25 Million Flows Later – Large-scale Detection of DOM-based XSS

CCS 2013, Berlin
Sebastian Lekies, Ben Stock, Martin Johns
Agenda

- XSS & Attacker Scenario
  - WebSec guys: wake up once you see a cat
- Motivation
- Our contributions
- Summary
Cross-Site Scripting

- Execution of attacker-controlled code on the client in the context of the vulnerable app

- Three kinds:
  - Persistent XSS: guestbook, ...
  - Reflected XSS: search forms, ...
  - DOM-based XSS: also called local XSS
    - content dynamically added by JS (e.g. like button), ...

Server side

Client side
Cross-Site Scripting: attacker model

- Attacker wants to inject own code into vuln. app
  - steal cookie
  - take arbitrary action in the name of the user
  - pretend to be the server towards the user
  - ...

Source: http://blogs.sfweekly.com/thesnitch/cookie_monster.jpg
Cross-Site Scripting: problem statement

- **Main problem:** attacker’s content ends in document and is not properly filtered/encoded
  - common for server- and client-side flaws
- *Flow of data:* from attacker-controllable *source* to security-sensitive *sink*
- **Our Focus:** client side JavaScript code
  - **Sources:** e.g. the URL
  - **Sinks:** e.g. document.write
Example of a DOMXSS vulnerability

- Source: `location.hash`, Sink: `document.write`

- Intended usage:
  - `http://example.org/#mypage`
  - `<img src='//adve.rt/ise?hash=mypage'/>`

- Exploiting the vuln:
  - `http://example.org/#/<script>alert(1)</script>`
  - `<img src='//adve.rt/ise?hash='/>
    <script>alert(1)</script>
    '>'
How does the attacker exploit this?

a. Send a crafted link to the victim

b. Embed vulnerable page with payload into his own page

http://kittenpics.org

Source: http://www.hd-gbpics.de/gbbilder/katzen/katzen2.jpg
Our motivation and contribution

- Perform Large-scale analysis of DOMXSS vulnerabilities
  - Automated, dynamic detection of suspicious flows
  - Automated validation of vulnerabilities

- Our key components
  - Taint-aware browsing engine
  - Crawling infrastructure
  - Context-specific exploit generator
  - Exploit verification using the crawler
Building a taint-aware browsing engine to find suspicious flows
Our approach: use dynamic taint tracking

- **Taint tracking**: Track the flow of *marked* data from source to sink
- **Implementation**: into Chromium (Blink+V8)

**Requirements for taint tracking**

- Taint all relevant values / propagate taints
- Report all sinks accesses
- **be as precise as possible**
  - taint details on EVERY character
Representing sources

- In terms of DOMXSS, we have 14 sources
- additionally, **three** relevant, built-in encoding functions
  - escape, encodeURI and encodeURIComponent
  - .. may prevent XSS vulnerabilities if **used properly**
- Goal: store **source + bitmask of encoding functions for each character**
Representing sources (cntd)

- 14 sources ⇒ 4 bits sufficient
- 3 relevant built-in functions ⇒ 3 bits sufficient
- 7 bits < 1 byte

⇒ 1 Byte sufficient to store source + encoding functions
- encoding functions and counterparts set/unset bits
- hard-coded characters have source 0
Representing sources (cntd)

- Each source API (e.g. URL or cookie) attaches taint bytes
  - identifying the source of a char
  - `var x = location.hash.slice(1);`
  - `x = escape(x);`
Detecting sink access

● Taint propagated through all relevant functions
● Security-sensitive sinks report flow and details
  ● such as text, taint information, source code location

● Chrome extension to handle reporting
  ● keep core changes as small as possible
  ● repack information in JavaScript
  ● stub function directly inside V8
Empirical study on suspicious flows
Crawling the Web (at University scale)

- Crawler infrastructure consisting of:
  - modified, taint-aware browsing engine
  - browser extension to direct the engine
  - Dispatching and reporting backend
- In total, we ran 6 machines
Empirical study

● Shallow crawl of Alexa Top 5000 Web Sites
  ● Main page + first level of links
  ● 504,275 URLs scanned in roughly 5 days
    ● on average containing ~8,64 frames
  ● total of 4,358,031 analyzed documents

● Step 1: Flow detection
  ● 24,474,306 data flows from possibly attacker-controllable input to security-sensitive sinks
Context-Sensitive Generation of Cross-Site Scripting Payloads
Validating vulnerabilities

● Current Situation:
  ● Taint-tracking engine delivers suspicious flows
  ● Suspicious flow != Vulnerability

● Why may suspicious flows not be exploitable?
  ● e.g. custom filter, validation or encoding function

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

● Validation needed: working exploit
Anatomy of an XSS Exploit

- Cross-Site Scripting exploits are context-specific:
  - HTML Context
    - Vulnerability: `document.write("<img src='pic.jpg?hash=" + location.hash.slice(1) + "'>");`
    - Exploit: `<script>alert(1)</script><textarea>`
  - JavaScript Context
    - Vulnerability: `eval("var x = " + location.hash + "");`
    - Exploit: `'; alert(1); //`
Anatomy of an XSS Exploit

- **Context-Sensitivity**
  - Breakout-Sequence: Highly context sensitive (generation is difficult)
  - Payload: Not context sensitive (arbitrary JavaScript code)
  - Comment Sequence: Very easy to generate (choose from a handful of options)
Breaking out of JavaScript contexts

- JavaScript Context

```javascript
<script>
  var code = 'function test(){
    var x = "' + location.href + '";
    //inside function test
    + 'doSomething(x);'
    + '}'
  //top level
  eval(code);
</script>
```

- Visiting http://example.org/ in our engine

```javascript
eval('function test() {
  var x = "http://example.org";
  doSomething(x);
}
');
```
Syntax tree to working exploit

function test() {
    var x = "http://example.org";
    doSomething(x);
}

- Two options here:
  - break out of string
  - break out of function definition

- Latter is more reliable
  - function test not necessarily called automatically on „normal“ execution

Tainted value aka injection point
Generating a valid exploit

- Traverse the AST upwards and “end” the branches
  - Breakout Sequence: “};
  - Comment:  //
  - Exploit: "};alert(1);//
  - Visit: http://example.org/#};alert(1);//

```javascript
function test() {
  var x = "http://example.org";
  alert(1); //
  doSomething(x); }
```
Validating vulnerabilities

● Our focus: directly controllable exploits
  ● *Sinks*: direct execution sinks
    ○ HTML sinks (document.write, innerHTML, ...)
    ○ JavaScript sinks (eval, ...)
  ● *Sources*: location and referrer
  ● Only unencoded strings

● Not in the focus (yet): second-order vulnerabilities
  ● to cookie and from cookie to eval
  ● ...
Empirical study

● Step 2: Flow reduction
  ● Only JavaScript and HTML sinks: 24,474,306 → 4,948,264
  ● Only directly controllable sources: 4,948,264 → 1,825,598
  ● Only unencoded flows: 1,825,598 → 313,794

● Step 3: Precise exploit generation
  ● Generated a total of 181,238 unique test cases
  ● rest were duplicates (same URL and payload)
    ● basically same vuln twice in same page
Empirical study

• Step 4: Exploit validation
  • 69,987 out of 181,238 unique test cases triggered a vulnerability

• Step 5: Further analysis
  • 8,163 unique vulnerabilities affecting 701 domains
  • …of all loaded frames (i.e. also from outside Top 5000)
  • 6,167 unique vulnerabilities affecting 480 Alexa top 5000 domains
  • At least, 9.6 % of the top 5000 Web pages contain one or more XSS problems
  • This number only represents the lower bound (!)
Limitations

- No assured code coverage
  - e.g. debug GET-param needed?
  - also, not all pages visited (esp. stateful applications)
- Fuzzing might get better results
  - does not scale as well
- Not yet looking at the „harder“ flows
  - found one URL ➔ Cookie ➔ eval „by accident“
Summary

- We built a tool capable of **detecting** flows
  - taint-aware Chromium
  - Chrome extension for crawling and reporting

- We built an **automated exploit generator**
  - taking into account the **exact taint information**
  - ... and specific contexts

- We found that at least **480** of the top **5000** domains carry a DOM-XSS vuln
Thank you very much for your attention!

Ben Stock
@kcotsneb
ben.stock@fau.de
## Outlook on future work

### Sources

<table>
<thead>
<tr>
<th></th>
<th>URL</th>
<th>Cookie</th>
<th>Referrer</th>
<th>window.name</th>
<th>postMessage</th>
<th>WebStorage</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTML</td>
<td>1,356,796</td>
<td>1,535,299</td>
<td>240,341</td>
<td>35,446</td>
<td>35,103</td>
<td>16,387</td>
<td>3,219,392</td>
</tr>
<tr>
<td>JavaScript</td>
<td>22,962</td>
<td>359,962</td>
<td>511</td>
<td>617,743</td>
<td>448,311</td>
<td>279,383</td>
<td>1,728,872</td>
</tr>
<tr>
<td>URL</td>
<td>3,798,228</td>
<td>2,556,709</td>
<td>313,617</td>
<td>83,218</td>
<td>18,919</td>
<td>28,052</td>
<td>6,798,743</td>
</tr>
<tr>
<td>Cookie</td>
<td>220,300</td>
<td>10,227,050</td>
<td>25,062</td>
<td>1,328,634</td>
<td>2,554</td>
<td>5,618</td>
<td>11,809,218</td>
</tr>
<tr>
<td>post Message</td>
<td>451,170</td>
<td>77,202</td>
<td>696</td>
<td>45,220</td>
<td>11,053</td>
<td>117,575</td>
<td>702,916</td>
</tr>
<tr>
<td>Web Storage</td>
<td>41,739</td>
<td>65,772</td>
<td>1,586</td>
<td>434</td>
<td>194</td>
<td>105,440</td>
<td>215,165</td>
</tr>
<tr>
<td>Total</td>
<td>5,891,195</td>
<td>14,821,994</td>
<td>581,813</td>
<td>2,110,715</td>
<td>516,134</td>
<td>552,455</td>
<td>24,474,306</td>
</tr>
<tr>
<td>Encoded</td>
<td>64,78%</td>
<td>52,81%</td>
<td>83,99%</td>
<td>57,69%</td>
<td>1,57%</td>
<td>30,31%</td>
<td></td>
</tr>
</tbody>
</table>

### Outlets

Outlook on future work
## Outlook on future work

### Sources

<table>
<thead>
<tr>
<th></th>
<th>URL</th>
<th>Cookie</th>
<th>Referrer</th>
<th>window.name</th>
<th>postMessage</th>
<th>WebStorage</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTML</td>
<td>1,356,796</td>
<td>1,535,299</td>
<td>240,341</td>
<td>35,446</td>
<td>35,103</td>
<td>16,387</td>
<td>3,219,392</td>
</tr>
<tr>
<td>JavaScript</td>
<td>22,962</td>
<td>359,962</td>
<td>511</td>
<td>617,743</td>
<td>448,311</td>
<td>279,383</td>
<td>1,728,872</td>
</tr>
<tr>
<td>URL</td>
<td>3,798,228</td>
<td>2,556,709</td>
<td>313,617</td>
<td>83,218</td>
<td>18,919</td>
<td>28,052</td>
<td>6,798,743</td>
</tr>
<tr>
<td>Cookie</td>
<td>220,300</td>
<td>10,227,050</td>
<td>25,062</td>
<td>1,328,634</td>
<td>2,554</td>
<td>5,618</td>
<td>11,809,218</td>
</tr>
<tr>
<td>post Message</td>
<td>451,170</td>
<td>77,202</td>
<td>696</td>
<td>45,220</td>
<td>11,053</td>
<td>117,575</td>
<td>702,916</td>
</tr>
<tr>
<td>Web Storage</td>
<td>41,739</td>
<td>65,772</td>
<td>1,586</td>
<td>434</td>
<td>194</td>
<td>105,440</td>
<td>215,165</td>
</tr>
<tr>
<td>Total</td>
<td>5,891,195</td>
<td>14,821,994</td>
<td>581,813</td>
<td>2,110,715</td>
<td>516,134</td>
<td>552,455</td>
<td>24,474,306</td>
</tr>
<tr>
<td>Encoded</td>
<td>64,78%</td>
<td>52,81%</td>
<td>83,99%</td>
<td>57,69%</td>
<td><strong>1,57%</strong></td>
<td>30,31%</td>
<td></td>
</tr>
</tbody>
</table>

### Sinks

Outlook on future work

Sources

<table>
<thead>
<tr>
<th></th>
<th>Encoding</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>URL</td>
<td>64,78%</td>
<td>30,31%</td>
</tr>
<tr>
<td>Cookie</td>
<td>97,81%</td>
<td>30,31%</td>
</tr>
<tr>
<td>post Message</td>
<td>83,99%</td>
<td>30,31%</td>
</tr>
<tr>
<td>Web Storage</td>
<td>95,01%</td>
<td>30,31%</td>
</tr>
</tbody>
</table>

Total: 24,474,306

Encoded: 30,31%
<table>
<thead>
<tr>
<th>Sources</th>
<th>URL</th>
<th>Cookie</th>
<th>Referrer</th>
<th>window.name</th>
<th>postMessage</th>
<th>WebStorage</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTML</td>
<td>1,356,796</td>
<td>1,535,299</td>
<td>240,341</td>
<td>35,446</td>
<td>35,103</td>
<td>16,387</td>
<td>3,219,392</td>
</tr>
<tr>
<td>JavaScript</td>
<td>22,962</td>
<td>359,962</td>
<td>511</td>
<td>617,743</td>
<td>448,311</td>
<td>279,383</td>
<td>1,728,872</td>
</tr>
<tr>
<td>URL</td>
<td>3,798,228</td>
<td>2,556,709</td>
<td>313,617</td>
<td>83,218</td>
<td>18,919</td>
<td>28,052</td>
<td>6,798,743</td>
</tr>
<tr>
<td>Cookie</td>
<td>220,300</td>
<td>10,227,050</td>
<td>25,062</td>
<td>1,328,634</td>
<td>2,554</td>
<td>5,618</td>
<td>11,809,218</td>
</tr>
<tr>
<td>post Message</td>
<td>451,170</td>
<td>77,202</td>
<td>696</td>
<td>45,220</td>
<td>11,053</td>
<td>117,575</td>
<td>702,916</td>
</tr>
<tr>
<td>Web Storage</td>
<td>41,739</td>
<td>65,772</td>
<td>1,586</td>
<td>434</td>
<td>194</td>
<td>105,440</td>
<td>215,165</td>
</tr>
<tr>
<td>Total</td>
<td>5,891,195</td>
<td>14,821,994</td>
<td>581,813</td>
<td>2,110,715</td>
<td>516,134</td>
<td>552,455</td>
<td>24,474,306</td>
</tr>
<tr>
<td>Encoded</td>
<td>64,78%</td>
<td>52,81%</td>
<td>83,99%</td>
<td>57,69%</td>
<td>1,57%</td>
<td>30,31%</td>
<td></td>
</tr>
</tbody>
</table>

Outlook on future work
### Outlook on future work

#### Sources

<table>
<thead>
<tr>
<th></th>
<th>URL</th>
<th>Cookie</th>
<th>Referrer</th>
<th>window.name</th>
<th>postMessage</th>
<th>WebStorage</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTML</td>
<td>1,356,796</td>
<td>1,535,299</td>
<td>240,341</td>
<td>35,446</td>
<td>35,103</td>
<td>16,387</td>
<td>3,219,392</td>
</tr>
<tr>
<td>JavaScript</td>
<td>22,962</td>
<td>359,962</td>
<td>511</td>
<td>617,743</td>
<td>448,311</td>
<td>279,383</td>
<td>1,728,872</td>
</tr>
<tr>
<td>URL</td>
<td>3,798,228</td>
<td>2,556,709</td>
<td>313,617</td>
<td>83,218</td>
<td>18,919</td>
<td>28,052</td>
<td>6,798,743</td>
</tr>
<tr>
<td>Cookie</td>
<td>220,300</td>
<td>10,227,050</td>
<td>25,062</td>
<td>1,328,634</td>
<td>2,554</td>
<td>5,618</td>
<td>11,809,218</td>
</tr>
<tr>
<td>post Message</td>
<td>451,170</td>
<td>77,202</td>
<td>696</td>
<td>45,220</td>
<td>11,053</td>
<td>117,575</td>
<td>702,916</td>
</tr>
<tr>
<td>Web Storage</td>
<td>41,739</td>
<td>65,772</td>
<td>1,586</td>
<td>434</td>
<td>194</td>
<td>105,440</td>
<td>215,165</td>
</tr>
<tr>
<td>Total</td>
<td>5,891,195</td>
<td>14,821,994</td>
<td>581,813</td>
<td>2,110,715</td>
<td>516,134</td>
<td>552,455</td>
<td>24,474,306</td>
</tr>
<tr>
<td>Encoded</td>
<td>64,78%</td>
<td>52,81%</td>
<td>83,99%</td>
<td>57,69%</td>
<td>1,57%</td>
<td>30,31%</td>
<td></td>
</tr>
</tbody>
</table>

#### Sinks

<table>
<thead>
<tr>
<th></th>
<th>URL</th>
<th>Cookie</th>
<th>Referrer</th>
<th>window.name</th>
<th>postMessage</th>
<th>WebStorage</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outlook on future work</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>