Flash Parameter Injection

A SECURITY ADVISORY

A whitepaper from IBM Rational Application Security Team

Yuval Baror
Ayal Yogev
Adi Sharabani

September 24, 2008

Table of Contents

ABSTRACT

This paper covers a new attack called Flash Parameter Injection (FPI).

Prior research on Flash attacks, mainly by Stefano Di-Paola, has shown that when an attacker is able to access and control global Flash parameters, he can achieve attacks such as cross-site scripting through Flash, cross-site flashing, and changing the flow of the Flash movie. These attacks require direct access to the Flash movie itself, setting the parameters through the URI. They cannot be deployed when the Flash movie is embedded in an HTML page and unable to access the original DOM.

In many cases a Flash file will only launch within its parent HTML, and will not load correctly as a URI. In such cases a regular cross site flashing attack is not possible.

This paper shows some techniques in which an attacker can inject global Flash parameters when the movie is embedded in a parent HTML page. These injected parameters can grant the attacker full control over the page DOM, as well as control over other objects within the Flash movie. This can lead to more elaborate attacks that take advantage of the interaction between the Flash movie and the HTML page in which it is embedded. The FPI techniques described in this paper thus increase the surface area of possible Flash attacks, introducing a new set of vulnerable Web sites.

In this paper we will discuss five different techniques. We will start with a simple DOM-based injection in which an attacker can use the URI of the original HTML page to inject Flash parameters in such a way that they go undetected by Web servers, thus rendering protection mechanisms like IDS and IPS useless. (It will also be shown that the function "encodeURI", often used to sanitize the URI, is not sufficient in this case.) The second technique utilizes HTML form or URI parameters for injecting global Flash parameters. In the third technique shown we will use a direct Flash movie reference to override the value of such parameters. Fourthly we will describe a technique that uses the attributes of the HTML "object" tag to achieve FPI. Finally we will show a persistent Flash parameter injection. This technique causes the injection to become persistent, meaning it will remain active between different sessions, and even after the vulnerability allowing the injection has been fixed.

Some of the techniques covered in this paper have been discovered in widely used real-world applications.

BACKGROUND

Adobe Flash is a widely deployed technology used for adding animations and interactivity to Web pages. Flash movies can either be embedded in HTML pages or played in a standalone Flash player.

Flash movies can contain simple scripts in a language called "ActionScript". These scripts are used to enhance Flash movie capabilities and allow the creation of more dynamic Web pages and of rich Internet applications.

EMBEDDING FLASH MOVIES IN HTML

Embedding a Flash movie inside an HTML page can be done in the following way:

```
<body>
   <h1>My Flash Movie</h1>

   <object type="application/x-shockwave-flash" width="550" height="400">
      <param name="allowScriptAccess" value="sameDomain" />
      <param name="movie" value="myMovie.swf" />
      <param name="quality" value="high" />
      <param name="bgcolor" value="#ffffff" />

      <embed src="myMovie.swf" width="550" height="400"></embed>
   </object>
</body>
```

Example 1: Embedding Flash in HTML

When the Web page is accessed, the Flash movie is automatically downloaded by the browser and displayed to the user within the Web page.

When embedded within an HTML page, the Flash movie has access to the page's DOM; meaning that JavaScript code can be executed through the movie's ActionScript code. Since the whole DOM can be accessed by the movie, the movie can, for example, access cookies on the user's computer belonging to the domain from which the Flash is loaded.

Another way in which Flash movies can be embedded in an HTML page is directly through the browser, when the user browses to the URL of the Flash movie. In this case, a "dummy" HTML page is created, that contains the Flash movie as an embedded object, and this page is displayed to the user:

```
<html>
   <body marginwidth="0" marginheight="0">
      <embed width="100%" height="100%" name="plugin"
       src="http://URL/movie.swf" type="application/x-shockwave-flash"/>
   </body>
</html>
```

Example 2: HTML created on direct Flash reference

Note that in this case, even if the original Flash movie was embedded in an HTML page, the Flash movie is no longer embedded in *that* HTML page and therefore will not have access to the its DOM.

PASSING ARGUMENTS TO FLASH MOVIES

There are several methods for passing arguments to the Flash movie from its embedding object. The arguments passed to the movie are then available within the Flash movie as global variables located under the '_root' movie clip object.

Here is an example of ActionScript2 code using a global argument called 'a':

```
if (_root.a == undefined) {
      _root.a = 0; // Default value
}
```

Example 3: ActionScript 2 code reading a global variable

This is a very common code snippet in Flash movies. In the Flash attack papers mentioned above, Stefano Di-Paola showed how an attacker might take advantage of such code to achieve cross-site scripting, cross site-flashing or other attacks. We will mention three of these methods.

Method 1: Direct Reference

The first method for passing arguments is to reference the Flash file directly and pass the arguments in the URI itself (this is the same as HTTP parameters using the GET method).

For example:

```
http://URL/myMovie.swf?a=5&b=hello
```

Example 4: Passing arguments using direct reference

In this example, two arguments are passed to the Flash file. The first is 'a' that receives the value '5' and the second is 'b' that receives the value 'hello'. When this method is used the Flash file is not embedded in the original HTML page, but is instead embedded in the "dummy" HTML page that is created automatically.

Method 2: Embedded URI

The second method is to pass the arguments in the URI of the embedded object. For example:

```
<body>
    <object type="application/x-shockwave-flash"
            data="myMovie.swf?a=5&b=hello"
            width="600" height="345">
    </object>
</body>
```

Example 5: Passing arguments in an embedded URI

In this case the Flash file is still embedded in the original HTML page.

Method 3: Using the 'flashvars' attribute

The third method uses the 'flashvars' attribute. This attribute can be specified within the <object> tag or as a <parameter> tag.

Here is an example of an embedded Flash file receiving arguments embedded inside an HTML page:

```
<body>
    <object type="application/x-shockwave-flash" data="myMovie.swf"
            flashvars="a=5&b=hello" width="600" height="345">
    </object>
</body>
```

Example 6: Passing arguments using 'flashvars'

Like the previous method, this method can be used to pass arguments to a Flash movie embedded inside an HTML page.

As mentioned in the Abstract, in the papers released so far the method used for overriding global Flash parameters was using direct access to the Flash movie and passing the arguments through the URI (the first method described here). This attack method is unable to access the DOM of the original HTML page that the Flash was embedded in, thus limiting the attack's scope.

TECHNIQUES

In this section we will present new ways for overriding the values of global Flash parameters while the Flash movie is embedded in the parent HTML page, thus gaining access to the page's DOM.

We will demonstrate several techniques for achieving Flash Parameter Injection. We will start with simple techniques, show that some sanitization functions such as "encodeURI" do not protect against them, and work our way to Persistent Flash Parameter Injection – a method used to infect Flash files in such a way that remains active every time the user plays the Flash movie. Persistent FPI will cause an attack on the victim every time the vulnerable Flash file is viewed. The victim will stay infected even after the vulnerability has been fixed in the Flash file itself.

1. DOM BASED FLASH PARAMETER INJECTION

This form of FPI can be achieved when the JavaScript 'document.location' variable is used as part of a Flash argument.

Here is an example of part of a vulnerable HTML file:

```
<script type="text/javascript" language="JavaScript">
    var s = '';
    var loc = encodeURI(document.location);

    s += '<object>';
    s += '  <embed src="myFlash.swf" flashvars="location='+ loc +'">';
    s += '  </embed>';
    s += '</object>';
    document.write(s);
</script>
```

Example 7: javascript code creating a Flash object

Normal use of this Webpage would result in the following HTML page being generated from the JavaScript above:

```
<object>
    <embed src="myFlash.swf" flashvars="location=http://URL/index.htm">
    </embed>
</object>
```

Example 8: HTML generated by the javascript code

In this case, the developer of the HTML page passed the URL of the page to the Flash file in the global parameter "location". The developer was aware of Web application security issues and used the function "encodeURI" in order to avoid attacks such as DOM based cross-site scripting.

Despite the developer's caution, an attacker can still lure an unsuspecting victim to click on a link such as this:

```
http://URL/index.htm#&globalVar=e-v-i-l
```

Example 9: Attack URI

Clicking on the above link will result in the following HTML page:

```
<object>
    <embed src="myFlash.swf"
           flashvars="location=http://URL/index.htm#&globalVar=e-v-i-l">
    </embed>
</object>
```

Example 10: HTML created from the attack

This time, two variables are passed to the Flash file. The first is "location" with the value "http://URL/index/htm#", and the second is "globalVar" with the value "e-v-i-l". This means that the attacker can control any global Flash variable of the Flash file embedded inside the original HTML page.

Using '?' instead of '#' would also work and result in the same attack. However, the advantage of using the crosshatch sign (#) is that the data that follows it is not sent to the server. This makes the attack invisible to an IDS or an IPS on the attacked site.

The attack succeeds because the JavaScript function "encodeURI" does not encode all characters (for example: '&' and '?'). Although the function is widely used for protecting against DOM based cross site scripting, it is not immune to DOM based Flash Parameter Injection. The sanitation in this case should have been done according to the context and the methods "escape" or "encodeURIComponent" should have been used.

2. REFLECTED FLASH PARAMETER INJECTION

When the name of the Flash movie is exposed as a form or a URL parameter, the application may be vulnerable to Flash parameter injection.

An attacker may control the Flash movie loaded in order to load a malicious Flash movie, but even if precautions have been taken, by only allowing the loading of legitimate movies from a specific domain, using a direct Flash reference can facilitate overriding global parameters of a Flash movie embedded inside the HTML page.

For example, consider the following Perl-CGI script deployed on a Web server, where 'params' is an associative array of all the HTML parameters:

```
# Embed the Flash movie
print '<object type="application/x-shockwave-flash" data="' . $params{movie}
. '"></object>';
```

Example 11: Perl-CGI code creating HTML with a dynamic Flash movie

In this code, the name of the Flash movie is taken from the form or URL parameters received in the request and placed in the generated HTML page as the name of the Flash movie to load.
An attacker may create the following link to override some global Flash parameters:

```
http://URL/index.cgi?movie=myMovie.swf?globalVar=e-v-i-l
```

Example 12: Attack URI example

Notice the movie name contains a question mark followed by parameters that will be passed to the Flash file. When the victim clicks on this link, the following HTML snippet will be generated and sent to him:

```
    <object type="application/x-shockwave-flash"
            data="myMovie.swf?globalVar=e-v-i-l" >
    </object>
```

Example 13: The HTML generated by the CGI script with the injected parameter

The browser will load the embedded Flash file with the extra global parameters that the attacker supplied.

To control more than one argument at a time, the attacker simply separates them using URI encoded ampersands:

```
http://URL/index.cgi?movie=myMovie.swf?globalVar=e-v-i-l%26otherVar=b-a-d
```

Example 14: Injecting multiple parameters

Even if the Web application developer made sure that the name of the movie was loaded from a specific domain, the Flash parameter injection vulnerability still exists.

Another version of this attack can be achieved when the attack controls the name of a Flash movie loaded within another Flash movie.

Here's an example of ActionScript2 code that is vulnerable to this attack. This code loads a Flash file into the existing Flash file:

```
if (_root.moviename == undefined) {
      _root.moviename = "http://URL/myMovie.swf"; // Default value
}

loadMovieNum(_root.moviename, 1);
```

Example 15: ActionScript code loading an internal Flash file

In case the Flash file is loaded in the following manner:

```
http://URL/myMovie.swf?moviename=internalMovie.swf?globalVar=e-v-i-l
```

Example 16: Attack example

The internal Flash file will be loaded with the global parameter "globalVar" set to "e-v-i-l".

3. REFLECTED FLASH PARAMETER INJECTION (PIGGYBACKING FLASHVARS)

When one or more of the arguments passed in the 'flashvars' attribute are received from HTML parameters without proper sanitization, it is possible to carry out an FPI attack via HTML parameters.

Let's suppose a Perl-CGI script on the server contains the following code (params is an associative array containing all the parameters passed to the page):

```
# Read the 'language' parameter
my $language;
if (exists($params{language})) {
      $language = $params{language};
}
else {
      $langauge = "English"; # Default value
}

# Embed the Flash movie
print '<object type="application/x-shockwave-flash" data="myMovie.swf"
flashvars="language=' . $language . '"></object>';
```

Example 17: Perl-CGI code creating HTML with a Flash object

If an attacker can cause the victim to click on the following link:

```
http://URL/index.cgi?language=English%26globalVar=e-v-i-l
```

Example 18: Attack URI example

The '%26' will be encoded into an ampersand (&), causing the creation of the following HTML code:

```
<object type="application/x-shockwave-flash" data="myMovie.swf"
        flashvars="language=English&globalVar=e-v-i-l">
</object>
```

Example 19: HTML generated by the CGI code with an additional argument

The attacker has now gained control over any global Flash variable of the Flash file embedded in the original HTML page.

This attack can also be performed if the arguments to the Flash movie are passed in the embedded URI (the second method shown for passing arguments to Flash files).

4. FLASHVARS INJECTION

In certain cases, some of the attributes of the <object> tag are received as parameters. For example, the width and height of the Flash movie could be created dynamically by the following Perl-CGI script ('params' is an associative array of all the HTML parameters):

```
# Embed the flash movie
print '<object type="application/x-shockwave-flash" ' .
      'data="myMoive.swf" ' .
      'width="' . $params{width} .
      '" height="' . $params{height} .
      '"></object>';
```

Example 20: A Perl-CGI script creating a Flash object with a dynamic width and height

In this example, the 'width' and 'height' parameters have been passed directly into the output HTML file without proper sanitization. A malicious attacker can cause an unsuspecting victim to click on the following link:

```
http://URL/myMovie.cgi?width=600&height=600%22%20flashvars=%22globalVar=e-v-
i-l
```

Example 21: Attack example

This URL will be decoded to:
`http://URL/myMovie.cgi?width=600&height=600" flashvars="globalVar=e-v-i-l`

Clicking on this link will cause the following HTML page to be sent to the client's browser:

```
<object type="application/x-shockwave-flash"
        data="myMovie.swf"
        width="600"
        height="600" flashvars="globalVar=e-v-i-l">
</object>
```

Example 22: The HTML generated from the CGI script

In this way the 'flashvars' attribute has been injected into the <object> tag without the creator of the Web page ever intending to allow arguments to be passed into the Flash file.

5. PERSISTENT FLASH PARAMETER INJECTION

Shared local Flash objects (sometimes referred to as Flash cookies) are used to allow Flash movies to save data that will persist across multiple sessions. The data inside that shared object can be saved and loaded inside the Flash movie.

Here is an example for storing data inside shared Flash objects:

```
// Create a shared object
mySharedObject = SharedObject.getLocal("sharedObjectName");

// Store data in the shared object
mySharedObject.data.name = "jsmith";
mySharedObject.data.homepage = "http://demo.testfire.net";

// Flush
mySharedObject.flush();
```

Example 23: storing data inside shared objects

And here's an example for loading data from shared objects:

```
// Create a new shared object or read an existing one
mySharedObject = SharedObject.getLocal("sharedObjectName");

// Check whether variable name exists
if (mySharedObject.data.name == null)
{
  // Shared object doesn't exist
}
else
{
  // Read the name
  name = mySharedObject.data.name;

  // Read the homepage
  homepage = mySharedObject.data.homepage;
}
```

Example 24: loading data from shared objects

If shared objects are used to save data that is entered by the user and later inserted into one of the "potentially dangerous native functions" (PDNF) mentioned in Stefano Di-Paola's whitepapers, persistent Flash parameter injection may occur, with malicious code being injected into the Flash file and executed, every time the Flash movie is loaded.

Here is a code example of a Flash file vulnerable to persistent Flash parameter injection:


```
// Create a new shared object or read an existing one
mySharedObject = SharedObject.getLocal("flashToLoad");

if (_root.flashfile == undefined)
{
    // Check whether there is a shared object saved
    if (mySharedObject.data.flash == null)
    {
        // Set a default value
        _root.flashfile = "defaultFlash.swf";
    }
    else
    {
      // Read the flash file to load from the shared object
      _root.flashfile = mySharedObject.data.flash;
    }
}

// Store the flash file's name in the shared object
mySharedObject.data.flash = _root.flashfile;

// Load the flash file
getURL(_root.flashfile);
```

Example 25: Flash vulnerable to persistent FPI

In this Flash file, if the global variable "_root.flashfile" is set to undefined, its value is set either by reading a stored in a shared object (if there is one), or by setting a default value. This value is later stored into a shared object and used as an argument to the potentially dangerous native function "getURL".

If an unsuspecting user is lured by an attacker to click on link like this:

```
http://URL/vulnerable.swf?flashfile=javascript:alert(document.domain)
```

Example 26: Attack example

The result will be not merely a one-time execution of the JavaScript code in the victim's browser in the context of the domain with the vulnerable Flash file, but every time the Flash is loaded, whether by direct reference or embedded inside the same domain, the JavaScript will be executed again.

Tracking devices such as an IDS or an IPS, deployed on the vulnerable site, will notice the infection only, not the cause (and even this can be made invisible using a small modification and the hatch sign). Subsequent requests to the site will appear completely normal, but the attack will still be executed on the victim's browser.

This attack enhances the previous Flash attacks presented, enabling them to become persistent.

A shrewd attacker may use this attack to execute his attack only when the Flash is embedded inside the HTML page. The first execution is used to infect the victim. Subsequently, every time the code is executed it will check whether the Flash is embedded in the correct HTML page and then begin the attack.

One of the most important impacts of this attack is its persistence. Even if the code is later fixed to prevent the injecting of parameter values, values already injected will remain within the shared object indefinitely, causing repeat attacks each time the Flash file is loaded.

IMPACT

The techniques described in this paper have one major advantage over those discussed in existing papers. Due to the fact that the Flash movie is attacked while embedded inside its parent HTML page, more elaborate attacks can be achieved, that take advantage of the interaction between the Flash file and the parent page. These attacks will work even if the Flash movie creator has blocked the execution of the Flash file outside the parent HTML page.

RECOMMENDATIONS

When embedding Flash movies in Web pages, Web application developers must take care to sanitize input. Developers must be aware that that the JavaScript function 'encodeURI' is not always sufficient for sanitizing input intended for global Flash parameters. A more specific sanitation method should be applied or, better yet, a positive security model stating the exact characters accepted as input.

Web applications vulnerable to DOM based FPI cannot assume that their clients have not been attacked, even if traffic is monitored by security mechanisms such as IPS or IDS, since the attack fragment (the part of the URI after the cross hatch sign [#]) is not sent to the Web server by the browser, so the Web server has no way of detecting the attack.

Flash movie developers must be aware that saving information received as global parameters in local shared objects can lead to persistent Flash parameter injection. Flash movies vulnerable to persistent FPI will remain vulnerable to the Flash attack caused by overriding the global parameter (cross-site scripting, cross-site flashing, etc.) even after the FPI vulnerability has been fixed. In such cases the Flash file itself will need to be fixed and recompiled, changing the name of the local shared object used.

REFERENCES

1. ActionScript 2.0 Language Reference: http://livedocs.adobe.com/flash/8/main/Part4_ASLR2.html
2. OWASP Flash Security Project homepage: http://www.owasp.org/index.php/Category:OWASP_Flash_Security_Project
3. Testing Flash Applications (Stefano Di-Paola): http://www.owasp.org/images/8/8c/OWASPAppSec2007Milan_TestingFlashApplications.ppt
4. Finding Vulnerabilities in Flash Applications (Stefano Di-Paola): http://www.owasp.org/images/d/d8/OWASP-WASCAppSec2007SanJose_FindingVulnsinFlashApps.ppt

