

LE RETOUR DE GPCODE

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

MALWARE CORNER

mots-clés : CODES MALICIEUX / RANSOMWARE / RSA / ANALYSE DE CODE / AES

25 mars 2011, alors que vous surfiez tranquillement sur Internet, le fond d'écran de votre Windows change et Notepad s'ouvre soudainement pour vous informer que tous vos fichiers personnels ont été chiffrés à l'aide de cryptographie forte (RSA 1024) et qu'il est impossible de les récupérer sans payer 125 dollars en carte pré-payée. Vous ne rêvez pas, vous êtes la victime du nouveau Gpcode.

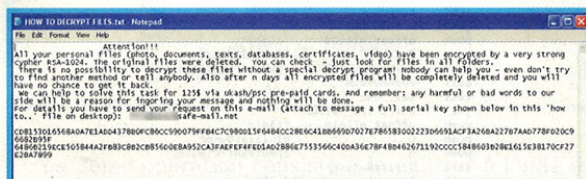


Figure 1 : Demande de rançon par Gpcode

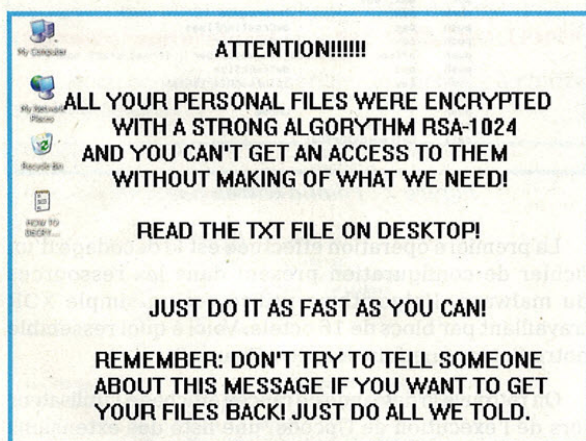


Figure 2 : Fond d'écran modifié par Gpcode

Tout ceci est dû à la visite d'un site malveillant (*drive by downloads*) par une machine non mise à jour. Une fois exécuté, Gpcode génère une clé AES 256 aléatoire puis la chiffre à l'aide de la clé publique des criminels, qui n'est autre que du RSA 1024. Nous allons maintenant voir les détails dans une analyse du code.

1 Obfuscation

La dernière version de Gpcode apparue en novembre 2010 était simplement compressée à l'aide d'UPX.

Une fois décompressée (**upx -d**), nous obtenions un binaire prêt à être analysé. Bien qu'UPX soit toujours présent dans la nouvelle version, le binaire obtenu après décompression est encore crypté. En effet, nous sommes en présence d'un packer « custom » utilisé pour rendre l'analyse du code plus compliquée :

```
.text:00412187      push    ebp
.text:00412188      mov     ebp, esp
.text:0041218A      add     esp, 0FFFFFFDh
.text:0041218D      add     edi, ebx
.text:0041218F      call    IsUPX_removed
.text:00412194      mov     eax, edx
.text:00412196      neg     edi
.text:00412198      dec     edx
.text:00412199      inc     edx
.text:0041219A      not     edi
.text:0041219C      jmp     loc_4012AC
.text:0041219C      start   endp : op-analysis failed
.text:0041219C
```

Figure 3 : Point d'entrée du second packer

L'analyse de ce packer dans son intégralité remplirait à lui seul le *Malware Corner*, je vous invite à lire mon article dans le numéro 51 de MISC, qui présente des techniques similaires à celles employées ici.

On notera cependant une routine intéressante utilisée pour détecter l'unpacking de la première couche UPX :

```

.txt:00401080  IsUpX_removed      proc near                                ; CODE XREF: start+81p
.txt:00401080          push     ebp
.txt:00401081          dec     ecx
.txt:00401082          not     esi
.txt:00401084          mov     ebp, esp
.txt:00401086          neg     esi
.txt:00401088          mov     edx, edi
.txt:0040108A          add     esi, edx
.txt:0040108C          add     esp, 0FFFFFFDh
.txt:0040108F          dec     esi
.txt:00401010          mov     edi, edx
.txt:00401012          mov     ecx, esp
.txt:00401014          inc     ebx
.txt:00401015          mov     edx, ebx
.txt:00401017          not     edi
.txt:00401019          add     ecx, 0FFFFFFFh
.txt:0040101C          dec     esi
.txt:0040101D          cmp     eax, ecx          ; if EAX = ECX then UPX still present
.txt:0040101F          jz      short_not_unpacked
.txt:00401021          inc     eax
.txt:00401022          dec     ecx
.txt:00401023          leave
.txt:00401024          mov     ecx, edx
.txt:00401026          leave
.txt:00401027          add     edx, esi
.txt:00401028          inc     ecx
.txt:0040102A          retm
; RET Leads to ExitThread

```

Figure 4 : Fonction de détection de l'unpacking d'UPX

Le fonctionnement de la routine est très simple. Elle utilise le fait qu'UPX place une adresse de la pile dans le registre EAX juste avant d'exécuter l'application décompressee. Lors de l'appel de la routine de vérification, EAX doit contenir cette adresse pour passer le « CMP EAX, ECX ».

Le packer effectue des opérations simples pour obtenir une adresse sur la pile qui sera la même que celle placée dans EAX par UPX. En cas d'unpacking préalable (de la première couche UPX), le registre EAX ne sera pas correctement initialisé et l'unpacking sera détecté. L'exécution se terminera par l'appel de la fonction **ExitThread**.

Pour analyser le packer, il suffit de ne pas retirer la couche UPX, ou de patcher le « JZ » en « JMP » à l'adresse « 0x40101F ».

Débuguer l'intégralité du packer prend pas mal de temps. Il existe cependant une petite astuce pour y arriver en moins de 10 secondes. Si vous avez lu l'article dans le numéro 51, vous vous souviendrez de l'allocation de mémoire pour décrypter le fichier original sur le heap. La majorité des packers employés pour protéger des programmes malveillants utilisent ces techniques.

Il semblerait que les développeurs, dans un souci de propreté du code, utilise la fonction **VirtualFree** pour désallouer la mémoire contenant une copie du fichier unpacké dans son intégralité.

Il suffit donc de patcher l'anti unpacking du point d'entrée, de placer un point d'arrêt sur **VirtualFree** et d'exécuter notre ransomware pour arriver à l'appel :

```
0012F3BC 003D18B8 CALL to VirtualFree from 003D18B6
0012F3C0 00910000 Address = 00910000
0012F3C4 00002400 Size = 2400 (9216.)
0012F3C8 00004000 FreeType = NEH_DECOMMIT
0012F3CC 0012F5C0
```

Figure 5 : Appel à la fonction VirtualFree

Le paramètre de **VirtualFree** est l'adresse à désallouer, ici 0x910000. Avant d'effectuer le nettoyage, il est possible de récupérer un fichier décrypté. Pour se faire, il suffit d'utiliser la fonction **follow in dump** d'ollydbg, par exemple. Un rapide coup d'œil à cette adresse nous donne ceci :

Address	Hex	dump	ASCII
00910000	04 50 90 00	03 00 00 00	04 00 00 00 FF FF 00 00
00910001	08 00 00 00	00 00 00 00	04 00 00 00 00 00 00 00
00910002	00 00 00 00	00 00 00 00	00 00 00 00 00 00 00 00
00910003	00 00 00 00	00 00 00 00	00 00 00 00 80 00 00 00
00910004	0C 1F 00 00	00 00 00 00	00 00 00 00 00 00 00 00
00910005	69 73 20 00	72 6F 62 62	61 62 63 64 65 66 67 68
00910006	74 20 00 00	20 72 75 6E	69 6A 6B 6C 6D 6E 6F 70
00910007	60 6F 6A 05	2F 00 00 00	2A 00 00 00 00 00 00 00
00910008	50 05 00 00	4C 01 03 02	52 36 0F 4C 00 00 00 00

Figure 6 : Dump de la mémoire allouée

Nous avons ici l'exécutable malicieux totalement décrypté. Il ne nous reste plus qu'à le dumper, comme décrit dans l'article précédemment cité (clic droit sur le dump hexadécimal, **Backup** et **Save data to File**).

À notre grande surprise, cet exécutable est une fois de plus compressé par UPX. Un rapide **upx -d** nous donne enfin l'exécutable final. Temps total d'unpacking : moins de 30 secondes.

2 Analyse du Ransomware

Voici à quoi ressemble le point d'entrée de Gpcode une fois celui-ci totalement décompressé et décrypté :

```
start      public start
proc near
call       Decrypt_config_file
test      eax, eax
jz         loc_4019DD
push      offset aIld      ; "Ild"
push      0                ; bInheritHandle
push      MUTEX_ALL_ACCESS ; dwDesiredAccess
call      OpenMutexA
test      eax, eax
jnz       short loc_4019DD
push      offset aIld      ; "Ild"
push      0                ; bInitialOwner
push      0                ; lpMutexAttributes
call      CreateMutexA
call      Generate_random_AES_256_key
test      eax, eax
jz         short loc_4019DD
call      Encrypt_AES_key_with_RSA1024
xor        eax, eax
push      eax               ; lpThreadId
push      eax               ; dwCreationFlags
push      eax               ; lpParameter
push      offset drop_random_txt_wallpaper ; Threat start address
push      eax               ; dwStackSize
push      eax               ; lpThreadAttributes
call      CreateThread
push      1
call      SetErrorMode      ; uMode
call      GetLogicalDrives
```

Figure 7 : Point d'entrée unpacké

La première opération effectuée est le décodage d'un fichier de configuration présent dans les ressources du malware. L'algorithme utilisé est un simple XOR travaillant par blocs de 16 octets. Voici à quoi ressemble notre fichier une fois décodé : Figure 8.

On retrouve la demande de rançon affichée à l'utilisateur lors de l'exécution de Gpcode, une liste des extensions à chercher sur l'ordinateur de la victime (pour chiffrer les fichiers), et un blob RSA qui contient les paramètres « e » et « n », soit la clé publique des criminels qui sera utilisée pour chiffrer la clé AES 256 employée lors du chiffrement des données. Il contient aussi des informations telles que le pourcentage du fichier à chiffrer.

La seconde opération de Gpcode consiste à générer une clé AES de 256 bits ; Figure 9

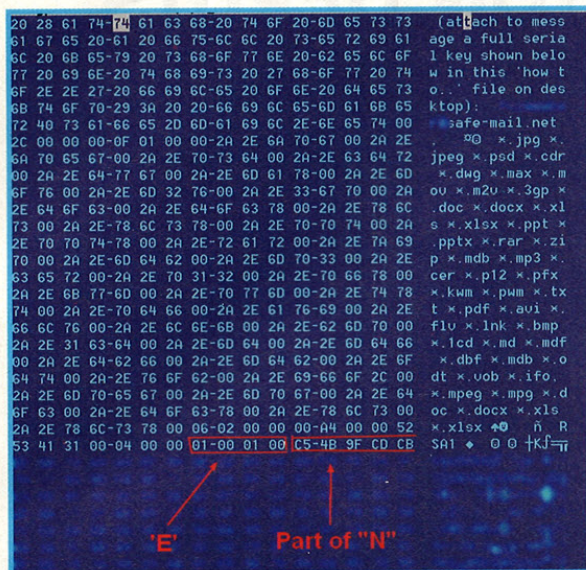


Figure 8 : Fichier de configuration décodé

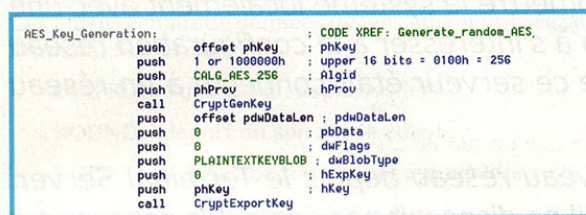


Figure 9 : Génération de la clé AES 256

La 3ème opération consiste à la chiffrer à l'aide de la clé publique (RSA 1024) des criminels. Un thread est créé et sert à la création du fichier TXT sur le bureau de la victime ainsi qu'au changement du fond d'écran (**SystemParametersInfo** avec **SPI_SETDESKWALLPAPER**).

Gpcode commence à chercher les fichiers à chiffrer sur le disque :

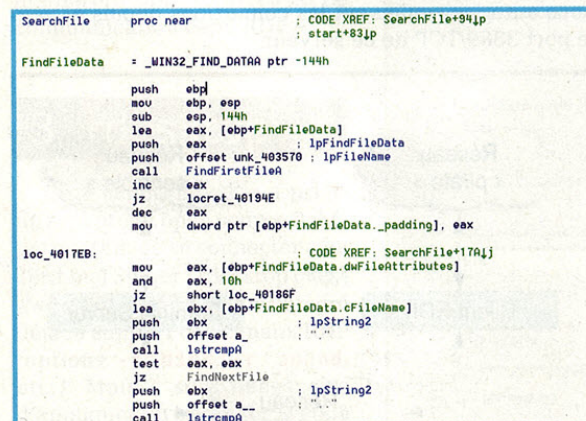


Figure 10 : Recherche de fichiers à chiffrer

Une fois un fichier à chiffrer découvert, celui-ci sera ouvert et chiffré directement :

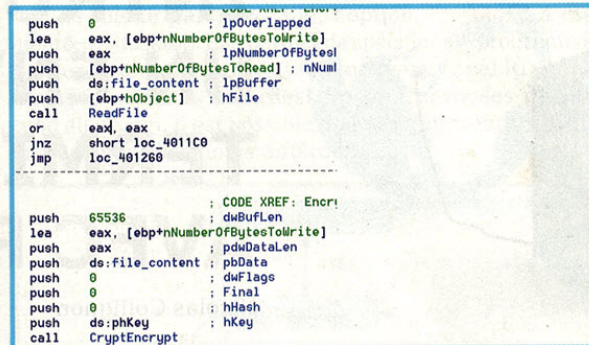


Figure 11 : Routine de chiffrement de Gpcode

Il est important de noter que le fichier n'est pas chiffré dans son intégralité. En effet, le fichier de configuration contient le pourcentage du fichier à chiffrer.

Une fois le fichier chiffré, celui-ci est renommé en **fichier_original.extension.ENCODED**.

Les premières versions de Gpcode faisaient une copie du fichier à chiffrer et effaçaient l'original avec un effacement classique. Il était possible de récupérer le fichier original à l'aide d'outils de récupération de données. Cette vulnérabilité n'est plus présente dans les dernières versions.

Gpcode passe aux fichiers suivants jusqu'à avoir parcouru tous les disques. Une fois les fichiers chiffrés, la clé AES 256 est détruite à l'aide de la fonction **CryptDestroyKey** :

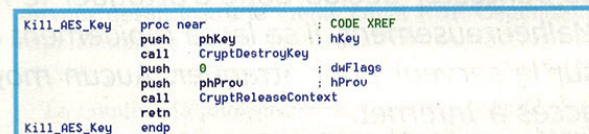


Figure 12 : Destruction de la clé AES

Il est alors impossible de retrouver la clé AES, même en dumpant la mémoire physique d'une machine infectée. Pour pouvoir récupérer les données, il faudrait freezer Gpcode et dumper la mémoire avant que la clé soit détruite.

Pour terminer, Gpcode génère un fichier **.BAT** avant d'appeler la fonction **ExitProcess**. Ce fichier tente d'effacer l'exécutable de Gpcode et y parviendra une fois celui-ci terminé.

Ainsi s'achève l'analyse de Gpcode. Il n'existe à ma connaissance aucune attaque permettant de récupérer les fichiers chiffrés ou la clé AES générée sur une machine affectée dans les conditions réelles. L'utilisation de sauvegardes reste la seule solution pour récupérer les fichiers pris en otage.

Il est aussi intéressant de noter que les criminels ont commencé à utiliser les cartes pré-payées (Ukash) plutôt que les transferts d'argent pour récupérer la rançon. En novembre 2010, il fallait toujours effectuer un versement d'argent pour payer celle-ci.

Quelques jours avant la découverte de Gpcode, un autre ransomware beaucoup moins dangereux se faisant passer pour une alerte de la police fédérale allemande demandait aussi le paiement par cartes pré-payées. ■