Riding out DOMsday:
Toward Detecting and Preventing
DOM Cross-Site Scripting

William Melicher
Anupam Das
Mahmood Sharif
Lujo Bauer
Limin Jia

Carnegie Mellon University
XSS vulnerabilities account for 25% of web vulnerabilities

DOM XSS: vulnerability is inside JavaScript run on client

document.write('"<a href="' + document.location + '">Link</a>"');
Current client-side defenses are still inadequate

Example: CSP is often not configured properly

Example: Web application firewall filters easily bypassable

More promising solution: Detect bugs ahead of time

State of the art: taint tracking and recognize vulnerable flows [1]

Our contributions

1. Improved methodology for detecting DOM XSS
2. Studied prevalence of DOM XSS in real world
3. Examined whether static analysis tools help
What are vulnerable flows?

Sources: document.location, cross-origin messages, referrer, ...

Sinks: document.write, innerHTML, eval, ...

```javascript
var the_url = document.location.href;
var markup = '<a href="' + the_url + '">Link</a>'; 
document.write(markup);
```
What are vulnerable flows?

```javascript
var the_url = document.location.href;
var markup = '<a href="' + encodeURI(the_url) + '">Link</a>'; 
document.write(markup);
```
What are vulnerable flows?

```javascript
var the_url = document.location.href;
var markup = '<a href="' + encodeURI(the_url) + '">Link</a>";
document.write(markup);
```

Encoding function used
Detecting vulnerable flows using taint tracking

```
var markup = '<a href='' + document.location + ''>Link</a>';

'<a href="url.com/page#"></a><script>CODE</script>"Link</a>'
```
Taint tracking inside Chromium

```javascript
document.write('document.write('<a href="url.com/page#"></a><script>CODE</script>">Link</a>');
```

Log tainted call:
- Code location
- Value of tainted argument
- Taint information
- ...
Vulnerability confirmation: at-end injection

**Original URL:**
url.com/path?param=test&a=b

**Our confirmation URL:**
url.com/path?param=test&a=b#INJECT

document.write('<a href="' + document.location + '">Link</a>);

document.write('<a href="url.com/path?param=test&a=b">Link</a>);

document.write('<a href="url.com/path?param=test&a=b#INJECT">Link</a>);
Vulnerability confirmation: in-parameter injection

Original URL:
url.com/path?link=test&a=b

```
var data = getQueryParameter('link');
document.write('<a href="../' + data + '">Link</a>');
document.write('<a href="../test">Link</a>');
```

Our confirmation URL:
url.com/path?a=b#&link=INJECT&a=b

```
document.write('<a href="../INJECT">Link</a>');
```
Results
Our contributions

1. Improved methodology for detecting DOM XSS

2. Studied prevalence of DOM XSS in real world

3. Examined whether static analysis tools help
DOM XSS vulnerabilities on the Internet

- **10k** seed domains
- **45k** web pages
- **285k** flows URL sources to JS/HTML sinks
- **55k** flows after removing blocked by encoding
- **5,217** unique potentially vulnerable flows

Crawl 1-link deep subpages

Focus on a common category of exploitable flows

encodeURI, encodeURIComponent, ...

Uniqueness: domain, script URL, and script location
How we confirm potentially vulnerable flows

- **At-end method**: 715 unique confirmed vulnerable flows
- **In-parameter method**: 1,465 unique confirmed vulnerable flows
- **Both methods**: 1,039 unique confirmed vulnerable flows

Total: 3,219 unique confirmed vulnerable flows

83% more confirmed vulnerabilities using new in-parameter method
How are vulnerabilities distributed across domains?

1. Some very buggy domains
2. Long tail of many domains with one bug
How are vulnerabilities distributed by category?

Top 3 categories:
1. Web ads/analytics
2. News/media
3. Entertainment
What is causing the vulnerabilities?

- Simple concatenation without effort to sanitize data
  
  ```javascript
  document.write('<a href="' + document.location + '">Link</a>');
  ```

- Custom HTML templating code
  
  ```html
  '<a href="%s">Link</a>'
  ```

- Ad-hoc sanitization
  
  ```javascript
  if (markup.indexOf("<script>") != -1) ...  
  ```
Have things changed over time?

- Using same methodology as past experiment
- More flows per page: 92.6 vs. 48.5
- Larger ratio of vulnerabilities per page: 0.039 vs. 0.012
- Larger fraction of flows vulnerable: 0.04% vs. 0.03%

**Trend towards more DOM XSS vulnerabilities**

Our contributions

1. Improved methodology for detecting DOM XSS
2. Studied prevalence of DOM XSS in real world
3. Examined whether static analysis tools help
Can static analysis tools help?

What we did:
- Sampled confirmed vulnerabilities
- Checked if they are found by some off-the-shelf tools

No tool found more than 10% of vulnerabilities we tested
- Burp Suite found 10% and had 0% false positives, and found other bugs
- Other tools had high FP rate (95%)
Toward Detecting and Preventing DOM Cross-Site Scripting

- Improved measurement methodology for DOM XSS vulnerabilities
- Gained insight into causes and distribution of vulnerabilities
- Found that DOM XSS vulnerabilities may be increasing
- Showed that static analysis tools likely do not find many vulnerabilities

github.com/wrmelicher/ChromiumTaintTracking

William Melicher, Anupam Das, Mahmood Sharif, Lujo Bauer, Limin Jia
{billy, anupam, msharif, lbauer, limin}@cmu.edu

Carnegie Mellon University