Lecture 26
Content Security Policy

ECE 422: Reliable and Secure Systems Design

Instructor: An Ran Chen
Term: 2024 Winter

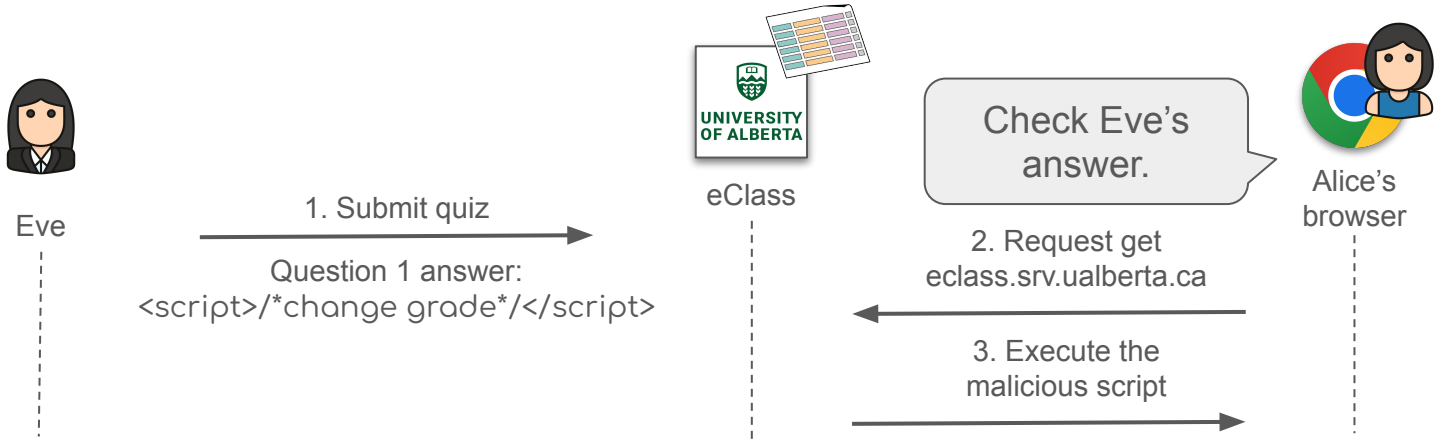
Schedule for today

- Key concepts from last classes
- Cross Site Scripting Prevention
  - Content Security Policy (CSP)
  - CSP Directives
  - Key concepts into practice
  - Challenge: Nested scripts
- Next class
  - “CSP is Dead” paper by Google in 2016
  - Additional CSP protections: strict-dynamic
- Polls for Week 12 - 13 materials

Example of stored XSS

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

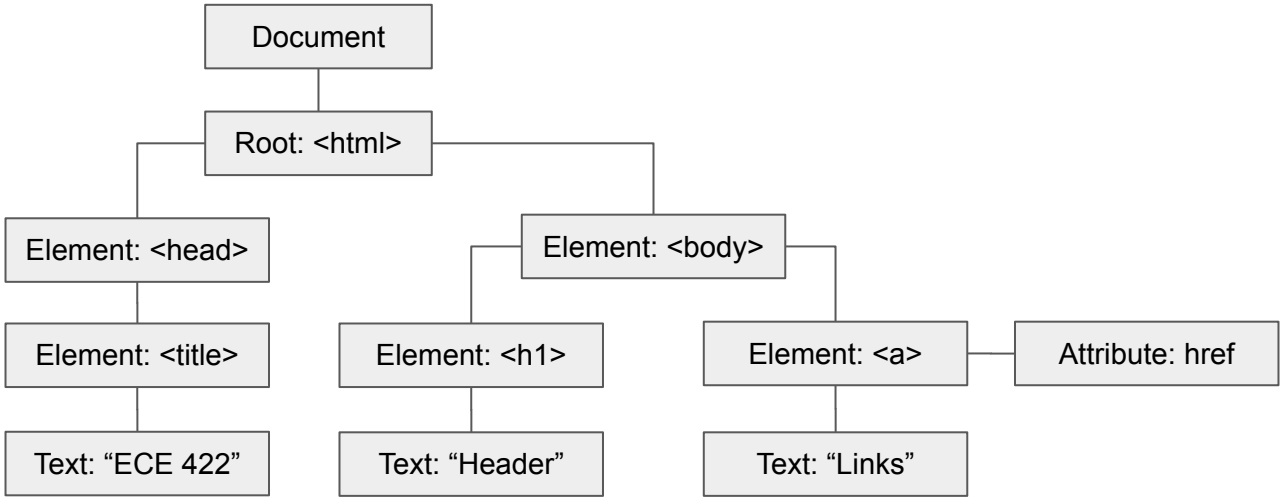
- Web server stores the scripts in the database
- Alice sends a request to eClass for Eve's answer
- The script is activated when Eve's answer is displayed on Alice's browser



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)



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 HttpOnly: 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

Take-homes on HttpOnly

Take-homes:

- Can HttpOnly mitigate the risks associated with XSS? Yes
- Can HttpOnly prevent XSS? No

A typical XSS scenario uses JavaScript to steal the cookie information (e.g., session fixation attack), however, there are many other capabilities of XSS, for example:

- Attacker can carry out the attack from the victim's browser
- E.g., Use XSS to make a malicious request directly to the server

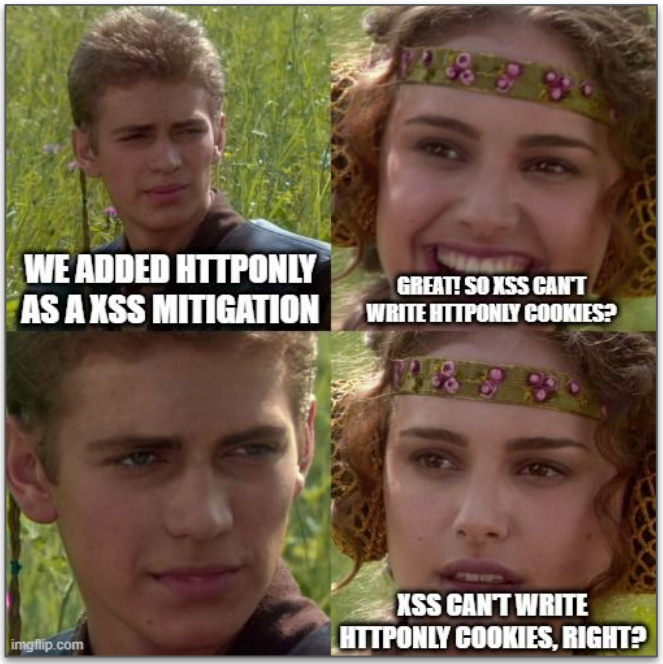
HttpOnly flag does not prevent XSS attacks completely. It only prevents those that try to steal cookie data through JavaScript.

Limitations of HttpOnly

- HttpOnly is not a preferred approach where cookie access is required by JavaScript for the basic functionality of the system
- HttpOnly only mitigates the risks associated with one potential XSS vulnerability
  - Stealing cookies is not the only threat in XSS
- Cookies do not always contain useful information
  - Session cookies are not always security relevant

One more comment on HttpOnly

[Overwriting HttpOnly cookies with Javascript]




```
for (let i=0; i<1000; i++) {
   document.cookie = "cookie"+i+"=overflow";
}

 document.cookie = "victimcookie=new_value"
```

- Overflow the cookie jar
  - Chrome has a limited cookie jar
  - Delete old ones when over the limit
- Overwrite the cookie (including HttpOnly flag)

Image source

HTML escaping

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

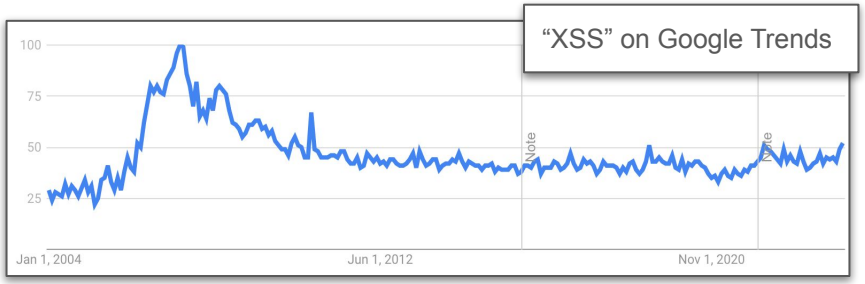
- Escape these characters to prevent them to become code

Example of HTML escaping (attacker-controllable characters):

- (<) with <
- (>) with >
- (") with "
- (’) with '
- (&) with &

Limitations of HTML escaping

- Context matters, HTML escaping only deals with the “HTML context”
- Each context may have very different control characters to sanitize
- Difficult to scale up with new technologies
- Any data that does not go through this process creates a vulnerability



Schedule for today

- Key concepts from last classes
- Cross Site Scripting Prevention
  - Content Security Policy (CSP)
  - CSP Directives
  - Key concepts into practice
  - Challenge: Nested scripts
- Next class
  - “CSP is Dead” paper by Google in 2016
  - Additional CSP protections: strict-dynamic
- Polls for Week 12 - 13 materials

Content Security Policy (CSP)

Content Security Policy (CSP) restricts which resources (e.g., JavaScript, CSS) can be loaded, and the URLs that they can be loaded from.

- Part of the HTTP response from the server

```
Content-Security-Policy: directives 'value';
```

  - directives specify type of resources, e.g., scripts, fonts, frames, images, audio and etc.
  - 'value' specifies domains, e.g., 'self', example.com, *.example.com

- Example: show all content from the site's own origin

```
Content-Security-Policy: default-src 'self';
```

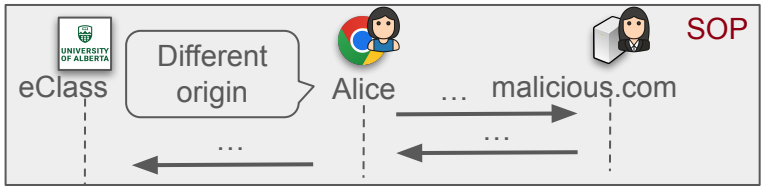
Analogous to default keyword in switch statements, when CSP is not defined

Site's own origin

Content Security Policy (CSP)

Content Security Policy (CSP) restricts which resources (e.g., JavaScript, CSS) can be loaded, and the URLs that they can be loaded from.

- Part of the HTTP response from the server



- Inverse version of Same Origin Policy (SOP)
  - SOP: preventing other sites from sending requests to eClass
  - CSP: preventing eClass from sending requests to other sites
- Goal: Mitigate the damage if the attacker gains access to user sessions
  - Extra layer of security to detect and mitigate XSS attacks
  - By only executing scripts loaded in source files received from those allowed domains, restrict all other scripts (including inline scripts or event-handling HTML attributes)

Common CSP directives and values

Definition of CSP: Content-Security-Policy: directives 'value';

Directives	Example	Description
default-src	default-src 'self'	Default policy to allow any resources (JavaScript, Fonts, CSS, etc) from the site's own origin

The directive default-src can be overwritten by more precise directives:

- E.g., default-src 'self'; script-src example.com
- Executable script is only allowed from example.com
- Overridden by script-src, and not inheriting 'self' from the default-src

Common CSP directives and values

Definition of CSP: Content-Security-Policy: directives 'value';

Directives	Example	Description
default-src	default-src 'self'	Default policy to allow any resources (JavaScript, Fonts, CSS, etc) from the site's own origin
img-src	img-src *	Allow images from anywhere (* as wildcard)

The directive img-src specifies that images may load from anywhere

- By using the wildcard annotation *

Common CSP directives and values

Definition of CSP: Content-Security-Policy: directives 'value';

Directives	Example	Description
default-src	default-src 'self'	Default policy to allow any resources (JavaScript, Fonts, CSS, etc) from the site's own origin
img-src	img-src *	Allow images from anywhere (* as wildcard)
script-src	script-src example.org example.com	Allow scripts from example.org and example.com

The directives exclude subdomains unless specified (whitelisting resources):

- Executable script is only allowed from example.org and example.com (excluding subdomains)
- Use wildcard * to specify all subdomains
    - E.g., script-src example.org *.example.org example.com *.example.com

Common CSP directives and values

Definition of CSP: Content-Security-Policy: directives 'value';

Directives	Example	Description
default-src	default-src 'self'	Default policy to allow any resources (JavaScript, Fonts, CSS, etc) from the site's own origin
img-src	img-src *	Allow images from anywhere (* as wildcard)
script-src	script-src 'self' example.com	Allow scripts from its own origin and example.com

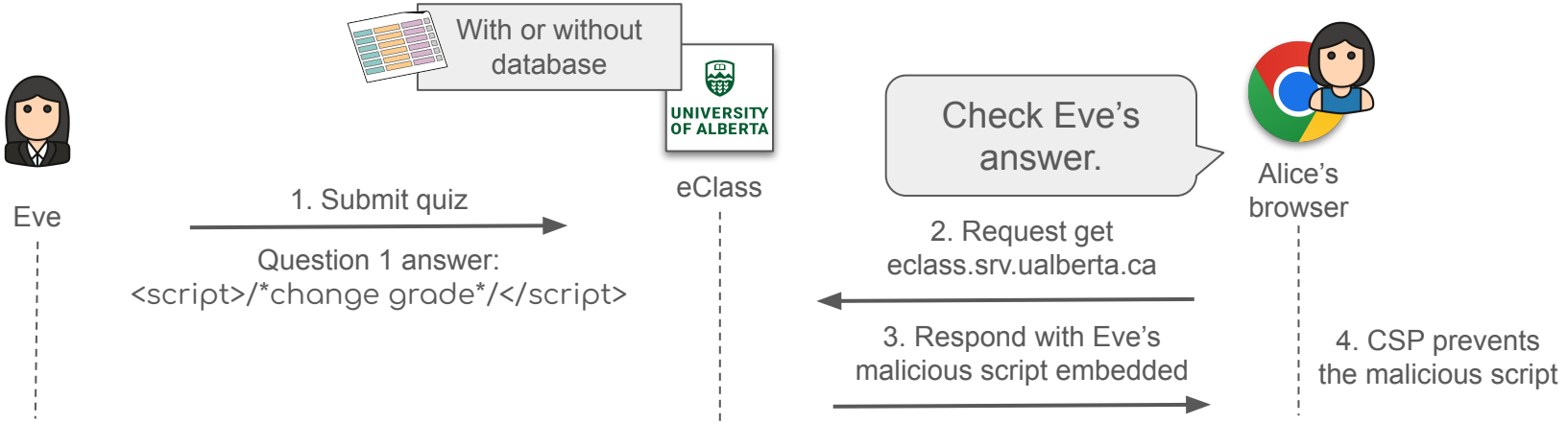
However, the script-src directive (when specified) blocks inline scripts and event handling attributes:

- Even if 'self' is defined
- To allow this, use 'unsafe-inline' in script-src, but it is bad
    - Same as not having CSP

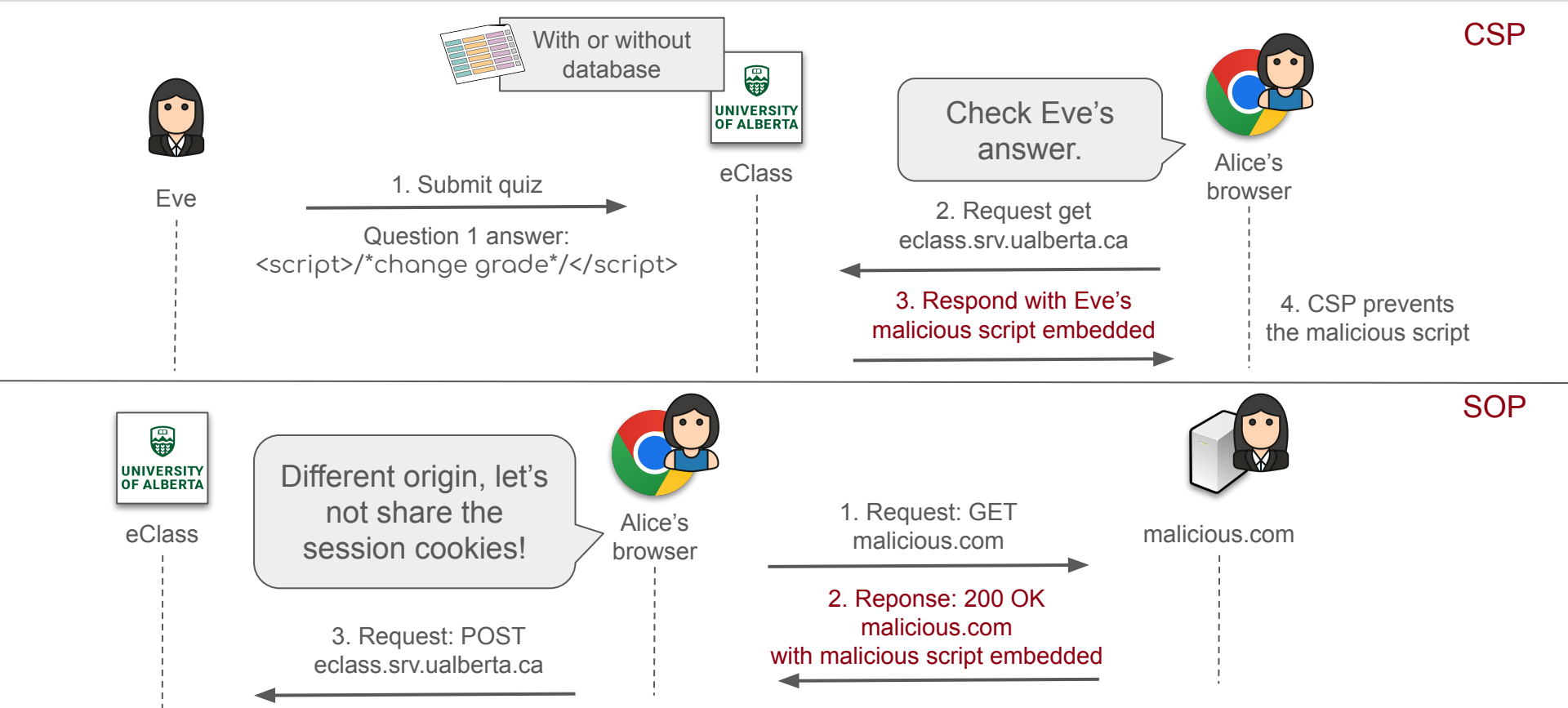
Reflected and stored XSS with CSP

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
- CSP checks the origin of the script: different origin – the malicious script is prevented (inline scripts are also prevented)!



CSP vs SOP


CSP
With or without database
UNIVERSITY OF ALBERTA
eClass
Eve
1. Submit quiz
Question 1 answer:
<script>/*change grade*/</script>
Check Eve's answer.
Alice's browser
2. Request get
eclass.srv.ualberta.ca
3. Respond with Eve's malicious script embedded
4. CSP prevents the malicious script
SOP
UNIVERSITY OF ALBERTA
eClass
Different origin, let's not share the session cookies!
Alice's browser
1. Request: GET
malicious.com
malicious.com
2. Reponse: 200 OK
malicious.com
with malicious script embedded
3. Request: POST
eclass.srv.ualberta.ca

More on CSP directives

Complete list of CSP directives available here. Some examples:

Directives	Description
default-src	Default policy
img-src	Restrict images or icon shortcut (i.e., favicons)
script-src	Restricts scripts
font-src	Restricts fonts
style-src	Restricts stylesheets (e.g., CSS)
object-src	Restricts sources for plugins (e.g., <object> in Flash)
media-src	Restricts media (e.g., <audio>, <video>)
base-uri	Restricts URLs in a document's <base> element

Use case for CSP

As a website admin, we want to allow images from any origin, but restrict video and audio to trusted providers, and all scripts only to a specific server that hosts trusted code:

- Starting point
  - Content-Security-Policy: default-src 'self';
- Images from any origin
  - Content-Security-Policy: default-src 'self'; img-src *;
- Video and audio from trusted providers
  - Content-Security-Policy: default-src 'self'; img-src *; media-src trusted.com;
- Scripts from a specific server
  - Content-Security-Policy: default-src 'self'; img-src *; media-src trusted.com; script-src script.server.com

Questions on CSP

Given Content-Security-Policy: default-src 'self' script-src 'self'

- Is <script src='/malicious.js'></script> allowed?
  - ❒ Yes
  - ❒ No
- Is <script src='https://malicious.com/malicious.js></script> allowed?
  - ❒ Yes
  - ❒ No
- Is <script>alert(document.cookie)</script> allowed?
  - ❒ Yes
  - ❒ No
- Is <svg onload='alert(document.cookie)'> allowed?
  - ❒ Yes
  - ❒ No

Schedule for today

- Key concepts from last classes
- Cross Site Scripting Prevention
  - Content Security Policy (CSP)
  - CSP Directives
  - Key concepts into practice
  - Challenge: Nested scripts
- Next class
  - “CSP is Dead” paper by Google in 2016
  - Additional CSP protections: strict-dynamic
- Polls for Week 12 - 13 materials

Key concepts into practice

Understanding UAlberta website:

- Network indicates some clarity.js scripts (demo) that are continuously running
  - Inspect - Network - Name - collect
  - Request URL: google-analytics; URL parameters: epn.percent_scrolled and gtm
  - Conclusion: scripts running the background that track user behaviors

Key concepts into practice

Implementing CSP on UAlberta website:

```
<script async
src='https://www.google-analytics.com/analytics.js'></script>

<script>
        window.GoogleAnalyticsObject = 'ga'
        function ga () { window.ga.q.push(arguments) }
        window.ga.q = window.ga.q || []
        window.ga.l = Date.now()
        window.ga('create', 'UA-XXXXXXX-XX', 'auto')
        window.ga('send', 'pageview')
</script>

Content-Security-Policy:
        default-src: 'self';
        script-src: 'self'
```

What problem can happen?

Key concepts into practice

Implementing CSP on UAlberta website:

```
<script async
src='https://www.google-analytics.com/analytics.js'></script>

<script>
        window.GoogleAnalyticsObject = 'ga'
        function ga () { window.ga.q.push(arguments) }
        window.ga.q = window.ga.q || []
        window.ga.l = Date.now()
        window.ga('create', 'UA-XXXXXXX-XX', 'auto')
        window.ga('send', 'pageview')
</script>

Content-Security-Policy:
        default-src: 'self';
        script-src: 'self' https://www.google-analytics.com
```

What problem can happen?

Problem 1

- CSP does not allow script from different origin

Solution

- Include googletagmanaager.com into script-src

Key concepts into practice

Implementing CSP on UAlberta website:

```
<script async
src='https://www.google-analytics.com/analytics.js'></script>

<script>
        window.GoogleAnalyticsObject = 'ga'
        function ga () { window.ga.q.push(arguments) }
        window.ga.q = window.ga.q || []
        window.ga.l = Date.now()
        window.ga('create', 'UA-XXXXXXX-XX', 'auto')
        window.ga('send', 'pageview')
</script>

Content-Security-Policy:
        default-src: 'self';
        script-src: 'self' https://www.google-analytics.com
```

What problem can happen?

Problem 2

- script-src blocks inline JavaScript;

Solution

- Move the inline script to /script.js
- Same origin, not inline script

Key concepts into practice

Implementing CSP on UAlberta website:

```
<script async
src='https://www.google-analytics.com/analytics.js'></script>

/*create /script.js*/
<script>
        window.GoogleAnalyticsObject = 'ga'
        function ga () { window.ga.q.push(arguments) }
        window.ga.q = window.ga.q || []
        window.ga.l = Date.now()
        window.ga('create', 'UA-XXXXXXX-XX', 'auto')
        window.ga('send', 'pageview')
</script>

Content-Security-Policy:
        default-src: 'self';
        script-src: 'self' https://www.google-analytics.com
```

What problem can happen?

Key concepts into practice

Implementing CSP on UAlberta website:

```
<script async
src='https://www.google-analytics.com/analytics.js'></script>

/*create /script.js*/
<script>
        window.GoogleAnalyticsObject = 'ga'
        function ga () { window.ga.q.push(arguments) }
        window.ga.q = window.ga.q || []
        window.ga.l = Date.now()
        window.ga('create', 'UA-XXXXXXX-XX', 'auto')
        window.ga('send', 'pageview')
</script>

Content-Security-Policy:
        default-src: 'self';
        script-src: 'self' https://www.google-analytics.com
```

What problem can happen?

- analytics.js can call other scripts that do not originate from google-analytics.com
- Those scripts with different origin will be blocked!
- … get worst when we have nested scripts inside nested scripts inside …

Schedule for today

- Key concepts from last classes
- Cross Site Scripting Prevention
  - Content Security Policy (CSP)
  - CSP Directives
  - Key concepts into practice
  - Challenge: Nested scripts
- Next class
  - “CSP is Dead” paper by Google in 2016
  - Additional CSP protections: strict-dynamic
- Polls for Week 12 - 13 materials

Challenge: Nested scripts

Problem in CSP's original design: How do we make sure that CSP is respected while new scripts can be executed from trusted sources?

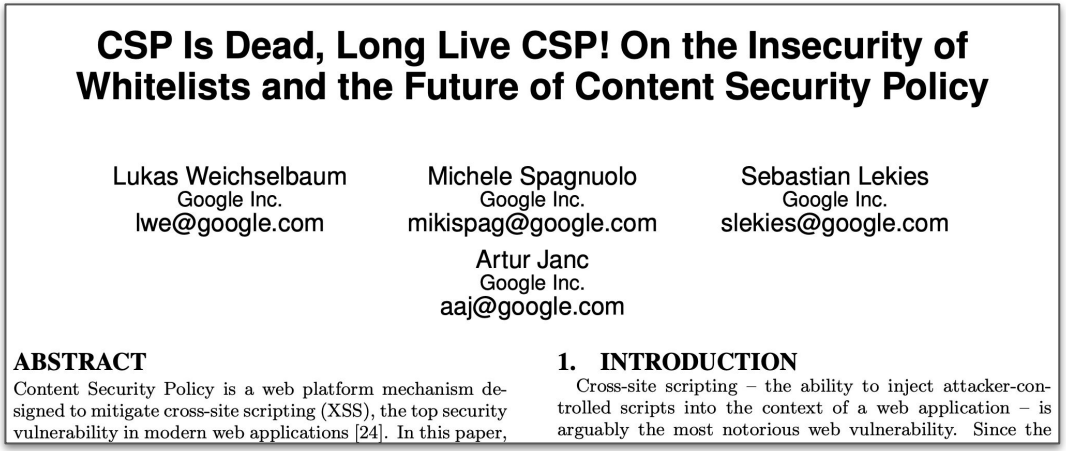
- Intuition: propagate trust from the initial script to any nested scripts

This can be achieved with more advanced CSP such as strict-dynamic

- Explicitly trust a script with a *nonce* or a *hash*, which shall be propagated to all the scripts loaded by that root script.
  - *Nonce* = similar to a secure random token
  - No longer need whitelisting resources

CSP Is Dead, Long Live CSP

In 2016, Google published a paper titled "CSP Is Dead, Long Live CSP" at CCS conference (presentation video available):



CSP Is Dead, Long Live CSP! On the Insecurity of Whitelists and the Future of Content Security Policy

Lukas Weichselbaum
Google Inc.
lwe@google.com

Michele Spagnuolo
Google Inc.
mikispag@google.com

Sebastian Lekies
Google Inc.
slekies@google.com

Artur Janc
Google Inc.
aaj@google.com

ABSTRACT

Content Security Policy is a web platform mechanism designed to mitigate cross-site scripting (XSS), the top security vulnerability in modern web applications [24]. In this paper,

1. INTRODUCTION

Cross-site scripting – the ability to inject attacker-controlled scripts into the context of a web application – is arguably the most notorious web vulnerability. Since the

- 94.68% of CSP that attempt to limit script execution are ineffective
- 99.34% of hosts use policies that offer no benefit against XSS

Solution: The use of strict-dynamic as a value in script-src

Schedule for today

- Key concepts from last classes
- Cross Site Scripting Prevention
  - Content Security Policy (CSP)
  - CSP Directives
  - Key concepts into practice
  - Challenge: Nested scripts
- Next class
  - “CSP is Dead” paper by Google in 2016
  - Additional CSP protections: strict-dynamic
- Polls for Week 12 - 13 materials

Poll (Vote here)

Poll link: https://xoyondo.com/ap/210mrue7s3bqh0t

- Database
  - Relational database, Apache Hadoop
- Networks
  - TCP, UDP, Denial of Service (DoS)
- Bitcoin
  - Blockchain, Mining principles
- Large-language models
  - Prompting, In-context learning
- Program analysis
  - Static and dynamic analysis, Program Slicing
- Automated testing (Selenium)
  - Automate testing, Web scripting; more practical, demo

Poll (Backup for round 2)

Poll link: https://xoyondo.com/ap/65tv9v8r1877pth

- Database
  - Relational database, Apache Hadoop
- Networks
  - TCP, UDP, Denial of Service (DoS)
- Bitcoin
  - Blockchain, Mining principles
- Large-language models
  - Prompting, In-context learning
- Program analysis
  - Static and dynamic analysis, Program Slicing
- Automated testing (Selenium)
  - Automate testing, Web scripting; more practical, demo