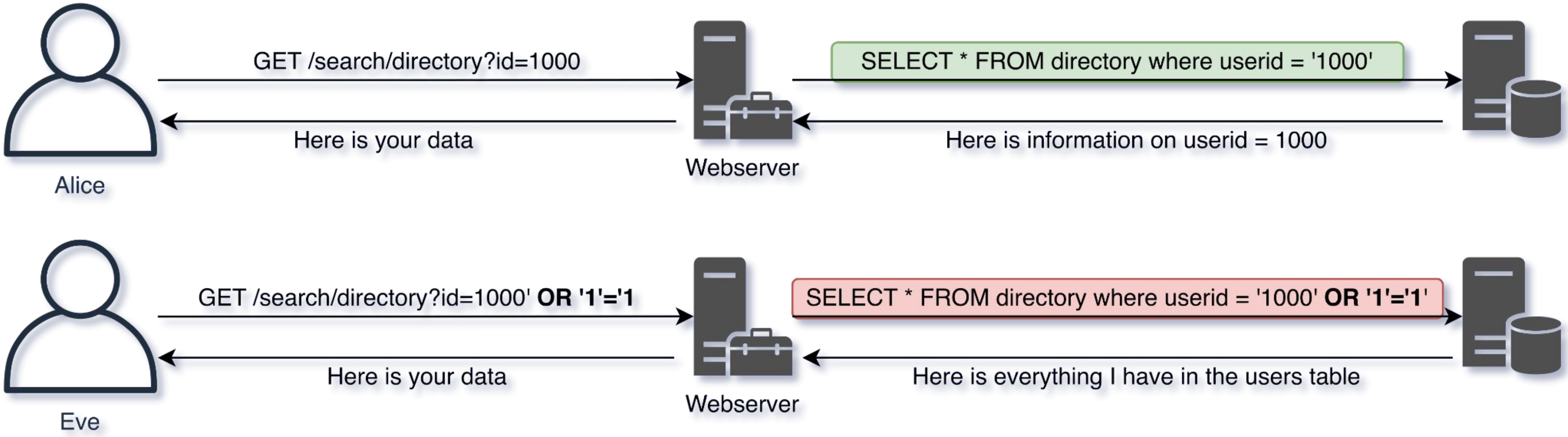
SQL Injection Attacks

- **Attacker goal:** inject or modify database **commands** to **read** or **alter** info



SQL Injection Attacks

- **Attacker goal:** inject or modify database **commands** to **read** or **alter** info



Project 3: SQL Injection Tips

1. Identify **how the input is processed** on the server-side

Project 3: SQL Injection Tips

1. Identify **how the input is processed** on the server-side
   - E.g., for **SQL Inject #0**:

```
SELECT * FROM users WHERE username='$username' AND password='$password'
```

Project 3: SQL Injection Tips

1. Identify **how the input is processed** on the server-side
   - E.g., for **SQL Inject #0**:

```
SELECT * FROM users WHERE username='$username' AND password='$password'
```

2. What **input fields** are under our control?

Project 3: SQL Injection Tips

1. Identify **how the input is processed** on the server-side
   - E.g., for **SQL Inject #0**:

```
SELECT * FROM users WHERE username='$username' AND password='$password'
```

2. What **input fields** are under our control?
   - The **$**`username` and **$**`password` fields

Project 3: SQL Injection Tips

1. Identify **how the input is processed** on the server-side
   - E.g., for **SQL Inject #0**:

```
SELECT * FROM users WHERE username='$username' AND password='$password'
```

2. What **input fields** are under our control?
   - The **$**`username` and **$**`password` fields

3. What is **the goal** of our SQL injection attack?

Project 3: SQL Injection Tips

1. Identify **how the input is processed** on the server-side
   - E.g., for **SQL Inject #0**:

```
SELECT * FROM users WHERE username='$username' AND password='$password'
```

2. What **input fields** are under our control?
   - The `$username` and `$password` fields

3. What is **the goal** of our SQL injection attack?
   - A SQL query that **logs us in as** “`victim`”

Project 3: SQL Injection Tips

1. Identify **how the input is processed** on the server-side
   - E.g., for **SQL Inject #0**:

```
SELECT * FROM users WHERE username='$username' AND password='$password'
```

2. What **input fields** are under our control?
   - The `$username` and `$password` fields

3. What is **the goal** of our SQL injection attack?
   - A SQL query that **logs us in as** “`victim`”

4. What **steps** are needed for our attack to work?

Project 3: SQL Injection Tips

1. Identify **how the input is processed** on the server-side
   - E.g., for **SQL Inject #0**:

```
SELECT * FROM users WHERE username='$username' AND password='$password'
```

2. What **input fields** are under our control?
   - The `$username` and `$password` fields
3. What is **the goal** of our SQL injection attack?
   - A SQL query that **logs us in as** "`victim`"
4. What **steps** are needed for our attack to work?
   1. Set `$username` to "`victim`"
   2. Set `$password` to their password

The correct **password** would log us in…

But **we do not know** the user's **password!**

Project 3: SQL Injection Tips

- **Solution:** craft a query that **closes-out unknowable fields**, **resolves to TRUE**

```
SELECT * FROM users WHERE username='$username' AND password='$password'
```

Example Attack:

```
... AND password='$password'
```

Project 3: SQL Injection Tips

- **Solution:** craft a query that **closes-out unknowable fields**, **resolves to TRUE**

```
SELECT * FROM users WHERE username='$username' AND password='$password'
```

Example Attack:

```
... AND password='$password'
```

```
... AND password=''
```

- **Closes-out** unknowable password

Project 3: SQL Injection Tips

- **Solution:** craft a query that **closes-out unknowable fields**, **resolves to TRUE**

```
SELECT * FROM users WHERE username='$username' AND password='$password'
```

Example Attack:

```
... AND password='$password'
```

```
... AND password='' OR '1'='1'
```

- **Closes-out** unknowable password
- `'1'='1'` always resolves **TRUE**

Project 3: SQL Injection Tips

- **Solution:** craft a query that **closes-out unknowable fields, resolves to TRUE**

```
SELECT * FROM users WHERE username='$username' AND password='$password'
```

Example Attack:

```
... AND password='$password'
```

```
... AND password='' OR '1'='1'
```

- **Closes-out** unknowable password
- `'1'='1'` always resolves **TRUE**

Example Attack:

```
... AND password='$password'
```

```
... AND password='foo'
```

Project 3: SQL Injection Tips

- **Solution:** craft a query that **closes-out unknowable fields**, **resolves to TRUE**

```
SELECT * FROM users WHERE username='$username' AND password='$password'
```

Example Attack:

```
... AND password='$password'
```

```
... AND password='' OR '1'='1'
```

- **Closes-out** unknowable password
- `'1'='1'` always resolves **TRUE**

Example Attack:

```
... AND password='$password'
```

```
... AND password='foo'
```

- Creates a **FALSE** string comparison

Project 3: SQL Injection Tips

- **Solution:** craft a query that **closes-out unknowable fields, resolves to TRUE**

```
SELECT * FROM users WHERE username='$username' AND password='$password'
```

Example Attack:

```
... AND password='$password'
```

```
... AND password='' OR '1'='1'
```

- **Closes-out** unknowable password
- `'1'='1'` always resolves **TRUE**

Example Attack:

```
... AND password='$password'
```

```
... AND password='foo' = ''
```

- Creates a **FALSE** string comparison

Project 3: SQL Injection Tips

- **Solution:** craft a query that **closes-out unknowable fields**, **resolves to TRUE**

```
SELECT * FROM users WHERE username='$username' AND password='$password'
```

Example Attack:

```
... AND password='$password'
```

```
... AND password='' OR '1'='1'
```

- **Closes-out** unknowable password
- `'1'='1'` always resolves **TRUE**

Example Attack:

```
... AND password='$password'
```

```
... AND password='foo' = ''
```

- Creates a **FALSE** string comparison
- But **FALSE == ''** ends up **TRUE**

Project 3: SQL Injection Tips

- **Solution:** craft a query that **closes-out unknowable fields, resolves to TRUE**

```
SELECT * FROM users WHERE username='$username' AND password='$password'
```

Example Attack:

```
... AND password='$password'
```

```
... AND password='' OR '1'='1'
```

- **Closes-out** unknowable password
- `'1'='1'` always resolves **TRUE**

Example Attack:

```
... AND password='$password'
```

```
... AND int(FALSE) == int('')
```

- Creates a **FALSE** string comparison
- But `FALSE == ''` ends up **TRUE**

Project 3: SQL Injection Tips

- **Solution:** craft a query that **closes-out unknowable fields, resolves to TRUE**

```
SELECT * FROM users ... assword='$password'
```

Example Atta... ...mple Attack:

```
... AND password='$...      ...sword='$password'
... AND password='...      ...(FALSE) == int('')
```

- **Closes-out** unknowa...
- '1'='1' always re...

...FALSE string comparison
...E == '' ends up TRUE

Key idea: identify how you can exploit **SQL's command syntax** and **queries that resolve TRUE**

Result: Attacker does not need to know **the victim's password**!

Project 3: SQL Injection Tips

- **Write-out** your query and **how the server processes it**
  - Are you **closing-out** fields? **Commenting-out** the line?

- **Trial-and-error** with different **TRUE-resolving queries**
  - Pay attention to what server tells you!
    - E.g., "`Incorrect username or password`" versus "`Error in MySQL query`"

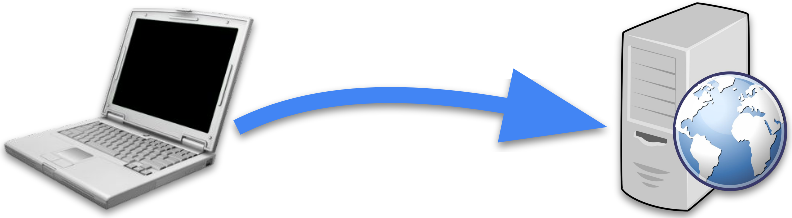
```
AND password='' OR '1'='1'
```

```
AND password='' OR '12345'
```

```
AND password='' = ''
```

Interacting with Web Applications

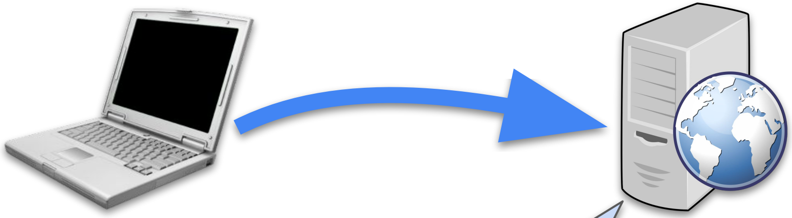
- **GET request:** parameters in **???**



Interacting with Web Applications

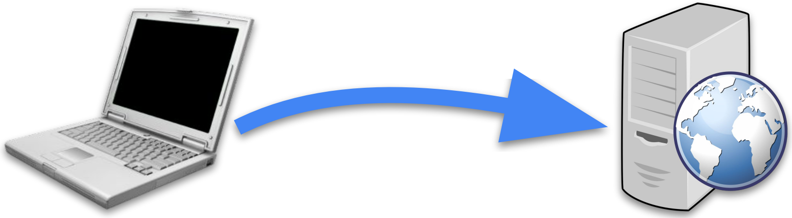
- **GET request:** parameters in **URL**

1 `www.bank.com/send.asp?to=snagy&amt=100`



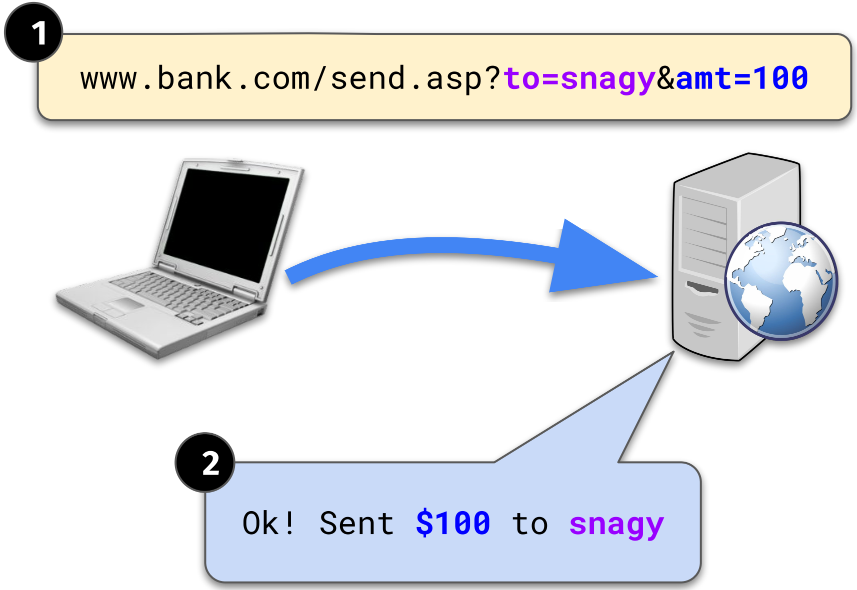
2 `Ok! Sent $100 to snagy`

- **POST request:** parameters in **???**

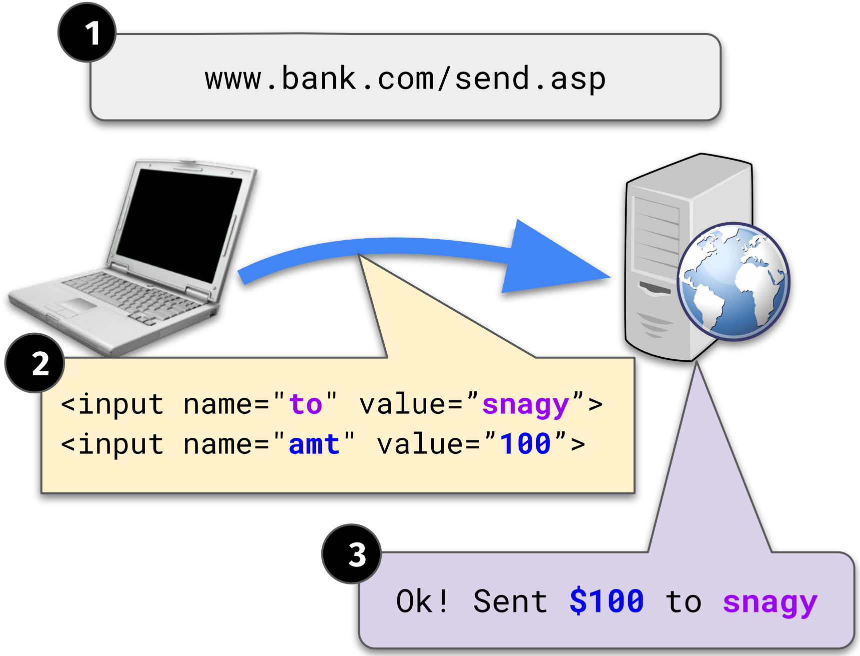


Interacting with Web Applications

- **GET request:** parameters in **URL**



- **POST request:** parameters in **body**

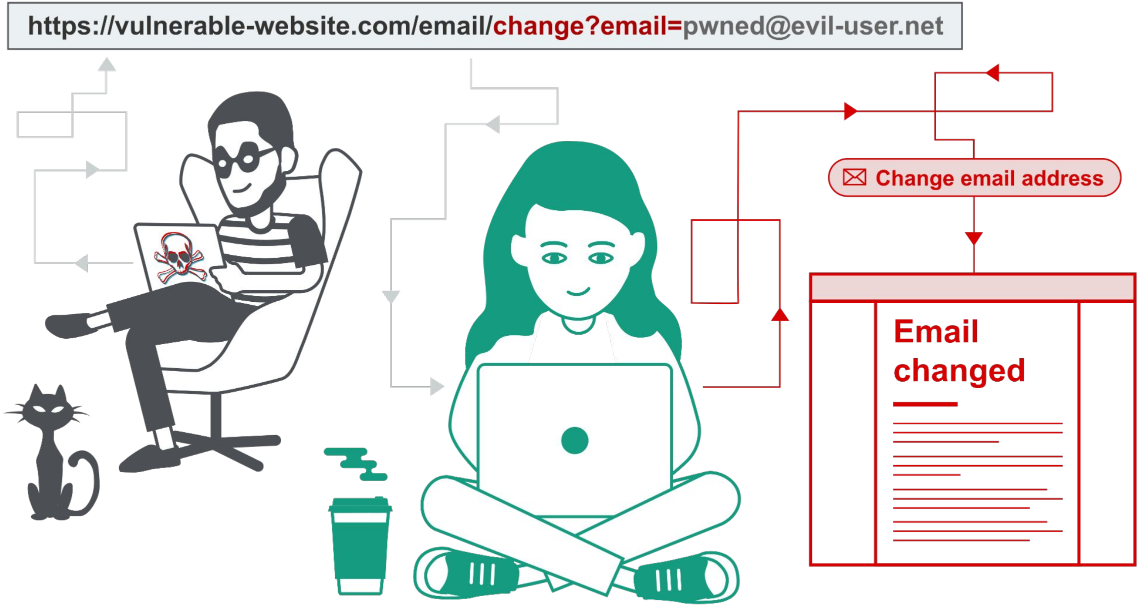


Cross-site Request Forgery (CSRF)

- **Attacker goal: ???**

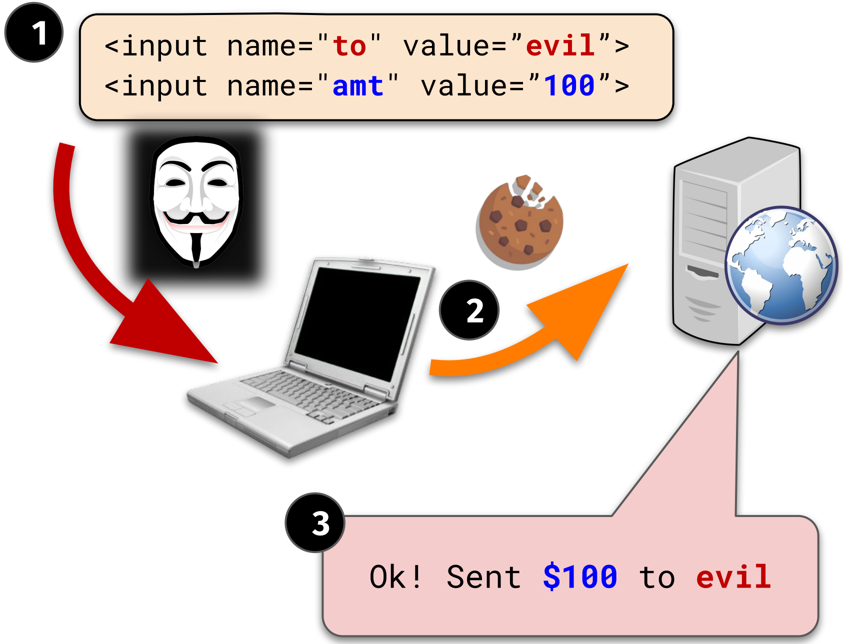
Cross-site Request Forgery (CSRF)

- **Attacker goal:** leverage **user's session** to **execute malicious commands**
  - Trick user into accessing **specially-crafted URLs (GET)** or **HTML pages (POST)**



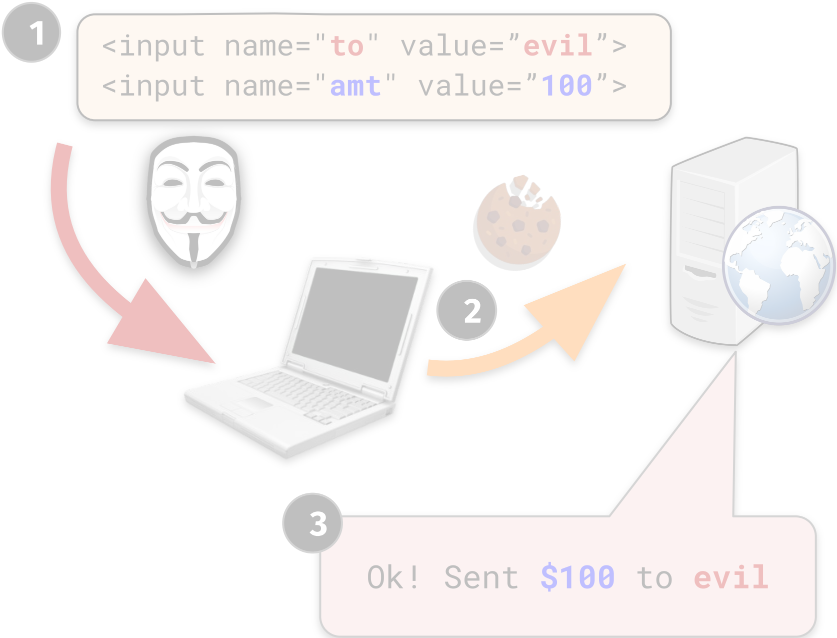
CSRF Attacks

- **POST-based CSRF (evil webpage)**

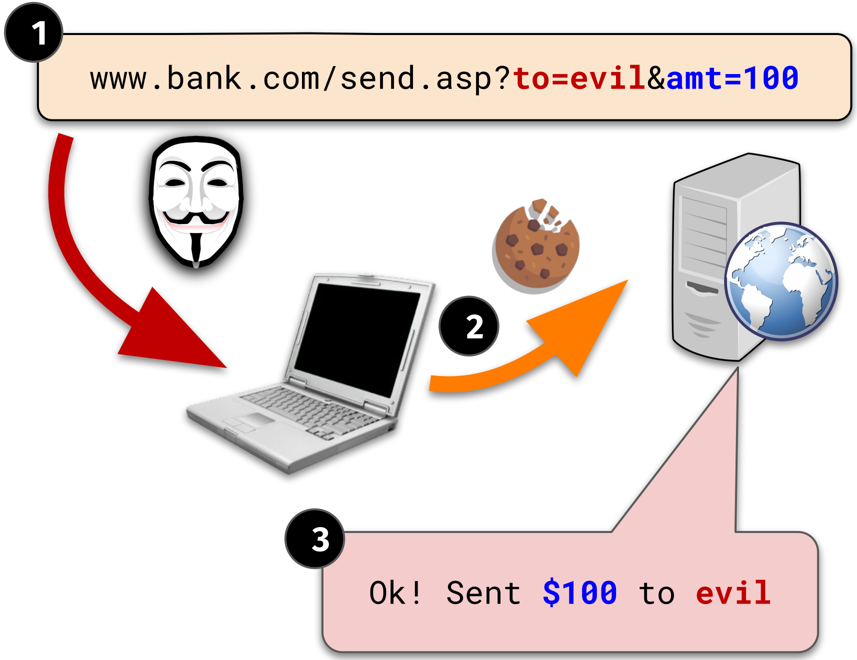


CSRF Attacks

- POST-based CSRF (evil webpage)



- GET-based CSRF (evil URL)



Interacting with Dynamic Web Applications

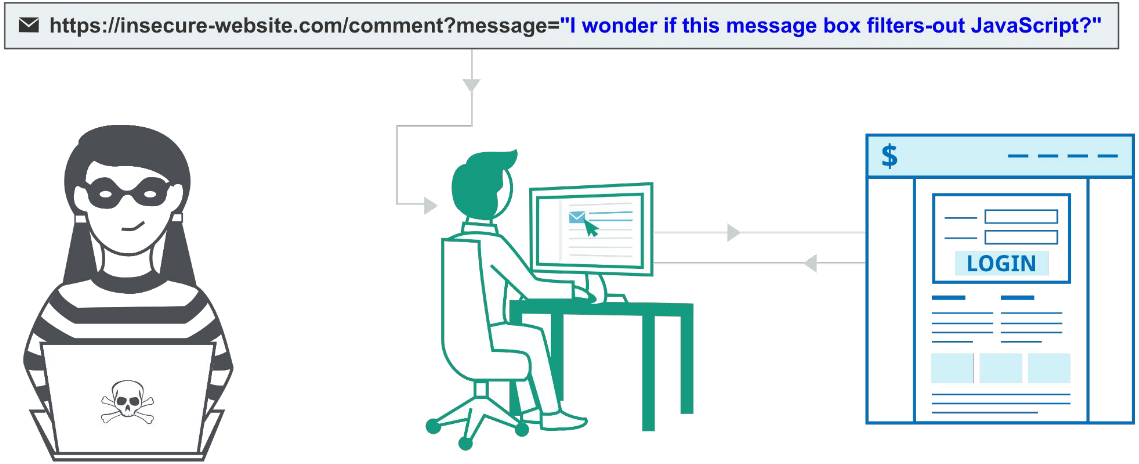
- **A powerful, popular web programming language**
  - Transmitted as **text**, rendered by **client's browser**
    - Can alter webpage contents, track events, read/set cookies, issue requests, read requests' replies, etc.

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

```
<img src="picture.gif"
onMouseOver="javascript:hello()">
```

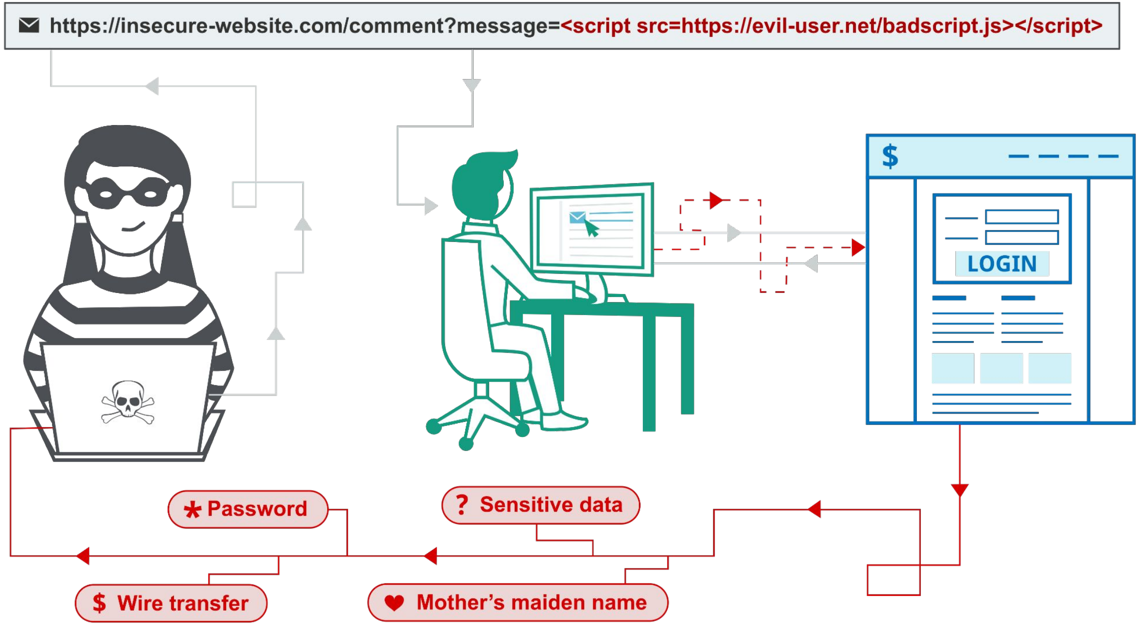
Cross-site Scripting (XSS)

- **Attacker goal: ???**



Cross-site Scripting (XSS)

- **Attacker goal:** submit **code as data** to website, get victim to **execute it**



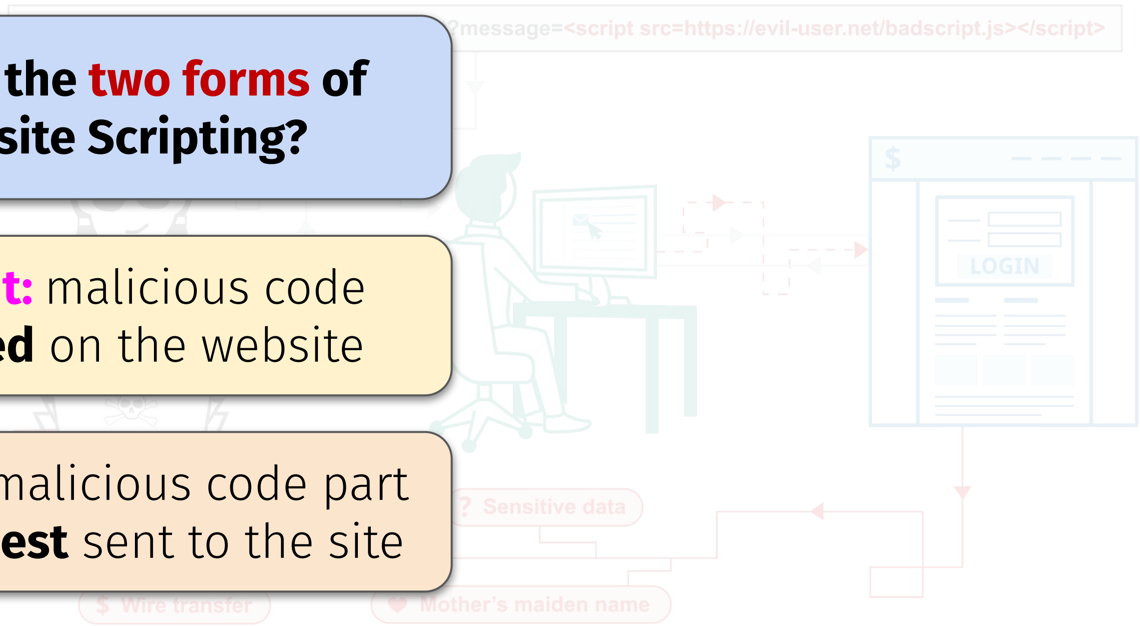
Cross-site Scripting (XSS)

- **Attacker goal:** submit **code as data** to website, get victim to **execute it**

What are the two forms of Cross-site Scripting?

Persistent: malicious code **embedded** on the website

Reflected: malicious code part of the **request** sent to the site



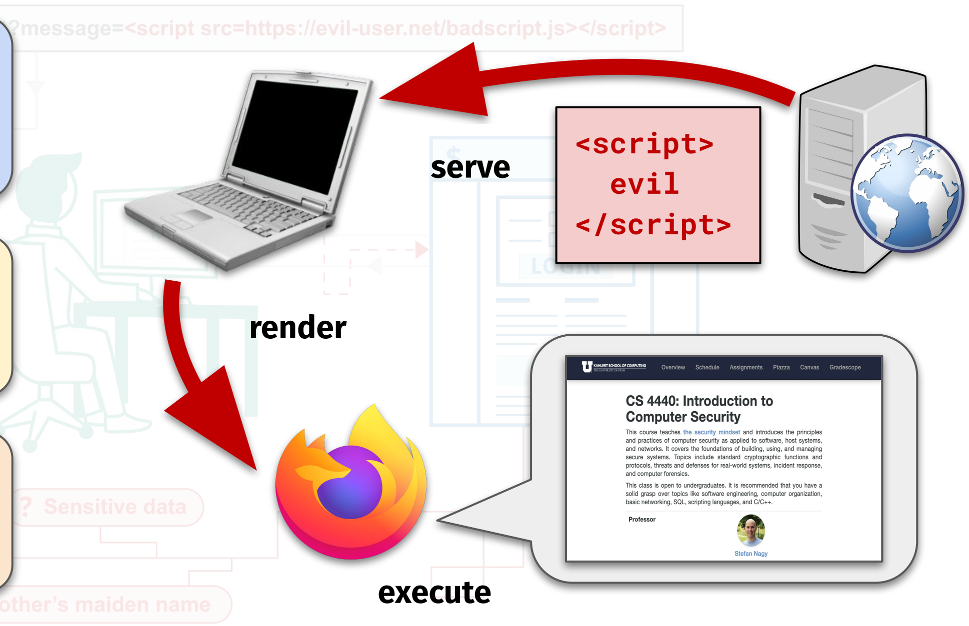
Cross-site Scripting (XSS)

- **Attacker goal:** submit **code as data** to website, get victim to **execute it**

What are the two forms of Cross-site Scripting?

Persistent: malicious code **embedded** on the website

Reflected: malicious code part of the **request** sent to the site



Project 3: Beginner CSRF & XSS Tips

- Understand how your target **takes input**
  - `LOGIN` page: **POST** requests
  - `SEARCH` page: **GET** requests

Project 3: Beginner CSRF & XSS Tips

- Understand how your target **takes input**
  - `LOGIN` page: **POST** requests
  - `SEARCH` page: **GET** requests

- Set up your **attack parameters** accordingly
  - Desired username, password, method, etc.
  - Template makes this easy—use the **form**!

Project 3: Beginner CSRF & XSS Tips

- Understand how your target **takes input**
  - **LOGIN** page: **POST** requests
  - **SEARCH** page: **GET** requests
- Set up your **attack parameters** accordingly
  - Desired username, password, method, etc.
  - Template makes this easy—use the **form**!
- **BSF 1–3:** exploiting the **SEARCH** page
  - **Weakness:** improperly filters search terms...
    - Can we leverage this to inject code?

Example SEARCH Input:

```
<input name="q" value="
    <script>
        alert(0);
    </script>
">
```

Project 3: Beginner CSRF & XSS Tips

- Understand how your target **takes input**
  - LOGIN page: POST requests
  - SEARCH page: GET requests
- Set up your **attack parameters** accordingly
  - Desired username, password, method, etc.
  - Template makes this easy—use the **form**!
- **BSF 1–3:** exploiting the SEARCH page
  - **Weakness:** improperly filters search terms…
    - Can we leverage this to inject code?
- Test out **simple payloads first**, then move on to building your **full attacks**!

Example SEARCH Input:

```
<input name="q" value="
    <script>
        alert(0);
    </script>
">
```

Project 3: Advanced XSS Tips

- **Builds off your skills** from **Part 2**
  - Master those first before attempting these!

Project 3: Advanced XSS Tips

- **Builds off your skills** from **Part 2**
  - Master those first before attempting these!
- **Part 2:** **page-reflected** XSS
  - Attack embedded in a **static page**

```
<input name="q" value="
  <script>alert(0);</script>
">
```

Project 3: Advanced XSS Tips

- **Builds off your skills** from **Part 2**
  - Master those first before attempting these!
- **Part 2: page-reflected** XSS
  - Attack embedded in a **static page**
- **Part 3: URL-reflected** XSS
  - Attack embedded in a **URL**

```
<input name="q" value="
  <script>alert(0);</script>
">
```

```
http://cs4440.eng.utah.edu/project3
/search?q=%3Cscript%3E...
```

Project 3: Advanced XSS Tips

- **Builds off your skills** from **Part 2**
  - Master those first before attempting these!
- **Part 2:** **page-reflected** XSS
  - Attack embedded in a **static page**
- **Part 3:** **URL-reflected** XSS
  - Attack embedded in a **URL**

```
<input name="q" value="
  <script>alert(0);</script>
">
```

```
http://cs4440.eng.utah.edu/project3
/search?q=%3Cscript%3E...
```

- Test your attack by first embedding it in an **HTML page**, **then move to a URL**!

Project 3: Advanced XSS Tips

- **Builds off your skills** from **Part 2**
  - Master those first before attempting these!
- **Part 2:** **page-reflected** XSS
  - Attack embedded in a **static page**

```
<input name="q" value="
  <script>alert(0);</script>
">
```

- **Part 3:** **URL-reflected** XSS
  - Attack embedded in a **URL**

```
http://cs4440.eng.utah.edu/project3
/search?q=%3Cscript%3E...
```

- Test your attack by first embedding it in an **HTML page**, **then move to a URL**!
  - **Hint:** write a program to convert **JavaScript code characters** to a **URL-friendly encoding**
    - See https://www.w3schools.com/tags/ref_urlencode.ASP

Questions?

This time on CS 4440…

Browser-side Web Security
Isolation and Sandboxing
The Same-origin Policy
HTTPS, SSL, and TLS

Principles of Web Security

- **Privacy**
  - ???

Principles of Web Security

- **Privacy**
  - Malicious websites should not be able to **spy on me or my activities** online
- **Integrity**
  - **???**

Principles of Web Security

- **Privacy**
  - Malicious websites should not be able to **spy on me or my activities** online
- **Integrity**
  - Malicious websites should not be able to violate the **integrity of my computer** or my **information on other websites**
- **Confidentiality**
  - **???**

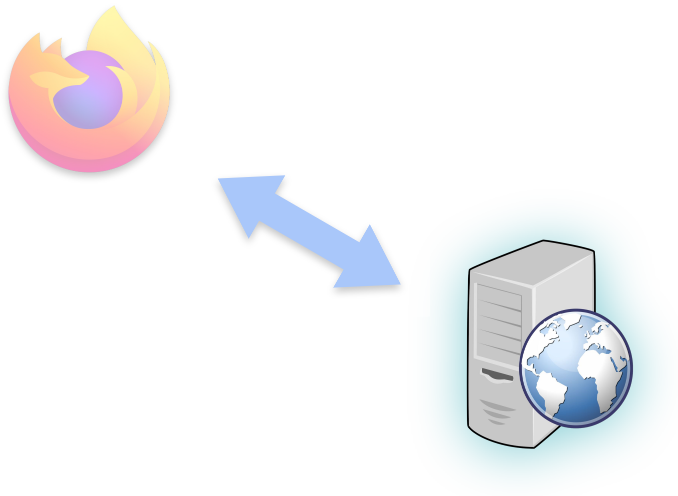
Principles of Web Security

- **Privacy**
  - Malicious websites should not be able to **spy on me or my activities** online
- **Integrity**
  - Malicious websites should not be able to violate the **integrity of my computer** or my **information on other websites**
- **Confidentiality**
  - Malicious websites should not be able to **learn confidential information** from my computer or from other websites

Web Security Risks

- **Risk #1:** `TotallySafeSite.com` should keep my **information secure**
  - E.g., database breaches, stolen login credentials, disgruntled employee, etc.

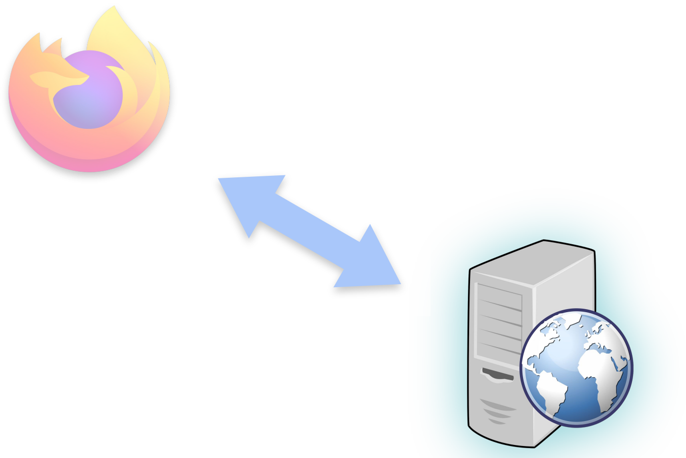
- **Defenses: server-side security**
  - **???**



Web Security Risks

- **Risk #1:** `TotallySafeSite.com` should keep my **information secure**
  - E.g., database breaches, stolen login credentials, disgruntled employee, etc.

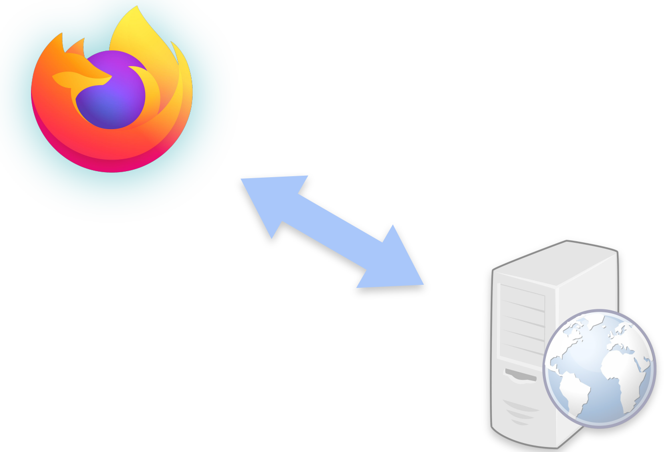
- **Defenses: server-side security**
  - Not storing info in plaintext
  - Principle of Least Privilege
  - Multi-factor authentication
  - Fix all server security bugs



Web Security Risks

- **Risk #2** visiting `TotallySafeSite.com` may access my **files and programs**
  - E.g., install malware, read sensitive information, alter local files, etc.

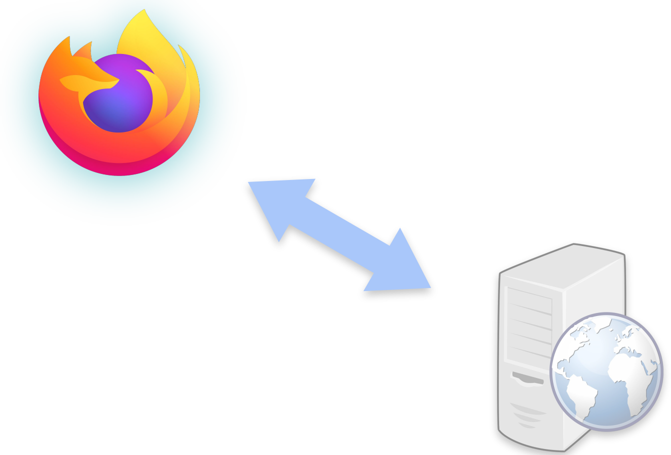
- **Defenses: browser-side security**
  - **???**



Web Security Risks

- **Risk #2** visiting `TotallySafeSite.com` may access my **files and programs**
  - E.g., install malware, read sensitive information, alter local files, etc.

- **Defenses: browser-side security**
  - Fix browser security bugs
  - Enable automatic updates
  - Privilege separation
  - **Sandbox all code** (e.g., JavaScript)



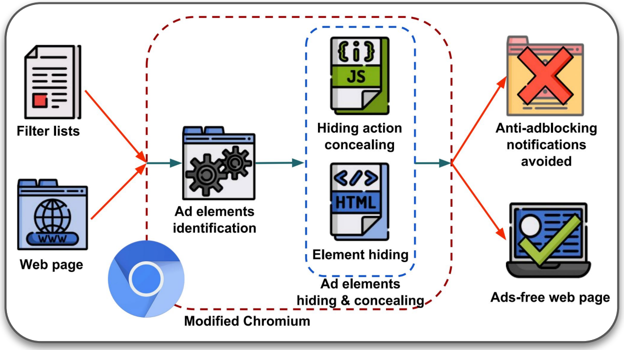
Client-side Web Defenses

Browser Sandboxing Techniques

- **General Process Sandboxing**
  - See Week 6B's lecture

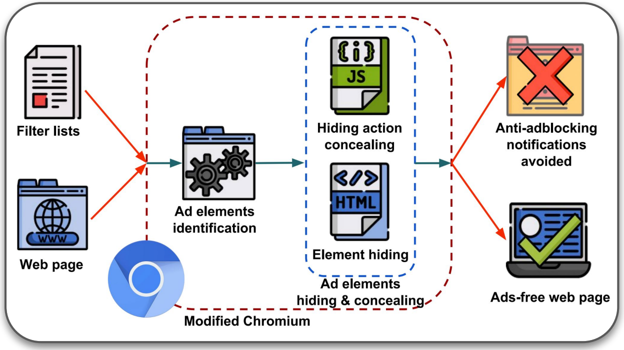
Browser Sandboxing Techniques

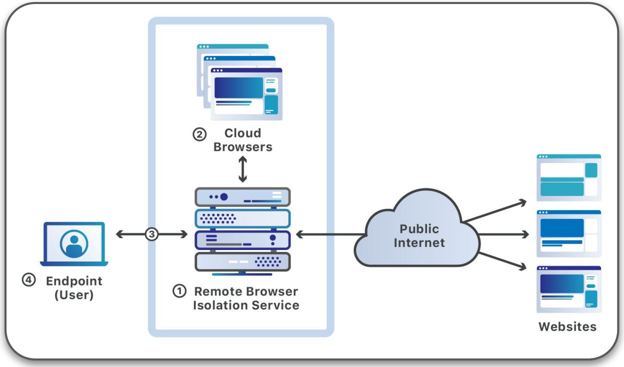
- **General Process Sandboxing**
  - See Week 6B's lecture
- **DOM Mirroring**
  - Filter-out unsafe DOM elements
  - E.g., anti-adblocking functionality



Browser Sandboxing Techniques

- **General Process Sandboxing**
  - See Week 6B's lecture
- **DOM Mirroring**
  - Filter-out unsafe DOM elements
  - E.g., anti-adblocking functionality
- **Pixel Streaming / Remote Browser**
  - Render page remotely (e.g., container)
  - **Pixel Reconstruction:** client only gets the final pixel array, not the application code
  - **Remote Browser:** all interaction encrypted





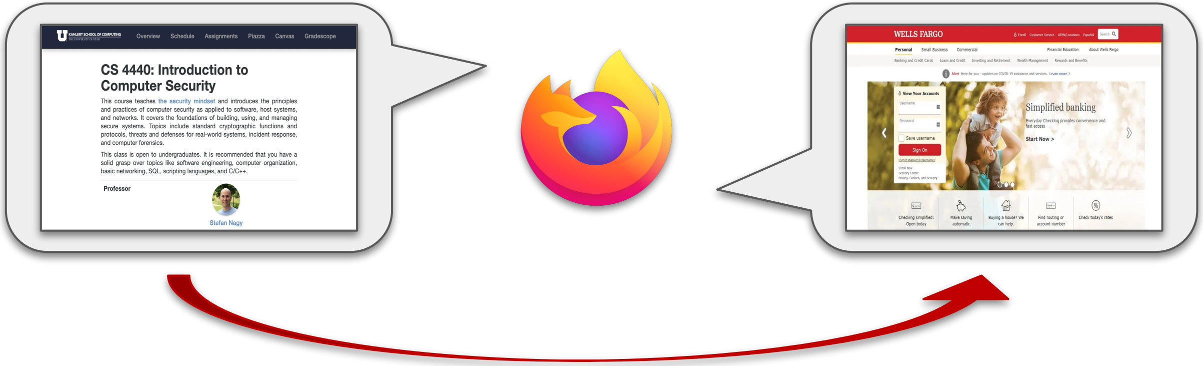
Web Security Risks

- **Risk #3:** `TotallySafeSite.com` tracks my info/interaction **with other sites**
  - E.g., spying on my GMail emails, purchasing things with my Amazon, etc.

- **Defenses: maintain site isolation**
  - Same-origin Policy
  - Multi-process browsing

Same-origin Policy

- **Goal:** make sure that scripts **don't abuse** the power of **JavaScript**

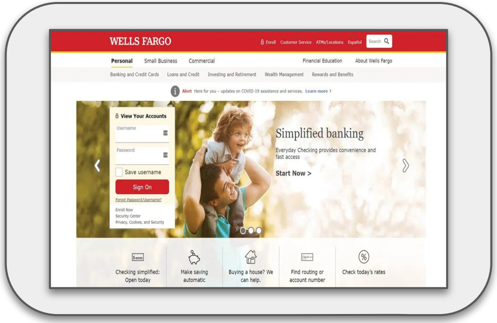


- Scripts from **CS 4440 website shouldn't** read **cookies** on **FellsWargo site**
  - … or alter FellsWargo site's **layout**, or its read **keystrokes** typed by user to FellsWargo site

Same-origin Policy

- **Origin** = the **protocol** + the **hostname**

- **Example:** `http://www.cs.utah.edu/class...`
  - **Protocol:** HTTP
  - **Hostname:** www.cs.utah.edu

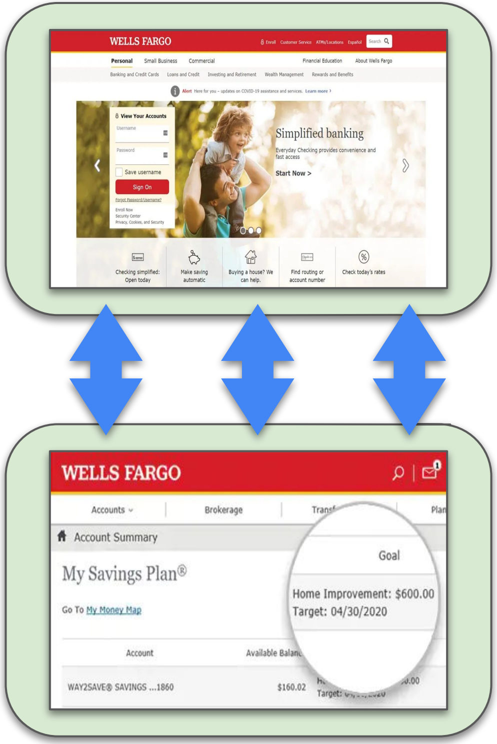


Same-origin Policy

- **Origin** = the **protocol** + the **hostname**

- **Example:** `http://www.cs.utah.edu/class...`
  - **Protocol:** HTTP
  - **Hostname:** www.cs.utah.edu

- JavaScript from one page can **read**, **change**, and **interact freely** with all pages from same origin



Same-origin Policy

- **Origin** = the **protocol** + the **hostname**

- **Example:** `http://www.cs.utah.edu/class...`
  - **Protocol:** HTTP
  - **Hostname:** `www.cs.utah.edu`

- JavaScript from one page can **read**, **change**, and **interact freely** with **all pages from same origin**
  - Content cannot be accessed by **scripts of different origin**

Same-origin Policy

- Restricts access to content from the same **origin** (**protocol** + **host**)

Same-origin Policy

- Restricts access to content from the same **origin** (**protocol** + **host**)
- Try the following, comparing to `http://example.com/home.html`

Candidate Request	SOP Result	Explanation
`https://example.com/index.html`		

For http://example.com/home.html, does https://example.com/index.html violate the SOP?

Passes SOP

0%

Violates SOP

0%

None of the above

0%

Same-origin Policy

- Restricts access to content from the same **origin** (**protocol** + **host**)
- Try the following, comparing to `http://example.com/home.html`

Candidate Request	SOP Result	Explanation
`https://example.com/index.html`	**FAIL**	Different protocol (https)
`http://example.com/dir/other.html`		
`https://example.com/dir/inner/index.html`		
`http://example.com/dir/first/out/home.html`		
`http://en.example.com/dir/other.html`		

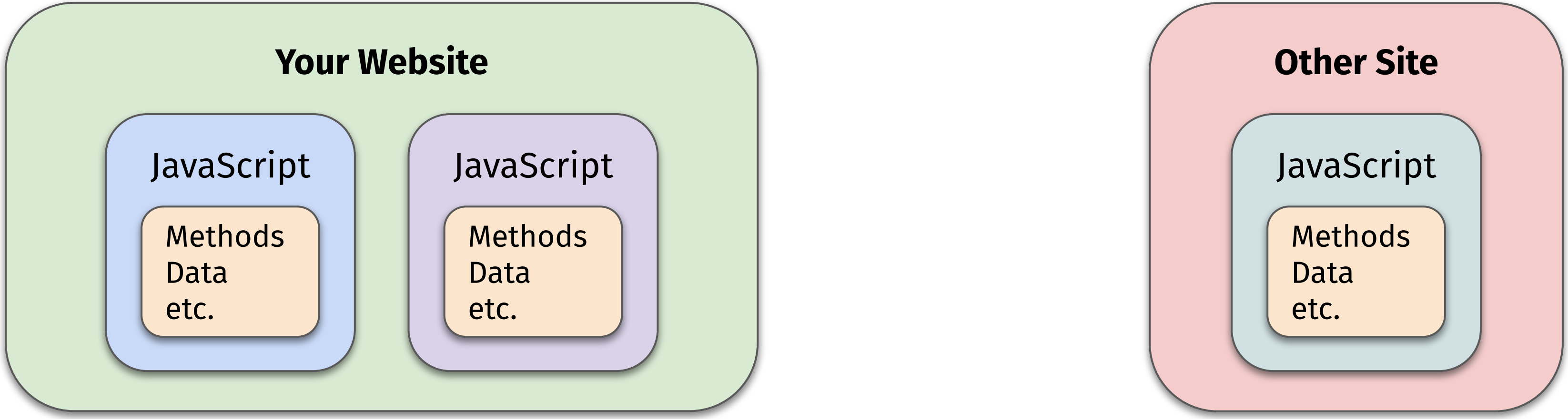
Same-origin Policy

- Restricts access to content from the same **origin** (**protocol** + **host**)
- Try the following, comparing to `http://example.com/home.html`

Candidate Request	SOP Result	Explanation
`https://example.com/index.html`	**FAIL**	Different protocol (https)
`http://example.com/dir/other.html`	**PASS**	Same protocol, same host
`https://example.com/dir/inner/index.html`	**FAIL**	Different protocol (https)
`http://example.com/dir/first/out/home.html`	**PASS**	Same protocol, same host
`http://en.example.com/dir/other.html`	**FAIL**	Different host (en)

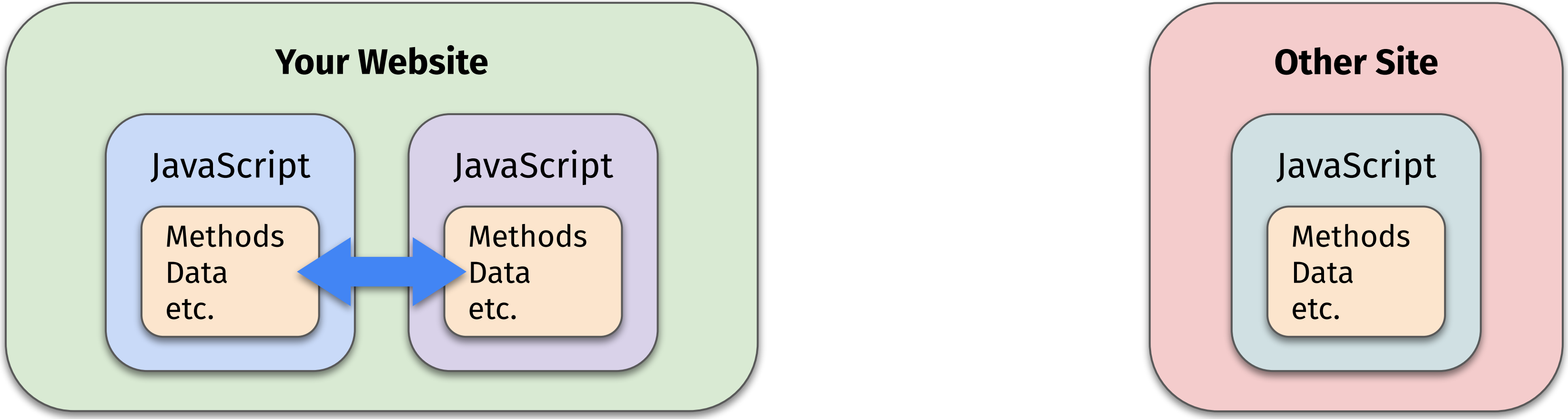
Same-origin Policy

- Implementation: **tagged sandboxing**



Same-origin Policy

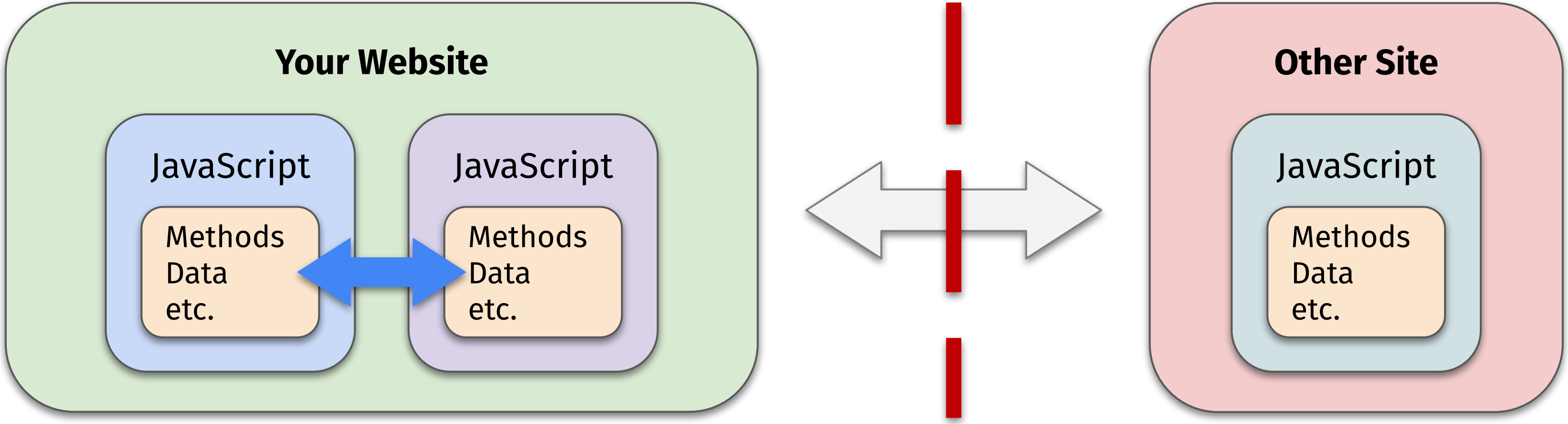
- Implementation: **tagged sandboxing**



  - Scripts within **same origin** can interface with each other

Same-origin Policy

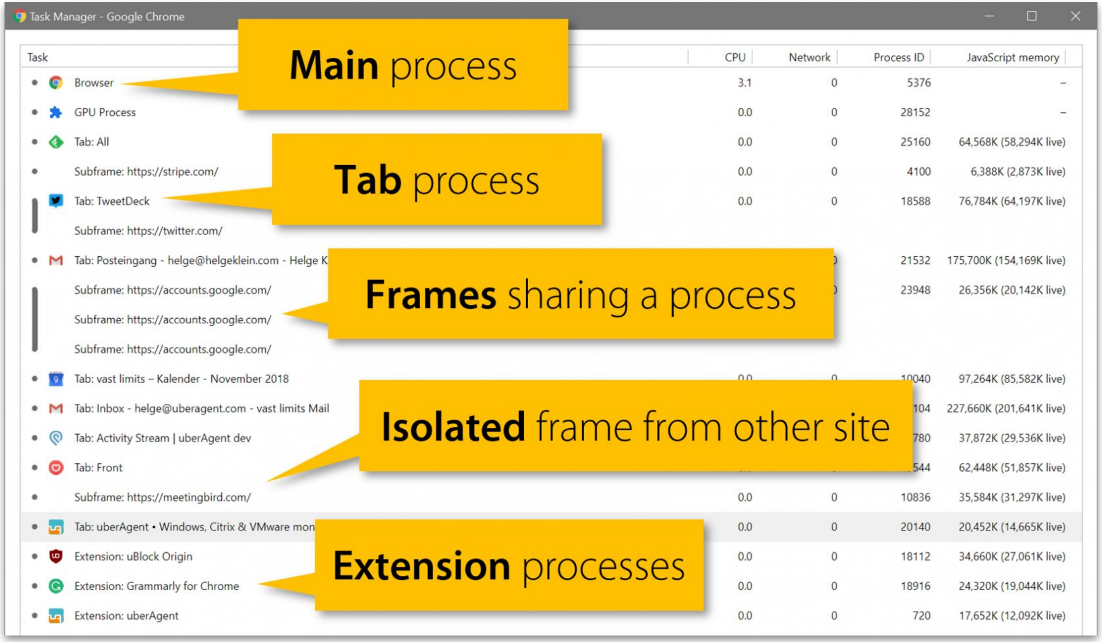
- Implementation: **tagged sandboxing**



- Scripts within **same origin** can interface with each other
- Scripts from **different origins** are completely blocked

Multi-process Browsing

- **Idea:** isolate “tabs” into **distinct processes**

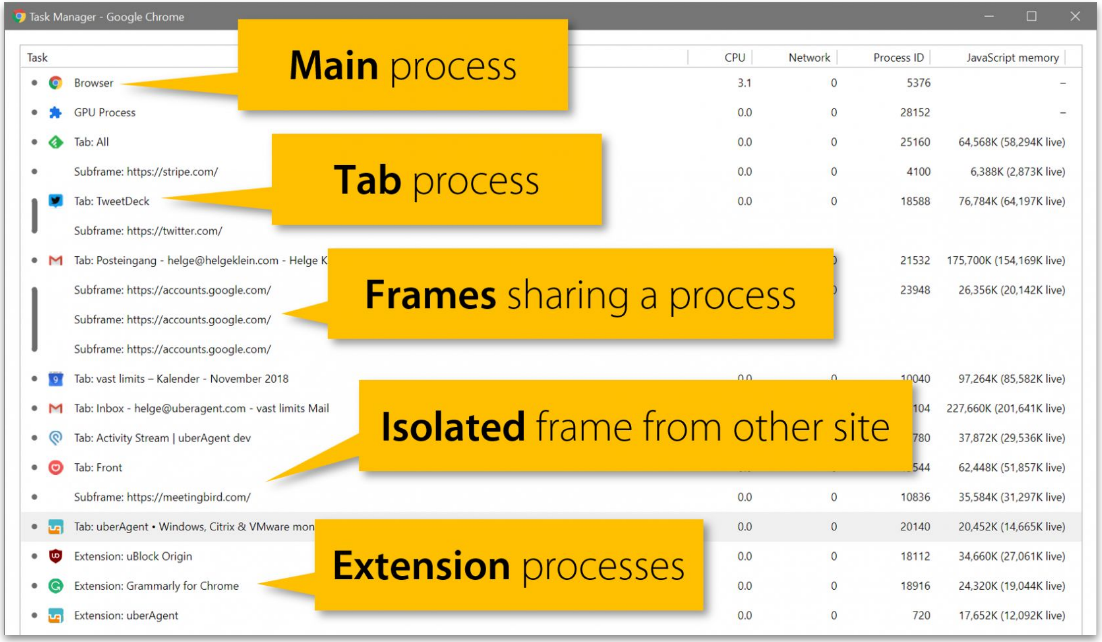


Multi-process Browsing

- **Idea:** isolate “tabs” into **distinct processes**
  - **Site-level isolation!**
  - Piggyback off of MMU

- **Most browsers do this**
  - Chrome
  - Firefox
  - Etc.

- **Downside: ???**

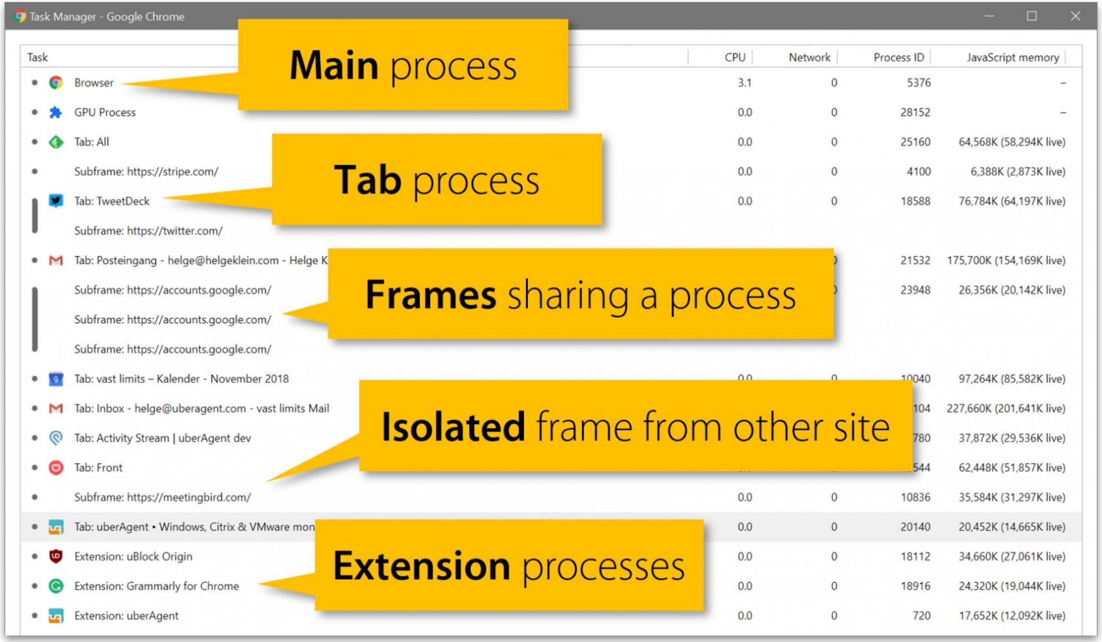


Multi-process Browsing

- **Idea:** isolate "tabs" into **distinct processes**
  - **Site-level isolation!**
  - Piggyback off of MMU

- **Most browsers do this**
  - Chrome
  - Firefox
  - Etc.

- **Downside: performance**
  - Lots of open tabs leads to lots of running processes!



Questions?

Secure Web Communication

Principles of Secure Web Communication

- **Authentication**
  - **???**

Principles of Secure Web Communication

- **Authentication**
  - The client must be able to verify that it is **talking to the desired server**
- **Integrity**
  - **???**

Principles of Secure Web Communication

- **Authentication**
  - The client must be able to verify that it is **talking to the desired server**
- **Integrity**
  - Data transmitted between client and server **must not be attacker-modifiable**
- **Confidentiality**
  - **???**

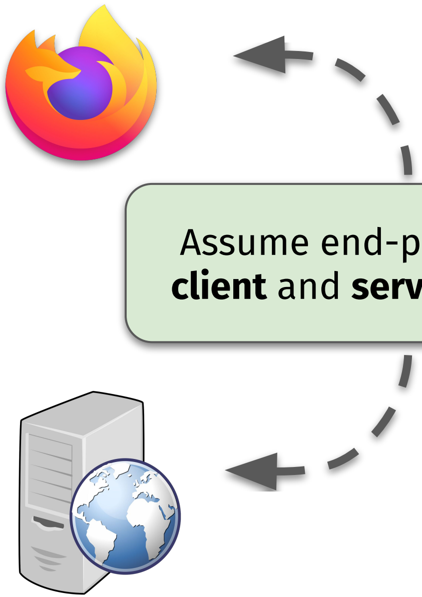
Principles of Secure Web Communication

- **Authentication**
  - The client must be able to verify that it is **talking to the desired server**
- **Integrity**
  - Data transmitted between client and server **must not be attacker-modifiable**
- **Confidentiality**
  - Data transmitted between the client and server **must not be attacker-visible**

Principles of Secure Web Communication

- **Authentication**
  - The client must be able to verify that it is talking to the desired server
- **Integrity**
  - Data transmitted between client and server must not be attacker-modifiable
- **Confidentiality**
  - Data transmitted between the client and server must not be attacker visible

Assumptions:



Assume end-points (the **client** and **server**) **secure**

Principles of Secure Web Communication

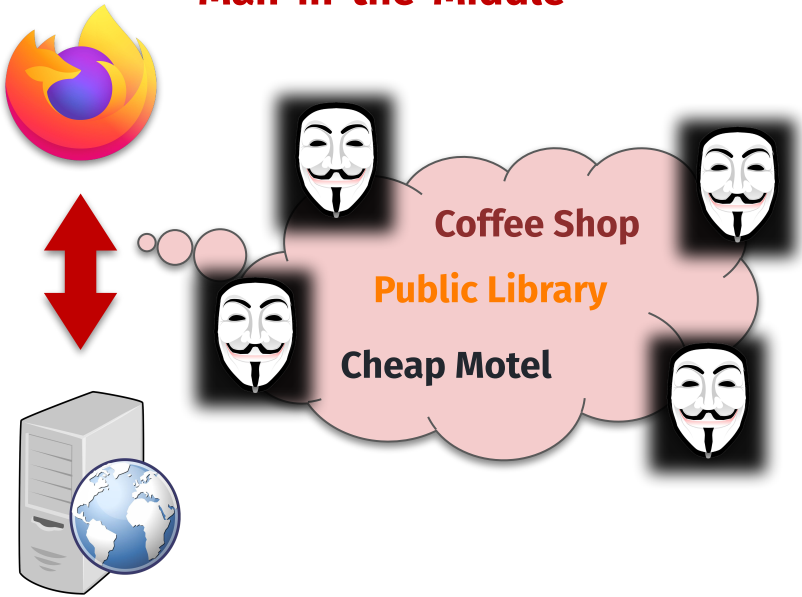
- **Authentication**
  - The client must be able to verify that it is talking to the desired server
- **Integrity**
  - Data transmitted between client and server must not be attacker-modifiable
- **Confidentiality**
  - Data transmitted between the client and server must not be attacker visible

Threat Model:

Principles of Secure Web Communication

- **Authentication**
  - The client must be able to verify that it is talking to the desired server
- **Integrity**
  - Data transmitted between client and server must not be attacker-modifiable
- **Confidentiality**
  - Data transmitted between the client and server must not be attacker visible

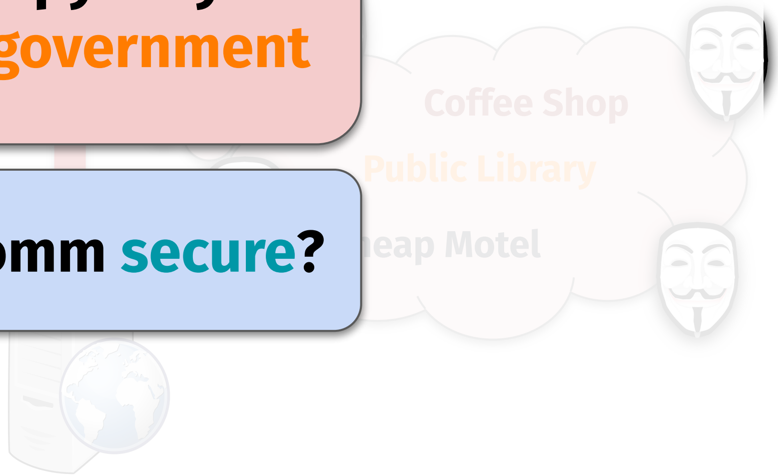
Threat Model:
Man-in-the-Middle



Principles of Secure Web Communication

- **Authentication**
  - The client must be able to verify that it is talking
- **Integrity**
  - Data tran… server must not be attacker-modifiable
- **Confidentia…**
  - Data tran… and server must not be attacker visible

Threat Model:
Man-in-the-Middle



Parties that are trying to **spy on you:**
Hackers, **your boss**, **the government**

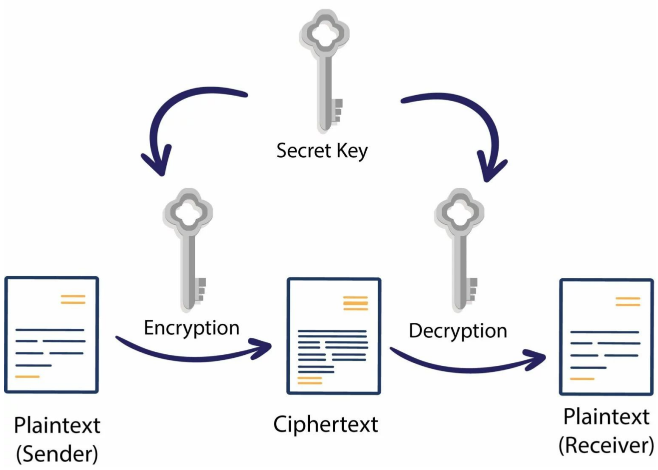
How can we make **web comm secure?**

Crypto to the rescue!

- **Symmetric Crypto:**

Crypto to the rescue!

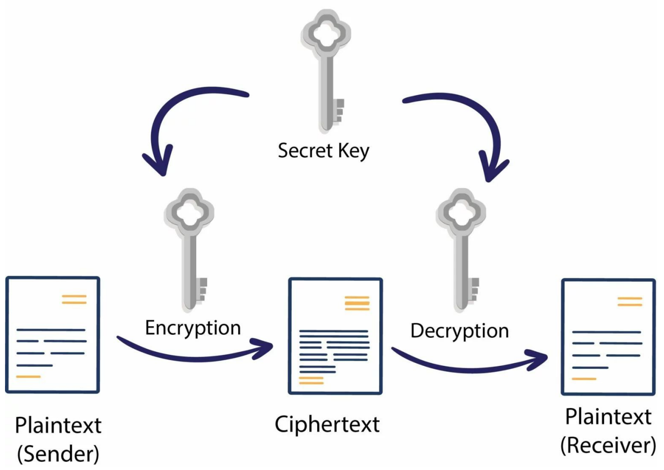
- **Symmetric Crypto:**



- **Problem: ???**

Crypto to the rescue!

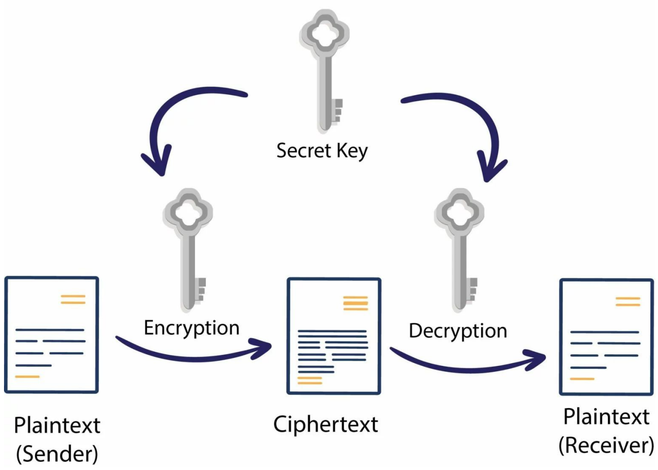
- **Symmetric Crypto:**



- **Problem: pre-sharing entire key**
  - If intercepted, whole scheme ruined!

Crypto to the rescue!

- **Symmetric Crypto:**

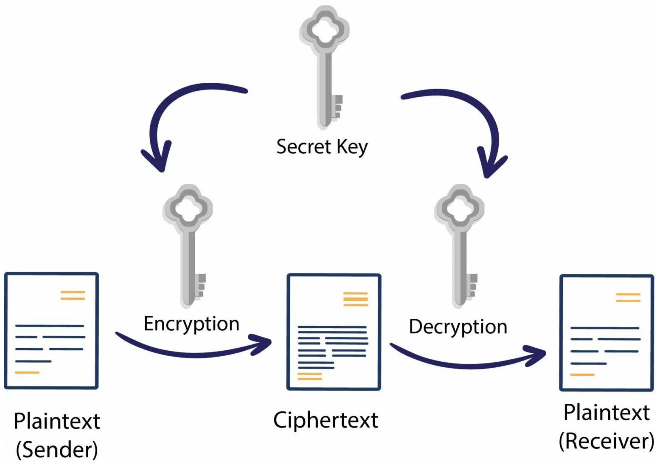


- **Problem: pre-sharing entire key**
  - If intercepted, whole scheme ruined!

- **Public-key Crypto:**

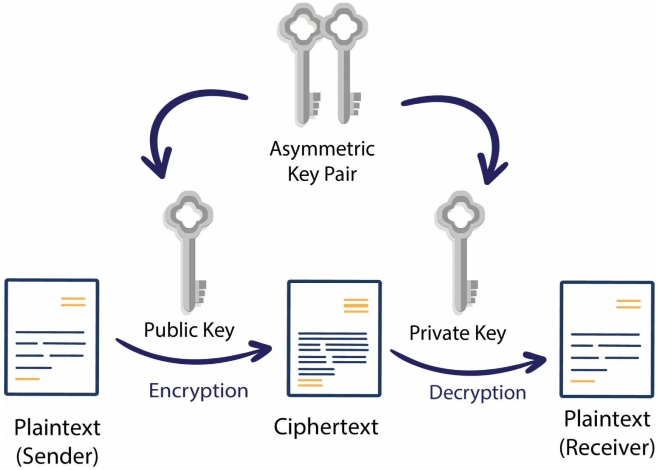
Crypto to the rescue!

- **Symmetric Crypto:**



- **Problem: pre-sharing entire key**
  - If intercepted, whole scheme ruined!

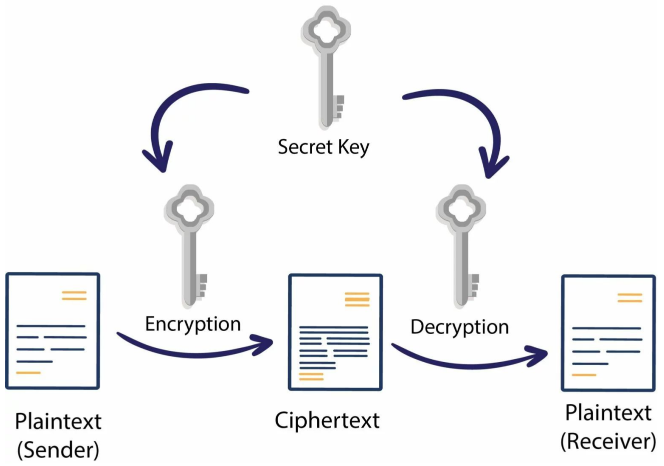
- **Public-key Crypto:**



- **Problem: ???**

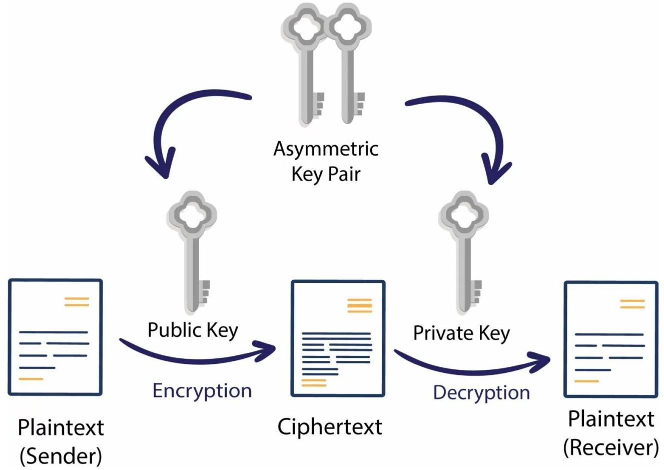
Crypto to the rescue!

- **Symmetric Crypto:**



- **Problem: pre-sharing entire key**
  - If intercepted, whole scheme ruined!

- **Public-key Crypto:**



- **Problem: lack of pre-authentication**
  - Is Bob's key really from the real Bob?

Crypto to the rescue!

- **Symmetric Crypto:**

- **Public-key Crypto:**

Parties that are trying to **spy on you:**
Hackers, **your boss**, **the government**

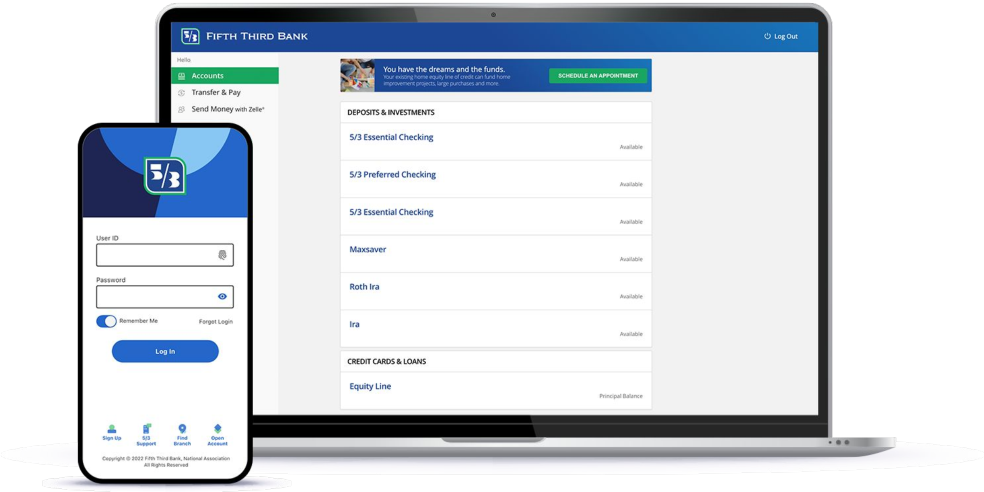
How can we **overcome pre-auth?**

- **Problem: pre-sharing entire key**
  - If intercepted, whole scheme ruined!
- **Problem: lack of pre-authentication**
  - Is Bob's key really from the real Bob?

HTTPS: HTTP over TLS

Recap: HyperText Transfer Protocol (HTTP)

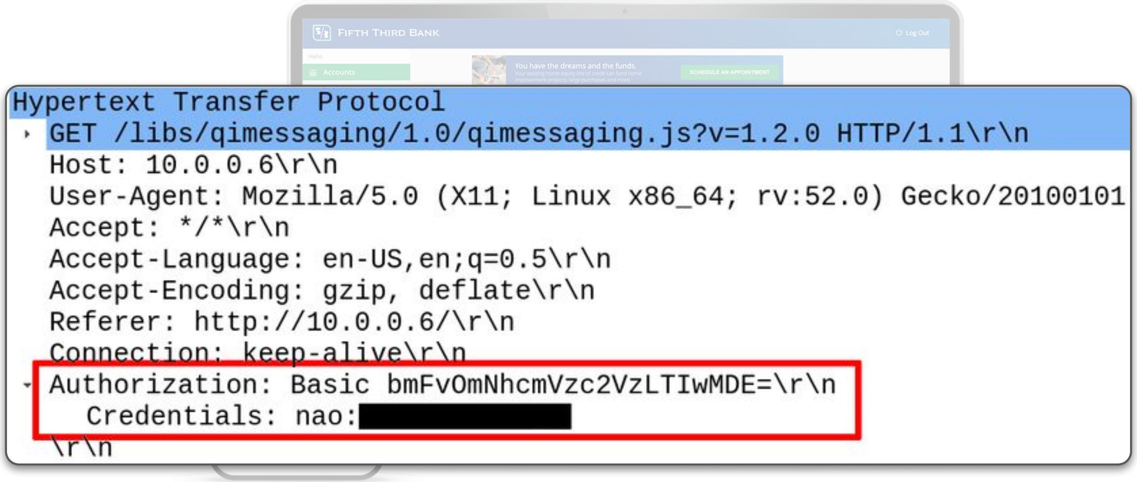
- **Protocol for transmitting hypermedia documents** (e.g., web pages)
  - Widely used
  - **Simple**
  - **Unencrypted**



Recap: HyperText Transfer Protocol (HTTP)

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

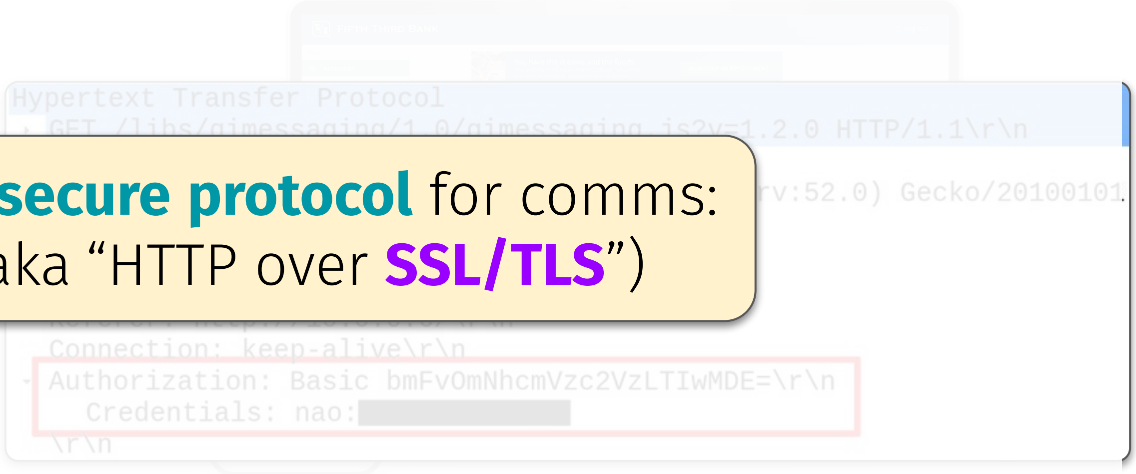
Problem: no way of keeping data hidden from **prying eyes**!



Recap: HyperText Transfer Protocol (HTTP)

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

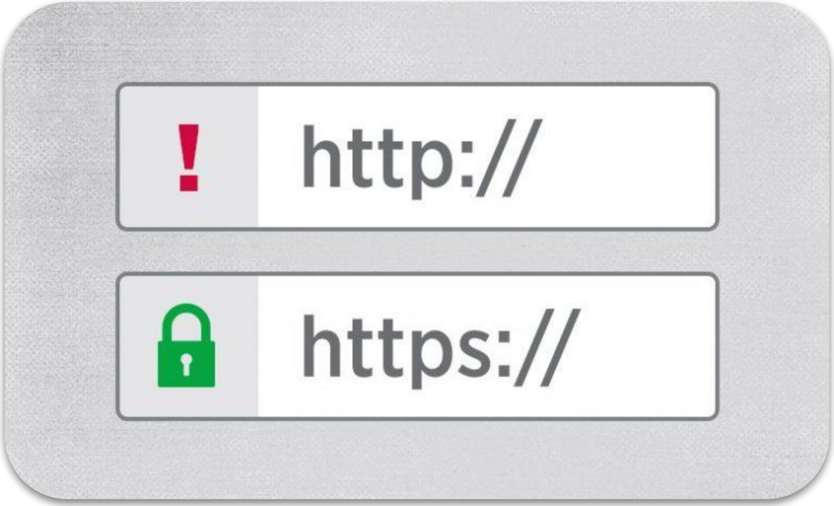
Problem
keeping d
from **prying eyes**!



We need a **secure protocol** for comms:
HTTPS (aka “HTTP over **SSL/TLS**”)

SSL and TLS

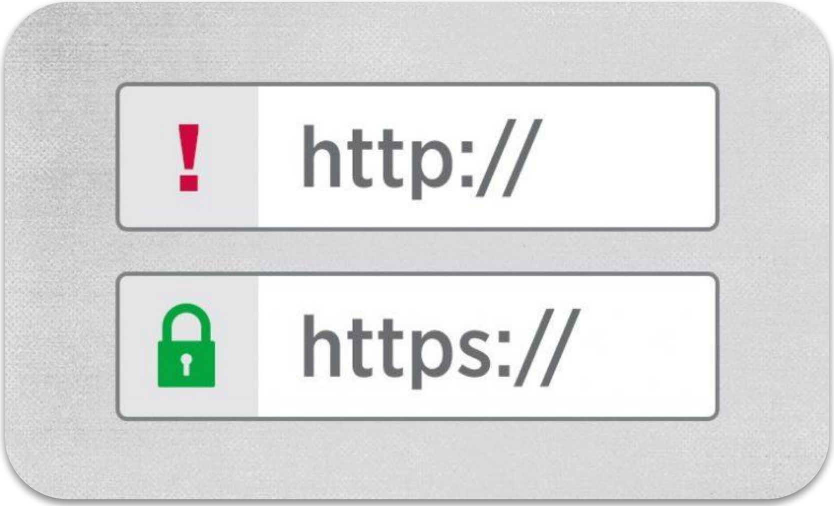
- The **physical protocols** by which **HTTPS public-key encryption** works



SSL and TLS

- The **physical protocols** by which **HTTPS public-key encryption** works

- **SSL (Secure Socket Layer)**
  - Developed by Netscape
  - Obsolete—stop using it!

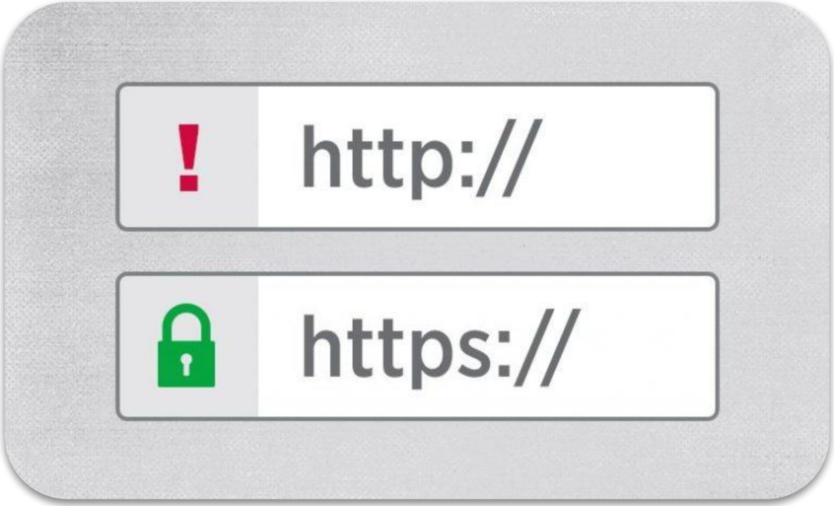


SSL and TLS

- The **physical protocols** by which **HTTPS public-key encryption** works

- **SSL (Secure Socket Layer)**
  - Developed by Netscape
  - Obsolete—stop using it!

- **TLS (Transport Layer Security)**
  - Successor to SSL
  - Versions 1.0, 1.1, 1.2, 1.3
  - **Current IETF approved standard**



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

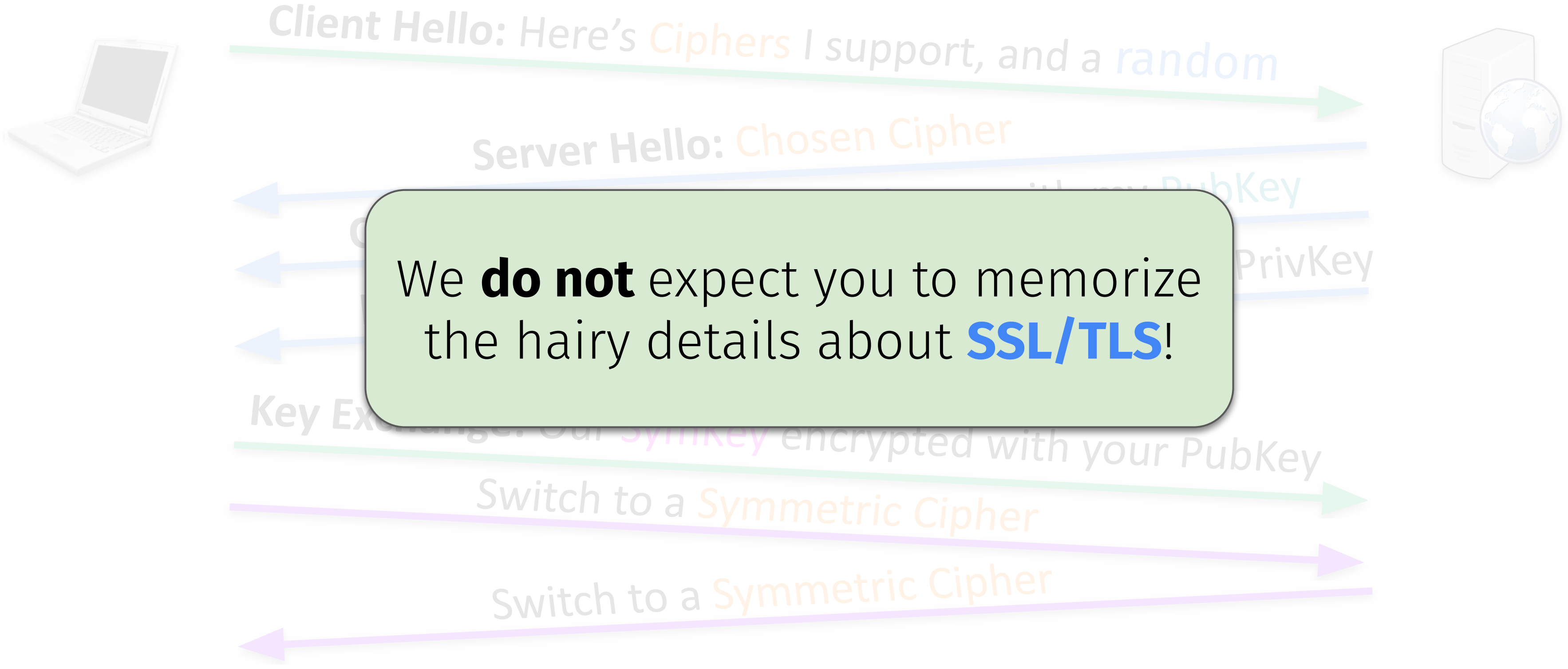
Here's your random back encrypted with my PrivKey

Key Exchange: Our SymKey encrypted with your PubKey

Switch to a Symmetric Cipher

Switch to a Symmetric Cipher

The TLS Handshake



We **do not** expect you to memorize the hairy details about **SSL/TLS**!

Higher-level TLS Handshake

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

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**."

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**."

Handling Pre-authentication

- A **trusted authority vouches** that a certain public key belongs to a particular site
  - Format called x.509 (complicated)
- Browsers ship with public keys for large number of trusted **Certificate Authorities**
- **Important fields:**
  - Common Name (CN) (e.g., *.google.com)
  - Expiration Date (e.g., 2 years from now)
  - **Subject's Public Key**
  - Issuer (e.g., Verisign)
  - Issuer's signature
- **Common Name field**
  - Explicit name, e.g. cs.utah.edu
  - Or wildcard, e.g. *.utah.edu

Handling Pre-authentication

- A **trusted authority vouches** that a certain public key belongs to a particular site
  - Format called x.509 (complicated)
- Browsers ship with public keys for large number of trusted **Certificate Authorities**
- **Important fields:**
  - Common Name (CN) (e.g., *.google.com)
  - Expiration Date (e.g., 2 years from now)
  - **Subject's Public Key**
  - Issuer (e.g., Verisign)
  - Issuer's signature
- **Common Name field**
  - Explicit name, e.g. cs.utah.edu
  - Or wildcard, e.g. *.utah.edu

The **CA ecosystem** aims to address comm **pre-auth**

Example x509 Certificate

```
Subject: C=US/O=Google Inc/CN=www.google.com
Issuer: C=US/O=Google Inc/CN=Google Internet Authority
Serial Number: 01:b1:04:17:be:22:48:b4:8e:1e:8b:a0:73:c9:ac:83
Expiration Period: Jul 12 2010 - Jul 19 2012
Public Key Algorithm: rsaEncryption
Public Key: 43:1d:53:2e:09:ef:dc:50:54:0a:fb:9a:f0:fa:14:58:ad:a0:81:b0:3d
7c:be:b1:82:19:b9:7c3:8:04:e9:1e5d:b5:80:af:d4:a0:81:b0:b0:68:5b:a4:a4
:ff:b5:8a:3a:a2:29:e2:6c:7c3:8:04:e9:1e5d:b5:7c3:8:04:e9:39:23:46
```

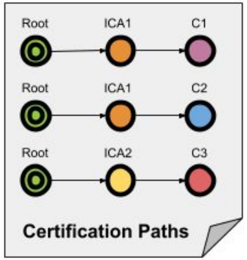
```
Signature Algorithm:  sha1WithRSAEncryption
```

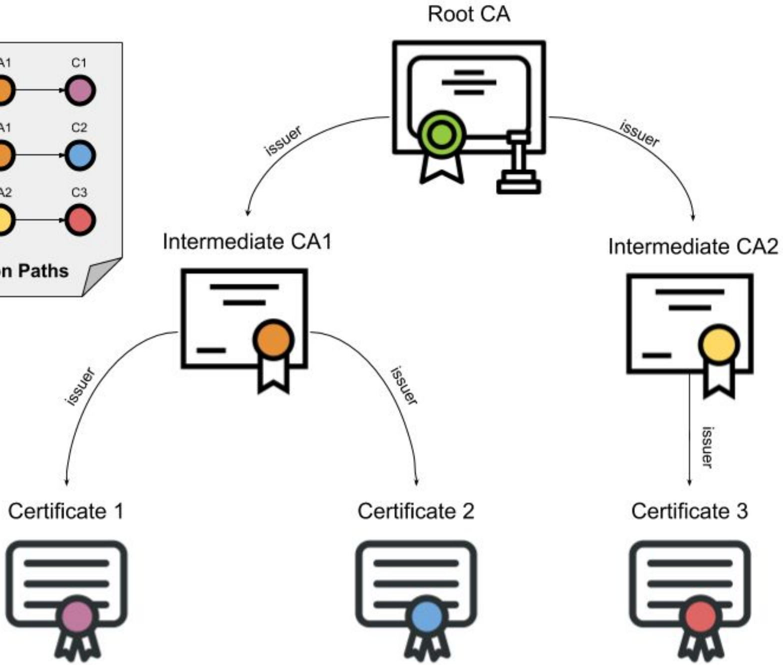
```
Signature: 39:10:83:2e:09:ef:ac:50:04:0a:fb:9a:f0:fa:14:58:ad:a0:81:b0:3d
7c:be:b1:82:19:b9:7c3:8:04:e9:1e5d:b5:80:af:d4:a0:81:b0:b0:68:5b:a4:a4
:ff:b5:8a:3a:a2:29:e2:6c:7c3:8:04:e9:1e5d:b5:7c3:8:04:e9:1e:5d:b5
```

Certificate Chaining

- **Root CA** signs a **certificate-issuing certificate** for delegated authority
  - Your browser "peels" this chain of certificates until finds one it trusts

- **Domain Validation:**
  - Is the certificate expired?
  - Does the registered email reply to me?
  - Does DNS record match the cert owner?
  - **More thorough, complicated certificate validation measures exist today**





Food for Thought

- Think of **CAs** like notaries or passport-issuing government entities

Is this ecosystem forever trustable?

Food for Thought

- Think of **CAs** like notaries or passport-issuing government entities

Is this ecosystem forever trustable?

What kinds of things could go wrong?

Next time on CS 4440…

Attacks on HTTPS, Networking 101