

# ***Rapport BE Virologie***

## ***binaire sudoku.exe***

**Rédacteurs :**

William VITAL  
Pierre ROLLAND

# SOMMAIRE

PREAMBULE .....	2
Environnement.....	2
Objectif du TP.....	2
Livrables .....	2
QUESTIONS .....	3
Question 1 : conséquences après exécution du malware.....	3
Question 2 : mécanismes de persistance.....	3
Question 3 : instances concurrentes.....	6
Question 4 : stratégie d'analyse .....	9
Question 5 : types de fichiers impactés .....	11
Question 6 : fichiers chiffrés une seule fois .....	11
Question 7 : type de transformations subies .....	11
Question 8 : restauration des fichiers.....	12
Question 9 : technique à mettre en œuvre.....	13
Question 10 : outil de suppression du cryptovirus.....	13
Question 11 : outil de déchiffrement .....	13

## PREAMBULE

Ce rapport est réalisé dans le cadre de l'évaluation du binaire « sudoku.exe » transmis par [REDACTED] le 27/02/2019.

## Environnement

L'environnement de travail est une machine virtuelle Windows 10 sous Virtualbox dotée de l'outil IDA pro.

## Objectif du TP

L'objectif de ce TP est d'aider Mme Michu à se sortir de cette situation délicate. Pour cela, plusieurs questions vous sont posées afin de vous guider dans cette tâche.

## Livrables

A l'issue de ce TP, il vous est demandé de fournir :

- un rapport récapitulant votre démarche, c'est-à-dire les réponses aux questions ci-après ;
- les codes sources des outils demandés ;
- la base IDB (Ida Pro) support de votre analyse.

Le code source des outils demandé est disponible sur <https://github.com/wvmcs/be-sudoku2>.

La base IDA support à l'analyse est également présente sur ce github.


# QUESTIONS

## Question 1 : conséquences après exécution du malware

Vous venez de récupérer le binaire incriminé. Afin de reproduire les problèmes observés par Mme Michu, vous téléchargez différents types de documents sur Internet (images, pdf, présentations, fichiers html, etc). Ensuite, vous lancez le binaire en question. Quelles sont les conséquences du malware que vous observez après exécution ?

Les fichiers présents dans les répertoires de l'utilisateur\* (png,jpg,pdf,html) sont désormais illisibles. A titre d'exemple, le fichier html commence par : GPGcryptor!!!

Un fichier « README.txt » a été créé dans chaque dossier :

 README.txt - Bloc-notes

Fichier Edition Format Affichage ?

GPGcryptor!!!  
Some files are protected.  
To buy decoder mail: md66@mail.re  
with subject: Help me!






→ Son contenu semble concerner un attaque de type ransomware (famille des Trojan horse selon la terminologie employée par Symantec).

\* Note : l'analyse approfondie permettra de constater que ce sont l'ensemble des fichiers du répertoire %HOMEDRIVE%\%HOMEPATH% (C:\Users\IEUser) et de ses sous-répertoires qui sont scannés par le malware (y compris les répertoires cachés) afin de réaliser le chiffrement des fichiers personnels de l'utilisateur (voir liste de ces fichiers un peu plus loin).

## Question 2 : mécanismes de persistance

Mme Michu supprime le binaire qu'elle a lancé et se rend compte, qu'après avoir redémarré sa machine, les mêmes problèmes se répètent. Expliquez précisément les mécanismes mis en jeu par le malware pour assurer sa persistance. Quels outils utilisez vous ? Identifier dans une base IDA pro le code en charge de cette persistance.

Le gestionnaire de tâches confirme le lancement d'un programme exécutable sous l'identité de notre utilisateur IEUser. Nous installons la suite Sysinternals<sup>1</sup> et lançons l'outil « Autoruns.exe ».

Autorun Entry	Description	Publisher	Image Path
 HKLM\SYSTEM\CurrentControlSet\Control\SafeBoot\Alternate Shell			
<input checked="" type="checkbox"/>  cmd.exe	Interpréteur de commandes Windows	(Verified) Microsoft Windows	c:\windows\system32\cmd.exe
 HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run			
<input checked="" type="checkbox"/>  bginfo			File not found: C:\BGinfo\Bginfo.exe ...
<input checked="" type="checkbox"/>  VBoxTray	VirtualBox Guest Additions Tray Appli...	(Verified) Oracle Corporation	c:\windows\system32\vbboxtray.exe
 HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run			
<input checked="" type="checkbox"/>  _SuDokU_	Microsoft OneDrive	(Verified) Microsoft Corporation	c:\users\ieuser\17845-3188-26527_....
<input checked="" type="checkbox"/>  OneDrive	Microsoft OneDrive	(Verified) Microsoft Corporation	c:\users\ieuser\appdata\local\micro...
 C:\Users\IEUser\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup			
<input checked="" type="checkbox"/>  14579-8708-26629_.exe			c:\users\ieuser\appdata\roaming\mi...
<input checked="" type="checkbox"/>  README.txt			c:\users\ieuser\appdata\roaming\mi...

Dans la base de registre, le programme ajoute une clé \_SuDokU\_ avec la valeur C:\Users\IEUser\17845-3188-26527\_.exe sous :

- HKEY\_CURRENT\_USER\Software\Microsoft\Windows\CurrentVersion\Run

<sup>1</sup> <https://docs.microsoft.com/en-us/sysinternals/downloads/sysinternals-suite>

*Note : la documentation Microsoft précise que cet enregistrement provoquera l'exécution automatique du programme indiqué à chaque log in de l'utilisateur.*

Il ajoute également un programme exécutable dans le dossier de lancement automatique :

- C:\Users\EUser\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\14579-8708-26629\_.exe

Windows > Menu Démarrer > Programmes > Démarrage					Rechercher dans : Démarrage
Nom	Modifié le	Type	Taille		
14579-8708-26629_.exe	27/02/2019 04:59	Application	123 Ko		
README.txt	27/02/2019 05:04	Document texte	1 Ko		

*Note : l'emplacement de ce répertoire de lancement automatique est déterminé par le malware en examinant la clé de registre HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders qui pointe sur le répertoire de lancement (startup). La documentation Microsoft indique que ce répertoire n'est utilisé que par les versions Windows XP, Vista, Windows 7 et Windows 8.*

Ces fichiers Sudoku.exe, 17845-3188-26527\_.exe, 14579-8708-26629\_.exe produisent la même empreinte md5 ou sha1 (validé avec l'outil Microsoft Certutil -hashfile ... , possible également en PowerShell) :

- 52fb8ba70fbc1ba8f9d665f21929bb50 (cette empreinte constitue un IOC !)

Ils sont donc des copies.

→ Le programme établit sa persistance en se répliquant dans le profil de l'utilisateur et en appelant ces 2 copies par l'ajout de clé de registre.

```
.text:004012D2
.text:004012D2 Install_persistence proc near          ; CODE XREF: StartAddress+7
.text:004012D2
.text:004012D2 string          = dword ptr -31Ch
.text:004012D2 cbData         = dword ptr -318h
.text:004012D2 phkResult      = dword ptr -314h
.text:004012D2 Filename       = byte ptr -310h
.text:004012D2 Data           = byte ptr -20Ch
.text:004012D2 NewFileName    = byte ptr -108h
.text:004012D2 var_4          = dword ptr -4
.text:004012D2
.text:004012D2 push          ebp
.text:004012D3 mov          ebp, esp
.text:004012D5 sub          esp, 31Ch
.text:004012D8 mov          eax, ___security_cookie
.text:004012E0 xor          eax, ebp
.text:004012E2 mov          [ebp+var_4], eax
.text:004012E5 push          ebx
.text:004012E6 push          esi
.text:004012E7 lea          eax, [ebp+phkResult]
.text:004012ED mov          [ebp+string], ecx
.text:004012F3 push          eax          ; phkResult
.text:004012F4 push          3            ; samDesired
.text:004012F6 xor          esi, esi
.text:004012F8 push          esi          ; ulOptions
.text:004012F9 call         near ptr Get_nom_HKCU
.text:004012FE mov          ebx, ds:RegOpenKeyExA
.text:00401304 push          eax          ; lpSubKey
.text:00401305 push          HKEY_CURRENT_USER ; hKey
.text:0040130A call         ebx ; RegOpenKeyExA
.text:0040130C test         eax, eax
.text:0040130E jnz         loc_4014A0
.text:00401314 lea          eax, [ebp+cbData]
.text:0040131A mov          [ebp+cbData], 104h
```

```

text:0040140C      push     1          ; samDesired
text:0040140E      push     0          ; ulOptions
text:00401410      call    Get_Explorer_shellFolders
text:00401415      push     eax         ; lpSubKey
text:00401416      push     HKEY_CURRENT_USER ; hKey
text:00401418      call    ebx ; RegOpenKeyExA
text:0040141D      test     eax, eax
text:0040141F      jnz     short loc_4014A0
text:00401421      lea     eax, [ebp+cbData]
text:00401427      mov     ebx, 104h
text:0040142C      push     eax         ; lpcbData
text:0040142D      lea     eax, [ebp+Data]
text:00401433      mov     [ebp+cbData], ebx
text:00401439      push     eax         ; lpData
text:0040143A      push     0          ; lpType
text:0040143C      push     0          ; lpReserved
text:0040143E      push     offset aStartup ; "startup"
text:00401443      push     [ebp+phkResult] ; hKey
text:00401449      call    esi ; RegQueryValueExA
text:0040144B      test     eax, eax
text:0040144D      jnz     short loc_4014A0
text:0040144F      call    ds:GetTickCount
text:00401455      push     eax
text:00401456      call    sub_4052FC
text:0040145B      pop      ecx

```

```

lea     eax, [ebp+cbData]
mov     ebx, 104h
push    eax         ; lpcbData
lea     eax, [ebp+Data]
mov     [ebp+cbData], ebx
push    eax         ; lpData
push    0          ; lpType
push    0          ; lpReserved
push    offset aStartup ; "startup"
push    [ebp+phkResult] ; hKey
call    esi ; RegQueryValueExA
test     eax, eax
jnz     short loc_4014A0

```

```

call    ds:GetTickCount
push    eax
call    sub_4052FC
pop      ecx
call    rand
push    eax
call    rand
push    eax
call    rand
push    eax
lea     eax, [ebp+Data]
push    eax
push    offset aS04u04u02uExe ; "%s\\%04u-%04u-%02u_.exe"
lea     eax, [ebp+NewFileName]
push    ebx
push    eax
call    snprintf?
add     esp, 1Ch
lea     eax, [ebp+NewFileName]
push    1          ; bFailIfExists
push    eax         ; lpNewFileName
lea     eax, [ebp+Filename]
push    eax         ; lpExistingFileName
call    ds:CopyFileA

```

### Question 3 : instances concurrentes

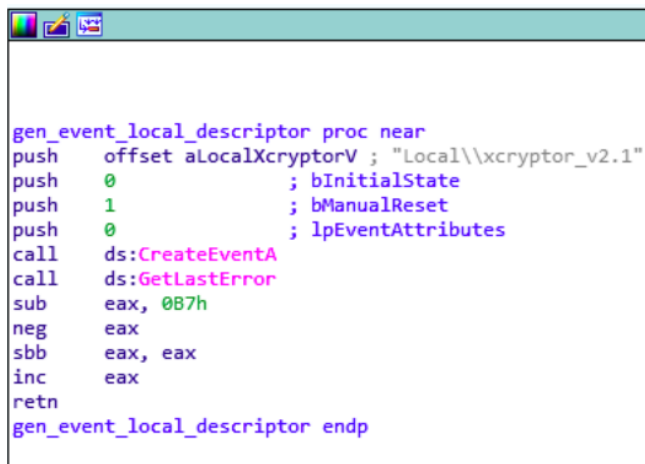
*Deux instances du malware peuvent-elles tourner simultanément ?*

Non. Le malware teste s'il est déjà en train de fonctionner en positionnant et testant un sémaphore avec `CreateEvent` (~ sémaphore Windows nommé).

*et pourquoi ?*

1 - Si l'attaquant ne le faisait pas, son programme empilerait des tâches qui n'auraient pas de valeur ajoutée car elles feraient toutes la même chose. Plusieurs instances lancées en parallèle pourraient attirer l'attention de l'utilisateur légitime.

2 – Les programmes pourraient entrer en conflit en tentant d'intervenir simultanément sur les mêmes fichiers (et aucune mécanique permettant d'éviter les conséquences de ces conflits n'est prévue).



```
gen_event_local_descriptor proc near
push    offset aLocalXcryptorV ; "Local\\xcryptor_v2.1"
push    0                      ; bInitialState
push    1                      ; bManualReset
push    0                      ; lpEventAttributes
call    ds:CreateEventA
call    ds:GetLastError
sub     eax, 0B7h
neg     eax
sbb     eax, eax
inc     eax
retn
gen_event_local_descriptor endp
```

```

mov     edi, 1058h
lea     eax, [ebp+Parameter]
push    edi
xor     esi, esi
push    esi
push    eax
call    sub_403FF0
add     esp, 0Ch
call    gen_event_local_descriptor
cmp     eax, 1
jz      short loc_4012C2

```

```

push    edi
lea     eax, [ebp+Parameter]
push    esi
push    eax
call    sub_403FF0
add     esp, 0Ch
lea     eax, [ebp+SystemTime]
push    eax ; lpSystemTime
call    ds:GetSystemTime
push    esi ; lpThreadId
push    esi ; dwCreationFlags
lea     eax, [ebp+Parameter]
push    eax ; lpParameter
push    offset StartAddress ; lpStartAddress
push    esi ; dwStackSize
push    esi ; lpThreadAttributes
call    ds:CreateThread
mov     esi, eax
test    esi, esi
jz      short loc_4012C2

```

```

push    0FFFFFFFh ; dwMilliseconds
push    esi ; hHandle
call    ds:WaitForSingleObject
push    esi ; hObject
call    ds:CloseHandle

```

```

loc_4012C2:
mov     ecx, [ebp+var_4]
pop     edi
xor     ecx, ebp
pop     esi
call    @__security_check_cookie@4 ; __security_check_cookie(x)
mov     esp, ebp
pop     ebp
retn
start_3 endp

```



```

mov     edi, 1058h
lea     eax, [ebp+Parameter]
push    edi
xor     esi, esi
push    esi
push    eax
call    sub_403FF0
add     esp, 0Ch
call    gen_event_local_descriptor
cmp     eax, 1
jz      short loc_4012C2

```

```

push    edi
lea     eax, [ebp+Parameter]
push    esi
push    eax
call    sub_403FF0
add     esp, 0Ch
lea     eax, [ebp+SystemTime]
push    eax ; lpSystemTime
call    ds:GetSystemTime
push    esi ; lpThreadId
push    esi ; dwCreationFlags
lea     eax, [ebp+Parameter]
push    eax ; lpParameter
push    offset StartAddress ; lpStartAddress
push    esi ; dwStackSize
push    esi ; lpThreadAttributes
call    ds:CreateThread
mov     esi, eax
test    esi, esi
jz      short loc_4012C2

```

```

push    0FFFFFFFFh ; dwMilliseconds
push    esi ; hHandle
call    ds:WaitForSingleObject
push    esi ; hObject
call    ds:CloseHandle

```

```

loc_4012C2:
mov     ecx, [ebp+var_4]
pop     edi
xor     ecx, ebp
pop     esi
call    @_security_check_cookie@4 ; __security_check_cookie(x)
mov     esp, ebp
pop     ebp
retn
start_3 endp

```

## Question 4 : stratégie d'analyse

*Résumer en quelques lignes la stratégie d'analyse que vous allez mettre en place pour venir en aide à Mme Michu (où commencez-vous l'analyse et pourquoi ? Que cherchez-vous?)*

1. Nous cherchons la routine de chiffrement qui agit sur les fichiers.
2. Nous recherchons la ou les clés de chiffrement utilisées
3. Nous identifierons le mode opératoire : les fichiers impactés et les éventuels autres effets (ex : ajouts de fichiers).
4. Nous neutraliserons la capacité du malware à produire ses effets en supprimant ces éléments de persistance (désinstallation).
5. Nous produirons un outil de déchiffrement des fichiers impactés.

Les trois premières étapes relèvent de l'analyse statique et elles doivent être réalisées en préalable afin d'être certain d'avoir identifié le comportement du malware avant toute intervention en remédiation (les deux dernières étapes). Nous devons supposer le malware doté de mécanismes éventuels de défense et lever cette hypothèse dans le cadre de l'analyse statique.

### Comportement attendu (mode opératoire recherché)

On s'attend à trouver un motif d'appels système qui parcourt les fichiers du filesystem ; qui se focalise sur les extensions (voir types de fichiers évoqués à la question 5) ; qui ouvre ces fichiers ; vérifie que le fichier n'a pas déjà été chiffré (pour éviter la surinfection) ; sinon il chiffre le fichier et le ré-écrit sur le disque.

### Méthodologie adoptée pour l'analyse statique (point de départ choisi)

Recherche de la routine d'ouverture de fichiers dans la liste des Imports.

Si l'attaquant n'a pas masqué sa trace, on élimine les faux positifs (il y en avait 3 et il n'en reste qu'un seul).

### Analyse statique

À partir de là, on regarde si l'on trouve le comportement attendu dans la routine appelante :

- C'est le cas dans la routine qui commence en 0x40156B (Encrypt\_file?). Cette fonction n'est appelée que depuis une fonction située en 0x4010C8 (Encrypt\_if\_mimes) dans laquelle on constate la présence d'une boucle qui vérifie l'extension des fichiers évoqués à la question 5.
- Cette fonction Encrypt\_if\_mimes n'est pas appelée directement. Elle est passée en paramètre à une fonction que l'on a appelée ParcourRepertoires en 0x40111A.
- ParcourRepertoires est appelée par le thread démarré à l'adresse StartAddress en 0x4014BC. ParcourRepertoires réalise une boucle parcourant l'ensemble des fichiers à chiffrer (parcours de tous les répertoires à partir des variables d'env. HOMEDRIVE et HOMEPATH). A noter :
  - A chaque entrée dans un nouveau répertoire un appel à Ecrire\_fichiers\_readme en 0x4017FB est réalisé ;
  - Ecrire\_fichiers\_readme produit un fichier texte qui est placé dans le répertoire ;
  - Si le fichier existait déjà il est écrasé, le paramètre dwCreationDisposition ayant été positionné à CREATE\_ALWAYS (2).
- StartAddress (point d'entrée du thread) :
  - Lit les variables d'environnement (HOMEDRIVE et HOMEPATH) ;
  - Mets en place la persistance (Install\_persistence en 0x4012D2) ;
  - Procède à l'initialisation des sous-clés de chiffrement (initKeys en 0x401BF5) ;
  - Lance ParcourRepertoires.

- Et termine le thread (ce qui combiné avec le comportement du thread principal – voir ci-dessous – termine également le thread principal du programme).
- En cherchant les références à StartAddress (0x4014BC), on constate que :
  - Cette adresse est utilisée comme point de départ d'un thread dans un appel à CreateThread
  - Le paramètre passé au Thread est l'adresse d'une structure SYSTEMTIME initialisée par un appel à GetSystemTime. Le contenu de cette structure de 16 octets sera utilisé comme clé principale pour le chiffrement des fichiers.
  - Le thread courant (thread principal du processus) se met en attente du thread fils créé et se termine une fois son fils terminé.
  - L'adresse StartAddress est elle-même appelée depuis le point d'activation principal du programme (starten 0x401FC3 ; appelle start\_2 en 0x4018F8 ; appelle start\_3 en 0x401240 ; appelle StartAddress).

→ Nous avons les éléments suffisants pour comprendre le système de chiffrement.

#### **Analyse de Encrypt\_file (0x40156B).**

- Ouverture du fichier passé en paramètre ;
- GetFileSize : vérification de la taille (<FileSizeHigh) ;
- Lecture du fichier en mémoire ;
- Teste si le fichier commence par la chaîne « GPGcryptor », si oui → abandon du chiffrement (le fichier ayant déjà été chiffré, le malware évite le sur-chiffrement\*).
- Sinon appel de la routine de chiffrement (Chiffrement qui est à l'adresse 0x40107B). On constate que cette routine utilise le mode d'opération CBC avec un IV égal à "@GPCODE\x00" (8 bytes). Rappel CBC :  $iv\_128 \text{ XOR } \text{bloc\_clair\_128} > \text{Chiffrement BlowFish de bloc 128 bits} > \text{bloc\_chiffré\_128}$ , utilisé comme IV pour le bloc suivant ; padding à 128 bits du dernier bloc.
- Ré-écriture du fichier à partir des données chiffrées en commençant par y placer la chaîne GPGcryptor, suivi de la longueur du fichier initial (sur 4 octets). La longueur stockée permettra, lors du déchiffrement, de neutraliser les caractères de padding imposés par le mode d'opération CBC

*\* Note : le malware est contraint de lire (charger en mémoire) tous les fichiers possédant les extensions ad-hoc afin de déterminer s'ils ont déjà été chiffrés. Il prend ensuite grand soin de ne pas provoquer le changement de la date de modification du fichier si aucun nouveau chiffrement n'est nécessaire. En effet une telle modification rendrait vaine toute tentative ultérieure de retrouver la clé de chiffrement pour ce fichier.*

*Une stratégie alternative, parfois rencontrée avec ce type de malware, aurait consisté à changer l'extension du fichier afin de pouvoir déterminer ceux qui sont chiffrés, sans être contraint de les ouvrir. Exemple : ransom.pots (rapporté par Symantec, 25/01/2019) qui ajoute des extensions supplémentaires aux fichiers déjà traités ( .djvu, .rumba, .tfudet, .tro).*

## Question 5 : types de fichiers impactés

Quels sont les types de fichiers impactés ?

```
ata:0041B080 mimes_type dd offset aDoc ; DATA XREF: sub_4010C8:loc_4010E7f
ata:0041B084 ; ".doc"
ata:0041B088 dd offset aDocx ; ".docx"
ata:0041B08C dd offset aPpt ; ".ppt"
ata:0041B090 dd offset aPptx ; ".pptx"
ata:0041B094 dd offset aXls ; ".xls"
ata:0041B098 dd offset aXlsx ; ".xlsx"
ata:0041B09C dd offset aRtf ; ".rtf"
ata:0041B0A0 dd offset aPdf ; ".pdf"
ata:0041B0A4 dd offset aJpg ; ".jpg"
ata:0041B0A8 dd offset aJpeg ; ".jpeg"
ata:0041B0AC dd offset aHtml ; ".html"
ata:0041B0B0 dd offset aHtm ; ".htm"
ata:0041B0B4 dd offset aPng ; ".png"
ata:0041B0B8 dd offset aGif ; ".gif"
ata:0041B088 align 10h
```

→ La fonction EncryptFile n'est appelée que pour les mimes\_type cités. Tous les fichiers personnels de types bureautiques, images et pages html sont impactés.

(.doc, .docx, .ppt, .pptx, .xls, .xlsx, .rtf, .pdf, .jpg, .jpeg, .html, .htm, .png, .gif)

→ cf. méthodologie adoptée : fonction située en 0x4010C8(Encrypt\_if\_mimes).

## Question 6 : fichiers chiffrés une seule fois

Expliquez pourquoi les fichiers ne sont transformés qu'une seule fois même si le malware s'exécute plusieurs fois.

Voir question 4 : abandon du chiffrement si le fichier est déjà chiffré, il commence alors par la chaîne « GPGcryptor ».

Cela permet d'éviter de re-chiffrer un fichier déjà modifié et de perdre le seul indice disponible pour remonter à la clé de chiffrement du fichier (la date de modification du fichier). Conceptuellement, surchiffrer ne présente pas d'avantage pour l'attaquant.

## Question 7 : type de transformations subies

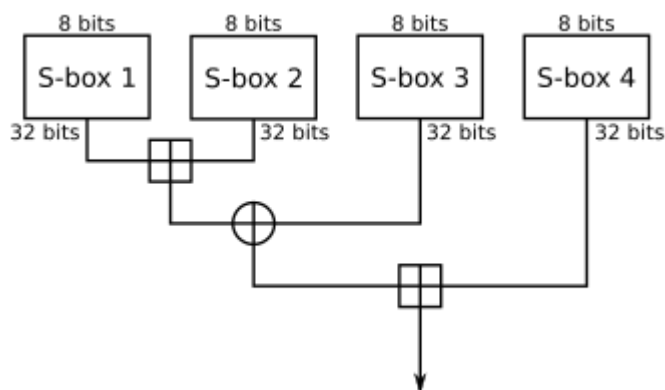
Caractérisez le type de transformations qu'ont subi les fichiers (vous pouvez vous aider d'un moteur de recherche)

NB : avant l'appel de la brique cryptographique ; il y a une transformation endianness.

Une fois que l'on a trouvé la primitive de chiffrement unitaire, on regarde où elle est utilisée. On s'aperçoit qu'il y a une autre routine qui l'utilise aussi (autre\_chiffrement\_XXXXX en 0x401BF5).

Dans cette routine, on repère des constantes caractéristiques qui valent 0xD1310BA6 et 0x243F6A88

→ En recherchant ces constantes<sup>2</sup>, elles sont associées à l'algorithme BLOWFISH, ce qui est cohérent avec les 16 tours relevés dans le code.



<sup>2</sup> <https://www.google.com/search?q=D1310BA6+243F6A88>

On l'appelle `autre_chiffrement_blowfish`. Cette fonction est appelée avec un paramètre qui est la zone de paramètres du thread :

- zone Parameter déclarée dans la pile, de taille 0x1058, initialisée à 0 ;
- à partir de 0x1048 se trouve une zone permettant de recevoir l'heure système ;
- on en déduit une structure avec deux champs (un de taille 0x1048 et un de taille 0x10) ;

On retrouve ici les caractéristiques d'un contexte de chiffrement Blowfish (Structure de 0x1048 octets) dont les S-box sont initialisées en faisant intervenir la clé racine (structure de 0x10 octets).

## Question 8 : restauration des fichiers

*Expliquez pourquoi il est possible de restaurer les fichiers d'origine.*

Le comportement du malware est décrit ci-dessus. En synthèse :

- la date/heure courante est demandée au système à la création du processus ;
- cette date/heure est utilisée comme clé de chiffrement pour l'ensemble des fichiers traités, cette clé n'est pas conservée d'une exécution du malware à l'autre ;
- les répertoires sont parcourus pour chercher des fichiers nouveaux à chiffrer.

Ce qui implique que certains fichiers qui auront été chiffrés vont avoir une date/heure de modification assez proche de la date/heure ayant servi de clé.

Il suffit donc de partir de la date/heure de modification d'un fichier chiffré et de décrémenter cette valeur par pas de 1ms (ce qui est la granularité de la clé) et de tester si le résultat obtenu par le déchiffrement avec cette clé correspond à un clair possible.

Un critère d'arrêt est nécessaire, il va consister à regarder si les premiers caractères (MAGIC) des clairs potentiels obtenus correspondent à ceux possibles dans un fichier correspondant au type (déterminé ici via l'extension du nom de fichier) :

"PK\x03\x04",	// docx pptx xlsx
"PK\x05\x06",	// doc ppt xls
"\xd0\xcf\x11\xe0\xa1\xb1\xa1\xe1",	// pdf
"%PDF",	// gif
"GIF8",	// png
"\x89PNG", "RIFF",	// jpg jpeg
"\xff\xd8\xff",	// rtf
"{\rtf1",	

Il convient d'ajouter un critère d'arrêt spécifique pour les fichiers .htm/.html, qui n'ont pas de MAGIC. On a choisi de vérifier que la première dizaine de caractère du fichier correspond à de l'ASCII. Il faudrait également vérifier la présence éventuelle d'un BOM en début de clair, et tester également les caractères UTF-xx.

Dès lors qu'une clé aura été trouvée, elle devra être essayée sur tous les fichiers suivants. En effet tous les fichiers chiffrés lors d'une même exécution du malware l'auront été avec la même clé.

Si des fichiers candidats ont été introduits après la dernière exécution du malware, ils auront été chiffrés avec une clé différente. Il convient donc de mémoriser l'ensemble des clés retrouvées et d'effectuer deux passes sur l'ensemble du répertoire pour être certain\* de récupérer l'ensemble des fichiers.

*\* sauf si des fichiers ont été ouverts et réécrits par un programme tiers qui aura changé leur date de modification au-delà de l'horizon que l'on se sera fixé pour conserver une durée raisonnable au traitement. L'horizon a été fixé expérimentalement à 1000 millisecondes (1000 essai de clé au maximum avant d'abandonner la recherche de la clé pour un fichier).*

## Question 9 : technique à mettre en œuvre

*Comment s'appelle la technique que vous allez mettre en œuvre ?*

Attaque de la clé de chiffrement à clair et chiffré connus, méthode de brut forçage à partir d'une valeur proche de la clé.

## Question 10 : outil de suppression du cryptovirus

*Écrire un programme (dans le langage que vous voulez) permettant de supprimer le malware de l'ordinateur de Mme Michu.*

Les étapes suivantes sont implémentées dans antisudoku.exe:

- Suppression du fichier pointé par la clé de registre \_SuDoku et suppression de la clé ;
- Vérification de la présence éventuelle d'un fichier de même signature dans le répertoire startup, suppression de ce fichier ;
- Recherche du fichier vecteur d'infection initial et de ses clones (on les reconnaît à leur hash) et suppression.

## Question 11 : outil de déchiffrement

*Écrire un programme (dans le langage que vous voulez) permettant de restaurer au moins un type de fichiers de Mme Michu (le programme peut se limiter à un seul fichier donné en argument, Mme Michu pouvant itérer elle-même sur tous ses fichiers).*

L'outil implémentant la suppression du cryptovirus (suppression de la persistance et des exécutables) et le déchiffrement des fichiers est disponible sur <https://github.com/wvmcs/be-sudoku2>.

### Exemple d'exécution

ANTISUDOKU: Restauration des fichiers chiffrés par le malware Sudoku.exe  
(identifié par Windows Defender comme Trojan:Win32/Sprisky.U!cl)

Actions exécutées:

- suppression de la valeur \_SuDoku de la clé de registre HKEY\_CURRENT\_USER\Software\Microsoft\Windows\CurrentVersion\Run
- suppression de toutes les copies du programme Sudoku.exe et de ses clones
- suppression des fichiers texte de demande de rançon
- déchiffrement des fichiers personnels

Point de départ pour le parcours des répertoires: C:\Users\IEUser\Downloads

log d'exécution

\*.exe: suppression de C:\Users\IEUser\Downloads\16268-7772-24292\_.exe

REG: clé \_SuDoku supprimée

\*.exe: suppression de C:\Users\IEUser\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\22087-5158-24341\_.exe

\*.exe: suppression de C:\Users\IEUser\Downloads\Sudoku.exe

Cyphered: C:\Users\IEUser\Downloads\cantadvortex.png (461942) TROUVE key= 11/03/2019-10:14:19:436

Cyphered: C:\Users\IEUser\Downloads\Facture FC0017 - AOUT 2018 - CL0001.pdf (35214) TROUVE key= 11/03/2019-10:14:19:436

Cyphered: C:\Users\IEUser\Downloads\html.jpg (68254) TROUVE key= 11/03/2019-10:14:19:436

Cyphered: C:\Users\IEUser\Downloads\pots.gif (470862) TROUVE key= 11/03/2019-10:14:19:436

Cyphered: C:\Users\IEUser\Downloads\test1.xls (26142) TROUVE key= 11/03/2019-10:14:19:436

Cyphered: C:\Users\IEUser\Downloads\test1.xlsx (8558) TROUVE key= 11/03/2019-10:14:19:436

#### Statistiques

Nombre de fichiers trouvés chiffrés	6
Nombre de fichiers déchiffrés avec succès	6
Nombre de répertoires parcourus	1
Nombre de fichiers README.txt supprimés	1
Nombre de copies du programme SUDOKU.exe supprimées	3
Nombre de fichiers qui n'ont pû être contrôlés (Erreurs)	0