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 (@kcotsneb) – From Facepalm to Brain Bender

Client-Side Cross-Site Scripting

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

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

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

```
<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

Motivation

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

- Our focus: analyze <u>real-world</u> 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

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

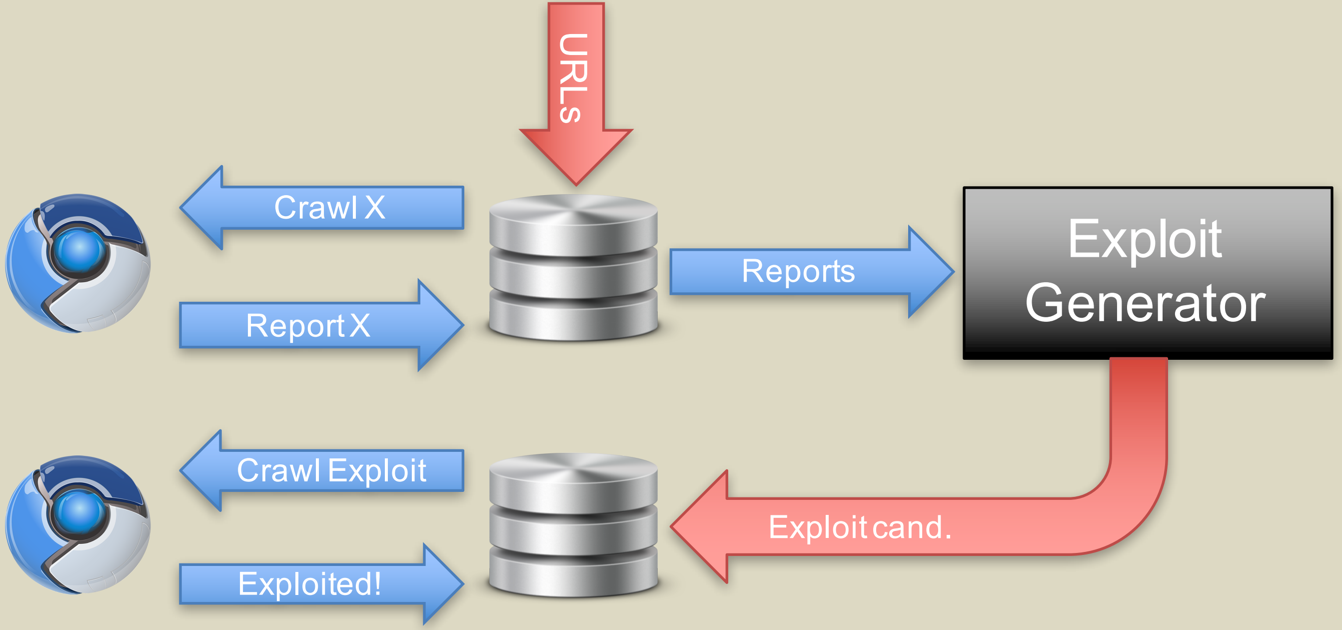
Suspicious Flow = Vulnerability?

- 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)

```
<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



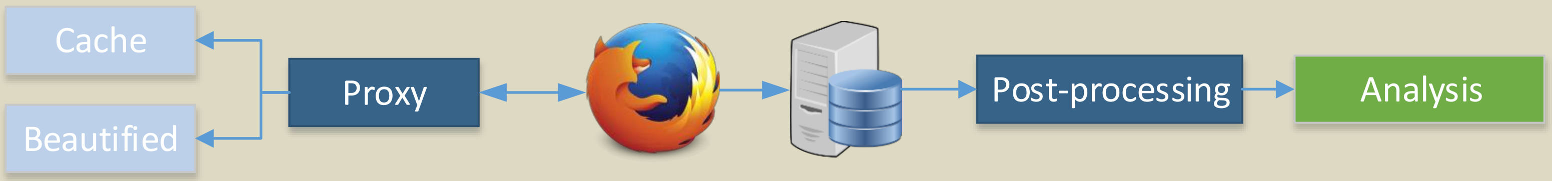
Resulting Vulnerabilities

- 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

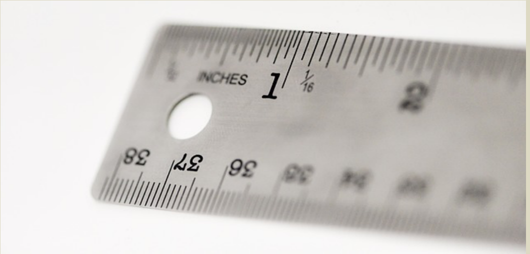
FLOW COMPLEXITY

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

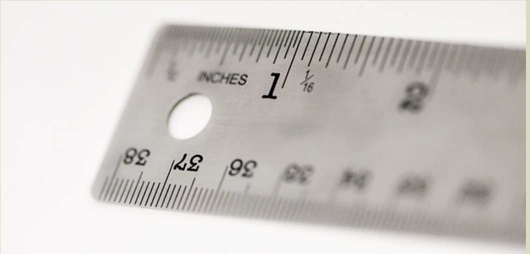
M_1: Number of operations on tainted data

- Intuition: more operations, more chance to miss something important



M_2: Number of involved functions

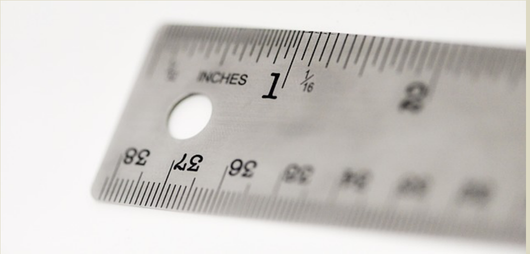
- Functionality can be split up into functions
- Intuition: The more functions, the harder it is to follow the data flow



ROMA MMXVI APPSEC EUROPE

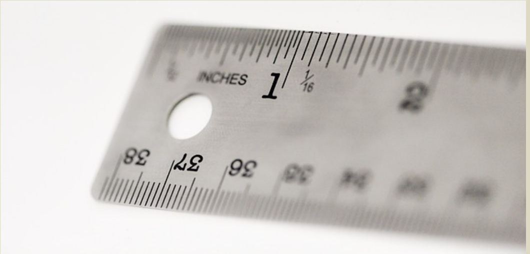
M_3: Number of involved contexts

- 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



M_4: Code locality of source and sink

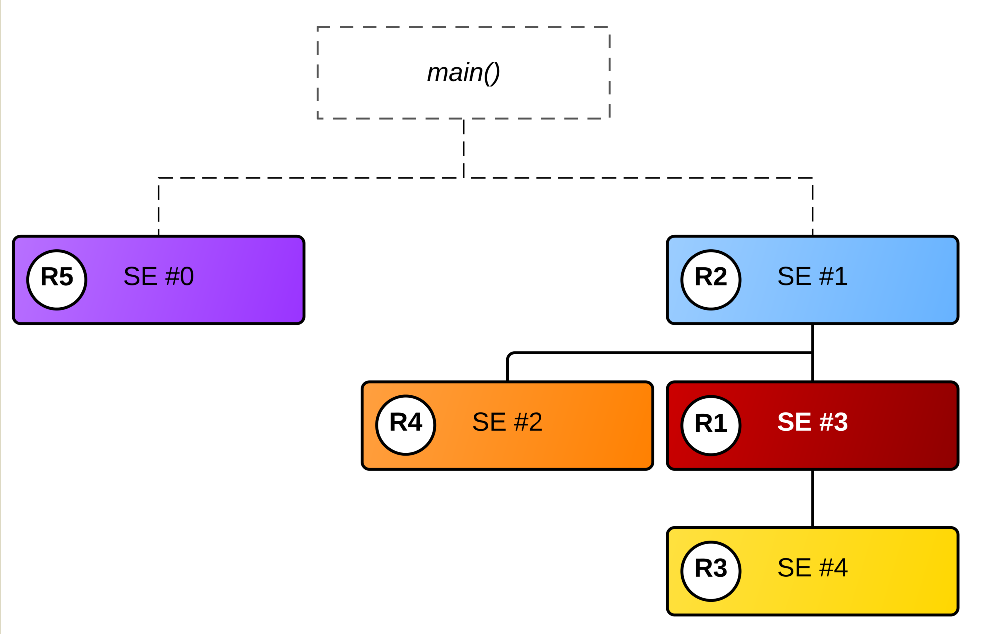
- Lines of code between source and sink
  - If they even reside within the same context
- Intution: Data flows within a couple of lines are easier to spot



APPSEC EUROPE
ROMA MMXVI

M_5: Call Stack Relation Source and Sink

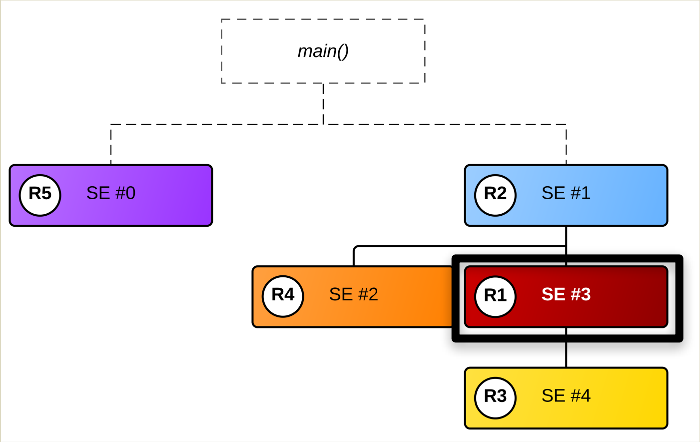
- Intuition: Detecting flows is harder when you cannot follow the flows directly



Relative to sink access in SE #3

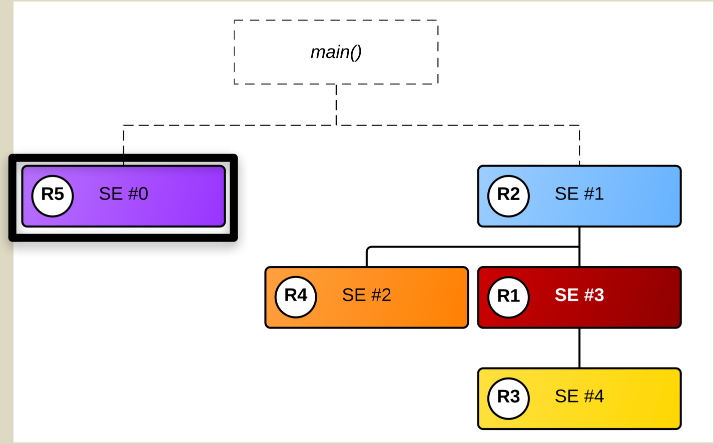
Relation 1

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

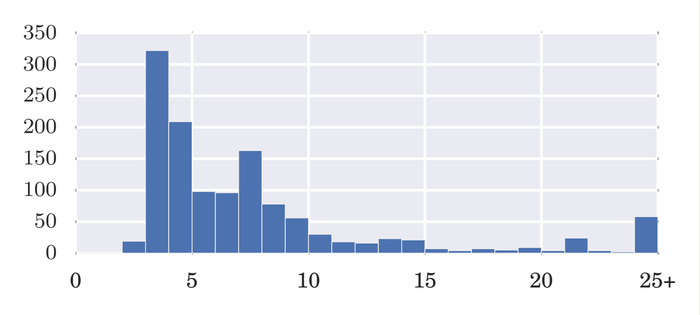


Relation 5

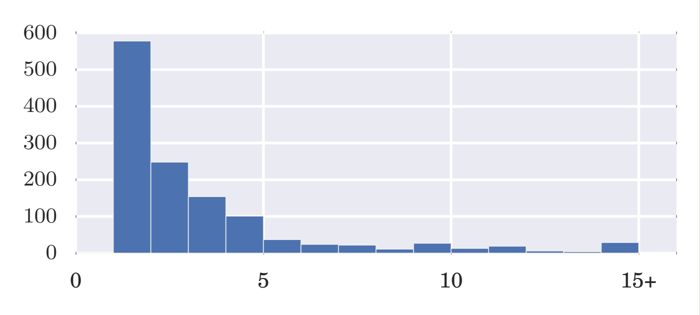
```
<script>
var global = location.href;
...
</script>
...
<script>
eval(global);
</script>
```



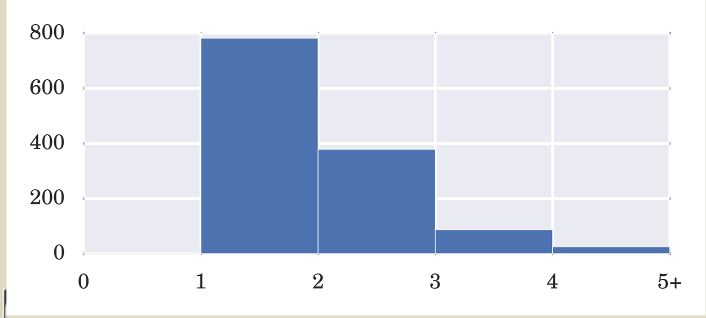
Metric Results



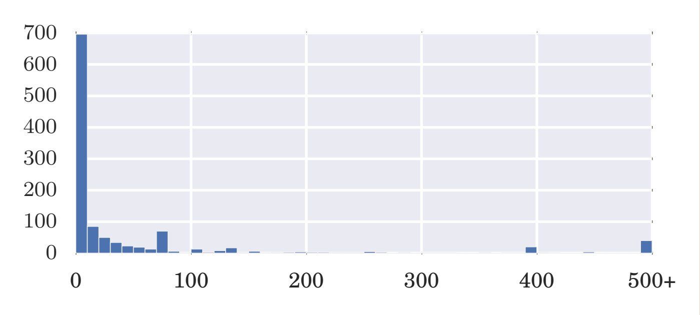
M_1: Operations



M_2: Functions



M_3: Contexts



M_4: Locality

Putting the Results into Perspective

- Derive 80th and 95th 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

	80th	95th	100th
M_1	<= 9	<= 22	> 22
M_2	<= 4	<= 10	> 10
M_3	<= 2	3	> 3
M_4	<= 75	<= 394	> 394
M_5	R1, R2	R3, R4	R5

Combined Classification

	Low Complexity	Medium Complexity	High Complexity
M_1	1,079	134	60
M_2	1,161	85	27
M_3	1,035	178	60
M_4	*920*	*179*	*51*
M_5	1,094	120	59
Combined	**813 (63.9%)**	**261 (20.5%)**	**199 (15.6%)**

Is Complexity the Causing Factor?

	80th	95th	100th
M_1	<= 9	[illegible]	[illegible]
M_2	<= 4	[illegible]	[illegible]
M_3	<= 2	3	[illegible]
M_4	<= 75	[illegible]	[illegible]
M_5	R1, R2	R3, R4	R5

Vulnerable flows

	80th	95th	100th
[illegible]	[illegible]	<= 44	> 44
[illegible]	[illegible]	<= 19	> 19
[illegible]	[illegible]	3	> 3
[illegible]	[illegible]	<= 1,208	> 1,208
M_5	R1, R2, R3	R4	R5

Randomly sampled flows

Maybe, but randomly sampled flows are more complex

ROMA MMXVI

APPSEC EUROPE

FACEPALMS AND BRAIN BENDERS

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

Is there something wrong here?

```
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!

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

innerHTML <u>does not execute</u> script elements

It does, however, allow to create event handlers...

Is there something wrong here?

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

There is something wrong here!

```
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?

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

There is something wrong here!

```
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 <u>including</u> application

BEST PRACTICES

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

```
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

```
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) {
    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

Erlend Oftedal @webtonull · 37 Min.
Fortune 500:
Feb 2014: 77%
Feb 2015: 68%
Oct 2015: 72%
Jun 2016: 83%
@RetireJS

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

Thank you!