IBM Application Security Insider: JavaScript Code Flow Manipulation, and a real world example advisory

IBM Application Security Insider: JavaScript Code Flow Manipulation, and a real world example advisory - Ory Segal, Adi Sharabani, blog.watchfire.com.

- Published: date not stated
- Original: <http://blog.watchfire.com/wfblog/2008/06/javascript-code.html>
- Preserved from: http://blog.watchfire.com/wfblog/2008/06/javascript-code.html (stored) on 2026-08-16
- Capture timestamp: 20140723103435
- 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.

We recently researched an interesting DOM-based XSS vulnerability in Adobe Flex 3 applications that exploits a scenario in which two frames (parent & son) interact with each other, without properly validating their execution environment.

In our research, we have seen that in some cases, it is possible to manipulate JavaScript code flow, by controlling the environment in which it runs. Specifically, we managed to return hacker-controlled boolean values to conditional statements, and by that force the application to be vulnerable to an existing DOM-based XSS, which was otherwise unexploitable.

The advisory presented herein, is a real world example of the research mentioned above, and contains two XSS variants. The **second** of which**,** makes use of the JavaScript Flow Manipulation technique.

Begin Advisory #

This advisory describes a new security vulnerability found in auto-generated code created by Adobe Flex 3 (Builder & SDK) that uses the default **HistoryManager** or **Deep Linking** support.

Attack Variant #1: DOM Based Cross-Site Scripting

The following text, which describes the HistoryManager and Deep Linking support in Adobe Flex, was taken from the official Adobe documentation:

> "The Flex History Manager lets users navigate through a Flex application by using the web browser's back and forward navigation commands. For example, a user can navigate through several Accordion container panes in a Flex application, and then click the browser's Back button to return the application to its previous states. The HistoryManager class provides a subset of functionality that is provided by the BrowserManager class and deep linking. In general, you should use the BrowserManager class and deep linking for maintaining state in an application and manipulating URLs and browser history, but the HistoryManager class can be useful under some circumstances, such as if you are maintaining a Flex 2.x application. For more information about deep linking and the BrowserManager class, see About deep linking. History management is implemented as a set of files that are referenced in the application's wrapper. By default, Adobe Flex Builder generates a wrapper that supports history management, but you can disable it. When you deploy an application that uses the HistoryManager, you must also deploy the history management files such as history.css, history.js, and historyFrame.html. These are the same files that are used by the BrowserManager for deep linking support. For more information, see Deploying applications that use deep linking."

The following code was taken from the historyFrame.html:

```
...

function processUrl()
{
    var pos = url.indexOf("?");
    url = pos != -1 ? url.substr(pos + 1) : "";
    if (!parent._ie_firstload) {
      parent.BrowserHistory.setBrowserURL(url);
      try {
        parent.BrowserHistory.browserURLChange(url);
```

```
            } catch(e) { }
        } else {
            parent._ie_firstload = false;
        }
}

var url = document.location.href;
processUrl();
document.write(url);
...
```

As can be seen from the code above, the url variable, holds the document.location.href string, and is later on written to the HTML document. So, in order for the XSS attack to work, the malicious payload should be injected into the URL. Here's an exploit URL:

http://www.some.site/flex_html_wrapper.html#<script>alert(document.cookie)</script</script)>

Note 1: due to how Flex HistoryManager is implemented, the above exploit URL will only work on Microsoft Internet Explorer

Note 2: in the example above, the file /flex_html_wrapper.html is the HTML wrapper file of the Flex SWF

**Attack Variant #2: DOM Based XSS Using JavaScript Flow Manipulation **

From a quick research into how Flex 3 applications support deep linking ("browser navigation integration"), we have noticed that the attack we described above, will only work if the developer explicitly makes use of either **HistoryManager** or **BrowserManager** classes. In such case, the application will use the vulnerable code presented above (i.e. it will load the vulnerable file **/history/historyFrame.html** into the browser).

This fact limits our attack vector to sites that were designed to actively use history management / deep linking (**note**: all applications that were compiled with "enable integration with browser

navigation" ship with the vulnerable file, but they don't load it into the browser unless the above objects are used in the code).

When we looked at the JavaScript code in **/history/historyFrame.html**, we saw that calling it directly in order to mount the DOM-based XSS attack (against all Flex 3 sites that include that file) will not work. This is because the JavaScript first calls the function processUrl before performing the vulnerable **document.write(url)**

The function processUrl, contains the following lines of code:

```
if (!parent._ie_firstload) {
    parent.BrowserHistory.setBrowserURL(url);
...
```

When there is no parent document (or if the parent document does not contain the object _ie_firstload), the condition inside the parenthesis is evaluated to TRUE. This in turn, causes the line:

parent.BrowserHistory.setBrowserURL(url);

to get executed, resulting in a runtime exception, since no such objects and methods actually exist (there is no parent document). At this point, the attack will fail.

In order to bypass this obstacle, we have created a (malicious) parent document, which includes an IFrame whose source is the vulnerable HTML page /**history/historyFrame.html**. In addition, we have added another IFrame (called _ie_firstload), whose role will be explained below.

The malicious HTML page looks like this (line wrapped):

```
<!-- HTML source of page, hosted on http://www.evil.site/ -->
```

```
<html>
```

```
 <body>
```

```
  <iframe name="_ie_firstload"></iframe>
```

```
  <iframe src="http://www.vuln.site/app/history/historyFrame.html?
```

```
#<script>alert('xss')</script>"></iframe>
```

```
</body>
```

```
</html>
```

Upon visiting this malicious page, which is hosted on http://www.evil.site/ the victim's browser issues a request to the vulnerable Flex application that is hosted on http://www.vuln.site/. The request exploits the DOM-based XSS vulnerability that was mentioned in the first section (variant #1).

Here's a quick explanation of the attack:

- The JavaScript code that is executed in */history/historyFrame.html* *(see JS code in the previous section), looks at the Boolean value that parent._ie_firstload returns.
- Since our malicious parent document actually includes a child node with the same name (a bogus IFrame element that we named "_ie_firstload"), the script flow is manipulated. (we do not enter the first block of the IF statement)
- The JavaScript code that was just manipulated goes ahead to the ELSE statement
- The function processUrl exits cleanly, and our DOM-based XSS attack succeeds.

Pay attention to #2 - the child IFrame was expecting its parent to return TRUE or FALSE depending on the existence of a JavaScript object (_ie_firstload), but since we controlled the parent's DOM, we have substituted the JavaScript object, with an IFrame. The IFrame's existence in the parent DOM allows us to fool the child IFrame into believing that such object exists.

We have decided to use an IFrame instead of a regular JavaScript object, since the browser's same origin policy will not allow the child IFrame to access JavaScript objects originating from a different domain (www.evil.site). **Nevertheless, the browser will allow the child IFrame to traverse the parent's IFrame structure. **In our case, the JavaScript code flow manipulation technique was relying on this browser behavior.

To sum things up, all a hacker needs to do in order to exploit this vulnerability, is host a malicious HTML page as shown above, which points to a Flex application that includes the file */history/historyFrame.html. **

This file exists by default in all Adobe Flex 3 applications that were created either by Flex 3 Builder or the Flex 3 SDK (regardless if the developer chose to use HistoryManager or BrowserManager)

Impact

First and foremost, this vulnerability is extremely severe, since every Flex web application that is developed using Flex Builder 3 or the Flex 3 SDK (see the next paragraph), includes the file **/history/historyFrame.html** (this file exists by default), and thus is vulnerable to DOM-based XSS.

In order for the vulnerable files to be included in the Flex application, the developer has to enable "integration with browser navigation". This option is **enabled by default**, and can be configured from the project properties.

The most severe impact of the vulnerability described in this document is achieving a successful DOM based cross-site scripting attack. If an application is vulnerable to attacks of this type, a

remote attacker can execute a malicious script in the context of the victim's browser which can access cookies, session tokens and other sensitive data kept by the browser for the vulnerable web application.

Fix Recommendations:

The following fix recommendations are taken from the Adobe security bulletin:

> Adobe recommends all Flex 3 developers who have enabled History Management update applications created with Flex 3 and their Flex 3 product installations with the following instructions: - Flex 3 users (both Flex 3 SDK and Flex Builder 3) should update their product installations with the Flex 3.0.2 SDK update. - Flex 3 users who have enabled History Management in their currently deployed Flex 3 web applications should update all instances of the **historyFrame.html** file with the updated file. The three instances of historyFrame.html in the Flex 3 SDK installation can be found in the following locations: {install root}/templates/html-templates/client-side-detection-with-history/history/historyFrame.html {install root}/templates/html-templates/express-installation-with-history/history/historyFrame.html {install root}/templates/html-templates/no-player-detection-with-history/history/historyFrame.html

Acknowledgements:

- We would like to commend Adobe for their quick response and the efficient way in which they handled this security issue. We wish that other vendors would be as responsible as Adobe for the protection of their customers.
- This research was performed by the following IBM Rational Application Security Group Members: Ory Segal & Adi Sharabani, with the help of Ayal Yogev.
- Special thanks goes to Amit Klein for his technical review and useful comments.

CVE ID: CVE-2008-2640

End Advisory # #