From Facepalm to Brain Bender – Exploring Client-Side Cross-Site Scripting

Ben Stock, Stephan Pfistner, Bernd Kaiser, Sebastian Lekies, Martin Johns
About me and this talk

• Postdoctoral Researcher at Center for IT-Security, Privacy and Accountability (CISPA)
• Focus on WebSec Research for PhD
• Now also on Systems and Network Security
• Repeat offender at OWASP
• Base for this talk is a paper at CCS 2015
Agenda

- **Client-Side what...?** (Intro & History of Client-Side XSS)
- **But why?** (Motivation and Contribution)
- **How to get a nice data set?** (Bragging about our work)
- **How complex is a flow?** (Sciency stuff)
- **So, highlights?** (Facepalms and Brain Benders + Quiz)
- **How to do it right?** (Best practices)
- **TL;DR?** (Conclusion)
INTRO AND HISTORY OF CLIENT-SIDE CROSS-SITE SCRIPTING

Ben Stock (@kootsneb) – From Facepalm to Brain Bender
Client-Side Cross-Site Scripting

- a.k.a. DOM-based Cross-Site Scripting
- ... caused by insecure JavaScript code

```javascript
document.write("<img src='//adve.rt/ise?hash=" + location.hash.slice(1)+ "'/>");
```

```html
<img src='//adve.rt/ise?hash=HASHVALUE'/>
```

```html
<img src='//adve.rt/ise?hash="/>
<script>alert(1)</script>
```

- Visit http://vuln.com/#"/><script>alert(1)</script>
A Brief History of Client-Side XSS

- 2005: Amit Klein coins the term „DOM-based XSS“
- 2011: Stefano di Paolo first releases DOMinator
  - Uses taint tracking to find data flows
- 2013: Lekies et al. conduct large-scale study
  - Find that more than 10% of Top 5k domains are vulnerable
- 2014: Stock et al. evaluate XSSAuditor and propose new defense using taint tracking
MOTIVATION AND CONTRIBUTION

Ben Stock (@kootsneb) – From Facepalm to Brain Bender
Motivation

• Previous research in this area focused on the **detection** and **mitigation** in the browser
• No analysis of **underlying issues**

• Our focus: analyze **real-world** vulnerabilities
Topics of this talk

• Analyze real-world client-side XSS vulnerabilities
• Answer a numer of questions:
  – Are analysts overwhelmed by the *complexity* of flows?
  – Are developers not aware of the pitfalls?
  – Are there special circumstances in the Web model that cause such flaws?
DATA SET

Ben Stock (@kootsneb) – From Facepalm to Brain Bender
Components

• Taint-Enhanced Browsing Engine
  – mark all user-provided data as "tainted"
  – precise information on source of each character
  – additional information about encoding
  – all relevant sinks report tainted access

• Crawling Extension
  – steers browser to crawl given set of domains
  – collects and transmits flow information
Suspicious Flow = Vulnerability?

• Taint tracking engine reports suspicious flows of data
  – From attacker-controllable source to sink, not encoded using any built-in function (e.g., escape or encodeURI)

```javascript
<script>
  if (/^[a-z][0-9]+$/.test(location.hash.slice(1)) { 
    document.write(location.hash.slice(1));
  }
</script>

• ➔ Not every flow is actually vulnerable
  – Need to verify that flow is exploitable
Infrastructure Overview

![Diagram showing the flow from URLs to Exploit Generator to Crawl Exploit to Exploited! and ReportX to Crawl X to URLs]

Ben Stock (@kootsneb) – From Facepalm to Brain Bender
Resulting Vulnerabilities

• 1,146 vulnerable URLs in Alexa Top 10,000 domains
  – Only slightly lower number vulnerable domains

• 1,273 distinct vulnerabilities
  – i.e., one page, multiple vulnerabilities
Resulting Vulnerabilities

- 1,273 real-world exploits
  - many of them minified
    - Causes issues with metrics
  - many of them not stable (e.g. banner rotation)
- Need to be normalized for a sound analysis
Normalizing the Data Set

1. Cache and beautify HTML, JavaScript
2. Proxy with „fuzzy matching“
3. Analyze pages with taint-aware engine to collect traces
4. Post-process reports (e.g. jQuery detection)
5. Application of Metrics / Additional Analysis
FLOW COMPLEXITY

Ben Stock (@kootsneb) – From Facepalm to Brain Bender
Measuring Complexity of Flows

• Existing approaches measure complexity of code base
  – e.g. McCabe: # of linearly independent paths through program
• Our notion: How hard is for an analyst to decide that a flow is actually vulnerable?
• Find measurable properties of complexity
M₁: Number of operations on tainted data

• Intuition: more operations, more chance to miss something important
M₂: Number of involved functions

• Functionality can be split up into functions
• Intuition: The more functions, the harder it is to follow the data flow
M₃: Number of involved contexts

- JavaScript may resides in several scripts elements
  - Inline scripts
  - Externally included JavaScript files
- Intuition: When you have to switch between inline scripts and external files, you might loose track
M₄: Code locality of source and sink

- Lines of code between source and sink
  - If they even reside within the same context
- Intuition: Data flows within a couple of lines are easier to spot
M₅: Call Stack Relation Source and Sink

- Intuition: Detecting flows is harder when you cannot follow the flows directly

Relative to sink access in SE #3
Relation 1

```html
<script>
var source = location.href;
...
document.write(source);
</script>
```
Relation 5

```
<script>
var global = location.href;
...
</script>
...
<script>
eval(global);
</script>
```
Metric Results

M₁: Operations

M₂: Functions

M₃: Contexts

M₄: Locality

Ben Stock (@kootsneb) – From Facepalm to Brain Bender
Putting the Results into Perspective

- Derive $80^{th}$ and $95^{th}$ percentile for all metrics
  - Either low, medium or high complexity
- Overall score = single highest rating of any classifier
  - Notion: see if metrics correlate or not

<table>
<thead>
<tr>
<th></th>
<th>80th</th>
<th>95th</th>
<th>100th</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_1$</td>
<td>&lt;= 9</td>
<td>&lt;= 22</td>
<td>&gt; 22</td>
</tr>
<tr>
<td>$M_2$</td>
<td>&lt;= 4</td>
<td>&lt;= 10</td>
<td>&gt; 10</td>
</tr>
<tr>
<td>$M_3$</td>
<td>&lt;= 2</td>
<td>3</td>
<td>&gt; 3</td>
</tr>
<tr>
<td>$M_4$</td>
<td>&lt;= 75</td>
<td>&lt;= 394</td>
<td>&gt; 394</td>
</tr>
<tr>
<td>$M_5$</td>
<td>R1, R2</td>
<td>R3, R4</td>
<td>R5</td>
</tr>
</tbody>
</table>

27
## Combined Classification

<table>
<thead>
<tr>
<th></th>
<th>Low Complexity</th>
<th>Medium Complexity</th>
<th>High Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>M₁</td>
<td>1,079</td>
<td>134</td>
<td>60</td>
</tr>
<tr>
<td>M₂</td>
<td>1,161</td>
<td>85</td>
<td>27</td>
</tr>
<tr>
<td>M₃</td>
<td>1,035</td>
<td>178</td>
<td>60</td>
</tr>
<tr>
<td>M₄</td>
<td>920</td>
<td>179</td>
<td>51</td>
</tr>
<tr>
<td>M₅</td>
<td>1,094</td>
<td>120</td>
<td>59</td>
</tr>
<tr>
<td>Combined</td>
<td><strong>813 (63.9%)</strong></td>
<td><strong>261 (20.5%)</strong></td>
<td><strong>199 (15.6%)</strong></td>
</tr>
</tbody>
</table>
Is Complexity the Causing Factor?

<table>
<thead>
<tr>
<th>Vulnerable flows</th>
<th>Randomly sampled flows</th>
</tr>
</thead>
<tbody>
<tr>
<td>M_1</td>
<td>&lt;= 9</td>
</tr>
<tr>
<td>M_2</td>
<td>&lt;= 4</td>
</tr>
<tr>
<td>M_3</td>
<td>&lt;= 2</td>
</tr>
<tr>
<td>M_4</td>
<td>&lt;= 75</td>
</tr>
<tr>
<td>M_5</td>
<td>R1, R2, R3, R4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>80th</th>
<th>95th</th>
<th>100th</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;= 9</td>
<td>&lt;= 4</td>
<td>&lt;= 2</td>
</tr>
<tr>
<td>&lt;= 4</td>
<td>&lt;= 4</td>
<td>&lt;= 2</td>
</tr>
<tr>
<td>&lt;= 2</td>
<td>&lt;= 2</td>
<td>&lt;= 2</td>
</tr>
<tr>
<td>&lt;= 75</td>
<td>&lt;= 75</td>
<td>&lt;= 75</td>
</tr>
<tr>
<td>R1, R2, R3, R4</td>
<td>R5, R4, R5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>80th</th>
<th>95th</th>
<th>100th</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;= 44</td>
<td>&gt; 44</td>
<td></td>
</tr>
<tr>
<td>&lt;= 19</td>
<td>&gt; 19</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>&gt; 3</td>
<td></td>
</tr>
<tr>
<td>&lt;= 1,208</td>
<td>&gt; 1,208</td>
<td></td>
</tr>
</tbody>
</table>

Maybe, but randomly sampled flows are more complex
FACEPALMS AND BRAIN BENDERS

Ben Stock (@kootsneb) – From Facepalm to Brain Bender
Facepalms

- 350 one liners
  - `document.write(location.href);`
- 542 with less than **five** operations
  - Mostly concat of hard-coded + user-controlled data
- Personal favorite: w3schools.com
  - `document.write("Page location is " + location.href);`
Brain Benders

- **59 non-linear control flows** (R5)
  - No means to follow the data flow
  - Sometimes even event-driven
- **31 functions** were passed in the most complex flow
- **up to 291 operations** conducted on tainted data
  - Mostly regexps tests for sub-domains, though
Involving Third-Parties

- **Included third-party** JavaScript code is executed in context of **including** site
  - Vulnerable third-party code → own site vulnerable
  - Code might change, even though URL remains the same
- **273** vulnerabilities caused only by third-party code
- **25 flaws** due to outdated, vulnerable version of jQuery
  - Same version on **472** pages, most did not use the vulnerable API
Non-linear control flow

// inline
var parts = window.location.href.split("#");
if (parts.length > 1) {
  var kw = decodeURIComponent(parts.pop());
  var meta = document.createElement('meta');
  meta.setAttribute('name', 'keywords');
  meta.setAttribute('content', kw);
  document.head.appendChild(meta);
}

// third-party
var kwds = getKwds();
document.write('<iframe src="...&loc=' + kwds + '"></iframe>');
QUIZ TIME

Ben Stock (@kootsneb) – From Facepalm to Brain Bender
Is there something wrong here?

```javascript
function escapeHtml(s) {
    var div = document.createElement('div');
    div.innerHTML = s;
    var scripts = div.getElementsByTagName('script');
    for (var i = 0; i < scripts.length; ++i) {
        scripts[i].parentNode.removeChild(scripts[i]);
    }
    return div.innerHTML;
}
```

There is something wrong here!

```javascript
function escapeHtml(s) {
    var div = document.createElement('div');
    div.innerHTML = s;
    var scripts = div.getElementsByTagName('script');
    for (var i = 0; i < scripts.length; i++) {
        scripts[i].parentNode.removeChild(scripts[i]);
    }
    return div.innerHTML;
}
```

**innerHTML does not execute script elements**

It does, however, allow to create event handlers...
Is there something wrong here?

```javascript
var slotId = parseInt(userdata, 10);
if (slotId) {
    AD_CLB_fillSlot(userdata);
}
```
var slotId = parseInt(userdata, 10);
if (slotId) {
    AD_CLB_fillSlot(userdata);
}

parseInt("1<script>") will not crash, but return 1
Is there something wrong here?

```javascript
jQuery("#warning404 .errorURL").html(
location.href.replace(/</,"&lt;"))
```
There is something wrong here!

```javascript
jQuery("#warning404 .errorURL").html(
location.href.replace(/</, "&lt;"))
```

First parameter is a regular expression, does not have global modifier
Underlying Causes

• Are analysts overwhelmed by the *complexity* of flows?
  – Some flows are quite complex, but randomly sampled flows are more complex on average

• Are developers not aware of the pitfalls?
  – Improper API usage, single line flaws, explicit decoding

• Are there special circumstances in the Web model that cause such flaws?
  – Third-party flaws cause vulnerability in *including* application
BEST PRACTICES

Ben Stock (@kootsneb) – From Facepalm to Brain Bender
Best practices: document.write

// vulnerable
document.write("<base href=' " + location.href "'>"));

// fixed
var base = document.createElement("base");
base.href = location.href;
document.body.appendChild(base);
// or
document.write(base.outerHtml);
Best practices: avoid eval

```javascript
if (url.indexOf('?') >= 0) {
    var qs = url.slice(url.indexOf('?') + 1).split('&');
    for (var i = 0; i < qs.length; i++) {
        var t_p = qs[i].split('=');
        if (t_p.length == 2) {
            eval('data.' + t_p[0] + '="' + t_p[1] + '"');
        }
    }
}
```
Best practices: avoid eval

```javascript
if (url.indexOf('?') >= 0) {
  var qs = url.slice(url.indexOf('?') + 1).split('&#38;');
  for (var i = 0; i < qs.length; i++) {
    var t_p = qs[i].split('&#61;');
    if (t_p.length == 2) {
      data[t_p[0]] = t_p[1];
    }
  }
}
```
Best practices: third parties

- Ask your advertisement provider if they know what DOM-based XSS is ;-) 
- Does your ad really need full access to your main domain?
  - Run it in a frame with a different sub domain to contain damage
Best practices: third parties

• Update your libraries!
  – Use retire.js to find them if necessary
SUMMARY AND CONCLUSION
Summary & Conclusion

- Covered basics and history of Client-Side XSS
- Investigated a data set of 1,273 real-world vulnerabilities
- Several causes: complexity, unawareness, third parties
- Bad examples and best practices