

Analyse de Locky

Auteur: Adrien Couëron

Introduction

Contexte

Ce document a été créé pour présenter une analyse du ransomware Locky afin d'améliorer les compétences personnels de l'auteur et de fournir une analyse approfondie du malware.

Le document a été réalisé alors que l'analyse n'était pas complète. Il permet de présenter les premiers résultats de l'analyse. Des mises à jour du document seront effectués tout au long de l'avancée de l'analyse.

Objectifs

Les principaux objectif de l'analyse est de trouver des moyens de bloquer le processus de chiffrement de fichiers (Communication avec le C&C, détection, ...) et de restaurer des fichiers déjà chiffrés (faiblesse de l'implémentation cryptographique, du partage de clés).

Structure du document

Le document se structure suivant les chapitres suivants:

- Informations sur les fichiers exécutables étudiés du malware
- Dépackage du malware
- Processus d'infection

- Détails de fonctionnement

Annotation

Des annotations du type ? pourront suivre des explications durant le rapport. Il signale un doute de l'auteur sur la véracité d'un fait. Il préfère de vous le signaler plutôt que de vous induire en erreur.

Si vous voulez affirmer ou corriger des doutes laissés dans ce rapport, il vous est possible d'entrer en contact avec l'auteur (adrien.coueron@wanadoo.fr). Tout doute levé améliorera la qualité de ce rapport et aidera les futurs lecteurs.

I-Informations sur les fichiers

A-Fichier packer

1-Hash

- MD5: 73304ca4e455286b7a63ed71af48390a
- SHA1: e8ea52e0d43f9420a65993a4123fc15d64bc880e
- SHA256:
3dc979164206c86823cab9684e662f84528d40a92027f48d31970c3d8f9f5114
- SHA512:
9d80839100d20c334a4c0f74bb8a2d4dc121c14bbd09d50a80eed7e94e514c8feb28c393f5fd90087b08e387bf83bffb7ac337a48578b86dcdb9b58d90a903c
- SSDEEP:
3072:wOM5W8c5FAswlJPY/ePTkflEVE/3WhKoxasMvzzzFVy0lv4p7RhPu/O3iXgOYbL:eW8c5KIJPY2LkflEVEPWhKnl+A6

2-Type de fichier

locky_packed: PE32 executable (GUI) Intel 80386, for MS Windows

3-Informations PE

- Date de compilation: 24/02/2016 10:53:51
- Machine cible: 0x14C Intel 386 et processeurs précédents compatibles
- Point d'entrée: 0x00417430

a-Sections

Nom	Adresse virtuelle	Taille virtuelle	Taille original	MD5
.text	0x00401000	0x17956	0x17A00	975653a6c2bd9ef08e
.core	0x00419000	0x200	0x17E00	e1596859847c1e1a1
.rsc	0x0041A000	0x108DC	0x10A00	51e53af42d7e5639c2

b-Imports

- ntdll.dll
 - RtlZeroMemory
- KERNEL32.dll
 - InitializeCriticalSection
 - Sleep
 - LeaveCriticalSection
 - GetProcAddress
 - EnterCriticalSection
 - LoadLibraryA
 - LocalAlloc
 - DeleteCriticalSection
 - ReleaseMutex
 - CloseHandle
 - LocalFree
 - CreateThread
 - lstrcpA
 - ExitProcess
 - GetLastError
- ADVAPI32.dll
 - RegCreateKeyExA
 - SetSecurityDescriptorDacl
 - RegCloseKey
 - FreeSid

- SetEntriesInAclA
- InitializeSecurityDescriptor
- AllocateAndInitializeSid
- COMCTL32.dll
 - InitCommonControlsEx
 - ImageList_Add

B-Fichier unpacker

1-Hash

- MD5: 45f4c705c8f4351e925aea2eb0a7f564
- SHA1: dc04128fd3e916e56ce734c06ff39653c32ade50
- SHA256:
034af3eff0433d65fe171949f1c0f32d5ba246d468f3cf7826c42831a1ef4031
- SHA512:
a4462f7d98ef88e325aac54d1acffd4b8f174baa77efd58f85cdd145201a99e7b03f9ba6f25bdd25265714aa25070a26f72d18401de5463a91b3d21b47d17b13
- SSDEEP:
3072:3072:pjNaly6K25gyi4x3gS6Y1TcVbrkijMziie:pAsah1wtLjMPe

2-Type de fichier

locky_packed: PE32 executable (GUI) Intel 80386, for MS Windows

3-Informations PE

- Date de compilation: 07:26:59 29/01/2002
- Machine cible: 0x14C Intel 386 et processeurs précédents compatibles
- Point d'entrée: 0x0040A344

a-Sections

Nom	Adresse virtuelle	Taille virtuelle	Taille original	MD5
.text	0x00401000	0xF28B	0xF400	009d0d91d06f2b87817

.rdata	0x00411000	0x60B8	0x6200	fd8ac6be745acedaca4
.data	0x00418000	0x1B64	0xE00	2eba3ead215cf9594aæ
.reloc	0x0041A000	0x21CA	0x2200	0107bbcaa901e0b260

b-Imports

- KERNEL32.dll
 - LeaveCriticalSection
 - GetCurrentThread
 - FindNextFileW
 - GetDiskFreeSpaceExW
 - GetVolumeInformationW
 - GetLogicalDrives
 - GetDriveTypeW
 - EnterCriticalSection
 - LoadLibraryW
 - HeapReAlloc
 - DeleteCriticalSection
 - InitializeCriticalSection
 - GetSystemTime
 - GetTempFileNameW
 - CreateProcessW
 - GetModuleHandleA
 - GetProcAddress
 - GetCurrentProcess
 - FindClose
 - GetVolumeNameForVolumeMountPointA
 - GetWindowsDirectoryA
 - GetLocaleInfoA
 - FindFirstFileW
 - MultiByteToWideChar

- WideCharToMultiByte
- WaitForSingleObject
- CreateThread
- CopyFileW
- GetTempPathW
- Sleep
- GetUserDefaultUILanguage
- GetUserDefaultLangID
- GetSystemDefaultLangID
- SetUnhandledExceptionFilter
- SetErrorMode
- MulDiv
- GetVersionExA
- ExitProcess
- GetModuleFileNameW
- GetLastError
- FlushFileBuffers
- SetFileTime
- GetSystemTimeAsFileTime
- SetFilePointer
- ReadFile
- SetFileAttributesW
- GetFileAttributesExW
- DeleteFileW
- MoveFileExW
- WriteFile
- GetFileSizeEx
- CreateFileW
- CloseHandle
- RtlUnwind
- GetCurrentProcessId

- GetTickCount
- QueryPerformanceCounter
- GetFileType
- InitializeCriticalSectionAndSpinCount
- SetHandleCount
- GetEnvironmentStringsW
- FreeEnvironmentStringsW
- GetModuleFileNameA
- GetStringTypeW
- LCMapStringW
- HeapCreate
- GetStdHandle
- TerminateProcess
- IsDebuggerPresent
- UnhandledExceptionFilter
- GetCurrentThreadId
- SetLastError
- TlsFree
- TlsSetValue
- TlsGetValue
- TlsAlloc
- HeapAlloc
- HeapFree
- GetCommandLineA
- HeapSetInformation
- GetStartupInfoW
- RaiseException
- IsProcessorFeaturePresent
- HeapSize
- GetModuleHandleW
- GetCPInfo

- InterlockedIncrement
 - InterlockedDecrement
 - GetACP
 - GetOEMCP
 - IsValidCodePage
- USER32.dll
 - DrawTextW
 - SystemParametersInfoW
 - ReleaseDC
 - FrameRect
 - FillRect
 - GetSystemMetrics
 - GetDC
- GDI32.dll
 - SetTextColor
 - GetDIBits
 - GetObjectA
 - SetBkMode
 - CreateSolidBrush
 - CreateCompatibleBitmap
 - SelectObject
 - CreateFontA
 - DeleteObject
 - GetDeviceCaps
 - CreateCompatibleDC
 - DeleteDC
- ADVAPI32.dll
 - CryptGetHashParam
 - AccessCheck
 - MapGenericMask
 - DuplicateToken

- OpenThreadToken
- GetFileSecurityW
- CryptHashData
- SetTokenInformation
- OpenProcessToken
- CryptDestroyHash
- CryptCreateHash
- RegSetValueExW
- RegQueryValueExA
- RegDeleteValueA
- RegSetValueExA
- RegCreateKeyExA
- RegCloseKey
- RegOpenKeyExA
- CryptAcquireContextA
- CryptGenRandom
- CryptReleaseContext
- CryptEncrypt
- CryptSetKeyParam
- CryptImportKey
- CryptDestroyKey
- SHELL32.dll
 - SHGetFolderPathW
 - ShellExecuteW
- WININET.dll
 - InternetOpenA
 - InternetCloseHandle
 - InternetSetOptionA
 - HttpOpenRequestA
 - InternetQueryOptionA
 - HttpSendRequestExA

- InternetWriteFile
- HttpEndRequestA
- HttpSendRequestA
- HttpQueryInfoA
- InternetCrackUrlA
- InternetReadFile
- InternetConnectA
- MPR.dll
 - WNetEnumResourceW
 - WNetCloseEnum
 - WNetAddConnection2W
 - WNetOpenEnumW
- NETAPI32.dll
 - DsRoleGetPrimaryDomainInformation
 - DsRoleFreeMemory

II-Unpack

Le fichier unpacké étant disponible et l'analyse du fonctionnement du malware étant la priorité, la procédure de dépackage sera réalisée et expliquée dans l'avenir.

III-Processus d'infection

A-Vue globale

Locky est un ransomware. Son but est de chiffrer les fichiers personnels de l'utilisateur d'un poste infecté. Par la suite, des instructions sont données à l'utilisateur pour lui expliquer comment payer et récupérer ses données. Les instructions sont présentées grâce à un fond d'écran et des fichiers textes dans le système de fichiers.

Une vue globale du processus d'infection analyse couvert par l'analyse est présenté dans l'illustration 1.

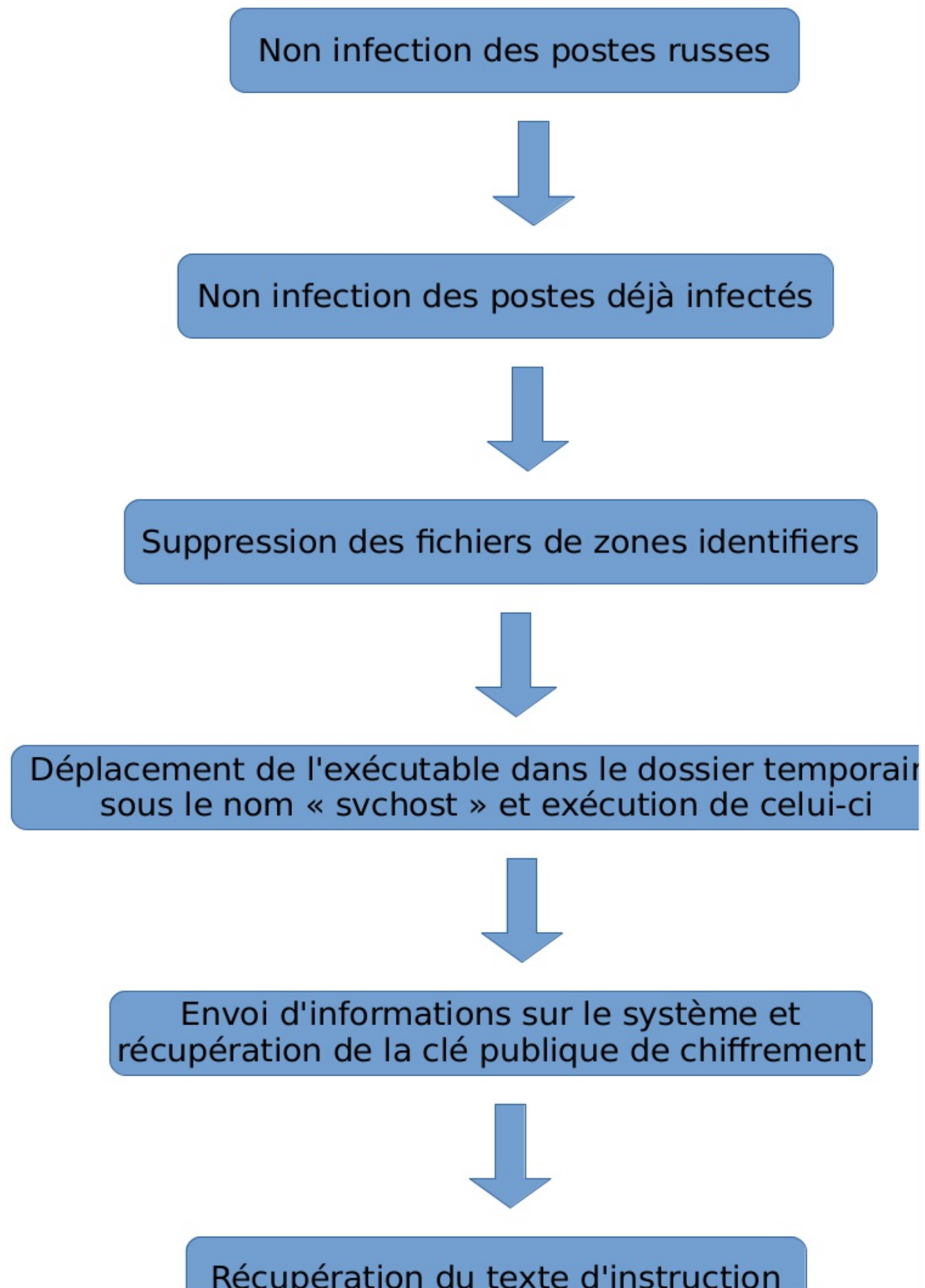


Illustration 1: Vue d'ensemble du processus d'infection

Le reste de ce chapitre présente les actions réalisées par le malware dans l'ordre chronologique.

B-Initialisation

1-Désactivation de la virtualisation

Le malware commence par changer les propriétés du token de son processus. Il désactive la "virtualisation". Cela permet au malware d'accéder aux fichiers et aux clés de registres globaux à la machine et non restreint à l'utilisateur?.

```
lea    eax, [ebp+accesstoken]
push   eax                ; TokenHandle
xor     ebx, ebx
push   TOKEN_ADJUST_DEFAULT ; DesiredAccess
                                ; Required to change default owner, primary group or DACL of access token
mov     [ebp+accessTokenInformation], ebx ; accessTokenInformation = 0
call    ds:GetCurrentProcess ; Get handle to the current process
push   eax                ; ProcessHandle
call    ds:OpenProcessToken ; Get current process access token
test    eax, eax
jz      short getAccessTokenFailed
```

```
push   INT_LENGTH        ; TokenInformationLength
lea     eax, [ebp+accessTokenInformation]
push   eax                ; TokenInformation
push   TokenVirtualizationEnabled ; TokenInformationClass
push   [ebp+accesstoken] ; TokenHandle
call    ds:SetTokenInformation ; unsetVirtualizationToThisProcessus
push   [ebp+accesstoken] ; hObject
call    ds:CloseHandle    ; stopUseAccessToken
```

Illustration 2: Désactivation de la virtualisation du processus

2-Désactivation des redirections WoW64

Ensuite le malware désactive les redirections WoW64 (Windows 32 bits on Windows 64 bits) du système de fichiers. Cela enlève les redirections transparentes vers les dossiers de compatibilité 32 bits sur les systèmes 64 bits.

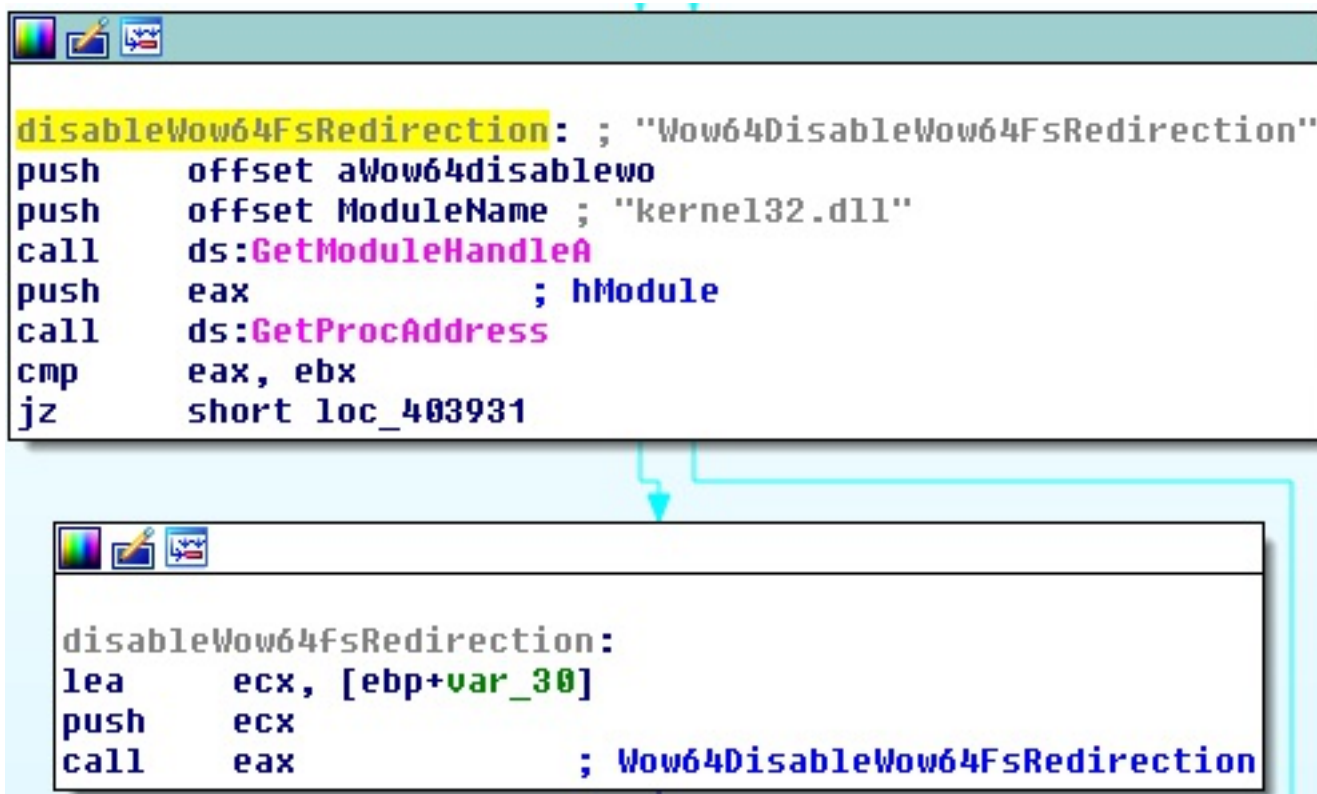


Illustration 3: Désactivation des redirections WoW64

3-Initialisation de la liste des adresses IP de C&C

Le malware contient dans sa configuration une liste d'adresse IP de C&C à contacter. Lors de cette étape, il crée un vecteur contenant chacun de ces adresses en vue de les utiliser plus tard.

```

:004137E0 configuration    T_configuration <3, 7, 1Eh, 0, 0, \
:004137E0                                ; DATA XREF: getPubkey+298↑r
:004137E0                                ; WinMain(x,x,x,x):listIpAddressesEmpty↑r ...
:004137E0                                '31.41.47.37,188.138.88.184,91.121.97.170,5.34.183.136'>

```

Illustration 4: Configuration du sample

Nous voyons ici la structure des données de configuration dont le troisième champs est une liste des adresses IP de C&C séparés par des virgules. Ce sont ces adresse IP qui sont extraites et rentrées dans une structure de données de type vector (Vraisemblablement une structure standard au langage de programmation utilisé).

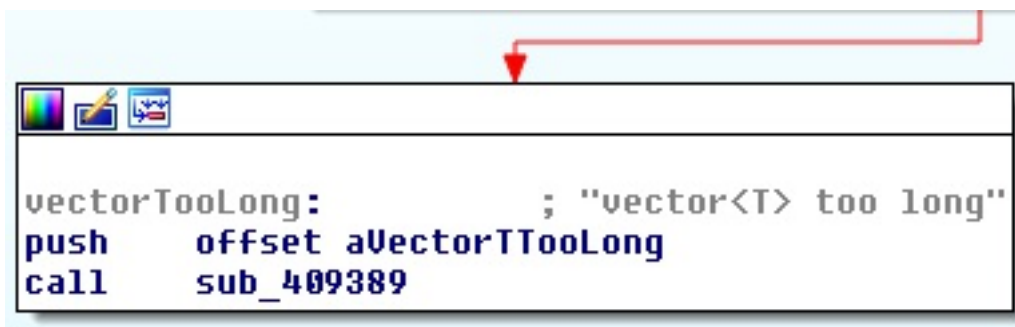


Illustration 5: Trace d'utilisation de structure de données de type vector

4-Vérification de non infection de postes russes

Le malware vérifie à travers trois paramètres du systèmes s'il n'est pas sur un poste russe (La langue du système, la langue de l'utilisateur et la langue de l'interface graphique). Si pour l'un, il s'avère que c'est le cas, le malware n'effectue pas la procédure de chiffrement.

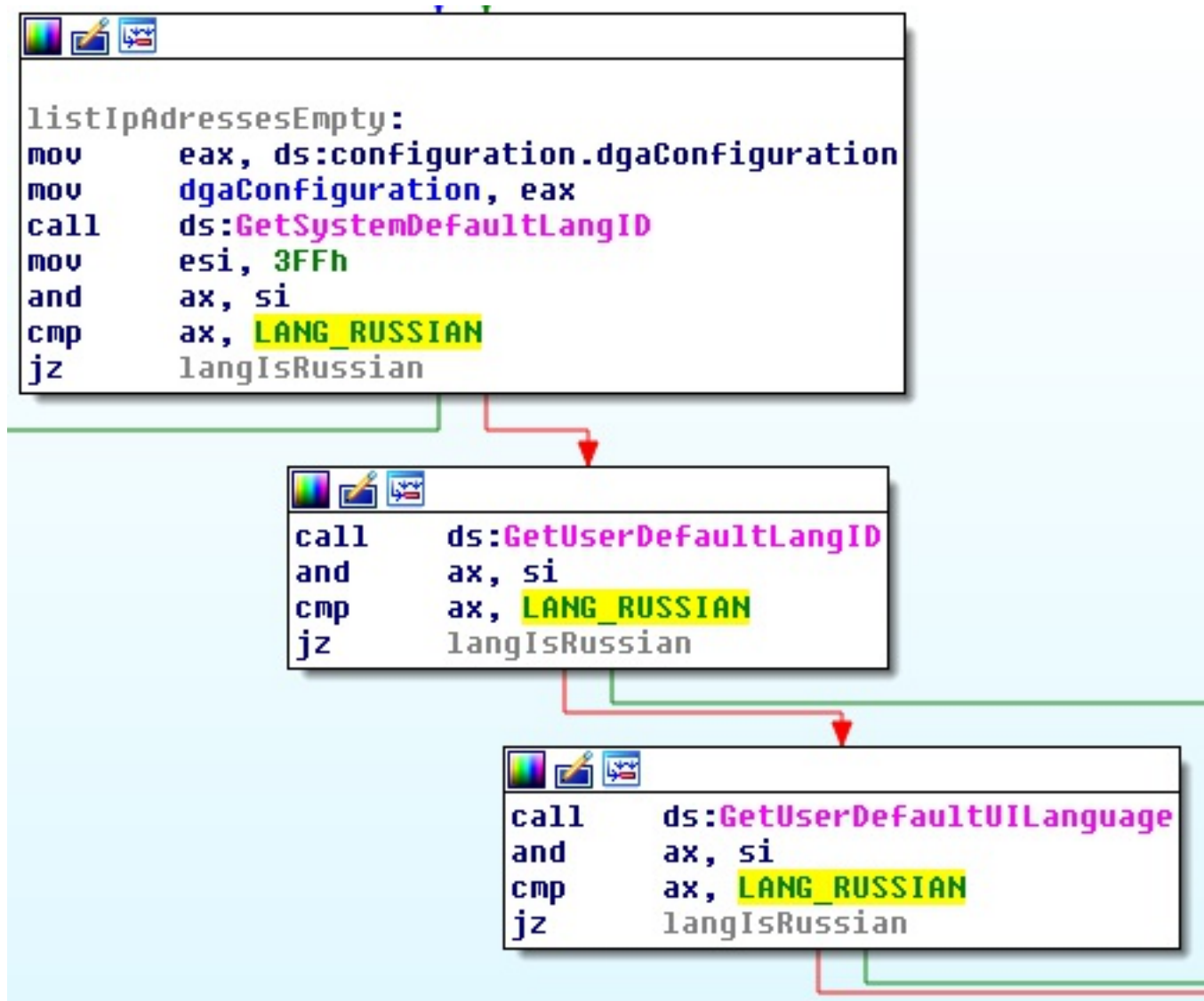


Illustration 6: Vérification de la non infection de poste russe (Vue précise)

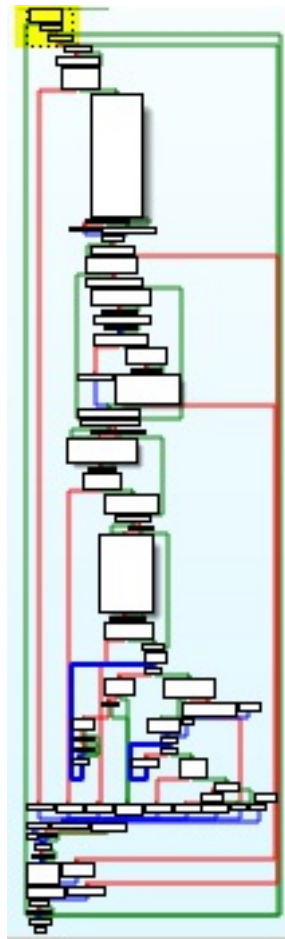


Illustration 7: Vérification de la non infection de poste russe (Vue macro)

L'illustration 7 montre que le flux d'exécution est dévié en fin de programme si le poste est Russe (Trois traits verts partant de la zone jaune).

Ceci permet de configurer le système pour le protéger d'une infection (cf Partie V).

5-Attente avant l'activation du malware

Le malware contient dans sa configuration une donnée qui définit le temps qu'il attendra avant de se déclencher. Il pourra ainsi attendre entre 0 et 9 heures pour s'activer.

```
004137E0 configuration    T_configuration <3, 7, 1Eh, 0, 0, \
004137E0                                ; DATA XREF: getPubkey+298↑r
004137E0                                ; WinMain(x,x,x,x):listIpAdressesEmpty↑r ...
004137E0                                '31.41.47.37,188.138.88.184,91.121.97.170,5.34.183.136'>
```

Illustration 8: Configuration du malware et temps d'attente

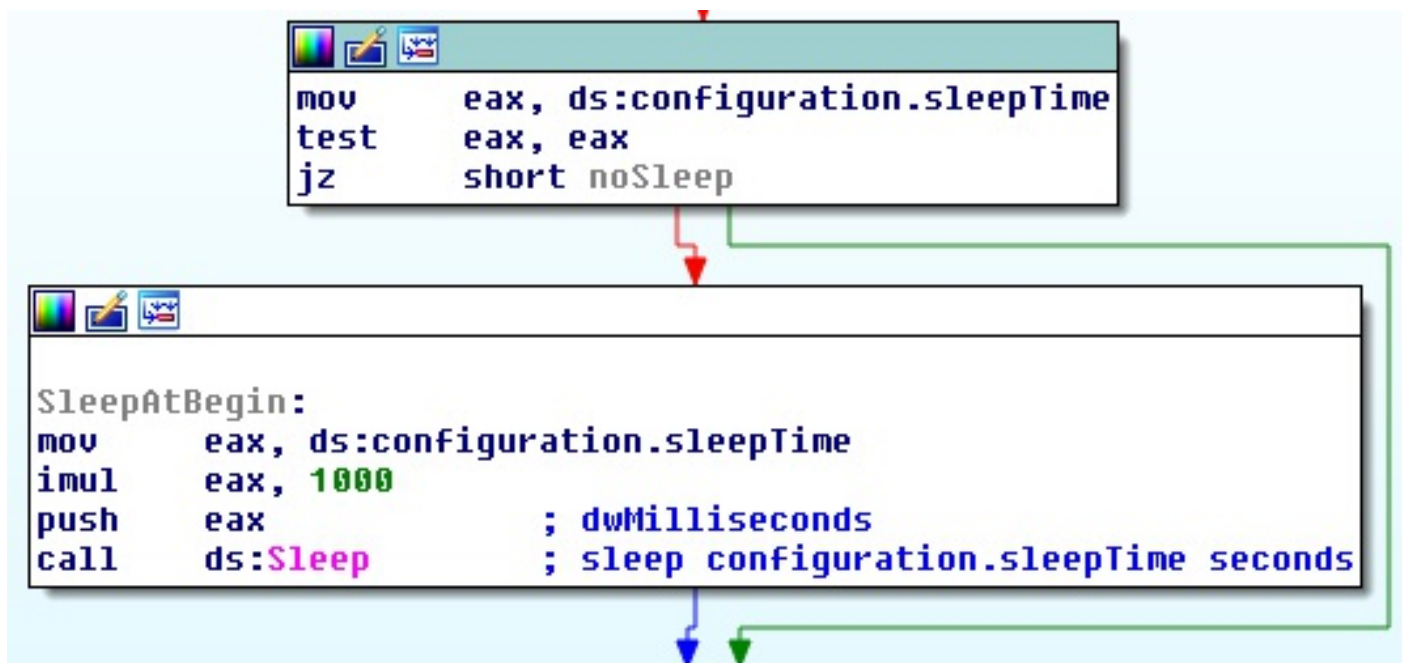


Illustration 9: Attente

Le sample étudié attendra 30 secondes.

6-Ouverture de la clé de registre principale

Le malware utilise une clé de registre du nom de "Locky" dans HKEY_USER\Software. Il y stocke l'identifiant de la victime, le texte d'explication pour la rançon, la clé publique pour le chiffrement et un marqueur de réalisation passée de l'attaque. Lorsqu'il ouvre la clé principale, si une erreur apparaît, il n'effectue pas le chiffrement.

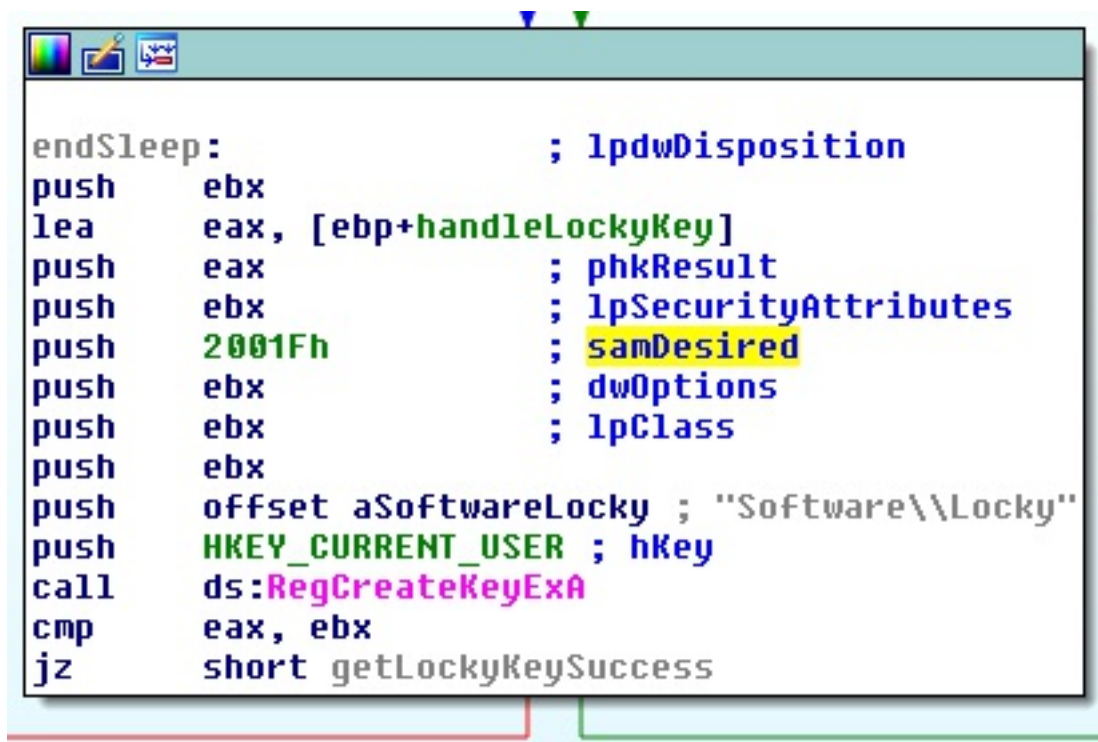


Illustration 10: Ouverture de la clé HKEY_USER\Software\Locky (Vue précise)

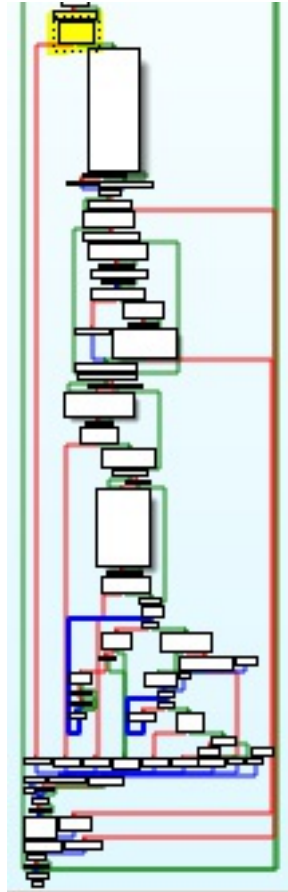


Illustration 11: Ouverture de la clé HKEY_USER\Software\Locky (Vue macro)

L'illustration 11 montre que si l'ouverture de la clé de registre n'est pas possible, le flux d'exécution est dévié jusqu'à la fin du programme (Trait rouge partant de la zone jaune).

Ceci permet de configurer le système pour le protéger d'une infection (cf Chapitre V).

7-Récupération des valeurs des sous-clés de registre

Locky prend les valeurs présentes dans les sous-clés de registre de clé publique, de texte d'explication, d'identifiant et le marqueur de fin avant de les sauvegarder dans des variables globales.

8-Calcul de l'identifiant de la victime

Le malware définit un identifiant à chaque victime suivant le GUID du disque contenant le système Windows. Celui-ci lui sert par la suite.

La procédure de génération d'identifiant est détaillée dans la partie IV-1.

9-Recherche des traces d'infection passée

a-Vérification de la validité du marqueur de fin

Le malware récupère la sous-clé, vérifie qu'il contient bien un entier. Si la clé n'existe pas, ne contient rien ou une valeur nulle, le malware s'exécutera. N'importe quelle valeur entière est donc un marqueur qui montre que Locky a déjà infecté le poste.

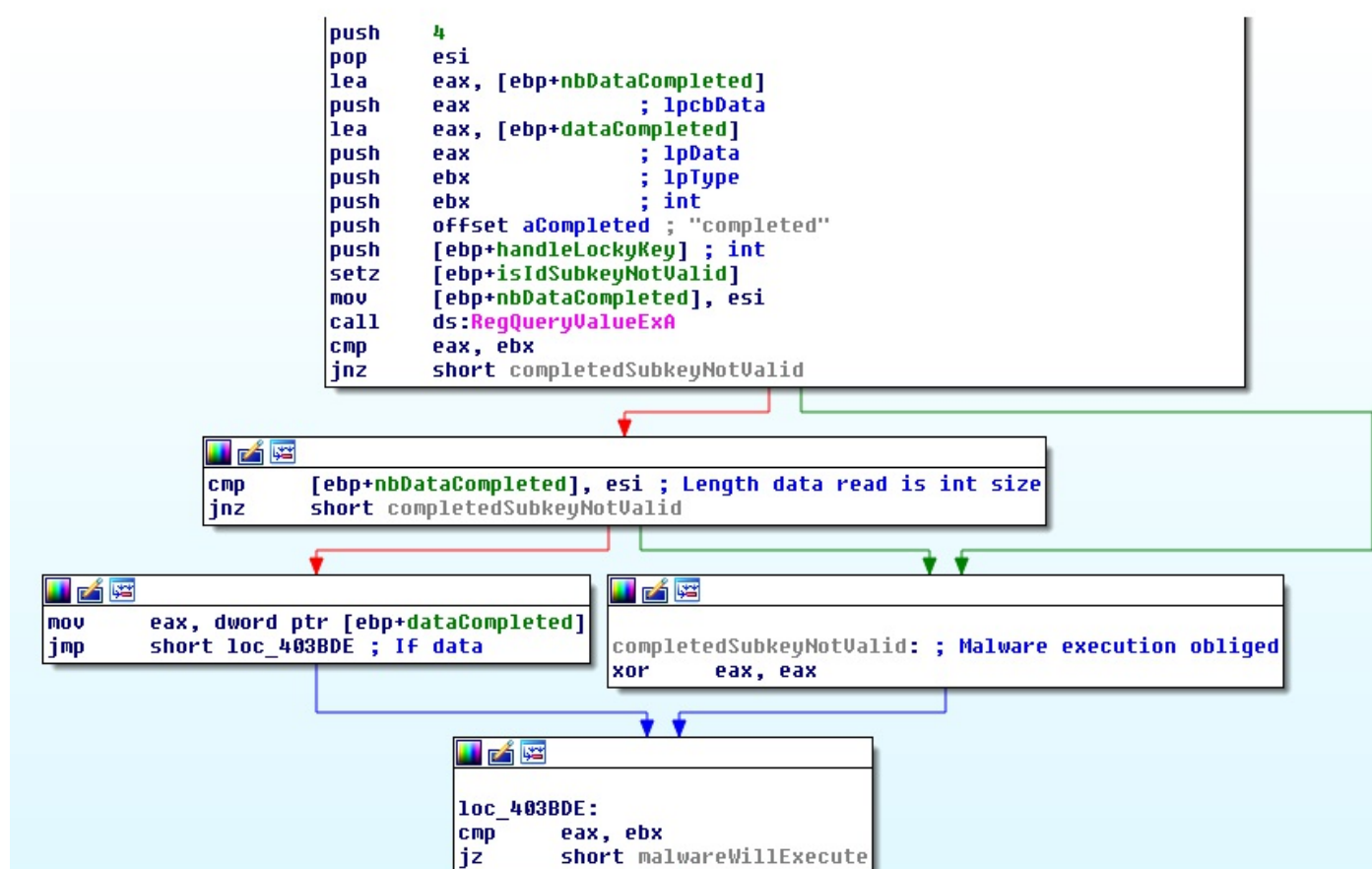


Illustration 12: Vérification de la validité de la sous-clé completed

b-Vérification de la validité de l'identifiant sauvegardé

Le malware compare la valeur de l'identifiant sauvegardé dans la sous-clé de registre id avec la valeur calculée et sauvegarde le résultat de la

comparaison dans une variable locale.

```
mov     edi, offset stringStrct_idSubkeyValue
lea     eax, [ebp+stringStrct_computedValueIdentifiant]
mov     edx, edi
mov     byte ptr [ebp+var_4], 9
call    compareStringStrct ; Compare id computed and id subkey value
xor     ebx, ebx
cmp     al, bl
push    4
pop     esi
lea     eax, [ebp+nbDataCompleted]
push    eax ; lpcbData
lea     eax, [ebp+dataCompleted]
push    eax ; lpData
push    ebx ; lpType
push    ebx ; int
push    offset aCompleted ; "completed"
push    [ebp+handleLockyKey] ; int
setz    [ebp+isIdSubkeyNotValid] ; 1 if idSubkey.value != valueComputed
                                ; 0 else
```

Illustration 13: Comparaison des identifiants sauvegardés dans la sous-clé de registre et celui calculé

Ensuite si le marqueur de fin est valide, le malware effectue une deuxième vérification d'égalité entre les deux identifiants. Le flux d'exécution contourne toute la charge malicieuse en cas de validité de l'identifiant de la sous-clé.

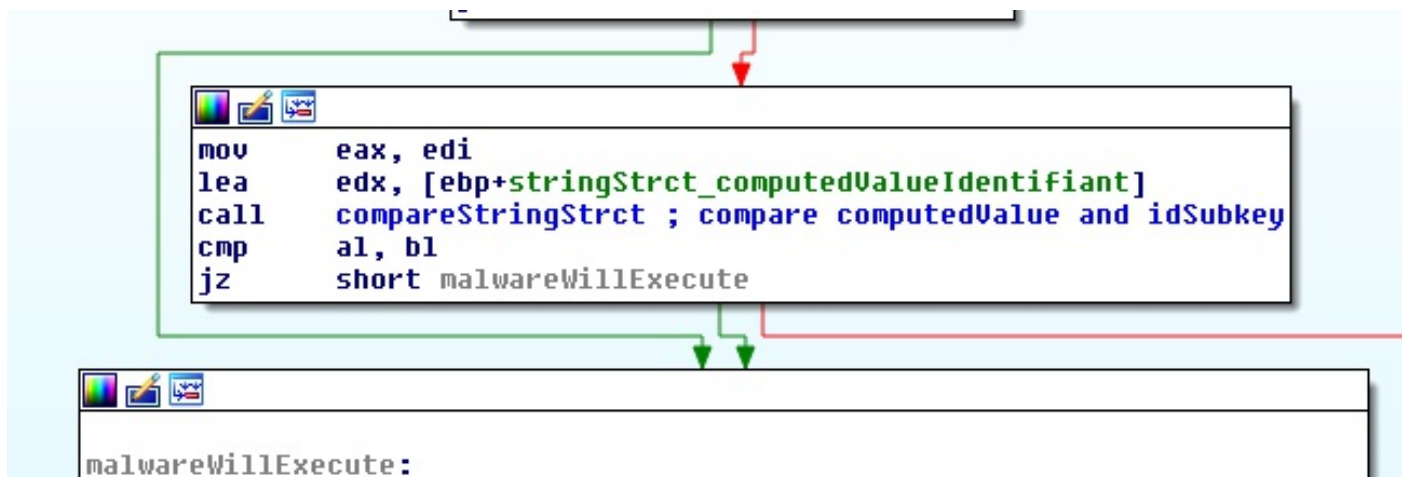


Illustration 14: Vérification de la validité de la sous-clé id

c-Contournement de la charge utile

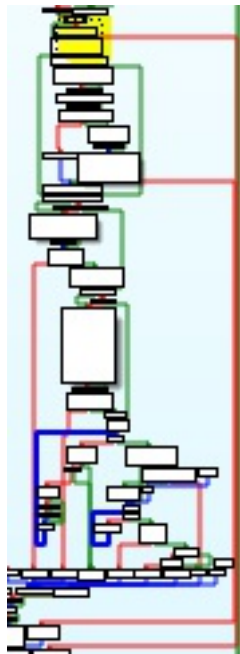


Illustration 15: Vérification de la validité de la sous-clé id (Macro)

L'illustration 15 montre que si les clés completed et id sont valides, le flux d'exécution est jusqu'à la fin du programme (Trait rouge partant du carré jaune).

10-Furtivité de l'exécution

Pour améliorer la furtivité, le malware se déplace dans un dossier temporaire sous le nom svchost.exe.

a-Choix de l'activation ou non du déplacement de l'exécutable

La procédure est conditionnée par la configuration du sample. Un champs de la configuration permet de contourner le déplacement de l'exécutable si cette valeur est nulle.

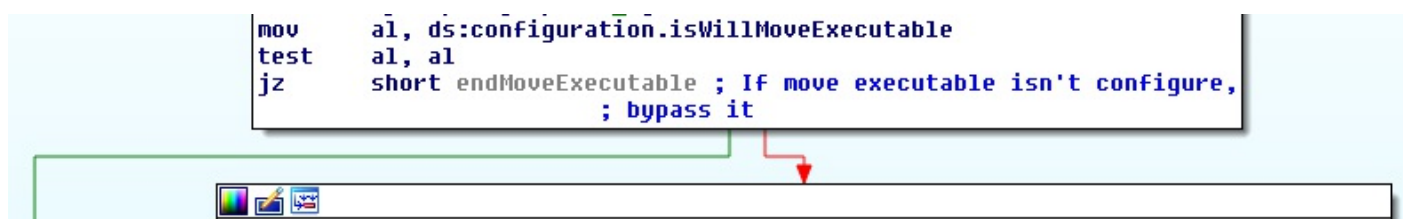


Illustration 16: Contournement du processus de furtivité suivant la configuration

```
004137E0 configuration    T_configuration <3, 7, 1Eh, 0, 0, \
004137E0                ; DATA XREF: getPubkey+298↑r
004137E0                ; WinMain(x,x,x,x):listIpAdressesEmpty↑r ...
004137E0                '31.41.47.37,188.138.88.184,91.121.97.170,5.34.183.136'>
```


Illustration 17: Configuration de la furtivité

Dans le sample étudié, la configuration désactive le déplacement de l'exécutable.

b-Récupération de l'actuel et du futur emplacements de l'exécutable

En premier, l'emplacement actuel de l'exécutable est récupéré via un simple appel à l'API Windows.

Ensuite, le chemin du futur exécutable est créé en concaténant le chemin du dossier temporaire et la chaîne "svchost.exe".

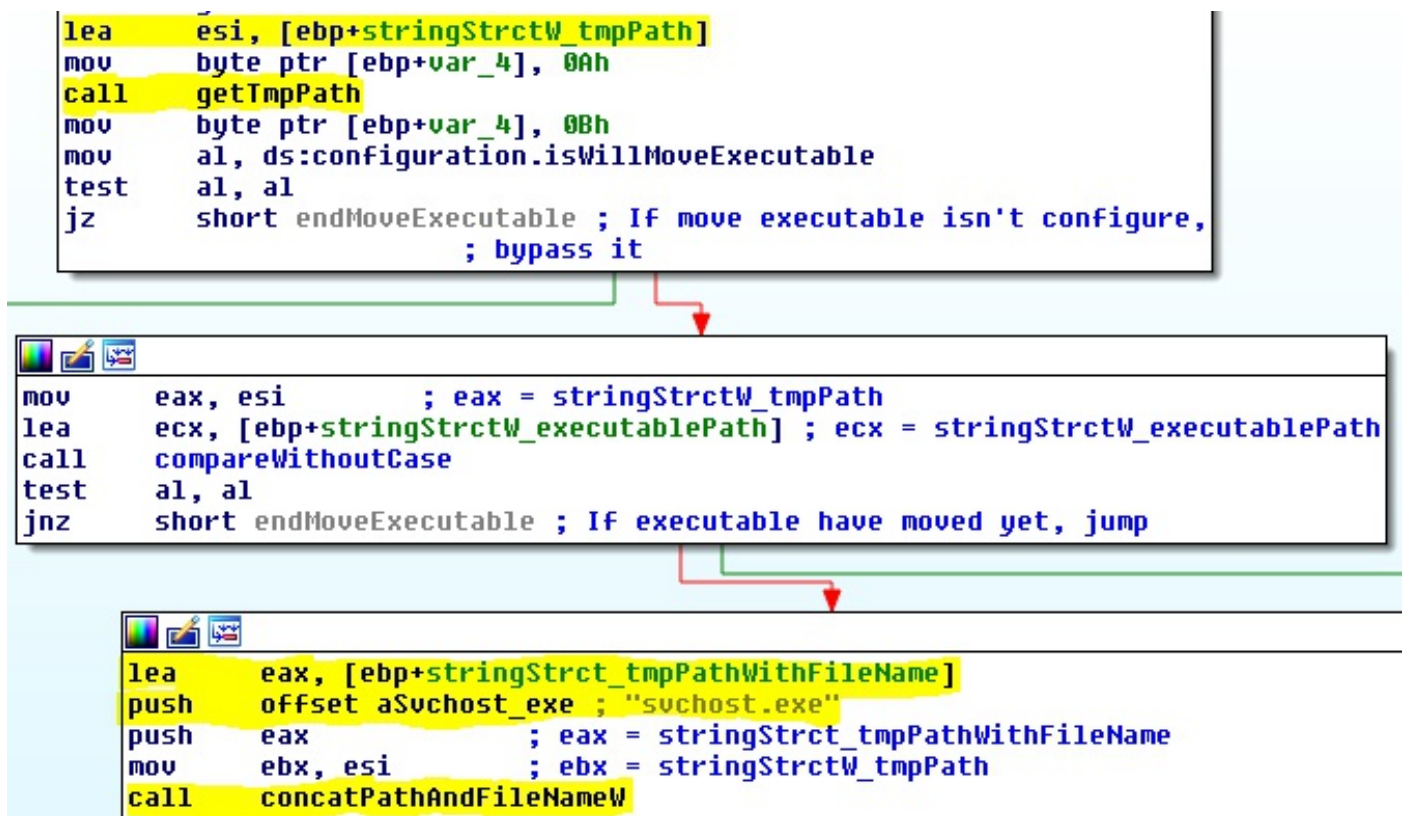
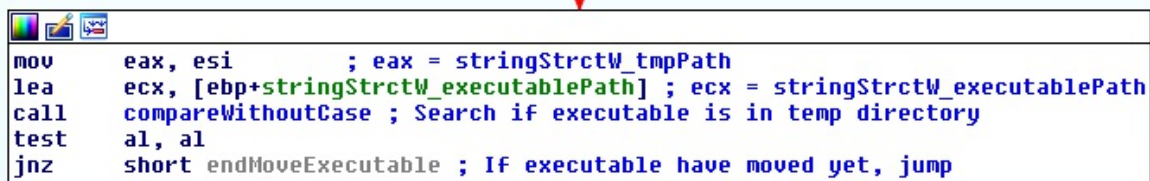


Illustration 18: Création du chemin de destination

c-Vérification de l'emplacement de l'exécutable

Le malware fait une vérification pour savoir s'il est déjà placé dans le dossier temporaire donc à l'emplacement voulu. Il compare le chemin de l'exécutable et celui du dossier temporaire. La comparaison s'arrête à la longueur du dossier temporaire pour exclure le nom du fichier. Si l'exécutable est déjà dans le bon dossier, la procédure de déplacement

n'est pas activée.



```
mov     eax, esi      ; eax = stringStrctW_tmpPath
lea     ecx, [ebp+stringStrctW_executablePath] ; ecx = stringStrctW_executablePath
call    compareWithoutCase ; Search if executable is in temp directory
test    al, al
jnz     short endMoveExecutable ; If executable have moved yet, jump
```

Illustration 19: Vérification que l'exécutable n'est pas dans le dossier temporaire

d-Copie de l'exécutable


Le fichier exécutable du malware est copié dans le dossier temporaire sous le nom "svchost.exe"

e-Suppression du fichier de zone identifier

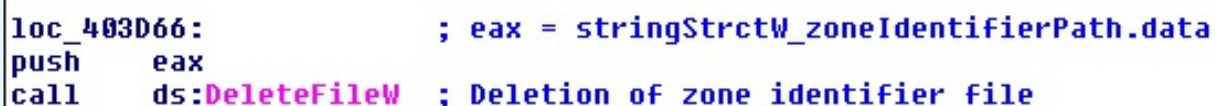
Avant le lancement du nouvel exécutable, le malware supprime le fichier zone identifier qui ferai apparaitre un message d'avertissement à l'utilisateur.



```
lea     eax, [ebp+stringStrctW_zoneIdentifierPath]
push    offset aZone_identifie ; ":Zone.Identifier"
push    eax ; int
lea     ebx, [ebp+stringStrctW_tmpPathWithFileName]
call    concatPathAndFileNameW
cmp     [eax+T_strStrctW.spaceReserved], LIMIT_MAX_DATAW_INTERNAL
pop     ecx
pop     ecx
jnb     short loc_403D66
```



```
mov     eax, dword ptr [eax+T_strStrct.data]
```



```
loc_403D66: ; eax = stringStrctW_zoneIdentifierPath.data
push     eax
call     ds:DeleteFileW ; Deletion of zone identifier file
```

Illustration 20: Suppression du fichier de zone identifier

f-Lancement du nouvel exécutable

Le fichier exécutable copié, le fichier de zone identifier supprimé, le malware peut se relancer sans éveiller les soupçons de l'utilisateur et se faire passer pour le processus légitime svchost.

g-Arret du processus parent

Si le nouveau processus s'est bien lancé, le processus parent va en fin de programme et laisse la main au fils.

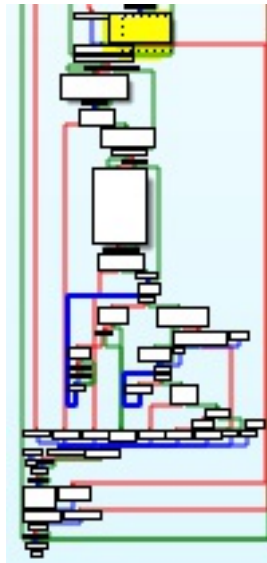


Illustration 21: Arret du processus parent

11-Récupération de la clé publique de chiffrement

a-Vérification de la validité de la clé publique en base de registre

Pour commencer le malware regarde la présence de données dans la valeur de la sous clé de registre pubkey. Si cette condition est vérifiée, il effectue une vérification sur la validité de l'identifiant stocké dans la sous clé id. Ainsi il s'assure que ce n'est pas un tier qui a placé une clé publique de chiffrement maîtrisée.

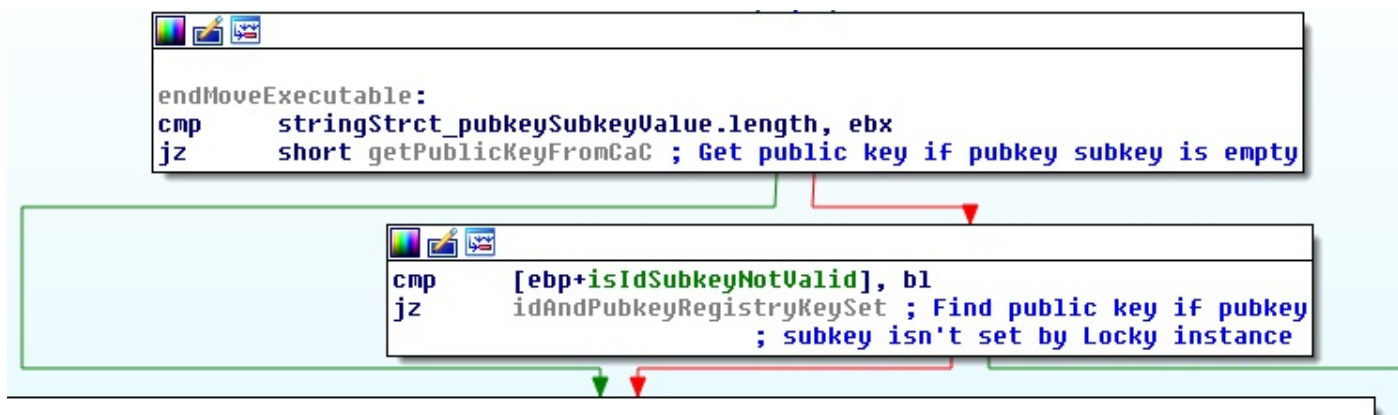


Illustration 22: Vérification de clé publique sauvegardée

b-Récupération des informations systèmes

Role de la machine

Locky définit le role d'une machine par deux critères: si elle est un serveur et si elle est dans une entreprise.

Role	Serveur/DC	Niveau dans l'entreprise
Poste autonome	0	0
Serveur autonome	1	0
Poste membre d'un AD	0	1
DC de backup	1	1
DC principal	1	2

Illustration 23: Tableau d'encodage des roles de poste infecté

- AC: Active Directory
- DC: Domain Controler

La valeur 1 dans serveur désigne un serveur ou un Domain Controler (Poste dont le fonctionneemnt est souvent très important). Le niveau dans l'entreprise est découpé en trois parties:

- 0: Non rattaché à une entreprise
- 1: Membre d'une entreprise

- 2: Membre vital d'une entreprise

Ces informations sont basées sur la fonction de l'API Windows `DsRoleGetPrimaryDomaininformation`.

Version de Windows

Le malware discrétine chaque version de Windows les numéros majeurs et mineurs de version, le type de produit (Serveur/Desktop) et une fonction propre à une version. Les versions supportées sont:

- Windows 2000
- Windows XP
- Windows 2003
- Windows 2003 R2
- Windows Vista
- Windows Server 2008
- Windows 7
- Windows 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

Le valeur de la version est ensuite encodé avec le Percent-encoding.

Architecture du processeur

Le malware récupère si le poste infecté à un processeur 32 ou 64 bits. L'information est passé sous le format binaire:

- 0: 32 bits

- 1: 64 bits

Version du Service Pack

Le malware récupère le numéro du Service Pack du système.

Langage de l'utilisateur

Le malware récupère le nom de la langue utilisée par l'interface utilisateur suivant la norme ISO 639.

Identifiant d'affiliation

Une dernière information est récoltée, il s'agit un numéro inscrit dans la configuration du sample. Plus tard envoyé sous le nom de "affid", nous avons supposé qu'il s'agit d'un numéro qui permet de pouvoir différencier les infections de différentes campagne d'attaque. Le nom de numéro d'affiliation a donc été choisi mais aucune preuve de la fonction de cette donnée ne pourra être présentée.

```

push    ds:configuration.affiliationId
lea     eax, [ebp+stringStrct_tmpConfigurationAffiliationId]
push    eax
call    setIntInStringStrct
mov     edi, eax      ; edi = stringStrct_tmpConfigurationAffiliationId
mov     ebx, [ebp+computedValueIdentifier]
lea     eax, [ebp+stringStrct_idUrlParam]
push    offset aId_0  ; "id="
push    eax
mov     byte ptr [ebp+var_4], 5
call    concat        ; stringStrct_idUrlParam =
                        ; "id=X"
push    offset aActGetkeyAffid ; "&act=getkey&affid="
push    eax
lea     eax, [ebp+stringStrct_idActUrlParam]
mov     byte ptr [ebp+var_4], 6
call    concat3       ; stringStrct_idActUrlParam =
                        ; "id=X&act=getkey&affid="
mov     ecx, eax      ; ecx = stringStrct_idActAffidUrlParam
mov     eax, edi      ; eax = stringStrct_tmpConfigurationAffiliationId
lea     edi, [ebp+stringStrct_idActAffidUrlParam]
mov     byte ptr [ebp+var_4], 7
call    concat5       ; stringStrct_idActAffidUrlParam =
                        ; "id=X&act=getkey&affid=X"
push    offset aId_0  ; "id="

```

Illustration 24: Indice sur la dernière information récupérée

c-Génération de la chaine de requête

Le malware génère une chaine qui décrit les différentes informations recueillies. Elle se présente sous le format:

```
id=X&act=getkey&affid=X&lang=X&corp=X&serv=X&os=X&sp=X&x64=X
```

- id: Identifiant de la victime
- act: Action demandé par la requête
- affid: Identifiant d'affiliation
- lang: Nom de la langue de l'utilisateur
- corp: Niveau dans l'entreprise
- serv: 0 ou 1 suivant si c'est un serveur ou non
- os: Version de Windows
- sp: Service Pack de l'OS
- x64: 0 ou 1 suivant si c'est une architecture 64 ou 32 bits

IV-Détails de fonctionnement

A-Génération de l'identifiant de victime

- Récupération du chemin du dossier Windows
- Recherche du nom du volume du point de montage
- Extraction du GUID du volume (Global Unique IDentifier)
- Hash MD5 du GUID
- Transformation du hash en caractères hexadécimaux (Majuscules)
- Selection des 16 premiers caractères du hash

B-Communication avec le C&C

Toutes les communications entre le malware et le C&C s'effectue en HTTP avec un protocole particulier.

1-Format des données en entrée

Les données à envoyées par la procédure sont sous le format:

```
field1=value1&field2=value2&field3=value3
```

Les champs sont encodés suivant le Percent-encoding.

2-Création du hash de contrôle

La première étape de la procédure est de réaliser un hash MD5 de la chaine de données.


```

addControlHashOfUrlParam: ; eax = pHandleCryptProvider
mov     eax, offset handleGlobalCryptProvider
lea     esi, [ebp+hCryptProvider_hMd5Hash_key_sizeIpList]
call    getMd5HashHandle ; hCryptProvider_hMd5Hash_indexOfIpListCurrentEnd = handleMd5Hash
mov     esi, eax          ; esi = hCryptProvider_hMd5Hash_indexOfIpListCurrentEnd
push    16                ; lengthMd5Out
lea     eax, [ebp+md5UrlParam_and_tmpIpAddress]
push    eax               ; hashDataOut
mov     eax, ebx          ; eax = stringStrct_urlParam
mov     [ebp+var_4], edi
call    hash              ; Hash URL param
                        ; eax = pHandleInstHash
push    eax               ; pHandleInstHash
call    getHashData       ; md5(stringStrct_urlParam.data)

```

*Illustration ?? : Réalisation du hash MD5 de la chaine de données
(Contenue dans stringStrct_urlParam)*

Ensuite le hash MD5 est concaténé au début de la chaine de données.

3-Chiffrement de la communication

L'étape suivante est une chiffrement maison de tout le buffer de données manipulées (MD5+données utiles). Le script suivant réimplémente la fonction de chiffrement (Coté malware) et le déchiffrement (Coté C&C):

```
#!/usr/bin/python
```

```
import sys,md5
```

```
def ror(n, dec):
    n &= 0xFFFFFFFF
    return ((n >> dec) | (n << (32-dec))) & 0xFFFFFFFF
```

```
def rol(n, dec):
    return ((n << dec) | (n >> (32-dec))) & 0xFFFFFFFF
```

```
def encryptMD5(plaintext):
    return "".join(map(lambda x:x[2:],map(hex,map(ord,encrypt(p
```

```
def encrypt(plaintext):
    m = md5.new()
    m.update(plaintext)
    m.digest()
```

```

md5s = m.digest()
data = map(ord,md5s+plaintext)
key = 0xCD43EF19
for i in range(len(data)):
    currentChar = data[i]
    data[i] = (((ror(key, 5) & 0xFF) - \
        rol(i, 0xD) & 0xFF) & 0xFF) ^ (data[i] & 0xFF)
    key = (rol(currentChar, i%32) + ror(key,1)) ^ \
        (ror(i, 23) + 0x53702F68)
return "".join(map(chr,data))

```

```

def decrypt(cipher):
    data = map(ord,cipher)
    key = 0xCD43EF19
    dec = []
    for i, e in enumerate(data):
        currentChar = data[i]
        dec.append((((ror(key, 5) & 0xFF) - \
            rol(i, 0xD) & 0xFF) & 0xFF) ^ currentChar)
        key = (rol(dec[i], i%32) + ror(key,1)) ^ \
            (ror(i, 23) + 0x53702F68)
    return "".join(map(chr,dec[16:]))

```

```

if __name__=='__main__':
    if(len(sys.argv) != 2):
        print "Usage: "+sys.argv[0]+" plaintext"
        exit(0)

```

```

plaintext = sys.argv[1]

```

```

print "=== Plaintext ==="
print plaintext
print "=== MD5(cipher(Plaintext)) ==="
print encryptMD5(plaintext)
print "=== Decrypt(cipher(Plaintext)) ==="
print decrypt(encrypt(plaintext))

```

4-Mélange de la liste d'adresses IP de C&C prédéfinies

L'étape suivante est de mélanger la liste d'adresses IP créée au début de l'infection par le malware. Ceci dans le but de rendre aléatoire l'ordre des tentatives de connexion au C&C lors d'une prochaine étape.

5-Temporisation

Le malware met en pause sont processus un temps aléatoire entre 10 et 20 secondes. L'objectif pourrait être d'augmenter la furtivité en ne s'arrêtant pas une durée fixe et en n'affilant pas les tentative de connexions trop vite?

6-Sélection du moyen de récupération de l'adresse de C&C

Un compteur itéré à chaque passage dans cette portion de code permet de savoir si les IP fixes de configuration ou l'algorithme de DGA. Tous les premieres tentatives seront sur les IP fixes puis dès que toute la liste a été tentée, le DGA est utilisé.

```
mov     eax, ipList.end ; eax = ipList.end
mov     ecx, ipList.begin ; ecx = ipList.begin
sub     eax, ecx        ; eax = space between ipList.end and ipList.begin
cdq
push    STRINGSTRUCT_LENGTH
pop     edi
idiv    edi             ; eax = size of ipList
xor     edx, edx        ; edx = 0
mov     ebx, eax        ; ebx = size of ipList
lea     eax, [ebx+8]    ; eax = size of ipList + number Domain generate by DGA
mov     [ebp+numberTmp], eax ; numberTmp = number of choices
mov     eax, compteurTryCaCConnection
div     [ebp+numberTmp] ; edx = compteurTryCaCConnection % numberTmp
inc     compteurTryCaCConnection ; compteurTryCaCConnection++
cmp     edx, ebx
jnb     short useDGA    ; If compteurTryCaCConnection lesser than number
                        ; configuration addresses, go on it, else go to DGA
```

useConfigurationIpAddressesList: ; edx = indexChosen * sizeof(stringStruct)

useDGA: ; ecx = indexOfChoice

Illustration ?? : Choix de la méthode de sélection de C&C

7-Sélection de l'adresse de C&C utilisée

Suite à l'étape précédente une des deux méthodes de sélection de l'adresse de C&C est choisie. Elles sont décrites dans les deux prochaines parties.

a-Sélection d'une adresse IP de la liste prédéfinie

Le compteur utilisé pour choisir entre les deux méthodes permet aussi de savoir quelle adresse IP de la liste est sélectionnée. Elles le sont toutes l'une après l'autre.

```
mov     eax, iplist.end ; eax = iplist.end
mov     ecx, iplist.begin ; ecx = iplist.begin
sub     eax, ecx        ; eax = space between iplist.end and iplist.begin
cdq
push    STRINGSTRUCT_LENGTH
pop     edi
idiv    edi             ; eax = size of iplist
xor     edx, edx        ; edx = 0
mov     ebx, eax        ; ebx = size of iplist
lea     eax, [ebx+8]    ; eax = size of iplist + number Domain generate by DGA
mov     [ebp+numberTmp], eax ; numberTmp = number of choices
mov     eax, compteurTryCaCConnection
div     [ebp+numberTmp], eax ; edx = compteurTryCaCConnection % numberTmp
inc     [ebp+numberTmp] ; compteurTryCaCConnection ; compteurTryCaCConnection++
cmp     edx, ebx
jnb     short useDGA    ; If compteurTryCaCConnection lesser than number
                        ; configuration addresses, go on it, else go to DGA
```

useConfigurationIpAddressesList: ; edx = indexChosen * sizeof(stringStruct)
useDGA: ; ecx = indexOfChoice

Illustration ?? : Choix d'une adresse IP de C&C dans la liste

Le mélange de la liste quelques étapes avant permet de ne pas avoir un ordre de tentatives de connexions fixe.

b-Utilisation du Domain Generation Algorithm

La deuxième méthode permettant une plus grande robustesse du malware est de générer un nom d'hôte de C&C grâce à un DGA. Cette partie va décrire les caractéristiques de celui-ci.

Premièrement, le DGA reçoit une entrée qui est une valeur entre 0 et 8. Injecté dans le calcul cela permet d'avoir 8 noms d'hôte de C&C différents possiblement générés à un même instant. Cette entrée est définie par le compteur précédemment utilisé manipulé.

Caractéristiques:

- Nombre de noms d'hôtes possiblement générés à un même instant: 8
- Fréquence de changement de noms d'hôtes générés: Tous les 2 jours
- Longueur du nom d'hôte généré: 8 à 18 caractères
 - TLD de 2 caractères
 - Préfixe de 5 à 15 caractères
- Préfixe constitué de caractère de 'a' à 'y'
- TLD utilisés: ru, pw, eu, in, yt, pm, us, fr, de, it, be, uk, nl, tf

Le script suivant réimplémente la deuxième version du DGA de Locky (Celle du sample étudié). Il permet de prévoir les différents noms d'hôtes générés pour une date voulue.

```
import sys, argparse, datetime

#
# DGA from Locky sample 45f4c705c8f4351e925aea2eb0a7f564
# Locky's DGA version 2
# Author: Adrien Coueron
#

def ror(n, dec):
    n &= BIG_INT
    return ((n >> dec) | (n << (32-dec))) & BIG_INT

def rol(n, dec):
    return ((n << dec) | (n >> (32-dec))) & BIG_INT

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("-d", "--date",
        help="Date pour laquelle les domaines doivent etre genere",
    parser.add_argument("-c", "--config",
        type=int, help="Configuration du DGA")
```

```

args = parser.parse_args()

if args.date:
    d = datetime.datetime.strptime(args.date, "%d/%m/%Y")
else:
    d = datetime.datetime.now()
if args.config:
    dgaConfiguration = args.config
else:
    dgaConfiguration = 7

BIG_INT = 0xFFFFFFFF

day = d.day
year = d.year
month = d.month
tld = ['ru', 'pw', 'eu', 'in', 'yt', 'pm', 'us',
        'fr', 'de', 'it', 'be', 'uk', 'nl', 'tf']

#Generation des 6 domaines pour la date donnee
for n in range(8):
    #Traitement sur les donnees de date et de configuration
    valueComputed = (((ror((((((ror((((ror((((ror((((ror(
        ((year + 0x1BF5) * 0xB11924E1) & BIG_INT, 7) +
        dgaConfiguration + 0x27100001) & BIG_INT) *
        0xB11924E1) & BIG_INT, 7)) + (day >> 1) +
        0x27100001) & BIG_INT) * 0xB11924E1) & BIG_INT, 7)) +
        month + 0x2709A354) & BIG_INT) * 0xB11924E1) &
        BIG_INT, 7) + rol(n & 7, 0x15)) & BIG_INT) +
        rol(dgaConfiguration, 0x11) + 0x27100001) &
        BIG_INT) * 0xB11924E1) & BIG_INT), 7)) + 0x27100001) &
        BIG_INT
    ####
    #Choix de la longueur du nom de domaine
    lengthDomainNamePrefix = (valueComputed % 11) + 5
    ####
    #Generation de chaque caractere
    domain = ""
    for i in range(lengthDomainNamePrefix):

```

```

        valueComputed = (ror((rol(valueComputed, i) *
        0xB11924E1) & BIG_INT, 7) + 0x27100001) & BIG_INT
        domain += chr((valueComputed % 25) + ord('a'))
####
#Ajout du TLD
domain += '.'
domain += tld[((ror((valueComputed * 0xB11924E1) &
        BIG_INT, 7) + 0x27100001) & BIG_INT) % 14]
####
print domain

```

Tout d'abord Locky effectue un traitement sur les entrées (date, configuration et index dans la liste des DGA possibles). De ce calcul définit la taille du nom d'hôte généré:

```

add     eax, ebx          ; eax = ror(eax = (var6 * 0xB11924E1) & 0xF1
push    11
mov     [ebp+valueComputed], eax
xor     edx, edx          ; edx = 0
pop     ecx               ; ecx = 11
div     ecx               ; edx = var7 % 11
lea     esi, [ebp+stringStrct_tmpDga]
lea     edi, [edx+5]       ; edi E [5..15] = length domain name prefix
lea     eax, [edi+3]       ; eax E [8..18] = length domain name

```

Illustration ?? : Choix de la longueur du nom d'hôte généré

Ensuite pour chaque caractère généré, cette valeur précédemment calculée subit un nouveau traitement et définit le nouveau caractère.

```

choiceNewChar:           ; ecx = stringStrct_tmpDga
xor     edx, edx          ; edx = 0
push    25
pop     esi               ; esi = 25
div     esi               ; edx = valueComputed % 25
mov     eax, [ebp+index]  ; eax = index
add     dl, 'a'           ; dl = offsetLetter + 'a'
inc     [ebp+index]
mov     [ecx+eax], dl     ; set chosen char in stringStrct of domain

```

Illustration ?? : Choix du nouveau caractère

Sources

[MSDN](#)

[Wikipedia](#)

[Récupération du sample - Github - eyecatchup](#)