

# **Malicious PDF File Analysis -**

## **No. 14**

### **Stage 2:**

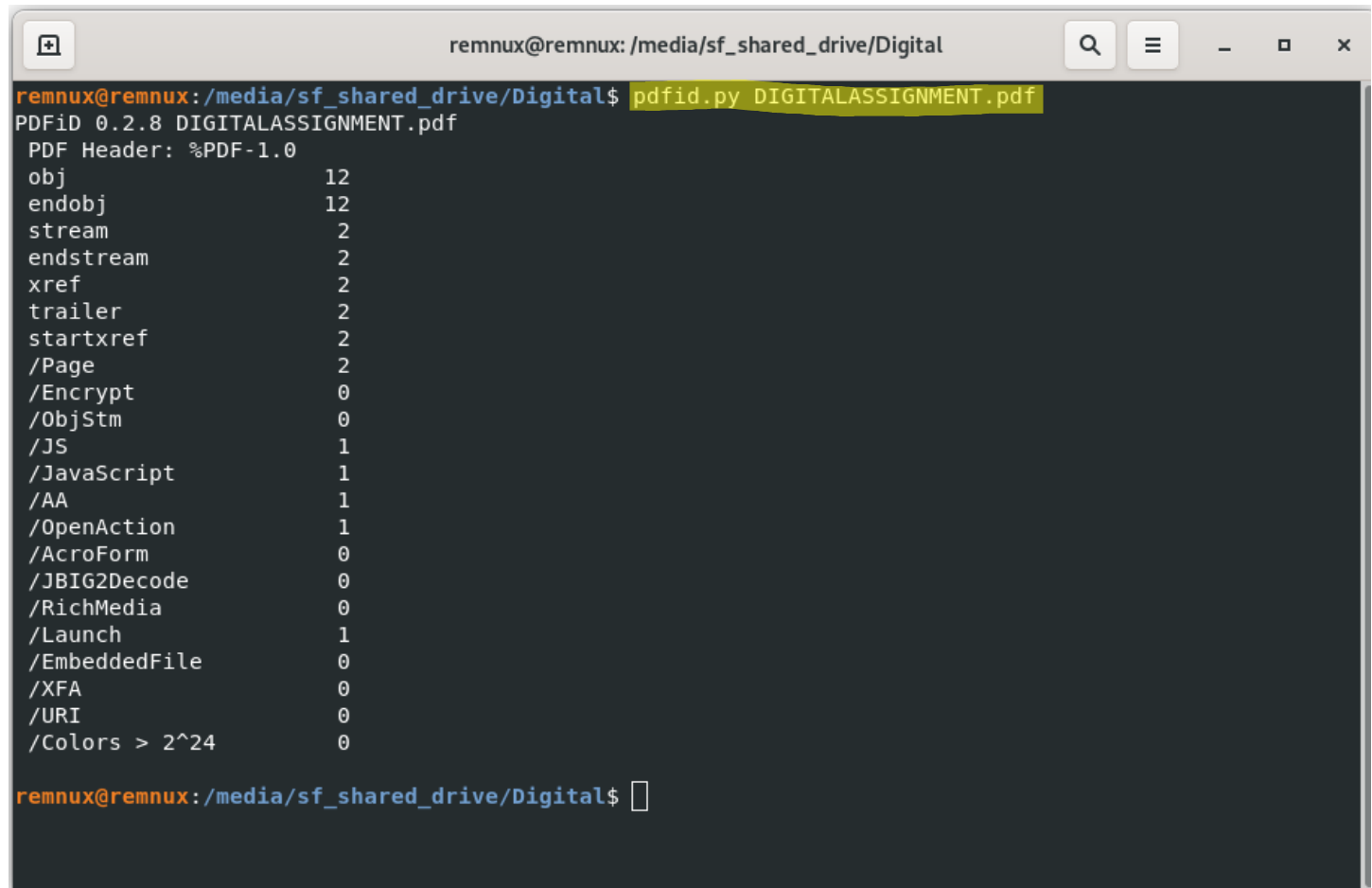
Your job is to investigate the content of a given malicious pdf file. Using the PDF analyzing tools offered by the REMnux tool, spider monkey, sctest, or PDF Stream Dumper, address the following questions/activities:

1. Report the number of objects in the file.
2. Determine whether the file is compressed or not.
3. Determine whether the file is obfuscated or not.
4. Find and Extract JavaScript.
5. De-obfuscate JavaScript.
6. Extract the shell code.
7. Create a shell code executable
8. Analyze shell code and determine what it does or even execute it using sctest or spider monkey.
9. What is the secret code?

## Given File to analyze – Group 14

### 1. Report the number of objects in the file.

We found 12 objects in the malicious pdf file using pdfid.py.



```
remnux@remnux: /media/sf_shared_drive/Digital
remnux@remnux:/media/sf_shared_drive/Digital$ pdfid.py DIGITALASSIGNMENT.pdf
PDFiD 0.2.8 DIGITALASSIGNMENT.pdf
PDF Header: %PDF-1.0
obj                12
endobj             12
stream             2
endstream          2
xref               2
trailer            2
startxref          2
/Page              2
/Encrypt           0
/ObjStm            0
/JS                1
/JavaScript         1
/AA                1
/OpenAction        1
/AcroForm           0
/JBIG2Decode       0
/RichMedia         0
/Launch            1
/EmbeddedFile      0
/XFA                0
/URI                0
/Colors > 2^24     0

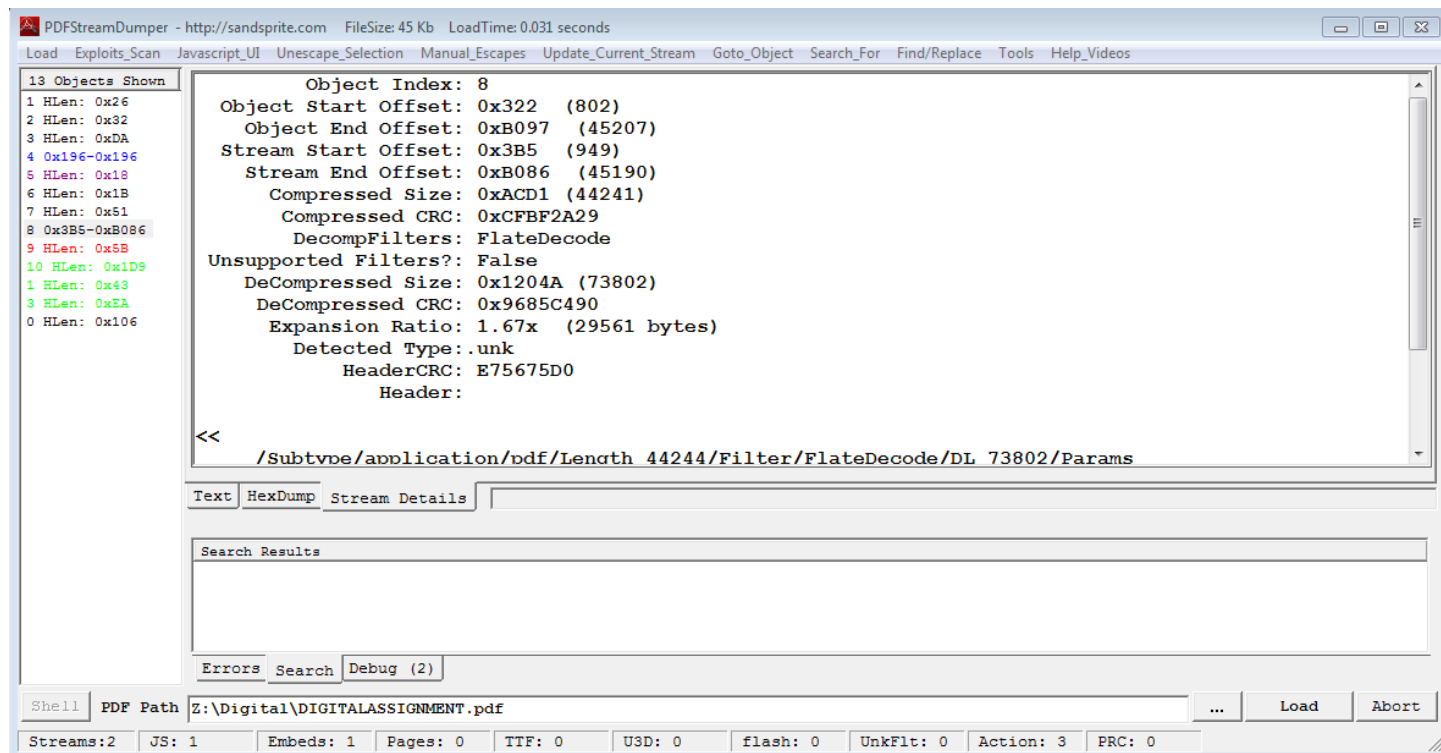
remnux@remnux:/media/sf_shared_drive/Digital$
```

## 2. Determine whether the file is compressed or not.

The given malicious file is compressed.

Command used – pdf-parser.py

When we used the Stream dumper it showed that the object8 is compressed using the filter and Flat code.



```

Usage: pdf-parser.py [options] pdf-file|zip-file|url
pdf-parser, use it to parse a PDF document

pdf-parser.py: error: ambiguous option: --obj (--object, --objstm?)
remnux@remnux: /media/sf_shared_drive/Digital$ pdf-parser.py DIGITALASSIGNMENT.pdf
PDF Comment '%PDF-1.0\r\n'

obj 1 0
Type: /Catalog
Referencing: 2 0 R

<<
  /Pages 2 0 R
  /Type /Catalog
>>

obj 2 0
Type: /Pages
Referencing: 3 0 R

<<
  /Count 1
  /Kids [ 3 0 R ]
  /Type /Pages
>>

obj 3 0
Type: /Page
Referencing: 4 0 R, 2 0 R

<<
  /Contents 4 0 R
  /Parent 2 0 R
  /Resources
    <<
      /Font
        <<
          /F1
            <<
              /Type /Font
              /Subtype /Type1
              /BaseFont /Helvetica
              /Name /F1
            >>
          >>
        >>
      >>
    >>
  /Type /Page
  /MediaBox [ 0 0 795 842 ]
>>

obj 4 0

```



```
Activities Terminal Oct 9 21:45
remnux@remnux: /media/sf_shared_drive/Digital

obj 3 0
Type: /Page
Referencing: 4 0 R, 2 0 R, 10 0 R

<<
  /Contents 4 0 R
  /Parent 2 0 R
  /Resources
    <<
      /Font
        <<
          /F1
            <<
              /Type /Font
              /Subtype /Type1
              /BaseFont /Helvetica
              /Name /F1
            >>
          >>
        >>
      >>
    >>
  /Type /Page
  /MediaBox [ 0 0 795 842 ]
  /AA
    <<
      /O 10 0 R
    >>
  >>
>>

xref
trailer
<<
  /Size 11
  /Prev 429
  /Root 1 0 R
  /Info 0 0 R
>>

startxref 46143
PDF Comment '%EOF\r\n'

remnux@remnux: /media/sf_shared_drive/Digital$
```

### 3. Determine whether the file is obfuscated or not.

The file is not obfuscated as we see the encoded stream is in Object 8, but the java script code resides in the object-9 we can clearly say that the file is not Obfuscated.

Package used – [peepdf.py](#)

```
Activities Terminal Oct 6 10:23
root@remnux: /media/sf_shared_drive

File: DIGITALASSIGNMENT.pdf
MD5: acc5473e6acac941d95ec206e19cfbae
SHA1: 33f04ce6d5a9bb64ea3203dd88e76435827999c3
SHA256: f5ad7e9a4f95860032ae3f994c4526c03dfc0d214ad18af10ef41a229fd3f559
Size: 46403 bytes
Version: 1.0
Binary: False
Linearized: False
Encrypted: False
Updates: 1
Objects: 12
Streams: 2
URIs: 0
Comments: 0
Errors: 0

Version 0:
  Catalog: 1
  Info: 0
  Objects (4): [1, 2, 3, 4]
  Streams (1): [4]
  Encoded (0): []

Version 1:
  Catalog: 1
  Info: 0
  Objects (8): [1, 3, 5, 6, 7, 8, 9, 10]
  Streams (1): [8]
  Encoded (1): [8]
  Objects with JS code (1): [9]
  Suspicious elements:
    /OpenAction (1): [1]
    /Names (2): [6, 1]
    /AA (1): [3]
    /JS (1): [9]
    /Launch (1): [10]
    /JavaScript (1): [9]
    /EmbeddedFiles: [5]

remnux@remnux: /media/sf_shared_drive/Digital$ js-
js-asci11 js-beautify js-file js-patched
remnux@remnux: /media/sf_shared_drive/Digital$ js-didier script
js-didier: command not found
remnux@remnux: /media/sf_shared_drive/Digital$ pdftextract -s DIGITALASSIGNMENT.pdf
[error] Breaking on: "ndstreamre..." at offset 0xb000
[error] Last exception: [Origami::InvalidObjectError] Failed to parse object (no:0,gen:0)
-> [Origami::InvalidStreamObjectError] Stream shall end with a "endstream" statement
Extracted 1 PDF streams to 'DIGITALASSIGNMENT.dump/streams'.
remnux@remnux: /media/sf_shared_drive/Digital$ js-dider
js-dider: command not found
remnux@remnux: /media/sf_shared_drive/Digital$ sudo apt install libmozjs-24-0v5 libmozjs-24-bin

[remnux@remnux: ~] [Digital] [Js52 Installation On A Debian, Ub...]
```

#### 4. Find and Extract JavaScript.

We can see that the JavaScript is hidden in the object 9 from the previous answer, so we used pdf-parser.py for the object 9 to extract the JavaScript.

```
Activities Terminal Oct 6 09:30
remnux@remnux: /media/sf_shared_drive/Digital

obj 9 0
Type: /Action
Referencing:
^M<</S/JavaScript/JS(this.exportDataObject({ cName: "template", nLaunch: 0 }));)/Type/Action>>

<<
/S /JavaScript
/JS (this.exportDataObject({ cName: "template", nLaunch: 0 }));)
/Type /Action
>>

^M<</S/JavaScript/JS(this.exportDataObject({ cName: "template", nLaunch: 0 }));)/Type/Action>>
^M<</S/JavaScript/JS(this.exportDataObject({ cName: "template", nLaunch: 0 }));)/Type/Action>>

~
~
~
~
```

#### 5. De-obfuscate JavaScript.

As we can see the JavaScript is not obfuscated, there is no need to de-obfuscate.

```
Activities Terminal Oct 6 09:30
remnux@remnux: /media/sf_shared_drive/Digital

obj 9 0
Type: /Action
Referencing:
^M<</S/JavaScript/JS(this.exportDataObject({ cName: "template", nLaunch: 0 }));)/Type/Action>>

<<
/S /JavaScript
/JS (this.exportDataObject({ cName: "template", nLaunch: 0 }));)
/Type /Action
>>

^M<</S/JavaScript/JS(this.exportDataObject({ cName: "template", nLaunch: 0 }));)/Type/Action>>
^M<</S/JavaScript/JS(this.exportDataObject({ cName: "template", nLaunch: 0 }));)/Type/Action>>

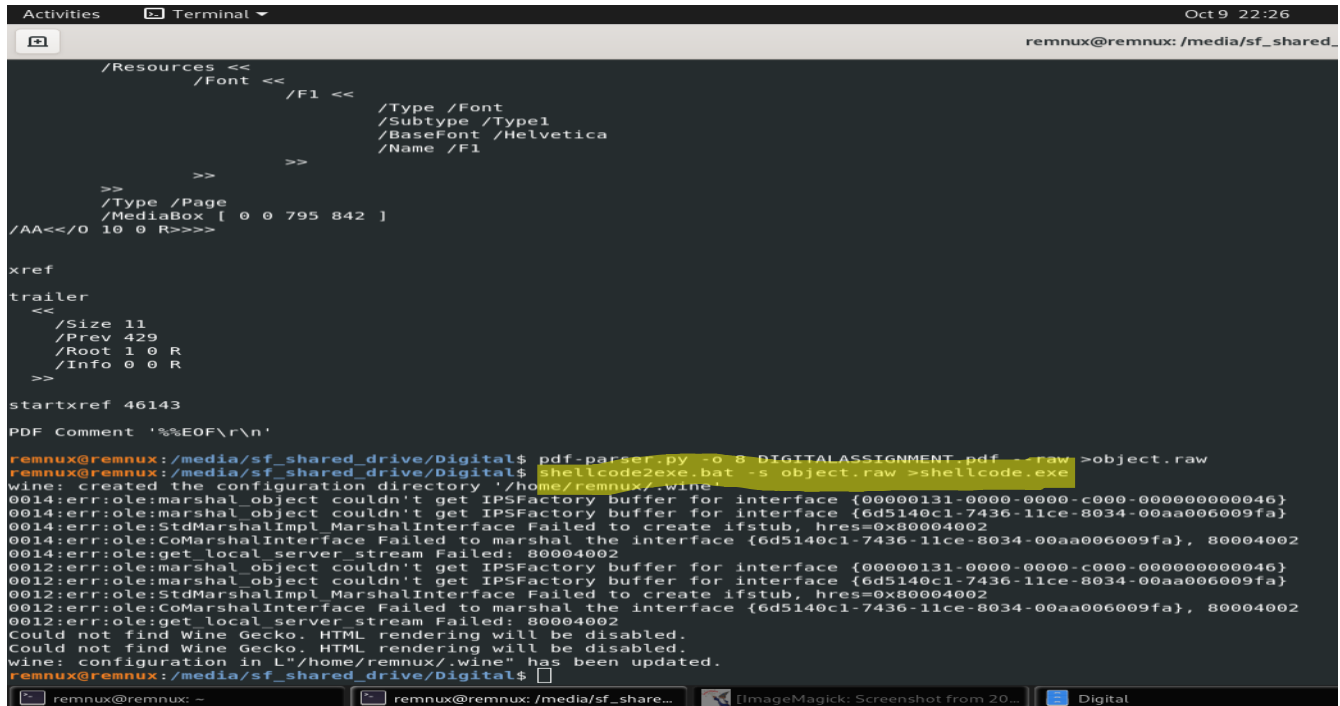
~
~
~
~
```

[illegible]



## 7. Create a shell code executable

To create a shell code executable, we used the command `shellcode2exe.bat` so that the contents of the object 8 are converted to the `shellcode.exe` which will be saved under the same path as the malicious file.



```
Activities Terminal Oct 9 22:26 remnux@remnux: /media/sf_shared_

/Resources <<
/Font <<
  /F1 <<
    /Type /Font
    /Subtype /Type1
    /BaseFont /Helvetica
    /Name /F1
  >>
>>
  /Type /Page
  /MediaBox [ 0 0 795 842 ]
/AA<</O
10 0 R>>>>

xref
trailer
  <<
    /Size 11
    /Prev 429
    /Root 1 0 R
    /Info 0 0 R
  >>
startxref 46143
PDF Comment '%%EOF\r\n'

remnux@remnux: /media/sf_shared_drive/Digital$ pdf-parser.py -o 8 DIGITALASSIGNMENT.pdf -raw >object.raw
remnux@remnux: /media/sf_shared_drive/Digital$ shellcode2exe.bat -s object.raw >shellcode.exe
wine: created the configuration directory '/home/remnux/.wine'
0014:err:ole:marshal object couldn't get IPSFactory buffer for interface {00000131-0000-0000-c000-000000000046}
0014:err:ole:marshal object couldn't get IPSFactory buffer for interface {6d5140c1-7436-11ce-8034-00aa006009fa}
0014:err:ole:StdMarshalImpl MarshalInterface Failed to create ifstub, hres=0x80004002
0014:err:ole:CoMarshalInterface Failed to marshal the interface {6d5140c1-7436-11ce-8034-00aa006009fa}, 80004002
0014:err:ole:get local server stream Failed: 80004002
0012:err:ole:marshal object couldn't get IPSFactory buffer for interface {00000131-0000-0000-c000-000000000046}
0012:err:ole:marshal object couldn't get IPSFactory buffer for interface {6d5140c1-7436-11ce-8034-00aa006009fa}
0012:err:ole:StdMarshalImpl MarshalInterface Failed to create ifstub, hres=0x80004002
0012:err:ole:CoMarshalInterface Failed to marshal the interface {6d5140c1-7436-11ce-8034-00aa006009fa}, 80004002
0012:err:ole:get local server stream Failed: 80004002
Could not find Wine Gecko. HTML rendering will be disabled.
Could not find Wine Gecko. HTML rendering will be disabled.
wine: configuration in L"/home/remnux/.wine" has been updated.
remnux@remnux: /media/sf_shared_drive/Digital$
```

## 8. Analyze shell code and determine what it does or even execute it using sctest or spider monkey.

When the dump is created for the obj-8 we see that the file is relatively small compared to the file is uncompressed as in the below image.

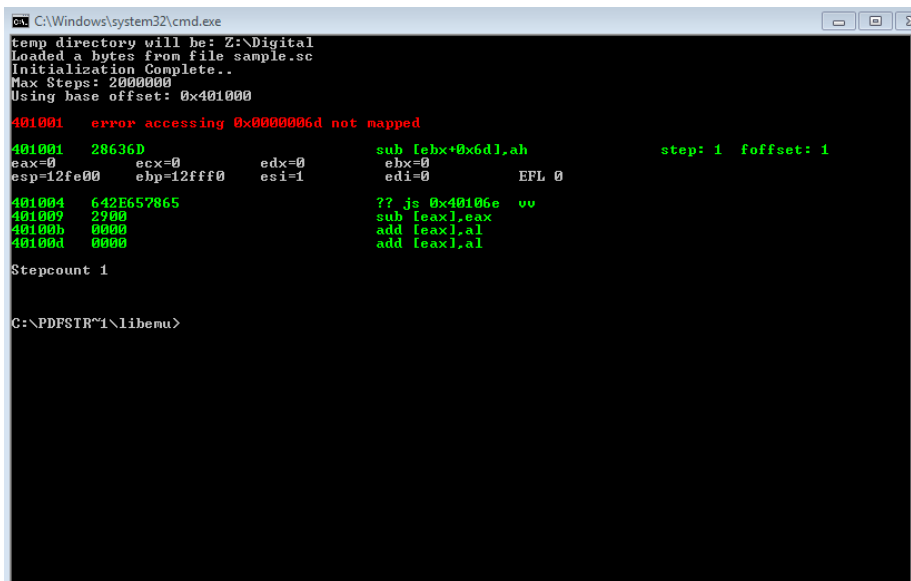
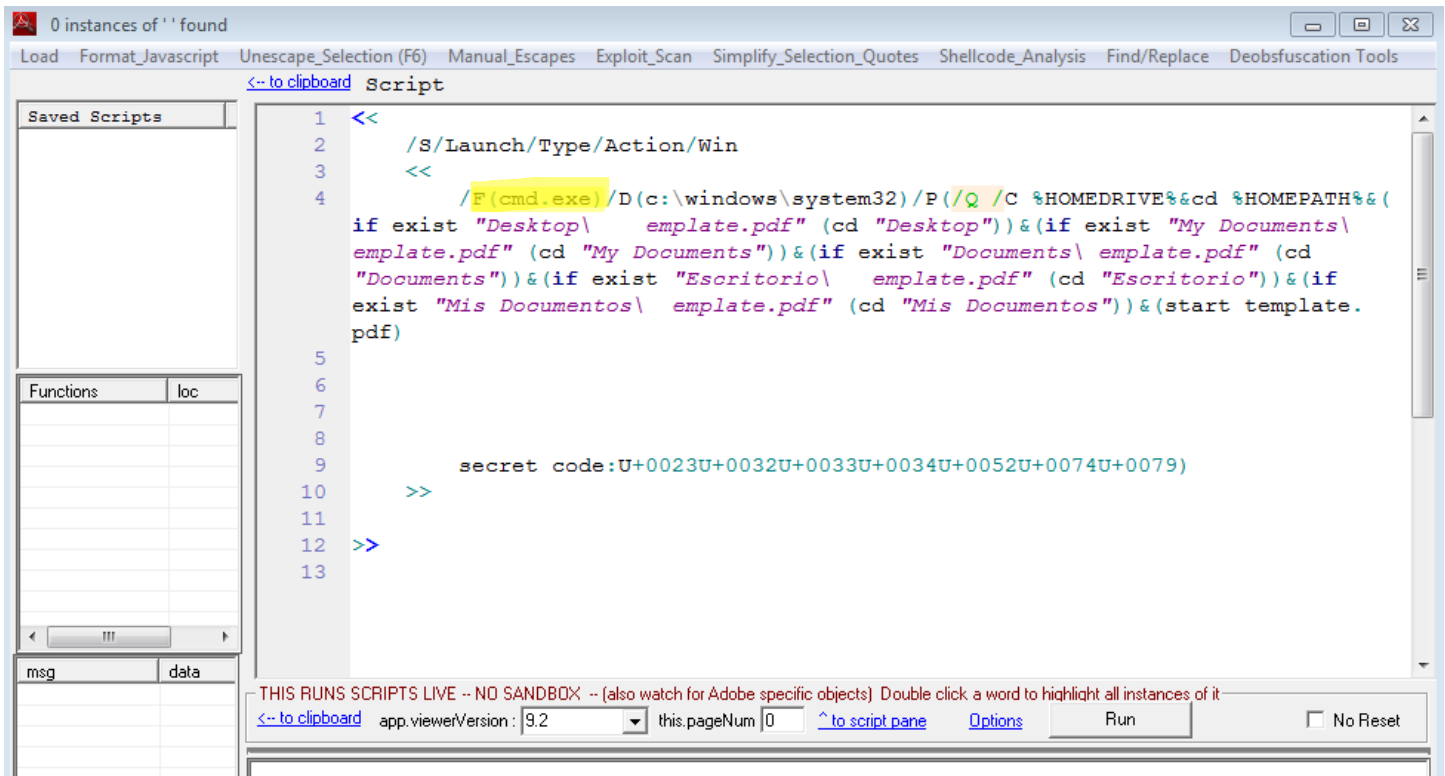
```
Activities Terminal Oct 13 20:27
remnux@remnux:/media/sf_shared_drive/Digital$ pdf-parser.py obj8 DIGITALASSIGNMENT.pdf>OBJ8.UN.dump
remnux@remnux:/media/sf_shared_drive/Digital$ pdf-parser.py -o 8 -f -d obj8.dump DIGITALASSIGNMENT.pdf
obj 8 0
Type:
Referencing:
Contains stream

remnux@remnux:/media/sf_shared_drive/Digital$ ls -l
total 133
drwxrwx--- 1 root vboxsf 4096 Oct 13 20:25 DIGITALASSIGNMENT.dump
-rwxrwx--- 1 root vboxsf 46403 Sep 29 00:25 DIGITALASSIGNMENT.pdf
-rwxrwx--- 1 root vboxsf 42 Oct 6 12:05 hex.txt
-rwxrwx--- 1 root vboxsf 73802 Oct 13 20:26 obj8.dump
-rwxrwx--- 1 root vboxsf 2987 Oct 13 20:26 OBJ8.UN.dump
remnux@remnux:/media/sf_shared_drive/Digital$
```

There is no need to analyze the shell code using sctest or spider monkey because we can clearly see that the shell code that is embedded in the payload is not compressed or obfuscated. We can see that cmd.exe is the code is available in the Obj-9 that is getting executed while opening the malicious file.

When we scan for the exploits, it shows that the possible exploit is at Object 10.

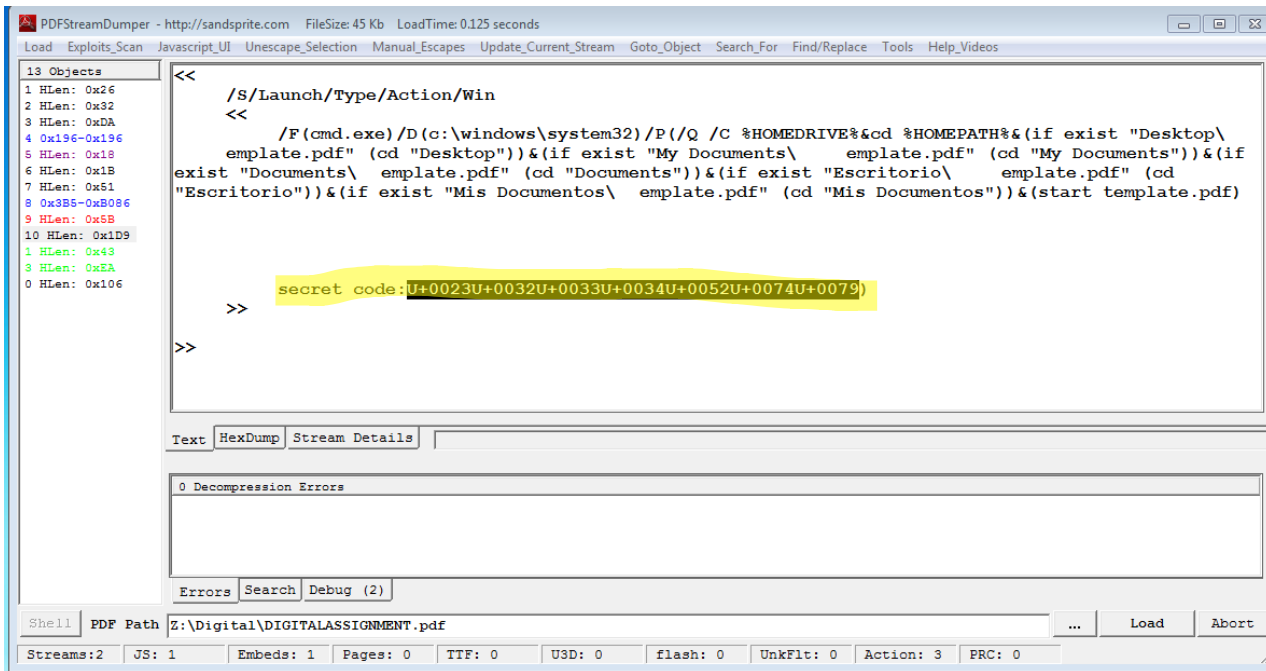
```
1300507210.txt - Notepad
File Edit Format View Help
Exploit Header contains a Launch Action - possible CVE-2010-1240 Date:6.29.10 v9.3.2 - */Launch*/Action* - found in stream: 10
Note other exploits may be hidden with javascript obfuscation
It is also possible these functions are being used in a non-exploit way.
```



This shows No-result.

## 9. What is the secret code?

The extracted secret code is **#234Rty**. The secret code is stored in hexadecimal format, so when we convert the Hex to utf-8 we can get the secret key.



## hexadecimal



U+0023U+0032U+0033U+0034U+0052U+0074U+0079

Import from file

Save as...

Copy to clipboard

## utf8

#234Rty

Chain with...

Save as...

Copy to clipboard

