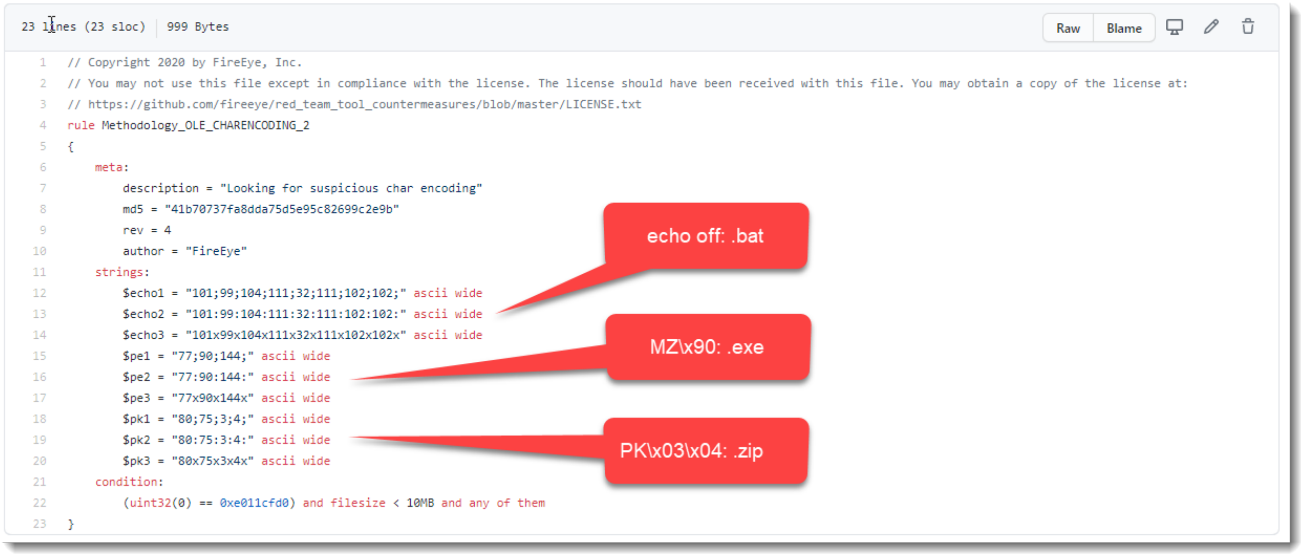
SANS ISC: InfoSec Handlers Diary Blog - SANS Internet Storm Center SANS Site Network Current Site SANS Internet Storm Center Other SANS Sites Help Graduate Degree Programs Security Training Security Certification Security Awareness Training Penetration Testing Industrial Control Systems Cyber Defense Foundations DFIR Software Security Government OnSite Training InfoSec Handlers Diary Blog

isc.sans.edu/diary/26882

Analyzing FireEye Maldocs

Published: 2020-12-15
Last Updated: 2020-12-15 07:16:52 UTC
by Didier Stevens (Version: 1)
0 comment(s)
When FireEye released YARA rules to detect their stolen red team tools, I was interested in their maldoc rules:



23 lines (23 sloc) | 999 Bytes

```
// Copyright 2020 by FireEye, Inc.
// You may not use this file except in compliance with the license. The license should have been received with this file. You may obtain a copy of the license at:
// https://github.com/fireeye/red_team_tool_countermeasures/blob/master/LICENSE.txt
rule Methodology_OLE_CHARENCODING_2
{
    meta:
        description = "Looking for suspicious char encoding"
        md5 = "41b70737fa8dda75d5e95c82699c2e9b"
        rev = 4
        author = "FireEye"
    strings:
        $echo1 = "101;99;104;111;32;111;102;102;" ascii wide
        $echo2 = "101:99:104:111:32:111:102:102:" ascii wide
        $echo3 = "101x99x104x111x32x111x102x102x" ascii wide
        $pe1 = "77;90;144;" ascii wide
        $pe2 = "77:90:144:" ascii wide
        $pe3 = "77x90x144x" ascii wide
        $pk1 = "80;75;3;4;" ascii wide
        $pk2 = "80:75:3:4:" ascii wide
        $pk3 = "80x75x3x4x" ascii wide
    condition:
        (uint32(0) == 0xe011cfd0) and filesize < 10MB and any of them
}
```

echo off: .bat

MZ\x90: .exe

PK\x03\x04: .zip

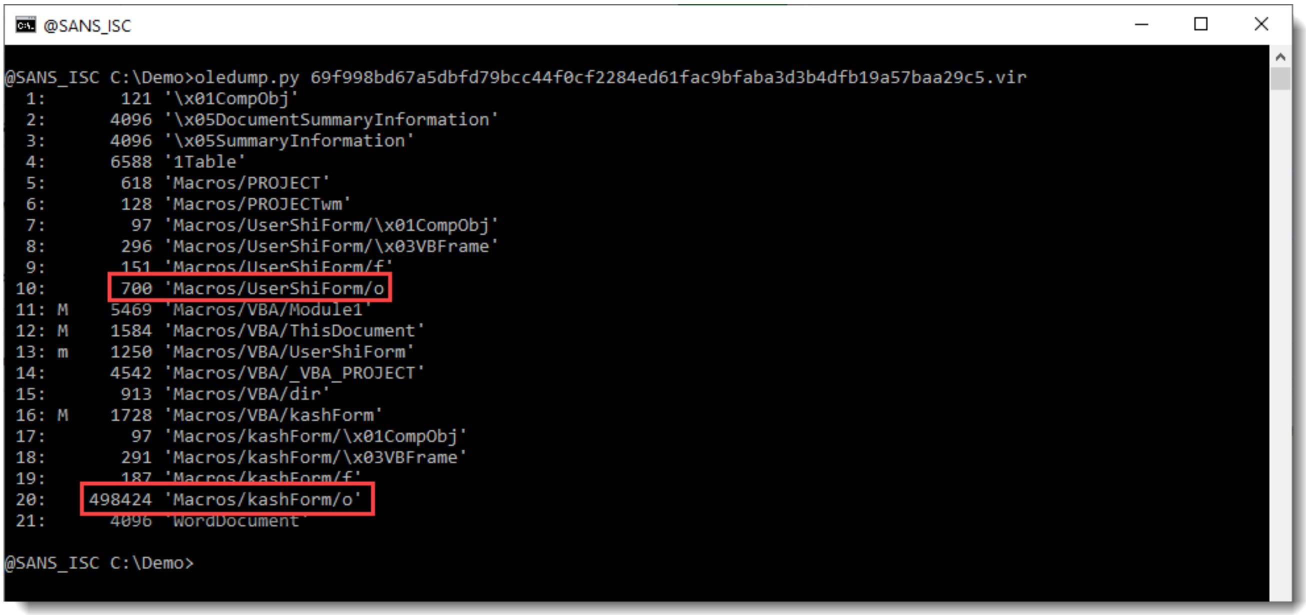
This rule here (Methodology_OLE_CHARENCODING_2) detects OLE files (.doc, .xls, ...) that contains sequences of decimal numbers. Converted to ASCII, these numbers reveal short strings: "echo off", "MZ", "PK".

That indicates to me that maldocs created with FireEye's tool embed a .BAT file, a .EXE and/or a .ZIP file.

The maldoc sample mentioned in the rule is available on VirusTotal: MD5 41b70737fa8dda75d5e95c82699c2e9b.

I analyze this maldoc as follows:

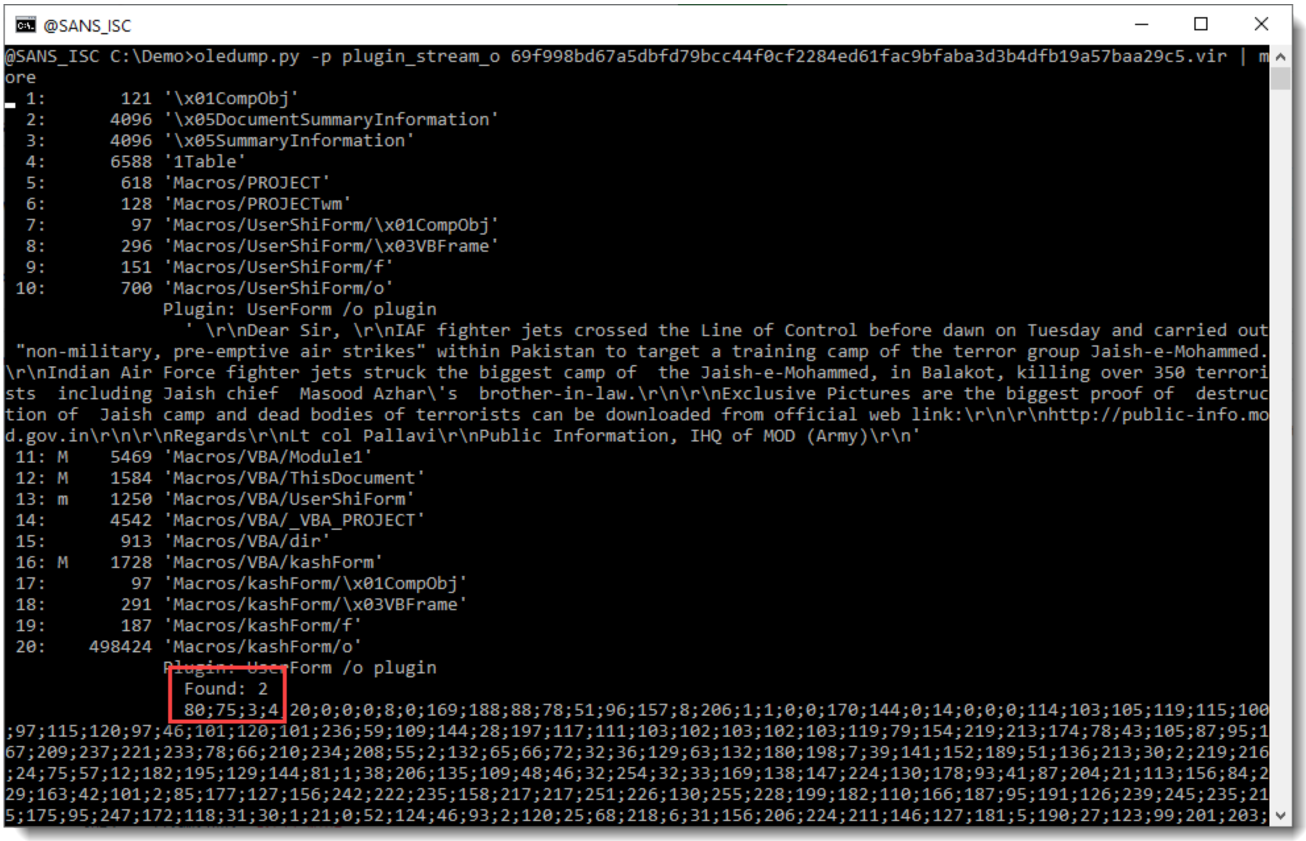
First I run my oledump tool:



```
@SANS_ISC C:\Demo>oledump.py 69f998bd67a5dbfd79bcc44f0cf2284ed61fac9bfaba3d3b4dfb19a57baa29c5.vir
  1:       121 '\x01CompObj'
  2:      4096 '\x05DocumentSummaryInformation'
  3:      4096 '\x05SummaryInformation'
  4:      6588 '1Table'
  5:       618 'Macros/PROJECT'
  6:       128 'Macros/PROJECTwm'
  7:        97 'Macros/UserShiForm/\x01CompObj'
  8:       296 'Macros/UserShiForm/\x03VBFrame'
  9:       151 'Macros/UserShiForm/f'
 10:       700 'Macros/UserShiForm/o
 11: M    5469 'Macros/VBA/Module1'
 12: M    1584 'Macros/VBA/ThisDocument'
 13: m    1250 'Macros/VBA/UserShiForm'
 14:      4542 'Macros/VBA/_VBA_PROJECT'
 15:       913 'Macros/VBA/dir'
 16: M    1728 'Macros/VBA/kashForm'
 17:        97 'Macros/kashForm/\x01CompObj'
 18:       291 'Macros/kashForm/\x03VBFrame'
 19:       187 'Macros/kashForm/f'
 20:    498424 'Macros/kashForm/o'
 21:      4096 'WordDocument'

@SANS_ISC C:\Demo>
```

The macro indicators (M and m) tell me that there is VBA code in this maldoc. But my attention is first drawn to the streams that end with /o (stream 10 and 20). Hiding payloads, scripts, ... inside VBA user form values is a well-known technique used by malware authors. I have a plugin to help with the analysis of maldocs that use this technique: plugin_stream_o.

This is the command:



```
@SANS_ISC C:\Demo>oledump.py -p plugin_stream_o 69f998bd67a5dbfd79bcc44f0cf2284ed61fac9bfaba3d3b4dfb19a57baa29c5.vir | m
ore
  1:       121 '\x01CompObj'
  2:      4096 '\x05DocumentSummaryInformation'
  3:      4096 '\x05SummaryInformation'
  4:      6588 '1Table'
  5:       618 'Macros/PROJECT'
  6:       128 'Macros/PROJECTwm'
  7:        97 'Macros/UserShiForm/\x01CompObj'
  8:       296 'Macros/UserShiForm/\x03VBFrame'
  9:       151 'Macros/UserShiForm/f'
 10:       700 'Macros/UserShiForm/o'
               Plugin: UserForm /o plugin
                 ' \r\nDear Sir, \r\nIAF fighter jets crossed the Line of Control before dawn on Tuesday and carried out
 "non-military, pre-emptive air strikes" within Pakistan to target a training camp of the terror group Jaish-e-Mohammed.
\r\nIndian Air Force fighter jets struck the biggest camp of  the Jaish-e-Mohammed, in Balakot, killing over 350 terrori
sts  including Jaish chief  Masood Azhar\'s  brother-in-law.\r\n\r\nExclusive Pictures are the biggest proof of  destruc
tion of  Jaish camp and dead bodies of terrorists can be downloaded from official web link:\r\n\r\nhttp://public-info.mo
d.gov.in\r\n\r\nRegards\r\nLt col Pallavi\r\nPublic Information, IHQ of MOD (Army)\r\n'
 11: M    5469 'Macros/VBA/Module1'
 12: M    1584 'Macros/VBA/ThisDocument'
 13: m    1250 'Macros/VBA/UserShiForm'
 14:      4542 'Macros/VBA/_VBA_PROJECT'
 15:       913 'Macros/VBA/dir'
 16: M    1728 'Macros/VBA/kashForm'
 17:        97 'Macros/kashForm/\x01CompObj'
 18:       291 'Macros/kashForm/\x03VBFrame'
 19:       187 'Macros/kashForm/f'
 20:    498424 'Macros/kashForm/o'
               Plugin: UserForm /o plugin
                 Found: 2
                 80;75;3;4 20;0;0;0;8;0;169;188;88;78;51;96;157;8;206;1;1;0;0;170;144;0;14;0;0;0;114;103;105;119;115;100
;97;115;120;97;46;101;120;101;236;59;109;144;28;197;117;111;103;102;103;102;103;119;79;154;219;213;174;78;43;105;87;95;1
67;209;237;221;233;78;66;210;234;208;55;2;132;65;66;72;32;36;129;63;132;180;198;7;39;141;152;189;51;136;213;30;2;219;216
;24;75;57;12;182;195;129;144;81;1;38;206;135;109;48;46;32;254;32;33;169;138;147;224;130;178;93;41;87;204;21;113;156;84;2
29;163;42;101;2;85;177;127;156;242;222;235;158;217;217;251;226;130;255;228;199;182;110;166;187;95;191;126;239;245;235;21
5;175;95;247;172;118;31;30;1;21;0;52;124;46;93;2;120;25;68;218;6;31;156;206;224;211;146;127;181;5;190;27;123;99;201;203;
```

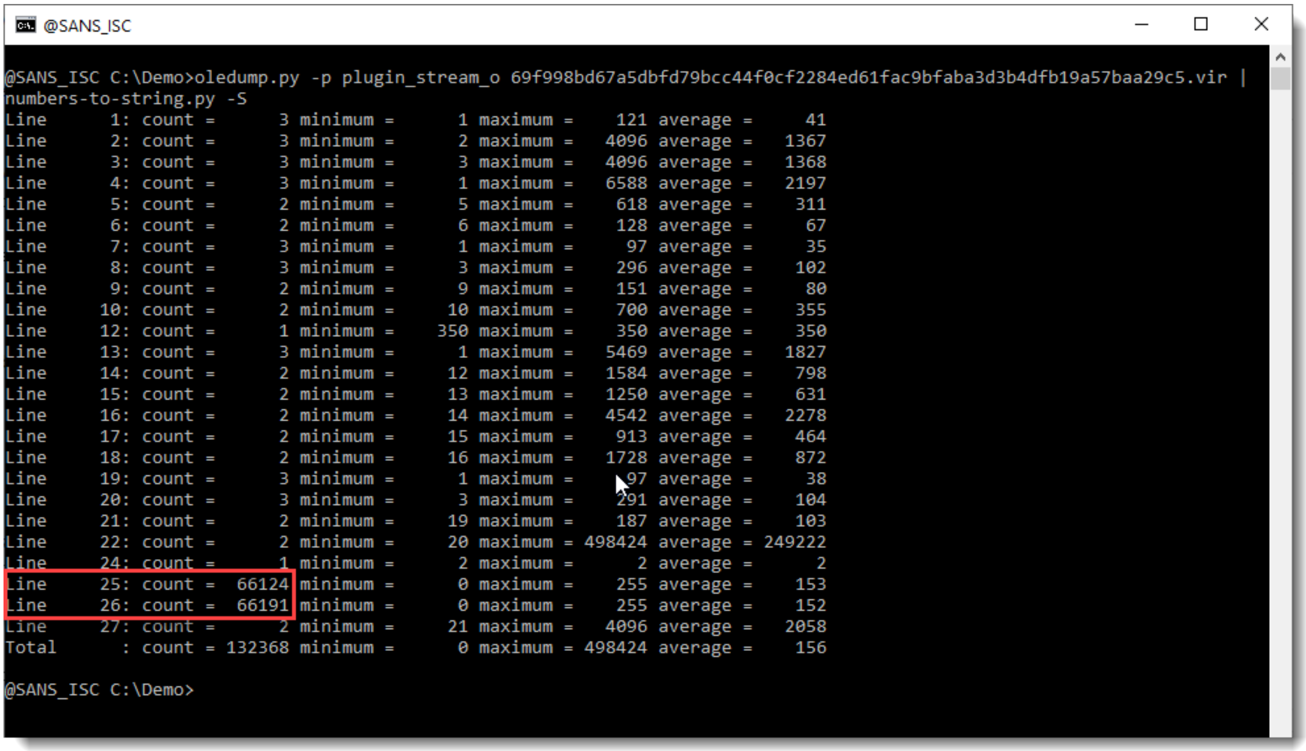
So stream 10 contains a value that looks like a message to be displayed by this maldoc.

And stream 20 contains the payload we are looking for: a long sequence of decimal numbers. It starts with 80;75;3;4: that's the YARA rule's detection string for a ZIP record.

Remark also the "Found: 2" message from the plugin: this is new since the last version. This means there are 2 values inside this stream (if there is only one value, this Found message is not displayed, just like older versions of the plugin do).

The next step now is to convert this sequence of decimal numbers to bytes. I have a tool for that: numbers-to-string.py.

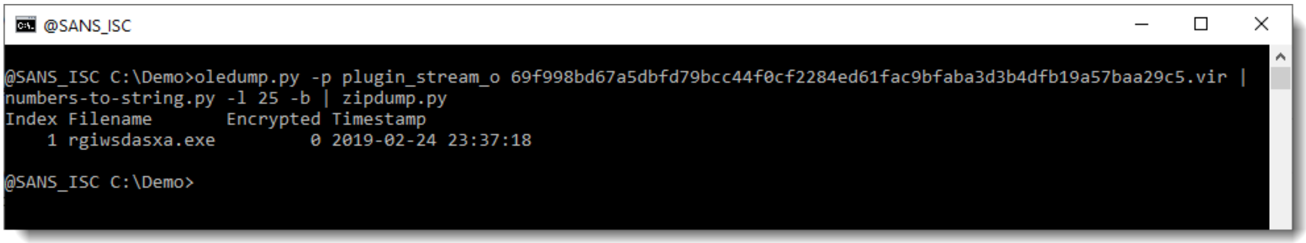
Since there are 2 values inside stream 20, I want to take a closer look first. I use option -S of numbers-to-string.py to produce statistics for each line of text with numbers:



```
@SANS_ISC C:\Demo>oledump.py -p plugin_stream_o 69f998bd67a5dbfd79bcc44f0cf2284ed61fac9bfaba3d3b4dfb19a57baa29c5.vir |
numbers-to-string.py -S
Line      1: count =      3 minimum =      1 maximum =    121 average =     41
Line      2: count =      3 minimum =      2 maximum =   4096 average =   1367
Line      3: count =      3 minimum =      3 maximum =   4096 average =   1368
Line      4: count =      3 minimum =      1 maximum =   6588 average =   2197
Line      5: count =      2 minimum =      5 maximum =    618 average =    311
Line      6: count =      2 minimum =      6 maximum =    128 average =     67
Line      7: count =      3 minimum =      1 maximum =     97 average =     35
Line      8: count =      3 minimum =      3 maximum =    296 average =    102
Line      9: count =      2 minimum =      9 maximum =    151 average =     80
Line     10: count =      2 minimum =     10 maximum =    700 average =    355
Line     12: count =      1 minimum =    350 maximum =    350 average =    350
Line     13: count =      3 minimum =      1 maximum =   5469 average =   1827
Line     14: count =      2 minimum =     12 maximum =   1584 average =    798
Line     15: count =      2 minimum =     13 maximum =   1250 average =    631
Line     16: count =      2 minimum =     14 maximum =   4542 average =   2278
Line     17: count =      2 minimum =     15 maximum =    913 average =    464
Line     18: count =      2 minimum =     16 maximum =   1728 average =    872
Line     19: count =      3 minimum =      1 maximum =     97 average =     38
Line     20: count =      3 minimum =      3 maximum =    291 average =    104
Line     21: count =      2 minimum =     19 maximum =    187 average =    103
Line     22: count =      2 minimum =     20 maximum = 498424 average = 249222
Line     24: count =      1 minimum =      2 maximum =      2 average =      2
Line     25: count =  66124 minimum =      0 maximum =    255 average =    153
Line     26: count =  66191 minimum =      0 maximum =    255 average =    152
Line     27: count =      2 minimum =     21 maximum =   4096 average =   2058
Total      : count = 132368 minimum =      0 maximum = 498424 average =    156

@SANS_ISC C:\Demo>
```

So there are 2 values inside stream 20 that are long sequences of decimal numbers. Line 25: 66124 values between 0 and 255, Line 26: 66191 values between 0 and 255. So it looks like we have 2 embedded files in here, probably 2 ZIP files.

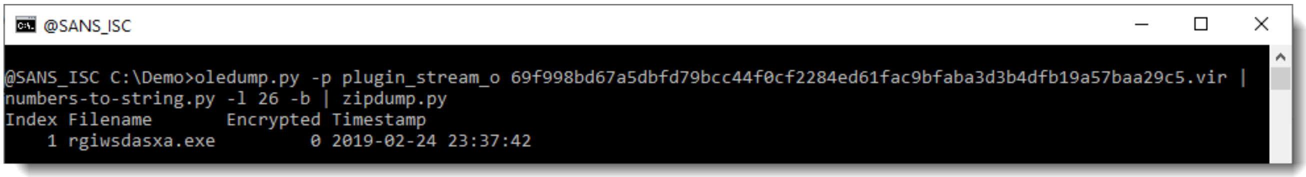
I select the first value (line 25), decode it as binary data (-b) and analyze it with my tool zipdump.py.



```
@SANS_ISC C:\Demo>oledump.py -p plugin_stream_o 69f998bd67a5dbfd79bcc44f0cf2284ed61fac9bfaba3d3b4dfb19a57baa29c5.vir |
numbers-to-string.py -l 25 -b | zipdump.py
Index Filename       Encrypted Timestamp
    1 rgiwsdasxa.exe         0 2019-02-24 23:37:18

@SANS_ISC C:\Demo>
```

So that is indeed a ZIP file, and it contains a .exe file.

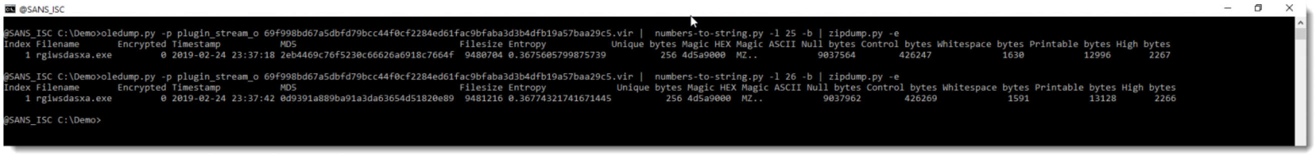
I do a quick check to see if the second value (line 26) also decodes to a ZIP file:



```
@SANS_ISC C:\Demo>oledump.py -p plugin_stream_o 69f998bd67a5dbfd79bcc44f0cf2284ed61fac9bfaba3d3b4dfb19a57baa29c5.vir |
numbers-to-string.py -l 26 -b | zipdump.py
Index Filename       Encrypted Timestamp
    1 rgiwsdasxa.exe         0 2019-02-24 23:37:42
```

And indeed, that one too is a .exe file.

With zipdump's option -e I get extra info, like the hash to look the file up on VirusTotal:



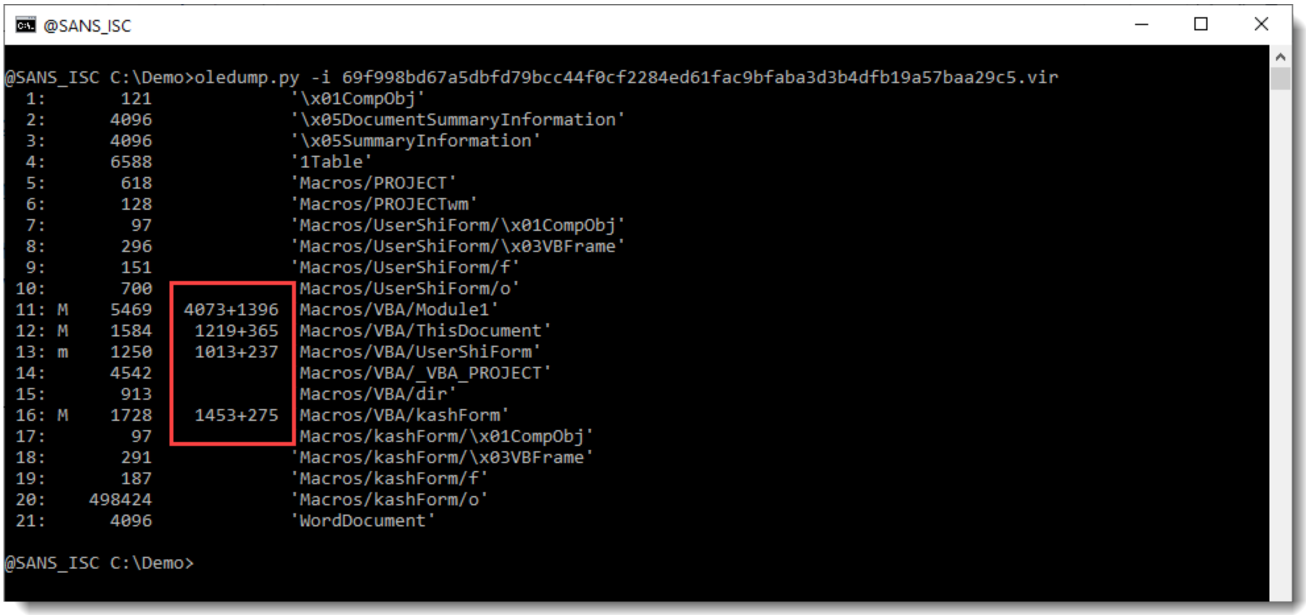
```
@SANS_ISC C:\Demo>oledump.py -p plugin_stream_o 69f998bd67a5dbfd79bcc44f0cf2284ed61fac9bfaba3d3b4dfb19a57baa29c5.vir |  numbers-to-string.py -l 25 -b | zipdump.py -e
Index Filename       Encrypted Timestamp           MD5                              Filesize Entropy            Unique bytes Magic HEX Magic ASCII Null bytes Control bytes Whitespace bytes Printable bytes High bytes
    1 rgiwsdasxa.exe         0 2019-02-24 23:37:18 2eb4469c76f5230c66626a6918c7664f  9480704 0.3675605799875739          256 4d5a9000  MZ..           9037564       426247             1630           12996       2267

@SANS_ISC C:\Demo>oledump.py -p plugin_stream_o 69f998bd67a5dbfd79bcc44f0cf2284ed61fac9bfaba3d3b4dfb19a57baa29c5.vir |  numbers-to-string.py -l 26 -b | zipdump.py -e
Index Filename       Encrypted Timestamp           MD5                              Filesize Entropy             Unique bytes Magic HEX Magic ASCII Null bytes Control bytes Whitespace bytes Printable bytes High bytes
    1 rgiwsdasxa.exe         0 2019-02-24 23:37:42 0d9391a889ba91a3da63654d51820e89  9481216 0.36774321741671445          256 4d5a9000  MZ..           9037962       426269             1591           13128       2266

@SANS_ISC C:\Demo>
```

Here are the samples: 2eb4469c76f5230c66626a6918c7664f and 0d9391a889ba91a3da63654d51820e89.

So this FireEye maldoc is not hard to analyze.

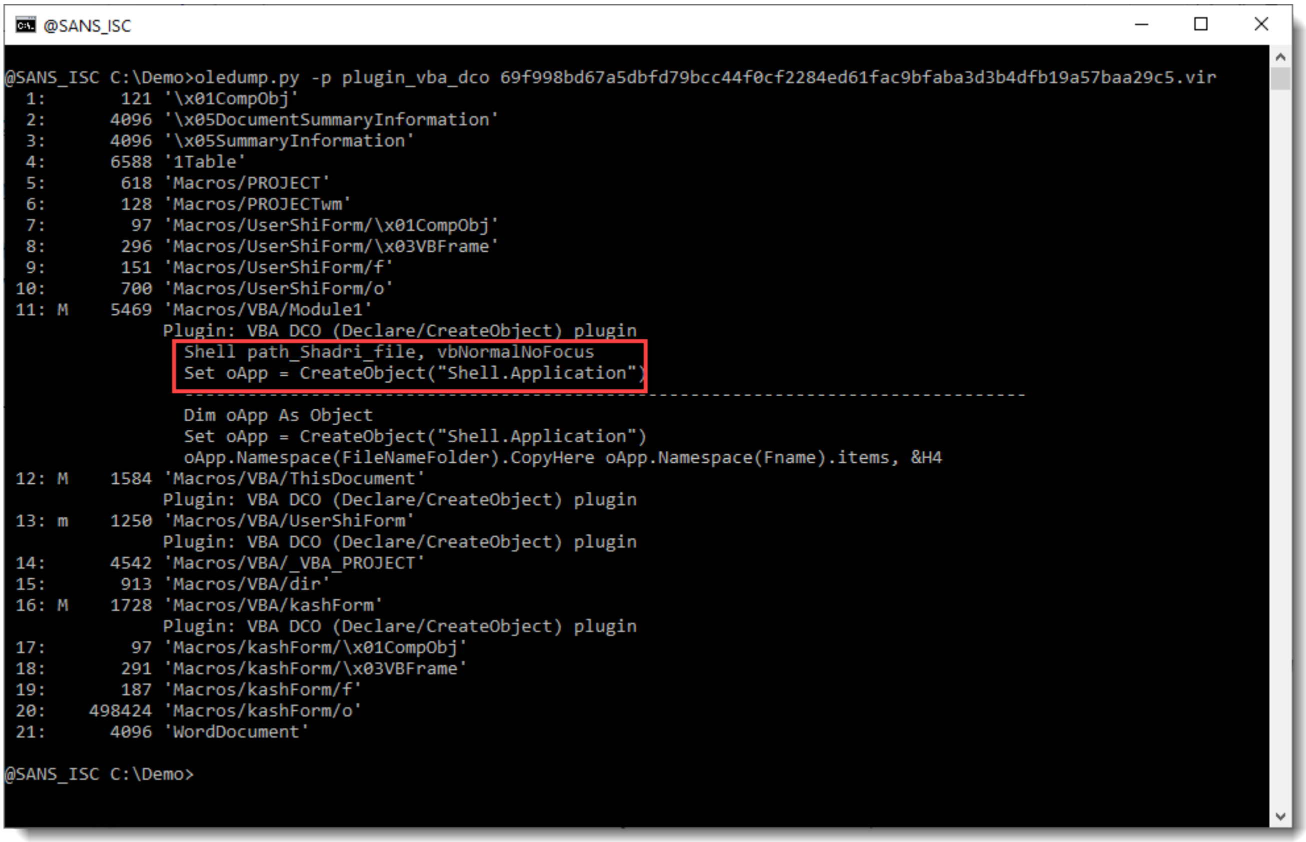
Remark that in the YARA rule, there are strings with separator : and x beside ;. It looks like there can be variations in the encoding, but that has no effect on the decoding of the decimal numbers by my tool.

I also checked if VBA stomping or purging was performed on this maldoc, but that doesn't seem to be the case:



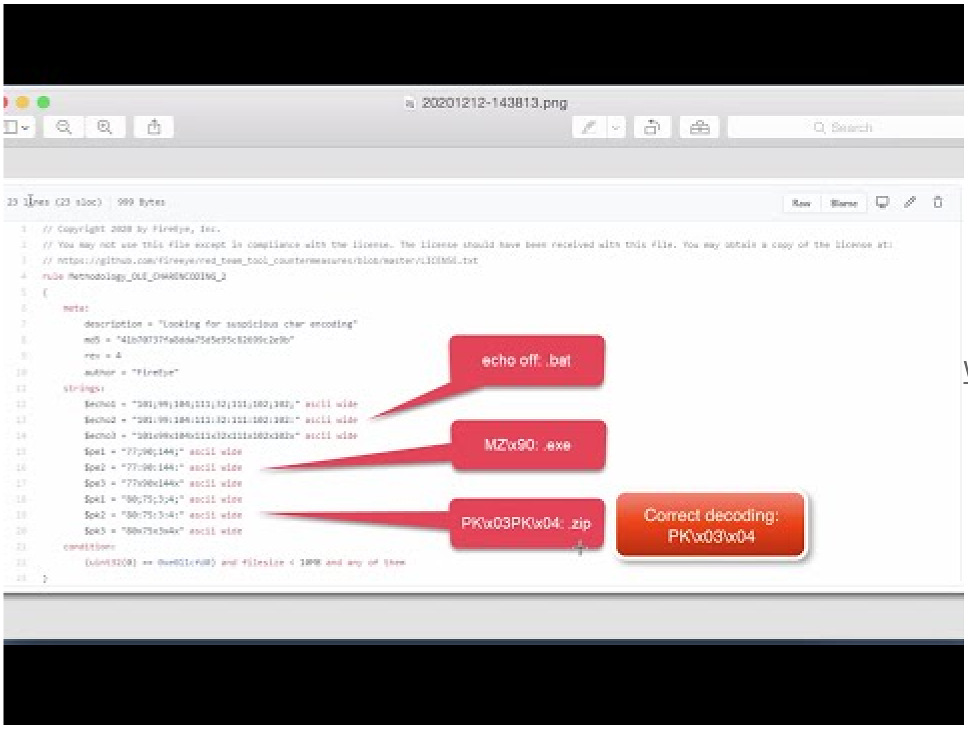
```
@SANS_ISC C:\Demo>oledump.py -i 69f998bd67a5dbfd79bcc44f0cf2284ed61fac9bfaba3d3b4dfb19a57baa29c5.vir
  1:       121             '\x01CompObj'
  2:      4096             '\x05DocumentSummaryInformation'
  3:      4096             '\x05SummaryInformation'
  4:      6588             '1Table'
  5:       618             'Macros/PROJECT'
  6:       128             'Macros/PROJECTwm'
  7:        97             'Macros/UserShiForm/\x01CompObj'
  8:       296             'Macros/UserShiForm/\x03VBFrame'
  9:       151             'Macros/UserShiForm/f'
 10:       700              Macros/UserShiForm/o'
 11: M    5469   4073+1396  Macros/VBA/Module1'
 12: M    1584    1219+365  Macros/VBA/ThisDocument'
 13: m    1250    1013+237  Macros/VBA/UserShiForm'
 14:      4542              Macros/VBA/_VBA_PROJECT'
 15:       913              Macros/VBA/dir'
 16: M    1728    1453+275  Macros/VBA/kashForm'
 17:        97              Macros/kashForm/\x01CompObj'
 18:       291             'Macros/kashForm/\x03VBFrame'
 19:       187             'Macros/kashForm/f'
 20:    498424             'Macros/kashForm/o'
 21:      4096             'WordDocument'

@SANS_ISC C:\Demo>
```

There is compiled code and VBA code inside the module streams. So the compiled VBA code has not been purged, and neither has the source code been stomped, since I can find VBA source code with Shell statements and CreateObject calls:



```
@SANS_ISC C:\Demo>oledump.py -p plugin_vba_dco 69f998bd67a5dbfd79bcc44f0cf2284ed61fac9bfaba3d3b4dfb19a57baa29c5.vir
  1:       121 '\x01CompObj'
  2:      4096 '\x05DocumentSummaryInformation'
  3:      4096 '\x05SummaryInformation'
  4:      6588 '1Table'
  5:       618 'Macros/PROJECT'
  6:       128 'Macros/PROJECTwm'
  7:        97 'Macros/UserShiForm/\x01CompObj'
  8:       296 'Macros/UserShiForm/\x03VBFrame'
  9:       151 'Macros/UserShiForm/f'
 10:       700 'Macros/UserShiForm/o'
 11: M    5469 'Macros/VBA/Module1'
               Plugin: VBA DCO (Declare/CreateObject) plugin
                 Shell path_Shadri_file, vbNormalNoFocus
                 Set oApp = CreateObject("Shell.Application")
                 ---------------------------------------------------------------------------------
                 Dim oApp As Object
                 Set oApp = CreateObject("Shell.Application")
                 oApp.Namespace(FileNameFolder).CopyHere oApp.Namespace(Fname).items, &H4
 12: M    1584 'Macros/VBA/ThisDocument'
               Plugin: VBA DCO (Declare/CreateObject) plugin
 13: m    1250 'Macros/VBA/UserShiForm'
               Plugin: VBA DCO (Declare/CreateObject) plugin
 14:      4542 'Macros/VBA/_VBA_PROJECT'
 15:       913 'Macros/VBA/dir'
 16: M    1728 'Macros/VBA/kashForm'
               Plugin: VBA DCO (Declare/CreateObject) plugin
 17:        97 'Macros/kashForm/\x01CompObj'
 18:       291 'Macros/kashForm/\x03VBFrame'
 19:       187 'Macros/kashForm/f'
 20:    498424 'Macros/kashForm/o'
 21:      4096 'WordDocument'

@SANS_ISC C:\Demo>
```

I recorded a video of this analysis, where I also take a look at the VBA code:



23 lines (23 sloc) | 999 Bytes

```
// Copyright 2020 by FireEye, Inc.
// You may not use this file except in compliance with the license. The license should have been received with this file. You may obtain a copy of the license at:
// https://github.com/fireeye/red_team_tool_countermeasures/blob/master/LICENSE.txt
rule Methodology_OLE_CHARENCODING_2
{
    meta:
        description = "Looking for suspicious char encoding"
        md5 = "41b70737fa8dda75e5e95c82699c2e9b"
        rev = 4
        author = "FireEye"
    strings:
        $echo1 = "101;99;104;111;32;111;102;102;" ascii wide
        $echo2 = "101:99:104:111:32:111:102:102:" ascii wide
        $echo3 = "101x99x104x111x32x111x102x102x" ascii wide
        $pe1 = "77;90;144;" ascii wide
        $pe2 = "77:90:144:" ascii wide
        $pe3 = "77x90x144x" ascii wide
        $pk1 = "80;75;3;4;" ascii wide
        $pk2 = "80:75:3:4:" ascii wide
        $pk3 = "80x75x3x4x" ascii wide
    condition:
        (uint32(0) == 0xe011cfd0) and filesize < 10MB and any of them
}
```

Watch Video At:

https://youtu.be/VRPNwaWPJiE

Didier Stevens
Senior handler
Microsoft MVP
blog.DidierStevens.com DidierStevensLabs.com

Keywords: fireeye maldoc

0 comment(s)

DEV522
Defending Web Application Security Essentials
LEARN MORE
Learn to defend your apps before they're hacked

Top of page
×

Diary Archives