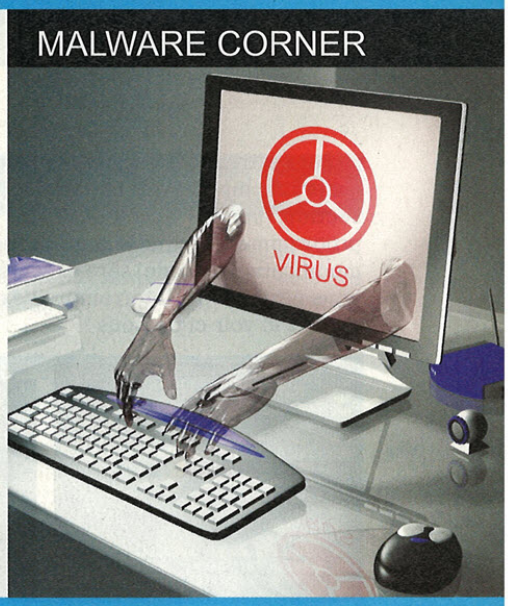
ROGUE AV : UTILISATION DU GESTIONNAIRE DES TÂCHES POUR EFFRAYER LES UTILISATEURS

Nicolas Brulez – nicolas.brulez@kaspersky.fr
Senior Malware Researcher – Global Research and Analysis Team
Kaspersky Lab France



mots-clés : CODES MALICIEUX / REVERSE ENGINEERING / ROGUE / ANALYSE DE CODE / FAKEAV / SCAREWARE / INJECTION

Les faux antivirus (Rogue AV) sont présents depuis de nombreuses années et tentent d'effrayer les utilisateurs en leur faisant croire que leurs machines sont infectées. Le FBI affirme que le montant des fraudes est supérieur à 150 Millions de dollars, et pour obtenir une telle somme, les cybercriminels sont sans cesse à la recherche de techniques alarmantes et effrayantes pour les utilisateurs.

Cet article présente une technique récente utilisée pour les convaincre d'acheter leur antivirus factice.

Une DLL utilisant un *packer* (*loader*) polymorphique (pour ralentir l'analyse et empêcher la création d'une simple signature) est injectée dans le gestionnaire des tâches (**taskmgr.exe**) pour afficher des informations complémentaires sur les processus en mémoire. Nous allons voir en détails comment fonctionne notre DLL.

1 Le packer/loader

Notre Rogue n'utilise pas un packer à proprement parler, c'est-à-dire que le fichier à protéger n'est pas modifié directement. Alors qu'un packer classique modifie l'application à chiffrer en ajoutant une routine de déchiffrement directement dans celle-ci, nous sommes ici en présence d'un autre type de protection. L'exécutable original n'est pas modifié, mais incorporé dans un autre programme loader, dont le seul but est de charger et d'exécuter l'application à protéger, en mémoire seulement. A aucun moment, l'application originale n'est copiée sur le disque. La coquille est polymorphique et change donc à chaque fois qu'une application est protégée.

C'est ainsi que tous les composants du Rogue sont protégés. Voici par exemple le point d'entrée de l'un d'eux :

```
; BOOL __stdcall DllEntryPoint(HINSTANCE hinstDLL, DWORD
                public DllEntryPoint
DllEntryPoint   proc near

hinstDLL        = dword ptr  4
fdwReason       = dword ptr  8
lpReserved      = dword ptr  0Ch

                mov     ds:dword_10017DA8, ecx
                push    ebx
                pop     ds:dword_10017D78
                push    edx
                pop     ds:dword_10017C53
                mov     ds:dword_10017C4B, edi
                push    esi
                pop     ds:dword_10017D6A
                lea     edi, unk_10017BB8
                push    edi
                mov     ch, ds:byte_100163D4
                mov     ds:dword_10016018, edx
                pop     eax
                add     byte ptr ds:dword_10016048, 82h
                push    ebp
                mov     ecx, 595Eh
                shr     edx, 1
                mov     ds:dword_1001603C, 0A3Fh
                pop     dword ptr [eax]
                sub     esi, 5D11h
                mov     ds:dword_10016010, ebx
                mov     ds:dword_10016040, 1BCEh
                push    ecx
                lea     esi, [eax+60DEh]
                inc     ecx
                push    esi
                adc     edi, 6251h
                shl     esi, 3
                call    sub_100020BE
                jmp     loc_1000947E
DllEntryPoint   endp
```

Fig. 1 : Point d'entrée d'un fichier « packé »

Pour extraire les fichiers originaux, il est nécessaire de poser un point d'arrêt matériel au début de la première section (sur les accès écriture) et d'exécuter notre programme. Lorsque la routine de déchiffrement (ou de modification de code) sera exécutée, notre point d'arrêt nous permettra d'interrompre l'exécution, comme vous pouvez le voir ci-dessous :

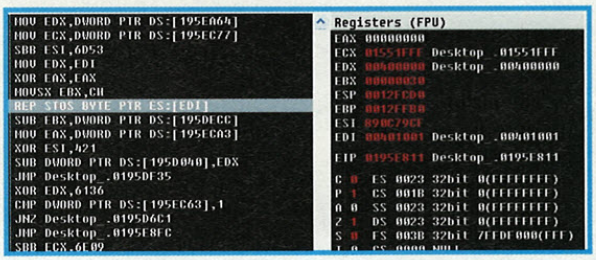


Fig. 2 : Routine d'écrasement des sections

Nous sommes en présence d'une routine d'écrasement. Les deux premières sections (CODE et DATA) se voient remplies de 0x0. Cette opération s'avère inutile, puisque quelques instructions plus tard, ce même emplacement (ainsi que les *headers* du fichier) est écrasé par une nouvelle routine. Pendant l'exécution de notre programme polymorphe, celui-ci a alloué de la mémoire pour y déchiffrer l'application originale :

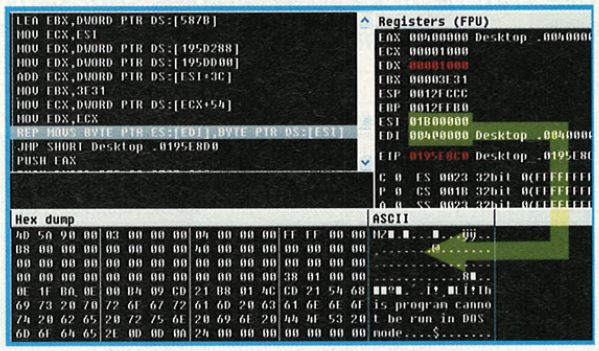


Fig. 3 : Routine de copie du fichier original

La routine d'écrasement utilise les registres ESI et EDI, respectivement les *buffers* source et destination. Le buffer source contient l'application originale déchiffrée, alors que le second n'est autre que le début du programme polymorphique, qui s'apprête à être écrasé. A ce stade, nous avons la possibilité d'enregistrer l'application originale sur le disque en dumpant cette adresse mémoire. Il n'est pas nécessaire de continuer l'exécution du loader. En effet, celui-ci va maintenant charger et exécuter le *malware* en mémoire. Il est préférable de dumper avant l'altération de celui-ci.

```
DllEntryPoint   proc near

var_1C          = dword ptr -1Ch
ms_exc          = CPPEH_RECORD ptr -18h
hinstDLL        = dword ptr  8
fdwReason       = dword ptr  0Ch
lpvReserved     = dword ptr  10h

                push    0Ch
                push    offset stru_10007270
                call    __SEH_prolog
                xor     eax, eax
                inc     eax
                mov     [ebp+var_1C], eax
                mov     esi, [ebp+fdwReason]
                xor     edi, edi
                cmp     esi, edi
                jnz     short loc_10001D7E
                cmp     dword_10009E3C, edi
                jz      loc_10001E31
```

Fig. 4 : Point d'entrée du fichier « unpacké »

L'autre avantage est qu'il n'est pas nécessaire de reconstruire les imports, le fichier est directement opérationnel et analysable à l'aide d'un désassembleur : voir Figure 4.

2 Injection du gestionnaire de tâches

Lorsqu'il s'agit d'effrayer les utilisateurs, les faux antivirus nous ont habitués à l'affichage de pop-ups alarmistes :

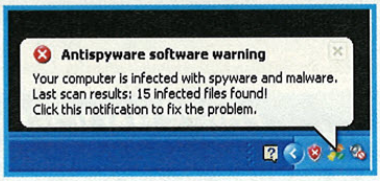


Fig. 5 : Notification d'infection

ou encore :

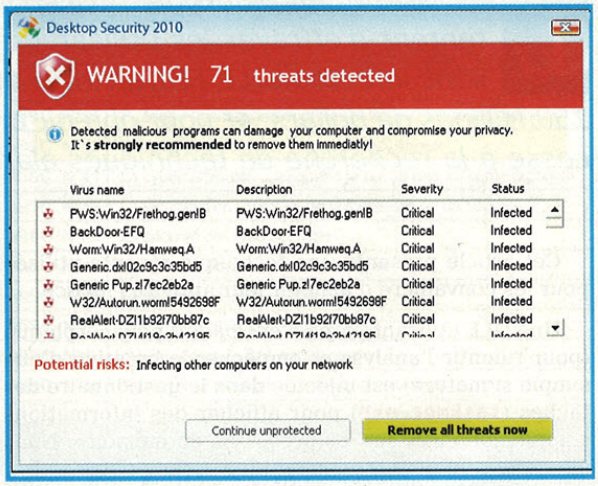


Fig. 6 : Résultats du scan du faux antivirus

Je vais maintenant présenter une technique récente utilisée en plus des pop-ups alarmistes présentés ci-dessus. Le faux antivirus Desktop Security 2010 utilise le gestionnaire de tâches de Windows (`taskmgr.exe`) pour effrayer les utilisateurs. En effet, une DLL est injectée dans ce processus lorsqu'il est exécuté pour afficher des informations factices : voir Figure 7.

Les mots « *infected* » et « *virus free* » ont été ajoutés en face des processus. Nous allons maintenant voir comment notre malware exécute cette opération.

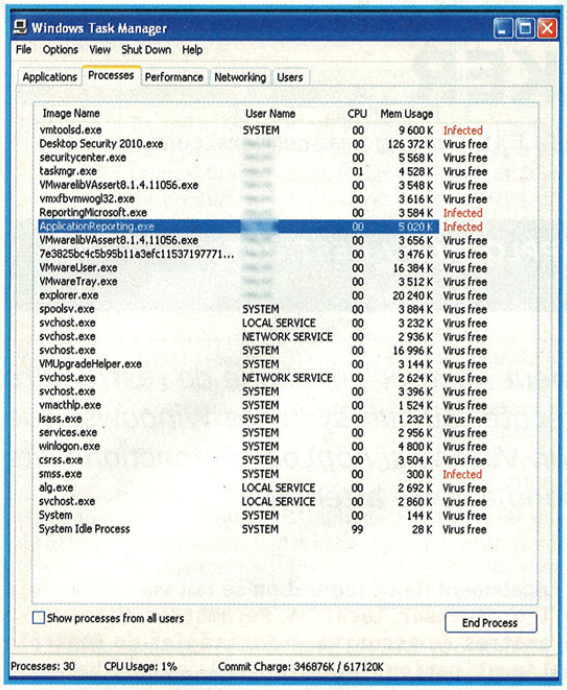


Fig. 7 : Gestionnaire des tâches après injection

L'exécutable principal de notre malware, « **Desktop Security 2010.exe** », énumère les processus à la recherche de **taskmgr.exe**. Une fois le processus trouvé, celui-ci se voit allouer de la mémoire à l'aide de la fonction **VirtualAllocEx**, qui contiendra le chemin complet de la DLL malicieuse du malware. En général : **C:\Documents and Settings\%USERNAME%\Application Data\Desktop Security 2010\taskmgr.dll.**

```
push    offset String1  ; "taskmgr.exe"
call    Search_Process_Name
pop     ecx
mov     [ebp+dwProcessId], eax
cmp     [ebp+dwProcessId], 0
jz      short loc_457305
push    [ebp+dwProcessId] ; dwProcessId
push    0               ; bInheritHandle
push    43Ah            ; dwDesiredAccess
call    ds:OpenProcess
mov     [ebp+hObject], eax
cmp     [ebp+hObject], 0
jz      short loc_457300
lea     eax, [ebp+Buffer]
push    eax             ; lpBuffer
push    [ebp+hObject]   ; hProcess
call    allocate_memory_and_inject_dll_string
```

Fig. 8 : Routine de recherche du taskmgr

Une fois le chemin injecté dans le gestionnaire des tâches, notre malware utilise **CreateRemoteThread**. Au lieu d'injecter un bout de code responsable du chargement de la DLL, l'adresse de **LoadLibraryA** est passée directement en paramètre de **CreateRemoteThread**. Le paramètre du *thread* est le pointeur vers le chemin de la DLL, ce qui a pour résultat de charger la DLL dans le gestionnaire de tâches sans avoir à injecter de code.

3 Modification de l'affichage

Pour pouvoir afficher les mots « infected » et « virus free », notre DLL utilise une technique simple mais efficace. Les appels successifs aux fonctions **SetTextColor** et **DrawTextA** en sont responsables :

```
test    al, al
jnz     short loc_10001FF
push    0FFh            ; Text Color> Red:0FFh Green:00 Blue:00 --> Texte Rouge
push    esi             ; hdc
call    ds:SetTextColor
push    24h
lea     edx, [esp+17Ch+rc]
push    edx
push    8
push    offset aInfected ; "Infected"
jmp     short loc_10001516

                        ; CODE XREF: sub_10001340+199↑j
                        ; sub_10001340+1A1↑j
push    8               ; Text Color> Red:00 Green:00 Blue:08 --> Texte Noir
push    esi             ; hdc
call    ds:SetTextColor
push    24h             ; format
lea     eax, [esp+17Ch+rc]
push    eax             ; lprc
push    0Ah             ; cchText
push    offset chText   ; "Virus Free"

                        ; CODE XREF: sub_10001340+1BD↑j
push    esi             ; hdc
call    ds:DrawTextA
mov     ecx, [esp+178h+wParam]
mov     eax, [esp+178h+var_164]
inc     ecx
dec     eax
mov     [esp+178h+wParam], ecx
mov     [esp+178h+var_164], eax
jnz     loc_100013E8
```

Fig. 9 : Routine de modification de l'affichage du taskmgr

J'ai pris soin de commenter les paramètres **Color** de la première fonction. Ce paramètre est en fait la représentation RGB (Rouge Vert Bleu) de la couleur en hexadécimal. Il suffit d'utiliser 0xFF pour le rouge, et 0x0 pour le vert et le bleu pour obtenir une couleur rouge. Le noir (bleu foncé) est obtenu avec un 0x8 pour le bleu et rien pour le rouge et vert.

Avant de pouvoir afficher du texte, notre programme malicieux doit d'abord dessiner un rectangle et « calculer » l'emplacement de celui-ci. Ensuite, chaque ligne est écrite une par une, pour donner l'illusion d'une information relative au processus qui se trouve sur la même ligne.

Avant d'effectuer la modification, la présence des autres composants du Rogue est vérifiée. En cas d'absence de ces derniers, aucune modification n'est apportée. Il s'agit probablement d'empêcher le test de la DLL sans installer le malware, mais aussi de parer un éventuel vol de leur DLL par d'autres faux antivirus.

Conclusion

Les techniques alarmantes utilisées par les Rogues sont de plus en plus évoluées. Au-delà du simple pop-up, nous avons maintenant des techniques plus évoluées, telles que l'injection présentée dans cet article. A noter que notre malware sait aussi « détecter » l'exécution de tout programme et affiche de temps en temps un message personnalisé, contenant le nom de l'application ainsi qu'une infection factice lors de son exécution. L'utilisateur se voit donc informé que ses programmes sont infectés, il ne s'agit plus de noms de fichiers aléatoires. ■