Lecture 24
Cross Site Scripting

ECE 422: Reliable and Secure Systems Design

Instructor: An Ran Chen
Term: 2024 Winter

Schedule for today

- Key concepts from last classes
- Cookie attributes
- Cross-Site Request Forgery
  - Demo with eClass
- Same Origin Policy
- Cross Site Scripting (XSS)
  - Reflected XSS
- TODOs

Cookies

Cookies are small piece of data that a server sends to a user's web browser.

- Web browser may store the cookie and send it back to the same server with later requests

Goals of cookies:

- Personalization: provide experiences based on personal preferences
  - E.g., preferred language to automatically deliver the right content
- Session management: make it easier for users to access accounts
  - E.g., stored login information to avoid login every time
- Tracking: track and analyze how users interact with the website
  - E.g., user behavior to enhance performance

Creating session cookies

After receiving an HTTP request, a server can send Set-Cookie headers with the response:

- Use the Set-Cookie response header to send cookies from the server to the user agent.
- Set-Cookie: <cookie-name>=<cookie-value>
- E.g., Set-Cookie: theme=dark

Then, with every subsequent request to the server, a client can send Cookie headers with a request:

- Use the Cookie request header to send cookies back to the server.
- Cookie: <cookie-name>=<cookie-value>
- E.g., Cookie: theme=dark

Signing cookies

To avoid session fixation attack

Session fixation attack attempts to fixate (find) another user's session identifier to gain access to the account by:

- Stealing cookies
- Guessing session identifiers

Solution: Session IDs should be signed, transient, revocable and unpredictable

- Signed: integrity of the cookies
- Transient: timeout after a certain period of time
- Revocable: ready for worst-case scenario (e.g., reset password = clean up)
- Unpredictable: randomness in session ID generation
    - E.g., HMAC hash for one-way function

Question on sessions and cookies

Which of the following statements is true?

A. A web cookie is a small piece of data sent from a website and stored in user's web browser while a user is browsing a website.

B. A web cookie is a small piece of data sent from user and stored in the server while a user is browsing a website.

C. A web cookie is a small piece of data sent from root server to all servers.

D. None of these

Cookie attributes

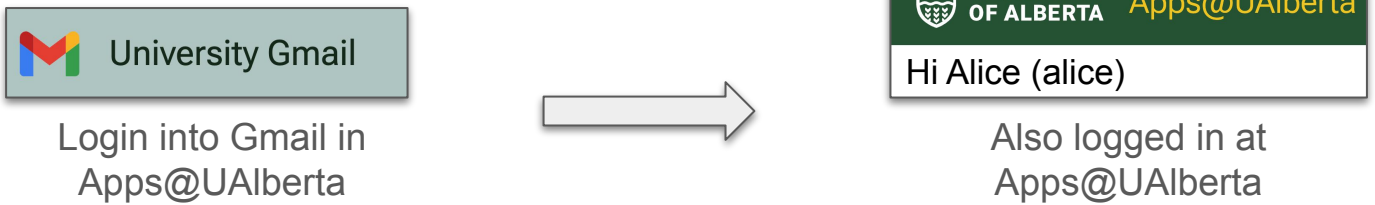
Cookie attributes are what restrict/protect the cookies from malicious users:

- Domain attribute: allows the cookie to be set to a broader domain
- Path attribute: scopes the cookie to a path prefix
- Expires attribute: specifies an expiration date
  - Expires date and time are relative to client the cookie is being set on, not the server
  - Defining lifetime of a cookie is necessary to avoid session fixation attacks

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



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

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

Overall idea

Desired properties for sessions:

- Browser remembers user
  - Cookies for better user experience
- User cannot modify session cookie to login as another user
  - Signed and unpredictable session IDs to avoid session fixation attack
- Session cookies only last as long as the browser is open
  - Desired, but different in practice
- Sessions can be managed by the server
  - Worst-case scenario: revoke the session IDs
- Sessions expire after some time
  - Avoid session fixation attack

Schedule for today

- Key concepts from last classes
- Cookie attributes
- Cross-Site Request Forgery
  - Demo with eClass
- Same Origin Policy
- Cross Site Scripting (XSS)
  - Reflected XSS
- TODOs

Ambient authority

Ambient authority is an access control based on global and persistent properties of the requester.

- Involve implicit trust and privileges to a user

In the context of web application security, the authority is automatically granted to a user based on their session state

For example:

- User logins into a web application
- Server sends the session ID back to the user
- Ambient authority: web application implicitly authenticates and authorizes all the actions performed by the user based on the session ID

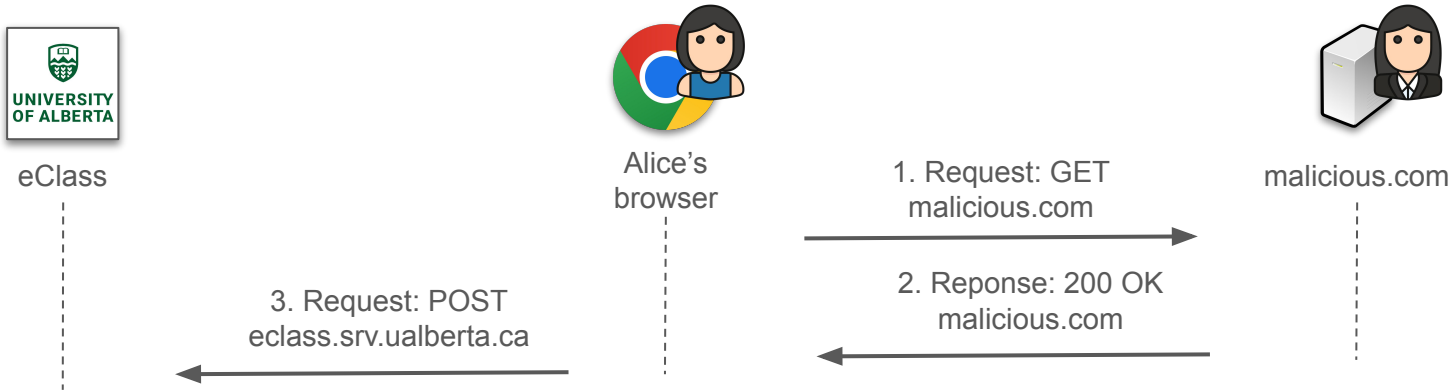
Problem with ambient authority

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:

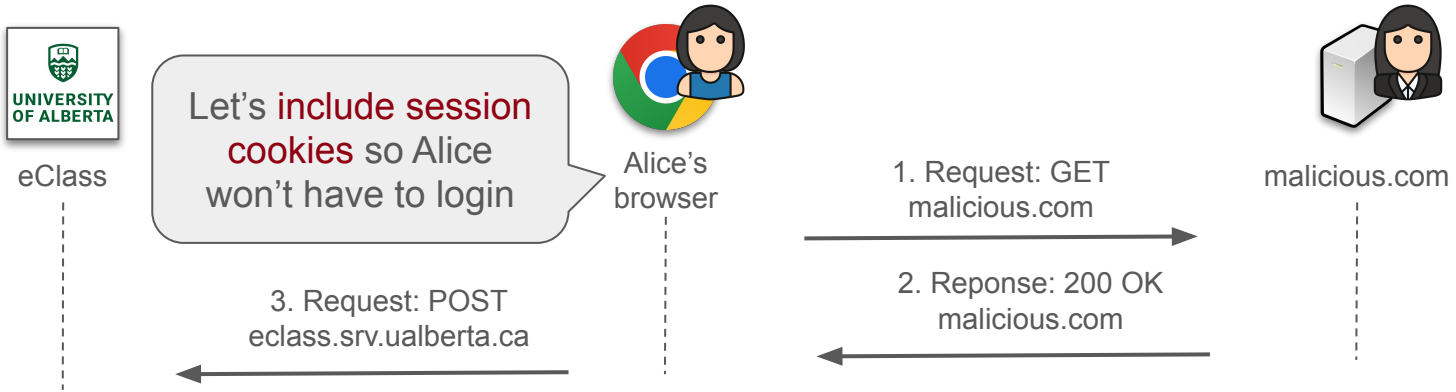
- Eve sets up malicious.com with an embedded HTML tag:



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!



Problem with ambient authority

While ambient authority improves the user experience and facilitates session management, it also introduces security risks.

- Attackers can uses a victim's logged-in session to perform any action
- E.g., Eve uses Alice’s logged-in eClass session to change the grade

Problem: It is unclear who initiated the request

This is an example of Cross-Site Request Forgery (CSRF)

- CSRF attack tricks the victim into submitting a malicious request
- It inherits the identity of the victim

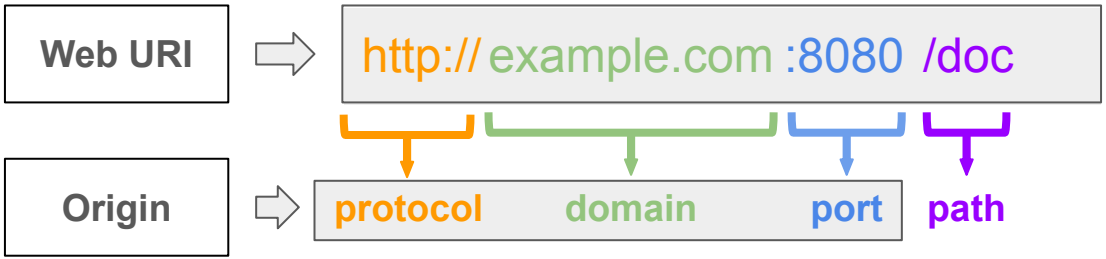
Same Origin Policy (SOP)

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



Example: origin matching

Check if the Same Origin Policy can be satisfied:

Website	⇨	http:// example.com :80 /doc

Requests coming from:

Request 1	⇨	http://example.com	
Request 2	⇨	https://example.com/doc	
Request 3	⇨	http://event.example.com	
Request 4	⇨	http://example.com:8080	

Example: origin matching

Check if the Same Origin Policy can be satisfied:

Website	⇨	http:// example.com :80 /doc

Requests coming from:

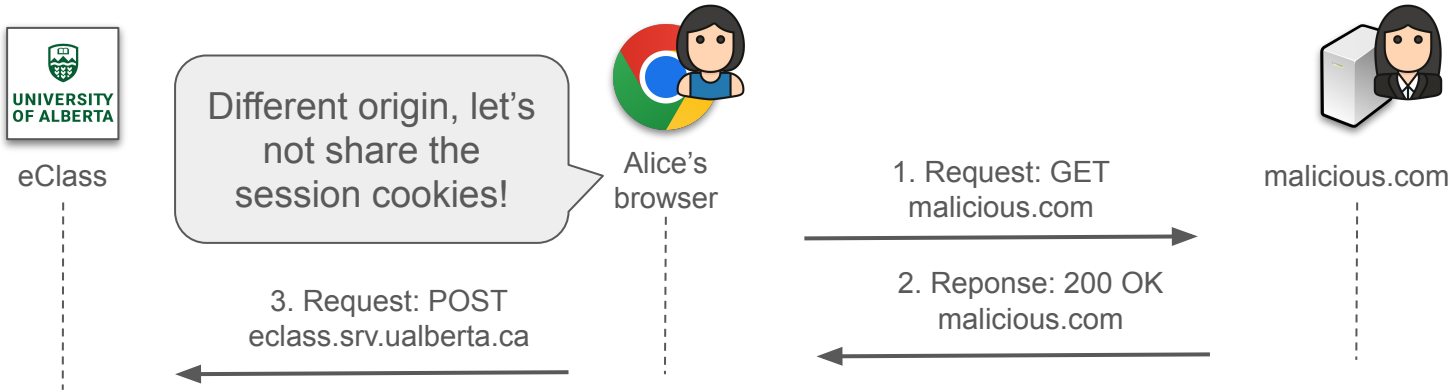
Request 1	⇨	http://example.com	Default port for HTTP: 80	✓
Request 2	⇨	https://example.com/doc	Different protocol	✗
Request 3	⇨	http://event.example.com	Different subdomain	✗
Request 4	⇨	http://example.com:8080	Different port	✗

Same Origin Policy prevents Cross-Site Request Forgery

Browser will not be allowed to include the session cookies:

- When an instructor (logged in) goes to malicious.com, the HTTP request will be triggered
- The browser permits cross-origin request, but prevents reading data from another origin
- Alice's session cookies remain safe

Same Origin Policy: prevents scripts running under one origin to read data from another origin



Schedule for today

- Key concepts from last classes
- Cookie attributes
- Cross-Site Request Forgery
- Same Origin Policy
- Cross Site Scripting (XSS)
  - Reflected XSS
  - Stored XSS
- TODOs

Samy worm

In 2005, Samy Kamkar released the Samy worm onto MySpace.

- Samy worm was a self-propagating cross-site scripting worm that caused the victim to send a friend request to Kamkar without knowing it
- Problem: each user who viewed his profile will have the same script planted on their own profile
- Consequences:
  - Within 20 hours, over one million users was affected
  - MySpace was forced shut down

Samy Kamkar at Black Hat conference (2010)

History of Samy

Cross-site scripting (XSS)

Cross-site scripting (XSS) is an attack in which attackers injects malicious executable scripts into the website.

- This happens when untrusted user data unexpectedly becomes code

How it works?

- Attackers can inject malicious code to a vulnerable website
- Website then returns malicious code to users
- When the malicious code executes inside a victim's browser, the attacker can fully control and compromise their interaction

Without proper data sanitization, the website may execute the malicious code on other users' system

XSS as a code injection vulnerability

In cross-site scripting (XSS), malicious JavaScript scripts are injected into an HTML document.

```
<html> Welcome back,
        <script> ... </script>
<html>
```

XSS: untrusted user data unexpectedly becomes code

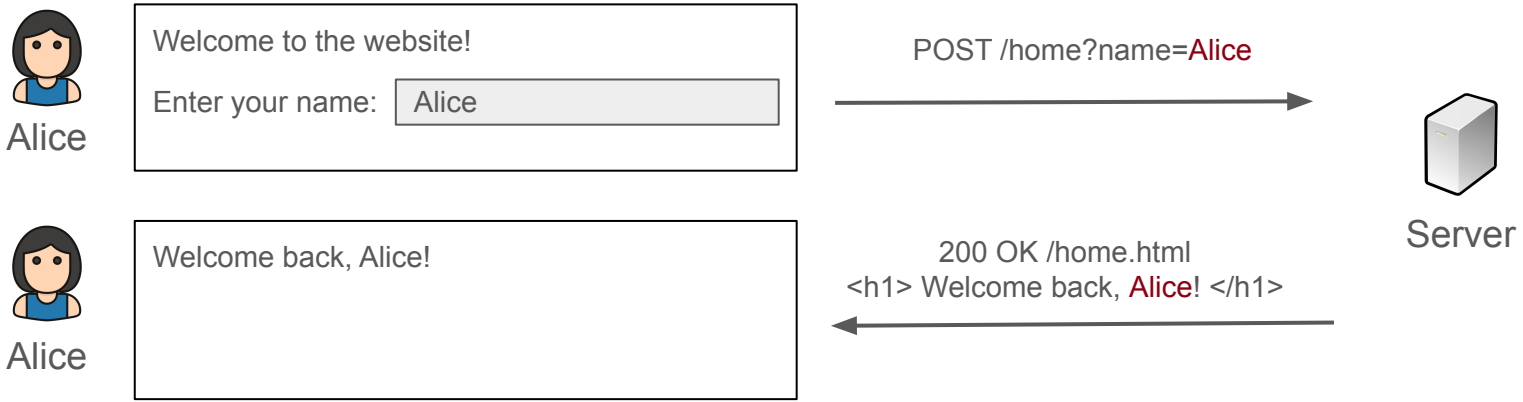
With XSS, attackers can perform session fixation attacks:

- View user's cookies
- Send HTTP requests with the user's cookies

Without XSS

Suppose Alice sends her name to the web server:

- Alice sends a request with the parameter name as: Alice
- Server responds with the home page html with the name parameter
- Input Alice is being reflected back in the response



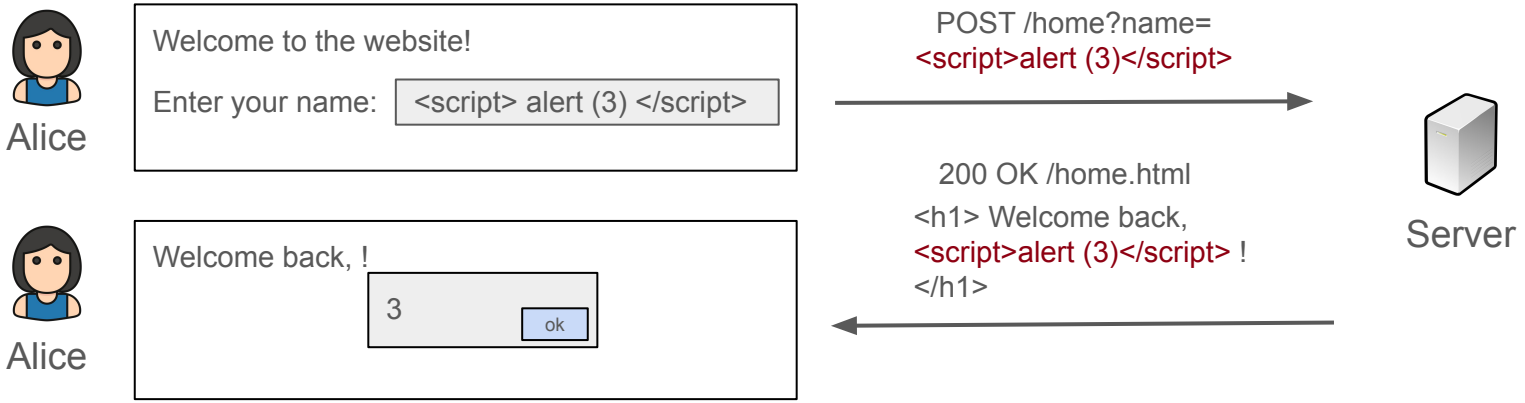
with XSS

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



Types of XSS

There are 3 common types of XSS:

- Reflected (non-persistent) XSS
  - Malicious script is reflected off of a web application to the victim's browser
- Stored (persistent) XSS
  - Malicious script is stored on the target server (e.g., database)
- DOM-based XSS
  - Malicious script only exists in client-side code, usually by writing the data back to the DOM

Reflected XSS

Reflected (non-persistent) XSS arises where the malicious script is reflected off in the victim's browser.

- The script is activated through a link, which sends a request to a website with a vulnerability that enables execution of malicious scripts

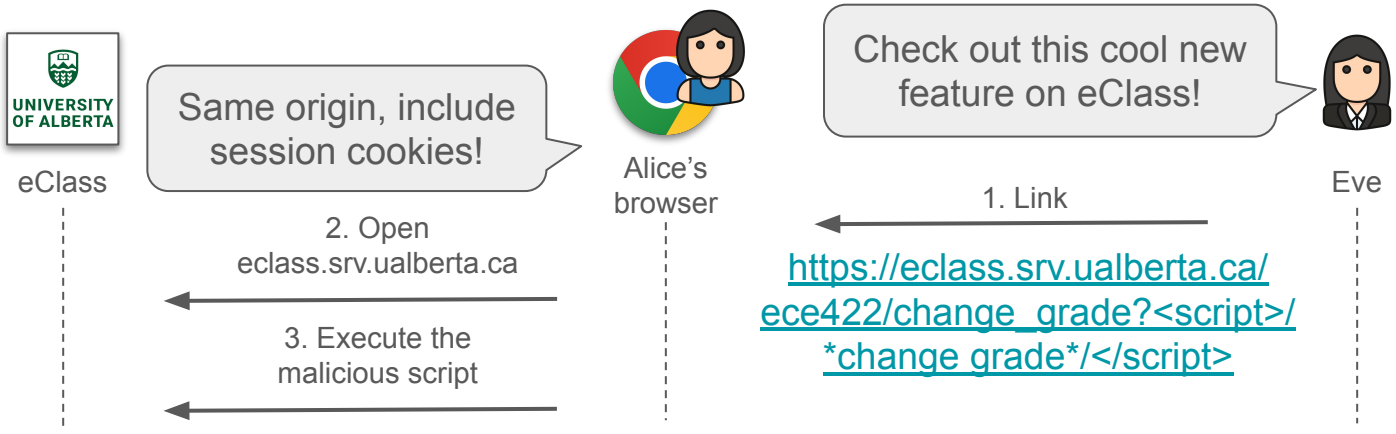
Attackers need to:

- Find a webpage vulnerable to XSS
  - Find a URL that they can make victims visit that can include the attack scripts
- Send the URL with injected scripts to the victims

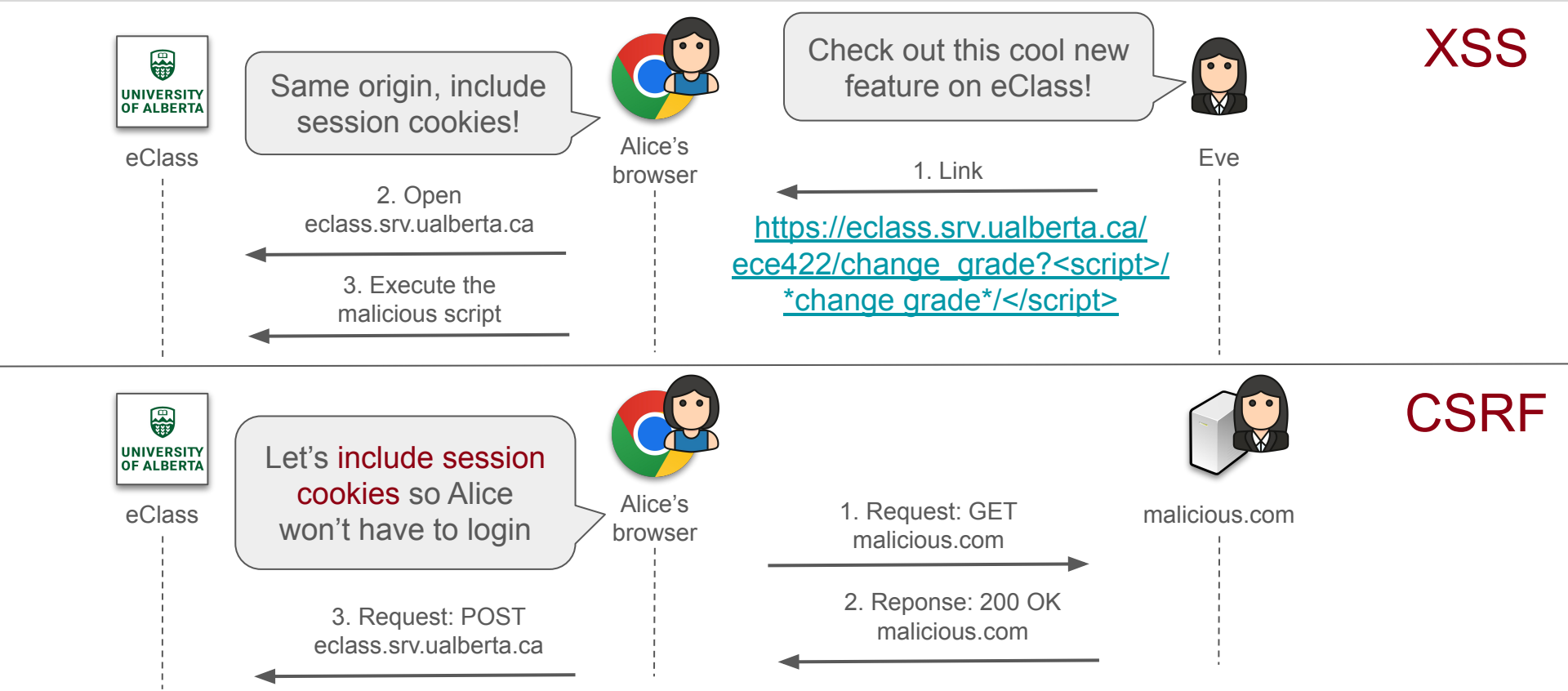
Example of reflected XSS

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)



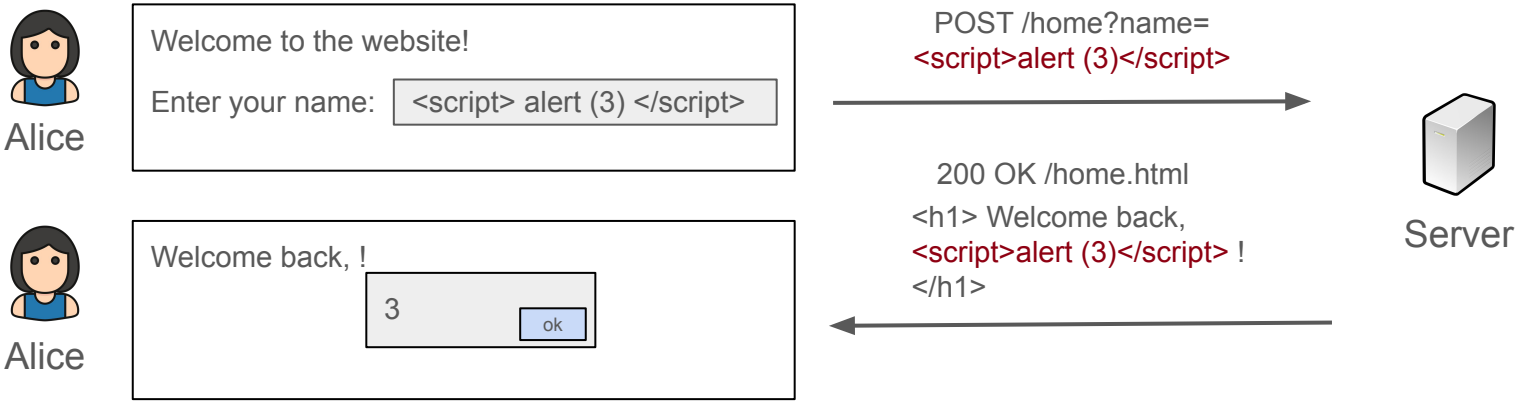
CSRF vs XSS


XSS
UNIVERSITY OF ALBERTA
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
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

with XSS

Note that, while alert(3) is a typical payload we use to find XSS vulnerabilities:

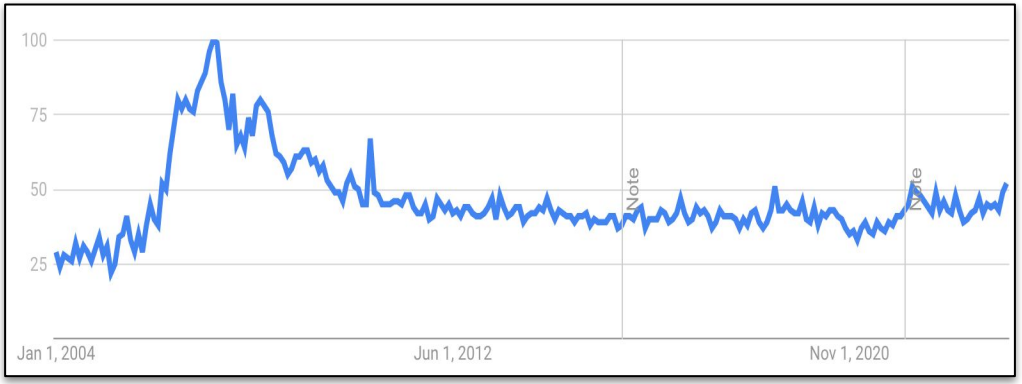
- Avoid using it in Capture The Flag (CTFTime, Google CTF) competitions
- Avoid using it in Bug Bounty Programs
- Google Bug Hunters: Don't use alert(1)



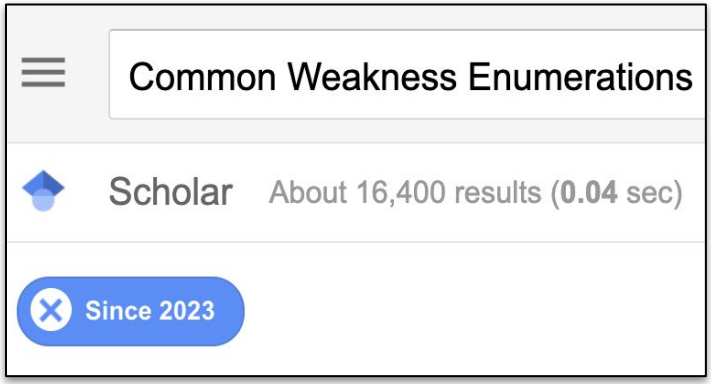
Google Trends on XSS

Although XSS became popular in 2005, it is still relevant today.

- OWASP's Top Web App Security Risks: XSS attacks remain in the Top 3
- 33 Common Weakness Enumerations (CWEs) mapped into this category have the second most occurrences in applications



"XSS" on Google Trends



"Common Weakness Enumerations" on Google Scholar

TODOs

Next class:

- Stored (persistent) XSS + Defences against XSS
- Deliverable due on Friday, March 15, 23:59 MST.
- Poll for materials from Week 12 - 13 (March 25 ~ April 5)
  - 2-minute overview on each advanced subject, same thing with Database and Networks modules
  - Poll remains open until Monday, March 18 so you have more time to look up each subject

- No new material on Week 14 (April 8)
- No class on Week 15 (April 15)
- Final exam during Week 16, on Wednesday, April 24