Protection, Usability and Improvements in Reflected XSS Filters

Riccardo Pelizzi

System Security Lab
Department of Computer Science
Stony Brook University

May 2, 2012
Outline

• XSS Attacks
Outline

- XSS Attacks
- Existing XSS Filters
Outline

• XSS Attacks
• Existing XSS Filters
• XSSFilt
Outline

- XSS Attacks
- Existing XSS Filters
- XSSFilt
- Results
Outline

• XSS Attacks
• Existing XSS Filters
• XSSFilt
• Results
• Conclusions and Q&A
Same-Origin Policy

User
Please visit evil.com

Malicious Server

Attacker

Browser
payload = send bank.com cookies to evil.com

payload = send bank.com cookies to evil.com

User
Click on link

Malicious Server

Payload

Send response + payload

Malicious Server

E x ecu te payload
on evil.com

S O P V iolation

payload = send bank.com cookies to evil.com
Same-Origin Policy

Attacker

Please visit evil.com

User

Click on link

Browser

Malicious Server

payload = send bank.com cookies to evil.com
Same-Origin Policy

Attacker

Please visit evil.com

User

Browser

Malicious Server

Click on link

GET evil.com

payload = send bank.com cookies to evil.com
Same-Origin Policy

Attacker

User

Browser

Malicious Server

Please visit evil.com

Click on link

GET evil.com

Send response + payload

Execute payload on evil.com

payload = send bank.com cookies to evil.com
Same-Origin Policy

Attacker

Please visit evil.com

User

Click on link

Browser

GET evil.com

Malicious Server

Send response + payload

Execute payload on evil.com

payload = send bank.com cookies to evil.com

3 / 19 Riccardo Pelizzi

Improvements in Reflected XSS Filters
Same-Origin Policy

Attacker

User

Browser

Malicious Server

Please visit evil.com

Click on link

GET evil.com

Send response + payload

Execute payload on evil.com

SOP Violation

payload = send bank.com cookies to evil.com
XSS Attacks

Attacker

User/Browser

bank.com

payload = send bank.com cookies to Attacker

Adds payload to bank.com pages

payload = send bank.com cookies to Attacker

V isits bank.com

Response + payload

Cookies

Riccardo Pelizzi

Improvements in Reflected XSS Filters
XSS Attacks

Attacker

User/Browser

bank.com

payload = send bank.com cookies to Attacker

Adds payload to bank.com pages

Visits bank.com

payload = send bank.com cookies to Attacker

Cookies
XSS Attacks

Attacker

User/Browser

bank.com

payload = send bank.com cookies to Attacker

Visit bank.com

Response + payload

payload = send bank.com cookies to Attacker
XSS Attacks

Attacker

User/Browser

bank.com

Payload: send bank.com cookies to Attacker

Visits bank.com

Response + payload

Execute payload on bank.com

payload = send bank.com cookies to Attacker
XSS Attacks

Attacker

User/Browser

bank.com

Adding payload to bank.com pages

Visits bank.com

Response + payload

Cookies

Payload = send bank.com cookies to Attacker

Execute payload on bank.com
Reflected XSS Attacks

Attacker

User

Please visit
bank.com/index.php?branch=payload

Browser

bank.com

payload = send bank.com cookies to Attacker

payload = send bank.com cookies to Attacker

User

Please visit
bank.com/index.php?branch=payload

payload = send bank.com cookies to Attacker

payload = send bank.com cookies to Attacker

Response

Response

Executes payload on bank.com

payl oad = send bank.com cookies to Attacker
Reflected XSS Attacks

Attacker

Please visit
bank.com/index.php?branch=payload

User

Clicks on link

Browser

payload = send bank.com cookies to Attacker

payload = send bank.com cookies to Attacker

bank.com

payload = send bank.com cookies to Attacker

payload = send bank.com cookies to Attacker
Reflected XSS Attacks

Payload = send bank.com cookies to Attacker
Reflected XSS Attacks

payload = send bank.com cookies to Attacker
Reflected XSS Attacks

User
Please visit bank.com/index.php?branch=payload

bank.com

Attacker

Payload = send bank.com cookies to Attacker
Reflected XSS Attacks

Please visit
bank.com/index.php?branch=payload

Clicks on link
GET bank.com+payload
Response+payload

Cookies

Execute payload
on bank.com

payload = send bank.com cookies to Attacker
Reflected XSS Example

- Whole Script Injection

- Partial Script Injection
Reflected XSS Example

- **Whole Script Injection**
  - Server: `<b>Your search query: <?=$GET['query']?></b>`
  - Request: `bank.com/search.php?query=<script>alert(1)</script>`

- **Partial Script Injection**

Improvements in Reflected XSS Filters
Reflected XSS Example

- **Whole Script Injection**
  - Server: `<b>Your search query: <?=$_GET["query"]?></b>`
  - Request:
    ```
    bank.com/search.php?query=<script>alert(1)</script>
    ```

- **Partial Script Injection**
  - Request:
    ```
    bank.com/search.php?id=');do_xss();document.write('```
Blocking Reflected XSS Attacks

• Recognize malicious requests.

• Recognize malicious scripts.
Blocking Reflected XSS Attacks

• Recognize malicious requests.
  • Recognize suspicious requests.
  • Modify suspicious requests.

• Recognize malicious scripts.
Blocking Reflected XSS Attacks

- Recognize malicious requests.
  - Recognize suspicious requests.
  - Modify suspicious requests.
- Recognize malicious scripts.
  - Recognize injected scripts.
  - Prevent injected scripts from executing.
NoScript

- Uses complex regular expressions to find JavaScript in outgoing requests.
NoScript

- Uses complex regular expressions to find JavaScript in outgoing requests.
- Obscure logic for filtering.
NoScript

- Uses complex regular expressions to find JavaScript in outgoing requests.
- Obscure logic for filtering.
- False positives.
NoScript

- Uses complex regular expressions to find JavaScript in outgoing requests.
- Obscure logic for filtering.
- False positives.
- Sanitized requests are often invalid.
IE8

• Using regular expressions, find parameters in response scripts. Modifies HTML responses before they are parsed.
IE8

- Using regular expressions, find parameters in response scripts. Modifies HTML responses before they are parsed.
- Hard to identify scripts in the page.
IE8

- Using regular expressions, find parameters in response scripts. Modifies HTML responses before they are parsed.
- Hard to identify scripts in the page.
- Hard to sanitize scripts.
XSSAuditor

- Examines scripts as they are fed to the JavaScript engine. Looks for the script in the URL.
XSS Auditor

- Examines scripts as they are fed to the JavaScript engine. Looks for the script in the URL.
- Policies do not protect against partial script injections.
XSSAuditor

- Examines scripts as they are fed to the JavaScript engine. Looks for the script in the URL.
- Policies do not protect against partial script injections.
- Can only cope with known, simple string transformations.
Key Benefits of our Approach

- Protection against partial script injections, to increase attack vector coverage.
Key Benefits of our Approach

- Protection against partial script injections, to increase attack vector coverage.
- Taint Inference, to account for arbitrary string transformations.
Key Benefits of our Approach

- Protection against partial script injections, to increase attack vector coverage.
- Taint Inference, to account for arbitrary string transformations.
- Policies that minimize false positives.
Taint Inference

- Taint-Inference is an approximate substring matching algorithm with acceptable overhead.
- Quadratic in the worst case, but mostly linear.
Overview
Policies

• Inline Policy for inlined code (<script>...</script>, data URLs, etc)

• External Policy for external resources (<script src=...>, objects, etc):
Policies

• Inline Policy for inlined code (<script>...</script>, data URLs, etc)
  • Whole Script Injection: a substring of a parameter is found at the beginning of a script.
  • Partial Injection: a parameter is found in a substring of a script, breaking out of a JavaScript string.

• External Policy for external resources (<script src=...>, objects, etc):
Policies

- **Inline Policy for inlined code** (\(<script>...</script>\), data URLs, etc):
  - Whole Script Injection: a substring of a parameter is found at the beginning of a script.
  - Partial Injection: a parameter is found in a substring of a script, breaking out of a JavaScript string.

- **External Policy for external resources** (\(<script src=...>\), objects, etc):
  - Permit same-origin resources and maintain a cache of valid origins to check each origin only once.
Policies

- Inline Policy for inlined code (<script>...</script>, data URLs, etc)
  - Whole Script Injection: a substring of a parameter is found at the beginning of a script.
  - Partial Injection: a parameter is found in a substring of a script, breaking out of a JavaScript string.

- External Policy for external resources (<script src=...>, objects, etc):
  - Permit same-origin resources and maintain a cache of valid origins to check each origin only once.
  - Partial Injection: only flag as attack if the attacker controls the domain.
Protection

<table>
<thead>
<tr>
<th>Dataset</th>
<th>XSSFilt</th>
<th>XSSAuditor</th>
<th>NoScript</th>
</tr>
</thead>
<tbody>
<tr>
<td>xssed</td>
<td>399/400</td>
<td>379/400</td>
<td>400/400</td>
</tr>
<tr>
<td>cheatsheet</td>
<td>20/20</td>
<td>18/20</td>
<td>20/20</td>
</tr>
</tbody>
</table>

Figure: Results for xssed and cheatsheet dataset

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Partial Script Injection</th>
<th>String Transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td>xssed</td>
<td>16</td>
<td>5</td>
</tr>
<tr>
<td>cheatsheet</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure: XSSAuditor failures
Protection

<table>
<thead>
<tr>
<th>Dataset</th>
<th>XSSFilt</th>
<th>XSSAuditor</th>
<th>NoScript</th>
</tr>
</thead>
<tbody>
<tr>
<td>xssed</td>
<td>399/400</td>
<td>379/400</td>
<td>400/400</td>
</tr>
<tr>
<td>cheatsheet</td>
<td>20/20</td>
<td>18/20</td>
<td>20/20</td>
</tr>
</tbody>
</table>

Figure: Results for xssed and cheatsheet dataset

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Partial Script Injection</th>
<th>String Transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td>xssed</td>
<td>16</td>
<td>5</td>
</tr>
<tr>
<td>cheatsheet</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure: XSSAuditor failures

- 8% of web pages from the xssed website are vulnerable to partial injection.
Protection

<table>
<thead>
<tr>
<th>Dataset</th>
<th>XSSFilt</th>
<th>XSSAuditor</th>
<th>NoScript</th>
</tr>
</thead>
<tbody>
<tr>
<td>xssed</td>
<td>399/400</td>
<td>379/400</td>
<td>400/400</td>
</tr>
<tr>
<td>cheatsheet</td>
<td>20/20</td>
<td>18/20</td>
<td>20/20</td>
</tr>
</tbody>
</table>

Figure: Results for xssed and cheatsheet dataset

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Partial Script Injection</th>
<th>String Transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td>xssed</td>
<td>16</td>
<td>5</td>
</tr>
<tr>
<td>cheatsheet</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure: XSSAuditor failures

- 8% of web pages from the xssed website are vulnerable to partial injection.
- 18% for a large-scale survey we performed.
Protection

<table>
<thead>
<tr>
<th>Dataset</th>
<th>XSSFilt</th>
<th>XSSAuditor</th>
<th>NoScript</th>
</tr>
</thead>
<tbody>
<tr>
<td>xssed</td>
<td>399/400</td>
<td>379/400</td>
<td>400/400</td>
</tr>
<tr>
<td>cheatsheet</td>
<td>20/20</td>
<td>18/20</td>
<td>20/20</td>
</tr>
</tbody>
</table>

Figure: Results for xssed and cheatsheet dataset

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Partial Script Injection</th>
<th>String Transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td>xssed</td>
<td>16</td>
<td>5</td>
</tr>
<tr>
<td>cheatsheet</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure: XSSAuditor failures

- 8% of web pages from the xssed website are vulnerable to partial injection.
- 18% for a large-scale survey we performed.
- 9% of pages from the 1000 most popular websites have dynamically generated scripts.
Compatibility

- Crawled 35000+ pages

<table>
<thead>
<tr>
<th>Filter</th>
<th>XSSFilt</th>
<th>XSSAuditor</th>
<th>NoScript</th>
</tr>
</thead>
<tbody>
<tr>
<td># of violations</td>
<td>8</td>
<td>6</td>
<td>15</td>
</tr>
</tbody>
</table>

Figure: Compatibility Comparison
Compatibility

• Crawled 35000+ pages

<table>
<thead>
<tr>
<th>Filter</th>
<th>XSSFilt</th>
<th>XSSAuditor</th>
<th>NoScript</th>
</tr>
</thead>
<tbody>
<tr>
<td># of violations</td>
<td>8</td>
<td>6</td>
<td>15</td>
</tr>
</tbody>
</table>

Figure: Compatibility Comparison

• Most requests did not contain special characters
Compatibility

- Crawled 35000+ pages

<table>
<thead>
<tr>
<th>Filter</th>
<th>XSSFilt</th>
<th>XSSAuditor</th>
<th>NoScript</th>
</tr>
</thead>
<tbody>
<tr>
<td># of violations</td>
<td>8</td>
<td>6</td>
<td>15</td>
</tr>
</tbody>
</table>

Figure: Compatibility Comparison

- Most requests did not contain special characters
- Most NoScript violations are puzzling.
  
  http://domain.com/dir/page.php?
  n=PHNjcmlwdP25p1Jmo9MCI%2BPC9zY3JpcHQ
  %2B h=55e1652a183
Compatibility

- Crawled 35000+ pages

<table>
<thead>
<tr>
<th>Filter</th>
<th>XSSFilt</th>
<th>XSSAuditor</th>
<th>NoScript</th>
</tr>
</thead>
<tbody>
<tr>
<td># of violations</td>
<td>8</td>
<td>6</td>
<td>15</td>
</tr>
</tbody>
</table>

Figure: Compatibility Comparison

- Most requests did not contain special characters
- Most NoScript violations are puzzling.
  - http://domain.com/dir/page.php?
    n=PHNjcmlwdP25p1Jmo9MCI%2BPC9zY3JpcHQ
    %2B h=55e1652a183
- Discounted 43 actual vulnerabilities from XSSFilt violations.
Performance

- XSSFilt has higher overhead. How much?
Performance

- XSSFilt has higher overhead. How much?
- modified $t_p$: 2.5%
  - Responses are served locally.
Performance

• XSSFilt has higher overhead. How much?
• modified $t_p^4$: 2.5%
  • Responses are served locally.
• Time spent in the taint-inference algorithm from a program trace / page load time: 0.5%
  • Ultimately negligible when factoring in network delay.
Implementation

- Firefox Trunk patch available shortly for download.
Implementation

• Firefox Trunk patch available shortly for download.
• Implemented as an extension to CSPs, with a separate XPCOM module for taint inference.
Implementation

- Firefox Trunk patch available shortly for download.
- Implemented as an extension to CSPs, with a separate XPCOM module for taint inference.
- A subset of the functionalities is being “properly” introduced into Firefox (Bug #528661)
Conclusions and Q&A

• XSSAuditor’s architecture offers many advantages.
Conclusions and Q&A

- XSSAuditor’s architecture offers many advantages.
- Our improvements over XSSAuditor increase protection without compromising usability or performance.
Conclusions and Q&A

• XSSAuditor’s architecture offers many advantages.
• Our improvements over XSSAuditor increase protection without compromising usability or performance.
• Questions?