xJS
Practical XSS Prevention for Web Application Development

Elias Athanasopoulos,
Vasilis Pappas, Antonis Krithinakis, Spyros Ligouras,
Evangelos P. Markatos
(FORTH-ICS)
Thomas Karagiannis
(Microsoft Research, Cambridge)
This talk is about

xJS

A practical framework for defending against Cross-Site Scripting attacks
Yet Another anti-XSS framework?
xJS Design Choices

- Web developer friendly
- Low computational overhead
- Backward compatibility in deployment
- DOM independent
- Cope with most of XSS
- Return-to-JavaScript attacks
Contributions

Return-to JavaScript attacks

xJS
Return-to JavaScript attacks
JavaScript Whitelisting

- Identify all JavaScript generated by the web application
- Mark all JavaScript as trusted (whitelist)
- Communicate the whitelist to the web browser
- The web browser executes only whitelisted scripts
BEEP

- For every trusted script keep a cryptographic hash (SHA1) in the whitelist
- The web browser executes only scripts, which their hash is found in the whitelist
Whitelisting is vulnerable to return-to-JavaScript attacks
## Return Oriented Programming

<table>
<thead>
<tr>
<th>Return-to Libc</th>
<th>Return-to JavaScript</th>
</tr>
</thead>
<tbody>
<tr>
<td>✺ Code injection does not contain code</td>
<td>✺ Code injection does not contain foreign code</td>
</tr>
<tr>
<td>✺ Code injection transfers execution to another place in the program’s code</td>
<td>✺ Code injection is based on whitelisted code from the web application</td>
</tr>
<tr>
<td>✺ Usually execution is transferred to libc</td>
<td>✺ Code injection alters the web application logic</td>
</tr>
</tbody>
</table>
Example 1

1: <html>
2: <head> <title> Blog! </title> <head>
3: <body>
4: <a href onclick="logout();">Logout</a>
5: <div class="blog_comments">
6:   { ... }
7: </div>
8: </body>
9: </html>
Annoyance

1: <html>
2: <head> <title> Blog! </title> <head>
3: <body>
4: <a href onclick="logout();">Logout</a>
5: <div class="blog_comments">
6: <img onload="logout();" src="logo.gif">
7: </div>
8: </body>
9: </html>
Example 2

1:   <html>
2:   <head> <title> Blog! </title> <head>
3:   <body>
4:   <div class="blog_entry" id="123">
5:     
6:     <input type="button" onclick="delete(123);">
7:   </div>
8:   <div class="blog_comments">
9:     
10: </div>
11: </body>
12: </html>
Data Loss

1: <html>
2: <head> <title> Blog! </title> <head>
3: <body>
4: <div class="blog_entry" id="123">
5:   { ... }
6: <input type="button" onclick="delete(123);"/>
7: </div>
8: <div class="blog_comments">
9:   <img onload="delete(123);"/>
10: </div>
11: </body>
12: </html>
Complete Takeover
xJS Architecture
Basic Components

🌟 Isolation operators

🌟 Action Based Policies
Isolation Operators

🌟 Inspired by Instruction Set Randomization (ISR)
🌟 Applied to portions of source (e.g. JavaScript)
🌟 Source isolation
XOR

- Fast, available as a hardware operation in most modern platforms
- No need for a JavaScript engine at the server side
**Action Based Policies**

- Multiple trust levels
- Policies expressed as actions:
  - De-isolate and execute
  - De-isolate and execute after under user confirmation
  - De-isolate with the X key and execute
xJS Session Example

Web Server

xJS Module

(modphp.so)

PHP

GET index.php

Accept: xjs

HTTP 200 OK

X-IO-KEY: 42

Web Browser

Web Browser
<table>
<thead>
<tr>
<th>Original Page</th>
<th>xJS Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: <code>&lt;div&gt;</code></td>
<td>1: <code>&lt;div&gt;</code></td>
</tr>
<tr>
<td>2: <code>&lt;img onload=&quot;render();&quot;&quot;&gt;</code></td>
<td>2: <code>&lt;img onload=&quot;AICtV...&quot;&gt;</code></td>
</tr>
<tr>
<td>3: <code>&lt;script&gt;</code></td>
<td>3: <code>&lt;script&gt;</code></td>
</tr>
<tr>
<td>4: <code>alert(&quot;Hello World&quot;);</code></td>
<td>4: <code>vpSUJTV2NHGwJyW/NHY...</code></td>
</tr>
<tr>
<td>5: <code>&lt;script&gt;</code></td>
<td>5: <code>&lt;/script&gt;</code></td>
</tr>
<tr>
<td>6: <code>&lt;/div&gt;</code></td>
<td>6: <code>&lt;/div&gt;</code></td>
</tr>
</tbody>
</table>
How Trusted JavaScript is Spotted?

- All JavaScript contained in files of a web application’s distribution is considered trusted
- JavaScript contained in a database is considered un-trusted
Evaluation
Attack Coverage

- 10,154 web pages from XSSed.com
- 1,381 still vulnerable
- Browse all 1,381 pages through a custom proxy
- All (100%) exploits were prevented
Server Side (Fast Ethernet)

Overhead of more than 60 msecs.
Server Side (DSL)

The overhead is fixed and less than a typical RTT of time!
Many short calls (less than 1 msec overhead).

Fewer longer calls (less than 10 msec overhead).

Fewer longer calls (less than 5 msec overhead).
User Experience

[Bar chart showing performance times for different tasks and browsers: WebKit (modified), WebKit (vanilla), Chromium (vanilla), Firefox (modified), Firefox (vanilla).]
Limitations
**eval() Semantics**

- eval() is changed to *de-isolate and evaluate*
- xeval() is provided to simply evaluate
Code-Mixing

Modern web applications mix server-side code (e.g. PHP) and JavaScript

```php
<?php if (user_exists($user)) { ?>
var msg = '<?php echo "Welcome" ?>';
<?php } else { ?>
var msg = "Registration Needed.";
<?php } ?>
```
Antonis Krithinakis, Elias Athanasopoulos, and Evangelos P. Markatos.

Isolating JavaScript in Dynamic Code Environments.

In Proceedings of the 1st Workshop on Analysis and Programming Languages for Web Applications and Cloud Applications (APLWACA), co-located with PLDI.

June 2010, Toronto, Canada.
Take Aways

1. JavaScript whitelisting is not sufficient
   ✨ Return-to JavaScript attacks

2. xJS
   ✨ Practical framework based on XOR for isolating legitimate JavaScript
Thank You!

Elias Athanasopoulos
FORTH-ICS
elathan@ics.forth.gr