

QUAND UN RANSOMWARE DEVIENT UN KEYGENME

Nicolas Brulez – nicolas.brulez@kaspersky.fr – Senior Security Researcher -
Global Research and Analysis Team – Kaspersky Lab –
<http://www.kaspersky.com> – <http://www.viruslist.com>

mots-clés : CODES MALICIEUX / REVERSE ENGINEERING / RANSOMWARE /
ANALYSE DE CODE / SMS

Dans le numéro 46 de MISC, je vous présentais l'analyse d'un ransomware (application qui modifie une machine pour ensuite demander une rançon au propriétaire avant de la remettre dans l'état initial) qui chiffrait les fichiers afin d'extorquer les utilisateurs.

Dans ce numéro, je vais vous présenter un autre type de ransomware, détecté en novembre 2009 sous le nom de Trojan-Ransom.Win32.SMSer.qm
MD5: 2CE9D15F7B43B0DEC6C3935DE0743113

1 Analyse du ransomware

Aucun packer d'exécutables n'a été utilisé pour « protéger » le code du malware, il est alors possible de l'analyser directement dans IDA.

Lors de la première exécution, notre code malicieux génère un « Mutex » au nom de « {4F79CBDE-4C87-4af2-8114-7AD1420AEC0E} » puis modifie la base de registre pour s'exécuter automatiquement au lancement de Windows :

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\WinLogon\.

UserInit est modifié afin d'exécuter le ransomware à chaque démarrage : **C:\WINDOWS\system32\userinit.exe, "CHEMIN malware.exe"**

Il crée ensuite un fichier **.den** dans son répertoire qui contient des informations telles que le code d'extorsion, que nous découvrirons par la suite.

Ensuite, le malware s'exécute une seconde fois dans le but de protéger le processus parent et le relancer si celui-ci était arrêté par l'utilisateur qui se serait rendu compte de l'infection.

Pour cela, le processus enfant vérifie la présence du Mutex et relance le malware en cas de Mutex introuvable.

A partir de maintenant, lors du prochain redémarrage de Windows, une fenêtre apparaîtra sur l'écran de la machine infectée : voir Figure 1.

On peut lire :

L'accès Internet est bloqué pour des raisons de violation de licence du programme uFast download manager.

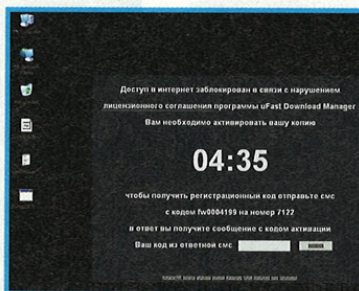
Vous devez « activer » votre copie.

Afin d'obtenir le code d'enregistrement, envoyez un SMS au numéro 7122 avec le code fw0004199 et vous recevrez le code d'activation.

Votre code d'activation reçu par SMS - [Champ]

Pour pouvoir réutiliser Internet, il faut envoyer un SMS à un numéro surtaxé afin d'obtenir le code de désactivation. L'argent est donc récupéré à l'aide d'un service SMS payant.

L'accès internet est véritablement désactivé par notre code malicieux. Après analyse du code à l'aide d'IDA Pro, il est possible de déterminer la technique employée : l'utilisation de fonctions **SetupDi*** de la DLL **SETUPAPI.dll**



Machine infectée

```

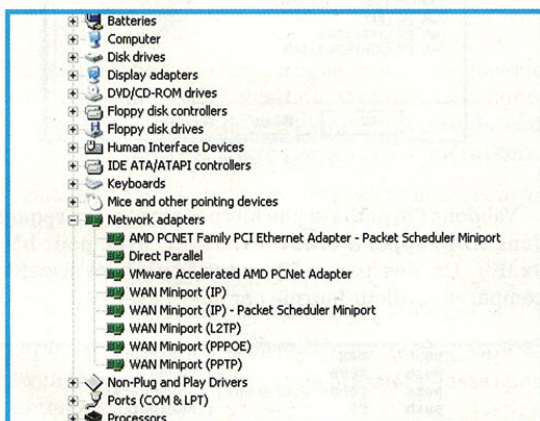
push    14h                ; CODE XREF: sub_40CEA3+4F1j
lea     eax, [ebp+1C8h+ClassInstallParamsSize]
push    eax                ; ClassInstallParams
push    [ebp+1C8h+var_234] ; DeviceInfoData
mov     ebx, 2              ; DeviceInfoSet
push    ebx                ; DeviceInfoSet
call    ds:SetupDiSetClassInstallParamsW
test    eax, eax
jz      short loc_40CF82
push    [ebp+1C8h+var_234] ; DeviceInfoData
push    ebx                ; DeviceInfoSet
push    edi                ; InstallFunction
call    ds:SetupDiCallClassInstaller
test    eax, eax
jz      short loc_40CF82
lea     eax, [ebp+1C8h+DeviceInstallParams]
push    eax                ; DeviceInstallParams
push    [ebp+1C8h+var_234] ; DeviceInfoData
mov     ebx, [ebp+1C8h+DeviceInstallParams.cbSize], 22Ch
push    ebx                ; DeviceInfoSet
call    ds:SetupDiGetDeviceInstallParamsW ; Désactive le device
test    eax, eax

```

Routine de désactivation du réseau

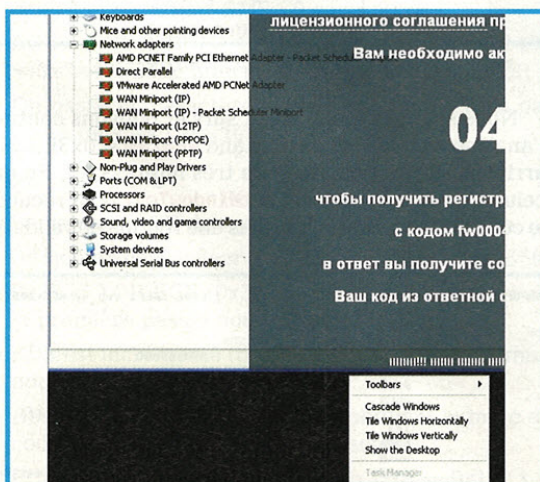
En utilisant la fonction **SetupDiClassGuidsFromNameExW** et la **class: net**, notre malware liste tous les devices réseaux et les désactive un par un. Il est important de lister les devices cachés pour tous les voir et les réactiver à la main :

- Avant infection :



Matériel : machine non infectée

- Après infection :



Matériel : machine infectée

2 Code d'activation

Même s'il est trivial de désinfecter ce malware et de retrouver une connexion internet, il est tout de même intéressant d'analyser la partie qui vérifie le code d'activation pour voir si le malware se désinstalle bien après obtention d'un code par SMS. Localiser la routine qui vérifie l'entrée du code est assez simple, voici la démarche.

Tout d'abord, il est nécessaire d'entrer un code et de le valider à l'aide d'un bouton pour désactiver la machine. A partir de là, il y a deux façons de procéder. La première est plus simple : puisque l'application va récupérer le texte entré au clavier, il est possible de chercher les fonctions de l'API windows qui auraient pu être utilisées. Effectivement, en quelques secondes, on retrouve un **GetWindowTextA** juste avant l'algorithme de validation.

Il existe de nombreuses façons de récupérer le texte entré, alors une deuxième technique plus sûre est possible : analyser l'interface et trouver le *handler* (gestionnaire) du bouton de vérification.

La première chose à faire est de localiser l'appel à **RegisterClassEx** pour identifier la windows proc. Vous pouvez voir le paramètre **LpfnWndProc** en rouge, ainsi que son adresse :

```

push    esi                ; hInstance
mov     [ebp+var_38.cbSize], 30h
mov     [ebp+var_38.style], 803h
mov     [ebp+var_38.lpfnWndProc], offset sub_4058EF
mov     [ebp+var_38.cbClsExtra], esi
mov     [ebp+var_38.cbWndExtra], esi
mov     [ebp+var_38.hInstance], eax
mov     [ebp+var_38.hIcon], esi
call    ds:LoadCursorW
mov     [ebp+var_38.hCursor], eax
mov     eax, [ebp+var_4]
mov     [ebp+var_38.lpszClassName], eax
lea     eax, [ebp+var_38]
push    eax                ; WNDCLASSEXW *
mov     [ebp+var_38.hbrBackground], 8
mov     [ebp+var_38.lpszMenuName], esi
mov     [ebp+var_38.hIconSm], esi
call    ds:RegisterClassExW

```

Récupération de la WndProc

A partir de là, on analyse le fonctionnement de la fenêtre et plus précisément du bouton de validation. Il suffit de double-cliquer sur l'adresse de la WinProc pour obtenir ceci :

```

push    ebp
lea     ebp, [esp-824h]
sub     esp, 8A4h
mov     eax, dword_41775C
xor     eax, ebp
mov     [ebp+824h+var_4], eax
mov     edx, [ebp+824h+WM]
cmp     edx, 0Fh           ; EDX = WM
push    esi
push    edi
mov     edi, [ebp+824h+hWndParent]
ja      above_0F

```

WndProc : Gestion des WM

La partie importante ici est le Windows Message (WM) passé dans EDX. Nous sommes intéressés par un message de type **WM_COMMAND** (utilisé lorsque l'on clique

sur des contrôles de la fenêtre, par exemple), et ce type de message a une valeur supérieure à 0xF. Nous pouvons donc suivre le JA pour nous rendre ici :

```

above_0F:
    mov     ecx, edx             ; CODE XREF: sub_4058EF+2C7j
    mov     eax, WM_COMMAND      ; ECX = EDX = WM
    sub     ecx, eax             ; EAX = WM_COMMAND
    jz      WM_COMMAND_received ; Oui.
    dec     ecx
    dec     ecx
    jz      loc_405B14

```

WndProc : WM_COMMAND ?

La suite parle d'elle-même. Lorsque le message **WM_COMMAND** est reçu, l'exécution du code suit son cours ici (Premier JZ exécuté) :

```

WM_COMMAND_received:
    mov     edx, [ebp+824h+uParam] ; CODE XREF: sub_4058EF+1057j
    movzx   ecx, dx
    sub     ecx, 3E8h
    jz      short uParam_3E8
    dec     ecx                 ; Décrémente. EDX était >= 0x3E9
    jz      short uParam_3E9
    push    [ebp+824h+1Param] ; 1Param
    push    edx                ; uParam
    push    eax                ; Hsg

```

WndProc : Gestion du WM_COMMAND

Pour finir, nous sommes en présence d'un dernier test : le **wParam**. Pour simplifier, il a pour but de vérifier qui a reçu le message **WM_COMMAND**.

La première vérification traite 0x3E8 et exécute la routine que j'ai appelée **wParam_3E8** en cas de test positif.

S'en suit une décrémentation de ECX, et une autre vérification. Si le résultat de la décrémentation est zéro, alors nous exécutons la routine **wParam_3E9** (0x3E9 - 0x3E8 lors de la première vérification résulterait un 0x1, qui une fois décrémenté donnerait un zéro, exécutant le JZ **wParam_3E9**.)

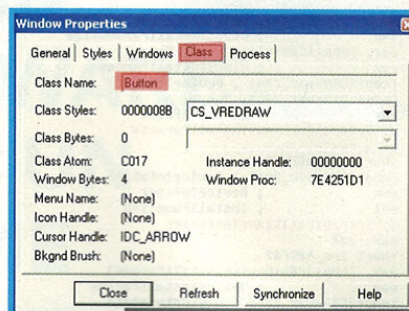
Pour savoir quel **wParam** nous intéresse, les imports de notre application (en particulier **CreateWindow**) fournissent des informations précieuses. Il est en effet possible de trouver la création des contrôles et de vérifier le type de classe employé : **Button**, **Edit**, etc.

En utilisant un débogueur, par exemple, on place un point d'arrêt sur les appels à **CreateWindowEx** pour retrouver les classes et les hMenu utilisés. On s'aperçoit ici que le **hMenu** est 0x3E8 et **Class = EDIT**. Cela correspond à notre **input box** :

PUSH EAX	hInst
PUSH 3E8	hMenu = 000003E8
PUSH DWORD PTR SS:[EBP+8]	hParent
MOV EBX, 17E	
PUSH 19	Height = 19 (25.)
PUSH 78	Width = 78 (120.)
PUSH EBX	Y = 17E (382.)
PUSH 154	X = 154 (340.)
PUSH 50000040	Style = WS_CHILD WS_VISIBLE 40
PUSH 2ce9d15f.004126a4	WindowName = ""
PUSH EDI	Class = "EDIT"
MOV EDI, DWORD PTR DS:[<USER32.CreateWin	USER32.CreateWindowExW
PUSH 0	ExtStyle = 0
CALL EDI	CreateWindowExW

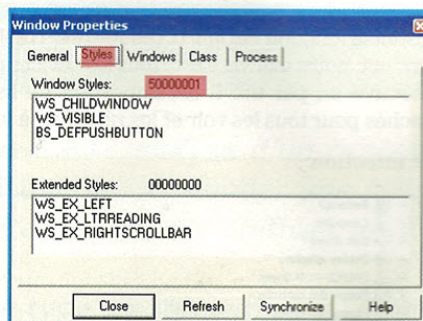
Vérification du type de « Class »

Nous savons maintenant que le 0x3E9 est sûrement notre bouton de validation, puisque 0x3E8 est l'input box. Il est possible de vérifier à l'aide de l'outil **Spy++** en cherchant des informations sur notre bouton :



« Class » dans Spy++

Pour le style :



« Styles » dans Spy++

Validons l'hypothèse une fois pour toutes, en regardant dans IDA l'appel à **CreateWindowEx** avec pour hMenu 0x3E9. Un des paramètres est **Styles** et il suffit de comparer la valeur fournie par **spy++** avec celle du code :

```

push     eax                ; hInstance
push     3E9h               ; hMenu
push     [ebp+hWndParent] ; hParent
push     25                 ; nHeight
push     80                 ; nWidth
push     ebx                ; Y
push     480                ; X
push     50000001h          ; dwStyle
push     [ebp+lpWindowName] ; lpWindowName
push     [ebp+var_14]        ; lpClassName
push     0                  ; dwExStyle
call     edi ; CreateWindowExW

```

Création du bouton

Nous sommes maintenant sûrs et pouvons continuer l'analyse du code exécuté quand **wParam = 0x3E9**. Nous arrivons sur un bout de code très intéressant. En effet, celui-ci appelle la fonction **GetWindowText** pour récupérer le code de validation entré, puis une routine de validation :

```

uParam_3E9:
    shr     edx, 10h          ; CODE XREF: sub_4058EF+2907j
    test    dx, dx
    jnz     short uParam_3E8
    push    40h               ; nMaxCount
    lea     eax, [ebp+824h+Activation_code]
    push    eax               ; lpString
    push    hWnd              ; hWnd
    call    ds:GetWindowTextA ; Récupère le code entré.
    lea     eax, [ebp+824h+Activation_code]
    push    eax               ; input
    push    edi               ; hWnd
    call    Valid_Activation_Code ; Routine de validation

```

Handler du bouton



3 Analyse de la routine de validation

Je passerai certains détails pour aller droit au cœur de l'algorithme. Au début de l'article, je mentionnais un nouveau fichier .den qui contenait le code à envoyer par SMS. Celui-ci est utilisé dans un algorithme très simple, pour confirmer le code de validation.

Ce code est tout d'abord utilisé pour gérer un Validation ID qui sera ensuite comparé au code de validation entré par la personne qui désire débloquent sa machine. Voici l'algorithme en question :

```
text:0040C399
text:0040C399 loc_40C399:
text:0040C399 ; CODE XREF: sub_40C35A+561j
text:0040C399 movzx edi, [ebp+edx+var_10] ; [address] points to ransom code
text:0040C39C add edi, 250Fh
text:0040C3A4 xor edi, ecx
text:0040C3A6 inc edx
text:0040C3A7 cmp edi, esi
text:0040C3A9 lea ecx, [ecx+edi+1402h]
text:0040C3B0 j1 short loc_40C3B2
text:0040C3B2 loc_40C3B2:
text:0040C3B2 ; CODE XREF: sub_40C35A+301j
text:0040C3B2 cmp ecx, eax ; Does it match entered input?
text:0040C3B4 mov ecx, [ebp+var_A]
text:0040C3B7 setz al
```

Algorithme validation

J'en vois déjà certains rire après lecture de ces quelques lignes. La génération du Validation ID se fait en six lignes, très simples à comprendre. Chaque caractère du code à envoyer par SMS est traité par diverses opérations.

Explication avec le code : fw0004199 (voir screenshot au début d'article) :

- Caractère « f »

- La valeur ASCII de « f » est 0x66 ;
- On ajoute à cette valeur 0x25DF pour obtenir : 0x2645 ;
- 0x2645 XOR ECX (0 lors de la première passe) pour obtenir : 0x2645 ;
- EDX est incrémenté (position du caractère courant, pour passer à « w ») ;
- EDX et ESI sont comparés. ESI contient la taille du code à envoyer par SMS : 9 caractères ;
- $ECX = ECX + EDI + 0x1402 = 0 + 0x2645 + 1402 = 0x3A47$. Code intermédiaire utilisé par le XOR ;
- On passe au caractère suivant si nous n'avons pas traité tous les caractères.

- Caractère « w »

- La valeur ASCII de « w » est 0x77 ;
- On ajoute à cette valeur 0x25DF pour obtenir : 0x2656 ;
- 0x2656 XOR ECX (0x3A47 - code intermédiaire de la première passe) pour obtenir : 0x1C11 ;
- EDX est incrémenté (position du caractère courant, pour passer à « 0 ») ;
- EDX et ESI sont comparés. ESI contient la taille du code à envoyer par SMS : 9 caractères ;
- $ECX = ECX + EDI + 0x1402 = 0x3A47 + 0x1C11 + 1402 = 0x6A5A$. Code intermédiaire utilisé par le XOR ;

- On passe au caractère suivant si nous n'avons pas traité tous les caractères.

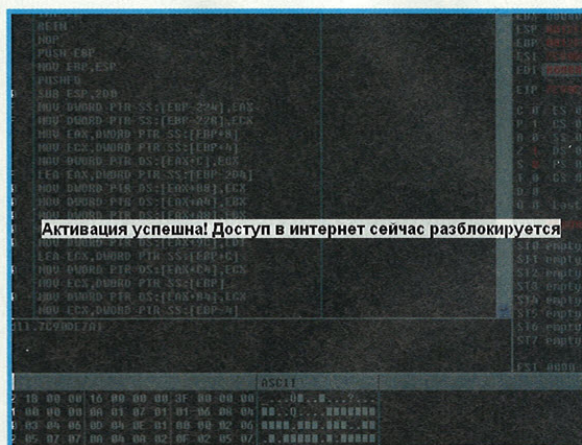
- Etc. pour tous les caractères.

A la fin de la boucle, $ECX = 0x3FCC5E$. ECX est comparé à $EAX = 075BCD15$.

Nous savons d'où vient la valeur ECX, mais pas celle de EAX, probablement du code entré pour la validation. En effet, la valeur d'EAX est la représentation hexadécimale du code entré. $075BCD15 = 123456789$ que j'ai entré pour la rédaction de l'article.

Si les deux valeurs sont identiques, alors le ransomware pense que nous avons payé par SMS, et il va ensuite nettoyer la machine. Le code à entrer sur ma machine de test est donc la représentation décimale de la valeur d'ECX, soit 4181086.

Une fois le code entré, nous obtenons ceci :



Activation terminée

Traduction : l'activation s'est terminée avec succès. L'accès internet va se débloquent.

Notre ransomware débloquent la connexion internet, se supprime et nettoie la base de registre comme prévu.

Conclusion

Pour terminer l'article, j'insisterais une fois de plus sur le fait que la majorité des ransomwares n'utilisent pas d'algorithmes sûrs et qu'il est donc inutile de paniquer (excepté quelques cas, comme Gpcode). Il existe des dizaines de souches de la même famille avec ce genre de pratique, ayant des interfaces plus ou moins différentes (fond d'écran rouge avec création d'un nouveau bureau, souris désactivée, etc.). Merci de ne pas payer...

Remerciements à Natalia pour la traduction des messages en russe :-)