Needle in a haystack

virusbulletin.com/virusbulletin/2014/02/needle-haystack

2014-02-03

Gabor Szappanos

Sophos, Hungary **Editor:** Helen Martin

Abstract

Sometimes what looks like a genuine MP3 encoder library, and even works as a functional encoder, actually hides malicious code deep amongst a pile of clean code. Gabor Szappanos reveals the lengths to which one piece of malware goes to hide its tracks.

Malware authors engaged in Advanced Persistent Threat (APT) operations put great effort into making sure their creations live up to their name and achieve persistence over the course of months or years; in order to do so, the threats must remain undetected by security products.

The authors try both to conceal the presence of the threats on infected systems and to hide their code from analysis and detection. Most crimeware authors achieve the latter by applying sophisticated execryptors and protectors to their code.

Over the past year, however, we have spotted a different approach: malicious code is compiled into an open source library, hidden among a large pile of clean library code, with only a single export pointing to the trojan functionality. The deployment and progression of this malware spans about two years now – however its versioning suggests that its development started longer ago than that.

This malware doesn't take anything for granted: even common system tools like rundll32.exe and wscript.exe, which are present on all *Windows* systems, are carried with the installer and dropped when needed.

The malware goes to great lengths to cover its tracks. All of the string constants that could reveal the nature of the backdoor are protected with strong encryption. Additionally, the backdoor itself is disguised as a legitimate MP3 encoder library. In fact, it *is* a legitimate and functional MP3 library – and a bit more besides.

Exploited carrier workbook

In a handful of cases we have been able to identify the original exploited document that leads to the system infection. At the time of finalizing this paper, three exploited workbooks have been found that install this threat.

All of them are protected *Excel* workbooks with the default password (for more details see [1]). In short: the workbooks are password protected (that is, checked before opening). It is possible to leave the password field blank – in which case *Excel* encrypts the content using the default password: 'VelvetSweatshop'. On the other hand, if a workbook is protected with exactly this password, *Excel* assumes that there is no password, and opens the document transparently. As a result, the document content is encrypted and hidden from normal analysis, but opening it will execute the shellcode without further prompting.

The workbooks exploit the CVE-2012-0158 vulnerability, which triggers the execution of shellcode within the document.

After the workbooks are opened, the intended operation is to open a decoy workbook – a clean file that grabs the attention of the user while malicious activities proceed in the background. The themes of the decoys give us some idea as to the areas of interest of the target audience of this malware distribution.

Workbook 1

Filename: 300访民联署签名.xls (rough translation: '300 petitioners cosigned.xls')

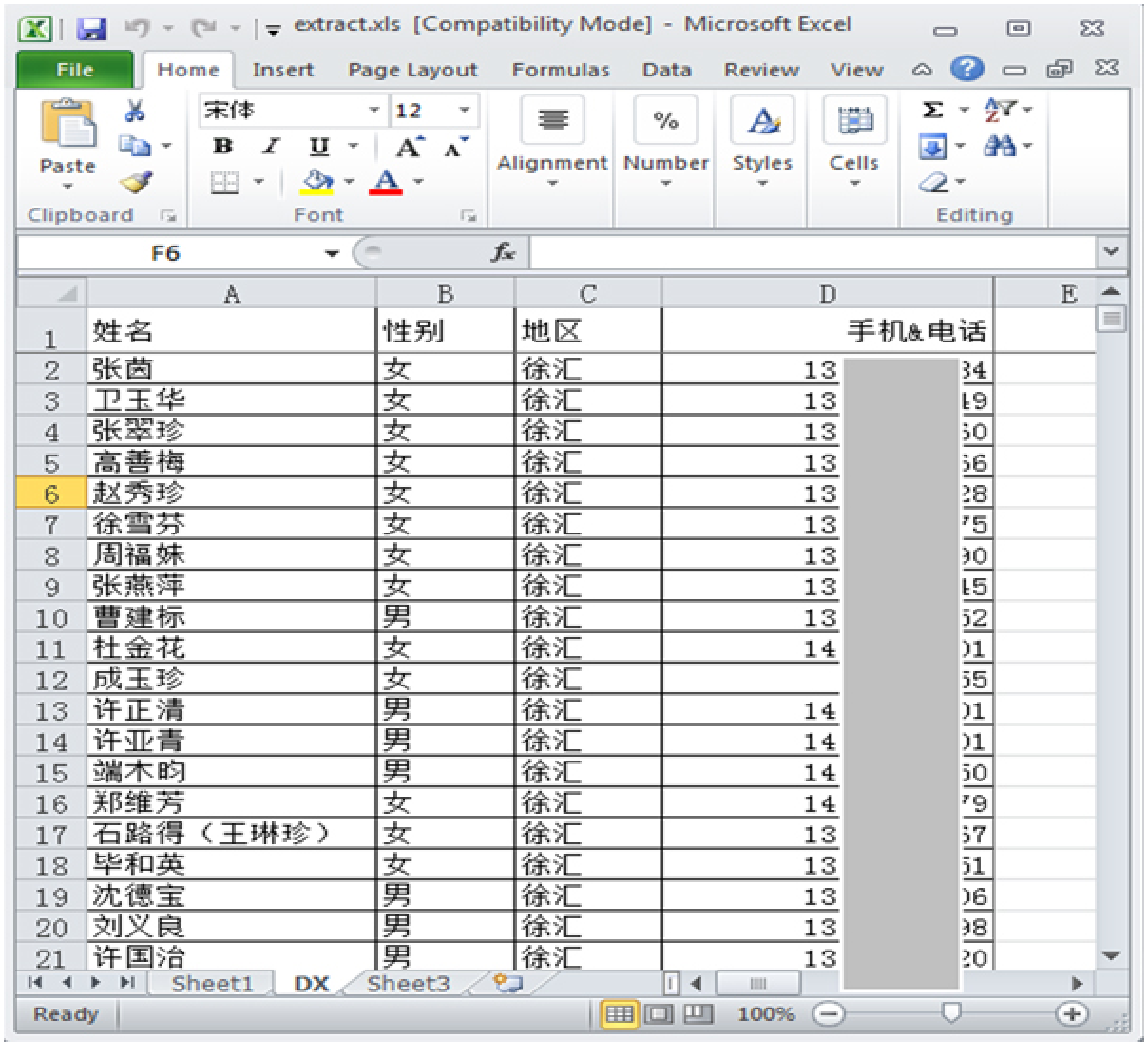
File size: 839756 bytes

SHA1: 066998e20ad44bc5f1ca075a3fb33f1619dd6313

MD5: 5c370923119f66e64a5f9accdd3d5fb

This does not display any decoy document, just closes the *Excel* window. Nevertheless, the shellcode execution proceeds.

If the file was opened, it would display a workbook with a list of names, gender, region and phone numbers of Chinese individuals.



	A	B	C	D
1	姓名	性别	地区	手机&电话
2	张茵	女	徐汇	13[redacted]34
3	卫玉华	女	徐汇	13[redacted]19
4	张翠珍	女	徐汇	13[redacted]50
5	高善梅	女	徐汇	13[redacted]56
6	赵秀珍	女	徐汇	13[redacted]28
7	徐雪芬	女	徐汇	13[redacted]75
8	周福妹	女	徐汇	13[redacted]90
9	张燕萍	女	徐汇	13[redacted]15
10	曹建标	男	徐汇	13[redacted]52
11	杜金花	女	徐汇	14[redacted]01
12	成玉珍	女	徐汇	[redacted]55
13	许正清	男	徐汇	14[redacted]01
14	许亚青	男	徐汇	14[redacted]01
15	端木昀	男	徐汇	14[redacted]50
16	郑维芳	女	徐汇	14[redacted]79
17	石路得（王琳珍）	女	徐汇	13[redacted]57
18	毕和英	女	徐汇	13[redacted]51
19	沈德宝	男	徐汇	13[redacted]06
20	刘乂良	男	徐汇	13[redacted]98
21	许国治	男	徐汇	13[redacted]20

Figure 1. Decoy content for 066998e20ad44bc5f1ca075a3fb33f1619dd6313.

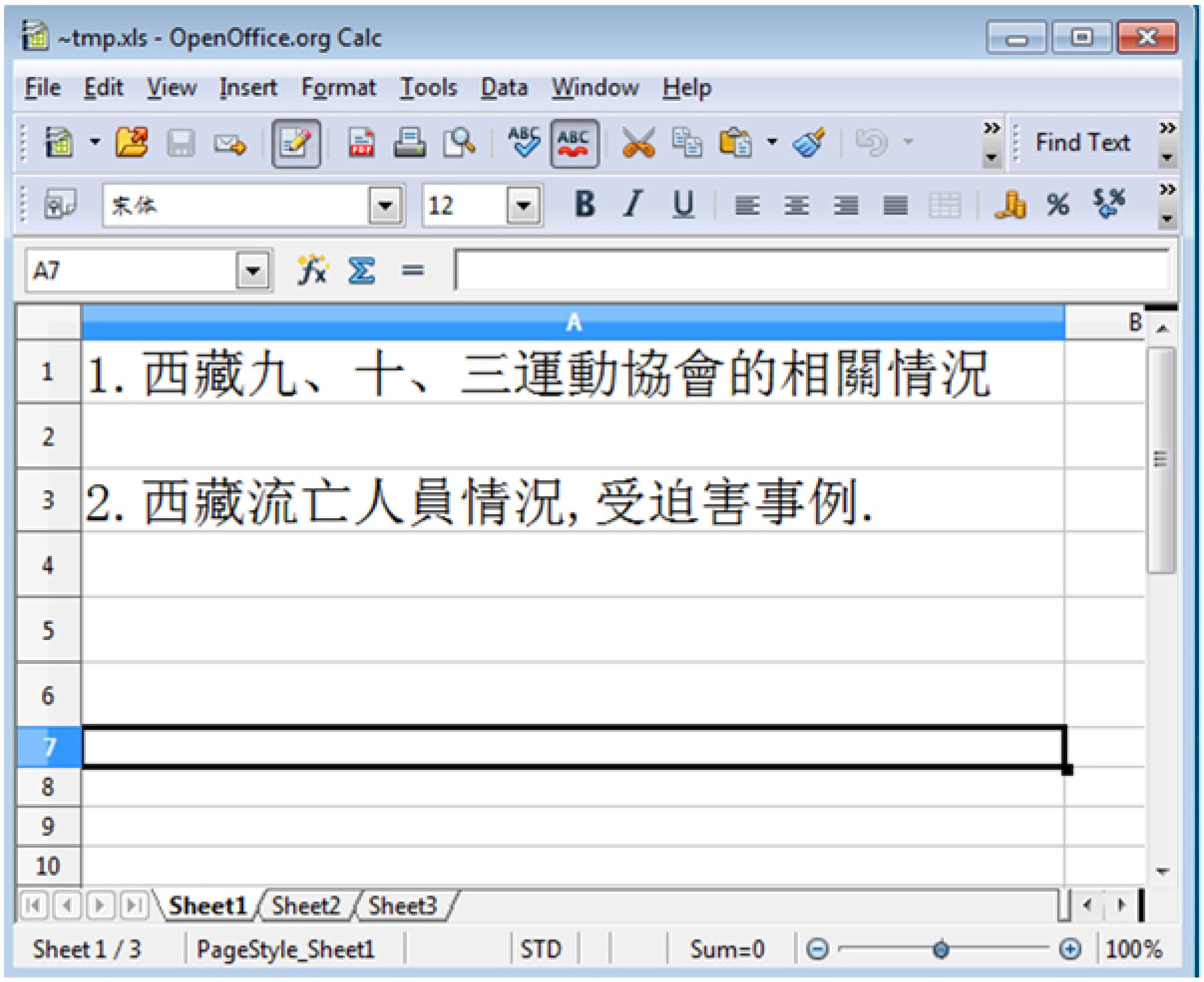
Workbook 2

Filename: sample.xls

File size: 638912 bytes

SHA1: e5e183e074d26416d7e6adfb14a80fce6d9b15c2

MD5: 2066462274ed6f6a22d8275bd5b1da2b


~tmp.xls - OpenOffice.org Calc
File Edit View Insert Format Tools Data Window Help
Find Text
宋体
12
A7
Sheet1 Sheet2 Sheet3
Sheet 1 / 3
PageStyle_Sheet1
STD
Sum=0
100%

1. 西藏九、十、三運動協會的相關情況

2. 西藏流亡人員情況, 受迫害事例.

Figure 2. Decoy content for e5e183e074d26416d7e6adfb14a80fce6d9b15c2.

Workbook 3

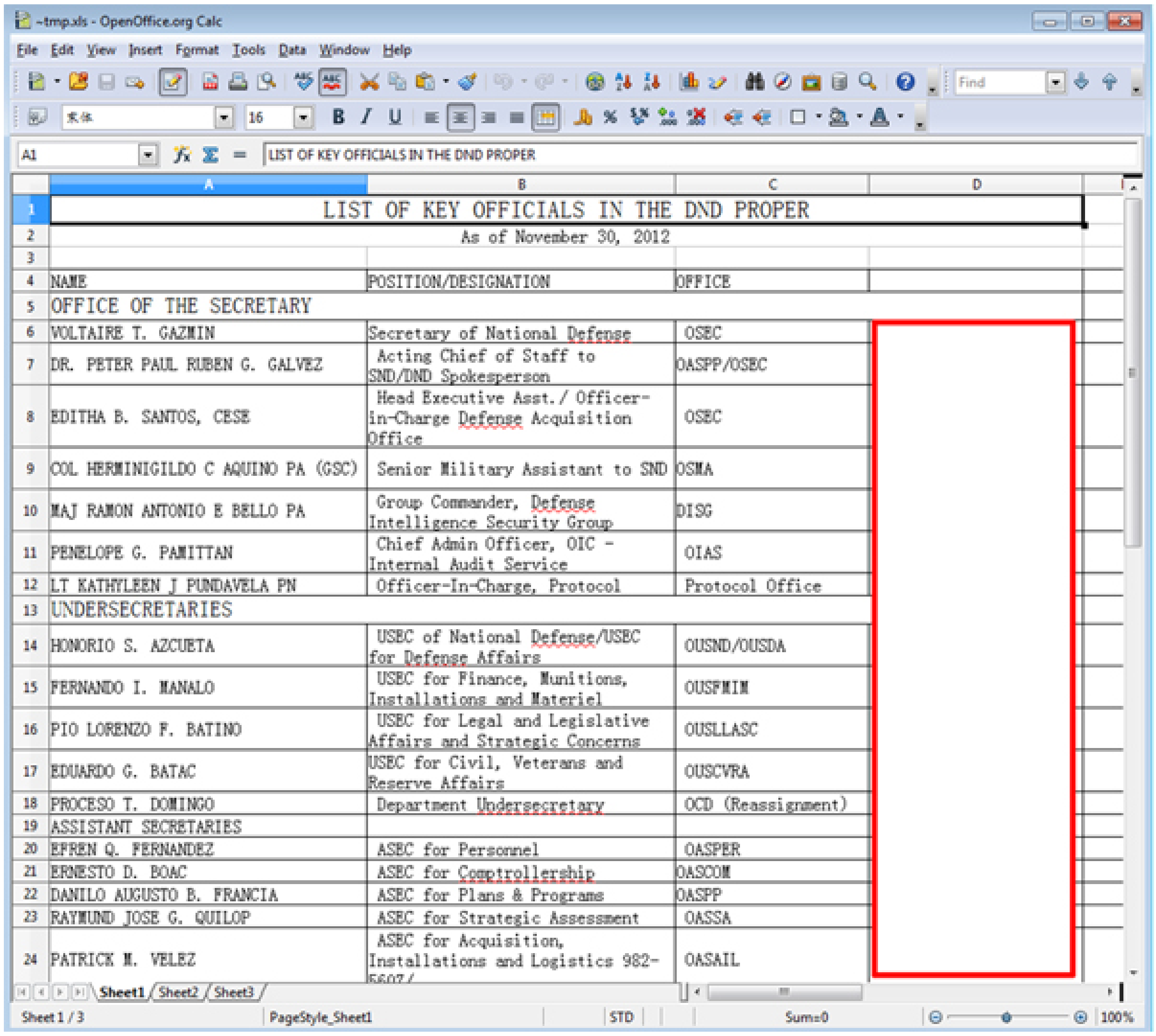
Filename: LIST OF KEY OFFICIALS IN THE DND PROPER.xls

File size: 638912 bytes

SHA1: d80b527df018ff46d5d93c44a2a276c03cd43928

MD5: 80857a5541b5804895724c5d42abd48f

This decoy workbook contains information about key officials in the Philippines Department of National Defense (DND).


~tmp.xls - OpenOffice.org Calc
File Edit View Insert Format Tools Data Window Help
Find
16
A1
Sheet1 Sheet2 Sheet3
Sheet 1 / 3
PageStyle_Sheet1
STD
Sum=0
100%

LIST OF KEY OFFICIALS IN THE DND PROPER

As of November 30, 2012

NAME	POSITION/DESIGNATION	OFFICE
OFFICE OF THE SECRETARY		
VOLTAIRE T. GAZMIN	Secretary of National Defense	OSEC
DR. PETER PAUL RUBEN G. GALVEZ	Acting Chief of Staff to SND/DND Spokesperson	OASPP/OSEC
EDITHA B. SANTOS, CESE	Head Executive Asst./ Officer-in-Charge Defense Acquisition Office	OSEC
COL HERMINIGILDO C AQUINO PA (GSC)	Senior Military Assistant to SND	OSMA
MAJ RAMON ANTONIO E BELLO PA	Group Commander, Defense Intelligence Security Group	DISG
PENELOPE G. PAMITTAN	Chief Admin Officer, OIC - Internal Audit Service	OIAS
LT KATHYLEEN J PUNDAVELA PN	Officer-In-Charge, Protocol	Protocol Office
UNDERSECRETARIES		
HONORIO S. AZCUETA	USEC of National Defense/USEC for Defense Affairs	OUSND/OUSDA
FERNANDO I. MANALO	USEC for Finance, Munitions, Installations and Materiel	OUSFMIM
PIO LORENZO F. BATINO	USEC for Legal and Legislative Affairs and Strategic Concerns	OUSLLASC
EDUARDO G. BATAC	USEC for Civil, Veterans and Reserve Affairs	OUSCVRA
PROCESO T. DOMINGO	Department Undersecretary	OCD (Reassignment)
ASSISTANT SECRETARIES		
EFREN Q. FERNANDEZ	ASEC for Personnel	OASPER
ERNESTO D. BOAC	ASEC for Comptrollership	OASCOM
DANILO AUGUSTO B. FRANCIA	ASEC for Plans & Programs	OASPP
RAYMUND JOSE G. QUILOP	ASEC for Strategic Assessment	OASSA
PATRICK M. VELEZ	ASEC for Acquisition, Installations and Logistics 982-5607/	OASAIL

Figure 3. Decoy content for d80b527df018ff46d5d93c44a2a276c03cd43928.

In the rest of this article, unless specified otherwise, we refer to the operation resulting from infection via Workbook 1 – but the overall operations (dropped filenames, registry keys, backdoor functions) are the same in each case.

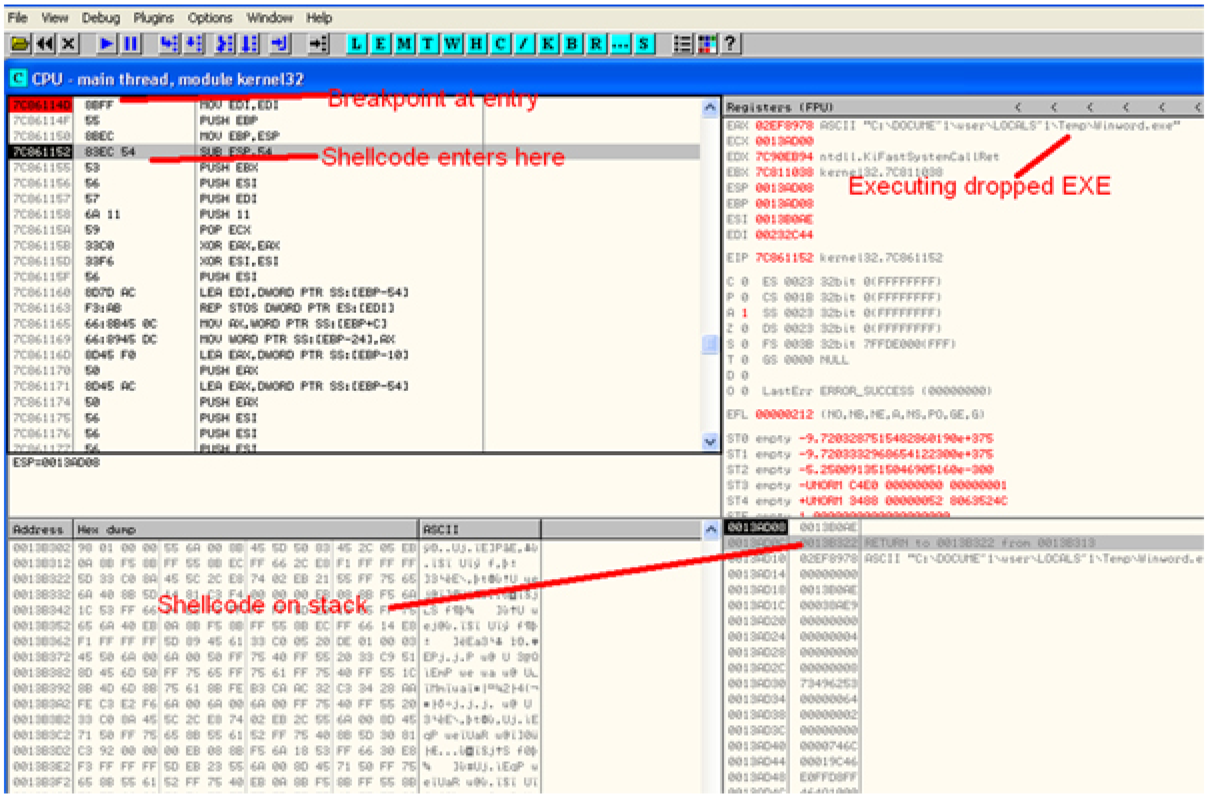
When mining our sample collection for related samples we were able to spot other examples – however, in these instances the initial dropper was not available for our analysis, only the temporary dropper executables or the final payloads could be located. In these cases we don’t have complete information about the system infection, but it is safe to assume that similar exploitation schemes were utilized.

Shellcode

The shellcode features an interesting anti-debugging trick that I have come across quite regularly in APT samples lately. Most of the *Windows* API functions are resolved and called normally, but some of the critical ones (such as WinExec and CreateFile) are not entered at the entry address (as stored in the kernel32.dll export table), but five bytes after it instead.

These functions are responsible for the most critical operations of the code (dropping the payload executable and executing it), which would reveal unusual activity in the scope of an ordinary *Excel* process.

As most tracers and debuggers would place the breakpoint or hijack function right at the entry of the API function, skipping the first few bytes is a good way to avoid API tracing and debugging.


File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S
CPU - main thread, module kernel32
Breakpoint at entry
Shellcode enters here
Executing dropped EXE
Shellcode on stack

```
7C861140  8BFF        MOV EDI,EDI
7C86114F  55          PUSH EBP
7C861150  8BEC        MOV EBP,ESP
7C861152  83EC 54     SUB ESP,54
7C861155  53          PUSH EBX
7C861156  56          PUSH ESI
7C861157  57          PUSH EDI
7C861158  6A 11       PUSH 11
7C86115A  59          POP ECX
7C86115B  33C0        XOR EAX,EAX
7C86115D  33F6        XOR ESI,ESI
7C86115F  56          PUSH ESI
7C861160  8D7D AC     LEA EDI,DWORD PTR SS:[EBP-54]
7C861163  F3:AB       REP STOS DWORD PTR ES:[EDI]
7C861165  66:8B45 0C  MOV AX,WORD PTR SS:[EBP+C]
7C861169  66:8945 DC  MOV WORD PTR SS:[EBP-24],AX
7C86116D  8D45 F0     LEA EAX,DWORD PTR SS:[EBP-10]
7C861170  50          PUSH EAX
7C861171  8D45 AC     LEA EAX,DWORD PTR SS:[EBP-54]
7C861174  50          PUSH EAX
7C861175  56          PUSH ESI
7C861176  56          PUSH ESI
7C861177  56          PUSH ESI
ESP=0013AD08
```

```
Registers (FPU)
EAX 02EF8978 ASCII "C:\DOCUME~1\user\LOCALS~1\Temp\Winword.exe"
ECX 0013AD08
EDX 7C90EB94 ntdll.KiFastSystemCallRet
EBX 7C811038 kernel32.7C811038
ESP 0013AD08
EBP 0013AD08
ESI 0013B0AE
EDI 00232C44

EIP 7C861152 kernel32.7C861152

C 0  ES 0023 32bit 0(FFFFFFFF)
P 0  CS 001B 32bit 0(FFFFFFFF)
A 1  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDE000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00000212 (NO,NB,NE,A,NS,PO,GE,G)

ST0 empty -9.720328751548286019 0e+375
ST1 empty -9.720333296865412230 0e+375
ST2 empty -5.250091351504690516 0e-300
ST3 empty -UNORM C4E0 00000000 00000001
ST4 empty +UNORM 3488 00000052 0063524C
```

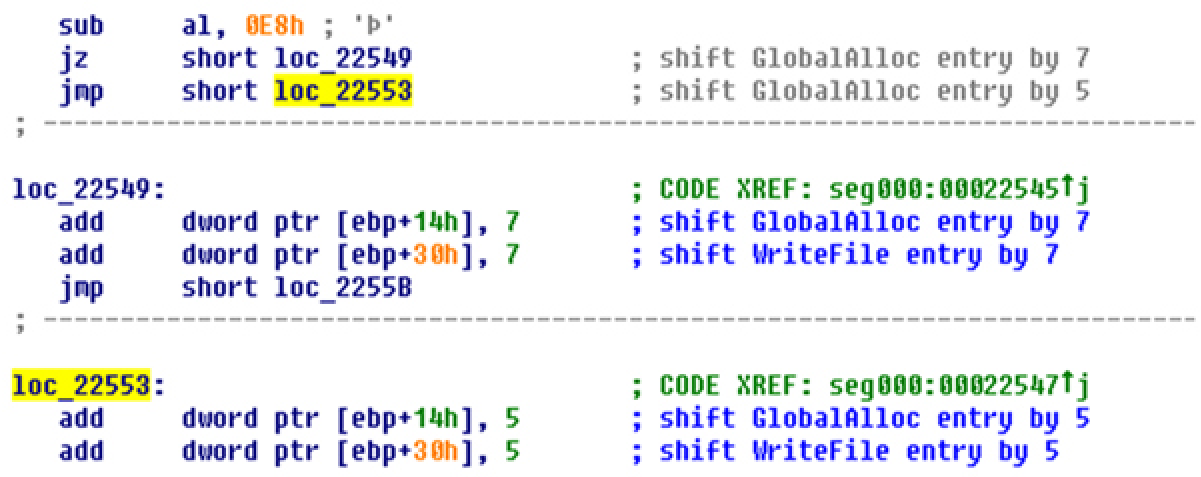
```
Address   Hex dump                                         ASCII
00138302  90 01 00 00 55 6A 00 8B 45 5D 50 03 45 2C 05 E8  [illegible]
00138312  0A 08 F5 08 FF 55 08 EC FF 66 2C E8 F1 FF FF FF  [illegible]
00138322  5D 33 C0 8A 45 5C 2C E8 74 02 EB 21 55 FF 75 65  [illegible]
00138332  6A 40 8B 5D [illegible] 00 00 00 E8 08 08 F5 6A  [illegible]
00138342  1C 53 FF 66 [illegible] FF 75                    [illegible]
00138352  65 6A 40 EB 0A 08 F5 08 FF 55 08 EC FF 66 14 E8  [illegible]
00138362  F1 FF FF FF 5D 89 45 61 33 C0 05 20 DE 01 00 03  [illegible]
00138372  45 50 6A 00 6A 00 50 FF 75 40 FF 55 20 33 C9 51  EPj.j.P u@ U 3╔Q
00138382  8D 45 6D 50 FF 75 65 FF 75 61 FF 75 40 FF 55 1C  [illegible]
00138392  8B 4D 6D 8B 75 61 8B FE 83 CA AC 32 C3 34 28 AA  [illegible]
001383A2  FE C3 E2 F6 6A 00 6A 00 6A 00 FF 75 40 FF 55 20  [illegible]
001383B2  33 C0 8A 45 5C 2C E8 74 02 EB 2C 55 6A 00 8D 45  [illegible]
001383C2  71 50 FF 75 65 8B 55 61 52 FF 75 40 8B 5D 30 81  [illegible]
001383D2  C3 92 00 00 00 EB 08 08 F5 6A 18 53 FF 66 30 E8  [illegible]
001383E2  F3 FF FF FF 5D EB 23 55 6A 00 8D 45 71 50 FF 75  [illegible]
001383F2  65 8B 55 61 52 FF 75 40 EB 0A 08 F5 08 FF 55 08  [illegible]
```

```
0013AD08  0013B0AE
0013AD0C  00138322  RETURN to 00138322 from 00138313
0013AD10  02EF8978  ASCII "C:\DOCUME~1\user\LOCALS~1\Temp\Winword.e
0013AD14  00000000
0013AD18  0013B0AE
0013AD1C  0003B0AE9
0013AD20  00000000
0013AD24  00000004
0013AD28  00000000
0013AD2C  00000000
0013AD30  73496253
0013AD34  00000064
0013AD38  00000002
0013AD3C  00000000
0013AD40  0000746C
0013AD44  00019C46
0013AD48  E0FFD8FF
```

Figure 4. Anti-tracing trick.

(Click here to view a larger version of Figure 4.)

The same happens with WriteFile and GlobalAlloc, but this time, depending on whether or not there is a call right at the entry of the function, the displacement will be either five or seven bytes.

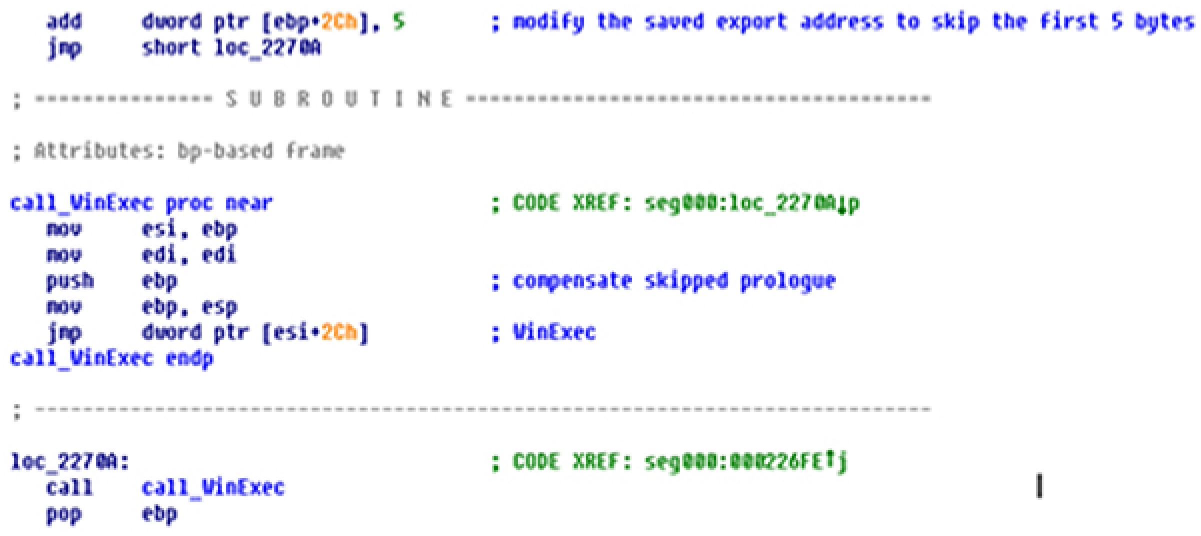


```
    sub     al, 0E8h ; 'Þ'
    jz      short loc_22549             ; shift GlobalAlloc entry by 7
    jmp     short loc_22553             ; shift GlobalAlloc entry by 5
; ---------------------------------------------------------------------------

loc_22549:                              ; CODE XREF: seg000:00022545↑j
    add     dword ptr [ebp+14h], 7      ; shift GlobalAlloc entry by 7
    add     dword ptr [ebp+30h], 7      ; shift WriteFile entry by 7
    jmp     short loc_2255B
; ---------------------------------------------------------------------------

loc_22553:                              ; CODE XREF: seg000:00022547↑j
    add     dword ptr [ebp+14h], 5      ; shift GlobalAlloc entry by 5
    add     dword ptr [ebp+30h], 5      ; shift WriteFile entry by 5
```

Figure 5. Anti-tracing hook initialization.

As a result of the functions not being entered at their usual entry points, the first few instructions are missed. As these are still essential for the stack management, the code is compensated within the shellcode, where a standard function prologue (stack frame creation push ebp, move ebp,esp) is executed.

For system functions compiled with standard compilers, the first few instructions are fixed on the entry point, but anything after that can't be taken for granted. The shellcode can't enter further than five or seven bytes into the API function, otherwise it could end up in the middle of a multi byte instruction, easily crashing the application.



```
    add     dword ptr [ebp+2Ch], 5      ; modify the saved export address to skip the first 5 bytes
    jmp     short loc_2270A

; =============== S U B R O U T I N E =======================================

; Attributes: bp-based frame

call_WinExec proc near                  ; CODE XREF: seg000:loc_2270A↓p
    mov     esi, ebp
    mov     edi, edi
    push    ebp                         ; compensate skipped prologue
    mov     ebp, esp
    jmp     dword ptr [esi+2Ch]         ; WinExec
call_WinExec endp

; ---------------------------------------------------------------------------

loc_2270A:                              ; CODE XREF: seg000:000226FE↑j
    call    call_WinExec
    pop     ebp
```

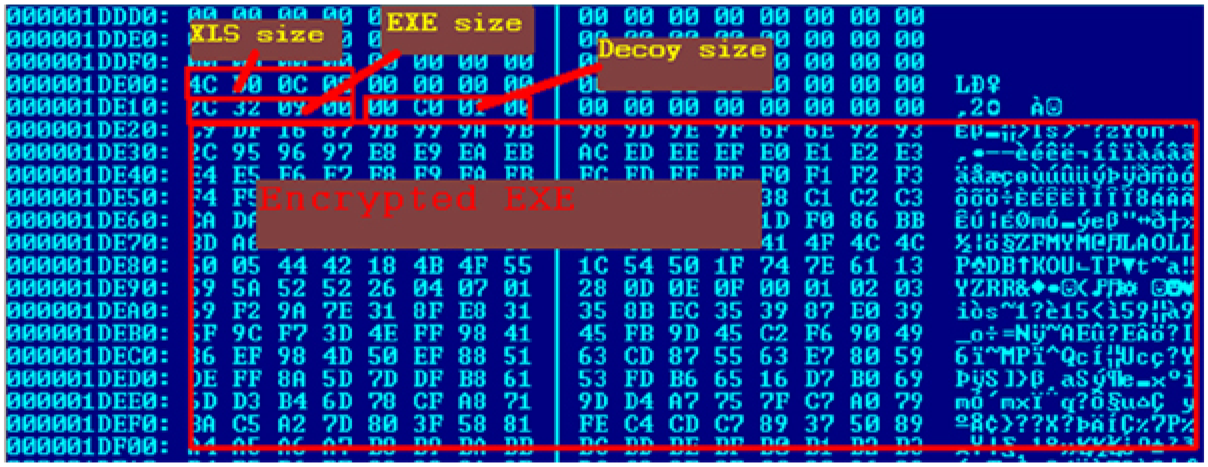
Figure 6. Anti-tracing used in practice.

In order to extract the embedded executable, the shellcode needs to find the carrier workbook. It does this using the fact that, at the time of the exploitation, the workbook must remain open in *Excel*. The code enumerates all possible handles and tries to call GetFileSize on each of them. If the function fails, because the handle does not belong to an open file (it could belong to many other objects such as directory, thread, event or registry key), or the file size is smaller than the expected size of the workbook (minus the appended encrypted EXE), 1de10h bytes, it skips to the next handle value.

Next, it reads four bytes from offset 0x1de00; the value found there should be equal to the size of the carrier workbook (this time including the appended EXE).

At this position, in the appended content following the OLE2 document structure, a short header is stored that contains the full carrier workbook size, the embedded EXE size and the embedded decoy workbook size. These values are used by the shellcode. The encrypted EXE content follows.

Organizing the code and structure in this manner makes the carrier/dropper workbook component and the dropped payload executable completely independent – it is possible to replace the payload with a new variant without changing a bit in the carrier encrypted workbook.



```
000001DDD0: 00 00 00 00 [illegible]          00 00 00 00 00 00 00 00
000001DDE0: [illegible]                      [illegible] 00 00 00
000001DDF0: [illegible]          00 00 00    [illegible] 00 00 00
000001DE00: 4C [illegible] 0C [illegible] 00 00 00 [illegible]   [illegible] 00 00 00   LD♀
000001DE10: 2C 32 [illegible] 00 00 C0 [illegible]   00 00 00 00 00 00 00 00   ,2○  à☺
000001DE20: [illegible]
000001DE30: 2C 95 96 97 E8 E9 EA EB   AC ED EE EF E0 E1 E2 E3
000001DE40: E4 E5 F6 F7 F8 F9 FA FB   FC FD FE FF F0 F1 F2 F3
000001DE50: F4 F5 [illegible]         38 C1 C2 C3
000001DE60: CA DF [illegible]         1D F0 86 BB
000001DE70: 3D A6 [illegible]         41 4F 4C 4C
000001DE80: 50 05 44 42 18 4B 4F 55   1C 54 50 1F 74 7E 61 13   P♠DB↑KOU∟TP▼t~a‼
000001DE90: 59 5A 52 52 26 04 07 01   28 0D 0E 0F 00 01 02 03   YZRR&♦•☺(♪♫☼ ☺☻♥
000001DEA0: 69 F2 9A 7E 31 8F E8 31   35 8B EC 35 39 87 E0 39   iòs~1?è15<ì59‡à9
000001DEB0: 5F 9C F7 3D 4E FF 98 41   45 FB 9D 45 C2 F6 90 49   _o÷=Nÿ~AEû?Eâö?I
000001DEC0: 36 EF 98 4D 50 EF 88 51   63 CD 87 55 63 E7 80 59   6ï~MPï^QcÍ‡Ucç?Y
000001DED0: DE FF 8A 5D 7D DF B8 61   53 FD B6 65 16 D7 B0 69   Þÿš]}ß¸aSý¶e▬×°i
000001DEE0: 6D D3 B4 6D 78 CF A8 71   9D D4 A7 75 7F C7 A0 79   mÓ´mxÏ¨q?Ô§u⌂Ç y
000001DEF0: 3A C5 A2 7D 80 3F 58 81   FE C4 CD C7 89 37 50 89   ºÅ¢}??X?þÄÍÇ‰7P‰
000001DF00: [illegible]
```

Figure 7. Appended header and payload.

Once the hosting workbook is found, the code proceeds with decoding the embedded executable (using a one byte XOR algorithm with running key plus an additional one byte XOR with a fixed key), saving it to a file named ‘Winword.exe’ in the %TEMP% directory, then executing it. At this point, the decoy workbook content is dropped (using the same algorithm: one byte XOR with running key plus one byte XOR with fixed key, only this key differs from the one used in decoding the EXE).

Temporary dropper

This file is the dropper and installer for the final payload. It has an initial anti-debug layer.

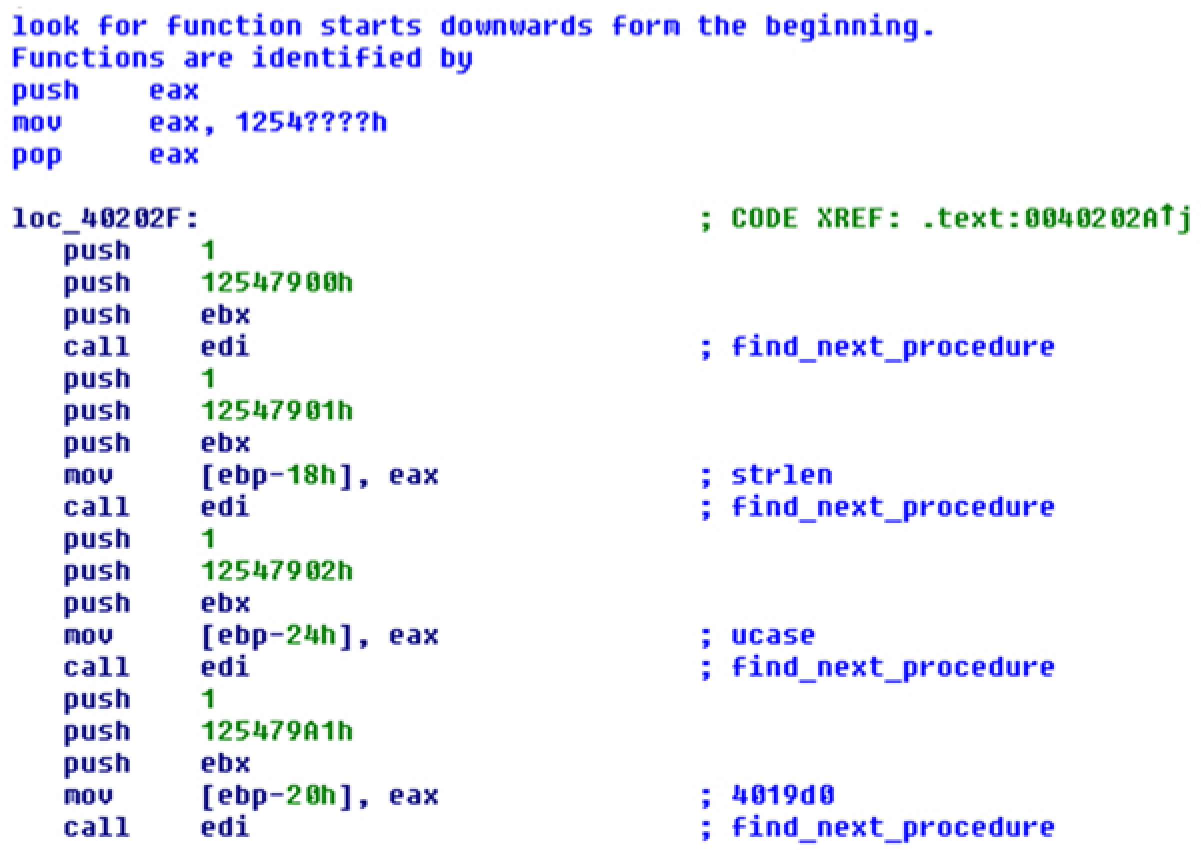
The address of the GetVersion function is patched in the import table, to contain an internal function virtual address instead of an imported function address, which is normally expected at that position. The code around the entry point uses the stored value to redirect execution:

```
mov  large fs:0, esp
sub  esp, 58h
push ebx
push esi
push edi
mov  [ebp-18h], esp
call ds:dword_41A188
```

The execution actually goes to the address stored at dword_41A188, which is the memory location 00402440.

The program has only one export, LoadLibrary, thus when the operating system loads the program and resolves the external dependencies, this value, stored within the import table region, remains intact. The trick completely fools *IDA Pro*, which can't be convinced that the location is an internal position and not an external import. This makes static analysis a bit more complicated. The necessary imported function addresses are later resolved dynamically by the initialization code of the dropper.

The major procedures of the dropper program are not called directly; instead, the trojan builds a function pointer table, and calls to procedures are performed via indexing into this table, as shown in Figure 8.



```
look for function starts downwards form the beginning.
Functions are identified by
push     eax
mov      eax, 1254????h
pop      eax

loc_40202F:                                  ; CODE XREF: .text:0040202A↑j
    push     1
    push     12547900h
    push     ebx
    call     edi                             ; find_next_procedure
    push     1
    push     12547901h
    push     ebx
    mov      [ebp-18h], eax                  ; strlen
    call     edi                             ; find_next_procedure
    push     1
    push     12547902h
    push     ebx
    mov      [ebp-24h], eax                  ; ucase
    call     edi                             ; find_next_procedure
    push     1
    push     125479A1h
    push     ebx
    mov      [ebp-20h], eax                  ; 4019d0
    call     edi                             ; find_next_procedure
```

Figure 8. Building the function pointer table.

The key procedures are identified by having the following instruction sequence near the prologue:

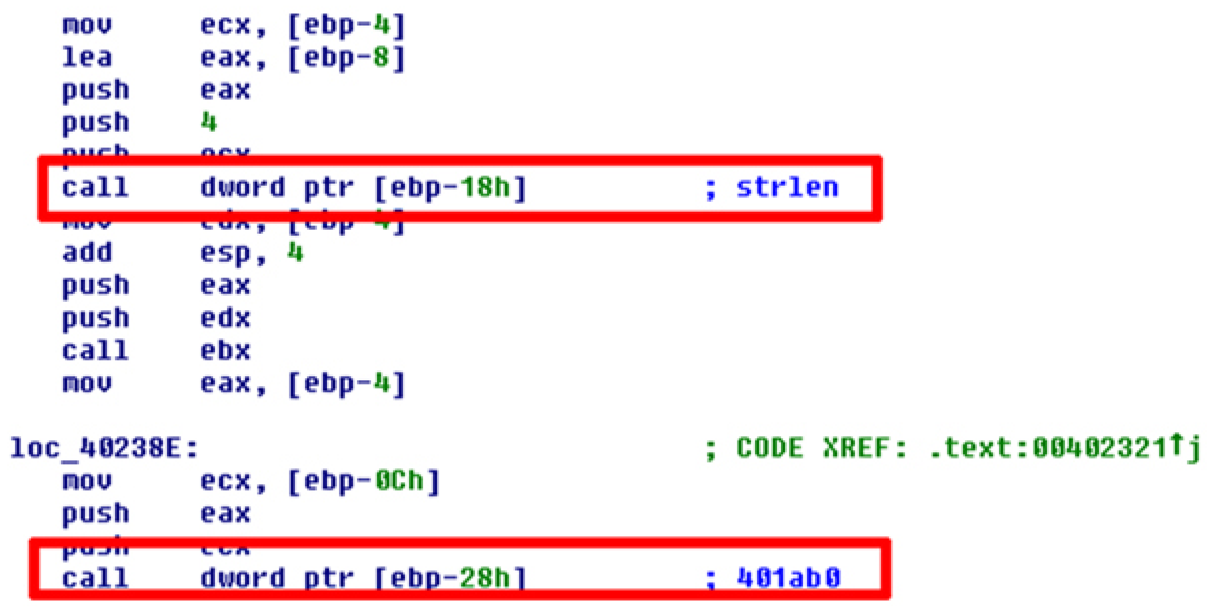
```
push ebp
mov  ebp, esp
push eax
mov  eax, 12547908h
pop  eax
```

The value stored in the EAX register is a combination of two elements: 1254 is the marker; 7908 is the numeric ID for the function.

The entry is located by searching backwards for the standard prologue:

```
push ebp
mov  ebp, esp
```

The procedures are later invoked by calling indexes from the function pointer table (see Figure 9).



```
    mov     ecx, [ebp-4]
    lea     eax, [ebp-8]
    push    eax
    push    4
    push    ecx
    call    dword ptr [ebp-18h]         ; strlen
    mov     edx, [ebp-4]
    add     esp, 4
    push    eax
    push    edx
    call    ebx
    mov     eax, [ebp-4]

loc_40238E:                             ; CODE XREF: .text:00402321↑j
    mov     ecx, [ebp-0Ch]
    push    eax
    push    ecx
    call    dword ptr [ebp-28h]         ; 401ab0
```

Figure 9. Using the function pointer table.

Winword.exe normally drops three major components into the system:

- %PROGRAM FILES%\Common Files\ODBC\AppMgmt.dll – the final payload (*Windows* DLL file)
- %PROGRAM FILES%\Common Files\DBEngin.EXE – a copy of rundll32.exe (a clean *Windows* system file, used for executing the payload)
- %PROGRAM FILES%\Common Files\WUAUCTL.EXE – another rundll32.exe (a clean *Windows* system file, used for executing the payload).

Additionally, two registry export files named jus*.tmp (with a random number added after jus) are dropped into %TEMP%. These are the old and new hives of the HKLM\SYSTEM\CurrentControlSet\Services\AppMgmt registry location – a location at which the trojan registers itself in order to execute automatically upon each system boot. Saving the hives to a file makes it possible to modify the registry in one shot using RegRestoreKey.

Also dropped is a 301,445 byte long jus*.tmp file, which is a CAB archive containing the payload DLL.

The execution flow takes a different route if the presence of running security products is detected. The following process names are checked: KVMonXP.exe, RavMonD.exe, RsTray.exe, ccsvchst.exe, QQPCTray.exe, zhudongfangyu.exe, 360sd.exe, 360Tray.exe, zatray.exe, bdagent.exe, ksafetray.exe, kxetray.exe and avp.exe. However, not all of the security processes are checked at the same time – only a couple of selected ones are checked before each major operation.

As an example, if zatray.exe, RsTray.exe or RavMonD.exe is running, then AppMgmt.dll is not dropped and instead, the 400MB vbstdcomm.nls is created (the large size is due to an enormous amount of junk appended at the end of the file). Finally, a VBScript file is created and executed with the help of a dropped copy of wscript.exe (both files are saved to the %TEMP% folder, as lgt*.tmp.vbs and lgt*.tmp.exe, respectively). An encrypted copy of Winword.exe is created in %CommonProgramFiles%\ODBC\odbc.txt, using a one byte XOR algorithm with key 0xCC. Vbstdcomm.nls, which serves as a backup installer, takes the encrypted copy of Winword.exe, decodes it and simply executes.

The dropper registers AppMgmt.dll as a service. This is not achieved by creating a new service entry, rather by taking over the role of an already installed service, AppMgmt, redirecting the service DLL from the clean library to the dropped malware payload:

HKLM\SYSTEM\CurrentControlSet\Services\AppMgmt\Parameters: ServiceDll

%SystemRoot%\System32\appmgmts.dll -> C:\Program Files\Common Files\ODBC\AppMgmt.dll

In addition, the start up mode is changed from auto to demand in the location:

HKLM\SYSTEM\CurrentControlSet\Services\AppMgmt: Start

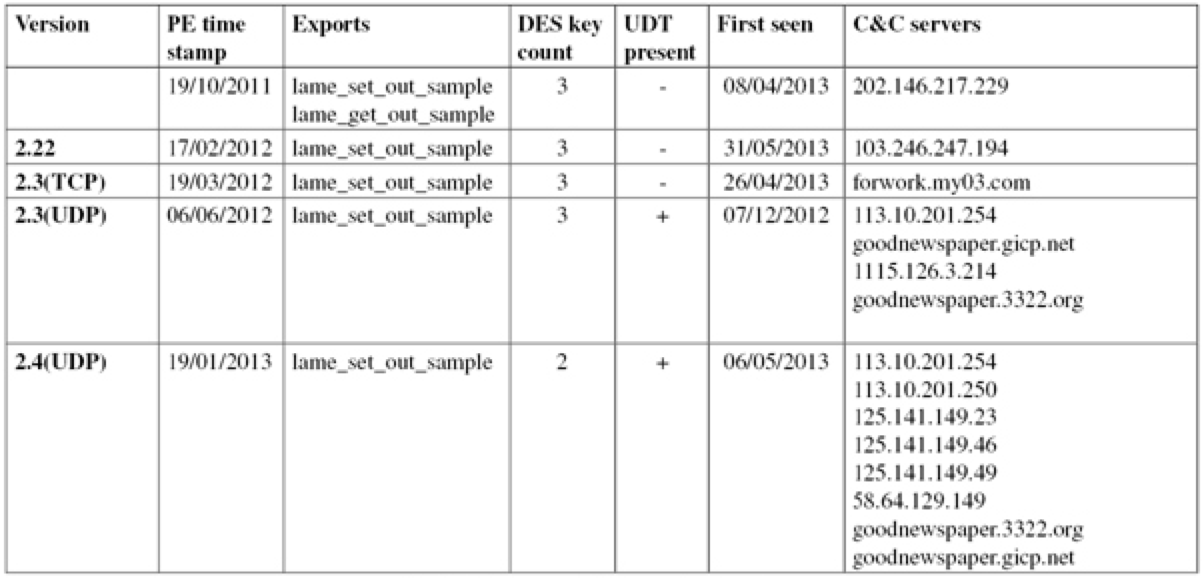
Then it changes the error control settings in the registry key HKLM\SYSTEM\CurrentControlSet\Services\AppMgmt:ErrorControl from *normal* (this would mean that if the driver fails to load, the start up process proceeds, but a warning is displayed) to *ignore* (in this case if the driver fails to load, start up proceeds, and no warning is displayed). The change is designed to avoid raising suspicion, should start up fail for any reason.

Finally, it executes the dropped DLL by executing net start AppMgmt.

Payload

We have identified five different versions of the final payload. Two of them were replicated from the exploited workbooks detailed earlier; the other three were found when we were digging through our sample collection searching for samples with similar characteristics.

The main characteristics of the five variants are summarized in Table 1 shown below (detailed descriptions of the columns are provided later in this section).

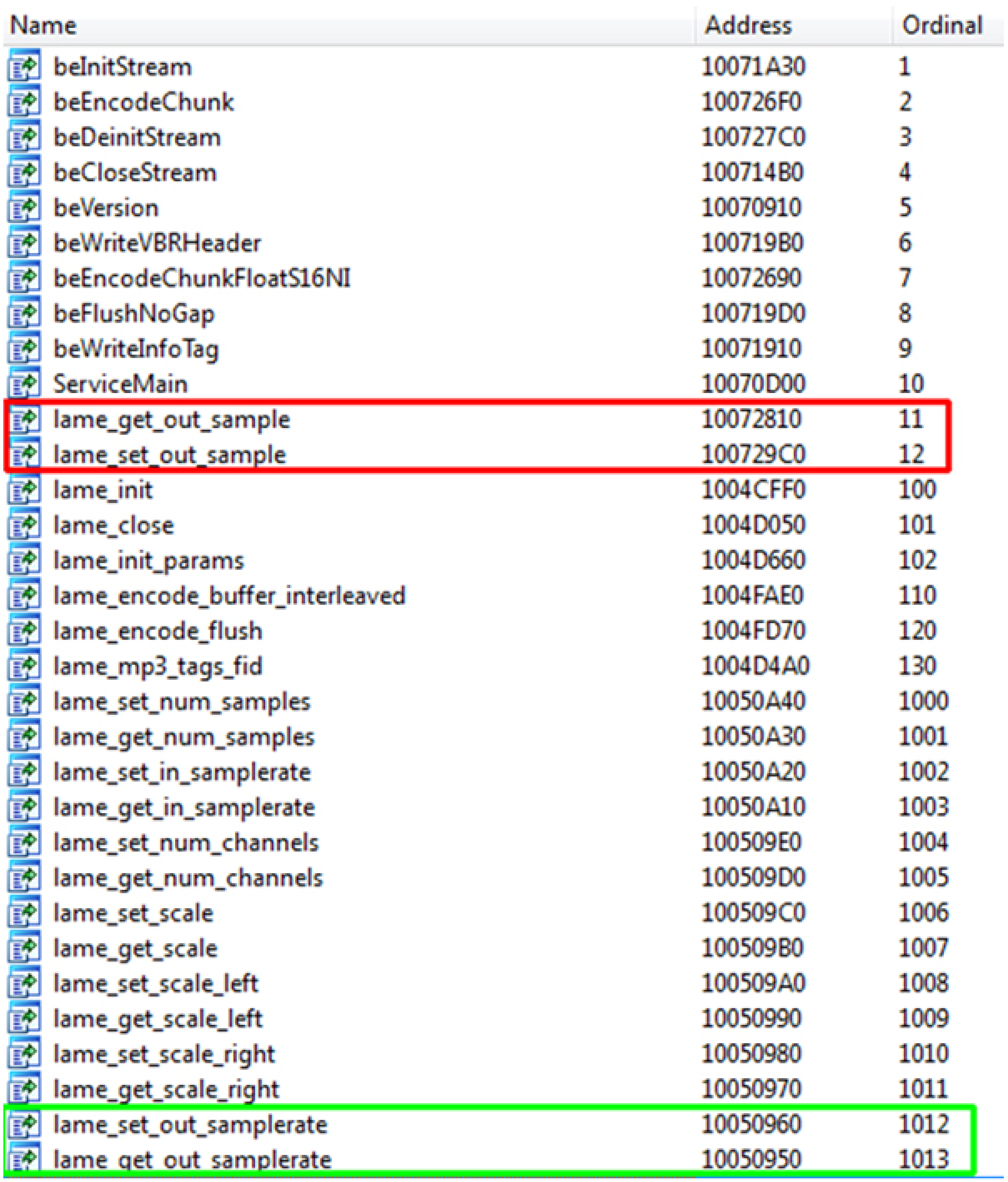


Version	PE time stamp	Exports	DES key count	UDT present	First seen	C&C servers
	19/10/2011	lame_set_out_sample lame_get_out_sample	3	-	08/04/2013	202.146.217.229
2.22	17/02/2012	lame_set_out_sample	3	-	31/05/2013	103.246.247.194
2.3(TCP)	19/03/2012	lame_set_out_sample	3	-	26/04/2013	forwork.my03.com
2.3(UDP)	06/06/2012	lame_set_out_sample	3	+	07/12/2012	113.10.201.254 goodnewspaper.gicp.net 1115.126.3.214 goodnewspaper.3322.org
2.4(UDP)	19/01/2013	lame_set_out_sample	2	+	06/05/2013	113.10.201.254 113.10.201.250 125.141.149.23 125.141.149.46 125.141.149.49 58.64.129.149 goodnewspaper.3322.org goodnewspaper.gicp.net

Table 1: Summary of the payload versions.

(Click here to view a larger version of Table 1.)

This DLL is built from the LAME MP3 encoder source [2]. The full library has been compiled, and in addition, a couple of malicious exports have been added to the code: lame_set_out_sample and lame_get_out_sample.



Name	Address	Ordinal
beInitStream	10071A30	1
beEncodeChunk	100726F0	2
beDeinitStream	100727C0	3
beCloseStream	100714B0	4
beVersion	10070910	5
beWriteVBRHeader	100719B0	6
beEncodeChunkFloatS16NI	10072690	7
beFlushNoGap	100719D0	8
beWriteInfoTag	10071910	9
ServiceMain	10070D00	10
lame_get_out_sample	10072810	11
lame_set_out_sample	100729C0	12
lame_init	1004CFF0	100
lame_close	1004D050	101
lame_init_params	1004D660	102
lame_encode_buffer_interleaved	1004FAE0	110
lame_encode_flush	1004FD70	120
lame_mp3_tags_fid	1004D4A0	130
lame_set_num_samples	10050A40	1000
lame_get_num_samples	10050A30	1001
lame_set_in_samplerate	10050A20	1002
lame_get_in_samplerate	10050A10	1003
lame_set_num_channels	100509E0	1004
lame_get_num_channels	100509D0	1005
lame_set_scale	100509C0	1006
lame_get_scale	100509B0	1007
lame_set_scale_left	100509A0	1008
lame_get_scale_left	10050990	1009
lame_set_scale_right	10050980	1010
lame_get_scale_right	10050970	1011
lame_set_out_samplerate	10050960	1012
lame_get_out_samplerate	10050950	1013

Figure 10. Additional malicious imports.

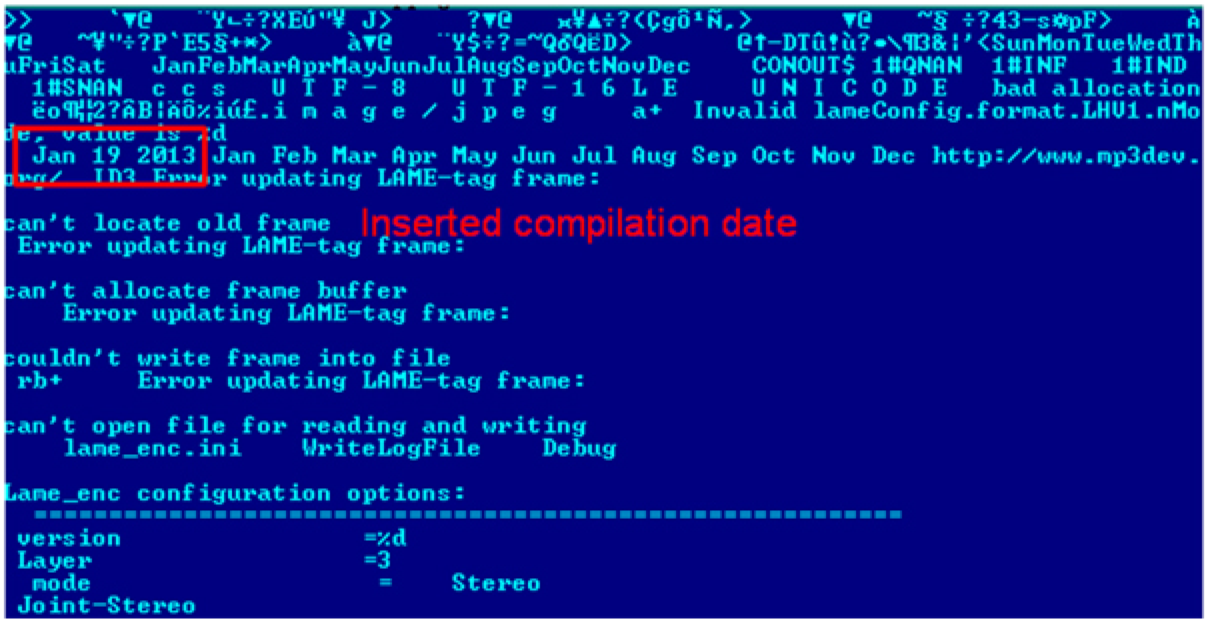
Note that the names of the additional exports are strikingly similar to the legitimate exports, lame_set_out_samplerate and lame_get_out_samplerate, which are present in the LAME source – thus it is not very obvious that the additional exports belong to something completely different.

One of the extra exports, lame_get_out_sample, is missing from newer versions of the malware. However, the function that would invoke this export is still present in the code. Clearly, the code was not cleaned up properly when the export was removed.

The backdoor contains many encrypted strings, one of which serves as an internal version number. In Table 1 we list the version numbers as they appear in the code. Collected information suggests that the most widely distributed was version 2.3(UDP), making its rounds in the wild in early December 2012.

Table 1 also lists the date when we first saw each particular backdoor variant – either arriving in our collection, reported in cloud look ups or seen elsewhere on the Internet. Additionally, the compilation date is listed, as taken from the PE header.

An interesting quirk comes from the usage of the LAME source: one of the original source functions, beVersion(), inserts the compilation date into the data section of the executable.



```
>>      `▼@   ¨Y∟÷?XEú"¥ J>      ?▼@    ¤¥▲÷?<Çgô¹Ñ,>        ▼@    ~§ ÷?43-s#pF>       À
▼@   ~¥"÷?P`E5§+*>      à▼@   ¨Y$÷?=~QδQËD>       @↑-DIû!ù?•\¶3&¦'<SunMonTueWedTh
uFriSat   JanFebMarAprMayJunJulAugSepOctNovDec     CONOUT$ 1#QNAN  1#INF   1#IND
  1#SNAN  c c s   U T F - 8   U T F - 1 6 L E     U N I C O D E   bad allocation
  ëo¶╬2?âB¦Äõ%iú£.i m a g e / j p e g      a+  Invalid lameConfig.format.LHV1.nMo
de, value is %d
  Jan 19 2013 Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec http://www.mp3dev.
org/  ID3 Error updating LAME-tag frame:

can't locate old frame
 Error updating LAME-tag frame:

can't allocate frame buffer
    Error updating LAME-tag frame:

couldn't write frame into file
 rb+     Error updating LAME-tag frame:

can't open file for reading and writing
    lame_enc.ini    WriteLogFile    Debug

Lame_enc configuration options:
  ==========================================================
 version                   =%d
 Layer                     =3
  mode                      =    Stereo
 Joint-Stereo
```

Figure 11. Compilation date in code.

This provides an independent method of determining the creation date of the variant aside from the PE time stamp. There was no trick, however – the two dates matched in all cases.

It is notable that there is always a large gap between the compilation date and the date of the first observation of each variant. There are several possible reasons for this:

- Small-scale targeted attacks don't provide much telemetry information; the smaller the number of targets, the slimmer our chances of finding out about their infection.
- The trojan looks very similar to a real LAME encoder library; infected victims are reluctant to submit it for analysis.
- There may be an intentional delay (some sort of testing period) in the release process of the malware.

The backdoor uses different approaches for handling C&C communication. Earlier versions used the standard Windows socket communication functions (send, recv) to exchange data with the C&C server. The newer versions linked the UDT data transfer library (available from udt.sourceforge.net) for communication. The versioning of the variants suggests that some time around March 2012, the code forked into a socket communication branch (TCP) and a UDT powered communication branch (UDP).

The backdoor features all the basic functionality that is expected from a piece of malware of its class. It is able to:

- Create screenshots
- Get drive type (FAT, FAT32, NTFS, CDFS) and free space
- Enumerate files and directories and send the list to the server
- Rename files
- Create directories
- Delete files.

The last character of the ModuleFileName (without extension) is checked on execution: if it is not of one of the expected values – 'T', 't' (executed via net.exe), 'R', 'r', 'N', 'n' (executed via DBEngin.EXE), '2' (rundll32.exe), 'L' or 'l' – it builds and injects a simple piece of code to load AppMgmt.dll properly.

For this purpose, it creates a new suspended process (with command line: c:\windows\system32\svchost.exe), calls GetThreadContext on it, and gets EAX from the CONTEXT structure, using the fact that in the case of a suspended process the EAX register always points to the entry point of the process. Then it writes the starter code to this entry point and resumes the thread. The suspended thread is not visible in the process list at that point. This way, the trojan can escape analysis, if not executed in a natural form, and still execute.

Configuration data is stored in a file named DbTrans.db, XOR encrypted with key 0x58.

The string constants (API names, DLL names, process names) are all stored in encrypted form using a strong encryption algorithm. The strings are stored aligned (Unicode strings to 0x90 bytes, ASCII strings to 0x38 bytes boundary), decrypted in eight byte chunks using the DES ECB algorithm, and referenced by IDs that index into this name pool. The encrypted strings contain padding bytes at the end, where zeros are encoded.

The strings are decrypted on the fly before being used and filled with zeros after use. This way there are no visible strings in the memory that would give away more information about the internals of the backdoor.

There are three nearly identical encryption functions (and accompanying encrypted string tables and encryption keys) in all variants: one is for the Unicode strings, one for the ordinary ASCII constants, and a third one for the *Windows* API function names (also stored as ASCII strings) that are used in the code. We found that only the encryption keys were different for the three cases. The following key seeds remain the same throughout the variants:

For ASCII strings: 82 C5 D3 59 2B 38 00 00

For Unicode strings: 5E 97 CC 42 8E CD 00 00

For API function names: 5B 5F CB 8D E5 F5 00 00

In the last version, the two ASCII functions are merged into a single function.

The C&C addresses are hard coded into the backdoor, and protected with a simple byte wise XOR (key:0x58) encryption. This is an interesting choice, given that all other string constants are protected with a string DES algorithm – perhaps the server addresses are changed more frequently (indeed, there is a minimal overlap between the different versions' C&C addresses) than the authors are comfortable with re-encrypting the strings – but no evidence was found for it in the few samples we have found.

The string constants of the code are referenced by IDs and decrypted on the fly. However, there are strings that are never used in the code. These could belong to an earlier or internal version, and simply have not been cleaned up from the string pool, as illustrated in this example:

```
push 9   ; ,lame_set_out_sample
call Get_String_A
push 0Ah ; ,
call Get_String_A
push 1Eh ; DBEngin.exe
call Get_String_A
push 8   ; EXPL.EXE
pop  eax
call Get_String_W
push offset s_expl_exe
push [ebp+var_254]
call StrCpyW
push 8
pop  eax
xor  ecx, ecx
call set_mem
push ebx
push 2
call CreateToolhelp32Snapshot
```

Some of these strings could be internal configuration options for the development environment (I suspect these are access details to an internal server):

```
kazafei
192.168.1.98
80
```

Other strings provide status information about the current operation of the backdoor:

```
Client RecvData Complete
A File Search Task has start already !!!
File Search Task Success
File Search Task Failed, Please Check
Upload Client Failed
Upload Client Success
Delete File Success
Delete File Failed
Rename File Success
Rename File Failed
Create Folder Success
Create Folder Failed
```

A few constants indicate undocumented or debug functionality:

```
X:\Windows\System32\rundll32.exe
X:\Windows\msacm32.drv
MagicMutex
D:\Resume.dll
D:\delete.dll
D:\delete2.dll
```

Conclusion

When looking into APT attack scenarios, one has to be extra careful. Often we see that clean programs and libraries are dropped onto systems to hide the operation of malicious applications [3]. But sometimes, what looks to be a genuine MP3 encoder library, and even works as a functional encoder, actually hides malicious additions buried deep in a large pile of clean code. One has to be very thorough when it comes to targeted attacks, and one cannot afford to make any assumptions.

Bibliography

[1] Baccas, P. When is a password not a password? When Excel sees “VelvetSweatshop”. http://nakedsecurity.sophos.com/2013/04/11/password-excel-velvet-sweatshop/.

[2] LAME (Lame Aint an MP3 Encoder). http://sourceforge.net/projects/lame/.

[3] Szappanos, G. Targeted malware attack piggybacks on Nvidia digital signature. http://nakedsecurity.sophos.com/2013/02/27/targeted-attack-nvidia-digital-signature/.

Latest articles:

Dissecting the design and vulnerabilities in AZORult C&C panels

Aditya K Sood looks at the command-and-control (C&C) design of the AZORult malware, discussing his team's findings related to the C&C design and some security issues they identified during the research.

Bulletin Archive