

Initiation au reverse engineering

Analyse d'un ransomware



@r00tbsd - Paul Rascagneres

malware.lu

15 June 2012

Plan

1 Introduction

- Projet
- Législation
- Outils

2 Analyse

- Wallpaper
- Lister les fichiers
- Encodeur
- Roue de secours

3 Conclusion

Pour les gens sur Internet



Pour les personnes souhaitant suivre le workshop en remote voici une archive avec tout le nécessaire.

<http://www.malware.lu/pses/workshop.tgz>

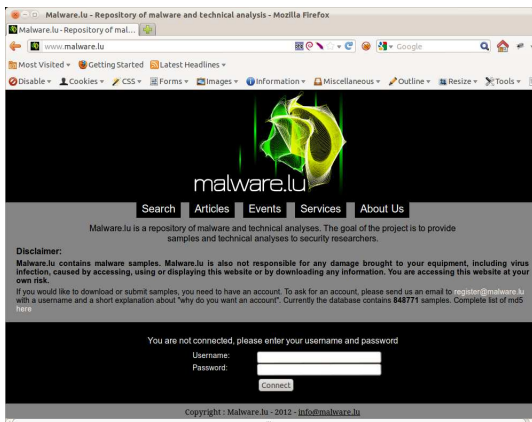
Introduction



Presentation du projet malware.lu.

Liste des mainteneurs:

- @r00tbsd - Paul Rascagneres
- @y0ug - Hugo Caron



Quelques chiffres



Le projet en quelques chiffres:

- 848771 Samples
- 18 articles
- 600 utilisateurs
- 511 followers on twitter (@malwarelu)

Extrait du nouvel article Art. L. 335-3-1 introduit l'article 22 du DADVSI :

I. - Est puni de 3 750 EUR d'amende le fait de porter atteinte sciemment, à des fins autres que la recherche, à une mesure technique efficace telle que définie à l'article L. 331-5, afin d'altérer la protection d'une œuvre par un décodage, un décryptage ou toute autre intervention personnelle destinée à contourner, neutraliser ou supprimer un mécanisme de protection ou de contrôle, (...)

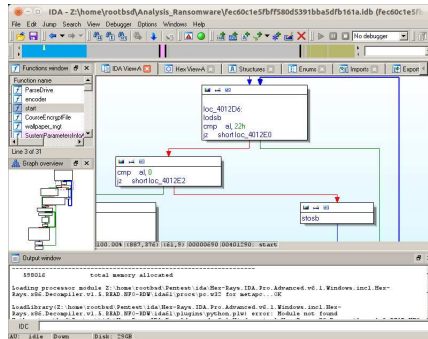
II. - Est puni de six mois d'emprisonnement et de 30 000 EUR d'amende le fait de procurer ou proposer sciemment à autrui, directement ou indirectement, des moyens conçus ou spécialement adaptés pour porter atteinte à une mesure technique efficace (...)

IV. - Ces dispositions ne sont pas applicables aux actes réalisés à des fins de recherche (...) ou de sécurité informatique, dans les limites des droits prévus par le présent code.

IDA Pro est un désassembleur (permettant d'avoir le programme assembleur d'un binaire) commercial utilisé pour le reverse engineering. Il existe plusieurs versions:

- pro (sous licences)
- demo
- free

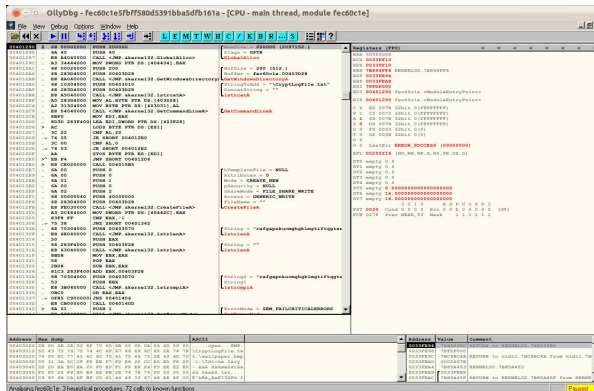
Screenshot:



Quelques raccourcis utiles:

- **N** : renommer une fonction
- **:** : mettre du commentaire
- **double click** : entrer dans une fonction
- **Echap** : retour a la fonction précédente
- **X** : Lister les endroits ou est référencé l'objet sélectionné

OllyDbg est un debugger pour Microsoft Windows.
Ce logiciel est gratuit.



Quelques raccourcis utiles:

- **F7** : pas à pas en entrant dans les fonctions
- **F8** : pas à pas sans entrer dans les fonctions
- **F9** : demarrer l'exécution
- **CTRL-F9** : exécuter jusqu'au prochain return
- **F2** : mettre un breakpoint

Début de l'analyse



Nous allons analyser le sample
fec60c1e5fbff580d5391bba5dfb161a.

Pour ce faire nous aurons besoin de:

- IDA Pro
- OllyDbg
- une machine Virtuelle (pour éviter d'infecter notre machine maitre).

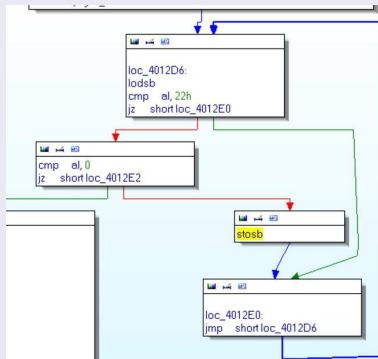
Début de l'analyse



Entry point

```
; Attributes: noreturn
public start
start proc near
push 200000h ; dwBytes
push 40h ; uFlags
call GlobalAlloc
mov lpBuffer, eax
push 200h ; uSize
push offset Buffer ; lpBuffer
call GetWindowsDirectoryA
push offset aCryptlogfile_t_ ; "\\CryptLogFile.txt"
push offset Buffer ; lpString1
call lstrcatA
mov al, Buffer
mov FileName, al
call GetCommandLineA
mov esi, eax
lea edi, byte_403F28
```

Entry point suite



Début de l'analyse



Instruction LODS

LODS
LODSB
LODSW
LODSD

Note: Loads DS:[SI] (ESI for LODSD) value into AL, AX, EAX and increments SI

Instruction STOS

STOS
STOSB
STOSW
STOSD

Note: Stores AL/AX/EAX in DS:[(E)DI] and increments (E)DI.

Ces boucles parcourent donc ESI pour stocker ces qu'il y a entre "" dans EDI.

EDI contient le nom du binaire.

Wallpaper

Nous arrivons maintenant:

sub_4015B5

```
; Attributes: bp-based frame
sub_4015B5 proc near

lpBuffer= dword ptr -10h
Buffer= dword ptr -0Ch
lDistanceToMove= dword ptr -8
hObject= dword ptr -4

push    ebp
mov     ebp, esp
add     esp, 0FFFFFFF0h
push    0             ; hTemplateFile
push    0             ; dwFlagsAndAttributes
push    3             ; dwCreationDisposition
push    0             ; lpSecurityAttributes
push    1             ; dwShareMode
push    80000000h     ; dwDesiredAccess
push    offset byte_403F28, lpFileName
call    CreateFileA
cmp     eax, 0FFFFFFFh
jnz     short loc_4015DE
```

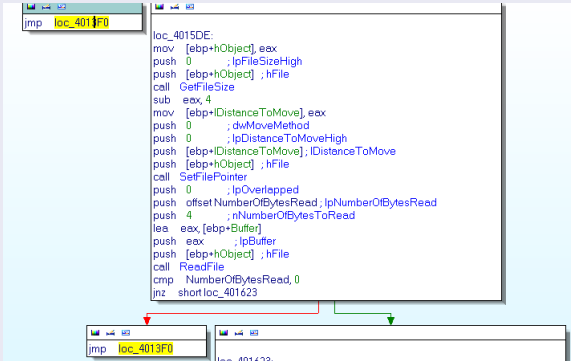
jmp loc_4013F0

loc_4015DE:

Le programme s'ouvre lui même en lecture.

Wallpaper

Suite sub_4015B5



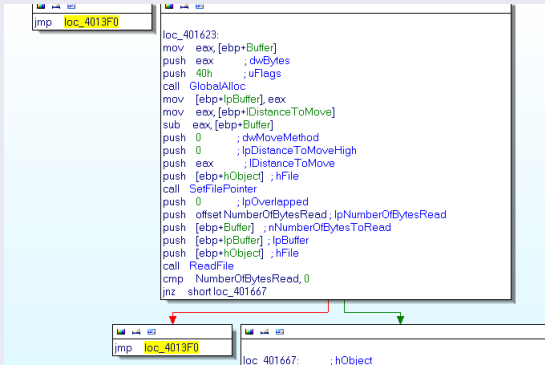
```
loc_4015DE:
mov     [ebp+hObject], eax
push    0             ; lpFileSizeHigh
push    [ebp+hObject] ; hFile
call    GetFileSize
sub     eax, 4
mov     [ebp+lDistanceToMove], eax
push    0             ; dwMoveMethod
push    0             ; lpDistanceToMoveHigh
push    [ebp+lDistanceToMove]; lDistanceToMove
push    [ebp+hObject] ; hFile
call    SetFilePointer
push    0             ; lpOverlapped
push    offset NumberOfBytesRead ; lpNumberOfBytesRead
push    4             ; nNumberOfBytesToRead
lea     eax, [ebp+Buffer]
push    eax           ; lpBuffer
push    [ebp+hObject] ; hFile
call    ReadFile
cmp     NumberOfBytesRead, 0
jnz     short loc_401623

loc_4013F0:
jmp     loc_4013F0
```

Il calcule la taille du fichier et retire 0x4.
Ensuite il se place à la fin du fichier -0x4.
Il lit les 4 bytes.

Wallpaper

Suite sub_4015B5



```
loc_401623:
mov     eax, [ebp+Buffer]
push    eax                ; dwBytes
push    40h                ; uFlags
call    GlobalAlloc
mov     [ebp+lpBuffer], eax
mov     eax, [ebp+IDistanceToMove]
sub     eax, [ebp+Buffer]
push    0                  ; dwMoveMethod
push    0                  ; lpDistanceToMoveHigh
push    eax                ; IDistanceToMove
push    [ebp+hObject]      ; hFile
call    SetFilePointer
push    0                  ; lpOverlapped
push    offset NumberOfBytesRead ; lpNumberOfBytesRead
push    [ebp+Buffer]       ; nNumberOfBytesToRead
push    [ebp+lpBuffer]     ; lpBuffer
push    [ebp+hObject]      ; hFile
call    ReadFile
cmp     NumberOfBytesRead, 0
jnz     short loc_401667

loc_401667:                ; hObject

jmp     loc_4013F0
```

Il va à la fin du fichier - 0x4 - la valeur lue précédemment. Et stock le contenu dans lpBuffer.

Wallpaper

Suite sub_4015B5

```
jmp loc_4013F0

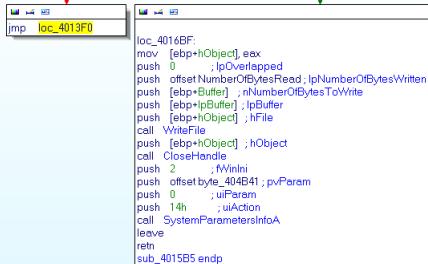
loc_401667: ; hObject
push [ebp+hObject]
call CloseHandle
push 200h ; nSize
push offset byte_404B41 ; lpBuffer
push (offset aTmp+2) ; lpName
call GetEnvironmentVariableA
push offset aWallpaper_bmp ; "\\wallpaper.bmp"
push offset byte_404B41 ; lpString1
call lstrcatA
push offset byte_404B41 ; lpFileName
call DeleteFileA
push 0 ; hTemplateFile
push 0 ; dwFlagsAndAttributes
push 2 ; dwCreationDisposition
push 0 ; lpSecurityAttributes
push 3 ; dwShareMode
push 0C000000h ; dwDesiredAccess
push offset byte_404B41 ; lpFileName
call CreateFileA
cmp eax, 0FFFFFFFFh
jnz short loc_4016BF
```

```
jmp loc_4013F0
```

Il crée un fichier C:\temp\Wallpaper.bmp

Wallpaper

Suite sub_4015B5



The screenshot shows a debugger window with two panes. The left pane displays a jump instruction: `jmp loc_4013F0`. A red arrow points from this instruction to the right pane. The right pane displays the assembly code for the function `loc_4016BF`, which is the continuation of `sub_4015B5`. The code includes a `mov` instruction for `eax`, followed by several `push` instructions for `lpOverlapped`, `NumberOfBytesRead`, `lpNumberofBytesWritten`, `lpBuffer`, `hFile`, `hObject`, `WinIni`, `pvParam`, `uiParam`, and `uiAction`. It then calls `WriteFile`, `CloseHandle`, and `SystemParametersInfoA`, before `leave`ing and `retn`ing. The function ends with `sub_4015B5 endp`. A green arrow points from the `retn` instruction back to the `jmp` instruction in the left pane.

```
loc_4016BF:
mov     [ebp+hObject], eax
push    0             ; lpOverlapped
push    offset NumberOfBytesRead ; lpNumberofBytesWritten
push    [ebp+Buffer] ; lpBuffer
push    [ebp+lpBuffer] ; lpBuffer
push    [ebp+hObject] ; hFile
call    WriteFile
push    [ebp+hObject] ; hObject
call    CloseHandle
push    2             ; fWinIni
push    offset byte_404B41 ; pvParam
push    0             ; uiParam
push    14h          ; uiAction
call    SystemParametersInfoA
leave
retn
sub_4015B5 endp
```

Il écrit dans le fichier `C:\temp\Wallpaper.bmp` ce qu'il a précédemment extrait du binaire.

Wallpaper

Voici le script en ruby de récupération du wallpaper

```
1 #!/usr/bin/env ruby
2
3 if ARGV.length != 2
4   puts "#{File.basename(__FILE__)} malware outputfile"
5   exit
6 end
7
8 malwareFile = File.open(ARGV[0], 'r')
9 malwareFile.seek(-0x4, IO::SEEK_END)
10 size = malwareFile.sysread(0x4)
11 size = size.unpack('V*')
12 size = size[0] + 0x4
13 puts "Offset : 0x#{size.to_s(16)}"
14
15 malwareFile = File.open(ARGV[0], 'r')
16 malwareFile.seek(-size, IO::SEEK_END)
17 bmp = malwareFile.sysread(size)
18 bmpFile = File.open(ARGV[1], 'w')
19 bmpFile.write(bmp)
```

extract_wallpaper.rb

Lister les fichiers

loc_401342

```
loc_401342: ; uMode  
push 1  
call SetErrorMode  
call GetLogicalDrives  
mov ecx, 19h
```

```
loc_401353:  
mov ebx, 1  
shl ebx, cl  
end ebx, eax  
jz short loc_40138B
```

```
add cl, 41h  
mov byte ptr dword_403327+1, cl  
sub cl, 41h  
mov dword_403327+2, 2E2A5C3Ah  
mov byte_40332D, 2Ah  
mov byte_40332E, 0  
push eax  
push ecx  
call sub_401000  
pop ecx  
pop eax
```

```
loc_40138B:  
dec ecx  
jge short loc_401353
```

Explication

Cette portion de code récupère la liste des disques et les passe en argument de la fonction `sub_401000`.

sub_401000

Nous n'allons pas faire une analyse complète de cette fonction mais seulement le portion de code interessantes.

loc_401342

```
loc_4010E9:  
mov     ecx, ebx  
dec     ecx  
imul    ecx, 4  
add     ecx, offset a_doc_xlsdocxxl ; ".doc.xlsdocxxl"  
mov     ecx, [ecx]  
mov     String2, ecx  
mov     byte_404D4A, 0  
test    cl, cl  
jnz     short loc_401117
```

```
push    eax  
mov     eax, dword_404D41  
mov     cl, al  
mov     String2, ecx  
pop     eax
```

```
loc_401117:  
push    eax  
push    offset String2 ; lpString2  
push    offset dword_404D41 ; lpString1  
call    lstrcmpiA  
test    eax, eax  
pop     eax  
jz      short loc_401138
```

Explication

Cette portion de code vérifie si l'extension est intéressantes dans a_doc_xlsdocxxl.

sub_401000

loc_401342

```
pop     eax
jz      short loc_401138
```

```
dec     ebx
cmp     ebx, 0
jz      loc_401242
```

```
loc_401138:      ; lpString
push     (offset dword_403327+1)
call     strlenA
sub      eax, 2
push     eax      ; iMaxLength
push     (offset dword_403327+1); lpString2
push     offset byte_403828; lpString1
call     strcpynA
lea      eax, [ebp+FindFileData.cFileName]
push     eax      ; lpString2
push     offset byte_403828; lpString1
call     strcpyA
push     0         ; hTemplateFile
push     0         ; dwFlagsAndAttributes
push     3         ; dwCreationDisposition
push     0         ; lpSecurityAttributes
push     3         ; dwShareMode
push     0C000000h ; dwDesiredAccess
push     offset byte_403828; lpFileName
call     CreateFileA
inc      eax
or       eax, eax
jz       loc_401242
```

Explication

Si l'extension est présente dans la liste le fichier est ouvert en écriture.

sub_401000



loc_401342

```
dec     eax
mov     hFile, eax
push    0          ; lpFileSizeHigh
push    eax        ; hFile
call    GetFileSize
mov     dword_404430, eax
push    0          ; lpOverlapped
push    offset NumberOfBytesRead ; lpNumberOfBytesRead
push    200000h    ; nNumberOfBytesToRead
push    lpBuffer   ; lpBuffer
push    hFile      ; hFile
call    ReadFile
cmp     NumberOfBytesRead, 0
jnz     short loc_4011C3
```

```
jmp     short loc_4011FC
```

```
loc_4011C3:
mov     eax, NumberOfBytesRead
call    sub_401263
push    0          ; dwMoveMethod
push    0          ; lpDistanceToMoveHigh
push    0          ; lpDistanceToMove
push    hFile      ; hFile
call    SetFilePointer
push    0          ; lpOverlapped
push    offset NumberOfBytesWritten ; lpNumberOfBytesWritten
push    NumberOfBytesRead ; nNumberOfBytesToWrite
push    lpBuffer   ; lpBuffer
push    hFile      ; hFile
call    WriteFile
```

Explication

Le fichier est ouvert et le contenu est utilisé par la fonction sub_401263.

Le retour de cette fonction est écrit dans le fichier.

Nous pouvons facilement en deduire que la fonction sub_401263 est l'encodeur.

loc_401342



```
loc_4011FC: ; hObject
push     hFile
call     CloseHandle
push     offset byte_403828 ; lpString
call     lstrlenA
push     0 ; lpOverlapped
push     offset NumberOfBytesWritten ; lpNumberOfBytesWritten
push     eax ; nNumberOfBytesToWrite
push     offset byte_403828 ; lpBuffer
push     dword_40442C ; hFile
call     WriteFile
push     0 ; lpOverlapped
push     offset NumberOfBytesWritten ; lpNumberOfBytesWritten
push     2 ; nNumberOfBytesToWrite
push     offset aTmp ; "\\\\.\\tmp"
push     dword_40442C ; hFile
call     WriteFile
```

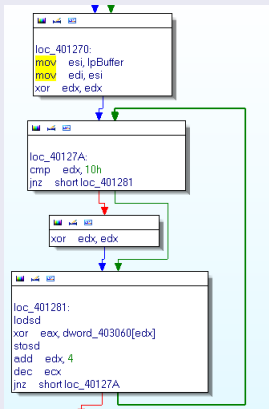
Explication

Pour finir le nom du fichier encodé est mis dans le fichier `CryptLogFile.txt`.

Ce fichier contient donc la liste des fichiers encodés.

Encodeur

sub_401263



Explication

L'encodeur est simple un XOR avec pour clé la valeur de `dword_403060` (sur 0x10 caracteres).

L'avantage de ce type de codage est qu'il est très facile à décoder. Il suffit de faire le même XOR.

$$A \text{ XOR } B = C$$

$$C \text{ XOR } B = A$$

Decodeur



Voici le script en ruby de le decodage d'un fichier encoder.

```
1 #!/usr/bin/env ruby1.9.1
2
3 malwareFile = File.open(ARGV[0], 'r')
4 malwareFile.seek(0x1060, IO::SEEK_SET)
5 key = malwareFile.sysread(0x10)
6
7 encryptedFile = File.open(ARGV[1], 'r')
8 encryptedFile.seek(0x0, IO::SEEK_SET)
9 file = encryptedFile.sysread(File.stat(ARGV[1]).size)
10 i=0
11
12 file.each_byte { |x|
13   puts x ^ key[i%0x10].ord
14   i+=1
15 }
```

decode.rb

Roue de secours



Le developpeur c'est laissé 2 roues de secours pour décoder les fichiers avec le binaire du malware lui même.

Nous allons voir ses deux roues de secours dans cette partie.

Roue de secours - 1

Existence du fichier CryptLogFile.txt

```

push 0 ; lpSecurityAttributes
push 2 ; dwShareMode
push 40000000h ; dwDesiredAccess
push offset Buffer ; lpFileName
call CreateFileA
mov dword_40442C, eax
cmp eax, 0FFFFFFFFh
jnz short loc_401342

push offset String ; "rfagpnkucmghgklmgtiftq"
call strlenA
push eax
push offset byte_403F28 ; lpString
call strlenA
mov ebx, eax
pop eax
sub ebx, eax
add ebx, offset byte_403F28
push offset String ; "rfagpnkucmghgklmgtiftq"
push ebx ; lpString1
call strcmpiA
or eax, eax
jnz loc_401406

call sub_40140D

loc_401342: ; uMode
push 1
call SetLastError

```

Explication

Si le fichier existe déjà, le programme ne suis pas son fonctionnement normal. Il va dans un bloque de code que nous verrons plus tard.

En effet le programme ne doit pas être executé 2 fois car, via le principe même du XOR, les fichiers seraient décodés.

Mais en supprimant simplement ce fichier, le programme va s'executer une deuxième fois et décoder tous ce qui avait été codé précédement!!

Roue de secours - 2

rafgapnku...

```
push offset String ; "rafgapnkucmghgklmgtiftqgtswqtrim"
call strlenA
push eax
push offset byte_403F28 ; lpString
call strlenA
mov ebx, eax
pop eax
sub ebx, eax
add ebx, offset byte_403F28
push offset String ; "rafgapnkucmghgklmgtiftqgtswqtrim"
push ebx ; lpString1
call lstrcmpA
or eax, eax
jnz loc_401406
```

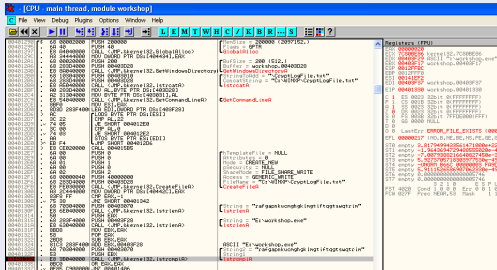
```
call sub_40140D
```

Explication

Cette fonction verifie si l'extension du malware est "rafgapnkucmghgklmgtiftqgtswqtrim".

Si l'extension est bien celle-ci, la fonction sub_40140D est appelée.
(celle-ci appel ensuite la fonction d'encodage.

Nous voici donc en présence d'une 2eme roue de secours permettant de decoder les fichiers.



Pour que le test réussisse il faut bien que le binaire finisse par
rafgapnkucmghgklmgitiftqgtswoqtrim.

Conclusion

En conclusion merci à tous pour votre présence.

J'espère vous avoir montré que le reverse n'était pas si obscure que ça et qu'il pouvait être très pratique dans certains cas.

Maintenant place aux questions (ou à un blanc gênant !!)