Understanding and Mitigating the Security Risks of Content Inclusion in Web Browsers

PhD Thesis Defense

Sajjad Arshad

Friday, 12 April 2019
Content Inclusion on the Web

➔ Websites include various types of content to create interactive user interfaces
  ◆ JavaScript
  ◆ Cascading Style Sheets (CSS)

➔ 93% of the most popular websites include JavaScript from external sources
  ◆ JavaScript libraries are hosted on fast content delivery networks (CDNs)
  ◆ Integration with advertising networks, analytics frameworks, and social media

➔ Browser extensions enhance browsers with additional capabilities
  ◆ Fine-grained filtering of content
  ◆ Access cross-domain content
  ◆ Perform network requests
Security Risks

➔ *Malvertising* by third-party content
  ◆ Launch drive-by downloads
  ◆ Redirect visitors to phishing sites
  ◆ Generate fraudulent clicks on ads
Malvertising campaign targets Apple users with malicious code hidden in images

New malvertising group named VeryMal hijacked over five million web sessions to redirect Apple users to sites offering malware-laced software.

Double trouble: Two-pronged cyber attack infects victims with data-stealing trojan malware and ransomware

A ‘prolific’ malvertising campaign has been used to distribute the Vidar information stealer and GandCrab ransomware.
Security Risks

➔ **Malvertising** by third-party content
  ◆ Launch drive-by downloads
  ◆ Redirect visitors to phishing sites
  ◆ Generate fraudulent clicks on ads

➔ **Ad(vertisement) injection** by browser extensions
  ◆ Divert revenue from content publishers
  ◆ Harm the reputation of the publisher from the user’s perspective
  ◆ Expose users to malware and phishing
Ad(vertisement) Injection

Chrome extension with millions of users is now serving popup ads
Good extension turns bad and is now showing unwanted ads for an ad-blocker to millions of users.


Firms buy popular Chrome extensions to inject malware, ads
Are adware companies offering lucrative deals to acquire popular Chrome extensions -- and the trust of an extension’s users?

By Charlie Osborne for Between the Lines | January 20, 2014 -- 11:19 GMT (03:19 PST) | Topic: Google
Security Risks

➔ *Malvertising* by third-party content
  ◆ Launch drive-by downloads
  ◆ Redirect visitors to phishing sites
  ◆ Generate fraudulent clicks on ads

➔ *Ad(vertisement) injection* by browser extensions
  ◆ Divert revenue from content publishers
  ◆ Harm the reputation of the publisher from the user’s perspective
  ◆ Expose users to malware and phishing

➔ *Style injection* by relative path overwrite (RPO)
  ◆ Sniffing users’ browsing histories
  ◆ Exfiltrate secrets from the website
Thesis Contributions

In this thesis, I investigate the feasibility and effectiveness of novel approaches to understand and mitigate the security risks of content inclusion for website publishers as well as their users. I show that our novel techniques are complementary to the existing defenses.

➔ Detection of Malicious Third-Party Content Inclusions ⇒ *Excision*
➔ Identifying Ad Injection in Browser Extensions ⇒ *OriginTracer*
➔ Analysis of Style Injection by Relative Path Overwrite (RPO)
Excision
Detection of Malicious Third-Party Content Inclusions
Third-Party Content Defenses

➔ Same-origin policy (SOP)
➔ iframe-based isolation
➔ Language-based isolation
➔ Policy enforcement
➔ Content Security Policy (CSP)
Content Security Policy

Content-Security-Policy: default-src 'self'; script-src: js.trusted.com

- Access control policy sent by web apps, enforced by browsers
- Whitelist of allowed origins to load resource from
- ISPs and browser extensions modify CSP rules
- Non-trivial to deploy
  - Ad syndication or real-time ad auctions
  - Arbitrary third-party resource inclusion by Browser extensions
Excision

Block malicious content **automatically** before it attacks the browser
Inclusion Tree


```html
<html>
<head><title>.../title</head>
<body>
  <ul><li>.../li</ul>
  <a href="..."></a>
  <div>
    <script src="script.js"></script>
    <img src="b.net/img.jpg">
    <script src="c.org/script.js"></script>
    <link href="c.org/style.css">
  </div>
  <img src="c.org/style.css"/></script>
  <script src="d.com/script.js"></script>
  <iframe src="e.net/frame.html">
    <html>
      <head></head>
      <body>
        <script>.../script>
        <object data="f.org/flash.swf"></object>
      </body>
    </html>
  </iframe>
  <script src="g.com/script.js"></script>
  <img src="h.org/img.jpg"/>
</body>
</html>
```
Inclusion Sequence Classification

Goal: Given trained models, label inclusion sequences as either benign or malicious

➔ Hidden Markov Models (HMM)
  ◆ Model inter-dependencies between resources in the sequence
  ◆ Trained one HMM for the benign class and one for the malicious class
  ◆ 20 states that are fully connected

➔ Training HMM is computationally expensive, but computing the likelihood of a sequence is instead very efficient
  ◆ Good choice for real-time classification
Classification Features

\[ R_0 \rightarrow R_1 \rightarrow \ldots \rightarrow R_n \Rightarrow [F_0, \ldots, F_{24}] \rightarrow [F_0, \ldots, F_{24}] \rightarrow \ldots \rightarrow [F_0, \ldots, F_{24}] \]

➔ Convert the inclusion sequence into sequence of feature vectors
➔ 12 feature types from three categories (DNS, String, Role)
  ◆ Compute *individual* and *relative* features for each type (24 features)
  ◆ Continuous features are normalized on [0-1] and discretized
➔ Continuous relative feature values are computed by comparing the individual value of the resource to its parent's individual value
  ◆ less, equal, or more
  ◆ *none* for the root resource
DNS Features

➔ Domain Level
  ◆ www.google.com has level 2
  ◆ Divide by maximum allowed levels (126)

➔ Alexa Ranking
  ◆ Divide the ranking by 1M

➔ Top-level Domain (TLD)

➔ Host Type
# DNS Features (TLD)

## Individual

<table>
<thead>
<tr>
<th>Value</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>IPs, Extensions</td>
</tr>
<tr>
<td>gen</td>
<td>*.com, *.org</td>
</tr>
<tr>
<td>gen-subdomain</td>
<td>*.us.com</td>
</tr>
<tr>
<td>cc</td>
<td>*.us, *.de, *.cn</td>
</tr>
<tr>
<td>cc-subdomain</td>
<td>*.co.uk, *.com.cn</td>
</tr>
<tr>
<td>cc-int</td>
<td>*.xn--p1ai (ru)</td>
</tr>
<tr>
<td>other</td>
<td>*.biz, *.info</td>
</tr>
</tbody>
</table>

## Relative

<table>
<thead>
<tr>
<th>Value</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>root resource</td>
</tr>
<tr>
<td>{got, lost}-tld</td>
<td>Ext. → *.de, *.us → IP</td>
</tr>
<tr>
<td>gen-to-{cc,other}</td>
<td><em>.org → {</em>.de, *.info}</td>
</tr>
<tr>
<td>cc-to-{gen,other}</td>
<td><em>.uk → {</em>.com, *.biz}</td>
</tr>
<tr>
<td>other-to-{gen,cc}</td>
<td><em>.info → {</em>.net, *.uk}</td>
</tr>
<tr>
<td>same-{gen,cc,other}</td>
<td>*.com → *.com</td>
</tr>
<tr>
<td>diff-{gen,cc,other}</td>
<td>*.info → *.biz</td>
</tr>
</tbody>
</table>
# DNS Features (Type)

<table>
<thead>
<tr>
<th>Individual Value</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>ipv6</td>
<td>2607:f0d0:::4</td>
</tr>
<tr>
<td>ipv4-private</td>
<td>192.168.0.1</td>
</tr>
<tr>
<td>ipv4-public</td>
<td>4.2.2.4</td>
</tr>
<tr>
<td>extension</td>
<td>Ext. Scripts</td>
</tr>
<tr>
<td>dns-sld</td>
<td>google.com</td>
</tr>
<tr>
<td>dns-sld-sub</td>
<td><a href="http://www.google.com">www.google.com</a></td>
</tr>
<tr>
<td>dns-non-sld</td>
<td>abc.dyndns.org</td>
</tr>
<tr>
<td>dns-non-sld-sub</td>
<td>a.b.dyndns.org</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Relative Value</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>root resource</td>
</tr>
<tr>
<td>same-site</td>
<td>w.google.com → ad.google.com</td>
</tr>
<tr>
<td>same-sld</td>
<td>1.dyndns.org → 2.dyndns.org</td>
</tr>
<tr>
<td>same-company</td>
<td>ad.google.com → <a href="http://www.google.de">www.google.de</a></td>
</tr>
<tr>
<td>same-eff-tld</td>
<td>bbc.co.uk → london.co.uk</td>
</tr>
<tr>
<td>same-tld</td>
<td>bbc.co.uk → london.uk</td>
</tr>
<tr>
<td>different</td>
<td>google.com → facebook.net</td>
</tr>
</tbody>
</table>
String Features

➔ Non-alphabetic characters
➔ Unique characters
➔ Character frequency
➔ Length
➔ Entropy
Role Features

➢ Three binary features
  ◆ Ad network
  ◆ Content delivery network (CDN)
  ◆ URL shortening service

➢ Manually compiled list
Implementation

➔ Modifications to Chromium browser
  ◆ Blink
  ◆ Extension Engine
➔ ~1,000 SLoC (C++) and several lines of JavaScript
➔ Tracking the start and termination of JavaScript execution
➔ Tracking content scripts injection and execution
➔ Tracks network requests
➔ Callbacks registered for events and timers
Data Collection

➔ Alexa Top 200K from June 2014 to May 2015
➔ Alexa Top 20 shopping sites with 292 ad-injecting extensions for one week in June 16th-22nd, 2015
➔ Anti-cloaking
➔ Anti-fingerprinting

<table>
<thead>
<tr>
<th>Item</th>
<th>Website Crawl</th>
<th>Extension Crawl</th>
</tr>
</thead>
<tbody>
<tr>
<td>Websites Crawled</td>
<td>234,529</td>
<td>20</td>
</tr>
<tr>
<td>Unavailable Websites</td>
<td>7,412</td>
<td>0</td>
</tr>
<tr>
<td>Unique Inclusion Trees</td>
<td>47,789,268</td>
<td>35,004</td>
</tr>
<tr>
<td>Unique Inclusion Sequences</td>
<td>27,261,945</td>
<td>61,489</td>
</tr>
<tr>
<td>Unique URLs</td>
<td>546,649,590</td>
<td>72,064</td>
</tr>
<tr>
<td>Unique Domains</td>
<td>1,368,021</td>
<td>1,144</td>
</tr>
<tr>
<td>Unique Sites</td>
<td>459,615</td>
<td>749</td>
</tr>
<tr>
<td>Unique SLDs</td>
<td>419,119</td>
<td>723</td>
</tr>
<tr>
<td>Unique Companies</td>
<td>384,820</td>
<td>719</td>
</tr>
<tr>
<td>Unique Effective TLDs</td>
<td>1,115</td>
<td>21</td>
</tr>
<tr>
<td>Unique TLDs</td>
<td>404</td>
<td>21</td>
</tr>
<tr>
<td>Unique IPs</td>
<td>9,755</td>
<td>3</td>
</tr>
</tbody>
</table>
Building Labeled Dataset

Trained classifiers using VirusTotal as ground truth

- host is reported malicious by at least 3 out of the 62 URL scanners

<table>
<thead>
<tr>
<th>Dataset</th>
<th>No. of Inclusion Sequences</th>
<th>No. of Terminal Domains</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Website Crawl</td>
<td>Extention Crawl</td>
</tr>
<tr>
<td>Benign</td>
<td>3,706,451</td>
<td>7,372</td>
</tr>
<tr>
<td>Malicious</td>
<td>25,153</td>
<td>19</td>
</tr>
</tbody>
</table>
Detection Results

→ 10-fold cross-validation
  ◆ FP ⇒ 0.59%
  ◆ FN ⇒ 6.61%

→ Features effectiveness
  ◆ D ⇒ DNS
  ◆ S ⇒ String
  ◆ R ⇒ Role
Comparison with URL Scanners

➔ Compared detection results on new data from June 1 to July 14, 2015
➔ Found 89 suspicious hosts that were likely dedicated redirectors
  ◆ 44% were recently registered in 2015
  ◆ 23% no longer resolve to an IP address
➔ Detected 177 new malicious hosts later reported in VT
  ◆ 78% of the malicious hosts were not reported during the first week
Early Detection
Performance

➔ Automatically visited the Alexa Top 1K with original and modified Chromium browsers for 10 times
➔ Installed 5 popular Chrome extensions
  ◆ Adblock Plus, Google Translate, Google Dictionary, Evernote WebClipper, and Tampermonkey
➔ Average 12.2% page latency overhead
➔ 3.2 seconds delay on browser startup time
Usability

➔ 10 students that self-reported as expert Internet users
➔ Each participant explored 50 random websites from Alexa Top 500
  ◆ Excluded websites requiring a login or involving sensitive subject matter
➔ Out of 5,129 web pages visited:
  ◆ 31 malicious inclusions
  ◆ 83 errors (mostly high latency resource loads)
➔ No broken extension was reported
OriginTracer
Identifying Ad Injection in Browser Extensions
Ad Injection
Ad Injection
Motivation

➔ Centralized dynamic analysis is non-trivial
  ◆ Hiding behaviors during the analysis time, triggering ad injection
➔ Third-party content injection or modification is quite common
➔ Non-trivial to delineate between wanted and unwanted behavior

Users are best positioned to make this judgment

<table>
<thead>
<tr>
<th>Extension</th>
<th>No. of Users</th>
<th>Injected Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adblock Plus</td>
<td>10,000,000+</td>
<td>&lt;iframe&gt;</td>
</tr>
<tr>
<td>Google Translate</td>
<td>6,000,000+</td>
<td>&lt;div&gt;</td>
</tr>
<tr>
<td>Tampermonkey</td>
<td>5,800,000+</td>
<td>&lt;img&gt;</td>
</tr>
<tr>
<td>Evernote Web Clipper</td>
<td>4,300,000+</td>
<td>&lt;iframe&gt;</td>
</tr>
<tr>
<td>Google Dictionary</td>
<td>3,000,000+</td>
<td>&lt;div&gt;</td>
</tr>
</tbody>
</table>
OriginTracer

OriginTracer adds fine-grained content provenance tracking to the web browser

➔ Provenance tracked at level of individual DOM elements
➔ Indicates origins contributing to content injection and modification
➔ Trustworthy communication of this information to the user
Provenance Labels

→ Labels are generalizations of web origins

\[ L = \langle S, I, P, X \rangle \]
\[ S = \{ \text{scheme} \} \cup \{ \text{“extension”} \} \]
\[ I = \{ \text{host} \} \cup \{ \text{extension-identifier} \} \]
\[ P = \{ \text{port} \} \cup \{ \text{null} \} \]
\[ X = \{ 0, 1, 2, \ldots \} \]
Label Propagation

1. Publisher
   - \{l_0\}

2. Script Host 1
   - \{l_1\}

3. DOM
   - \{l_0, l_1\}

4. Script Host 2
   - \{l_2\}

5. Extension
   - \{l_3\}

Browser

- HTML
- JS

\{l_0, l_1, l_2\}
Provenance Indicators

Provenance must be communicated to the user in a trustworthy and an easy-to-comprehend way
Implementation

➔ Modifications to Chromium browser
➔ ~900 SLoC (C++), several lines of JavaScript
➔ Mediates DOM APIs for node creation and modification
➔ Mediates node insertion through document writes
➔ Callbacks registered for events and timers
User Study Setup

➔ Study population: 80 students of varying technical sophistication
➔ Participants exposed to six Chromium instances (unmodified and modified), each with an ad-injecting extension installed
   ◆ Auto Zoom, Alpha Finder, X-Notifier, Candy Zapper, uTorrent, Gethoneybadger
➔ Participants were asked to visit three retail websites
   ◆ Amazon, Walmart, Alibaba
Reported Injected Ads

Are users able to correctly recognize injected advertisements?

![Graphs showing the percentage of reported injected ads for different applications and advertising brands.](image-url)
Susceptibility to Ad Injection

Are users generally willing to click on the advertisements presented to them?

![Box plot showing willingness to click on ads.]
Ability to Identify Injected Ads

Do content provenance indicators assist users in recognizing injected advertisements?

![Graph showing reported injected ads in unassisted and assisted conditions.](image-url)
Usability of Content Provenance

Would users be willing to adopt a provenance tracking system to identify injected advertisements?
Performance

➔ Configured an unmodified Chromium and OriginTracer instance to visit the Alexa Top 1K
   Broad spectrum of static and dynamic content on most-used websites
   Browsers configured with five benign extensions
➔ Average 10.5% browsing latency overhead
➔ No impact on browser start-up time
Usability

- Separate user study on 13 students of varying technical background
- Asked participants to browse 50 websites out of Alexa Top 500
- Asked users to report errors
  - Type I: browser crash, page doesn’t load, etc.
  - Type II: abnormal load time, page appearance not as expected
- Out of almost 2K URLs, two Type I and 27 Type II errors were reported
- No broken extensions was reported
Analysis of Style Injection by Relative Path Overwrite (RPO)
Relative Path Overwrite (RPO)

➔ Browser’s interpretation of URL may be different than the web server
  ◆ Browsers basically treat URLs as file system-like paths
  ◆ *However, URL may not correspond to an actual server-side file system structure, or web server may internally rewrite parts of the URL*

➔ RPO exploits the semantic disconnect between browsers and web servers in interpreting relative paths ⇒ *path confusion*
  ◆ Injects style (CSS) instead of script (JS)
  ◆ Turns a simple *text injection* vulnerability into a *style sink*
  ◆ “Self-reference”: Attacked document uses “itself” as stylesheet

➔ Threat model of RPO resembles that of XSS
  ◆ e.g., steal sensitive information
Path Confusion

Web Page: http://example.com/dir/page.php
Relative Style: files/style.css
Absolute Style: http://example.com/dir/files/style.css
Path Confusion

Web Page: http://example.com/dir/page.php
Relative Style: files/style.css
Absolute Style: http://example.com/dir/files/style.css

Web Page: http://example.com/dir/page.php/
Relative Style: files/style.css
Absolute Style: http://example.com/dir/page.php/files/style.css
Style Injection

Browser
http://example.com/"{background-image:url(...)}/

Server
Style Injection

Browser

```
http://example.com/{background-image:url(...)}/
```

Server

```
<html>
<head>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  Not found:
  http://example.com/{background-image:url(...)}/
</body>
</html>
```
Self-reference

Browser

http://example.com/*{background-image:url(...)}*/

Server
Self-reference

Browser

http://example.com/*{background-image:url(...)}*/

...<link rel="stylesheet" href="style.css">...

Server
Self-reference

Browser

http://example.com/*{background-image:url(...)}/

...<link rel="stylesheet" href="style.css"/>

Server

http://example.com/*{background-image:url(...)}/style.css
Self-reference

Browser

http://example.com/*{background-image:url(...)}/

...<link rel="stylesheet" href="style.css">

http://example.com/*{background-image:url(...)}/style.css

<html>
...
Not found:
http://example.com/*{background-image:url(...)}/style.css
...
</html>
Scriptless (Style-based) Attacks

→ Script injection is NOT always possible
   ◆ Input sanitization
   ◆ Browser-based XSS filters
   ◆ Content Security Policy (CSP)

→ Successful attacks are possible by injecting CSS
   ◆ Exfiltrating credit card number and CSRF tokens (Heiderich et al., CCS 2012)
     ● CSS attribute accessor, content property, animation features, media queries

→ Attacks typically consist of payload & injection technique

→ Our work is not concerned about the payload
   ◆ Focus on how to inject (“transport”) the payload
Preconditions for Successful Attack

1. Relative stylesheet path (no base tag) ⇒ Candidate Identification
2. Crafted URL causes style reflection in server response ⇒ Vulnerability Detection
3. Browser loads and interprets injected style directives ⇒ Exploitability Detection
Candidate Identification

➔ Common Crawl: extract pages with relative stylesheet path
  ◆ August 2016: >1.6B documents
  ◆ 203 M pages on nearly 6 M sites
➔ Filter 1: Alexa Top 1 million ranking
  ◆ 141 M pages on 223 K sites
➔ Filter 2: Group URLs using the same template
  ◆ Test only one random URL from each group
  ◆ 137 M pages on 222 K sites

<table>
<thead>
<tr>
<th>Group By</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query Parameter</td>
<td><a href="http://example.com/?lang=en">http://example.com/?lang=en</a></td>
</tr>
<tr>
<td></td>
<td><a href="http://example.com/?lang=fr">http://example.com/?lang=fr</a></td>
</tr>
<tr>
<td>Path Parameter</td>
<td><a href="http://example.com/028">http://example.com/028</a></td>
</tr>
<tr>
<td></td>
<td><a href="http://example.com/142">http://example.com/142</a></td>
</tr>
</tbody>
</table>
Vulnerability Detection

➔ Developed a lightweight crawler based on Python Requests API
  ◆ Randomly selects one representative URL from each group
  ◆ Mutates the URL according to a number of path confusion techniques
    ● PAYLOAD ⇒ %0A{.body{background:NONCE}}
  ◆ Requests the mutated URL
  ◆ Ignores the response if it contains <base> tag
  ◆ Extracts all relative stylesheet paths and expands them using the mutated URL
  ◆ requests each expanded stylesheet URL to find injected payload in the response
  ◆ Page would be vulnerable if at least one stylesheet’s response reflects the requested URL, referrer URL, parameters, or cookie

➔ Path confusion techniques
  ◆ Path Parameter, Encoded Path, Encoded Query, Cookie
  ◆ We assume the page references relative stylesheet path ../style.css
Path Confusion - Path Parameter

➔ Some web frameworks (e.g., PHP, ASP, JSP) accept the input parameters as a directory-like string following the name of the script in the URL
➔ Simply appends the payload as a subdirectory to the end of the URL
➔ CSS injection occurs if the page reflects page URL or referrer in the response

http://domain/dir/page.asp
http://domain/dir/page.asp/PAYLOAD//
http://domain/dir/page.asp/PAYLOAD/style.css
Path Confusion - Path Parameter

⇒ Different web frameworks handle path parameters differently, which is why we distinguish a few additional cases
  ◆ parameters separated by slashes in PHP/ASP and semicolons in JSP

http://domain/page.php/param
http://domain/page.php/PAYLOADparam/
http://domain/page.php/PAYLOADparam/style.css

http://domain/dir/page.jsp;param
http://domain/dir/page.jsp;PAYLOADparam/
http://domain/dir/page.jsp;PAYLOADparam/style.css
Path Confusion - Encoded Path

- This targets web servers such as IIS that decode encoded slashes in the URL for directory traversal, whereas web browsers DO NOT
- Use %2F, an encoded version of ‘/’, to inject our payload into the URL
- The canonicalized URL is equal to the original page URL
- CSS injection occurs if the page reflects page URL or referrer in the response

http://domain/dir/page.aspx
http://domain/PAYLOAD/..%2Fdir/PAYLOAD/..%2Fpage.aspx//
http://domain/PAYLOAD/..%2Fdir/PAYLOAD/..%2Fpage.aspx/style.css
Path Confusion - Encoded Query

➔ We replace the URL query delimiter ‘?’ with its encoded version %3F so that web browsers DO NOT interpret it as such
➔ We inject the payload into every value of the query string
➔ CSS injection happens if the page reflects either the URL, referrer, or any of the query values in the HTML response

http://domain/dir/page.html?key=value
http://domain/dir/page.html%3Fkey=PAYLOADvalue//
http://domain/dir/page.html%3Fkey=PAYLOADvalue/style.css
Path Confusion - Cookie

- Stylesheets referenced by a relative path are located in the same origin
  - Cookies are sent when requesting the stylesheet
- CSS injection may be possible if:
  - Attacker can create new cookies or tamper with existing ones, and
  - The page reflects cookie values in the response
- Payload is injected into each cookie value

```
http://domain/dir/page.php?key=value
http://domain/dir/page.php//?key=value
http://domain/dir/page.php/style.css

Original Cookie: name=val
Crafted Cookie: name=PAYLOADval
```
Exploitability Detection

➔ Verify whether the reflected CSS in the response is evaluated by the browser
   ◆ Built a crawler based on Google Chrome (and an extension for tainting cookie)
➔ Visit mutated vulnerable pages to check if injected style directives interpreted
   ◆ PAYLOAD ⇒ %0A}}]]}}body{background-image:url(/NONCE.gif)}
   ◆ Style is interpreted if injected image URL seen in network traffic
➔ Reflected CSS is not always interpreted by the browser
   ◆ Use of modern document types ⇒ browser doesn’t render page in quirks mode
➔ Overriding document types in Internet Explorer (IE)
   ◆ Load the page inside an iframe in Internet Explorer
   ◆ Used Fiddler for tainting cookies and recording HTTP requests/responses
   ◆ Turn on Compatibility View by setting “X-UA-Compliant” to “IE=EmulateIE7” via <meta> tag in the parent page
Limitations

➔ We only looked for reflected, not stored, injection of style directives
➔ Manual analysis of a site might reveal more opportunities for style injection that our crawler fails to detect automatically
➔ We did not analyze the vulnerability of pages not in Common Crawl
  ◆ We do not cover all sites, and we do not fully cover all pages within a site
➔ Results presented in this paper should be seen as a lower bound
➔ Our methodology can be applied to individual sites to analyze more pages
## Dataset

<table>
<thead>
<tr>
<th></th>
<th>Relative CSS</th>
<th>Alexa Top 1M</th>
<th>Candidate Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Pages</td>
<td>203,609,675</td>
<td>141,384,967</td>
<td>136,793,450</td>
</tr>
<tr>
<td>Tested Pages</td>
<td>53,725,270</td>
<td>31,448,446</td>
<td>30,991,702</td>
</tr>
<tr>
<td>Sites</td>
<td>5,960,505</td>
<td>223,212</td>
<td>222,443</td>
</tr>
<tr>
<td>Doc. Types</td>
<td>9,833</td>
<td>2,965</td>
<td>2,898</td>
</tr>
</tbody>
</table>
Six out of the ten largest sites are represented in our candidate set.

Candidate set contains a higher fraction of the largest sites and a lower fraction of the smaller sites.

Our results better represent the most popular sites, which receive most visitors, and most potential victims of RPO attacks.
Relative Stylesheet Paths

- CDF of total and maximum number of relative stylesheets per web page and site, respectively
- 63.1% of the pages contain multiple relative paths
  - Increases the chances of finding a successful RPO and style injection point
Vulnerability Analysis

➔ A page is vulnerable if its response:
  ◆ Reflects the injected CSS
  ◆ Does not contain a base tag
➔ 1.2% of pages are vulnerable to at least one of the injection techniques
➔ 5.4% of sites contain at least one vulnerable page
➔ Path parameter is the most effective technique against pages
➔ One third of the sites in Alexa Top 10, 8-10% in the Top 100K, and 4.9% in 100K-1M are vulnerable

<table>
<thead>
<tr>
<th>Technique</th>
<th>Pages</th>
<th>Sites</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path Parameter</td>
<td>309,079 (1.0%)</td>
<td>9,136 (4.1%)</td>
</tr>
<tr>
<td>Encoded Path</td>
<td>53,502 (0.2%)</td>
<td>1,802 (0.8%)</td>
</tr>
<tr>
<td>Encoded Query</td>
<td>89,757 (0.3%)</td>
<td>1,303 (0.6%)</td>
</tr>
<tr>
<td>Cookie</td>
<td>15,656 (&lt;0.1%)</td>
<td>1,030 (0.5%)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>377,043 (1.2%)</strong></td>
<td><strong>11,986 (5.4%)</strong></td>
</tr>
</tbody>
</table>
Base Tag

- Correctly configured base tag can prevent path confusion
- Base tag was removed after path confusion in 603 pages on 60 sites
- Internet Explorer fetches two URLs for stylesheet
  - One expanded according to the base URL specified in the tag
  - One expanded using the page URL as the base
- Internet Explorer always applied the “confused” stylesheet, even when the one based on the base tag URL loaded faster
Quirks Mode Doc. Types

➔ Browsers parse resources with a non-CSS content type when the page specifies a non-standard document type (or none at all)

➔ Total of 4,318 distinct doc. Types

➔ Roughly a third result in quirks mode
  ◆ 1,271 (29.4%) force all the browsers into quirks mode
  ◆ 1,378 (31.9%) cause at least one browser to use quirks mode

➔ Internet Explorer’s framing trick forced 4,248 (98.4%) into quirks mode

<table>
<thead>
<tr>
<th>Browser</th>
<th>Version</th>
<th>OS</th>
<th>Doc. Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chrome</td>
<td>55</td>
<td>Ubuntu 16.04</td>
<td>1,378 (31.9%)</td>
</tr>
<tr>
<td>Opera</td>
<td>42</td>
<td>Ubuntu 16.04</td>
<td>1,378 (31.9%)</td>
</tr>
<tr>
<td>Safari</td>
<td>10</td>
<td>macOS Sierra</td>
<td>1,378 (31.9%)</td>
</tr>
<tr>
<td>Firefox</td>
<td>50</td>
<td>Ubuntu 16.04</td>
<td>1,326 (30.7%)</td>
</tr>
<tr>
<td>Edge</td>
<td>38</td>
<td>Windows 10</td>
<td>1,319 (30.5%)</td>
</tr>
<tr>
<td>IE</td>
<td>11</td>
<td>Windows 7</td>
<td>1,319 (30.5%)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Doc. Type (shortened)</th>
<th>Pages</th>
<th>Sites</th>
</tr>
</thead>
<tbody>
<tr>
<td>(none)</td>
<td>1,818,595 (5.9%)</td>
<td>56,985 (25.6%)</td>
</tr>
<tr>
<td>&quot;-//W3C//DTD HTML 4.01 Transitional//EN&quot;</td>
<td>721,884 (2.3%)</td>
<td>18,648 (8.4%)</td>
</tr>
<tr>
<td>&quot;-//W3C//DTD HTML 4.0 Transitional//EN&quot;</td>
<td>385,656 (1.2%)</td>
<td>11,566 (5.2%)</td>
</tr>
<tr>
<td>&quot;-//W3C//DTD HTML 3.2 Final//EN&quot;</td>
<td>22,019 (&lt;0.1%)</td>
<td>1,175 (0.5%)</td>
</tr>
<tr>
<td>&quot;-//W3C//DTD HTML 3.2//EN&quot;</td>
<td>10,839 (&lt;0.1%)</td>
<td>927 (0.4%)</td>
</tr>
</tbody>
</table>

| All                  | 3,046,449 (9.6%) | 71,597 (32.2%) |
Standardized Doc. Types

➔ ~1K doc. types result in quirks mode
➔ ~3K doc. types cause standards mode
➔ But, number of standardized doc. types is several orders of magnitude smaller
  ◆ Only about 10 standards and quirks mode doc. types are widely used in sites
  ◆ Majority are not standardized
  ◆ Differ from the standardized ones only by small variations such as forgotten spaces or misspellings
➔ 9.6% of pages use quirks modes
➔ 32.2% of sites contain at least one page rendered in quirks mode
Exploitability Analysis

➔ Vulnerable pages that were exploitable
  ◆ **2.9%** in Chrome
  ◆ **14.5%** in Internet Explorer
    ● **5x** more than in Chrome

➔ 6 highest-ranked sites were not exploitable
  ◆ Between **1.6%** and **3.2%** of sites in the remaining buckets were exploitable

➔ IE is more effective *except in cookie*
  ◆ IE crawl was conducted one month later
  ◆ **Anti-framing** techniques
  ◆ **Anti-MIME-sniffing** header
Anti-Framing

1. **X-Frame-Options** response header (DENY, SAMEORIGIN, or ALLOW-FROM)
   - 4,999 vulnerable pages on 391 sites used it *correctly* and prevented the attack
   - However, 1,900 pages on 34 sites provided *incorrect values* for this header
     - Out of which, 552 pages on 2 sites were exploited in Internet Explorer

2. **frame-ancestors** directive in Content Security Policy (*not supported in IE*)
   - A whitelist of origins allowed to load the current page in a frame

3. Use JavaScript code to prevent framing of a page
   - i.e., redirecting the top frame if the page is not the top window itself
   - However, attackers can use the HTML5 sandbox attribute in the iframe tag and omit the allow-top-navigation directive to render JavaScript frame-busting code ineffective

We did not implement any of these techniques to allow framing, which means that more vulnerable pages could likely be exploited in practice
MIME Sniffing

➔ Many sites contain misconfigured content types
  ◆ Browsers attempt to infer the type based on the request context or file extension
    ● MIME sniffing, especially in quirks mode

➔ Setting **X-Content-Type-Options: nosniff** in response header block the request if the requested type is:
  ◆ "style" and the MIME type is not "text/css", or
  ◆ "script" and the MIME type is not, i.e., “application/javascript”

➔ Only Firefox, Internet Explorer, and Edge respected this header *at the time*
  ◆ Chrome started supporting the header since January 2018
  ◆ IE blocked our injected CSS payload when **nosniff** was set even with framing trick

➔ **96,618 vulnerable pages** across **232 sites** had a nosniff response header
  ◆ **23 pages** across **10 sites** were exploitable in Chrome but not in Internet Explorer
Many exploitable pages appeared to belong to well-known CMSes

- CMSes are installed on thousands of sites, fixing RPO vulnerability is impactful

Detected **23 CMSes** (Wappalyzer + manually)
- **41,288 pages** across **1,589 sites**

Installed the latest versions (or used the online demos)

Detected **4 exploitable** CMSes
- 1 declared no document type
- 1 used a quirks mode document type
- 2 were exploited in IE using framing trick
- **40,255 pages** across **1,197 sites** (nearly 32k sites world-wide)

Weaknesses were reported to the vendors
Mitigation Techniques

➔ Avoid path confusion
   ◆ Use only absolute (or root-relative) paths or <base> tag

➔ Avoid style injection
   ◆ Input sanitization (non-trivial)
     ● More targeted RPO attack variants can reference existing files

➔ Prevent stylesheets with syntax errors or no “text/css” content type
   ◆ Specify a modern document type: <!doctype html>
   ◆ Disable content type sniffing: X-Content-Type-Options

➔ Prevent Internet Explorer trick
   ◆ Disallow framing: X-Frame-Options
   ◆ Turn off compatibility view: X-UA-Compatible (IE=Edge)
Conclusion

➔ Excision
  ◆ Allows for preemptive blocking with moderate performance overhead
  ◆ Detected malicious hosts before appearing in the blacklists

➔ OriginTracer
  ◆ Allows users to make fine-grained trust decisions
  ◆ Evaluation shows it can be performed in an efficient and effective way

➔ RPO
  ◆ Style-based attacks require different countermeasures than XSS
  ◆ Easy-to-use and effective countermeasures exist to mitigate the attack
Third-party Content Inclusion ⇒ Excision

Ad Injection ⇒ OriginTracer

Relative Path Overwrite (RPO)
Deep Crawling

➔ The *Inclusion Tree* crawler has been evolving
   ◆ Written in NodeJS
   ◆ Uses Chrome Remote Debugging protocol

➔ Web Security
Deep Crawling

➔ Tracking and Privacy


◆ Muhammad Ahmad Bashir, Sajjad Arshad, Christo Wilson, “Recommended For You: A First Look at Content Recommendation Networks”, ACM Internet, Measurement Conference (IMC), 2016

◆ Muhammad Ahmad Bashir, Sajjad Arshad, Engin Kirda, William Robertson, Christo Wilson, “How Tracking Companies Circumvented Ad Blockers Using WebSockets”, ACM Internet Measurement Conference (IMC), 2018
Other Works

➔ Malware Detection

➔ Binary Analysis
◆ Reza Mirzazade farkhani, Saman Jafari, Sajjad Arshad, William Robertson, Engin Kirda, Hamed Okhravi, “On the Effectiveness of Type-based Control Flow Integrity”, Annual Computer Security Applications Conference (ACSAC), 2018
Thanks! Questions?

Sajjad Arshad

https://sajjadium.github.io/