

Securing Web Applications with FFP (FAUST, flagbot, polygl0ts)

Javascript and XSS

Moritz Schneider²

In collaboration with ¹FAUST (ctf@fsi.informatik.uni-erlangen.de), ²flagbot (ctf@vis.ethz.ch) and ³polygl0ts (ctf@epfl.ch)

October 9, 2021



Table of Contents

What is XSS?

Types of XSS

Mitigations

Workshop challenges

What is XSS?



What is XSS?

- ▶ XSS stands for *cross-site scripting*.
 - ▶ type of attack where a malicious user can inject scripts into a web page.



What is XSS?

- ▶ XSS stands for *cross-site scripting*.
 - ▶ type of attack where a malicious user can inject scripts into a web page.
- ▶ Why is it dangerous?
 - ▶ Can steal session cookies (if cookie not httpOnly)
 - ▶ Change content on the web page
 - ▶ redirect user
 - ▶ Log every interaction on page
 - ▶ make requests



Types of XSS



Types of XSS

- ▶ Reflected XSS
 - ▶ Payload is store in a URL parameter and "reflected" on the page.



Types of XSS

- ▶ Reflected XSS
 - ▶ Payload is store in a URL parameter and "reflected" on the page.

Demo

Simple reflected XSS



Types of XSS

- ▶ Persisted XSS
 - ▶ Payload is stored on the server
 - ▶ Common examples: Comments, Avatar



Types of XSS

- ▶ Persisted XSS
 - ▶ Payload is stored on the server
 - ▶ Common examples: Comments, Avatar

Demo

Guestbook



Types of XSS

In both cases

Client or server did not properly sanitize the input.



Mitigations



Mitigation

- ▶ Sanitize user input
 - ▶ Use `innerText` instead of `innerHTML`
 - ▶ Escape template variables on the server (usually default behaviour)
 - ▶ don't just concatenate HTML
- ▶ Use CSP (Content Security Policy)



- ▶ The Content Security Policy specifies in a header (or rarely also a `<meta>` tag) what resources can be loaded and executed.

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

- ▶ Will only allow scripts loaded from the same origin and disallow inline scripts like `` or `<script>alert("XSS")</script>`.

- ▶ The Content Security Policy specifies in a header (or rarely also a `<meta>` tag) what resources can be loaded and executed.

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

- ▶ Will only allow scripts loaded from the same origin and disallow inline scripts like `` or `<script>alert("XSS")</script>`.
- ▶ Can also be used to restrict style loading, fonts, images, etc.
- ▶ Can automatically report violations


CSP Bypasses

- ▶ Often CSP is too lax and can be bypassed.
 - ▶ CDNs
 - ▶ JSONP endpoints
 - ▶ Certain frameworks



CSP Bypasses

► There is a tool to check your CSP¹:



CSP Evaluator

CSP Evaluator allows developers and security experts to check if a Content Security Policy (CSP) serves as a strong mitigation against [cross-site scripting attacks](#). It assists with the process of reviewing CSP policies, which is usually a manual task, and helps identify subtle CSP bypasses which undermine the value of a policy. CSP Evaluator checks are based on a [large-scale study](#) and are aimed to help developers to harden their CSP and improve the security of their applications. This tool (also available as a [Chrome extension](#)) is provided only for the convenience of developers and Google provides no guarantees or warranties for this tool.

Content Security Policy

[Sample unsafe policy](#) [Sample safe policy](#)

```
script-src 'unsafe-inline' 'unsafe-eval' 'self' data: https://www.google.com http://www.google-analytics.com/gtm/js
https://*.gstatic.com/feedback/ https://ajax.googleapis.com;
style-src 'self' 'unsafe-inline' https://fonts.googleapis.com https://www.google.com;
default-src 'self' * 127.0.0.1 https://[2a00:79e8:1b2:b466:3fd9:dc72:f00e]/foobar;
img-src https: data:;
child-src data:;
foobar-src 'foobar';
report-uri http://csp.example.com;
```

CSP Version 3 (nonce based + backward compatibility checks) ▼ @

CHECK CSP

Evaluated CSP as seen by a browser supporting CSP Version 3 [expand/collapse all](#)

● script-src	Host whitelists can frequently be bypassed. Consider using 'strict-dynamic' in combination with CSP nonces or hashes.	▼
✓ style-src		▼
● default-src		▼
✓ img-src		▼
✓ child-src		▼
✗ foobar-src	Directive "foobar-src" is not a known CSP directive.	▼
● report-uri		▼
⚠ object-src [missing]	Can you restrict object-src to 'none'?	▼
⚠ require-trusted-types-for [missing]	Consider requiring Trusted Types for scripts to lock down DOM XSS injection sinks. You can do this by adding "require-trusted-types-for 'script'" to your policy.	▼

¹<https://csp-evaluator.withgoogle.com/>

Workshop challenges



Hands on challenges

- ▶ Challenge 1
 - ▶ Reflected XSS
 - ▶ Steal the admin session
 - ▶ Get the email
- ▶ Challenge 2
 - ▶ Persisted XSS
 - ▶ Steal the admin session
 - ▶ Read the guestbook entry
- ▶ Challenge 3
 - ▶ Exfiltrate the path of the image that only admin can view
 - ▶ Session cookie is httpOnly!
 - ▶ extract HTML-text
 - ▶ beware: `innerHTML` doesn't execute `<script>` Tags
 - ▶ but you can use event handlers ``



XSS Helper

- ▶ To have an admin visit a page, submit it on `/xss-helper`
- ▶ You can use the "Cookie Jar" for exfiltration
- ▶ Setting a cookie can be done via the dev tools



Let's gooooo

► Tasks: <http://ffp-ctf.0x4141.net/>



More resources

- ▶ <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/script-src>
- ▶ <https://book.hacktricks.xyz/pentesting-web/content-security-policy-csp-bypass>
- ▶ <https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/XSS>

