

Let's Break Stuff

---

# Content Security Policies

Because you all totally care about this, right?!

---

## About me

- ▶ Senior Software Engineer at Viva IT  
(those folks in orange hoodies at some conferences & events you may have been to)
- ▶ @Brunty
- ▶ @PHPem
- ▶ mfyu.co.uk
- ▶ matt@mfyu.co.uk

## Things I do

- ▶ Dungeon master for D&D campaigns
- ▶ Mentor, lead & teach apprentices & junior developers
- ▶ Run & organise PHP East Midlands
- ▶ Speak at user groups and conferences
- ▶ **Break production sites with incorrectly configured content security policies**

**Disclaimer:  
I don't work with WordPress**

Oh good, finally we're getting started

---

## A talk in 3 parts

- ▶ XSS

- ▶ CSP

- ▶ Break stuff

# The scary stuff

## What is Cross-Site-Scripting (XSS)?

- ▶ XSS enables an attacker to inject client-side scripts into non-malicious web pages viewed by other users
- ▶ In 2016 there was a 61% likelihood of a **browser-based** vulnerability being found in a web application
- ▶ Of those browser based vulnerabilities, 86% were found to be XSS related
- ▶ That's just over 52% of all web application vulnerabilities

<https://www.edgescan.com/assets/docs/reports/2016-edgescan-stats-report.pdf>

I mean, it's just a joke vulnerability, right?!

---

## What can be done with XSS?

- ▶ Put pictures of cats in web pages
- ▶ `alert('💩');`
- ▶ Rickroll a user
- ▶ Twitter self-retweeting tweet  
<https://www.youtube.com/watch?v=zv0kZKC6GAM>
- ▶ Samy worm  
[https://en.wikipedia.org/wiki/Samy\\_\(computer\\_worm\)](https://en.wikipedia.org/wiki/Samy_(computer_worm))



## What can be done with XSS?

- ▶ Make modifications to the DOM
- ▶ Load additional scripts, resources, styles, images etc
- ▶ Access HTML5 APIs - webcam, microphone, geolocation
- ▶ Steal cookies (and therefore steal session cookies)

📍 Location	Ask (default) ⚡
📷 Camera	Allow ⚡
🎤 Microphone	Allow ⚡
🔔 Notifications	Ask (default) ⚡
⌘ JavaScript	Allow (default) ⚡
🧩 Flash	Ask (default) ⚡
🖼️ Images	Allow (default) ⚡
🔗 Popups	Block (default) ⚡
🔄 Background Sync	Allow (default) ⚡
⬇️ Automatic downloads	Ask (default) ⚡
🎹 MIDI devices full control	Ask (default) ⚡

📍 Location	Ask (default) ⚡
📷 Camera	<div style="border: 1px solid #ccc; padding: 5px;"> <ul style="list-style-type: none"> <li>Use global default (Ask)</li> <li>✓ Always allow on this site</li> <li>Always block on this site</li> </ul> </div>
🎤 Microphone	Allow ⚡
🔔 Notifications	Ask (default) ⚡
⌘ JavaScript	Allow (default) ⚡
🧩 Flash	Ask (default) ⚡
🖼️ Images	Allow (default) ⚡
🔗 Popups	Block (default) ⚡
🔄 Background Sync	Allow (default) ⚡
⬇️ Automatic downloads	Ask (default) ⚡
🎹 MIDI devices full control	Ask (default) ⚡



It's really not a joke vulnerability

---

## What can be done with XSS?

KEVIN POULSEN SECURITY 03.28.08 08:00 PM

# HACKERS ASSAULT EPILEPSY PATIENTS VIA COMPUTER

<https://www.wired.com/2008/03/hackers-assault-epilepsy-patients-via-computer/>

### Stored XSS (AKA Persistent or Type I)

- ▶ Occurs when input is stored - generally in a server-side database, but not always
- ▶ This could also be within a HTML5 database, thus never being sent to the server at all
- ▶ who.is was a site Rickrolled by a TXT record in the DNS of a website (**yes, really**)

### Reflected XSS (AKA Non-persistent or Type II)

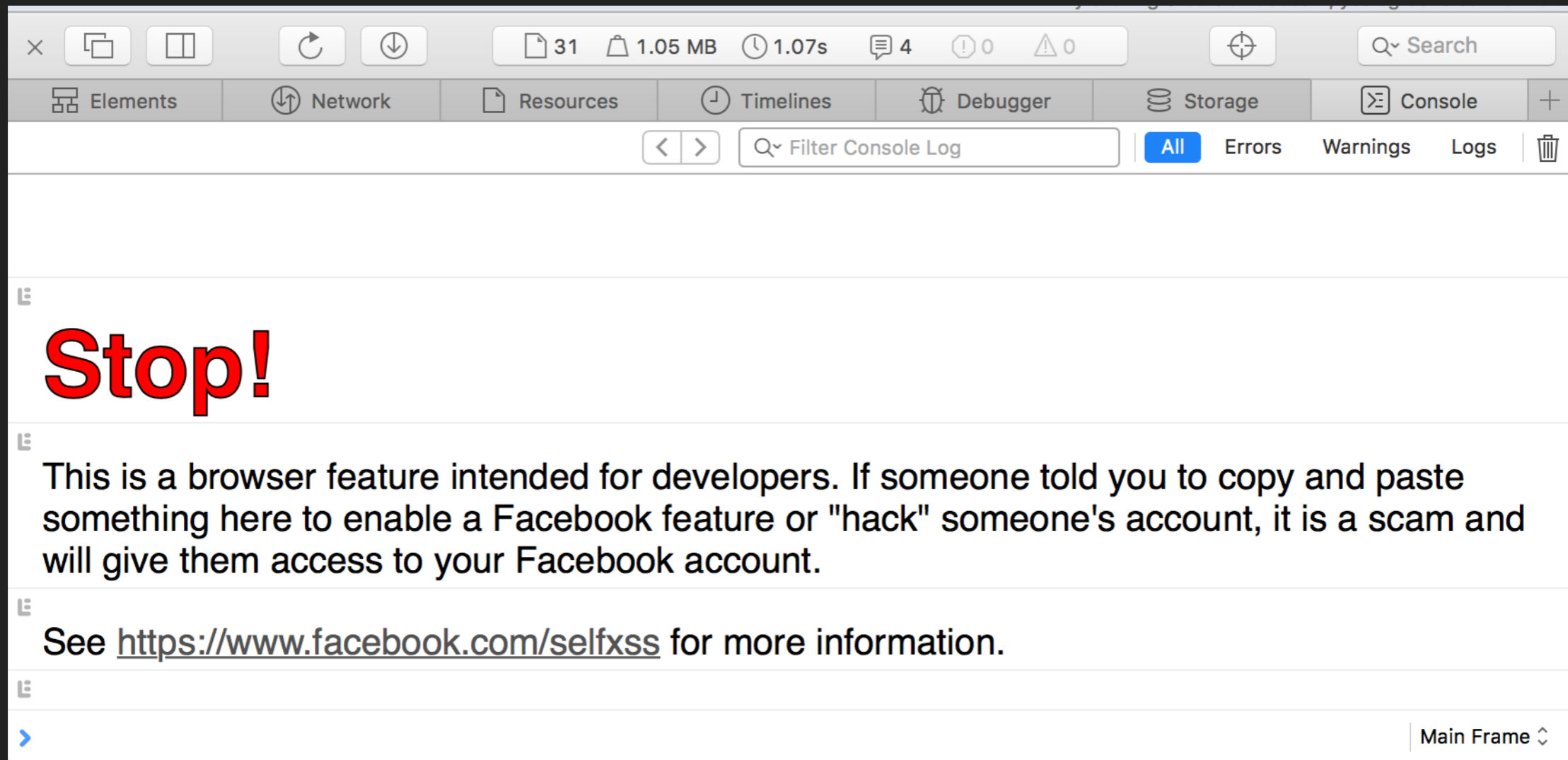
- ▶ Occurs when user input provided in the request is immediately returned - such as in an error message, search string etc
- ▶ Data is not stored, and in some instances, may not even reach the server (see the next type of XSS)

### DOM-Based XSS (AKA Type-0)

- ▶ The entire flow of the attack takes place within the browser
- ▶ For example, if JavaScript in the site takes input, and uses something like `document.write` based on that input, it can be vulnerable to a DOM-based XSS attack

### Self XSS

- ▶ Social-engineering form of XSS
- ▶ Requires the user to execute code in the browser
- ▶ Doing so via the console can't be protected by a lot of methods
- ▶ Not considered a 'true' XSS attack due to requiring the user to execute the code







# Let's fight back

Nottingham  
Hackspace  
Ask me about  
membership

# What is a CSP?

---

## HTTP response header to help reduce XSS risks

What is a CSP?

---

**It is not a silver bullet**

# What is a CSP?

---

**It is an extra layer of security**

# How does a CSP work?

---

**It declares what resources are allowed to load**

# Browser support

Header	Chrome	FireFox	Safari	IE	Edge
<code>Content-Security-Policy</code> <span>CSP Level 2</span>	40+ Full January 2015	31+ <i>Partial</i> July 2014	10+	-	Edge 15 build 15002+
<code>Content-Security-Policy</code> <span>CSP 1.0</span>	25+	23+	7+	-	Edge 12 build 10240+
<code>X-Content-Security-Policy</code> <span>Deprecated</span>	-	4+	-	10+ <i>Limited</i>	12+ <i>Limited</i>
<code>X-WebKit-CSP</code> <span>Deprecated</span>	14+	-	6+	-	-

Meh, it's alright(ish)  
Sorry IE users

## What can we protect?

- ▶ default-src
- ▶ script-src
- ▶ style-src
- ▶ img-src
- ▶ form-action
- ▶ update-insecure-requests



# Full reference:

<https://content-security-policy.com>

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy>

**img-src \***

**Allows any URL except data: blob:  
filesystem: schemes.**

```
object-src 'none'
```

**Don't load resources from any source**

```
style-src 'self'
```

**Allow loading from same scheme,  
host and port**

```
script-src 'unsafe-inline'
```

**Allows use of inline source elements such as style attribute, onclick, or script tag bodies**

# Don't use `unsafe-inline`

```
script-src 'self' 'nonce-$RANDOM'
```

```
<script nonce="$RANDOM">...</script>
```

```
Content-Security-Policy: default-src 'none'; script-  
src 'self' https://*.google.com 'nonce-random123';  
style-src 'self'; img-src 'self'; upgrade-insecure-  
requests; form-action 'self';
```



Learn from my mistakes

---

**I broke production with a bad CSP**







**Don't do what I did**

# Report-URI

**When a policy failure occurs, the browser sends a JSON payload to that URL**

```
{
  "csp-report": {
    "blocked-uri": "self",
    "document-uri": "https://mysite.com",
    "line-number": 1,
    "original-policy": "script-src 'self'",
    "script-sample": "try { for(var lastpass_iter=0; lastpass...",
    "source-file": "https://mysite.com",
    "violated-directive": "script-src 'self'"
  }
}
```

**report-uri.io**

Directive	Blocked URI	Raw	Count
All ▾	<input type="text" value="blocked hostname"/> <input type="text" value="blocked path"/>		All ▾
script-src	https://disqus.com/next/config.js	<a href="#">show/hide</a>	1 
script-src	eval	<a href="#">show/hide</a>	1 
script-src	https://c.disquscdn.com/next/embed/common.bundle.8acee1de90e869efdb244e45c7f66630.js	<a href="#">show/hide</a>	1 
script-src	eval	<a href="#">show/hide</a>	1 
script-src	eval	<a href="#">show/hide</a>	1 
script-src	https://c.disquscdn.com/next/embed/lounge.bundle.9becee0326ce4d1840f8985f1dc0ce21.js	<a href="#">show/hide</a>	1 

# Report-only



```
Content-Security-Policy-Report-Only: [policy]; report-  
uri https://app.report-uri.io/r/default/csp/reportOnly;
```

# Trial stuff before Enforcing

**There will be noise,  
lots of noise**

### Tips

- ▶ Have an easy and quick way to disable the CSP in production if required
- ▶ Better yet, have a way to switch it from enforced to report only so you can get violations reported to help you debug
- ▶ Add the CSP at an application level if you need a nonce

### Multiple policies

- ▶ They complicate things
- ▶ For a resource to be allowed, it must be allowed by all policies declared (problematic if an enforced policy)
- ▶ I tend to avoid them where possible on enforced policies
- ▶ But with report-only mode they can be very useful to deploy and test multiple policies at the same time (as nothing breaks for the user)

## Cryptographic nonces

- ▶ Don't generate multiple nonces in the same request (but **do** generate a new nonce on each separate request)
- ▶ If using a templating engine (such as twig) - add the nonce as a global so it's available in every template by default
- ▶ Write a helper in your template engine to generate script tags with a nonce if it's available

# The problem with CSPs and CMSs

# Plugins



**Inline scripts (without nonces) are the enemy**

---

**Stop building dodgy  
plugins, avoid inline  
scripts & eval**

**Troy Hunt**

---

**[look at] making  
WordPress more CSP  
friendly**

**Scott Helme**

**Inline scripts (without nonces) are the enemy**

**Demo time!**  
**Let's break stuff**

**@scott\_helme**

**He knows his stuff!**

**@mr\_goodwin**

**He first introduced me to  
what a CSP is**

## Links & further reading

- ▶ [https://www.owasp.org/index.php/Cross-site\\_Scripting\\_\(XSS\)](https://www.owasp.org/index.php/Cross-site_Scripting_(XSS))
- ▶ <https://content-security-policy.com>
- ▶ <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy>
- ▶ <https://report-uri.io>
- ▶ <https://scotthelme.co.uk/just-how-much-traffic-can-you-generate-using-csp/>
- ▶ <https://www.edgescan.com/assets/docs/reports/2016-edgescan-stats-report.pdf>
- ▶ <http://theharmonyguy.com/oldsite/2011/04/21/recent-facebook-xss-attacks-show-increasing-sophistication/>
- ▶ <https://github.com/Brunty/csp-demo>



**Thank you**

# Questions?

@Brunty

@PHPem

mfyu.co.uk

matt@mfyu.co.uk