Lecture 25

Cross Site Scripting Prevention

ECE 422: Reliable and Secure Systems Design

UNIVERSITY OF ALBERTA

Instructor: An Ran Chen
Term: 2024 Winter

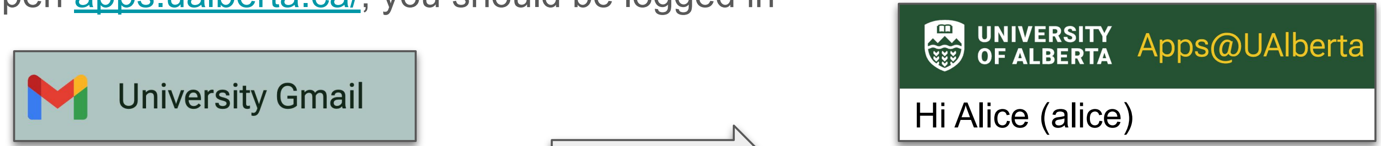
Schedule for today

- Key concepts from last classes
- Cross Site Scripting (XSS)
    - Stored XSS
- More details: Injection methods
- Cross Site Scripting Prevention
    - HttpOnly flag
    - HTML Escaping
    - Content Security Policy (CSP)
- Polls for Week 12 - 13 materials
    - Advanced Topics: Bitcoin, LLMs, Program Analysis, Automated Testing (Selenium)

Domain attribute

Domain attribute allows the cookies to be scoped to a broader domain.

- E.g., Gmail Apps@UAlberta could set a cookie for Apps@UAlberta
- Try it yourself in Incognito mode:
    - Login into your Gmail: apps.ualberta.ca/appslink/auth/feature/gmail
    - Open apps.ualberta.ca/, you should be logged in



Login into Gmail in Apps@UAlberta

Also logged in at Apps@UAlberta

If the domain attribute is set too loosely, then the server be may vulnerable to session fixation attack (e.g., allowing a third party to access the session id).

Path attribute

Path attribute scopes the cookie to a path prefix, which works in conjunction with the domain attribute to a particular request path prefix.

- E.g., cookies in path=/docs will match path=/docs/admin
- Try it yourself in Incognito mode:
  - Login into eClass: eclass.srv.ualberta.ca/course
  - Open eclass.srv.ualberta.ca/course/view.php?id=2187, you should be logged in

UNIVERSITY OF ALBERTA
IST eCLASS SUPPORT

Welcome to eClass for Students!

If the path attribute is set too loosely, it could leave the application vulnerable to attacks by other applications on the same server.

Problem with ambient authority

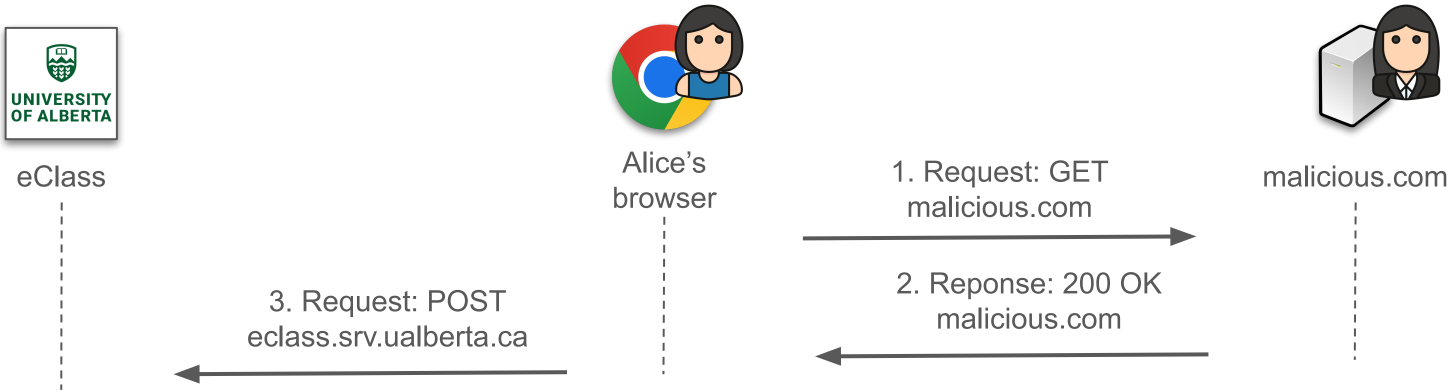
Recall: Ambient authority can be implemented with cookies

- If some properties (e.g., session ID) are valid, grant privileges to users

Consider the following scenario:

- Eve sets up malicious.com with an embedded HTML tag:

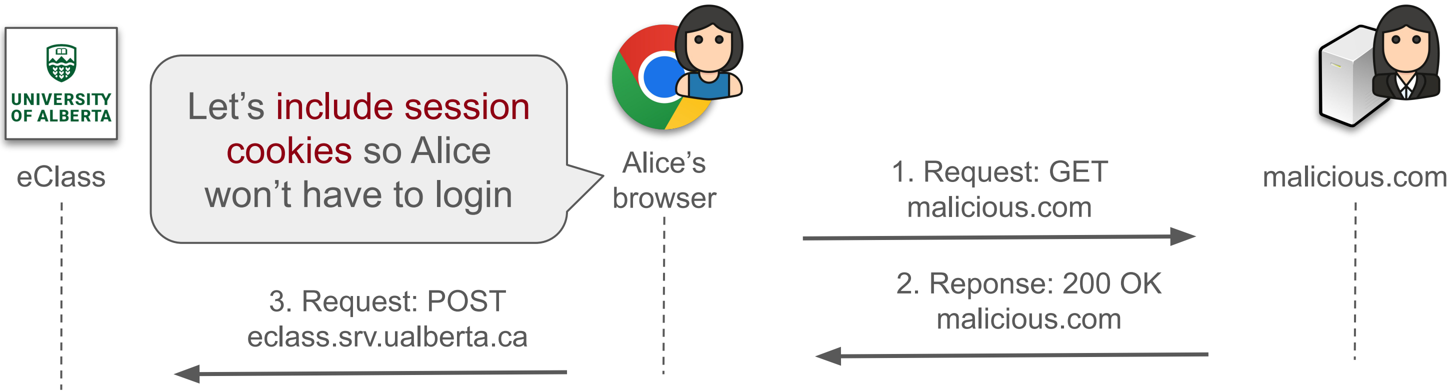
  <img src='https://https://eclass.srv.ualberta.ca/ece422/change_grade?user=eve&grade=100' />


UNIVERSITY OF ALBERTA
eClass
Alice's browser
malicious.com
1. Request: GET malicious.com
2. Reponse: 200 OK malicious.com
3. Request: POST eclass.srv.ualberta.ca

Problem with ambient authority

Browser helpfully includes the session cookies in all requests to eClass

- When an instructor (logged in) goes to malicious.com, the HTTP request will be triggered with his/her session cookies
- Since the session ID is valid, the request is accepted
- However, the request originated from malicious.com!



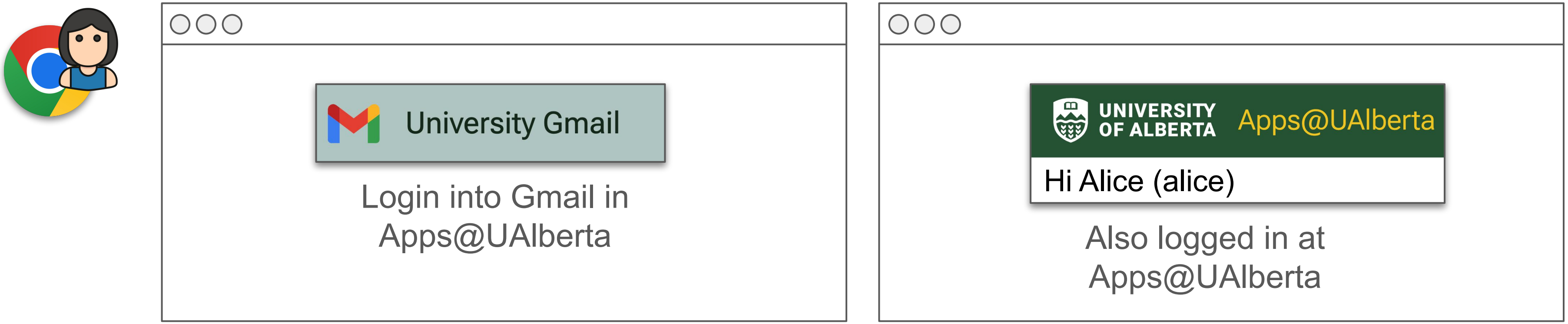
UNIVERSITY OF ALBERTA
eClass
Let’s include session cookies so Alice won’t have to login
Alice’s browser
malicious.com
1. Request: GET malicious.com
2. Reponse: 200 OK malicious.com
3. Request: POST eclass.srv.ualberta.ca

Recap on the demo

Demo: We tried to have access to eClass when we are logged in into Gmail

Result: It works because of it is coming from the same origin

- In case where the HTTP request to eClass is triggered by malicious.com, the Same-Origin Policy will restrict the access to Gmail session cookies



Login into Gmail in Apps@UAlberta

Also logged in at Apps@UAlberta

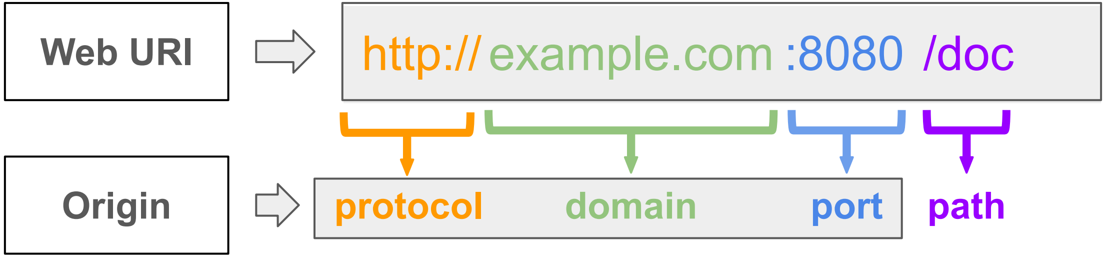
Same Origin Policy (SOP)

Same Origin Policy (SOP) is a web security mechanism that restricts how documents and scripts on one origin can interact with resources on another origin.

- Stop one website from interfering with another website

The policy checks if the origin matches the one from the website:

- Only when the origin (protocol + domain + port) is the same, the website allows read and write



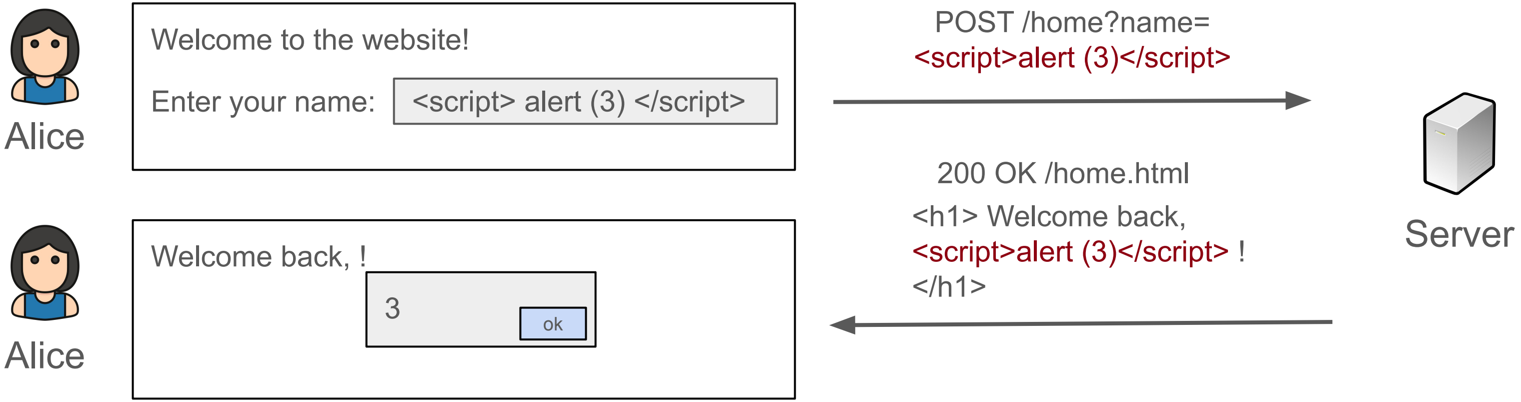
with XSS

Now suppose Alice injects a script in the request:

- Alice sends a request with the parameter name as:

  <script> alert (3) </script>

- Server responds with the home page html with the name parameter
- Input <script> … </script> is being reflected back in the response



Welcome to the website!

Enter your name: <script> alert (3) </script>

POST /home?name=
<script>alert (3)</script>

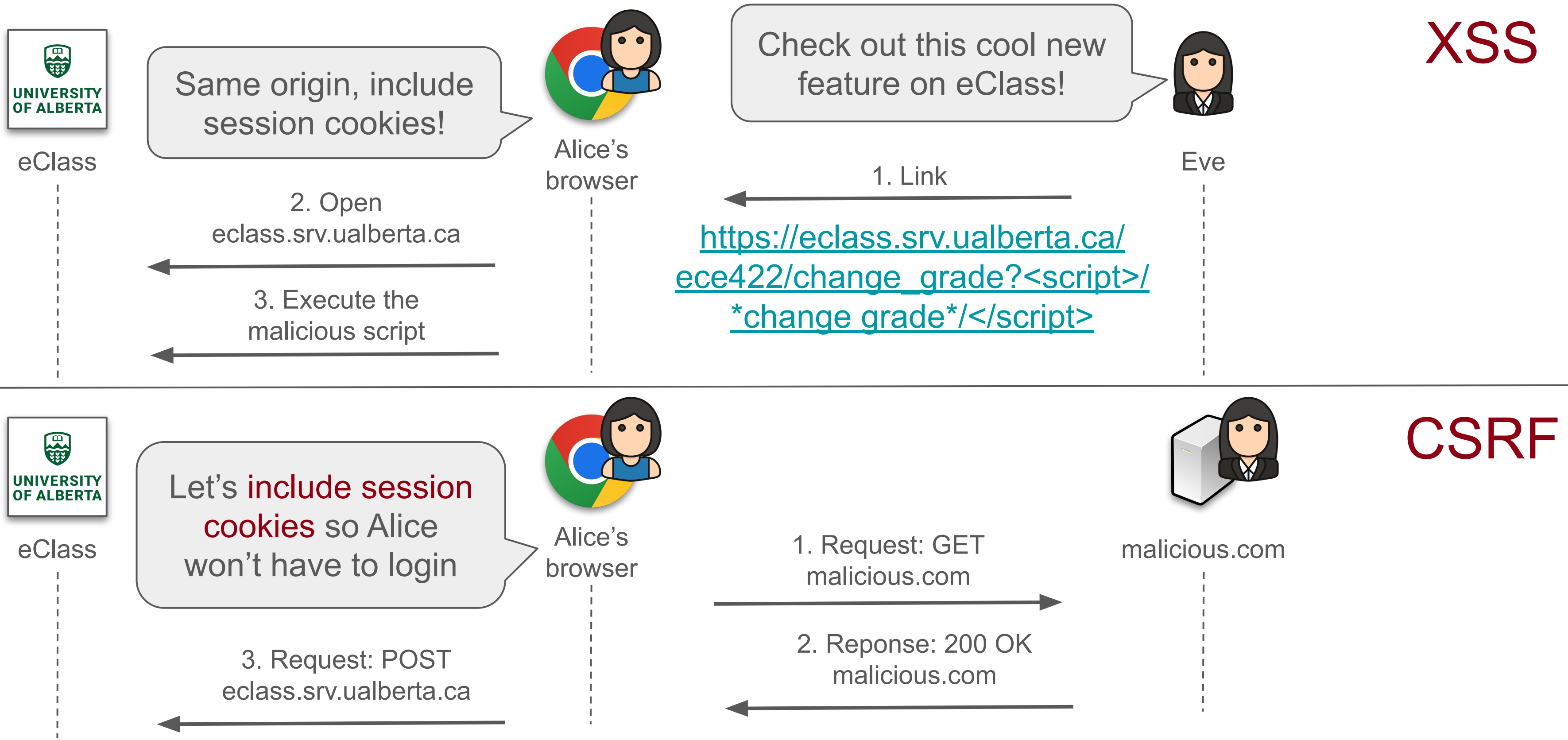
200 OK /home.html

```
<h1> Welcome back,
<script>alert (3)</script> !
</h1>
```

Welcome back, !

3

CSRF vs XSS



XSS

eClass

Same origin, include session cookies!

Alice's browser

Check out this cool new feature on eClass!

Eve

1. Link

https://eclass.srv.ualberta.ca/ece422/change_grade?<script>/*change grade*/</script>

2. Open eclass.srv.ualberta.ca

3. Execute the malicious script

CSRF

eClass

Let's include session cookies so Alice won't have to login

Alice's browser

malicious.com

1. Request: GET malicious.com

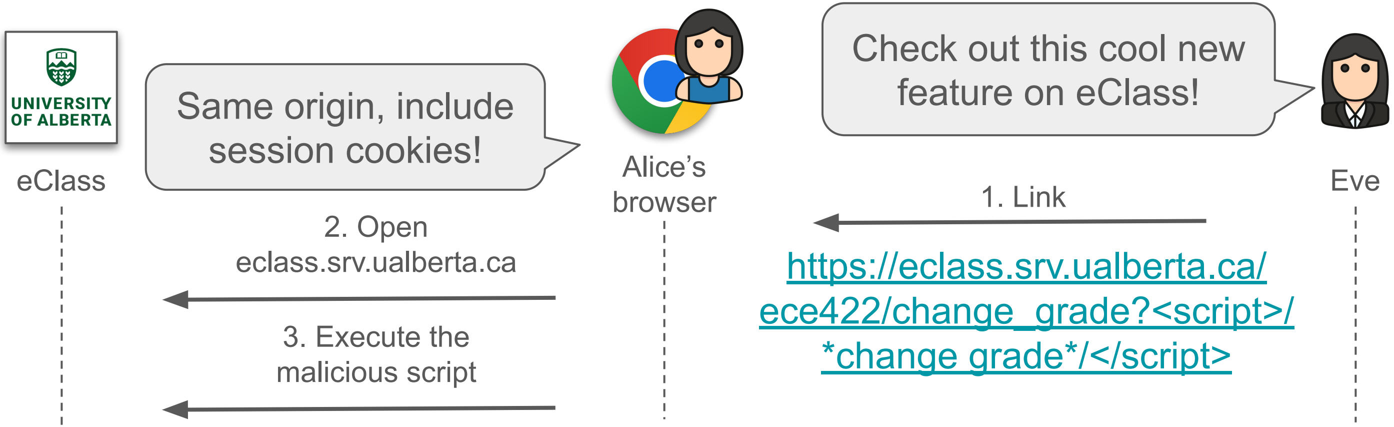
2. Reponse: 200 OK malicious.com

3. Request: POST eclass.srv.ualberta.ca

Example of reflected XSS

Eve sends an eClass link with some malicious script to Alice

- The script is activated once Alice clicked on the link
- The link sends a request to eClass which executes a malicious script that changes Eve's grade using Alice's session
- But the Same Origin Policy is never triggered (same origin)



eClass

Same origin, include session cookies!

2. Open eclass.srv.ualberta.ca

3. Execute the malicious script

Alice's browser

Check out this cool new feature on eClass!

Eve

1. Link

https://eclass.srv.ualberta.ca/ece422/change_grade?<script>/*change grade*/</script>

Question on previous materials

A security analyst reviews the following web server log:

```
[15/March/2024:12:00:00 +0100] “GET
/profile.php?id=<script>alert('www.malicious.com/grade_update.php')</script>”
```

Question: Which vulnerability is the attacker exploiting?

A. Ambient authority

B. Session fixation

C. Cross-Site Request Forgery (CSRF)

D. Cross Site Scripting (XSS)

Schedule for today

- Key concepts from last classes
- Cross Site Scripting (XSS)
  - Stored XSS
- More details: Injection methods
- Cross Site Scripting Prevention
  - HttpOnly flag
  - HTML Escaping
  - Content Security Policy (CSP)
- Polls for Week 12 - 13 materials
  - Advanced Topics: Bitcoin, LLMs, Program Analysis, Automated Testing (Selenium)

Stored XSS

Stored (persistent) XSS arises where the malicious script is stored on the target server (e.g., database).

- Malicious script is activated when a user retrieve the data without that data being made safe to render in the browser

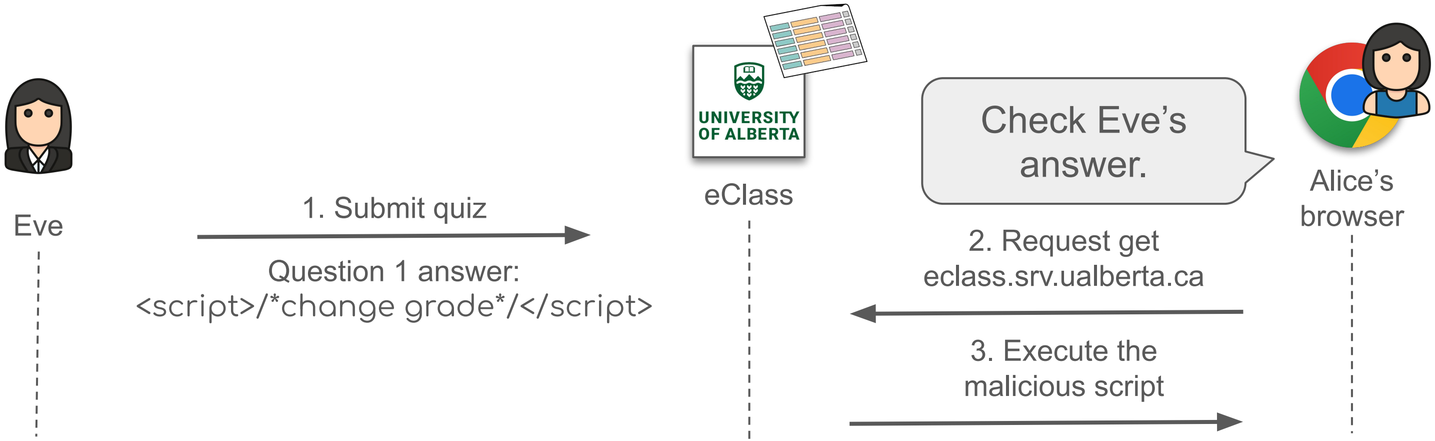
Attackers need to:

- Find a webpage vulnerable to XSS
- Send the malicious script as data to the server
- Wait for another user to retrieve the data and trigger the script

Example of stored XSS

Eve submit a malicious script to eClass as the answer to an online quiz question:

- Web server stores the scripts in the database
- Alice sends a request to eClass for Eve’s answer
- The script is activated when Eve’s answer is displayed on Alice’s browser


UNIVERSITY OF ALBERTA
eClass
Check Eve’s answer.
Alice’s browser
Eve
1. Submit quiz
Question 1 answer:
<script>/*change grade*/</script>
2. Request get eclass.srv.ualberta.ca
3. Execute the malicious script

Samy worm = stored XSS

Samy worm is an example of stored XSS:

- Malicious script was first uploaded by Samy on his own profile (his own profile entry in the database)
- When users visited Samy’s profile, the script also injected itself to the victim's profile (another user’s profile entry in the database)
- Then, the virus continues to spread each time a new user visits an affected user’s profile

Reflected vs Stored XSS

Reflected XSS: malicious code placed into the HTTP request and reflected in a HTTP response to the victim

- Possible vulnerabilities on the website: URL path or query parameters
- Attack: Passing injected scripts as parameters
- Victim: Anyone who visits the attacker’s URL

Stored XSS: malicious code persisted into the database

- Possible vulnerabilities on the website: any database access
- Attack: passing injected scripts as data in the database
- Victim: Anyone who access the affected data entry

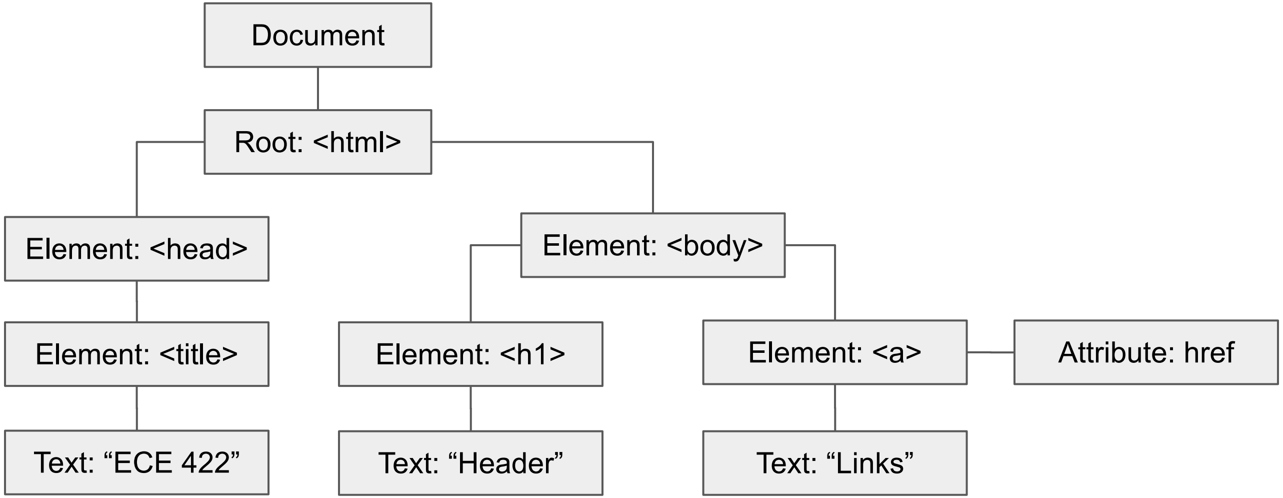
Schedule for today

- Key concepts from last classes
- Cross Site Scripting (XSS)
  - Stored XSS
- More details: Injection methods
- Cross Site Scripting Prevention
  - HttpOnly flag
  - HTML Escaping
  - Content Security Policy (CSP)
- Polls for Week 12 - 13 materials
  - Advanced Topics: Bitcoin, LLMs, Program Analysis, Automated Testing (Selenium)

Injecting methods

There are two common ways to inject code into the “HTML contexts”:

- Injecting UP: start a new code context (higher context)
- Injecting DOWN: introduce a new subcontext (lower context)



Injecting UP

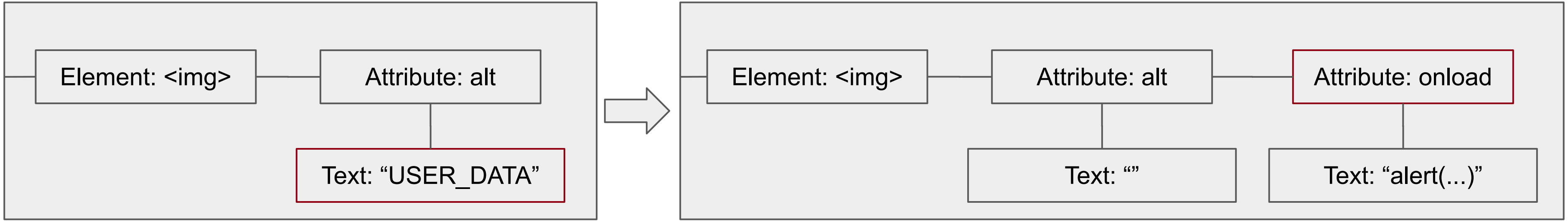
Injecting UP: start a new code context (higher context).

- For example, given the following html tag:

```
<img src='cat.png' alt='USER_DATA' />
```

- Close the alt attribute with ', open a new attribute onload
- Result:

```
<img src='cat.png' alt='' onload='alert(document.cookie)' />
```



Element: <img>

Attribute: alt

Text: “USER_DATA”

Element: <img>

Attribute: alt

Attribute: onload

Text: “”

Text: “alert(...)”

Injecting DOWN

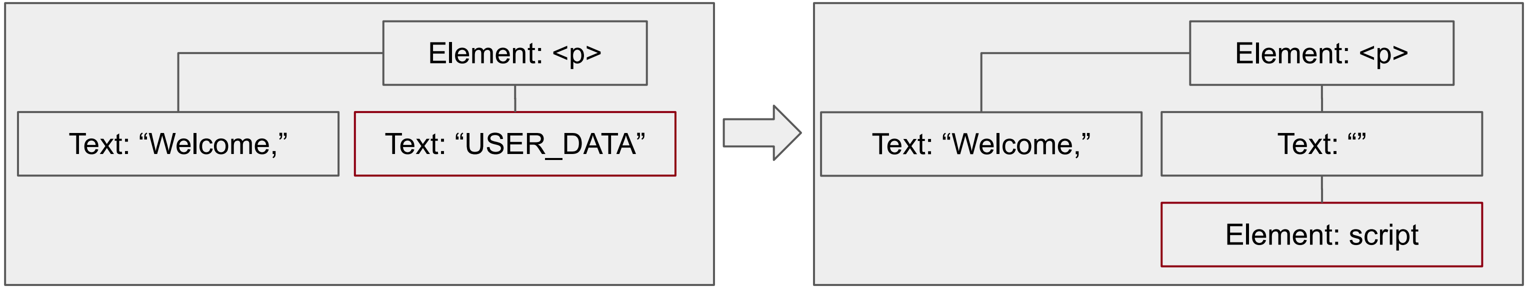
Injecting DOWN: introduce a new subcontext (lower context)

- For example, given the following html tag:

```
<p>Welcome, USER_DATA_HERE</p>
```

- Introduce a subcontext that allows scripting within the src attribute
- Result:

```
<p>Welcome, <script>alert(document.cookie)</script></p>
```



Element: <p>

Text: “Welcome,”

Text: “USER_DATA”

Element: <p>

Text: “Welcome,”

Text: “”

Element: script

Schedule for today

- Key concepts from last classes
- Cross Site Scripting (XSS)
  - Stored XSS
- More details: Injection methods
- Cross Site Scripting Prevention
  - HttpOnly flag
  - HTML Escaping
  - Content Security Policy (CSP)
- Polls for Week 12 - 13 materials
  - Advanced Topics: Bitcoin, LLMs, Program Analysis, Automated Testing (Selenium)

XSS Prevention

Problem with XSS: untrusted user data unexpectedly becomes code

- Analogous to SQL injection, but HTML injection instead
- Intuition: Sanitize the data so it will not become code

There are three common ways to sanitize the user input:

- HttpOnly
- HTML Escaping
- Content Security Policy

HttpOnly

HttpOnly flag restricts the client-side from accessing the cookies with JavaScript.

- Part of the Set-Cookie in HTTP response from the server
- First implemented in 2002 by Microsoft Internet Explorer developers

```
Set-Cookie: key=value; HttpOnly
```

HttpOnly

- For example, given the following html tag:

```
<img src='cat.png' alt='USER_DATA' />
```

- With reflected XSS:

```
<img src='cat.png'
alt='' onload='alert(document.cookie)' />
```

- Results with HTML escaping: Alert box shows nothing (empty cookie)

However, this also prevents web developers from accessing the cookies!

- Some sites may use JavaScript to read/write cookies to track the states

HTML escaping

HTML escaping is about “escaping” dangerous attacker-controllable characters.

- Escape these characters to prevent them to become code

Example of HTML escaping (attacker-controllable characters):

- (<) with &lt;
- (>) with &gt;
- (") with &quot;
- (’) with &#39;
- (&) with &amp;

HTML escaping

- For example, given the following html tag:

```
<img src='cat.png' alt='USER_DATA' />
```

- With reflected XSS:

```
<img src='cat.png'
alt='' onload='alert(document.cookie)' />
```

- Results with HTML escaping:

```
<img src='cat.png'
alt='&#39; onload=&#39;alert(document.cookie)' />
```

XSS prevented!

An important question: when to escape?

Option 1: Before the data is stored in the database

Option 2: When the data is rendered on user-side

An important question: when to escape?

Answer: Both, but always prioritize on rendering on the user-side

There are two reasons:

- Difficult to predict in what context the attack will appear in
  - Different programming languages will have different control characters (e.g., <, >, ", ', & in HTML)
- Never trust the data from the database
  - Data originated from users, never trust user input
  - A lot of things can go wrong, e.g., SQL injection

Key concepts into practice

- HTML escaping removes or sanitizes (replaces) dangerous characters
  - Test URL 1: Try to search for <script>alert(document.cookie)</script>
    - Returns no results but the query becomes scriptalertdocumentcookiescript
    - Conclusion: Some kind of data sanitization that detects and removes special characters

Key concepts into practice

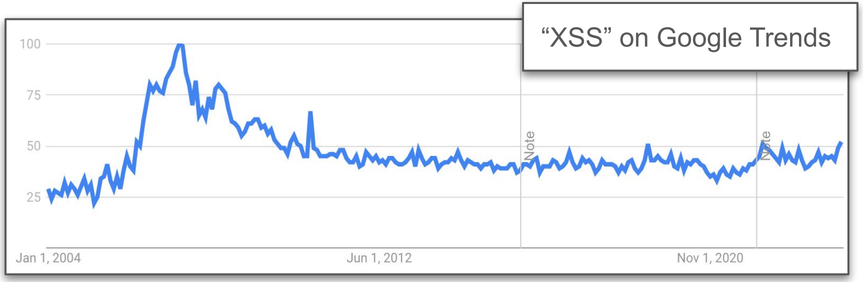
- HTML escaping removes or sanitizes (replaces) dangerous characters
  - Test URL 2: Try to search for <script>alert(document.cookie)</script>
    - Returns 403 Forbidden. Ok, now try something shorter: <script
    - Conclusion: another data sanitizer that looks for dangerous characters + keyword
    - Notice in the request URL, HTML escaping performed: keyword%5D=%3Cscript

Key concepts into practice

- HTML escaping removes or sanitizes (replaces) dangerous characters
  - Test URL 3: Try to search for <script>alert(document.cookie)</script>
    - Everything works perfectly
    - But notice the URL: q=<script>alert(document.cookie)<%2Fscript>
    - Conclusion: HTML escaping on forward slash (/)

On the prevalence of XSS

- Vulnerability may exist in many different contexts
  - Contexts = locations where attacker-controllable data appears
  - Different websites have different contexts to deal with
  - E.g., HTML contexts, URL contexts, CSS contexts
- Each context may have very different control characters to sanitize
  - E.g., Within HTML, there are at least five control characters: <, >, ", ', and &
- Any data that does not go through this process creates a vulnerability



“XSS” on Google Trends

Schedule for today

- Key concepts from last classes
- Cross Site Scripting (XSS)
  - Stored XSS
- More details: Injection methods
- Cross Site Scripting Prevention
  - HttpOnly flag
  - HTML Escaping
  - Content Security Policy (CSP)
- Polls for Week 12 - 13 materials
  - Advanced Topics: Bitcoin, LLMs, Program Analysis, Automated Testing (Selenium)

Database

- Database Basics
    - Relational data model
        - Foreign and primary keys
    - Understanding SQL Queries
- Inference attack and SQL Injection
- Security Requirements
    - Referential Integrity
    - Concurrency Controls
- Big Data Application Framework: Apache Hadoop

Networks

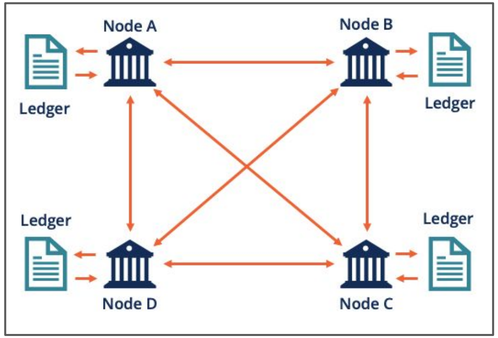
- Network Basics
  - Transmission Control Protocol (TCP)
  - User Datagram Protocol (UDP)
- WiFi Protected Access (WPA)
- Denial of Service (DoS)
  - Volume attacks
  - Application-based attacks
  - Disabled communications
  - Hardware or software failure

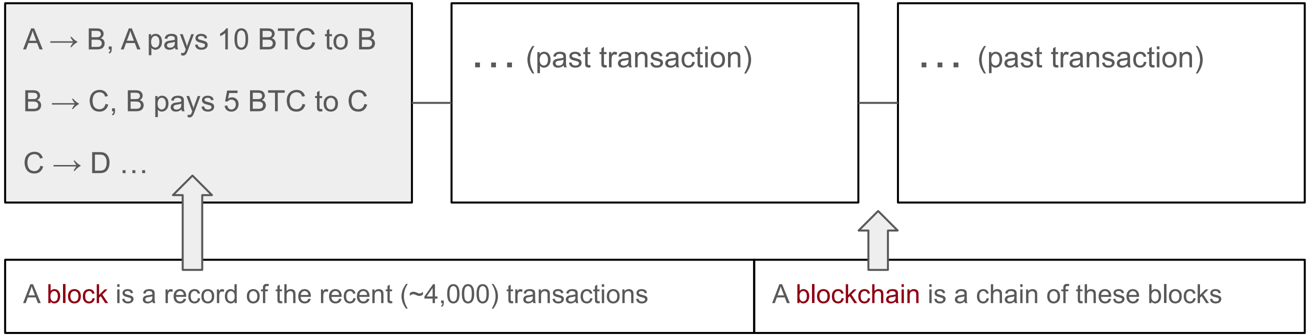
Security Topic: Bitcoin

Bitcoin is a digital or virtual currency.

- Introduced during 2008 financial crisis; “Satoshi Nakamoto” published a whitepaper titled “Bitcoin: A Peer-to-Peer Electronic Cash System”
- Transaction fee much cheaper than banks (flat vs percentage-based)

Decentralized distributed ledger: Everyone has a public ledger that contains the history of every bitcoin transaction





A → B, A pays 10 BTC to B

B → C, B pays 5 BTC to C

C → D …

. . . (past transaction)

. . . (past transaction)

A **block** is a record of the recent (~4,000) transactions

A **blockchain** is a chain of these blocks

Topics for Bitcoin

Basic knowledge

- Blockchain concepts
- Bitcoin mining principles

Advanced concepts

- Proof-of-Work
- Design features
  - Decentralization
  - Reaching consensus on transactions
  - Immutability

Security Topic: Large-Language Models (LLMs)

Large language models (LLMs) are very large deep learning models that are pre-trained on vast amounts of data.

- LLMs are trained by predicting the next word to learn about the world

From machine learning (ML) models to LLMs:

- ML: models trained for a specific task
- LLMs: models trained for natural language understanding
  - Extremely flexible to perform different tasks, e.g., text generation, summarization, translation

Example of LLMs: Llama 2

- Released by Meta, open source
- 70B parameters (~140GB) + code

Topics for LLMs

Basic knowledge

- Prompts (hard prompts vs soft prompts)
- Prompt engineering

Advanced concepts

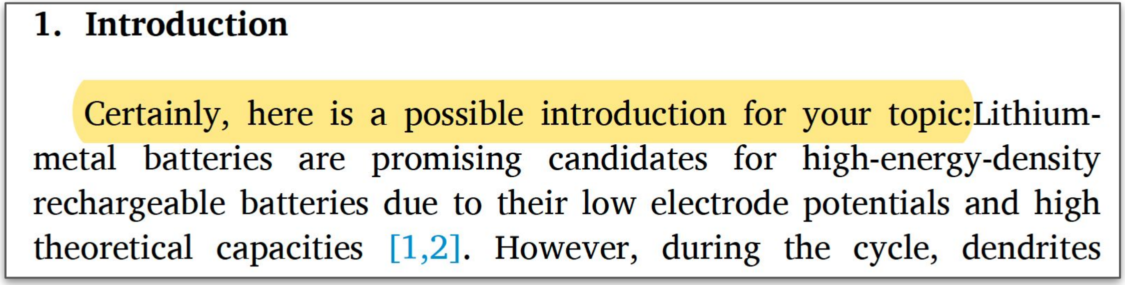
- Pre-training
- Fine-tuning
- In-context learning

Reality check

- Security challenges: Hallucination
- Use cases

Another comment …

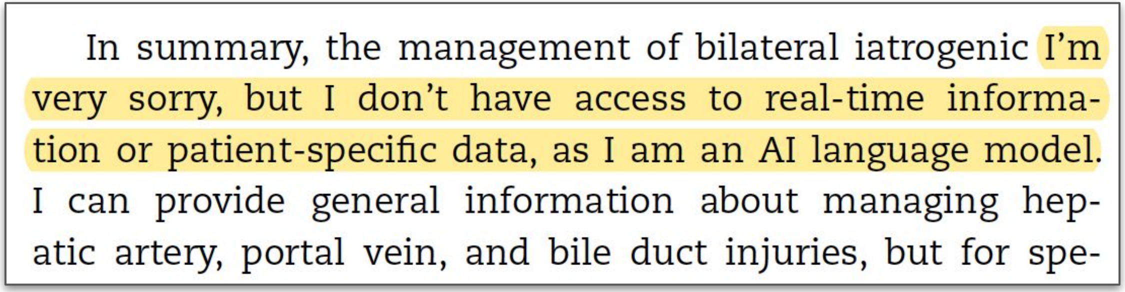
A recent research paper on Lithium metal batteries published in March 2024:



1. **Introduction**

Certainly, here is a possible introduction for your topic:Lithium-metal batteries are promising candidates for high-energy-density rechargeable batteries due to their low electrode potentials and high theoretical capacities [1,2]. However, during the cycle, dendrites

Another accepted research paper to be appeared in June 2024:



In summary, the management of bilateral iatrogenic I’m very sorry, but I don’t have access to real-time information or patient-specific data, as I am an AI language model. I can provide general information about managing hepatic artery, portal vein, and bile duct injuries, but for spe-

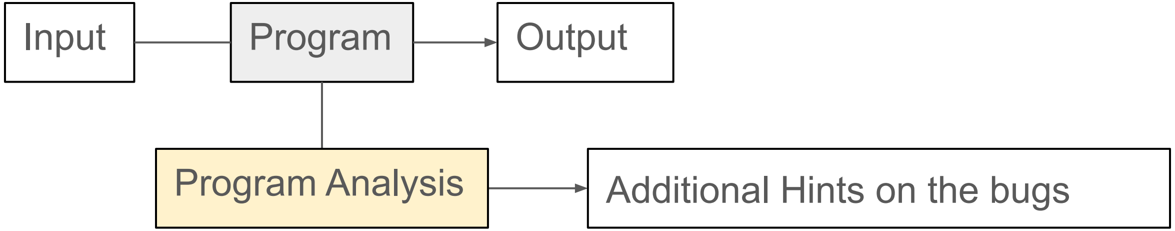
Reliability Topic: Program Analysis

What is Program Analysis? Automated analysis of program behavior

- Find software bugs
- Optimize performance

Why is this needed? Software maintenance is expensive

- All programs have bugs, manual testing and debugging is tedious and time consuming
- Asset to have for: Data Scientist, Data Engineer, Algorithm Engineer



Topics for Program Analysis

Basic knowledge

- Grammars
- Abstract Syntax Tree (AST)
- Control Flow Graph (CFG)

Advanced concepts

- Static and Dynamic Analysis
- Path Profiling
- Program Slicing
  - Static Slicing
  - Dynamic Slicing
- Introduction to Automated Debugging: Automated Program Repair

Reliability Topic: Automated Testing (Selenium)

What is Selenium? A tool to help automate browsers.

- Automate test cases (web application)
- Create web bots (e.g., cookie clicker)
- Web scraping for data

Why is this needed? Software Testing is expensive

- Asset to have for: Quality Assurance developers, Web developers

Note: this will be a lot more practical and technical.