COMPRENDRE

LE

Solution rédigée par BeatriX^FRET

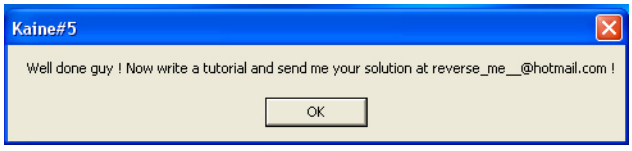
SOMMAIRE

1. INTRODUCTION

Je vais vous présenter une analyse détaillée du crackme le plus robuste que j'ai eu à étudier depuis que je fais du reversing. Il s'agit du 5ème crackme de Kaine, Kaine#5.exe.

Ce binaire est à classer dans la catégorie des « KeyFiles ». Ceci signifie qu'il exige la présence sur le disque C : d'un fichier contenant les informations nécessaires pour l'enregistrement. Au cas où le fichier n'existe pas ou s'il est mal rempli, le binaire vous envoie sans que vous n'ayez pu faire quoique ce soit le message suivant :

Au contraire, en cas de succès, c'est-à-dire si vous disposez du fichier « C:\License.key » rempli de cette chaine de caractères « JQVCSMHTFP », vous avez droit à ce petit message :



Pour réussir ce défi, il fallait donc dans l'ordre :

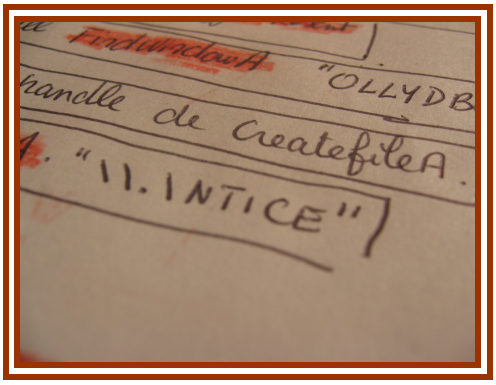
- Savoir qu'il fallait créer un keyfile !
- Savoir qu'il s'appelait License.key (et non Licence.key comme j'ai pu le croire pendant des heures !!)
- Savoir ce qu'il devait contenir, i.e une chaine de 10 caractères « JQVCSMHTFP ».

Dans le principe, Kaine#5.exe essaie d'ouvrir le fichier C:\License.key et le cas échéant effectue une série de tests sur son contenu pour voir s'il s'agit du bon code.

Dans la pratique, Kaine#5.exe est équipé d'une véritable armure numérique et son système de défense est redoutable. Son auteur est un adepte des techniques de packing et connaît très bien la plupart des grandes protections du marché actuel. C'est en plus un excellent codeur qui maîtrise parfaitement le user mode (ring3) et le kernel mode (ring0). Je ne pourrais pas lister toutes les techniques utilisées pour lutter contre toute analyse, néanmoins voici les grandes lignes :

- Kaine#5.exe est compressé par l'ApLib
- Kaine#5.exe est crypté par plusieurs centaines de couches de cryptages.
- Le code décrypté est « noyé » dans du junk code et a subi de nombreuses mutations.
- Les fonctions utilisées sont redirigées ou émulées systématiquement.
- Certaines fonctions sont non documentées (issues de ntdll.dll et ntoskrnl.exe)
- Kaine#5.exe utilise plus de 10 tricks anti-debugging. (Anti-OllyDbg, anti-Soft Ice, risques de BSOD)
- Kaine#5.exe utilise un driver pour hooker les interruptions logicielles INT1 et INT3.
- Kaine#5.exe utilise un second driver pour tester c:\License.key.
- Les deux drivers sont compressés et cryptés. Pire, le second driver démarre seulement en présence du premier. A défaut, le système se plante.
- Le second driver utilise une VM (machine virtuelle) obfusquée pour tester le contenu du fichier License.key. Le pseudo-code utilisé est également obfusqué.

Il m'a fallu 9 jours pour parvenir à faire céder la protection de ce binaire. 9 jours de travail acharné et de grosses galères ! BeatriX – 3 juin 2005 –



JOUR 1 : Samedi 21 mai 2005 : 11h00

Voilà bientôt 24h que le défi de Kaine a été posté sur FC .Kaine a eu la gentillesse de me prévenir par MP… et je commence tout juste à m'y intéresser. Kharneth est aussi sur le coup il me semble. Hmmm…. j'inspire profondément avant de me lancer. Je sais que ça va être long et éprouvant. Je sais aussi que Kharneth est très rapide…Je démarre OllyDbg…la grande bataille vient de commencer !

MA DEMARCHE

Comment m'y suis-je pris pour aborder ce binaire ?

Tout au long de cet article, je vous présenterai ma façon d'opérer pour contourner tel ou tel obstacle. Je précise néanmoins que je ne prétend pas proposer ici des techniques d'analyse subtiles ou efficaces. Et pour cause : j'ai tracé assez « bêtement » tout le crackme. Il m'a fallu des heures de patience et des tonnes de jurons pour réussir à atteindre le ExitThread final !

Je vous incite d'ailleurs vivement dans un premier temps à tracer pour voir ce qu'il se passe réellement. L'apprentissage du RE passe aussi par une bonne dose d'acharnement. Dans une démarche d'analyse d'une protection, on avance a priori en aveugle. Quelques notions de packing peuvent nous faire gagner du temps mais cela ne peut pas nous servir à « deviner » les intentions de l'auteur.

J'ai utilisé les outils classiques suivants pour reverser ce binaire : **OllyDebugger** 1.10 , Le plugin HideDebugger.dll pour OllyDbg, **LordPE** Deluxe de Yoda , **IDA Pro** 4.7, **Ultra Edit** 10.20d. J'ai également fait mes premières armes sur **Soft Ice Driver studio** 3.2 pour m'amuser à tracer les drivers (mais ce n'est pas indispensable).

2. SCHEMA DU CRACKME

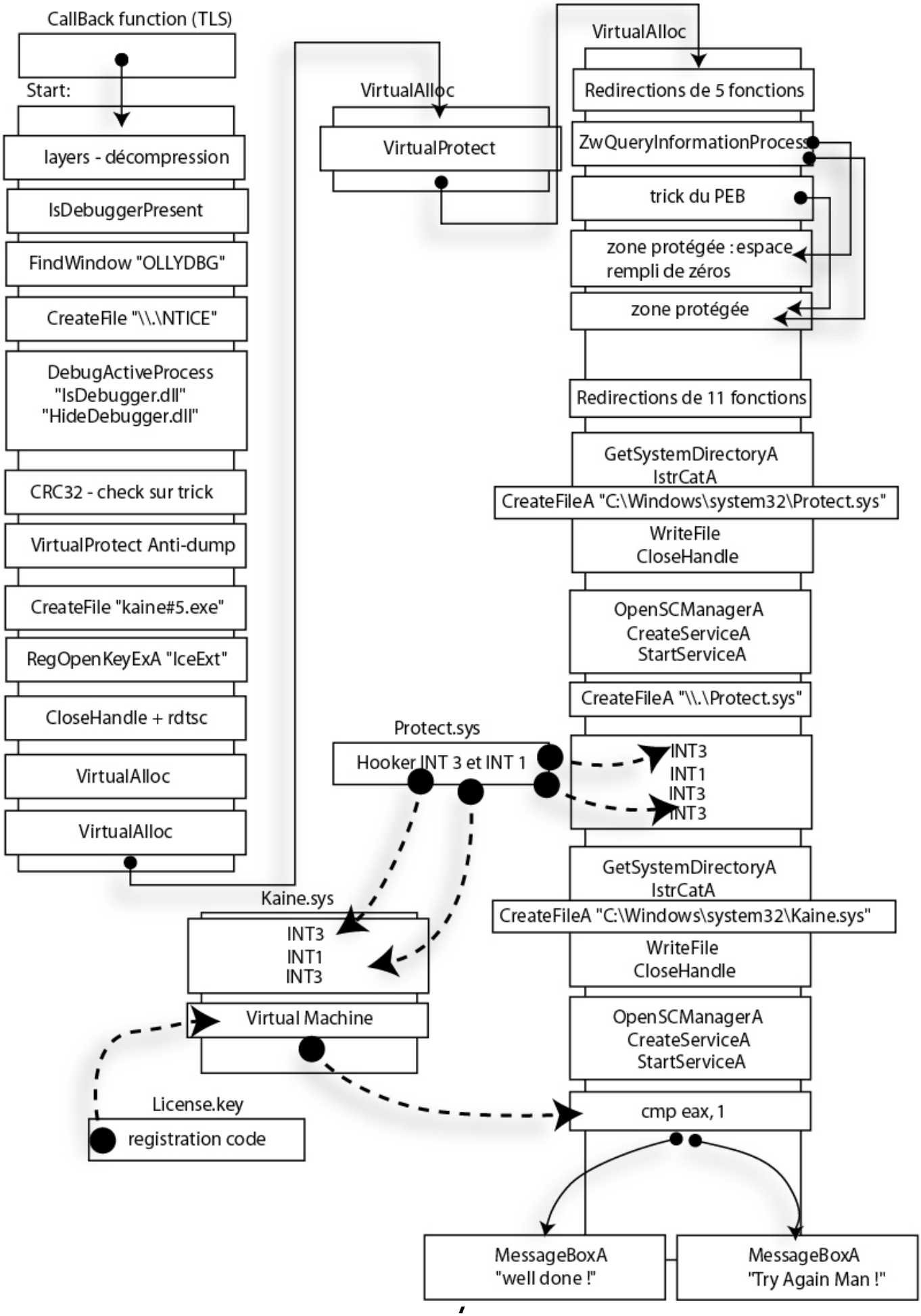
Pour commencer, je vais vous présenter une vue d'ensemble de ce crackme par un survol rapide de son comportement et notamment de son déroulement. Vous trouverez ci-dessous (fig.1) la totalité de la structure du crackme. Comme vous pouvez le constater sur la figure, le programme se compose de plusieurs parties :

- La première partie, située dans les sections du binaire constitue un premier rempart contre toute tentative d'analyse. Elle renferme des techniques d'anti-tracing et

d'anti-debugging mais ne présente en soi que l'entrée en matière.

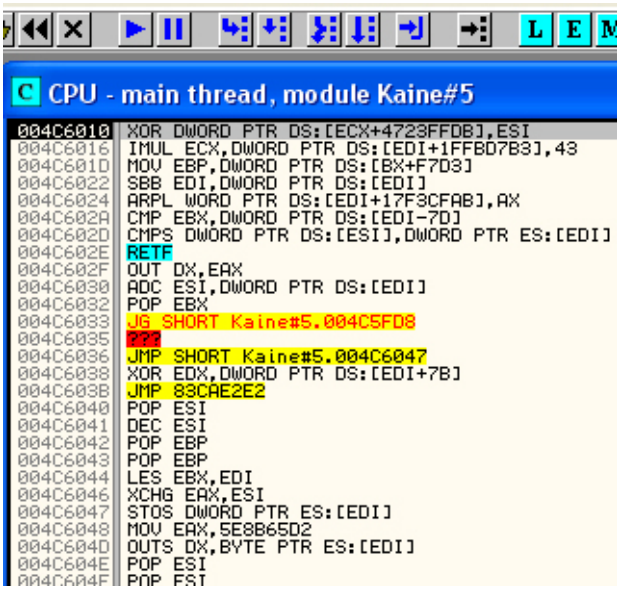
● La seconde partie, située dans une zone allouée par un appel à VirtualAlloc est nettement plus robuste et ardue à analyser. Elle s'occupe de la redirection des fonctions et est chargée de lancer 2 drivers en mémoire « Protect.sys » et « Kaine.sys ». Le premier driver permet de hooker les interruptions logicielles INT1 et INT3 ce qui met en très grande difficulté un debugger quelconque. Ces hooks sont de surcroît utilisés pour décrypter des portions du binaire Kaine#5.exe ainsi que des portions du second driver.

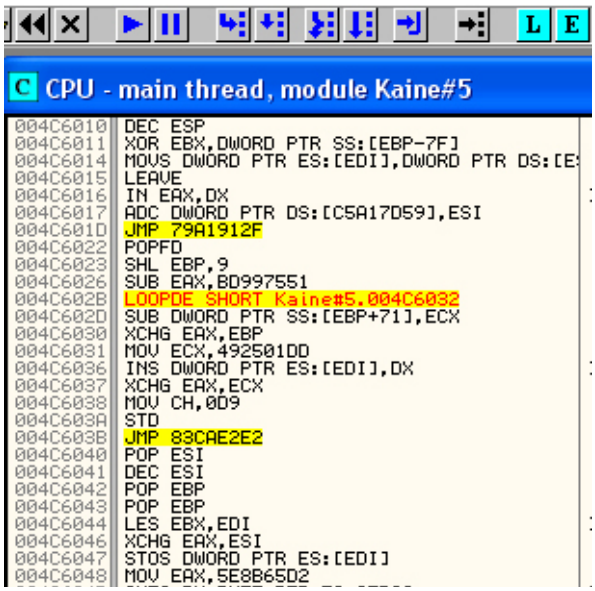
● La troisième partie, le driver « Kaine.sys », est chargée de tester la présence du keyfile ainsi que de la validité de la clé. Ce driver ne démarre qu'en présence du driver Protect.sys afin de décrypter ses différents layers (plus d'une centaine). Kaine.sys es équipé d'une VM chargée d'exécuter un pseudo-code obfusqué qui testera la validité de la clé.


CallBack function (TLS)
Start:
layers - décompression
IsDebuggerPresent
FindWindow "OLLYDBG"
CreateFile "\\.\NTICE"
DebugActiveProcess
"IsDebugger.dll"
"HideDebugger.dll"
CRC32 - check sur trick
VirtualProtect Anti-dump
CreateFile "kaine#5.exe"
RegOpenKeyExA "IceExt"
CloseHandle + rdtsc
VirtualAlloc
VirtualAlloc
VirtualAlloc
VirtualProtect
VirtualAlloc
Redirections de 5 fonctions
ZwQueryInformationProcess
trick du PEB
zone protégée : espace rempli de zéros
zone protégée
Redirections de 11 fonctions
GetSystemDirectoryA
lstrCatA
CreateFileA "C:\Windows\system32\Protect.sys"
WriteFile
CloseHandle
OpenSCManagerA
CreateServiceA
StartServiceA
CreateFileA "\\.\Protect.sys"
Protect.sys
Hooker INT 3 et INT 1
INT3
INT1
INT3
INT3
GetSystemDirectoryA
lstrCatA
CreateFileA "C:\Windows\system32\Kaine.sys"
WriteFile
CloseHandle
OpenSCManagerA
CreateServiceA
StartServiceA
Kaine.sys
INT3
INT1
INT3
Virtual Machine
License.key
registration code
cmp eax, 1
MessageBoxA
"well done !"
MessageBoxA
"Try Again Man !"

3. CALLBACK FUNCTION - TLS

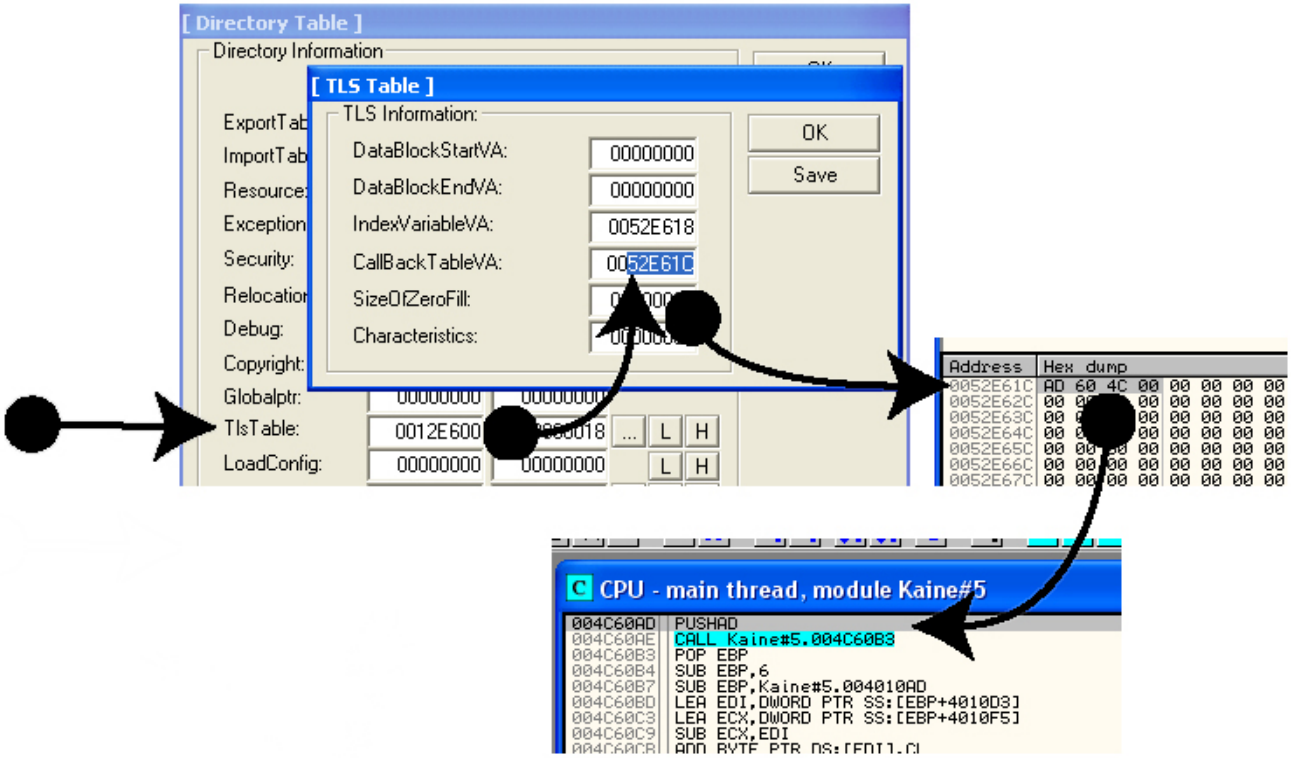
Les utilisateurs de Windows 2000 ont dû avoir une jolie surprise dès le premier désassemblage du binaire. A chaque lancement de OllyDbg, vous constatez en effet que le code situé entre les adresses 4C6010 et 4C603B change. Ceci est d'autant plus surprenant que le code qui y figure ne semble pas être correct et génère des exceptions… voici deux situations vues lors de deux désassemblages différents :





L'explication est à chercher du côté des TLS (Thread Local Storage). Il s'agit en fait d'exploiter ces fameux TLS pour exécuter une routine avant que le debugger breake sur l'Entry Point. La technique est semble-t-il connue et issue du monde des virus. Peter Szor l'explique très bien dans son ouvrage « Virus Research and Defense » à la page 154. Le Main entry point n'est pas nécessairement le premier entry point. Si le système détecte des entrées dans la table des TLS, il exécutera ces routines avant d'exécuter le Main Entry Point. Peter Ferry fit une démonstration de cette propriété en l'an 2000 durant des recherches heuristiques pour Symantec.

Regardons de plus près comment cela fonctionne. Si vous ouvrez le binaire avec LordPE, vous voyez clairement une entrée dans la Directory TLS. Dans la section CallBack Function, vous pouvez y voir une adresse hard codée qui est en fait un pointeur vers une table dans le binaire. Cette table est composée à son tour d'adresses qui vont chacune pointer vers une routine qui sera exécutée avant l'ENTRY POINT. Cette table se termine par 0.



Ce binaire appelle donc une routine située en 4C60AD avant de démarrer le programme principal. Cette fonction a pour objectif de remplir l'espace 4C6010-4C603B (Main Entry Point) d'octets choisis aléatoirement au cas où il y aurait un CCh (INT3) posé sur L'entry Point. Je précise à toute fin utile qu'un debugger breake à l'entry point justement parce qu'il pose (sans vous avertir) un BPX (INT3) sur le premier octet de l'Entry Point. Voici le code commenté de cette routine maléfique :

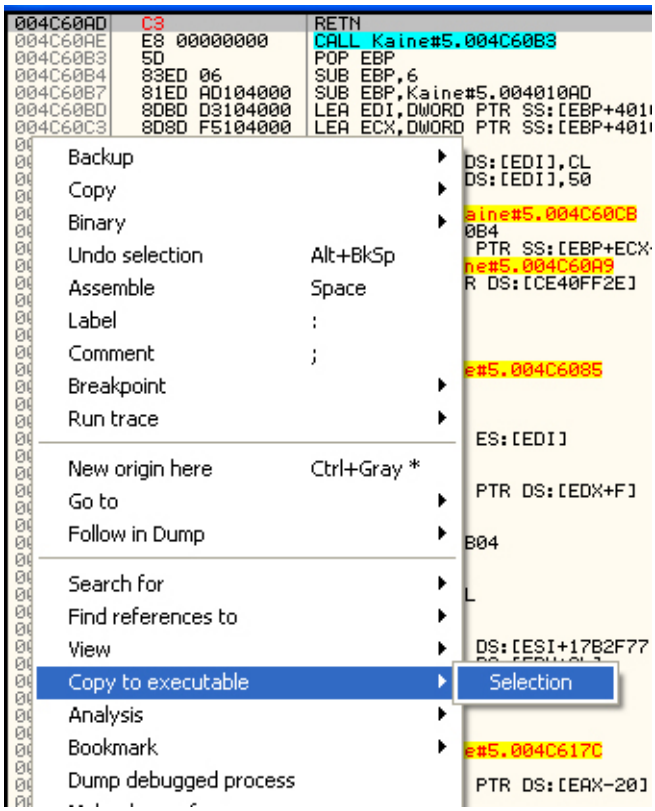
```
004C60AD  PUSHAD
004C60AE  CALL Kaine#5.004C60B3
004C60B3  POP EBP
004C60B4  SUB EBP,6
004C60B7  SUB EBP,Kaine#5.004010AD
004C60BD  LEA EDI,DWORD PTR SS:[EBP+4010D3]
004C60C3  LEA ECX,DWORD PTR SS:[EBP+4010F5]
004C60C9  SUB ECX,EDI
004C60CB  ADD BYTE PTR DS:[EDI],CL
004C60CD  XOR BYTE PTR DS:[EDI],50
004C60D0  INC EDI
004C60D1  LOOPD SHORT Kaine#5.004C60CB
004C60D3  LEA EAX, DWORD PTR SS:[EBP+401010]
004C60D9  CMP BYTE PTR DS:[EAX] ,0CC
004C60DC  JE SHORT Kaine#5.004C60E0
004C60DE  POPAD
004C60DF  RETN
004C60E0  LEA EDI, DWORD PTR SS:[EBP+401010]
004C60E6  LEA ECX, DWORD PTR SS:[EBP+40103B]
004C60EC  SUB ECX, EDI
004C60EE  RDTSC
004C60F0  STOS BYTE PTR ES:[EDI]
004C60F1  LOOPD SHORT Kaine#5.004C60EE
004C60F3  POPAD
004C60F4  RETN
```

décrypte la routine qui va détruire l'entry point

test la présence d'un INT3

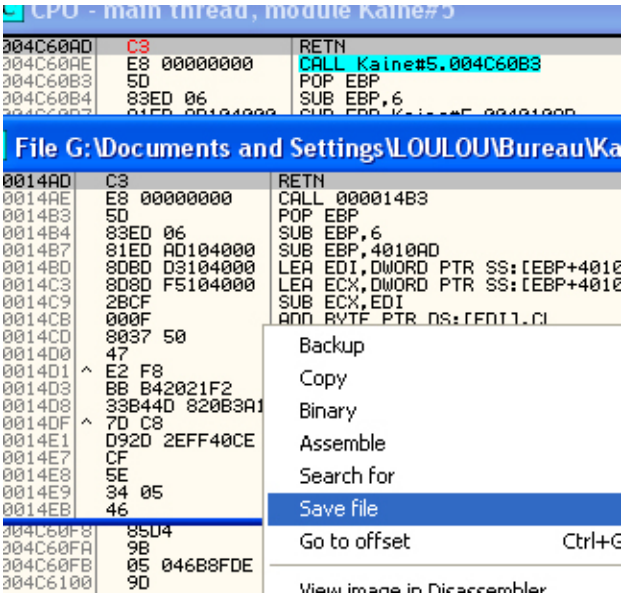
destruction de l'EP en utilisant l'instruction RDTSC

Pour contourner ce genre de technique, il suffit de poser un RET en 4C60AD et d'enregistrer la modification comme vous pouvez le voir ci-dessous :



1 . Taper RET en 4C60AD

2 . « Click droit » puis « Copy to exécutable »



3 . « click droit » à nouveau puis « save file »

4 . LA DECOMPRESSION - ApLib

L'ApLib n'est plus à présenter. Il s'agit de la fameuse lib de Joergen Ibsen pour compresser des données suivant le principe de compression de Lempel et Ziv LZ78. Ce genre de compression est basé sur l'existence de redondances dans toute chaîne de données. En utilisant un dictionnaire, on peut donc limiter les répétitions et ainsi réduire de façon significative l'espace occupé. Dans le K5, l'ApLib est utilisé à plusieurs reprises et peut se repérer si l'on sait à quoi ressemble l'algorithme de décompression. L'intérêt d'utiliser ce genre de compression est évidemment de réduire la taille du binaire mais également de « crypter » certaines parties du binaire. Il est utilisé notamment pour décompresser la section code du binaire, le lanceur de drivers situé dans une zone virtuelle, les deux drivers eux même.

K5 a utilisé pour la compression la fonction **aP_pack** de aplib.lib. Il utilise donc la routine de décompression nommée **aP_depack** (elle n'est pas contenue dans la lib). Elle est très facile à repérer de par ses très nombreux sauts conditionnels. Voici cette routine :

```
;;
;; aPLib compression library  -  the smaller the better :)
;;
;; TASM / MASM / WASM assembler depacker
;;
;; Copyright (c) 1998-2004 by Joergen Ibsen / Jibz
;; All Rights Reserved
;;
;; http://www.ibsensoftware.com/
;;

.386p
.MODEL flat

.CODE

PUBLIC _aP_depack_asm

_aP_depack_asm:
   pushad

   mov    esi, [esp + 36]    ; C calling convention
   mov    edi, [esp + 40]

   cld
   mov    dl, 80h
   xor    ebx, ebx

literal:
   movsb
   mov    bl, 2
nexttag:
   call   getbit
   jnc    literal

   xor    ecx, ecx
   call   getbit
   jnc    codepair
   xor    eax, eax
   call   getbit
   jnc    shortmatch
   mov    bl, 2
   inc    ecx
   mov    al, 10h
getmorebits:
   call   getbit
   adc    al, al
   jnc    getmorebits
   jnz    domatch
   stosb
   jmp    short nexttag
codepair:
   call   getgamma_no_ecx
   sub    ecx, ebx
   jnz    normalcodepair
   call   getgamma
   jmp    short domatch_lastpos

shortmatch:
   lodsb
   shr    eax, 1
   jz     donedepacking
   adc    ecx, ecx
   jmp    short domatch_with_2inc

normalcodepair:
   xchg   eax, ecx
   dec    eax
   shl    eax, 8
   lodsb
   call   getgamma
   cmp    eax, 32000
   jae    domatch_with_2inc
   cmp    ah, 5
   jae    domatch_with_inc
   cmp    eax, 7fh
   ja     domatch_new_lastpos

domatch_with_2inc:
   inc    ecx

domatch_with_inc:
   inc    ecx
```

```
domatch_new_lastpos:
   xchg   eax, ebp
domatch_lastpos:
   mov    eax, ebp

   mov    bl, 1

domatch:
   push   esi
   mov    esi, edi
   sub    esi, eax
   rep    movsb
   pop    esi
   jmp    short nexttag

getbit:
   add    dl, dl
   jnz    stillbitsleft
   mov    dl, [esi]
   inc    esi
   adc    dl, dl
stillbitsleft:
   ret

getgamma:
   xor    ecx, ecx
getgamma_no_ecx:
   inc    ecx
getgammaloop:
   call   getbit
   adc    ecx, ecx
   call   getbit
   jc     getgammaloop
   ret

donedepacking:
   sub    edi, [esp + 40]
   mov    [esp + 28], edi    ; return unpacked length in
eax

   popad
   ret
```

On peut légitimement se demander comment on peut utiliser l'ApLib pour compresser un binaire et surtout comment l'appliquer à un crackme tel que le K5. En fait, le crackme est fabriqué en plusieurs étapes. On commence par coder la partie du crackme que l'on veut compresser ; ce code doit être relogeable dans le cas où il est décompressé dans une zone virtuelle. Puis, on applique la compression en prenant soin d'introduire le décompresseur. Puis, on utilise ce nouveau binaire auquel on ajoute un nouveau code etc… Voici un exemple extrêmement simple d'une utilisation possible de la compression et de la décompression à l'aide de l'ApLib. Pour faire fonctionner le binaire, vous devez disposer au moment de la compilation de la lib aplib.lib et du fichier aplib.inc (que vous trouvez dans les sources fournies par l'archive de Joergen Ibsen). Ce petit exemple commence par compresser un bout de code complètement farfelu relogeable, puis le décompresse dans une zone virtuelle et enfin l'exécute.

```
.586
.Model Flat ,StdCall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32.lib
include aplib.inc
includelib aplib.lib
.Data

zone_de_travail     DWORD 0
code_compresse      DWORD 0
code_decompresse    DWORD 0
longueur            DWORD 0
.Code
Main:
  mov eax, offset fin_code_intial
  sub eax, offset code_intial
  mov longueur, eax

; ************************************* Alloue une zone de travail

  push longueur
  call aP_max_packed_size
```

```
  push PAGE_READWRITE
  push MEM_COMMIT + MEM_RESERVE
  push eax
  push 0
  call VirtualAlloc
  mov zone_de_travail,eax

; ************************************ Allouer une zone qui contiendra le code compressé
  push longueur
  call aP_workmem_size

  push PAGE_READWRITE
  push MEM_COMMIT + MEM_RESERVE
  push eax
  push 0
  call VirtualAlloc
  mov code_compresse,eax

; ************************************ Allouer une zone qui contiendra le code décompressé

  push PAGE_READWRITE
  push MEM_COMMIT + MEM_RESERVE
  push longueur
  push 0
  call VirtualAlloc
  mov code_decompresse,eax

; ************************************ Compresser le code situé à code_intial
  push 0
  push 0
  push zone_de_travail
  push longueur
  push code_compresse
  push offset code_intial
  call aP_pack

; ************************************ Decompresser le code situé dans code_compresse

  push code_decompresse
  push code_compresse
  call aP_depack

; ************************************ utiliser le code décompressé situé dans code_decompresse
  call code_decompresse

; ************************************ Libérer les espaces alloués

  push longueur
  call aP_max_packed_size

  push MEM_DECOMMIT
  push eax
  push zone_de_travail
  call VirtualFree

  push longueur
  call aP_workmem_size

  push MEM_DECOMMIT
  push eax
```

```
  push code_compresse
  call VirtualFree

  push MEM_DECOMMIT
  push longueur
  push code_decompresse
  call VirtualFree

  push 0
  call ExitProcess

; ***************************** code à compresser/décompresser

code_intial:
  push eax
  push ebx
  add eax, ebx
  inc eax
  dec eax
  inc eax
  dec eax
  nop
  nop
  nop
  pop ebx
  pop eax
  ret

fin_code_intial:

; ***************************** routine de décompression de l'ApLib
aP_depack:
   pushad

   mov    esi, [esp + 36]    ; C calling convention
   mov    edi, [esp + 40]

   cld
   mov    dl, 80h
   xor    ebx, ebx

literal:
   movsb
   mov    bl, 2
nexttag:
   call   getbit
   jnc    literal

   xor    ecx, ecx
   call   getbit
   jnc    codepair
   xor    eax, eax
   call   getbit
   jnc    shortmatch
   mov    bl, 2
   inc    ecx
   mov    al, 10h
getmorebits:
   call   getbit
   adc    al, al
   jnc    getmorebits
```

```
    jnz    domatch
    stosb
    jmp    short nexttag
codepair:
    call   getgamma_no_ecx
    sub    ecx, ebx
    jnz    normalcodepair
    call   getgamma
    jmp    short domatch_lastpos

shortmatch:
    lodsb
    shr    eax, 1
    jz     donedepacking
    adc    ecx, ecx
    jmp    short domatch_with_2inc

normalcodepair:
    xchg   eax, ecx
    dec    eax
    shl    eax, 8
    lodsb
    call   getgamma
    cmp    eax, 32000
    jae    domatch_with_2inc
    cmp    ah, 5
    jae    domatch_with_inc
    cmp    eax, 7fh
    ja     domatch_new_lastpos

domatch_with_2inc:
    inc    ecx

domatch_with_inc:
    inc    ecx

domatch_new_lastpos:
    xchg   eax, ebp
domatch_lastpos:
    mov    eax, ebp

    mov    bl, 1

domatch:
    push   esi
    mov    esi, edi
    sub    esi, eax
    rep    movsb
    pop    esi
    jmp    short nexttag

getbit:
    add    dl, dl
    jnz    stillbitsleft
    mov    dl, [esi]
    inc    esi
    adc    dl, dl
stillbitsleft:
    ret

getgamma:
```

```
    xor    ecx, ecx
getgamma_no_ecx:
    inc    ecx
getgammaloop:
    call   getbit
    adc    ecx, ecx
    call   getbit
    jc     getgammaloop
    ret

donedepacking:
    sub    edi, [esp + 40]
    mov    [esp + 28], edi    ; return unpacked length in eax

    popad
    ret

End Main
```

5. LES LAYERS

L'une des grandes difficultés de ce crackme est l'utilisation intensive et massive de couches de cryptage. A vue de nez, il doit bien y en avoir plus de 1000. Chaque couche de cryptage (appelée aussi layer) est décryptée par une routine complètement obfusquée. Ces routines sont de grande taille et sont constituées à 90 % de code inutile. Habituellement, les routines de décryptage de layers se terminent par un saut conditionnel (je, jne, …). Il suffit alors pour les franchir de poser un BPX juste après et de relancer l'exécution du programme. Ici, il n'en est rien !

De prime abord, les routines de décryptage sont difficiles à comprendre et à tracer. Il y a plusieurs facteurs qui rendent ces routines assez illisibles. La première raison est bien évidemment la taille de chaque routine qui avoisine les 400 lignes de code ! Le code a en plus subi des mutations féroces. Voici quelques exemples criants de ce genre de mutations :

Exemple 1 :

Voici le code d'origine avant la mutation :

```
sub byte [edi], 75h
```

Et le même code après la mutation :

```
dec esp
mov dl, byte ptr [edi]
mov byte ptr [esp], dl
mov al, 77h
dec al
dec al
sub byte ptr [esp], al
mov al, byte ptr [esp]
inc esp
mov byte ptr [edi], al
```

Exemple 2 :

Voici le code d'origine avant la mutation :

```
push edi
```

Et le même code après la mutation :

```
repne sub esp, 8
rep lea esp, [esp + 4]
repne mov dword ptr [esp], edi
```

Exemple 3 :

Voici le code d'origine avant la mutation :

```
pop edi
```

```
mov edi, dword ptr [esp]
add esp, 4
```

Et le même code après la mutation :

Bien que ces mutations soient déjà assez pénibles à tracer, l'auteur a ajouté du junk code qu'il a inséré aléatoirement dans les routines. La macro la plus facile à repérer est celle du « Double Call » vue également dans le packer SVKP (Slovak Protector) :

```
Junk1 MACRO
BYTE 0E8h,01h,00,00,00          call $+1
BYTE 0E8h
BYTE 0E8h,02h,00,00,00          call $+2
BYTE 0CDh,20h
BYTE 083h,04h,24h,0Bh           add dword ptr [esp], 0Bh
BYTE 083h,44h,24h,04h,13h       add dword ptr [esp+4], 13h
BYTE 0C3h                       ret
BYTE 0E9h
ENDM
```

On peut voir en plus de tout ça des opérations arithmétiques aléatoires effectuées sur les registres (sub - add - sbb - not - xor - lea - mov). Cependant, les registres modifiés sont préservés. L'auteur a évité d'utiliser le couple pushad/popad qui représente une faille trop évidente. Voici un exemple de structure que l'on peut voir dans une routine :

```
push edi
        opérations arithmétiques sur edi
push eax
        opérations arithmétiques sur edi et eax
pop eax
        opérations arithmétiques sur edi
pop edi
```

La combinaison de ces 2 techniques (junk code + mutations) donne un résultat impeccable. En voici maintenant une démonstration par l'exemple.

PHASE 1 - Voilà tout d'abord une routine de décryptage d'un layer sans mutations de codes et sans junk code ajouté :

```
---------lea edi, dword ptr [ebp + 461C71h]   Origine du layer
-------- lea ecx, dword ptr [ebp + 462339h]   fin du layer
-------- sub ecx, edi                         calcul de la taille du layer
419A53 sub byte ptr [edi], 75h                Décryptage octet par octet
-------- inc edi
-------- loopd 419A53                         decrémente ecx et boucle en 419A53 si ecx > 0
-------- jmp 461C71                           sauté sur le layer décrypté
```

PHASE 2 - Voici la même routine où l'on a remplacé le LOOPD par un JMP ESI astucieux :

```
-------- lea eax, [ebp + 461C71h]
```

```
-------- push eax                          pousser l'adresse du layer
-------- lea edi, [ebp + 461C71h]          Origine du layer
-------- lea ecx, [ebp + 462339h]          fin du layer
-------- sub ecx, edi                      calcul de la taille du layer
--------  lea ebx, [ebp + 419A53h]
-------- push ebx                          pousser l'adresse de la boucle de décryptage
419A53 sub byte ptr [edi], 75h             Décryptage octet par octet
--------  inc edi
--------  dec ecx                          decrémente ecx
--------  xor ebx, ebx
-------- cmp ecx, 0                        tester la valeur de ecx
--------  setz bl                          si ecx = 0, passer bl à 1 sinon le laisser à 0
--------  lea edx, [esp + ebx*4]
--------  mov esi, dword ptr [edx]
--------  jmp esi
```

Cette routine crée une table constituée de 2 adresses poussées sur la pile. Au moment où ecx == 0, setz arme bl à 1 et esi prend la valeur 461C71h.

PHASE 3 – Voici la même routine avec l'utilisation de mutations :

```
--------  lea eax, [ebp + 6A9A01B2h]
--------  repne sub esp, 8                 ┐
--------  rep lea esp, [esp + 4]           │   pousser l'adresse du layer
--------  repne mov dword ptr [esp], eax   ┘
--------  lea edi, [ebp + 461C71h]         Origine du layer
--------  lea ecx, [ebp + 462339h]         fin du layer
--------  sub ecx, edi                     calcul de la taille du layer
--------  lea ebx, [ebp + 6A957F77h]
--------  repne sub esp, 8                 ┐
--------  rep lea esp, [esp + 4]           │   pousser l'adresse de la boucle de décryptage
--------  repne mov dword ptr [esp], ebx   ┘

419A53 dec esp                             ┐
--------  mov dl, byte ptr [edi]           │
--------  mov byte ptr [esp], dl           │
--------  mov al, 77h                      │
--------  dec al                           │
--------  dec al                           │   sub byte ptr [edi], 75h
--------  sub byte ptr [esp], al           │
--------  mov al, byte ptr [esp]           │
--------  inc esp                          │
--------  mov byte ptr [edi], al           │
--------  lea edi, [edi+1]                 ┘
--------  lea ecx, [ecx-2]                 ┐   decrémente ecx
--------  inc ecx                          ┘
--------  xor ebx, ebx
--------  xor eax, eax
--------  repne sub esp, 8                 ┐
--------  rep lea esp, [esp + 4]           │   tester la valeur de ecx
--------  repne mov dword ptr [esp], ecx   │
--------  mov edx, dword ptr [esp]         ┘
```

```
-------- setz bl                    si ecx = 0, passer bl à 1 sinon le laisser à 0
-------- lea edx, [esp + ebx*4]
-------- mov esi, dword ptr [edx]
-------- jmp esi
```

PHASE 4 – Enfin, pour finir, voilà le listing complet de la routine de décryptage décortiquée par IDA. J'ai surligné en jaune le code important. Remarquez que le code de la routine est littéralement noyé dans du junk code !

```
.kaine0:00419861 ; ---------------------------------------------------------------------------
.kaine0:00419861                 add     esp, 4
.kaine0:00419864                 call    loc_41986A
.kaine0:00419864 ; ---------------------------------------------------------------------------
.kaine0:00419869                 db 0E8h ; Þ
.kaine0:0041986A ; ---------------------------------------------------------------------------
.kaine0:0041986A
.kaine0:0041986A loc_41986A:                             ; CODE XREF: .kaine0:00419864↑p
.kaine0:0041986A                 call    loc_419871
.kaine0:0041986A ; ---------------------------------------------------------------------------
.kaine0:0041986F                 db 0CDh
.kaine0:00419870                 db  20h
.kaine0:00419871 ; ---------------------------------------------------------------------------
.kaine0:00419871
.kaine0:00419871 loc_419871:                             ; CODE XREF: .kaine0:loc_41986A↑p
.kaine0:00419871                 add     dword ptr [esp], 0Bh
.kaine0:00419875                 add     dword ptr [esp+4], 13h
.kaine0:0041987A                 retn
.kaine0:0041987A ; ---------------------------------------------------------------------------
.kaine0:0041987B                 db 0E9h ; Ú
.kaine0:0041987C ; ---------------------------------------------------------------------------
.kaine0:0041987C                 repne sub esp, 8
.kaine0:00419880                 rep lea esp, [esp+4]
.kaine0:00419885                 repne mov [esp], edi
.kaine0:00419889                 call    loc_41988F
.kaine0:00419889 ; ---------------------------------------------------------------------------
.kaine0:0041988E                 db 0E8h
.kaine0:0041988F ; ---------------------------------------------------------------------------
.kaine0:0041988F
.kaine0:0041988F loc_41988F:                             ; CODE XREF: .kaine0:00419889↑p
.kaine0:0041988F                 call    loc_419896
.kaine0:0041988F ; ---------------------------------------------------------------------------
.kaine0:00419894                 db 0CDh ; -
.kaine0:00419895                 db 20h
.kaine0:00419896 ; ---------------------------------------------------------------------------
.kaine0:00419896
.kaine0:00419896 loc_419896:                             ; CODE XREF: .kaine0:loc_41988F↑p
.kaine0:00419896                 add     dword ptr [esp], 0Bh
.kaine0:0041989A                 add     dword ptr [esp+4], 13h
.kaine0:0041989F                 retn
.kaine0:0041989F ; ---------------------------------------------------------------------------
.kaine0:004198A0                 db 0E9h
.kaine0:004198A1 ; ---------------------------------------------------------------------------
.kaine0:004198A1                 lea     edi, [ecx+edi]
.kaine0:004198A4                 rep add edi, edx
.kaine0:004198A7                 sbb     edi, esi
.kaine0:004198A9                 db      0F3h
.kaine0:004198A9                 repne sub esp, 8
.kaine0:004198AE                 rep lea esp, [esp+4]
.kaine0:004198B3                 repne mov [esp], eax
.kaine0:004198B7                 sbb     edi, ebx
.kaine0:004198B9                 sub     eax, offset dword_4350EF
.kaine0:004198BE                 jnb     short loc_4198C3
.kaine0:004198C0                 jb      short loc_4198C3
.kaine0:004198C0 ; ---------------------------------------------------------------------------
.kaine0:004198C2                 db 0E9h
.kaine0:004198C3 ; ---------------------------------------------------------------------------
.kaine0:004198C3
.kaine0:004198C3 loc_4198C3:                             ; CODE XREF: .kaine0:004198BE↑j
.kaine0:004198C3                                         ; .kaine0:004198C0↑j
.kaine0:004198C3                 lea     eax, [eax+ebp]
.kaine0:004198C6                 rep lea edi, [edi]
.kaine0:004198C9                 repne add edi, ebp
.kaine0:004198CC                 rep add eax, eax
.kaine0:004198CF                 repne sub esp, 8
.kaine0:004198D3                 rep lea esp, [esp+4]
.kaine0:004198D8                 repne mov [esp], ecx
.kaine0:004198DC                 add     ecx, edi
.kaine0:004198DE                 rep sub ecx, eax
.kaine0:004198E1                 rep xor ecx, edx
.kaine0:004198E4                 not     ecx
.kaine0:004198E6                 repne lea eax, [edi+eax]
.kaine0:004198EA                 rep xor eax, ecx
.kaine0:004198ED                 repne sub eax, ebx
.kaine0:004198F0                 rep lea ecx, [ebx]
.kaine0:004198F3                 call    loc_4198F9
.kaine0:004198F3 ; ---------------------------------------------------------------------------
.kaine0:004198F8                 db 0E8h
.kaine0:004198F9 ; ---------------------------------------------------------------------------
.kaine0:004198F9
.kaine0:004198F9 loc_4198F9:                             ; CODE XREF: .kaine0:004198F3↑p
.kaine0:004198F9                 call    loc_419900
.kaine0:004198F9 ; ---------------------------------------------------------------------------
.kaine0:004198FE                 db 0CDh
.kaine0:004198FF                 db  20h
.kaine0:00419900 ; ---------------------------------------------------------------------------
.kaine0:00419900
.kaine0:00419900 loc_419900:                             ; CODE XREF: .kaine0:loc_4198F9↑p
.kaine0:00419900                 add     dword ptr [esp], 0Bh
.kaine0:00419904                 add     dword ptr [esp+4], 13h
.kaine0:00419909                 retn
.kaine0:00419909 ; ---------------------------------------------------------------------------
.kaine0:0041990A                 db 0E9h
.kaine0:0041990B ; ---------------------------------------------------------------------------
.kaine0:0041990B                 lea     ecx, [edx+eax]
```

```
.kaine0:0041990E                rep add ecx, ecx
.kaine0:00419911                mov     ecx, offset dword_406052
.kaine0:00419916                repne sar eax, 0
.kaine0:0041991A                jmp     short loc_41991D
.kaine0:0041991A ; ---------------------------------------------------------------------------
.kaine0:0041991C                db 0E9h ; Ú
.kaine0:0041991D ; ---------------------------------------------------------------------------
.kaine0:0041991D
.kaine0:0041991D loc_41991D:                            ; CODE XREF: .kaine0:0041991A↑j
.kaine0:0041991D                mov     ecx, [esp]
.kaine0:00419920                rep call loc_419927
.kaine0:00419920 ; ---------------------------------------------------------------------------
.kaine0:00419926                db 0E8h
.kaine0:00419927 ; ---------------------------------------------------------------------------
.kaine0:00419927
.kaine0:00419927 loc_419927:                            ; CODE XREF: .kaine0:00419920↑p
.kaine0:00419927                call    loc_41992E
.kaine0:00419927 ; ---------------------------------------------------------------------------
.kaine0:0041992C                db 0CDh ; -
.kaine0:0041992D                db 20h
.kaine0:0041992E ; ---------------------------------------------------------------------------
.kaine0:0041992E
.kaine0:0041992E loc_41992E:                            ; CODE XREF: .kaine0:loc_419927↑p
.kaine0:0041992E                add     dword ptr [esp], 0Bh
.kaine0:00419932                add     dword ptr [esp+4], 13h
.kaine0:00419937                retn
.kaine0:00419937 ; ---------------------------------------------------------------------------
.kaine0:00419938                db 0E9h
.kaine0:00419939 ; ---------------------------------------------------------------------------
.kaine0:00419939                add     esp, 4
.kaine0:0041993C                repne sub edi, eax
.kaine0:0041993F                sbb     edi, ebx
.kaine0:00419941                repne add eax, edx
.kaine0:00419944                rep sub eax, ebp
.kaine0:00419947                repne lea eax, [ebx+eax]
.kaine0:0041994B                sub     eax, edx
.kaine0:0041994D                repne sub eax, eax
.kaine0:00419950                rep lea eax, [edx+eax]
.kaine0:00419954                rep add eax, ecx
.kaine0:00419957                mov     eax, offset dword_406052
.kaine0:0041995C                repne sar eax, 0
.kaine0:00419960                jmp     short loc_419963
.kaine0:00419960 ; ---------------------------------------------------------------------------
.kaine0:00419962                db 0E9h ; Ú
.kaine0:00419963 ; ---------------------------------------------------------------------------
.kaine0:00419963
.kaine0:00419963 loc_419963:                            ; CODE XREF: .kaine0:00419960↑j
.kaine0:00419963                mov     eax, [esp]
.kaine0:00419966                rep call loc_41996D
.kaine0:00419966 ; ---------------------------------------------------------------------------
.kaine0:0041996C                db 0E8h
.kaine0:0041996D ; ---------------------------------------------------------------------------
.kaine0:0041996D
.kaine0:0041996D loc_41996D:                            ; CODE XREF: .kaine0:00419966↑p
.kaine0:0041996D                call    loc_419974
.kaine0:0041996D ; ---------------------------------------------------------------------------
.kaine0:00419972                db 0CDh ; -
.kaine0:00419973                db 20h
.kaine0:00419974 ; ---------------------------------------------------------------------------
.kaine0:00419974
.kaine0:00419974 loc_419974:                            ; CODE XREF: .kaine0:loc_41996D↑p
.kaine0:00419974                add     dword ptr [esp], 0Bh
.kaine0:00419978                add     dword ptr [esp+4], 13h
.kaine0:0041997D                retn
.kaine0:0041997D ; ---------------------------------------------------------------------------
.kaine0:0041997E                db 0E9h
.kaine0:0041997F ; ---------------------------------------------------------------------------
.kaine0:0041997F                add     esp, 4
.kaine0:00419982                add     edi, ebx
.kaine0:00419984                repne sbb edi, eax
.kaine0:00419987                rep add edi, ebp
.kaine0:0041998A                call    loc_419990
.kaine0:0041998A ; ---------------------------------------------------------------------------
.kaine0:0041998F                db 0E8h
.kaine0:00419990 ; ---------------------------------------------------------------------------
.kaine0:00419990
.kaine0:00419990 loc_419990:                            ; CODE XREF: .kaine0:0041998A↑p
.kaine0:00419990                call    loc_419997
.kaine0:00419990 ; ---------------------------------------------------------------------------
.kaine0:00419995                db 0CDh
.kaine0:00419996                db  20h
.kaine0:00419997 ; ---------------------------------------------------------------------------
.kaine0:00419997
.kaine0:00419997 loc_419997:                            ; CODE XREF: .kaine0:loc_419990↑p
.kaine0:00419997                add     dword ptr [esp], 0Bh
.kaine0:0041999B                add     dword ptr [esp+4], 13h
.kaine0:004199A0                retn
.kaine0:004199A0 ; ---------------------------------------------------------------------------
.kaine0:004199A1                db 0E9h
.kaine0:004199A2 ; ---------------------------------------------------------------------------
.kaine0:004199A2                add     edi, [esp+34h]
.kaine0:004199A6                lea     edi, [edx+eax]
.kaine0:004199A9                rep add edi, ecx
.kaine0:004199AC                mov     edi, offset dword_406052
.kaine0:004199B1                repne sar eax, 0
.kaine0:004199B5                jmp     short loc_4199B8
.kaine0:004199B5 ; ---------------------------------------------------------------------------
.kaine0:004199B7                db 0E9h ; Ú
.kaine0:004199B8 ; ---------------------------------------------------------------------------
.kaine0:004199B8
.kaine0:004199B8 loc_4199B8:                            ; CODE XREF: .kaine0:004199B5↑j
.kaine0:004199B8                mov     edi, [esp]
.kaine0:004199BB                rep call loc_4199C2
.kaine0:004199BB ; ---------------------------------------------------------------------------
.kaine0:004199C1                db 0E8h
.kaine0:004199C2 ; ---------------------------------------------------------------------------
.kaine0:004199C2
.kaine0:004199C2 loc_4199C2:                            ; CODE XREF: .kaine0:004199BB↑p
.kaine0:004199C2                call    loc_4199C9
.kaine0:004199C2 ; ---------------------------------------------------------------------------
.kaine0:004199C7                db 0CDh ; -
.kaine0:004199C8                db 20h
.kaine0:004199C9 ; ---------------------------------------------------------------------------
.kaine0:004199C9
```

```
.kaine0:004199C9 loc_4199C9:                             ; CODE
XREF: .kaine0:loc_4199C2↑p
.kaine0:004199C9                add     dword ptr [esp], 0Bh
.kaine0:004199CD                add     dword ptr [esp+4], 13h
.kaine0:004199D2                retn
.kaine0:004199D2 ; ---------------------------------------------------------------------------
.kaine0:004199D3                db 0E9h
.kaine0:004199D4 ; ---------------------------------------------------------------------------
.kaine0:004199D4                add     esp, 4
.kaine0:004199D7                lea     eax, [ebp+6A9A01B2h]
.kaine0:004199DD                repne sub esp, 8
.kaine0:004199E1                rep lea esp, [esp+4]
.kaine0:004199E6                repne mov [esp], eax
.kaine0:004199EA                lea     edi, [ebp+461C71h]
.kaine0:004199F0                call    loc_4199F6
.kaine0:004199F0 ; ---------------------------------------------------------------------------
.kaine0:004199F5                db 0E8h
.kaine0:004199F6 ; ---------------------------------------------------------------------------
.kaine0:004199F6
.kaine0:004199F6 loc_4199F6:                             ; CODE
XREF: .kaine0:004199F0↑p
.kaine0:004199F6                call    loc_4199FD
.kaine0:004199F6 ; ---------------------------------------------------------------------------
.kaine0:004199FB                db 0CDh ; -
.kaine0:004199FC                db 20h
.kaine0:004199FD ; ---------------------------------------------------------------------------
.kaine0:004199FD
.kaine0:004199FD loc_4199FD:                             ; CODE
XREF: .kaine0:loc_4199F6↑p
.kaine0:004199FD                add     dword ptr [esp], 0Bh
.kaine0:00419A01                add     dword ptr [esp+4], 13h
.kaine0:00419A06                retn
.kaine0:00419A06 ; ---------------------------------------------------------------------------
.kaine0:00419A07                db 0E9h
.kaine0:00419A08 ; ---------------------------------------------------------------------------
.kaine0:00419A08                lea     ecx, [ebp+462339h]
.kaine0:00419A0E                call    loc_419A14
.kaine0:00419A0E ; ---------------------------------------------------------------------------
.kaine0:00419A13                db 0E8h
.kaine0:00419A14 ; ---------------------------------------------------------------------------
.kaine0:00419A14
.kaine0:00419A14 loc_419A14:                             ; CODE
XREF: .kaine0:00419A0E↑p
.kaine0:00419A14                call    loc_419A1B
.kaine0:00419A14 ; ---------------------------------------------------------------------------
.kaine0:00419A19                db 0CDh ; -
.kaine0:00419A1A                db 20h
.kaine0:00419A1B ; ---------------------------------------------------------------------------
.kaine0:00419A1B
.kaine0:00419A1B loc_419A1B:                             ; CODE
XREF: .kaine0:loc_419A14↑p
.kaine0:00419A1B                add     dword ptr [esp], 0Bh
.kaine0:00419A1F                add     dword ptr [esp+4], 13h
.kaine0:00419A24                retn
.kaine0:00419A24 ; ---------------------------------------------------------------------------
.kaine0:00419A25                db 0E9h
.kaine0:00419A26 ; ---------------------------------------------------------------------------
.kaine0:00419A26                sub     ecx, edi
.kaine0:00419A28                call    loc_419A2E
.kaine0:00419A28 ; ---------------------------------------------------------------------------
.kaine0:00419A2D                db 0E8h
.kaine0:00419A2E ; ---------------------------------------------------------------------------
.kaine0:00419A2E
.kaine0:00419A2E loc_419A2E:                             ; CODE
XREF: .kaine0:00419A28↑p
.kaine0:00419A2E                call    loc_419A35
.kaine0:00419A2E ; ---------------------------------------------------------------------------
.kaine0:00419A33                db 0CDh ; -
.kaine0:00419A34                db 20h
.kaine0:00419A35 ; ---------------------------------------------------------------------------
.kaine0:00419A35
.kaine0:00419A35 loc_419A35:                             ; CODE
XREF: .kaine0:loc_419A2E↑p
.kaine0:00419A35                add     dword ptr [esp], 0Bh
.kaine0:00419A39                add     dword ptr [esp+4], 13h
.kaine0:00419A3E                retn
.kaine0:00419A3E ; ---------------------------------------------------------------------------
.kaine0:00419A3F                db 0E9h
.kaine0:00419A40 ; ---------------------------------------------------------------------------
.kaine0:00419A40                lea     ebx, [ebp+6A957F77h]
.kaine0:00419A46                repne sub esp, 8
.kaine0:00419A4A                rep lea esp, [esp+4]
.kaine0:00419A4F                repne mov [esp], ebx
.kaine0:00419A53                repne sub esp, 8
.kaine0:00419A57                rep lea esp, [esp+4]
.kaine0:00419A5C                repne mov [esp], ebx
.kaine0:00419A60                mov     ebx, eax
.kaine0:00419A62                repne sub ebx, edx
.kaine0:00419A65                db      0F3h
.kaine0:00419A65                repne sub esp, 8
.kaine0:00419A6A                rep lea esp, [esp+4]
.kaine0:00419A6F                repne mov [esp], edi
.kaine0:00419A73                sub     edi, edx
.kaine0:00419A75                repne lea edi, [ecx-1]
.kaine0:00419A79                call    loc_419A7F
.kaine0:00419A79 ; ---------------------------------------------------------------------------
.kaine0:00419A7E                db 0E8h
.kaine0:00419A7F ; ---------------------------------------------------------------------------
.kaine0:00419A7F
.kaine0:00419A7F loc_419A7F:                             ; CODE
XREF: .kaine0:00419A79↑p
.kaine0:00419A7F                call    loc_419A86
.kaine0:00419A7F ; ---------------------------------------------------------------------------
.kaine0:00419A84                db 0CDh ; -
.kaine0:00419A85                db 20h
.kaine0:00419A86 ; ---------------------------------------------------------------------------
.kaine0:00419A86
.kaine0:00419A86 loc_419A86:                             ; CODE
XREF: .kaine0:loc_419A7F↑p
.kaine0:00419A86                add     dword ptr [esp], 0Bh
.kaine0:00419A8A                add     dword ptr [esp+4], 13h
.kaine0:00419A8F                retn
.kaine0:00419A8F ; ---------------------------------------------------------------------------
.kaine0:00419A90                db 0E9h
.kaine0:00419A91 ; ---------------------------------------------------------------------------
.kaine0:00419A91                lea     ebx, [eax+1]
.kaine0:00419A94                repne sub esp, 8
.kaine0:00419A98                rep lea esp, [esp+4]
.kaine0:00419A9D                repne mov [esp], edx
.kaine0:00419AA1                sbb     edx, edx
.kaine0:00419AA3                repne lea edx, [eax+443A2Dh]
.kaine0:00419AAA                lea     edx, [edx+eax]
.kaine0:00419AAD                rep add edx, ecx
.kaine0:00419AB0                mov     edx, offset dword_406052
.kaine0:00419AB5                repne sar eax, 0
.kaine0:00419AB9                jmp     short loc_419ABC
```

```
.kaine0:00419AB9 ; ---------------------------------------------------------------------------
.kaine0:00419ABB                 db 0E9h ; Ú
.kaine0:00419ABC ; ---------------------------------------------------------------------------
.kaine0:00419ABC
.kaine0:00419ABC loc_419ABC:                             ; CODE XREF: .kaine0:00419AB9↑j
.kaine0:00419ABC                 mov     edx, [esp]
.kaine0:00419ABF                 rep call loc_419AC6
.kaine0:00419ABF ; ---------------------------------------------------------------------------
.kaine0:00419AC5                 db 0E8h
.kaine0:00419AC6 ; ---------------------------------------------------------------------------
.kaine0:00419AC6
.kaine0:00419AC6 loc_419AC6:                             ; CODE XREF: .kaine0:00419ABF↑p
.kaine0:00419AC6                 call    loc_419ACD
.kaine0:00419AC6 ; ---------------------------------------------------------------------------
.kaine0:00419ACB                 db 0CDh
.kaine0:00419ACC                 db  20h
.kaine0:00419ACD ; ---------------------------------------------------------------------------
.kaine0:00419ACD
.kaine0:00419ACD loc_419ACD:                             ; CODE XREF: .kaine0:loc_419AC6↑p
.kaine0:00419ACD                 add     dword ptr [esp], 0Bh
.kaine0:00419AD1                 add     dword ptr [esp+4], 13h
.kaine0:00419AD6                 retn
.kaine0:00419AD6 ; ---------------------------------------------------------------------------
.kaine0:00419AD7                 db 0E9h
.kaine0:00419AD8 ; ---------------------------------------------------------------------------
.kaine0:00419AD8                 add     esp, 4
.kaine0:00419ADB                 db      0F3h
.kaine0:00419ADB                 repne sub esp, 8
.kaine0:00419AE0                 rep lea esp, [esp+4]
.kaine0:00419AE5                 repne mov [esp], eax
.kaine0:00419AE9                 lea     edi, [edi+ecx]
.kaine0:00419AEC                 repne lea ebx, [ebp+453156h]
.kaine0:00419AF3                 sbb     eax, edx
.kaine0:00419AF5                 rep lea eax, [edx+eax]
.kaine0:00419AF9                 rep add eax, ecx
.kaine0:00419AFC                 mov     eax, offset dword_406052
.kaine0:00419B01                 repne sar eax, 0
.kaine0:00419B05                 jmp     short loc_419B08
.kaine0:00419B05 ; ---------------------------------------------------------------------------
.kaine0:00419B07                 db 0E9h ; Ú
.kaine0:00419B08 ; ---------------------------------------------------------------------------
.kaine0:00419B08
.kaine0:00419B08 loc_419B08:                             ; CODE XREF: .kaine0:00419B05↑j
.kaine0:00419B08                 mov     eax, [esp]
.kaine0:00419B0B                 rep call loc_419B12
.kaine0:00419B0B ; ---------------------------------------------------------------------------
.kaine0:00419B11                 db 0E8h
.kaine0:00419B12 ; ---------------------------------------------------------------------------
.kaine0:00419B12
.kaine0:00419B12 loc_419B12:                             ; CODE XREF: .kaine0:00419B0B↑p
.kaine0:00419B12                 call    loc_419B19
.kaine0:00419B12 ; ---------------------------------------------------------------------------
.kaine0:00419B17                 db 0CDh ; -
.kaine0:00419B18                 db 20h
.kaine0:00419B19 ; ---------------------------------------------------------------------------
.kaine0:00419B19
.kaine0:00419B19 loc_419B19:                             ; CODE XREF: .kaine0:loc_419B12↑p
.kaine0:00419B19                 add     dword ptr [esp], 0Bh
.kaine0:00419B1D                 add     dword ptr [esp+4], 13h
.kaine0:00419B22                 retn
.kaine0:00419B22 ; ---------------------------------------------------------------------------
.kaine0:00419B23                 db 0E9h
.kaine0:00419B24 ; ---------------------------------------------------------------------------
.kaine0:00419B24                 add     esp, 4
.kaine0:00419B27                 repne lea ebx, [eax+ecx]
.kaine0:00419B2B                 mov     edi, 40822Ah
.kaine0:00419B30                 rep sbb edi, edx
.kaine0:00419B33                 lea     edi, [edx+eax]
.kaine0:00419B36                 rep add edi, ecx
.kaine0:00419B39                 mov     edi, offset dword_406052
.kaine0:00419B3E                 repne sar eax, 0
.kaine0:00419B42                 jmp     short loc_419B45
.kaine0:00419B42 ; ---------------------------------------------------------------------------
.kaine0:00419B44                 db 0E9h ; Ú
.kaine0:00419B45 ; ---------------------------------------------------------------------------
.kaine0:00419B45
.kaine0:00419B45 loc_419B45:                             ; CODE XREF: .kaine0:00419B42↑j
.kaine0:00419B45                 mov     edi, [esp]
.kaine0:00419B48                 rep call loc_419B4F
.kaine0:00419B48 ; ---------------------------------------------------------------------------
.kaine0:00419B4E                 db 0E8h
.kaine0:00419B4F ; ---------------------------------------------------------------------------
.kaine0:00419B4F
.kaine0:00419B4F loc_419B4F:                             ; CODE XREF: .kaine0:00419B48↑p
.kaine0:00419B4F                 call    loc_419B56
.kaine0:00419B4F ; ---------------------------------------------------------------------------
.kaine0:00419B54                 db 0CDh ; -
.kaine0:00419B55                 db 20h
.kaine0:00419B56 ; ---------------------------------------------------------------------------
.kaine0:00419B56
.kaine0:00419B56 loc_419B56:                             ; CODE XREF: .kaine0:loc_419B4F↑p
.kaine0:00419B56                 add     dword ptr [esp], 0Bh
.kaine0:00419B5A                 add     dword ptr [esp+4], 13h
.kaine0:00419B5F                 retn
.kaine0:00419B5F ; ---------------------------------------------------------------------------
.kaine0:00419B60                 db 0E9h
.kaine0:00419B61 ; ---------------------------------------------------------------------------
.kaine0:00419B61                 add     esp, 4
.kaine0:00419B64                 rep add ebx, eax
.kaine0:00419B67                 sub     ebx, ecx
.kaine0:00419B69                 lea     ebx, [edx+eax]
.kaine0:00419B6C                 rep add ebx, ecx
.kaine0:00419B6F                 mov     ebx, offset dword_406052
.kaine0:00419B74                 repne sar eax, 0
.kaine0:00419B78                 jmp     short loc_419B7B
.kaine0:00419B78 ; ---------------------------------------------------------------------------
.kaine0:00419B7A                 db 0E9h ; Ú
.kaine0:00419B7B ; ---------------------------------------------------------------------------
.kaine0:00419B7B
.kaine0:00419B7B loc_419B7B:                             ; CODE XREF: .kaine0:00419B78↑j
.kaine0:00419B7B                 mov     ebx, [esp]
.kaine0:00419B7E                 rep call loc_419B85
.kaine0:00419B7E ; ---------------------------------------------------------------------------
.kaine0:00419B84                 db 0E8h
.kaine0:00419B85 ; ---------------------------------------------------------------------------
.kaine0:00419B85
```

```
.kaine0:00419B85 loc_419B85:                     ; CODE
XREF: .kaine0:00419B7E↑p
.kaine0:00419B85              call    loc_419B8C
.kaine0:00419B85 ; ---------------------------------------------------------------------
.kaine0:00419B8A              db 0CDh ; -
.kaine0:00419B8B              db 20h
.kaine0:00419B8C ; ---------------------------------------------------------------------
.kaine0:00419B8C
.kaine0:00419B8C loc_419B8C:                     ; CODE
XREF: .kaine0:loc_419B85↑p
.kaine0:00419B8C              add     dword ptr [esp], 0Bh
.kaine0:00419B90              add     dword ptr [esp+4], 13h
.kaine0:00419B95              retn
.kaine0:00419B95 ; ---------------------------------------------------------------------
.kaine0:00419B96              db 0E9h
.kaine0:00419B97 ; ---------------------------------------------------------------------
.kaine0:00419B97              add     esp, 4
.kaine0:00419B9A              dec     esp
.kaine0:00419B9B              xor     edx, edx
.kaine0:00419B9D              repne jnz short loc_419BA3
.kaine0:00419BA0              jz      short loc_419BA3
.kaine0:00419BA0 ; ---------------------------------------------------------------------
.kaine0:00419BA2              db 0E8h
.kaine0:00419BA3 ; ---------------------------------------------------------------------
.kaine0:00419BA3
.kaine0:00419BA3 loc_419BA3:                     ; CODE
XREF: .kaine0:00419B9D↑j
.kaine0:00419BA3                                 ; .kaine0:00419BA0↑j
.kaine0:00419BA3              call    loc_419BA9
.kaine0:00419BA3 ; ---------------------------------------------------------------------
.kaine0:00419BA8              db 0E8h
.kaine0:00419BA9 ; ---------------------------------------------------------------------
.kaine0:00419BA9
.kaine0:00419BA9 loc_419BA9:                     ; CODE
XREF: .kaine0:loc_419BA3↑p
.kaine0:00419BA9              call    loc_419BB0
.kaine0:00419BA9 ; ---------------------------------------------------------------------
.kaine0:00419BAE              db 0CDh
.kaine0:00419BAF              db  20h
.kaine0:00419BB0 ; ---------------------------------------------------------------------
.kaine0:00419BB0
.kaine0:00419BB0 loc_419BB0:                     ; CODE
XREF: .kaine0:loc_419BA9↑p
.kaine0:00419BB0              add     dword ptr [esp], 0Bh
.kaine0:00419BB4              add     dword ptr [esp+4], 13h
.kaine0:00419BB9              retn
.kaine0:00419BB9 ; ---------------------------------------------------------------------
.kaine0:00419BBA              db 0E9h ; Ú
.kaine0:00419BBB ; ---------------------------------------------------------------------
.kaine0:00419BBB              mov     dl, [edi]
.kaine0:00419BBD              repne mov [esp], dl
.kaine0:00419BC1              mov     al, 77h
.kaine0:00419BC3              dec     al
.kaine0:00419BC5              dec     al
.kaine0:00419BC7              call    loc_419BCD
.kaine0:00419BC7 ; ---------------------------------------------------------------------
.kaine0:00419BCC              db 0E8h
.kaine0:00419BCD ; ---------------------------------------------------------------------
.kaine0:00419BCD
.kaine0:00419BCD loc_419BCD:                     ; CODE
XREF: .kaine0:00419BC7↑p
.kaine0:00419BCD              call    loc_419BD4
.kaine0:00419BCD ; ---------------------------------------------------------------------
.kaine0:00419BD2              db 0CDh ; -
.kaine0:00419BD3              db 20h
.kaine0:00419BD4 ; ---------------------------------------------------------------------
.kaine0:00419BD4
.kaine0:00419BD4 loc_419BD4:                     ; CODE
XREF: .kaine0:loc_419BCD↑p
.kaine0:00419BD4              add     dword ptr [esp], 0Bh
.kaine0:00419BD8              add     dword ptr [esp+4], 13h
.kaine0:00419BDD              retn
.kaine0:00419BDD ; ---------------------------------------------------------------------
.kaine0:00419BDE              db 0E9h
.kaine0:00419BDF ; ---------------------------------------------------------------------
.kaine0:00419BDF              sub     [esp], al
.kaine0:00419BE2              call    loc_419BE8
.kaine0:00419BE2 ; ---------------------------------------------------------------------
.kaine0:00419BE7              db 0E8h
.kaine0:00419BE8 ; ---------------------------------------------------------------------
.kaine0:00419BE8
.kaine0:00419BE8 loc_419BE8:                     ; CODE
XREF: .kaine0:00419BE2↑p
.kaine0:00419BE8              call    loc_419BEF
.kaine0:00419BE8 ; ---------------------------------------------------------------------
.kaine0:00419BED              db 0CDh ; -
.kaine0:00419BEE              db 20h
.kaine0:00419BEF ; ---------------------------------------------------------------------
.kaine0:00419BEF
.kaine0:00419BEF loc_419BEF:                     ; CODE
XREF: .kaine0:loc_419BE8↑p
.kaine0:00419BEF              add     dword ptr [esp], 0Bh
.kaine0:00419BF3              add     dword ptr [esp+4], 13h
.kaine0:00419BF8              retn
.kaine0:00419BF8 ; ---------------------------------------------------------------------
.kaine0:00419BF9              db 0E9h
.kaine0:00419BFA ; ---------------------------------------------------------------------
.kaine0:00419BFA              mov     al, [esp]
.kaine0:00419BFD              inc     esp
.kaine0:00419BFE              call    loc_419C04
.kaine0:00419BFE ; ---------------------------------------------------------------------
.kaine0:00419C03              db 0E8h
.kaine0:00419C04 ; ---------------------------------------------------------------------
.kaine0:00419C04
.kaine0:00419C04 loc_419C04:                     ; CODE
XREF: .kaine0:00419BFE↑p
.kaine0:00419C04              call    loc_419C0B
.kaine0:00419C04 ; ---------------------------------------------------------------------
.kaine0:00419C09              db 0CDh ; -
.kaine0:00419C0A              db 20h
.kaine0:00419C0B ; ---------------------------------------------------------------------
.kaine0:00419C0B
.kaine0:00419C0B loc_419C0B:                     ; CODE
XREF: .kaine0:loc_419C04↑p
.kaine0:00419C0B              add     dword ptr [esp], 0Bh
.kaine0:00419C0F              add     dword ptr [esp+4], 13h
.kaine0:00419C14              retn
.kaine0:00419C14 ; ---------------------------------------------------------------------
.kaine0:00419C15              db 0E9h
.kaine0:00419C16 ; ---------------------------------------------------------------------
.kaine0:00419C16              mov     [edi], al
.kaine0:00419C18              lea     edi, [edi+1]
.kaine0:00419C1B              lea     ecx, [ecx-2]
```

```
.kaine0:00419C1E                inc     ecx
.kaine0:00419C1F                repne sub esp, 8
.kaine0:00419C23                rep lea esp, [esp+4]
.kaine0:00419C28                repne mov [esp], ecx
.kaine0:00419C2C                jnz     short loc_419C31
.kaine0:00419C2E                jz      short loc_419C31
.kaine0:00419C2E ; ---------------------------------------------------------------------------
.kaine0:00419C30                db 0E8h
.kaine0:00419C31 ; ---------------------------------------------------------------------------
.kaine0:00419C31
.kaine0:00419C31 loc_419C31:                            ; CODE XREF: .kaine0:00419C2C↑j
.kaine0:00419C31                                        ; .kaine0:00419C2E↑j
.kaine0:00419C31                sub     edx, 4050A3h
.kaine0:00419C37                lea     edx, [edi+eax]
.kaine0:00419C3A                jnb     short loc_419C3F
.kaine0:00419C3C                jb      short loc_419C3F
.kaine0:00419C3C ; ---------------------------------------------------------------------------
.kaine0:00419C3E                db 0E9h
.kaine0:00419C3F ; ---------------------------------------------------------------------------
.kaine0:00419C3F
.kaine0:00419C3F loc_419C3F:                            ; CODE XREF: .kaine0:00419C3A↑j
.kaine0:00419C3F                                        ; .kaine0:00419C3C↑j
.kaine0:00419C3F                add     edx, ebx
.kaine0:00419C41                sub     edx, edi
.kaine0:00419C43                rep lea edx, [esi+edx]
.kaine0:00419C47                add     edx, ecx
.kaine0:00419C49                repne sbb edx, ebp
.kaine0:00419C4C                call    loc_419C52
.kaine0:00419C4C ; ---------------------------------------------------------------------------
.kaine0:00419C51                db 0E8h
.kaine0:00419C52 ; ---------------------------------------------------------------------------
.kaine0:00419C52
.kaine0:00419C52 loc_419C52:                            ; CODE XREF: .kaine0:00419C4C↑p
.kaine0:00419C52                call    loc_419C59
.kaine0:00419C52 ; ---------------------------------------------------------------------------
.kaine0:00419C57                db 0CDh ; -
.kaine0:00419C58                db  20h
.kaine0:00419C59 ; ---------------------------------------------------------------------------
.kaine0:00419C59
.kaine0:00419C59 loc_419C59:                            ; CODE XREF: .kaine0:loc_419C52↑p
.kaine0:00419C59                add     dword ptr [esp], 0Bh
.kaine0:00419C5D                add     dword ptr [esp+4], 13h
.kaine0:00419C62                retn
.kaine0:00419C62 ; ---------------------------------------------------------------------------
.kaine0:00419C63                db 0E9h ; Ú
.kaine0:00419C64 ; ---------------------------------------------------------------------------
.kaine0:00419C64                lea     edx, [ecx+ebp]
.kaine0:00419C67                lea     edx, [esp+eax]
.kaine0:00419C6A                lea     edx, [edx+eax]
.kaine0:00419C6D                rep add edx, ecx
.kaine0:00419C70                mov     edx, offset dword_406052
.kaine0:00419C75                repne sar eax, 0
.kaine0:00419C79                jmp     short loc_419C7C
.kaine0:00419C79 ; ---------------------------------------------------------------------------
.kaine0:00419C7B                db 0E9h ; Ú
.kaine0:00419C7C ; ---------------------------------------------------------------------------
.kaine0:00419C7C
.kaine0:00419C7C loc_419C7C:                            ; CODE XREF: .kaine0:00419C79↑j
.kaine0:00419C7C                mov     edx, [esp]
.kaine0:00419C7F                rep call loc_419C86
.kaine0:00419C7F ; ---------------------------------------------------------------------------
.kaine0:00419C85                db 0E8h
.kaine0:00419C86 ; ---------------------------------------------------------------------------
.kaine0:00419C86
.kaine0:00419C86 loc_419C86:                            ; CODE XREF: .kaine0:00419C7F↑p
.kaine0:00419C86                call    loc_419C8D
.kaine0:00419C86 ; ---------------------------------------------------------------------------
.kaine0:00419C8B                db 0CDh ; -
.kaine0:00419C8C                db 20h
.kaine0:00419C8D ; ---------------------------------------------------------------------------
.kaine0:00419C8D
.kaine0:00419C8D loc_419C8D:                            ; CODE XREF: .kaine0:loc_419C86↑p
.kaine0:00419C8D                add     dword ptr [esp], 0Bh
.kaine0:00419C91                add     dword ptr [esp+4], 13h
.kaine0:00419C96                retn
.kaine0:00419C96 ; ---------------------------------------------------------------------------
.kaine0:00419C97                db 0E9h
.kaine0:00419C98 ; ---------------------------------------------------------------------------
.kaine0:00419C98                add     esp, 4
.kaine0:00419C9B                xor     eax, eax
.kaine0:00419C9D                jnz     short loc_419CA2
.kaine0:00419C9F                jz      short loc_419CA2
.kaine0:00419C9F ; ---------------------------------------------------------------------------
.kaine0:00419CA1                db 0E8h
.kaine0:00419CA2 ; ---------------------------------------------------------------------------
.kaine0:00419CA2
.kaine0:00419CA2 loc_419CA2:                            ; CODE XREF: .kaine0:00419C9D↑j
.kaine0:00419CA2                                        ; .kaine0:00419C9F↑j
.kaine0:00419CA2                jnz     short loc_419CA7
.kaine0:00419CA4                jz      short loc_419CA7
.kaine0:00419CA4 ; ---------------------------------------------------------------------------
.kaine0:00419CA6                db 0E8h
.kaine0:00419CA7 ; ---------------------------------------------------------------------------
.kaine0:00419CA7
.kaine0:00419CA7 loc_419CA7:                            ; CODE XREF: .kaine0:loc_419CA2↑j
.kaine0:00419CA7                                        ; .kaine0:00419CA4↑j
.kaine0:00419CA7                call    loc_419CAD
.kaine0:00419CA7 ; ---------------------------------------------------------------------------
.kaine0:00419CAC                db 0E8h ; Þ
.kaine0:00419CAD ; ---------------------------------------------------------------------------
.kaine0:00419CAD
.kaine0:00419CAD loc_419CAD:                            ; CODE XREF: .kaine0:loc_419CA7↑p
.kaine0:00419CAD                call    loc_419CB4
.kaine0:00419CAD ; ---------------------------------------------------------------------------
.kaine0:00419CB2                db 0CDh
.kaine0:00419CB3                db  20h
.kaine0:00419CB4 ; ---------------------------------------------------------------------------
.kaine0:00419CB4
.kaine0:00419CB4 loc_419CB4:                            ; CODE XREF: .kaine0:loc_419CAD↑p
.kaine0:00419CB4                add     dword ptr [esp], 0Bh
.kaine0:00419CB8                add     dword ptr [esp+4], 13h
.kaine0:00419CBD                retn
.kaine0:00419CBD ; ---------------------------------------------------------------------------
.kaine0:00419CBE                db 0E9h ; Ú
.kaine0:00419CBF ; ---------------------------------------------------------------------------
```

```
.kaine0:00419CBF            xor     ebx, ebx
.kaine0:00419CC1            sub     edx, eax
.kaine0:00419CC3            jnz     short loc_419CC8
.kaine0:00419CC5            jz      short loc_419CC8
.kaine0:00419CC5 ; ---------------------------------------------------------------------------
.kaine0:00419CC7            db 0E8h
.kaine0:00419CC8 ; ---------------------------------------------------------------------------
.kaine0:00419CC8
.kaine0:00419CC8 loc_419CC8:                            ; CODE XREF: .kaine0:00419CC3↑j
.kaine0:00419CC8                                        ; .kaine0:00419CC5↑j
.kaine0:00419CC8            setz    bl
.kaine0:00419CCB            lea     edx, [esp+ebx*4]
.kaine0:00419CCE            call    loc_419CD4
.kaine0:00419CCE ; ---------------------------------------------------------------------------
.kaine0:00419CD3            db 0E8h
.kaine0:00419CD4 ; ---------------------------------------------------------------------------
.kaine0:00419CD4
.kaine0:00419CD4 loc_419CD4:                            ; CODE XREF: .kaine0:00419CCE↑p
.kaine0:00419CD4            call    loc_419CDB
```

```
.kaine0:00419CD4 ; ---------------------------------------------------------------------------
.kaine0:00419CD9            db 0CDh ; -
.kaine0:00419CDA            db 20h
.kaine0:00419CDB ; ---------------------------------------------------------------------------
.kaine0:00419CDB
.kaine0:00419CDB loc_419CDB:                            ; CODE XREF: .kaine0:loc_419CD4↑p
.kaine0:00419CDB                                        ; .kaine0:00419CF6↓j
.kaine0:00419CDB            add     dword ptr [esp], 0Bh
.kaine0:00419CDF            add     dword ptr [esp+4], 13h
.kaine0:00419CE4            retn
.kaine0:00419CE4 ; ---------------------------------------------------------------------------
.kaine0:00419CE5            db 0E9h ; Ú
.kaine0:00419CE6 ; ---------------------------------------------------------------------------
.kaine0:00419CE6            mov     esi, [edx]
.kaine0:00419CE8
.kaine0:00419CE8 loc_419CE8:                            ; CODE XREF: .kaine0:00419D2D↓j
.kaine0:00419CE8            lea     esi, [esi-6A53E524h]
.kaine0:00419CEE            jmp     esi
```

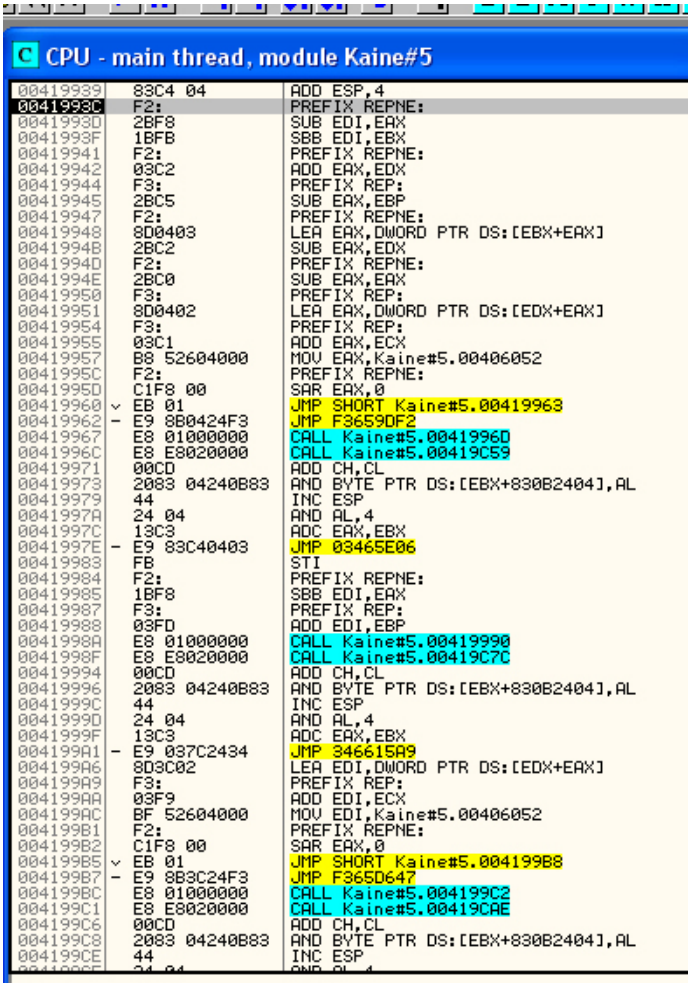
Comment franchir de telles routines ?

Tout d'abord, pour tracer ces routines, il faut tracer en « Step into » avec F7 (sous OllyDbg). Dans le cas contraire, le premier « double call » rencontré vous sera fatal. Ensuite, vous devez « localiser » le JMP ESI. Pour ça, avec un petit peu d'observation, vous repérez vite la fin de la routine à l'œil nu. Regardez les deux images ci-dessous. La première image vous présente le code vu lorsqu'on se situe en plein milieu de la routine et la seconde lorsqu'on s'approche de la fin…

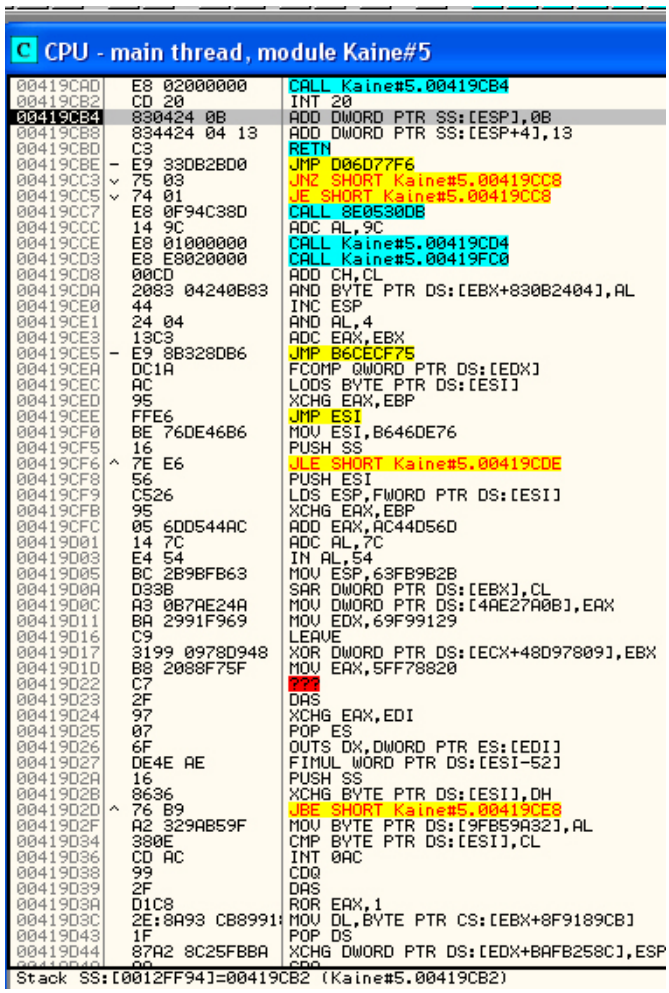
A partir de là, nous pouvons poser un BP conditionnel sur le JMP ESI qui se déclenchera dès que la condition entrée sera satisfaite. Pour cela, placez vous sur le JMP ESI, faites un MAJ + F2 et entrez l'une des deux conditions suivantes :

- ECX == 0
- ESI != 0x419A53

La première va breaker dès que le compteur de boucle ECX sera égal à 0. La seconde, équivalente à la première ici, va breaker dès que ESI ne pointera plus vers la routine de décryptage. Je précise cependant que toutes les routines de décryptage ne se terminent pas par un JMP ESI. Il existe également des JMP EDI, RET, JMP [EDX]. Dans le cas où vous adopteriez la condition 2, il faudra dans chaque cas adopter une condition adéquate : EDI != 0x419A53 , [ESP] != 0x419A53 ou [EDX] != 0x419A53 .



code de la routine (très régulier et répétitif)



code différent – nous sommes à la fin – on peut voir ici par chance le JMP ESI

JOUR 2 : Dimanche 22 mai 2005 : 10h00

J'ai bien avancé hier. J'ai franchi pas mal de layers, j'ai évité quelques tricks et même si j'ai pris un BSOD, je continue confiant. Mais… tous ces layers, ils sont si longs et si nombreux ! Je commence d'ailleurs à avoir mal au coude droit tellement je suis crispé sur F7…bizarre. J'espère au moins que ce n'est pas une tendinite.

6. LES ANTI-BPM - Exceptions et SEH

Utiliser les SEH (Structured Exception Handling) en tant qu'anti-debugger est une technique très ancienne et utilisée dans les premiers virus compatibles NT. Cette astuce permettait à l'époque de paralyser les émulateurs d'anti-virus, d'empêcher le tracing à l'aide de debuggers ring3 et d'écraser les debugs registers.

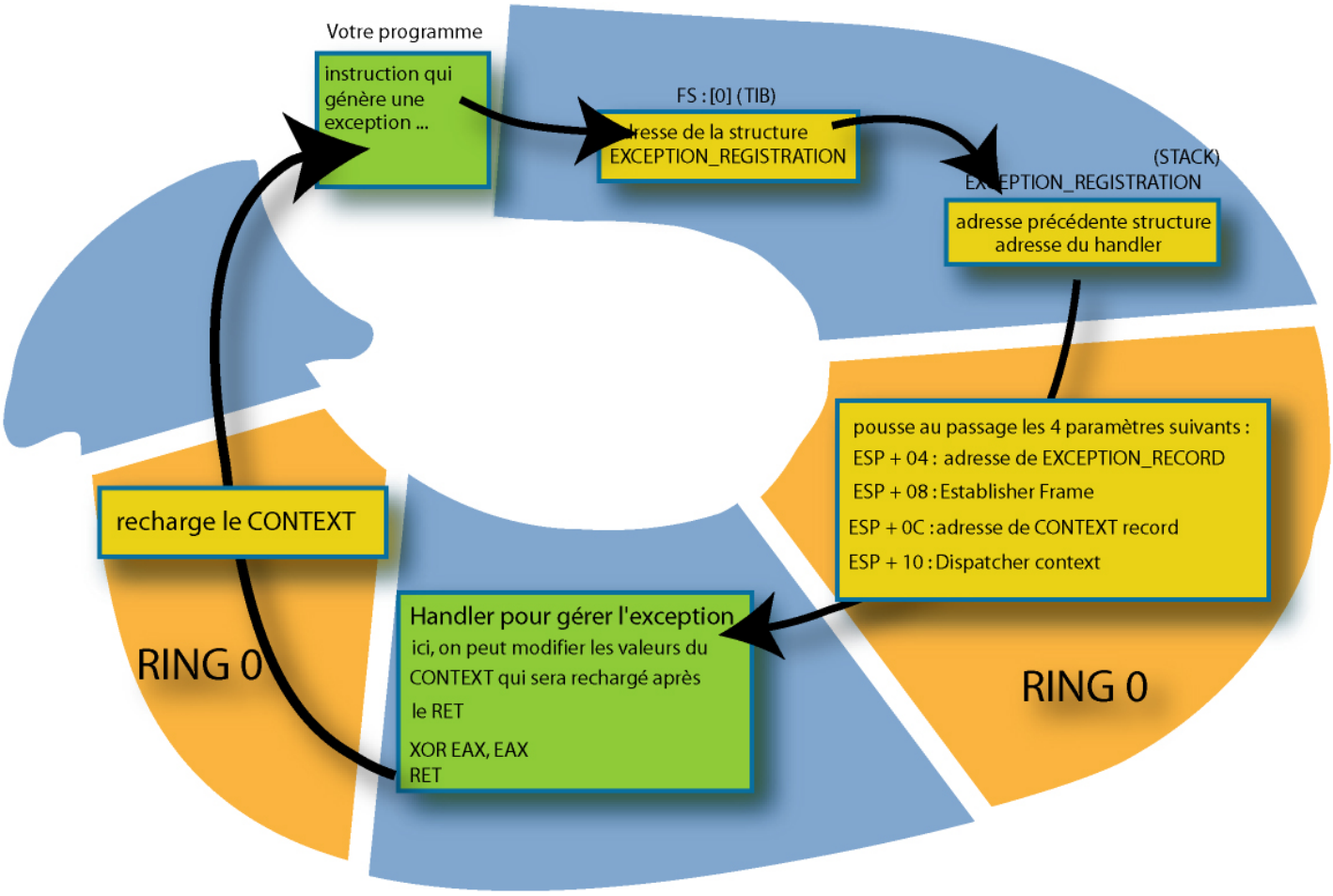
Il semblerait que le premier virus à utiliser cette technique soit Win32/Cabanas en 1997. L'auteur est inconnu mais était membre du célèbre groupe 29A. Voilà ce qu'on peut lire sur les SEH de Cabanas dans ***VIRUS BULLETIN NOVEMBER 1997*** « **Coping with Cabanas** » **de** ***Péter Ször*** :

> *The real problem is that the virus uses Structured Exception Handling (SEH) as an anti-debug function. Not knowing the form of C++'s __try and _except functions in assembly, I ran into this trap several times before it dawned on me - the goal of this function is to set a new SEH FRAME and generate an exception. When execution reaches the instruction which caused the exception, control is redirected to the operating system's Exception Handler (EIP will point into the kernel). This is very annoying and needs SoftIce to trace. The operating system's exception handler sets the exception type and returns to the application. As a result, no general protection fault will be displayed and the SEH FRAME will be removed.*

En ce qui nous concerne, K5 a la fâcheuse tendance à s'attaquer aux Memory Break Points (BPM) posés par l'intermédiaire des 4 registres de debug DR0, DR1, DR2 et DR3. En mode user (ring3), le seul moyen que je connaisse de supprimer des BPM est de générer une exception et de la contrôler à l'aide d'un SEH. Le système transmet alors au handler une copie du CONTEXT qui peut être modifiée à souhait. Dès que le handler rend la main au système, ce dernier recharge le CONTEXT avant de rendre à son tour la main au crackme.

Je reviens ici sur l'explication du fonctionnement d'un SEH. Vous pouvez survoler cette section si vous ne désirez pas voir en détail cette technique.

Comme je l'avais expliqué dans mon article sur TELock 0.98, voici illustré ce qu'il se passe au moment d'une exception.



Le système se branche sur une structure appelée EXCEPTION_REGISTRATION en passant par le TIB (Thread Information Block) alias le TEB (Thread Environment Block). Puis, il gère l'exception et pousse sur la pile 4 adresses :

ESP + 04 : pointeur vers une table appelée EXCEPTION RECORD qui contient :

EXCEPTION RECORD		
+0	ExceptionCode	contient le code de l'exception
+4	ExceptionFlag	0 = Continuable 1 = Non Continuable 2 = Stack is Unwinding
+8	NestedExceptionRecord	pointe vers une autre EXCEPTION RECORD si le handler crée à son tour une exception
+C	ExceptionAddress	Adresse où s'est produite l'exception
+10	NumberParameters	Nombre de paramètres additonnels (ci-dessous)
+14	Additionnal Information	Si ExceptionCode = C0000005 0 = read violation 1 = write violation
+18	Additionnal Information	Si ExceptionCode = C0000005 adresse de l'access violation

ESP + 08 : Establisher frame. Pointe vers la structure EXCEPTION_REGISTRATION

EXCEPTION REGISTRATION		
+0	Pointer to next SEH structure	pointe vers la prochaine structure EXCEPTION REGISTRATION
+4	SE Handler	Adresse du Handler

ESP + 0C : CONTEXT Record. Pointe vers la structure CONTEXT Record qui contient tous les registres du processus en cours et leur état au moment de l'exception :

CONTEXT RECORD

+0 context flags
(used when calling GetThreadContext)
DEBUG REGISTERS
+4 debug register #0
+8 debug register #1
+C debug register #2
+10 debug register #3
+14 debug register #6
+18 debug register #7
FLOATING POINT / MMX registers
+1C ControlWord
+20 StatusWord
+24 TagWord
+28 ErrorOffset
+2C ErrorSelector
+30 DataOffset
+34 DataSelector
+38 FP registers x 8 (10 bytes each)
+88 Cr0NpxState
SEGMENT REGISTERS
+8C gs register
+90 fs register
+94 es register
+98 ds register
ORDINARY REGISTERS
+9C edi register
+A0 esi register
+A4 ebx register
+A8 edx register
+AC ecx register
+B0 eax register
CONTROL REGISTERS
+B4 ebp register
+B8 eip register

+BC cs register
+C0 eflags register
+C4 esp register
+C8 ss register

ESP + 10 : DispatcherContext. Permet la communication entre le handler et la routine qui a appelé ce handler. Pointe entre autres, vers le ControlPC, PC de l'instruction qui a causé l'exception.

Ces valeurs peuvent être modifiées par le handler lui-même. Remarquez néanmoins que le CONTEXT n'est pas modifié tant que le système n'a pas repris la main. Le handler ne travaille ici que sur une COPIE de ce CONTEXT. Certains appellent cette petite astuce « le passage de ring3 à ring0 par SEH ». Vous pouvez donc tranquillement modifier tous les registres avant de les recharger. Le plus intéressant ici est la possibilité de modifier les « DEBUGS REGISTERS ».

DEBUGS REGISTERS :

Il s'agit de registres utilisés pour le debuggage. Ils sont au nombre de 8 (DR0 – DR7). Les 4 premiers sont réservés pour contenir des adresses linéaires où l'on veut poser un Hardware Break Point aussi appelé Memory Break Point (BPM). Le registre DR7 (Debug Control Register), permet de gérer les conditions d'application de ces Break Points. Pour appliquer la technique de l'anti-BPM, il suffit de mettre à zéro les 4 Break Points DR0, DR1, DR2 et DR3 et de fixer les bonnes conditions dans DR7. En général, on peut voir dans un handler les conditions suivantes :

```
MOV ECX,DWORD PTR SS:[ESP+0C]      Récupère l'adresse du CONTEXT Record
MOV DWORD PTR DS:[ECX+4],0         mise à zéro de DR0
MOV DWORD PTR DS:[ECX+8],0         mise à zéro de DR1
MOV DWORD PTR DS:[ECX+C],0         mise à zéro de DR2
MOV DWORD PTR DS:[ECX+10],0        mise à zéro de DR3
MOV DWORD PTR DS:[ECX+14],0        mise à zéro de DR6
MOV DWORD PTR DS:[ECX+18],155      fixer DR7 à 155
```

On met donc à zéro les premiers DR et on fixe à 155h le DR7. Voilà ce que signifie cette valeur :

155h = 101010101b

DR7																						
LEN 3	R/W 3	LEN 2	R/W 2	LEN 1	R/W 1	LEN 0	R/W 0	0	0	GD	0	0	1	GE	LE	G3	L3	G2	L2	G1	L1	G0	L0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**

On active un certain nombre de flags :

L0 = 1 : Activer DR0 local (uniquement valable pour ce process). Ce flag doit être armé si on veut que le système prenne en compte l'écrasement du DR0 !
G0 = 0 : Désactiver DR0 global (valable pour toutes les tâches)

L1 = 1 : Activer DR1 local (uniquement valable pour ce process). Même remarque
G1 = 0 : Désactiver DR1 global (valable pour toutes les tâches)

L2 = 1 : Activer DR2 local (uniquement valable pour ce process). Même remarque
G2 = 0 : Désactiver DR2 global (valable pour toutes les tâches)

L3 = 1 : Activer DR3 local (uniquement valable pour ce process). Même remarque
G3 = 0 : Désactiver DR3 global (valable pour toutes les tâches)

LE = 1 : Activé pour un souci de compatibilité.

Pour une vue complète des flags de DR7, je vous invite à consulter la documentation INTEL.

La technique des SEH est très connue dans le monde du packing et des virii. Elle est assez efficace pour se prémunir des BPM mais elle a malheureusement une très grosse faiblesse. Une exception arrête l'exécution d'un débuggage et constitue donc un Break Point naturel. Si le loader du crackme est équipé d'exceptions placées de façon linéaire du début à la fin, il suffit de s'en servir de Breaks Points naturels pour avancer très vite dans ce loader jusqu'à l'OEP. Les BPM sont alors annexes et peuvent même être posés après avoir rencontré la dernière exception. L'auteur de Kaine#5 connaît depuis longtemps cette énorme faiblesse et a réussi ici à contourner de façon ingénieuse cette difficulté. L'idée est d'utiliser le hasard ! Voilà, sur un exemple parmi tant d'autres, comment fonctionne cette astuce :

```
CALL L007                                  ; pousse sur la pile l'adresse du handler
PUSH ECX
MOV ECX,DWORD PTR SS:[ESP+10]              ; Récupère l'adresse du CONTEXT Record
XOR EAX,EAX
PUSH EBX
MOV EBX,DWORD PTR DS:[ECX+C4]              ; Récupère l'adresse située sur la pile (L008)
MOV EBX,DWORD PTR DS:[EBX]                 ; au moment de l'exception
ADD DWORD PTR DS:[ECX+C4],4                ; Ajoute 4 à ESP
MOV DWORD PTR DS:[ECX+B8],EBX              ; Changer EIP (EIP = L008)
MOV DWORD PTR DS:[ECX+4],EAX               ; mise à zéro de DR0
MOV DWORD PTR DS:[ECX+8],EAX               ; mise à zéro de DR1
MOV DWORD PTR DS:[ECX+C],EAX               ; mise à zéro de DR2
MOV DWORD PTR DS:[ECX+10],EAX              ; mise à zéro de DR3
MOV DWORD PTR DS:[ECX+14],EAX              ; mise à zéro de DR6
MOV DWORD PTR DS:[ECX+18],155              ; fixer DR7 à 155
POP EBX
POP ECX
RETN                                       ; rend la main au système - EAX = 0 implique
                                           ; que le CONTEXT sera rechargé
L007:
PUSH DWORD PTR FS:[0]                      ; Installation du SEH - Modifier le contenu du
MOV DWORD PTR FS:[0],ESP                   ; TIB
SUB EDI,EDI                                ; fais un tirage aléatoire
RDTSC                                      ; Si EAX = 1, il ne se passe rien et on saute
AND EAX,1                                  ; en L008
TEST EAX,EAX                               ; Si EAX = 0, on va générer une exception
JNZ L008
 RDTSC
AND EAX,0FFFFFFF                           ; Génère l'exception - appel une adresse
ADD EAX,C0000000                           ; inexistante
CALL EAX
L008:
NOP
POP DWORD PTR FS:[0]                       ; fin du SEH - restauration du contenu du TIB
ADD ESP,4
```

Comme vous pouvez le voir, l'exception qui déclenche le SEH n'est pas systématiquement exécutée. Notez aussi le « ADD EAX, C00000000 » qui permet de pointer vers une adresse réellement inaccessible. Sans cela, le résultat pourrait être désastreux. En effet, si le call pointe vers une adresse accessible en ring3, le code exécuté alors risque de modifier ESP. *(Je précise cela puisque j'ai moi-même tenté, il y a quelques mois, de générer des exceptions en utilisant une technique similaire. J'ai malheureusement été moins inspiré que Kaine puisque je n'ai pas pensé à fixer la partie supérieure de l'adresse et j'avais donc abandonné l'idée).* Ceci aura pour effet de crasher le programme puisque le SEH ne sera plus installé. Pour gérer le hasard, on utilise ici l'instruction RDTSC. Voici ce que l'on peut lire dans la documentation INTEL :

RDTSC—Read Time-Stamp Counter
Description
Loads the current value of the processor's time-stamp counter into the EDX:EAX registers. The time-stamp counter is contained in a 64-bit MSR. The high-order 32 bits of the MSR are loaded into the EDX register, and the low-order 32 bits are loaded into the EAX register. The processor monotonically increments the time-stamp counter MSR every clock cycle and resets it to 0 whenever the processor is reset. See "Time Stamp Counter" in Chapter 15 of the *IA-32 Intel Architecture Software Developer's Manual, Volume 3* for specific details of the time stamp counter behavior.

When in protected or virtual 8086 mode, the time stamp disable (TSD) flag in register CR4 restricts the use of the RDTSC instruction as follows. When the TSD flag is clear, the RDTSC instruction can be executed at any privilege level; when the flag is set, the instruction can only be executed at privilege level 0. (When in real-address mode, the RDTSC instruction is always enabled.)
The time-stamp counter can also be read with the RDMSR instruction, when executing at privilege level 0.
The RDTSC instruction is not a serializing instruction. Thus, it does not necessarily wait until all previous instructions have been executed before reading the counter. Similarly, subsequent instructions may begin execution before the read operation is performed.
This instruction was introduced into the IA-32 Architecture in the Pentium processor.

Il s'agit donc d'une instruction qui renvoie le nombre de cycles écoulés depuis le lancement du programme. EAX contient la partie basse de ce temps et « peut » donc servir comme valeur aléatoire. Attention cependant aux restrictions possibles sur certaines machines qui peuvent être équipées de drivers qui passent RDTSC comme une instruction privilégiée.

On ne peut donc pas profiter de façon régulière et sûre d'une exception particulière. On peut donc rapidement dresser un premier constat :

1 . Les BPM sont supprimés de façon intensive.
2 . Les SEH ne peuvent pas être utilisés de façon fiable comme BP naturels.
3 . Le crackme est crypté par plusieurs centaines de layers (peut-être plus d'un millier) et les routines de décryptage sont longues et très obfusquées.

Les techniques d'attaque sont ainsi réduites mais non anéanties. Le crackme utilise des fonctions de l'API qui vont nous permettre d'y loger des BPX. Ceci nous permettra de revenir à proximité d'un endroit précis lors d'une étude ultérieure. Néanmoins, le binaire ne va pas se laisser faire et va tenter de nous arrêter avec des techniques anti-debugging .

7. TECHNIQUES ANTI-DEBUGGING.

7.1. IsDebuggerPresent.

La première technique pour détecter la présence d'un debugger est l'utilisation de la fonction kernel32.IsDebuggerPresent qui renvoie 0 dans EAX si le programme n'est pas debuggé.

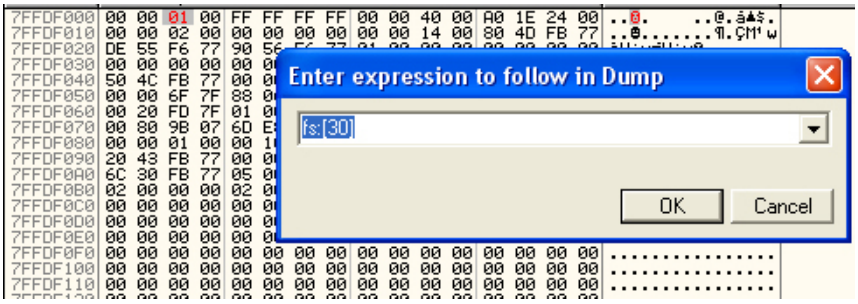
```
MOV EAX, DWORD PTR FS:[18]          Récupère l'adresse du TEB
MOV EAX, DWORD PTR DS:[EAX+30]      Récupère l'adresse du PEB
MOVZX EAX, BYTE PTR DS:[EAX+2]      Récupère la valeur du 3ème octet
RETN
```

Cette fonction accède au PEB (Process Environment Block), section du processus qui contient des informations précises sur celui-ci. On peut y accéder directement sans passer par le TEB en allant à l'adresse fs :[30] . Voici le début de cette structure :

PEB:

+000 byte InheritedAddressSpace
+001 byte ReadImageFileExecOptions
+002 byte BeingDebugged
+003 byte SpareBool
+004 void *Mutant
+008 void *ImageBaseAddress

Pour « masquer » le debugger, il suffit donc au démarrage du debuggage du binaire de placer la valeur 00 sur le 3ème octet du PEB. Vous allez à l'adresse fs :[30] avec CTRL + G dans la fenêtre de dump (la fenêtre en bas à gauche) et vous remplacez le 01 par 00 comme le montre l'image ci-dessous :



J'ai opté pour la facilité ici comme de nombreux reversers utilisant OllyDbg. Les plugins IsDebuggerPresent ou HideDebugger font ce travail à notre place. Personnellement, j'utilise HideDebugger.dll qui permet de fixer une bonne fois pour toutes cette option.

7.2. FindWindow.

La seconde fonction classique rencontrée est FindWindowA. Elle permet de récupérer le handle de la fenêtre principale dont la classe répond à un nom particulier. Ici, on essaie de récupérer le handle le fenêtre principale de OllyDebugger. Cette instance de classe est issue de la

classe nommée « OLLYDBG ». Si la fonction retourne un handle non nul, ceci signifie que OllyDbg est actif. Voilà comment mettre en place cette technique :

FICHIER “FindWindow.asm”

```
.386
.Model Flat ,StdCall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32.lib
include \masm32\include\user32.inc
includelib \masm32\lib\user32.lib

.Data

nom_de_classe     BYTE    "OLLYDBG",0
texte             BYTE"Type 02:",0Dh,0Ah,"Debugger Detection",0
titre             BYTE"#5 Error"

.Code
Main:
  push 0                    ; pas de titre spécifique
  push offset nom_de_classe
  call FindWindowA
  cmp eax, 0
  je @F
  push MB_OK + MB_ICONHAND + MB_APPLMODAL       ; OllyDbg est détecté
  push offset titre
  push offset texte
  push 0
  call MessageBoxA

@@:
  push 0
  call ExitProcess
End Main
```

Comme pour les tricks précédents, le plugin HideDebugger.dll permet de « masquer » la classe OLLYDBG. Si vous êtes joueur, vous pouvez également patcher vous-même OllyDbg. Il suffit de chercher le nom de la classe en cherchant la fonction RegisterClassA et de patcher ce nom.

7.3. CreateFile sur driver.

Voilà enfin le premier trick pour détecter réellement Soft Ice. Lorsque Soft Ice est actif, un service du nom de NTICE est chargé en mémoire. Le driver en question est ntice.sys et sa présence peut-être détectée comme suit :

FICHIER “CreateFile_NTICE.asm”

```
.386
.Model Flat ,StdCall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32.lib
include \masm32\include\user32.inc
includelib \masm32\lib\user32.lib
```

```
.Data

service    BYTE  "\\.\NTICE",0
titre      BYTE  "#5 Error",0
texte      BYTE  "Type : 07",0Dh,0Ah,"Description : Debugger Detection",0
.Code
Main:
        push 0
        push FILE_ATTRIBUTE_NORMAL
        push OPEN_EXISTING
        push 0
        push FILE_SHARE_READ + FILE_SHARE_WRITE
        push GENERIC_READ + GENERIC_WRITE
        push offset service
        Call CreateFileA
        cmp eax, -1
        jz @F
        push MB_OK + MB_ICONHAND + MB_APPLMODAL        ; Soft Ice est détecté
        push offset titre
        push offset texte
        push 0
        call MessageBoxA

@@:
        push 0
        call ExitProcess
End Main
```

Le plugin IceExt permet de contourner cette détection sans difficulté.

JOUR 3 : Lundi 23 mai 2005 : 13h35

J'ai enfin atteint un zone virtuelle et j'ai réussi à tracer assez loin il me semble. Enfin, je le crois… malheureusement, je viens de tomber sur une zone remplie de zéros et pas moyen d'aller plus loin ! Bea !! rien n'est magique ! mais je ne vois pas le truc… pfff..et Kharneth doit avoir une belle avance. Tu es trop lent Bea, tu es trop lent ! Et ce maudit coude qui me fait souffrir !

7.4 .DebugActiveProcess / Injection de code dans un Processus.

Il s'agit du premier trick « inédit » découvert par Kaine mais déjà utilisé dans Kaine#4.exe. L'idée est assez cocasse. Le binaire va scanner tous les processus actifs et va rechercher dans chacun d'entre eux deux dll : IsDebug.dll et HideDebugger.dll . Il s'agit de deux plugins de OllyDbg. Si l'un de ces deux modules est trouvé, le crackme va alors debugger le processus qui la contient. Le Debuggee (Kaine#5.exe) va debugger le debuggueur (OllyDbg) !! En plus de cela, il va modifier le code du debugger pour qu'il crashe complètement. Voici l'idée en détail.

Pour commencer, K5 va scanner tous les processus actifs à la recherche de celui de OllyDbg. Dès qu'il l'a repéré, il s'y attache et modifie le binaire Olly.exe en remplaçant le code suivant :

```
004788E1  PUSH EAX                                  ; /hThread
004788E2  CALL <JMP.&KERNEL32.ResumeThread>         ; \ResumeThread
```

par celui-ci :

```
004788E1  PUSH 0
004788E3  CALL <JMP.&KERNEL32.ExitProcess>
```

Puis, il laisse le processus vivre sa vie. Le premier effet visible est un ralentissement du debugger puis, si OllyDbg tente d'exécuter son ResumeThread, il va exécuter le code malicieux ExitProcess. Ce ExitProcess est très sournois : il arrête le processus « debuggee » OllyDbg mais en même temps il arrête le processus « debugger » OllyDbg !

Les dégâts de ce trick sont assez spectaculaires et considérables : OllyDbg ralentit, se fige, et au final (quelques secondes après), le système d'exploitation se crashe littéralement : BSOD ! Et vous êtes bon pour redémarrer la machine !!

Voilà comment se met en place une telle technique :

FICHIER "DebugActiveProcess.asm"

```
;*************************************************************
;
;          Kaine's trick seen in Kaine#5.exe
;
; source code from BeatriX's solution
;*************************************************************

.386
.Model Flat ,StdCall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32.lib
include \masm32\include\user32.inc
includelib \masm32\lib\user32.lib


.Data
lpNumberOfBytesWritten  DWORD   0

buffer_resumethread     BYTE    06Ah,00                     ; push 0
                        BYTE    0FFh,015h,0D4h,0D2h,50h,00,00   ; call ExitProcess

Module_HideDebugger     BYTE    "HideDebugger.dll",0
Module_IsDebugger       BYTE    "isdebug.dll",0
hProcessSnap            DWORD   0
hModuleSnap             DWORD   0
pProcessEntry           PROCESSENTRY32 <>
pModuleEntry            MODULEENTRY32 <>
lpDebugEvent            DEBUG_EVENT <>
titre                   BYTE    "Error#5",0
texte                   BYTE    "Type : 06",0Dh,0Ah,"Description  : None",0


.Code
```

```
Main:
  push 0
  push TH32CS_SNAPPROCESS
  call CreateToolhelp32Snapshot                 ; Snapshot of processes
  mov hProcessSnap, eax



  mov pProcessEntry.dwSize, SIZEOF PROCESSENTRY32
  push offset pProcessEntry
  push hProcessSnap
  call Process32First                           ; Get first process

Test_Process:
  cmp eax, 0
  jnz Scan_Process
  jmp sortie

Scan_Process:

  push pProcessEntry.th32ProcessID
  push TH32CS_SNAPMODULE
  call CreateToolhelp32Snapshot                 ; Snapshot of modules in the current process
  mov hModuleSnap, eax


  mov pModuleEntry.dwSize, SIZEOF MODULEENTRY32
  push offset pModuleEntry
  push hModuleSnap
  call Module32First                            ; Get first Module

Test_Module:
  cmp eax, 1
  jnz Next_Process

  push offset pModuleEntry.szModule
  push offset Module_IsDebugger
  call lstrcmpiA                                ; compare with "isdebug.dll"
  test eax, eax
  jnz @Compare2
  jmp CRASH_OLLY


@Compare2:
  push offset pModuleEntry.szModule
  push offset Module_HideDebugger
  call lstrcmpiA                                ; compare with "HideDebugger.dll"
  test eax, eax
  jnz NextModule
  jmp CRASH_OLLY

NextModule:
  push offset pModuleEntry
  push hModuleSnap
  call Module32Next
  jmp Test_Module

Next_Process:
  push hModuleSnap
  call CloseHandle
  push offset pProcessEntry
```

```
  push hProcessSnap
  call Process32Next
  jmp Test_Process

@Error6:
  push MB_OK + MB_ICONHAND
  push offset titre
  push offset texte
  push 0
  call MessageBoxA
  push 0
  call ExitProcess

; ********************************** Debug OllyDbg and crash it !

CRASH_OLLY:
  push pProcessEntry.th32ProcessID
  call DebugActiveProcess

@DEBUG:
  push INFINITE
  push offset lpDebugEvent
  call WaitForDebugEvent

  lea ebx, lpDebugEvent.dwDebugEventCode
  cmp dword ptr [ebx], 5              ; EXIT_THREAD_DEBUG_EVENT
  je @Error6
  cmp Dword ptr [ebx], 3              ; CREATE_PROCESS_DEBUG_EVENT
  jnz @F
  push lpNumberOfBytesWritten
  push 8
  push offset buffer_resumethread
  push 4788E1h
  push pProcessEntry.th32ProcessID
  call WriteProcessMemory
  jmp Continue_Debug

@@:
  cmp dword ptr [ebx],1               ; EXCEPTION_DEBUG_EVENT
  jnz Continue_Debug
  cmp dword ptr [ebx+0Ch], 80000003h
  jnz Continue_Debug

  push DBG_CONTINUE
  push lpDebugEvent.dwThreadId
  push lpDebugEvent.dwProcessId
  call ContinueDebugEvent
jmp @DEBUG

Continue_Debug:
  push DBG_EXCEPTION_NOT_HANDLED
  push lpDebugEvent.dwThreadId
  push lpDebugEvent.dwProcessId
  call ContinueDebugEvent
jmp @DEBUG

sortie:

      push 0
      call ExitProcess
End Main
```

Pour contourner cette difficulté, il suffit de changer le nom des deux dll recherchées comme ceci (par exemple) :

IsDebug.dll sera renommée Is_Debug.dll.
HideDebugger.dll sera renommée Hide_Debugger.dll.

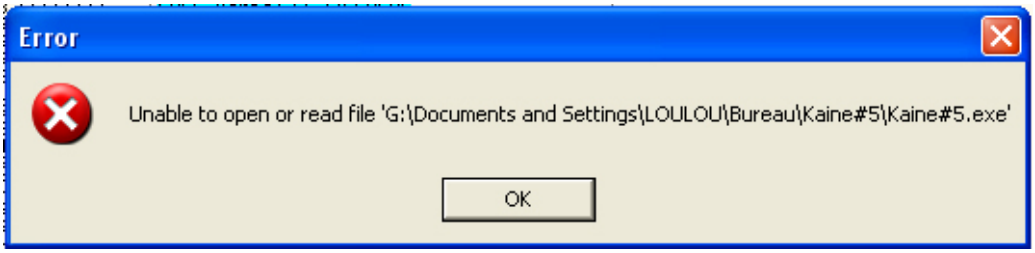
7.5 .CreateFileA sur « Kaine#5.exe ».

Voilà une autre idée assez intéressante. Quand vous debuggez un exécutable avec OllyDbg, vous êtes souvent amené et ce, à plusieurs reprises, à relancer le programme en cliquant sur la double flèche située en haut à gauche ou en saisissant le raccourci CTRL+F2. Ce trick bloque complètement cette possibilité de relance et nous oblige donc à fermer OllyDbg puis à le relancer !! Voilà comment mettre en œuvre cette technique :

FICHIER “CreateFile.asm”

```
.386
.Model Flat ,StdCall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32.lib

.Data

file    db  "createfile.exe",0

.Code
Main:
        push 0
        push FILE_ATTRIBUTE_NORMAL
        push OPEN_EXISTING
        push 0
        push 0
        push GENERIC_READ       ; ouvrir en lecture
        push offset file        ; nom du fichier
        Call CreateFileA
        push eax                ; <-- on ne peut plus recharger le binaire à partir d'ici
        call CloseHandle
        push 0
        call ExitProcess
End Main
```

Le fichier s’ouvre lui-même en lecture (pas en écriture) et du coup, on obtient ceci si on essaie de relancer le programme :



A moins d'empêcher le binaire de s'ouvrir en posant un BPX sur CreateFileA, vous ne pouvez pas contourner ce petit handicap. J'ai personnellement opté pour travailler avec ce problème sans essayer de m'en débarrasser…

7.6 .RegOpenKeyExA sur « IceExt ».

On s'attaque encore une fois à Soft Ice. Comme nous l'avons vu plus haut, grâce à l'extension IceExt, Soft Ice peut se prémunir des tricks classiques. Néanmoins, cette extension peut être détectée et sa faiblesse se situe dans la base de registres. Quand ce service est installé, il crée une clé nommée « IceExt » que l'on peut détecter en faisant ceci :

FICHIER "RegOpenKeyEx.asm"

```
.386
.Model Flat ,StdCall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32.lib
include \masm32\include\advapi32.inc
includelib \masm32\lib\advapi32.lib
include \masm32\include\user32.inc
includelib \masm32\lib\user32.lib

.Data

phandle DWORD   0
subkey  BYTE    "SYSTEM\CurrentControlSet\Services\IceExt\",0
titre   BYTE    "#5 Error",0
texte   BYTE    "Type : 03",0Dh,0Ah,"Description : Debugger Detection",0

.Code
Main:
  push offset phandle
  push KEY_READ
  push 0
  push offset subkey
  push HKEY_LOCAL_MACHINE
  call RegOpenKeyExA
  cmp eax,0                              ; ERROR_SUCCESS
  jne @F
  push phandle
  call RegCloseKey
  push MB_OK + MB_ICONHAND + MB_APPLMODAL
  push offset titre
  push offset texte
```

```
  push 0
  call MessageBoxA
@@:
  push 0
  call ExitProcess
End Main
```

7.7 .Closehandle + RDTSC.

Voilà une technique originale et intéressante. Elle repose sur un principe très simple. Si, au cours d'un debuggage, une exception est causée de façon involontaire ou intentionnelle, le debugger arrête l'exécution du programme et dans le cas d'une exception « CONTINUABLE », propose à l'utilisateur de relancer l'exécution en tapant l'une des séquences SHIFT+F7, SHIFT+F8 ou SHIFT+F9. Le programme reprend alors son cours normal sans même avoir besoin de gérer l'exception.

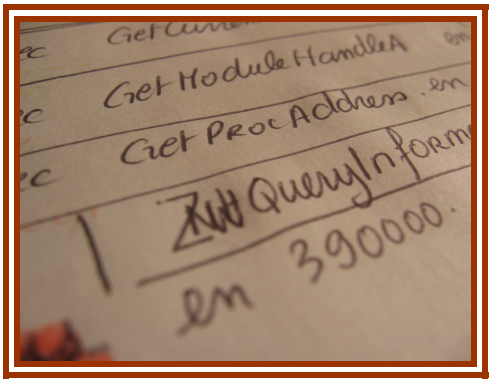
L'idée ici est d'obliger le debugger à arrêter l'exécution du binaire en générant volontairement une exception de type « INVALID_HANDLE ». Cette tâche est réalisée de façon très simple en passant comme paramètre un handle inexistant à la fonction Closehandle. Cela dit, arrêter le debugger en pleine course ne pose pas de problème en soi.

Ici, en plus de stopper l'exécution, le binaire prend en sandwich CloseHandle avec deux RDTSC. Si la différence des deux relevés est supérieure à une certaine valeur, cela signifie qu'un debugger est actif. Voilà comment se met en place cette technique :

FICHIER "CloseHandle.asm"

```
.586
.Model Flat ,StdCall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32.lib


.Code
Main:
jmp @F
adresse dword   0               ; adresse qui va contenir le handle invalide
@@:
  rdtsc
  mov adresse, eax
  push eax
  call CloseHandle              ; génère une exception de type INVALID_HANDLE
  rdtsc
  sub eax, adresse
  cmp eax, 0E0000h
  jb @F
  rdtsc
  jmp eax                       ; saute n'importe où !
@@:
  push 0
  call ExitProcess
End Main
```

Je n'ai pas trouvé de moyen ingénieux pour contourner ceci. J'ai donc, à chaque fois, fixé EAX à zéro avant que la comparaison ne se fasse.



JOUR 4 : Mardi 24 mai 2005 : 11h40

ça y est ! j'ai réussi à passer ce trick qui m'a pourri l'existence durant quelques heures : ZwQueryInformationProcess ! C'est très bien vu ça ! j'ai aussi récupéré un driver nommé Protect.sys..il semble être crypté ! Pas moyen d'utiliser IDA !! J'ai appris également que Kharneth s'est attardé sur le trick du CloseHandle…héhé ☺ j'ai donc de l'avance…

7.8 .ZwQueryInformationProcess.

Voici l'un des tricks « furtifs » de K5. Il est furtif car son effet n'est pas perçu immédiatement. En effet, si le trick fonctionne, K5 va alors écraser **trois zones** de son propre code qui se situe bien en aval de ce trick et le malheureux reverser se rendra compte des dégâts au moment où il tombera sur les zones sinistrées. Il faut alors « remonter » tant bien que mal jusqu'à la source de ce problème. C'est ainsi que j'ai découvert ici l'usage de cette fonction en tant qu'anti-debugger.

Voilà d'abord ce que l'on peut obtenir dans « **Windows NT2000 Native API** » de Gary Nebett à propos de cette fonction :

ZwQueryInformationProcess

```
ZwQueryInformationProcess retrieves information about a process object.
NTSYSAPI
NTSTATUS
NTAPI
ZwQueryInformationProcess(
IN HANDLE ProcessHandle,
IN PROCESSINFOCLASS ProcessInformationClass,
OUT PVOID ProcessInformation,
IN ULONG ProcessInformationLength,
OUT PULONG ReturnLength OPTIONAL
);
```

Parameters

ProcessHandle
A handle to a process object.The handle must grant PROCESS_QUERY_INFORMATION access. Some information classes also require PROCESS_VM_READ access.

ProcessInformationClass
Specifies the type of process information to be set.The permitted values are drawn from the enumeration **PROCESSINFOCLASS**, described in the following section.

ProcessInformation
Points to a caller-allocated buffer or variable that contains the process information to

be set.

ProcessInformationLength
Specifies the size in bytes of ProcessInformation, which the caller should set according to the given ProcessInformationClass.

Return Value
Returns STATUS_SUCCESS or an error status, such as STATUS_ACCESS_DENIED, STATUS_INVALID_HANDLE, STATUS_INVALID_INFO_CLASS, STATUS_INFO_LENGTH_MISMATCH, STATUS_PORT_ALREADY_SET, STATUS_PRIVILEGE_NOT_HELD, or STATUS_PROCESS_IS_TERMINATING.
Related Win32 Functions
SetProcessAffinityMask, SetProcessPriorityBoost, SetProcessWorkingSetSize, SetErrorMode.
Remarks
None.

PROCESSINFOCLASS

typedef enum _PROCESSINFOCLASS {

ProcessBasicInformation,	0
ProcessQuotaLimits,	1
ProcessIoCounters,	2
ProcessVmCounters,	3
ProcessTimes,	4
ProcessBasePriority,	5
ProcessRaisePriority,	6
ProcessDebugPort,	7
ProcessExceptionPort	8
ProcessAccessToken,	9
ProcessLdtInformation,	10
ProcessLdtSize,	11
ProcessDefaultHardErrorMode,	12
ProcessIoPortHandlers,	13
ProcessPooledUsageAndLimits,	14
ProcessWorkingSetWatch,	15
ProcessUserModeIOPL,	16
ProcessEnableAlignmentFaultFixup,	17
ProcessPriorityClass,	18
ProcessWx86Information,	19
ProcessHandleCount,	20
ProcessAffinityMask,	21
ProcessPriorityBoost,	22
ProcessDeviceMap,	23

A priori, cette fonction ne semble pas prévue pour de l'anti-debugging. En réalité, si on lui passe le paramètre ***ProcessDebugPort***, elle peut détecter la présence d'un debugger ring3. Cette fonction est connue depuis au moins janvier 1997 puisque Matt Pietrek, dans son fameux article « **Under The Hood** » écrit à son propos :

ProcessDebugPort *NtQueryInformationProcess fills in a DWORD with the port number of the debugger for the process being queried. The ProcessInformationLength parameter to NtQueryInformationProcess should be set to sizeof(DWORD). The debug port is a value that's useless to ring 3 code. However, you can infer that a nonzero debug port means that the process is being run under the control of a ring 3 debugger such as the Visual C++ IDE or Turbo Debugger.*

Voici comment est utilisée cette fonction dans le K5 :

FICHIER “ZwQueryInformationProcess.asm”

```
.386
.Model Flat ,StdCall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32.lib
include \masm32\include\ntdll.inc
includelib \masm32\lib\ntdll.lib
include \masm32\include\user32.inc
includelib \masm32\lib\user32.lib

.const

ProcessDebugPort = 7

.Data

ProcessInfo DWORD   0
titre       BYTE    "Information : ",0
texte       BYTE    "No debugger detected",0

.Code
Main:
 push 0
 push 4
 push offset ProcessInfo
 push ProcessDebugPort
 push -1
 call ZwQueryInformationProcess
 mov eax, ProcessInfo
 test eax, eax
 je @F
 mov edi, offset Protected_area
 mov ecx, offset End_Protected_area
 sub ecx, edi
 xor eax, eax
 rep stosb
@@:
 nop
 nop
 nop
 nop
 nop
 nop
 nop
Protected_area:
 push 0
 push offset titre
 push offset texte
```

```
push 0
call MessageBoxA
End_Protected_area:
push 0
call ExitProcess
End Main
```

Comme vous pouvez le constater, j'ai essayé de reproduire le comportement de K5. J'ai simplifié en réalité le comportement du crackme. Si OllyDbg est attaché au process, la fonction ZwQueryInformationProcess renverra inévitablement -1. Dans le cas contraire, elle renverra 0. Vous constatez alors que le programme écrase une partie du code située bien après (simulé ici par des nop). Si OllyDbg est absent, la messagebox s'affiche sans problème. En cas de debuggage, le programme crashe sur la zone des « zéros ». K5 se comporte un peu différemment :

1) K5 appelle ZwQueryInformationProcess

2) Si un debugger est détecté, il modifie 3 zones sensibles qui sont :

Zone1 : EP2 + 80CA1h → EP2 + 80FB1h remplie par des « E9h »

Zone2 : EP2 + 9A5D3h → EP2 + 9A839h : remplie par des « 66h », « 65h », « 64h », etc…

Zone3 : EP2 + 9761Ch → EP2 + 99447h : remplie par des octets tirés aléatoirement par un RDTSC.

3) La Zone 2 est décryptée plus tard comme ceci :

Sub byte ptr [edi], cl (avec cl variant de 66h à 0). Ceci a pour effet de mettre à zéros tous les octets si le trick anti-debug a été mis en place. Si vous tracez tranquillement, sans trop regarder ce qu'il se passe, vous allez tomber sur un **OCEAN DE ZEROS.**

Pour contourner cette difficulté, il suffit de contourner la fonction ZwQueryInformationProcess. Au début du debbugage, je cherche la fonction en faisant un click droit sur la fenêtre de code et en sélectionnant « Search for » « Name in all modules ». Sous XP, j'obtiens une fonction qui ressemble à ça :

```
77F66035  MOV EAX,9A
77F6603A  MOV EDX,7FFE0300
77F6603F  CALL EDX
77F66041  RETN 14
```

Dès que j'arrive sur cette fonction, je modifie EIP pour qu'il pointe sur le RET14. On peut aussi poser des Bp on memory write sur les zones sensibles attaquées par ce trick et contourner les boucles d'écrasement.

7.9 .PEB / PEB_LDR_DATA.

Voilà encore un trick « furtif ». Il est utilisé selon le même principe que le trick précédent. Si le piège fonctionne, il écrase une zone sensible située beaucoup plus loin dans le code. Il s'agit en l'occurrence de la Zone 3 déjà modifiée une première fois par le piège précédent. Cette zone est en fait très critique puisqu'elle précède une série de redirections de fonctions qui vont être utilisées pour lancer le premier driver Protect.sys. A la différence du trick précédent, celui-ci n'utilise pas de fonctions de l'API. C'est ce qu'on peut appeler une véritable découverte ! Elle a été trouvée par Neitsa et j'aurai bien du mal à vous expliquer pourquoi cela fonctionne. J'ai réalisé un test pour mettre en évidence cette propriété assez mystérieuse pour moi, je dois l'avouer.

Voici tout d'abord le fameux code que j'ai clarifié puisqu'il se trouve dans le K5 muté et très obfusqué :

FICHIER "PEB.asm"

```
;*************************************************************
;
;          Neitsa's trick seen in Kaine#5.exe
;
; source code from BeatriX's solution
;*************************************************************

.386
.Model Flat ,StdCall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32.lib
include \masm32\include\user32.inc
includelib \masm32\lib\user32.lib
assume fs:nothing
.Data

titre   BYTE    "Information :",0
texte   BYTE    "No debugger detected",0
hModule DWORD   0

.Code
Main:
push 0
call GetModuleHandleA
mov hModule, eax

push dword ptr fs:[30h]
mov eax, dword ptr [esp]
lea eax, dword ptr [eax+0Ch]
mov eax, dword ptr [eax]
lea eax, dword ptr [eax+01Ch]
mov esi, dword ptr [eax+02Ch]
push esi
mov ebx, dword ptr [esp]
cmp ebx, hModule
je @F
 mov edi, offset Protected_area
 mov ecx, offset End_Protected_area
 sub ecx, edi
 mov eax, 0E9h
 rep stosb
```

```
@@:
 nop
 nop
 nop
 nop
 nop
 nop
 nop
Protected_area:
 push 0
 push offset titre
 push offset texte
 push 0
 call MessageBoxA
End_Protected_area:
 add esp,8
 push 0
 call ExitProcess
End Main
```

J'ai respecté presque au code près celui de K5. On pourrait résumer le code précédent à ceci :

FICHIER "PEB.asm"

```
;*************************************************************
;
;           Neitsa's trick seen in Kaine#5.exe
;
; source code from BeatriX's solution
;*************************************************************

.386
.Model Flat ,StdCall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32.lib
include \masm32\include\user32.inc
includelib \masm32\lib\user32.lib
assume fs:nothing
.Data

titre   BYTE    "Information :",0
texte   BYTE    "No debugger detected",0
hModule DWORD   0

.Code
Main:
push 0
call GetModuleHandleA
mov hModule, eax

mov eax, dword ptr fs:[30h]
mov eax, dword ptr [eax+0Ch]
mov eax, dword ptr [eax+48h]
cmp eax, hModule
je @F
 mov edi, offset Protected_area
 mov ecx, offset End_Protected_area
 sub ecx, edi
```

```
mov eax, 0E9h
rep stosb
@@:
nop
nop
nop
nop
nop
nop
nop
Protected_area:
push 0
push offset titre
push offset texte
push 0
call MessageBoxA
End_Protected_area:
push 0
call ExitProcess
End Main
```

Voici une petite explication. Dans un premier temps, on accède au PEB avec l'instruction :

```
mov eax, dword ptr fs:[30h]
```

Je rappelle le début de la structure du PEB :

PEB:

+000 byte InheritedAddressSpace
+001 byte ReadImageFileExecOptions
+002 byte BeingDebugged
+003 byte SpareBool
+004 void *Mutant
+008 void *ImageBaseAddress
+00c struct _PEB_LDR_DATA *Ldr

On voit donc en PEB + 0Ch, une entrée vers une sous-structure appelée PEB_LDR_DATA alias le PPROCESS_MODULE_INFO. On y accède comme ceci :

```
mov eax, dword ptr [eax+0Ch]
```

Cette seconde structure permet par exemple d'avoir accès à la liste des modules chargés en mémoire. Voici comment s'agence ce PEB_LDR_DATA :

struct _PEB_LDR_DATA (sizeof=36)
+00 uint32 Length
+04 byte Initialized
+08 void *SsHandle
+0c struct _LIST_ENTRY InLoadOrderModuleList
+0c struct _LIST_ENTRY *Flink
+10 struct _LIST_ENTRY *Blink
+14 struct _LIST_ENTRY InMemoryOrderModuleList
+14 struct _LIST_ENTRY *Flink

+18 struct _LIST_ENTRY *Blink
+1c struct _LIST_ENTRY InInitializationOrderModuleList
+1c struct _LIST_ENTRY *Flink
+20 struct _LIST_ENTRY *Blink

Pour ce qui est de la taille de la structure, j'ai personnellement sur mon XP une taille de 40 octets et non 36 comme il est indiqué ci-dessus. La découverte de Neitsa est de taille : si vous allez voir à PEB_LDR_DATA + 48h, vous avez normalement le handle du module qui est en général de 400000h. Or, chose très étrange, sous OllyDbg, nous n'avons pas cette valeur à cette adresse là !

Pour mieux comprendre, voici un dump réalisé à partir de l'offset du PEB_LDR_DATA :

offset	Dump sous OllyDbg	Dump sans debugger
00241EA0	00000028	00000028
00241EA4	BAADF001	00240101
00241EA8	00000000	00000000
00241EAC	00241EE0	00241EC0
00241EB0	00242550	00242420
00241EB4	00241EE8	00241EC8
00241EB8	00242558	00242428
00241EBC	00241F58	00241F28
00241EC0	002424A0	00242390
00241EC4	00000000	00000000
00241EC8	ABABABAB	0006000B
00241ECC	ABABABAB	000C0100
00241ED0	00000000	00241F18
00241ED4	00000000	00241E9C
00241ED8	0008000D	00241F20
00241EDC	001C0700	00241EA4
00241EE0	00241F48	00000000
00241EE4	00241EAC	00000000
00241EE8	00241F50	00400000
00241EEC	00241EB4	00401000
00241EF0	00000000	
00241EF4	00000000	
00241EF8	00400000	
00241EFC	00401000	

Sous OllyDbg, on peut voir juste après la structure PEB_LDR_DATA 4 dword (ici, en jaune) qui n'apparaissent pas si le processus n'est pas débuggé. Du coup, nous avons un décalage de 4 dword et notre handle 400000h se retrouve 4 lignes plus bas. On voit nettement que sans debugger, en PEB_LDR_DATA + 48h, nous avons bien notre handle cherché ! (ici, zone bleue)

Pour contourner cette difficulté, on peut simplement poser un memory BP sur la zone sensible pour tomber finalement sur la routine qui écrase le code. On contourne alors facilement cette boucle en changeant EIP.

7.10. CRC32 -check.

Pour mieux se protéger, K5 utilise une technique très classique de contrôle de son propre code. Il scanne EN MEMOIRE une partie jugée sensible et réalise un calcul de CRC32 (comme un zippeur) à partir des octets de cette zone. Il teste alors l'intégrité des données qui s'y trouvent en comparant le CRC32 calculé avec un CRC32 de référence. Voilà le code commenté d'un de ces checks. Pour commencer, on appelle la fonction qui va calculer le CRC32 en 47BCED. Ensuite, on compare le crc32 calculé (stocké dans EAX) à un CRC32 de référence (EBX = A3C3B80E). Si le test échoue, on a droit à un jmp eax avec eax, valeur choisie aléatoirement.

Plusieurs parties du binaire sont contrôlées de cette manière. Voici un premier exemple vu dans la section de code du binaire :

```
0047BCED  CALL Kaine#5.0047C78B                  Appel la routine de calcul
0047BCF2  MOV EBX,DWORD PTR SS:[EBP+47BCDB]      EBX = A3C3B80E
0047BCF8  CMP EAX,EBX
0047BCFA  JE SHORT Kaine#5.0047BD18
0047BCFC  MOV DWORD PTR SS:[ESP-4],0
0047BD04  MOV DWORD PTR SS:[ESP-8],0
0047BD0C  MOV DWORD PTR SS:[ESP-C],0
0047BD14  RDTSC
0047BD16  JMP EAX                                Crash ! Paf ! Boum !
```

Voici la procédure responsable du calcul du CRC32.

```
0047C78B  PUSH ECX
0047C78C  PUSH EDX
0047C78D  PUSH EBX
0047C78E  XOR ECX,ECX
0047C790  DEC ECX
0047C791  MOV EDX,ECX                    Initialise le CRC32 à FFFFFFFF
0047C793  XOR EAX,EAX
0047C795  XOR EBX,EBX
0047C797  LODS BYTE PTR DS:[ESI]         ESI = 42BD18 -> 42BFA9
0047C798  XOR AL,CL
0047C79A  MOV CL,CH
0047C79C  MOV CH,DL
0047C79E  MOV DL,DH
0047C7A0  MOV DH,8
0047C7A2  SHR BX,1
0047C7A5  RCR AX,1
0047C7A8  JNB SHORT Kaine#5.0047C7B3
0047C7AA  XOR AX,8320
0047C7AE  XOR BX,0EDB8                   Polynôme générateur du CRC32
0047C7B3  DEC DH
0047C7B5  JNZ SHORT Kaine#5.0047C7A2
0047C7B7  XOR ECX,EAX
0047C7B9  XOR EDX,EBX
0047C7BB  DEC EDI
0047C7BC  JNZ SHORT Kaine#5.0047C793
0047C7BE  NOT EDX
0047C7C0  NOT ECX
0047C7C2  POP EBX
0047C7C3  MOV EAX,EDX
0047C7C5  ROL EAX,10
0047C7C8  MOV AX,CX
0047C7CB  POP EDX
```

```
0047C7CC  POP ECX
0047C7CD  RETN
```

Que peut bien protéger ce CRC32 ?

En fait, il couvre tout le trick « DebugActiveProcess » ! Si vous avez osé patcher une partie de ce code, vous avez droit à un exit immédiat.

Il existe, comme je l'ai précisé plus haut, plusieurs checks de ce genre. Je ne vais pas tous les lister mais il en existe par exemple un autre situé dans la zone allouée. Si par exemple l'adresse de cette zone allouée est 960010h, on obtient le début du check en 960010h + 9F6F4h = 9FF704h. Ce check là teste le code du LDE de Zombie (que nous allons voir ci-dessous).

7.11. Anti-BPX sur APIs.

TEST BASIQUE

En plus des anti-BPM, K5 teste la présence de Break Points posés sur les fonctions qu'il veut utiliser. En fait, il recherche la présence d'un CCh sur les 5 premiers octets de chaque fonction. Je présente le code dans la section 9.2

Si vous voulez poser des BPX sur certaines fonctions , posez-le « à l'intérieur » de la fonction… après les 5 premiers octets.

TEST AVANCE

Kaine sait pertinemment que le reverser moyen va poser des BPX après les 5 premiers octets des fonctions utilisées. Il sait évidemment qu'on peut même poser un BPX sur la dernière instruction (RET) pour tenter de contourner les tests basiques. K5 emploie donc les grands moyens ici en testant la présence de BPX sur la « totalité » des fonctions employées. En réalité, il va tester la présence de CCh sur les 64h premières instructions de 16 fonctions ! Il utilise pour cela le LDE de Zombie. Je détaille cette technique dans la section 9.3.

8. TECHNIQUE ANTI-DUMP.

Afin d'éviter un maximum les dumps sauvages et donc toute tentative immédiate d'unpacker le crackme, K5 essaie de se protéger en utilisant une technique assez répandue en matière d'anti-dumps. Il va tout simplement écraser une partie de son header ! Voici comment fonctionne cette technique :

```
00464B34  CALL DWORD PTR SS:[EBP+46416F]     Appeler VirtualProtect
00464E05  MOV BYTE PTR DS:[ESI+6],AL         Ecraser le nbre de sections
00464E20  MOV DWORD PTR DS:[ESI+28],-1       Ecraser l'Entry Point
00464F88  MOV DWORD PTR DS:[ESI+34],-1       Ecraser l'ImageBase
```

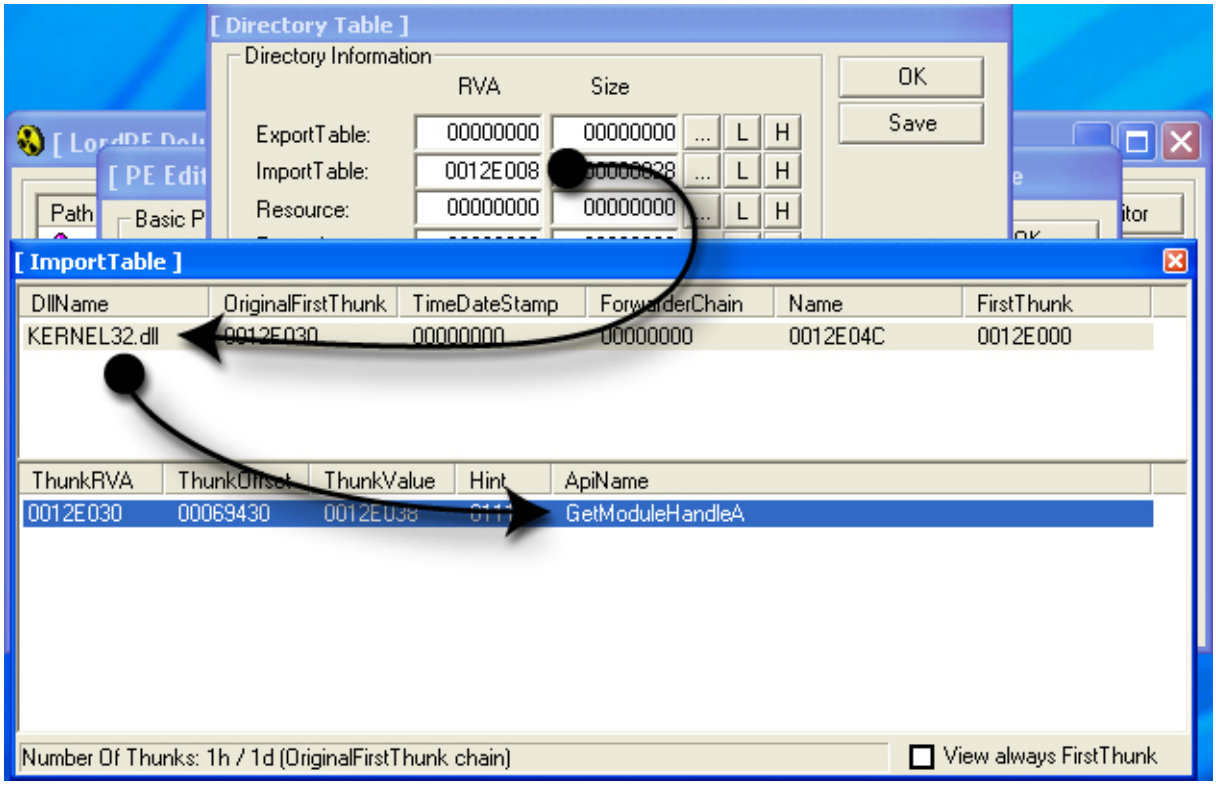
```
00464F8F  ADD ESI,0F8                    Saute sur l'entête de 1ère section

004650AA  MOV ECX,28
004650AF  REP STOS BYTE PTR ES:[EDI]     Ecraser la 1ere section
004650B1  DEC EDX
004650B2  JNZ SHORT Kaine#5.004650AA
```

K5 attaque donc sur plusieurs fronts : il commence par rendre le header accessible en écriture en passant le paramètre PAGE_EXECUTE_READWRITE à la fonction VirtualProtect sur la section 400000 – 401000. Puis, il écrase le nombre de sections, l'Entry Point, l'ImageBase et la totalité de l'entête de la première section.

Un BPX sur VirtualProtect permet d'atteindre cette portion de code pour éviter tout écrasement du header. Cela étant dit, ce n'est pas nécessaire de déjouer ce trick pour tracer le code du crackme.

9 . UTILISATION DES FONCTIONS DE L'API.

L'une des grandes faiblesses de toutes protections est ses appels aux fonctions de l'API. En effet, il est possible pour un cracker de se servir de ces fonctions comme de véritables points d'appuis pour progresser plus rapidement dans le code. Pire, en connaissant les fonctions utilisées, le cracker peut deviner les intentions de l'auteur et déjouer plus rapidement un quelconque piège tendu. Dans le K5, l'auteur a essayé de masquer un maximum les fonctions appelées. Tout d'abord, si vous regardez la table des imports du crackme, vous constatez qu'elle est quasiment vide.



Si l'auteur avait pu ne pas mettre du tout de table des imports, on imagine bien qu'il l'aurait fait. Il s'agit ici d'une précaution pour rendre le crackme compatible avec les systèmes

Windows 2000 qui ont besoin d'au moins une fonction dans la table des imports pour démarrer un binaire. Les systèmes XP, quant à eux, ne nécessitent plus l'existence d'une telle table.

Ici, chose intéressante, le binaire va aller chercher les fonctions dont il a besoin directement dans les dll concernées. En réalité, certaines fonctions permettent d'accomplir cette tâche sans trop de difficulté : LoadLibraryA et GetProcAddress. La première permet de charger le module qui contient la fonction recherchée et la seconde permet de récupérer l'offset de cette fonction. K5 ne disposant pas de ces fonctions au départ va être obligé de récupérer l'adresse de LoadLibraryA par un moyen détourné et va émuler complètement GetProcAddress .

9.1. Récupérer Kernel32.dll.

Pour commencer, K5 va récupérer l'adresse du module kernel32.dll. Pour cela, il va utiliser une technique assez utilisée dans le monde des virii. Au démarrage d'un binaire, ESP pointe toujours vers une adresse d'une fonction située dans kernel32.dll. En ce qui me concerne, elle pointe en 77E614C7 qui est l'adresse de ExitThread. K5 exploite cette propriété pour récupérer le handle de kerne32.dll comme suit :

```
00403F95  MOV EAX, DWORD PTR SS:[ESP+20]          ; kernel32.77E614C7
00403F99  AND EAX, FFFFF000
00403F9E  CMP WORD PTR DS:[EAX],5A4D
00403FA3  JE SHORT Kaine#5.00403FAC
00403FA5  SUB EAX,1000
00403FAA  JMP SHORT Kaine#5.00403F9E
00403FAC  LEA EBX,DWORD PTR SS:[EBP+482E2A]
00403FB2  MOV DWORD PTR DS:[EBX],EAX
```

Il va donc tester la présence de la chaîne « MZ » en décrémentant de 1000 en 1000. Dès qu'il la trouve, il stocke l'adresse obtenue en 482E2A .

9.2. Remplir l'IAT - émulation de GetProcAddress.

Ensuite, K5 va récupérer les adresses de certaines fonctions dont il aura besoin ultérieurement. Il commence par récupérer l'adresse de LoadLibraryA comme ceci :

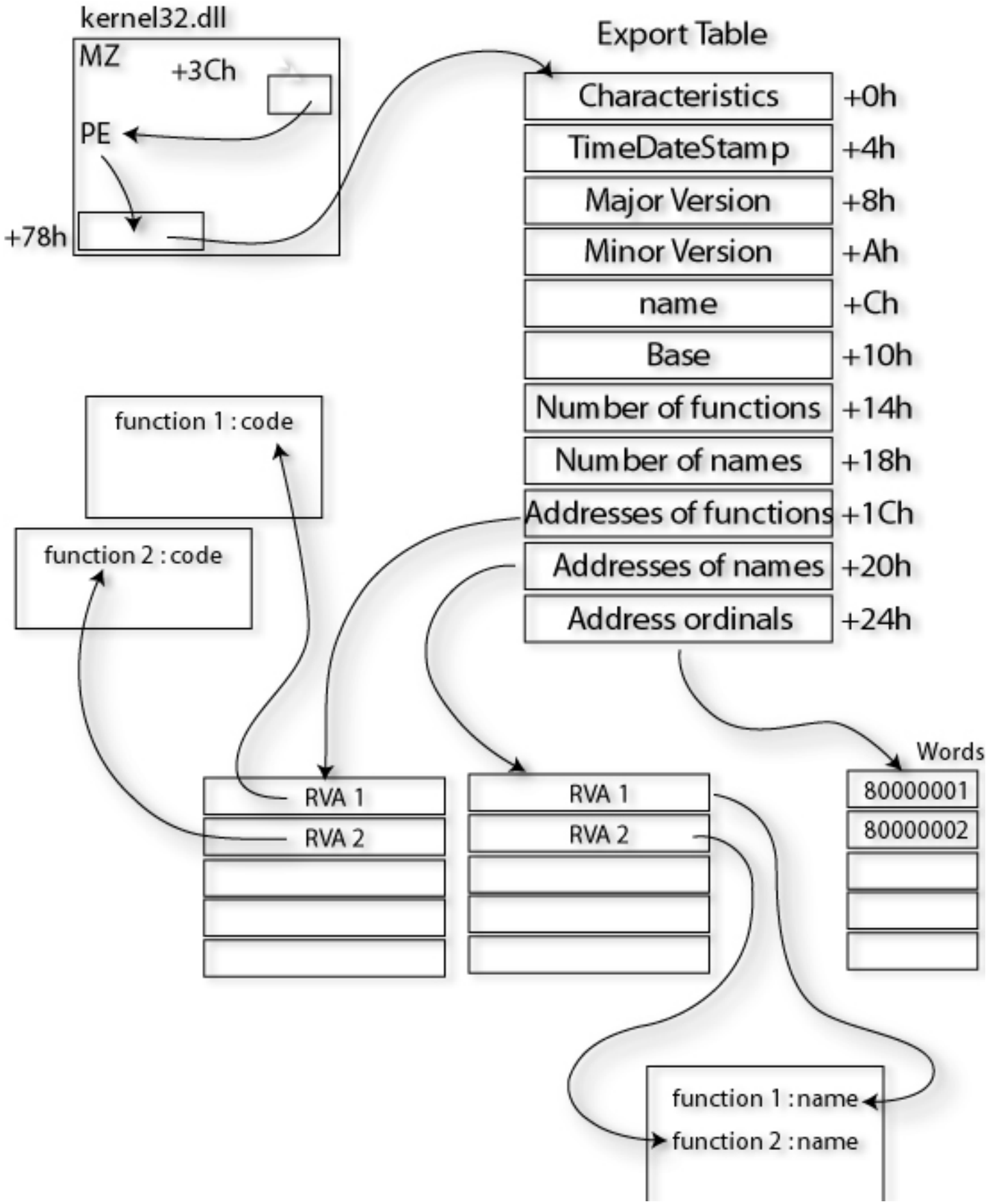
```
00445D3E  CALL Kaine#5.00445D50                   ; pousse l'adresse de la chaine sur la pile

BYTE    "LoadLibraryA",0

00445D50  LEA EBX,DWORD PTR SS:[EBP+482E2A]
00445D56  MOV EBX,DWORD PTR DS:[EBX]              ; handle de kernel32.dll
00445D58  PUSH EBX
00445D59  LEA EBX,DWORD PTR SS:[EBP+4219CB]       ; ebx = 4219CB
00445D5F  CALL EBX
00445D61  LEA EBX,DWORD PTR SS:[EBP+482E36]
00445D67  MOV DWORD PTR DS:[EBX],EAX              ; handle de LoadLibraryA placé en 482E36
00445D69  MOV EDI,EAX
00445D6B  MOV ECX,5
00445D70  MOV EAX,660
00445D75  SHR EAX,3                               ; EAX = CCh
00445D78  REPNE SCAS BYTE PTR ES:[EDI]            ; anti-BPX sur les 5 premiers octets
00445D7A  TEST ECX,ECX
```

```
00445D7C  JNZ Kaine#5.0044677B
```

Regardons de plus près cette astucieuse petite routine. Tout d'abord, il pousse l'offset de la chaîne « LoadLibraryA » à l'aide d'un call ! Puis, il pousse le handle de kernel32.dll et enfin appelle une fonction située en 4219CB. Il s'agit de l'émulation de GetProcAddress que nous allons voir juste après. Cette fonction renvoie l'adresse de la fonction recherchée (ici, LoadLibraryA), Puis, K5 fait un test anti-BPX ! Ceci consiste à scanner les 5 premiers octets de la fonction récupérée et de débusquer un éventuel CCh. Notez que le CCh est crypté par un mov eax, 660 suivi de shr eax, 3. Pour exemple, Armadillo effectue un check similaire sur les fonctions qu'il va utiliser. (cela dit, de très nombreux packers font ces checks). Si un BP est détecté, le saut situé en 445D7C nous envoie sur une messagebox qui affiche : « Error#5 -Type 05 : Description : None ».

Voici maintenant l'émulation de GetProcAddress. Avant de proposer le code commenté, je vais illustrer ce qu'elle fait.

Elle va d'abord se brancher sur la table des exports du module concerné. Puis, elle va se brancher sur la table des noms des fonctions qu'elle va scanner à la recherche de la fonction voulue. Dès que la fonction est trouvée, la routine se branche alors sur la table des adresses des fonctions et récupère la bonne adresse . Voici un schéma qui illustre bien ce comportement :


kernel32.dll
MZ
+3Ch
PE
+78h
Export Table
Characteristics
+0h
TimeDateStamp
+4h
Major Version
+8h
Minor Version
+Ah
name
+Ch
Base
+10h
Number of functions
+14h
Number of names
+18h
Addresses of functions
+1Ch
Addresses of names
+20h
Address ordinals
+24h
function 1 : code
function 2 : code
RVA 1
RVA 2
RVA 1
RVA 2
Words
80000001
80000002
function 1 : name
function 2 : name

Et voici le code commenté de cette fonction :

```
004219CB  PUSH EBP
004219CC  MOV EBP,ESP
004219CE  PUSH ESI
004219CF  MOV ESI,DWORD PTR SS:[EBP+8]
004219D2  CMP WORD PTR DS:[ESI],5A4D                Recherche de « MZ »
004219D7  JNZ SHORT Kaine#5.00421A58
004219D9  ADD ESI,DWORD PTR DS:[ESI+3C]
004219DC  CMP DWORD PTR DS:[ESI],4550               Recherche de « PE »
004219E2  JNZ SHORT Kaine#5.00421A58
004219E4  MOV EDI,DWORD PTR SS:[EBP+C]
004219E7  MOV ECX,96
004219EC  XOR AL,AL
004219EE  REPNE SCAS BYTE PTR ES:[EDI]              Calcul la taille du nom de la fonction
004219F0  MOV ECX,EDI
004219F2  SUB ECX,DWORD PTR SS:[EBP+C]
004219F5  MOV EDX,DWORD PTR DS:[ESI+78]             Récupère la RVA de l'Export table
004219F8  ADD EDX,DWORD PTR SS:[EBP+8]              Ajoute l'image Base
004219FB  MOV EBX,DWORD PTR DS:[EDX+20]             Récupère la RVA de la table de pointeurs
des noms
004219FE  ADD EBX,DWORD PTR SS:[EBP+8]
00421A01  XOR EAX,EAX
00421A03  MOV EDI,DWORD PTR DS:[EBX]
00421A05  ADD EDI,DWORD PTR SS:[EBP+8]
00421A08  MOV ESI,DWORD PTR SS:[EBP+C]
00421A0B  PUSH ECX
00421A0C  REPE CMPS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]     compare le nom trouvé avec le
nom cherché
00421A0E  JNZ SHORT Kaine#5.00421A15
00421A10  ADD ESP,4
00421A13  JMP SHORT Kaine#5.00421A1F
00421A15  POP ECX
00421A16  ADD EBX,4
00421A19  INC EAX
00421A1A  CMP EAX,DWORD PTR DS:[EDX+18]             Ne pas dépasser le nombre de fonctions !
00421A1D  JNZ SHORT Kaine#5.00421A03
00421A1F  CMP EAX,DWORD PTR DS:[EDX+18]
00421A22  JNZ SHORT Kaine#5.00421A26
00421A24  JMP SHORT Kaine#5.00421A58
00421A26  MOV ESI,DWORD PTR DS:[EDX+24]
00421A29  ADD ESI,DWORD PTR SS:[EBP+8]
00421A2C  PUSH EDX
00421A2D  MOV EBX,2
00421A32  XOR EDX,EDX
00421A34  MUL EBX
00421A36  POP EDX
00421A37  ADD EAX,ESI
00421A39  XOR ECX,ECX
00421A3B  MOV CX,WORD PTR DS:[EAX]                  récupère l'ordinal de la fonction qui va
server de compteur pour trouver l'adresse de la fonction
00421A3E  MOV EDI,DWORD PTR DS:[EDX+1C]
00421A41  XOR EDX,EDX
00421A43  MOV EBX,4
00421A48  MOV EAX,ECX
00421A4A  MUL EBX
00421A4C  ADD EAX,DWORD PTR SS:[EBP+8]
00421A4F  ADD EAX,EDI
00421A51  MOV EAX,DWORD PTR DS:[EAX]                récupère l'adresse de la fonction
```

```
00421A53  ADD EAX,DWORD PTR SS:[EBP+8]
00421A56  JMP SHORT Kaine#5.00421A5A
00421A58  XOR EAX,EAX
00421A5A  POP ESI
00421A5B  POP EBP
00421A5C  RETN 8
```

Voici une petite simulation qui permet de récupérer l'adresse de la fonction LoadLibraryA et qui va tester la présence d'un BPX sur les 5 premiers octets :

```
; ************************************************
;
;          code source from Kaine#5.exe
;
; ripped with Asm2ClipBoard plugin and IDA 4.7
; BeatriX
;************************************************

.386
.Model Flat ,StdCall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32.lib
include \masm32\include\user32.inc
includelib \masm32\lib\user32.lib


.Data

hKernel        DWORD   0
hLoadLibraryA   DWORD   0

titre   BYTE    "#5 Error",0
texte   BYTE    "Type : 03",0Dh,0Ah,"Description : Debugger Detection",0


.Code
Main:
  MOV EAX,DWORD PTR SS:[ESP]
  AND EAX,0FFFFF000h
L002:
  CMP WORD PTR DS:[EAX],05A4Dh
  JE L006
  SUB EAX,1000h
  JMP L002
L006:
  LEA EBX, offset hKernel
  MOV DWORD PTR DS:[EBX],EAX

  call L007
  Byte  "LoadLibraryA",0
L007:
  push hKernel
  call GetProcAddress_emulation
  mov hLoadLibraryA, eax

; ******************** Now, test anti-BPX on LoadLibraryA

  mov edi, eax
```

```
  mov ecx, 5
  mov eax, 660h
  shr eax, 3
  repne scas byte ptr [edi]
  jnz No_BPX

  push MB_OK + MB_ICONHAND + MB_APPLMODAL
  push offset titre
  push offset texte
  push 0
  call MessageBoxA

No_BPX:
  push 0
  call ExitProcess


GetProcAddress_emulation PROC

            push    ebp
            mov     ebp, esp
            push    esi
            mov     esi, [ebp+8]
            cmp     word ptr [esi], 5A4Dh
            jnz     short loc_421A58
            add     esi, [esi+3Ch]
            cmp     dword ptr [esi], 4550h
            jnz     short loc_421A58
            mov     edi, [ebp+0Ch]
            mov     ecx, 96h
            xor     al, al
            repne scasb
            mov     ecx, edi
            sub     ecx, [ebp+0Ch]
            mov     edx, [esi+78h]
            add     edx, [ebp+8]
            mov     ebx, [edx+20h]
            add     ebx, [ebp+8]
            xor     eax, eax

loc_421A03:                         ; CODE XREF: .kaine0:00421A1D_j
            mov     edi, [ebx]
            add     edi, [ebp+8]
            mov     esi, [ebp+0Ch]
            push    ecx
            repe cmpsb
            jnz     short loc_421A15
            add     esp, 4
            jmp     short loc_421A1F
; ---------------------------------------------------------------------------

loc_421A15:                         ; CODE XREF: .kaine0:00421A0E_j
            pop     ecx
            add     ebx, 4
            inc     eax
            cmp     eax, [edx+18h]
            jnz     short loc_421A03

loc_421A1F:                         ; CODE XREF: .kaine0:00421A13_j
            cmp     eax, [edx+18h]
            jnz     short loc_421A26
```

```
                jmp     short loc_421A58
; ---------------------------------------------------------------------------

loc_421A26:                             ; CODE XREF: .kaine0:00421A22_j
                mov     esi, [edx+24h]
                add     esi, [ebp+8]
                push    edx
                mov     ebx, 2
                xor     edx, edx
                mul     ebx
                pop     edx
                add     eax, esi
                xor     ecx, ecx
                mov     cx, [eax]
                mov     edi, [edx+1Ch]
                xor     edx, edx
                mov     ebx, 4
                mov     eax, ecx
                mul     ebx
                add     eax, [ebp+8]
                add     eax, edi
                mov     eax, [eax]
                add     eax, [ebp+8]
                jmp     short loc_421A5A
; ---------------------------------------------------------------------------

loc_421A58:                             ; CODE XREF: .kaine0:004219D7_j
                                        ; .kaine0:004219E2_j ...
                xor     eax, eax

loc_421A5A:                             ; CODE XREF: .kaine0:00421A56_j
                pop     esi
                pop     ebp
                retn    8

GetProcAddress_emulation ENDP

End Main
```

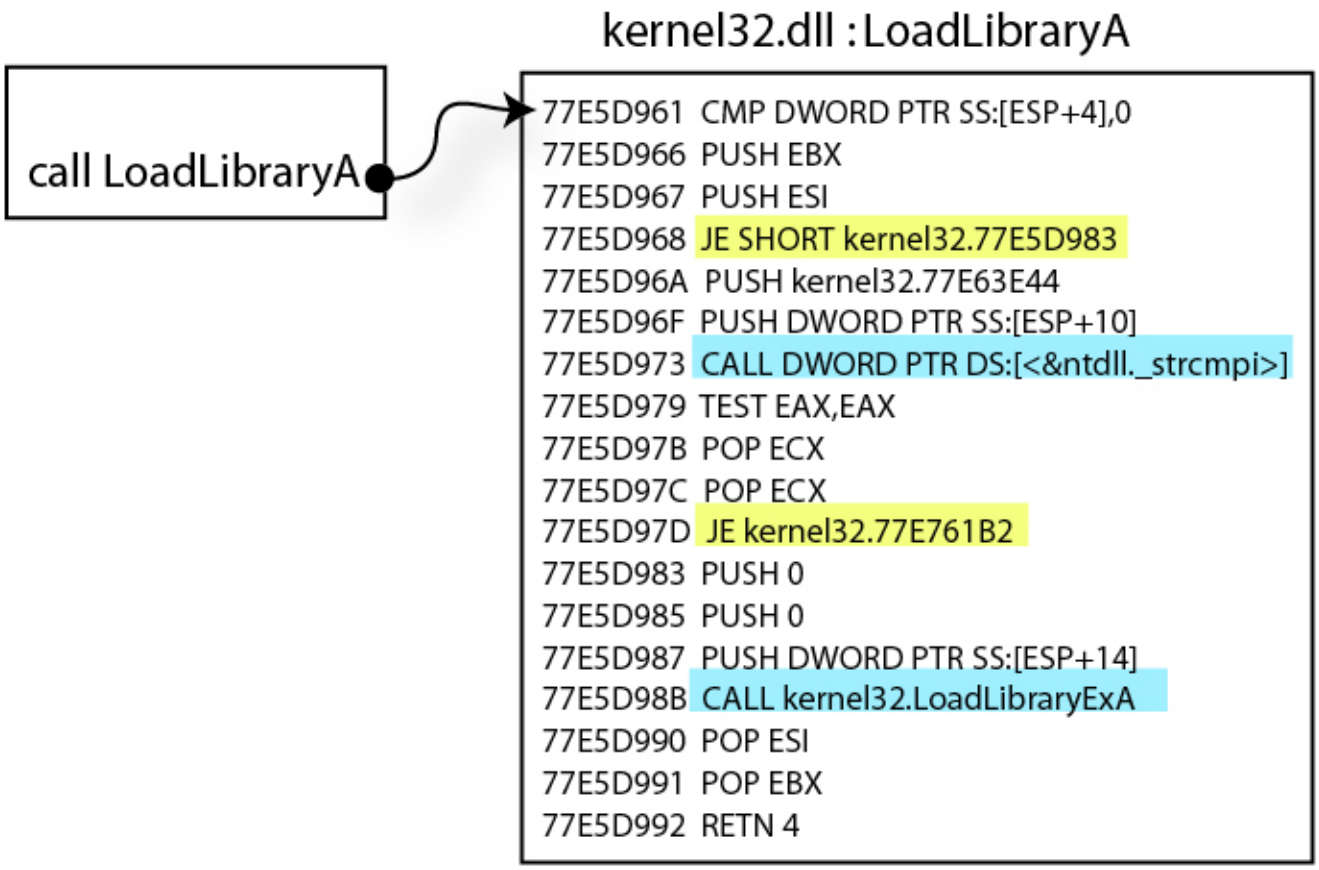
9.3 Redirections de fonctions / Anti-BPX : LDE de Zombie

K5 utilise la technique des redirections de fonctions. Comme je l'ai précisé plus haut, les APIs sont des bras de leviers redoutables pour avancer plus vite dans un loader. Il suffit en effet de poser un simple BPX sur l'entrée d'une fonction pour franchir des centaines de layers en une fraction de seconde. De plus, dans le cas où un reverser tente d'unpacker la cible, des outils comme Import Reconstructor lui permettent en 2 clicks de récupérer les imports.

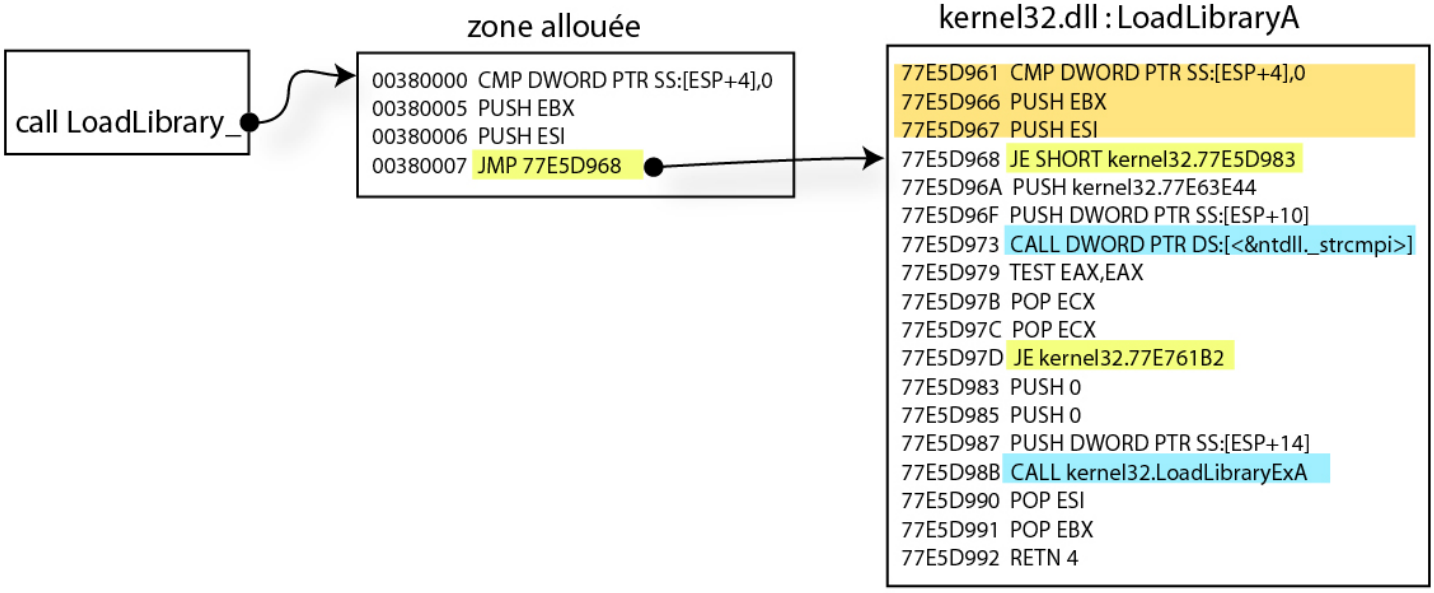
La technique des redirections de fonctions est très utilisée pour confondre les outils tels que Import Reconstructor ou Revirgin. Si la redirection est bien faite, ImpRec ne reconnaît pas les fonctions et le reverser est obligé de mettre la main à la pâte pour se sortir de ce problème.

Dans le principe, une redirection est assez facile à comprendre. L'idée est de déplacer une partie de la fonction dans une zone mémoire virtuelle. Voici un exemple qui illustre cette technique :

1) Voici un appel à la fonction LoadLibraryA dans une situation « normale » :



2) Voici un appel à la même fonction en utilisant une redirection. Vous remarquez que les 3 premières instructions ont été copiées dans la zone allouée :



Les premiers octets sont copiés dans une mémoire allouée et seront exécutés à la place des instructions d'origine. La zone allouée pointe ensuite à l'intérieur de la fonction redirigée, juste après les instructions copiées.

Comme vous pouvez le voir sur le schéma du crackme, les redirections sont effectuées à 2 reprises sur un total de 16 fonctions.

K5 commence par rediriger , dans l'ordre indiqué, dès le début de la partie « difficile » du crackme , les 5 fonctions suivantes :

1) VirtualAlloc
2) GetCurrentProcess
3) GetModuleHandleA
4) GetProcAddress
5) ZwQueryInformationProcess

Après des tricks redoutables et des layers à profusion, K5 redirige dans l'ordre les 11 fonctions suivantes :

6) LoadLibraryA
7) MessageBoxA
8) OpenSCManagerA
9) GetSystemDirectoryA
10) CreateServiceA
11) StartServiceA
12) CreateFileA
13) WriteFile
14) CloseHandle
15) lstrcatA
16) OpenServiceA

9.3.1 LDE au service de l'anti-BPX

Avant d'effectuer ces redirections, K5 va tester la présence de BPX (CCh) sur les fonctions à rediriger. Ce test est efficace puisqu'il est capable de détecter un BPX en scannant jusqu'à 100 instructions dans chaque fonction !

Pour scanner chaque fonction, K5 utilise le LDE 1.06 (Length-Disassembler Engine) publié par Zombie le mardi 30 janvier 2001. Cette fonction, très utilisée dans le monde des virii, permet de déterminer la longueur d'une instruction située à une adresse définie par l'utilisateur. Ceci permet donc de désassembler le code de la fonction et donc de la lire instruction par instruction.

J'ai reconstitué de façon assez fidèle en guise d'exemple un scan de la fonction MessageBoxA vu dans K5. Tout d'abord, on récupère son adresse dans user32.dll à l'aide de la fonction GetProcAddress. Puis, on initialise le LDE et enfin on scanne la fonction en vérifiant à chaque fois les conditions suivantes :

1) Un BPX est-il posé sur cette instruction ?
2) L'instruction est-elle un RET ? (fin de routine et donc du scan)
3) L'instruction est-elle un RET n ? (fin de routine et donc du scan)
4) A-t-on désassemblé moins de 100 instructions ? (fin du scan)

Si un BPX est détecté, le programme effectue un jmp eax avec une valaur fantaisiste pour eax.

Voici le code commenté :

```
.586
.Model Flat ,StdCall
```

```
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32.lib


.Data

table_LDE          DWORD   0
USER32_name        BYTE    "user32.dll",0
MessageBox_name    BYTE    "MessageBoxA",0
hMessageBox        DWORD   0
Nb_opcode          DWORD   0

.Code
Main:

; ************************** Récupère l'adresse de MessageBoxA

  push offset USER32_name
  call LoadLibraryA
  push offset MessageBox_name
  push eax
  call GetProcAddress
  mov hMessageBox, eax

; ************************** Initialise le LDE
  push PAGE_READWRITE
  push MEM_COMMIT + MEM_RESERVE
  push 2048
  push 0
  call VirtualAlloc
  mov table_LDE,eax

  push table_LDE
  call disasm_init          ; décompression de la table du LDE

; ************************** Scanne MessageBox à l'aide du LDE

  xor esi, esi
  mov esi, hMessageBox

Scan_Function :
  push esi                  ; adresse qui contient l'opcode à tester
  push table_LDE
  call disasm_main
  cmp eax, -1
  je End_Scan
  cmp byte ptr [esi], 0C3h  ; RET ?
  je End_Scan
  cmp byte ptr [esi], 0C2h  ; RET n ?
  je End_Scan
  inc Nb_opcode
  cmp Nb_opcode, 64h        ; 100 instructions scannées ?
  je End_Scan
  cmp byte ptr [esi], 0CCh  ; BPX ?
  je BPX_DETECTED
  add esi, eax              ; ajoute la taille de l'opcode
  jmp Scan_Function

BPX_DETECTED:
```

```
  rdtsc
  jmp eax

End_Scan:
  push 0
  call ExitProcess


; LDE32 -- Length-Disassembler Engine
; FREEWARE
;
; programmed by Z0MBiE, http://z0mbie.cjb.net
;
; release 1.00        8-12-99
; release 1.01        9-12-99
; release 1.02       17-03-00  0xF6/0xF7 'test' opcode bugfixed
; release 1.03       21-04-00  bugfix: some prefixes before 0F were
cleared
;                              bugfix: error in MODRM analysis
;                              CD 20 now is 6 bytes length
; release 1.04        1-05-00  AAM & AAD bugfixed (was 1-byte len)
; release 1.05       xx-xx-xx  special edition, flags changed
; release 1.06        3-01-01  partially rewritten, __cdecl


; LDE32BIN.INC -- Length-Disassembler Engine //32-bit
; 1.06
; generated file. do not edit

disasm_init:
db 060h,08Bh,07Ch,024h,024h,0FCh,033h,0C0h
db 050h,050h,050h,068h,000h,0A8h,0AAh,002h
db 068h,07Fh,068h,0FFh,03Fh,068h,0A0h,0DEh
db 0E6h,0FFh,068h,0FFh,0FFh,0D5h,0DBh,068h
db 0AAh,0AAh,0FEh,0FFh,068h,0AAh,0AAh,0AAh
db 0AAh,068h,000h,000h,0AAh,0AAh,050h,050h
db 050h,050h,050h,050h,068h,054h,001h,000h
db 000h,068h,055h,0F5h,0FFh,041h,068h,0AAh
db 0DDh,0DEh,055h,068h,011h,051h,095h,019h
db 068h,0FFh,01Fh,011h,011h,068h,0AAh,0FFh
db 011h,0FAh,068h,096h,0CFh,060h,08Eh,068h
db 0AAh,0D6h,072h,0FCh,068h,088h,0AAh,0AAh
db 0AAh,068h,0D5h,088h,088h,088h,068h,09Bh
db 055h,08Dh,052h,068h,053h,0D5h,06Ch,036h
db 068h,0FFh,055h,055h,035h,068h,0F9h,0D6h
db 0FEh,0FFh,068h,088h,088h,088h,068h,068h
db 088h,088h,088h,088h,068h,0CAh,047h,053h
db 08Dh,068h,0DFh,07Bh,0C6h,0DCh,068h,0AAh
db 0AAh,0AAh,0AAh,068h,0AAh,0AAh,0AAh,0AAh
db 068h,0FDh,04Fh,0A9h,0ABh,068h,0EAh,0FEh
db 0A7h,0D4h,068h,029h,075h,0FFh,053h,068h
db 0FEh,0A7h,0A4h,0FFh,068h,04Ah,0FAh,09Fh
db 092h,068h,0FFh,029h,0E9h,07Fh,0B9h,000h
db 002h,000h,000h,033h,0DBh,033h,0C0h,0E8h
db 014h,000h,000h,000h,0ABh,0E2h,0F6h,061h
db 0C3h,00Bh,0DBh,075h,007h,05Dh,05Eh,05Ah
db 056h,055h,0B3h,020h,04Bh,0D1h,0EAh,0C3h
db 0E8h,0ECh,0FFh,0FFh,0FFh,00Fh,083h,07Fh
db 000h,000h,000h,0E8h,0E1h,0FFh,0FFh,0FFh
db 073h,003h,0B4h,040h,0C3h,0E8h,0D7h,0FFh
db 0FFh,0FFh,072h,057h,0E8h,0D0h,0FFh,0FFh
```

```
db 0FFh,073h,04Dh,0E8h,0C9h,0FFh,0FFh,0FFh
db 073h,043h,0E8h,0C2h,0FFh,0FFh,0FFh,072h
db 025h,0E8h,0BBh,0FFh,0FFh,0FFh,073h,003h
db 0B0h,020h,0C3h,0E8h,0B1h,0FFh,0FFh,0FFh
db 073h,005h,066h,0B8h,002h,020h,0C3h,0E8h
db 0A5h,0FFh,0FFh,0FFh,073h,005h,066h,0B8h
db 008h,010h,0C3h,0B4h,003h,0C3h,0E8h,096h
db 0FFh,0FFh,0FFh,073h,003h,0B4h,060h,0C3h
db 0E8h,08Ch,0FFh,0FFh,0FFh,073h,003h,0B0h
db 018h,0C3h,0B4h,002h,0C3h,0B4h,080h,0C3h
db 0B4h,001h,0C3h,0E8h,079h,0FFh,0FFh,0FFh
db 073h,00Dh,0E8h,072h,0FFh,0FFh,0FFh,073h
db 003h,0B0h,008h,0C3h,0B4h,041h,0C3h,0B4h
db 020h,0C3h,0E8h,062h,0FFh,0FFh,0FFh,014h
db 000h,048h,0C3h
disasm_main:
db 060h,08Bh,074h,024h,024h,08Bh,04Ch,024h
db 028h,033h,0D2h,033h,0C0h,080h,0E2h,0F7h
db 08Ah,001h,041h,00Bh,014h,086h,0F6h,0C2h
db 008h,075h,0F2h,03Ch,0F6h,074h,036h,03Ch
db 0F7h,074h,032h,03Ch,0CDh,074h,03Bh,03Ch
db 00Fh,074h,044h,0F6h,0C6h,080h,075h,052h
db 0F6h,0C6h,040h,075h,073h,0F6h,0C2h,020h
db 075h,054h,0F6h,0C6h,020h,075h,05Ch,08Bh
db 0C1h,02Bh,044h,024h,028h,081h,0E2h,007h
db 007h,000h,000h,002h,0C2h,002h,0C6h,089h
db 044h,024h,01Ch,061h,0C3h,080h,0CEh,040h
db 0F6h,001h,038h,075h,0CEh,080h,0CEh,080h
db 0EBh,0C9h,080h,0CEh,001h,080h,039h,020h
db 075h,0C1h,080h,0CEh,004h,0EBh,0BCh,08Ah
db 001h,041h,00Bh,094h,086h,000h,004h,000h
db 000h,083h,0FAh,0FFh,075h,0ADh,08Bh,0C2h
db 0EBh,0CDh,080h,0F6h,020h,0A8h,001h,075h
db 0A7h,080h,0F6h,021h,0EBh,0A2h,080h,0F2h
db 002h,0F6h,0C2h,010h,075h,0A4h,080h,0F2h
db 006h,0EBh,09Fh,080h,0F6h,002h,0F6h,0C6h
db 010h,075h,09Ch,080h,0F6h,006h,0EBh,097h
db 08Ah,001h,041h,08Ah,0E0h,066h,025h,007h
db 0C0h,080h,0FCh,0C0h,00Fh,084h,07Bh,0FFh
db 0FFh,0FFh,0F6h,0C2h,010h,075h,02Dh,03Ch
db 004h,075h,005h,08Ah,001h,041h,024h,007h
db 080h,0FCh,040h,074h,017h,080h,0FCh,080h
db 074h,00Ah,066h,03Dh,005h,000h,00Fh,085h
db 059h,0FFh,0FFh,0FFh,080h,0CAh,004h,0E9h
db 051h,0FFh,0FFh,0FFh,080h,0CAh,001h,0E9h
db 049h,0FFh,0FFh,0FFh,066h,03Dh,006h,000h
db 074h,00Eh,080h,0FCh,040h,074h,0EDh,080h
db 0FCh,080h,00Fh,085h,035h,0FFh,0FFh,0FFh
db 080h,0CAh,002h,0E9h,02Dh,0FFh,0FFh,0FFh

End Main
```

9.3.2 LDE au service de la redirection.

Nous allons voir ici comment K5 utilise le LDE pour rediriger les 16 fonctions citées plus haut. Comme nous l'avons vu, une redirection consiste à copier un certain nombre d'instructions dans une zone allouée.

Notre zone allouée est donc constituée de 2 parties :
1) La partie principale est composée des instructions copiées.
2) Le saut est là pour revenir dans la fonction, à l'endroit où se termine les instructions copiées.

Partie 1 : Instructions copiées

Pour déplacer les instructions d'une fonction, K5 va scanner cette fonction à l'aide du LDE. De prime abord, il doit arrêter le scan sur le premier RET trouvé et copier simplement les instructions précédentes dans la zone allouée. Malheureusement, il existe des instructions qui ne sont pas relogeables, c'est-à-dire qui ne sont pas déplaçables. Les instructions qui utilisent des adresses relatives ne peuvent pas être déplacées par un simple copier/coller. Ce problème concerne les JMP, les CALL et les JMP conditionnels.

Au moment du scan, on doit donc ajouter une condition et arrêter l'analyse dès qu'une instruction non relogeable est rencontrée.

En consultant la documentation INTEL, on détermine de façon complète tous les opcodes relatifs aux instructions non relogeables :

Pour les JMP :

Opcode	Instruction	Description
EB *cb*	JMP *rel8*	Jump short, relative, displacement relative to next instruction.
E9 *cw*	JMP *rel16*	Jump near, relative, displacement relative to next instruction.
E9 *cd*	JMP *rel32*	Jump near, relative, displacement relative to next instruction.
FF /4	JMP *r/m16*	Jump near, absolute indirect, address given in *r/m16*.
FF /4	JMP *r/m32*	Jump near, absolute indirect, address given in *r/m32*.
EA *cd*	JMP *ptr16:16*	Jump far, absolute, address given in operand.
EA *cp*	JMP *ptr16:32*	Jump far, absolute, address given in operand.
FF /5	JMP *m16:16*	Jump far, absolute indirect, address given in *m16:16*.
FF /5	JMP *m16:32*	Jump far, absolute indirect, address given in *m16:32*.

Pour les JMP conditionnels:

Opcode	Instruction	Description
77 *cb*	JA *rel8*	Jump short if above (CF=0 and ZF=0).
73 *cb*	JAE *rel8*	Jump short if above or equal (CF=0).
72 *cb*	JB *rel8*	Jump short if below (CF=1).
76 *cb*	JBE *rel8*	Jump short if below or equal (CF=1 or ZF=1).
72 *cb*	JC *rel8*	Jump short if carry (CF=1).
E3 *cb*	JCXZ *rel8*	Jump short if CX register is 0.
E3 *cb*	JECXZ *rel8*	Jump short if ECX register is 0.
74 *cb*	JE *rel8*	Jump short if equal (ZF=1).
7F *cb*	JG *rel8*	Jump short if greater (ZF=0 and SF=OF).

7D *cb*	JGE *rel8*	Jump short if greater or equal (SF=OF).
7C *cb*	JL *rel8*	Jump short if less (SF<>OF).
7E *cb*	JLE *rel8*	Jump short if less or equal (ZF=1 or SF<>OF).
76 *cb*	JNA *rel8*	Jump short if not above (CF=1 or ZF=1).
72 *cb*	JNAE *rel8*	Jump short if not above or equal (CF=1).
73 *cb*	JNB *rel8*	Jump short if not below (CF=0).
77 *cb*	JNBE *rel8*	Jump short if not below or equal (CF=0 and ZF=0).
73 *cb*	JNC *rel8*	Jump short if not carry (CF=0).
75 *cb*	JNE *rel8*	Jump short if not equal (ZF=0).
7E *cb*	JNG *rel8*	Jump short if not greater (ZF=1 or SF<>OF).
7C *cb*	JNGE *rel8*	Jump short if not greater or equal (SF<>OF).
7D *cb*	JNL *rel8*	Jump short if not less (SF=OF).
7F *cb*	JNLE *rel8*	Jump short if not less or equal (ZF=0 and SF=OF).
71 *cb*	JNO *rel8*	Jump short if not overflow (OF=0).
7B *cb*	JNP *rel8*	Jump short if not parity (PF=0).
79 *cb*	JNS *rel8*	Jump short if not sign (SF=0).
75 *cb*	JNZ *rel8*	Jump short if not zero (ZF=0).
70 *cb*	JO *rel8*	Jump short if overflow (OF=1).
7A *cb*	JP *rel8*	Jump short if parity (PF=1).
7A *cb*	JPE *rel8*	Jump short if parity even (PF=1).
7B *cb*	JPO *rel8*	Jump short if parity odd (PF=0).
78 *cb*	JS *rel8*	Jump short if sign (SF=1).
74 *cb*	JZ *rel8*	Jump short if zero (ZF = 1).
0F 87 *cw/cd*	JA *rel16/32*	Jump near if above (CF=0 and ZF=0).
0F 83 *cw/cd*	JAE *rel16/32*	Jump near if above or equal (CF=0).
0F 82 *cw/cd*	JB *rel16/32*	Jump near if below (CF=1).
0F 86 *cw/cd*	JBE *rel16/32*	Jump near if below or equal (CF=1 or ZF=1).
0F 82 *cw/cd*	JC *rel16/32*	Jump near if carry (CF=1).
0F 84 *cw/cd*	JE *rel16/32*	Jump near if equal (ZF=1).
0F 84 *cw/cd*	JZ *rel16/32*	Jump near if 0 (ZF=1).
0F 8F *cw/cd*	JG *rel16/32*	Jump near if greater (ZF=0 and SF=OF).

Pour les CALL:

Opcode	Instruction	Description
E8 *cw*	CALL *rel16*	Call near, relative, displacement relative to next instruction
E8 *cd*	CALL *rel32*	Call near, relative, displacement relative to next instruction
FF /2	CALL *r/m16*	Call near, absolute indirect, address given in *r/m16*
FF /2	CALL *r/m32*	Call near, absolute indirect, address given in *r/m32*
9A *cd*	CALL *ptr16:16*	Call far, absolute, address given in operand
9A *cp*	CALL *ptr16:32*	Call far, absolute, address given in operand
FF /3	CALL *m16:16*	Call far, absolute indirect, address given in *m16:16*
FF /3	CALL *m16:32*	Call far, absolute indirect, address given in *m16:32*

De même, on récupère les opcodes possibles pour RET :

Opcode	Instruction	Description
C3	RET	Near return to calling procedure.
CB	RET	Far return to calling procedure.
C2 *iw*	RET *imm16*	Near return to calling procedure and pop *imm16* bytes from stack.
CA *iw*	RET *imm16*	Far return to calling procedure and pop *imm16* bytes from stack.

On peut maintenant établir une démarche pour rediriger une fonction :

1) Créer une zone virtuelle pour la redirection.
2) Récupérer l'adresse de la fonction à rediriger.
3) Récupérer la longueur de l'opcode avec le LDE.
4) S'agit-il d'un RET ou d'une instruction relogeable ?
Si oui, copier les octets dans la zone allouée.
Si non, passer à l'instruction suivante et recommencer à l'étape 3.

PARTIE 2 : Le JMP adresse_de_retour.

Pour effectuer le retour dans la fonction d'origine, on pourrait se contenter d'un simple JMP adresse_de_retour. Ici, K5 complique un peu le jeu en cherchant à brouiller les pistes. Tout d'abord, il remplace le JMP adresse_de_retour par :

```
push adresse_de_retour
ret
```

Ensuite, il crypte l'adresse_de_retour à l'aide d'une simple soustraction et le code précédent devient donc :

```
rdtsc
sub adresse_de_retour, eax          ← cryptage préalable

push adresse_de_retour              ← adresse cryptée
pushfd                              ← sauvegarde les flags
add dword ptr [esp+4], eax          ← décrypte l'adresse
popfd                               ← restaure les flags
ret
```

Voilà, nous sommes désormais capables de recoder une redirection dans le style K5. Je vous propose ici une redirection de la fonction MessageBoxA :

redirection de MessageBoxA

```
.586
.Model Flat ,StdCall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32.lib


.Data

table_LDE          DWORD   0
USER32_name        BYTE    "user32.dll",0
MessageBox_name    BYTE    "MessageBoxA",0
hMessageBox        DWORD   0
Nb_opcode          DWORD   0
MessageBox_        DWORD   0
titre              BYTE    "Information:",0
texte              BYTE    "MessageBox redirigée - Kaine#5",0
.Code
```

```
Main:

; ************************** Créer la zone de redirection

  push PAGE_EXECUTE_READWRITE
  push MEM_COMMIT + MEM_RESERVE
  push 1000h
  push 0
  call VirtualAlloc
  mov MessageBox_,eax


; ************************** Récupère l'adresse de MessageBoxA

  push offset USER32_name
  call LoadLibraryA
  push offset MessageBox_name
  push eax
  call GetProcAddress
  mov hMessageBox, eax

; ************************** Initialise le LDE
  push PAGE_READWRITE
  push MEM_COMMIT + MEM_RESERVE
  push 2048
  push 0
  call VirtualAlloc
  mov table_LDE,eax

  push table_LDE
  call disasm_init          ; décompression de la table du LDE

  xor esi, esi
  mov esi, hMessageBox
  mov edi, MessageBox_
  xor ecx, ecx
  xor eax, eax

Scan_Function :
  push esi                  ; adresse qui contient l'opcode à tester
  push table_LDE
  call disasm_main
  cmp eax, -1
  je End_Scan
  mov ecx, eax
  cmp byte ptr [esi], 0E8h
  je End_Scan
  cmp byte ptr [esi], 09Ah
  je End_Scan
  cmp byte ptr [esi], 0EBh
  je End_Scan
  cmp byte ptr [esi], 0E9h
  je End_Scan
  cmp byte ptr [esi], 0FFh
  je End_Scan
  cmp byte ptr [esi], 0Fh
  je End_Scan
  cmp byte ptr [esi], 0C2h
  je End_Scan
  cmp byte ptr [esi], 0C3h
  je End_Scan
```

```
  cmp byte ptr [esi], 0CBh
  je End_Scan
  cmp byte ptr [esi], 0CAh
  je End_Scan
  cmp byte ptr [esi], 0E3h
  je End_Scan
  cmp byte ptr [esi], 0EAh
  je End_Scan
  mov dl, byte ptr [esi]
  and dl, 0Fh
  cmp dl, 70h
  je End_Scan
  rep movsb                     ; Copier les octets dans la zone allouée
  jmp Scan_Function

End_Scan:                       ; création du JMP
  rdtsc
  sub esi, eax
  mov byte ptr [edi], 68h
  mov dword ptr [edi+1], esi
  add edi, 5
  mov byte ptr [edi], 9Ch
  inc edi
  mov dword ptr [edi], 4244481h
  add edi, 4
  mov dword ptr [edi], eax
  add edi, 4
  mov byte ptr [edi], 9Dh
  mov byte ptr [edi+1], 0C3h


; ************************************ Utiliser la MessageBox redirigée

  push 0
  push offset titre
  push offset texte
  push 0
  call MessageBox_


  push 0
  call ExitProcess


; LDE32 -- Length-Disassembler Engine
; FREEWARE
;
; programmed by Z0MBiE, http://z0mbie.cjb.net
;
; release 1.00        8-12-99
; release 1.01        9-12-99
; release 1.02       17-03-00  0xF6/0xF7 'test' opcode bugfixed
; release 1.03       21-04-00  bugfix: some prefixes before 0F were
cleared
;                              bugfix: error in MODRM analysis
;                              CD 20 now is 6 bytes length
; release 1.04        1-05-00  AAM & AAD bugfixed (was 1-byte len)
; release 1.05       xx-xx-xx  special edition, flags changed
; release 1.06        3-01-01  partially rewritten, __cdecl
```

```
; LDE32BIN.INC -- Length-Disassembler Engine //32-bit
; 1.06
; generated file. do not edit

disasm_init:
db 060h,08Bh,07Ch,024h,024h,0FCh,033h,0C0h
db 050h,050h,050h,068h,000h,0A8h,0AAh,002h
db 068h,07Fh,068h,0FFh,03Fh,068h,0A0h,0DEh
db 0E6h,0FFh,068h,0FFh,0FFh,0D5h,0DBh,068h
db 0AAh,0AAh,0FEh,0FFh,068h,0AAh,0AAh,0AAh
db 0AAh,068h,000h,000h,0AAh,0AAh,050h,050h
db 050h,050h,050h,050h,068h,054h,001h,000h
db 000h,068h,055h,0F5h,0FFh,041h,068h,0AAh
db 0DDh,0DEh,055h,068h,011h,051h,095h,019h
db 068h,0FFh,01Fh,011h,011h,068h,0AAh,0FFh
db 011h,0FAh,068h,096h,0CFh,060h,08Eh,068h
db 0AAh,0D6h,072h,0FCh,068h,088h,0AAh,0AAh
db 0AAh,068h,0D5h,088h,088h,088h,068h,09Bh
db 055h,08Dh,052h,068h,053h,0D5h,06Ch,036h
db 068h,0FFh,055h,055h,035h,068h,0F9h,0D6h
db 0FEh,0FFh,068h,088h,088h,088h,068h,068h
db 088h,088h,088h,088h,068h,0CAh,047h,053h
db 08Dh,068h,0DFh,07Bh,0C6h,0DCh,068h,0AAh
db 0AAh,0AAh,0AAh,068h,0AAh,0AAh,0AAh,0AAh
db 068h,0FDh,04Fh,0A9h,0ABh,068h,0EAh,0FEh
db 0A7h,0D4h,068h,029h,075h,0FFh,053h,068h
db 0FEh,0A7h,0A4h,0FFh,068h,04Ah,0FAh,09Fh
db 092h,068h,0FFh,029h,0E9h,07Fh,0B9h,000h
db 002h,000h,000h,033h,0DBh,033h,0C0h,0E8h
db 014h,000h,000h,000h,0ABh,0E2h,0F6h,061h
db 0C3h,00Bh,0DBh,075h,007h,05Dh,05Eh,05Ah
db 056h,055h,0B3h,020h,04Bh,0D1h,0EAh,0C3h
db 0E8h,0ECh,0FFh,0FFh,0FFh,00Fh,083h,07Fh
db 000h,000h,000h,0E8h,0E1h,0FFh,0FFh,0FFh
db 073h,003h,0B4h,040h,0C3h,0E8h,0D7h,0FFh
db 0FFh,0FFh,072h,057h,0E8h,0D0h,0FFh,0FFh
db 0FFh,073h,04Dh,0E8h,0C9h,0FFh,0FFh,0FFh
db 073h,043h,0E8h,0C2h,0FFh,0FFh,0FFh,072h
db 025h,0E8h,0BBh,0FFh,0FFh,0FFh,073h,003h
db 0B0h,020h,0C3h,0E8h,0B1h,0FFh,0FFh,0FFh
db 073h,005h,066h,0B8h,002h,020h,0C3h,0E8h
db 0A5h,0FFh,0FFh,0FFh,073h,005h,066h,0B8h
db 008h,010h,0C3h,0B4h,003h,0C3h,0E8h,096h
db 0FFh,0FFh,0FFh,073h,003h,0B4h,060h,0C3h
db 0E8h,08Ch,0FFh,0FFh,0FFh,073h,003h,0B0h
db 018h,0C3h,0B4h,002h,0C3h,0B4h,080h,0C3h
db 0B4h,001h,0C3h,0E8h,079h,0FFh,0FFh,0FFh
db 073h,00Dh,0E8h,072h,0FFh,0FFh,0FFh,073h
db 003h,0B0h,008h,0C3h,0B4h,041h,0C3h,0B4h
db 020h,0C3h,0E8h,062h,0FFh,0FFh,0FFh,014h
db 000h,048h,0C3h
disasm_main:
db 060h,08Bh,074h,024h,024h,08Bh,04Ch,024h
db 028h,033h,0D2h,033h,0C0h,080h,0E2h,0F7h
db 08Ah,001h,041h,00Bh,014h,086h,0F6h,0C2h
db 008h,075h,0F2h,03Ch,0F6h,074h,036h,03Ch
db 0F7h,074h,032h,03Ch,0CDh,074h,03Bh,03Ch
db 00Fh,074h,044h,0F6h,0C6h,080h,075h,052h
db 0F6h,0C6h,040h,075h,073h,0F6h,0C2h,020h
db 075h,054h,0F6h,0C6h,020h,075h,05Ch,08Bh
db 0C1h,02Bh,044h,024h,028h,081h,0E2h,007h
```

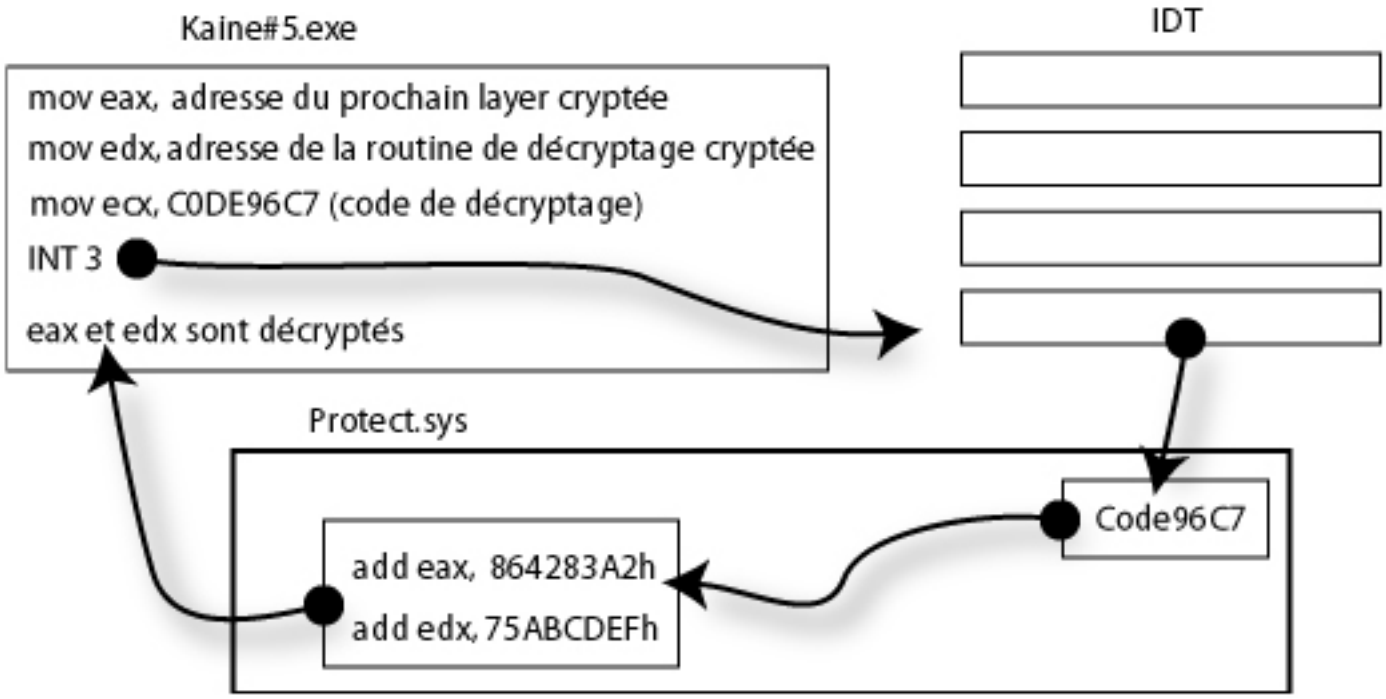
```
db 007h,000h,000h,002h,0C2h,002h,0C6h,089h
db 044h,024h,01Ch,061h,0C3h,080h,0CEh,040h
db 0F6h,001h,038h,075h,0CEh,080h,0CEh,080h
db 0EBh,0C9h,080h,0CEh,001h,080h,039h,020h
db 075h,0C1h,080h,0CEh,004h,0EBh,0BCh,08Ah
db 001h,041h,00Bh,094h,086h,000h,004h,000h
db 000h,083h,0FAh,0FFh,075h,0ADh,08Bh,0C2h
db 0EBh,0CDh,080h,0F6h,020h,0A8h,001h,075h
db 0A7h,080h,0F6h,021h,0EBh,0A2h,080h,0F2h
db 002h,0F6h,0C2h,010h,075h,0A4h,080h,0F2h
db 006h,0EBh,09Fh,080h,0F6h,002h,0F6h,0C6h
db 010h,075h,09Ch,080h,0F6h,006h,0EBh,097h
db 08Ah,001h,041h,08Ah,0E0h,066h,025h,007h
db 0C0h,080h,0FCh,0C0h,00Fh,084h,07Bh,0FFh
db 0FFh,0FFh,0F6h,0C2h,010h,075h,02Dh,03Ch
db 004h,075h,005h,08Ah,001h,041h,024h,007h
db 080h,0FCh,040h,074h,017h,080h,0FCh,080h
db 074h,00Ah,066h,03Dh,005h,000h,00Fh,085h
db 059h,0FFh,0FFh,0FFh,080h,0CAh,004h,0E9h
db 051h,0FFh,0FFh,0FFh,080h,0CAh,001h,0E9h
db 049h,0FFh,0FFh,0FFh,066h,03Dh,006h,000h
db 074h,00Eh,080h,0FCh,040h,074h,0EDh,080h
db 0FCh,080h,00Fh,085h,035h,0FFh,0FFh,0FFh
db 080h,0CAh,002h,0E9h,02Dh,0FFh,0FFh,0FFh

End Main
```

10 . LE DRIVER PROTECT.SYS

Afin de mieux se protéger, K5 utilise deux drivers : Protect.sys et Kaine.sys. Nous allons dès à présent étudier le premier d'entre eux.

Protect.sys est un kmd (kernel mode driver) de protection qui s'occupe de protéger à la fois Kaine#5.exe et Kaine.sys. Il communique avec ces deux binaires par l'intermédiaire des deux interruptions logicielles INT1 et INT3 et permet de décrypter des layers. En réalité, il dispose de 32 clés indispensables pour le décryptage de centaines de layers.

Le principe de fonctionnement est le suivant : dans certains layers, Kaine#5.exe et Kaine.sys vont appeler les interruptions INT1 et INT3. Protect.sys, ayant **hooké** au préalable ces deux interruptions, va intercepter l'appel et va récupérer un paramètre (ecx) et deux adresses cryptées (dans eax et edx) transmis par Kaine#5.exe ou Kaine.sys. Protect.sys va alors chercher 2 clés de décryptage dans ses tables et va décrypter les deux adresses eax et edx puis va rendre la main à Kaine#5.exe en lui transmettant ces deux nouvelles adresses. Kaine#5.exe (ou Kaine.sys) sera alors en mesure de décrypter le layer suivant. Voici le principe illustré :



10.1 Lancement de Protect.sys

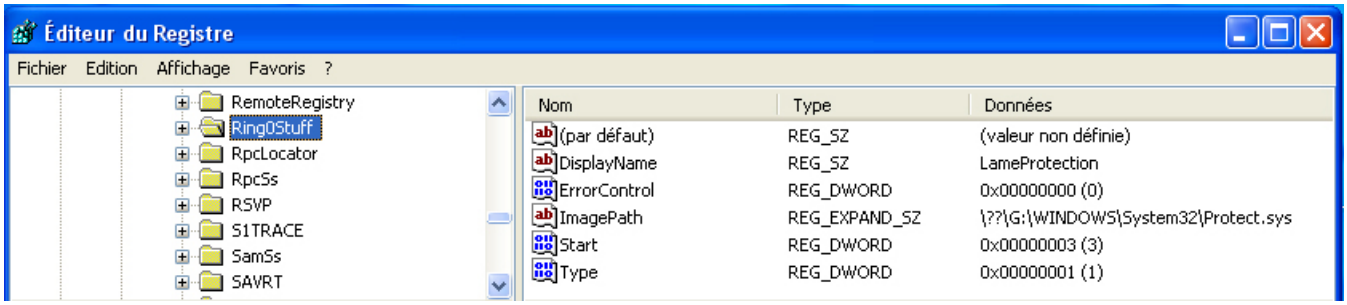
Le driver Protect.sys n'existe pas a priori. Il est créé durant le runtime par le binaire Kaine#5.exe et se trouve crypté dans le binaire lui-même. Une fois que le driver est créé, K5 le lance suivant une procédure standard que nous allons voir ci-dessous. Toutes les fonctions utilisées par K5 pour la création et le lancement de Protect.sys ont été redirigées au préalable. Pour « voir » le déroulement des opérations, il est donc nécessaire de poser des BPX sur les zones de redirection. Voici donc schématiquement le déroulement des opérations :

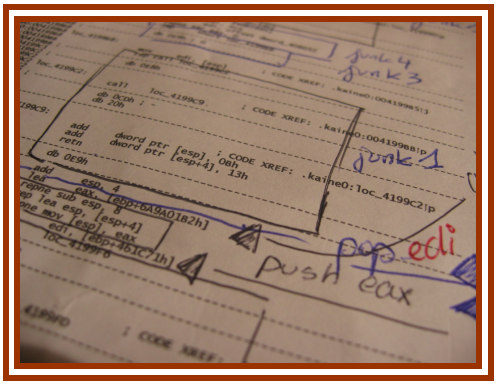
GetSystemDirectoryA	Détermine le chemin complet vers le dossier système. exemple : « C:\WINDOWS\SYSTEM32\ »
lstrcatA	Crée le chemin complet vers le driver qui va être crée : C:\WINDOWS\SYSTEM32\Protect.sys
CreateFileA	Créer le fichier Protect.sys dans le dossier système
WriteFile	Remplir le fichier crée par le code du driver
CloseHandle	Fermer le fichier crée.
OpenSCManagerA	Ouvrir le Service Control Manager
CreateServiceA	Créer le service kernel pour Protect.sys en créant une entrée dans la base de donnée du SCM

StarServiceA	☐	Démarrer le driver Protect.sys
CreateFileA	☐	Vérifier que le driver est bien lancé pour continuer. On essaie ici d'ouvrir le service « \\.\Protect.sys »

Les opération se déroulent en deux phases : on commence par créer le driver puis on le lance. Pour information, la base de registre liste les services du SCM. Pour y accèder, il suffit d'aller à **HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services.**

Voici la clé que l'on peut y voir :





JOUR 5 : Mercredi 25 mai 2005 : 14h30

Bon sang de bois ! Je parviens à « voir » le code du driver.. je crois qu'il hooke INT1 et INT3 mais je ne comprends pas comment il interagit avec le binaire…il faut que j'essaie de le debugger avec Soft Ice. Il faut aussi que je fasse quelque chose pour mon coude…

Voici le code asm d'une procédure de lancement d'un driver :

Charger Protect.sys

```
.586
.Model Flat ,StdCall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32.lib
include \masm32\include\advapi32.inc
includelib \masm32\lib\advapi32.lib


.Data

hSCManager      DWORD 0
hService        DWORD 0
DriverPath      BYTE  "C:\WINDOWS\SYSTEM32\Protect.sys",0
DisplayName     BYTE  "LameProtection",0
ServiceName     BYTE  "Ring0Stuff",0
```

```
.Code
Main:
  push SC_MANAGER_CREATE_SERVICE
  push NULL
  push NULL
  call OpenSCManagerA
  mov hSCManager, eax

  push NULL
  push NULL
  push NULL
  push NULL
  push NULL
  push offset DriverPath
  push SERVICE_ERROR_IGNORE
  push SERVICE_DEMAND_START
  push SERVICE_KERNEL_DRIVER
  push SERVICE_START + DELETE
  push offset DisplayName
  push offset ServiceName
  push hSCManager
  call CreateServiceA
  mov hService,eax

  push NULL
  push 0
  push hService
  call StartServiceA

  push hService
  call DeleteService

  push hService
  call CloseServiceHandle

  push 0
  call ExitProcess
End Main
```

Néanmoins, il arrive parfois (ça a dû m'arriver une bonne dizaine de fois en utilisant Soft Ice) qu'on lance le driver sans le vouloir et que le système se plante parce qu'on a fait une manipulation dangereuse (dans le style débugger le driver par exemple). Et la mauvaise surprise n'est pas le BSOD ! La mauvaise surprise est que **le service Ring0Stuff est toujours actif même si on a rebooté** !! Il est donc absolument nécessaire de le fermer si on veut exploiter le crackme de nouveau. Pour cela, il suffit d'ouvrir le service Ring0Stuff et de le supprimer comme ceci :

fermer le service Ring0Stuff

```
.586
.Model Flat ,StdCall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32.lib
include \masm32\include\advapi32.inc
includelib \masm32\lib\advapi32.lib
```

```
.Data

hSCManager      DWORD 0
hService        DWORD 0
DriverPath      BYTE  "C:\WINDOWS\SYSTEM32\Protect.sys",0
DisplayName     BYTE  "LameProtection",0
ServiceName     BYTE  "Ring0Stuff",0



.Code
Main:
  push SC_MANAGER_CREATE_SERVICE
  push NULL
  push NULL
  call OpenSCManagerA
  mov hSCManager, eax

  push SERVICE_ALL_ACCESS
  push offset ServiceName
  push hSCManager
  call OpenServiceA
  mov hService,eax

  push hService
  call DeleteService

  push hService
  call CloseServiceHandle

  push 0
  call ExitProcess
End Main
```

10.2 Comment récupérer Protect.sys ?

« Tous les chemins mènent à Rome ». J'ai personnellement choisi de faire le chemin à pied en me coltinant toutes les embûches. Pour récupérer le driver, j'ai posé un BPX sur la routine de redirection et j'ai guetté le moment où CloseHandle a été redirigé. J'ai alors posé un BPX sur cette redirection et j'ai lancé K5. J'ai alors tracé doucement de façon à ce que CloseHandle soit exécuté. Il suffit à partir de là d'aller dans le dossier système de windows et de faire un copier/coller du binaire Protect.sys.

JOUR 6 : Jeudi 26 mai 2005 : 19h10

Ouf ! J'ai enfin installé Soft Ice et j'arrive à breaker à l'OEP du driver… je commence à comprendre comment tout cela fonctionne. Kharneth n'a apparemment toujours pas réussi à comprendre le trick du CloseHandle… ☺ son ego en prend un coup à mon avis…il a décidé de continuer. TTO s'est aussi lancé dans l'aventure. Il essaie apparemment de tracer avec son « Traceur ».

10.3 Comment étudier Protect.sys ?

Une fois que nous disposons du driver, il reste le problème principal : étudier son contenu. OllyDbg est incapable en l'état d'étudier un driver (problème ring3 – ring0). Il ne nous reste que peu de possibilités. On peut essayer de désassembler Protect.sys avec IDA mais malheureusement, le binaire est crypté par des centaines de layers : impossible de désassembler à la main ! On peut utiliser Soft Ice mais on s'expose à de gros soucis : les risques de BSOD (Blue Screen Of Death) sont importants (croyez moi, j'ai fait le test). On va donc rester en ring3 et effectuer un travail au préalable sur le driver pour qu'on puisse le « lire » à l'aide de OllyDbg. J'expliquerai néanmoins comment faire avec Soft Ice pour tracer ce driver.

Si on regarde la structure du driver avec LordPE, on voit qu'il est composé de 4 sections dont les RVAs sont somme toute assez étranges :

Name	VOffset	Vsize	ROffset	RSize	Flags
.text	260	1962	260	1980	E0000040
.rdata	1BE0	70	1BE0	80	E0000040
INIT	1C60	B8	1C60	C0	E0000040
.reloc	1D20	213DC	1D20	213E0	E0000040

Pour pouvoir « lire » le driver à l'aide d'OllyDbg, il faut donc convertir ce binaire en un binaire « standard » avec des RVAs normales. En fait, on va faire plus simple, on va « coller » les sections du driver sur un exe déjà valide ! On va modifier l'EP de ce binaire de façon à pointer sur l'EP du driver ajouté et le tour sera joué ! Il y a néanmoins un gros inconvénient à fonctionner de cette manière : si le driver utilise des adresses fixes qui dépendent des RVAs d'origine, nous allons être obligés de convertir ces RVAs afin de faire correspondre les bonnes adresses dans notre nouveau driver.

1) Tout d'abord, nous allons récupérer les sections du driver toujours à l'aide de LordPE. Pour se faire, il suffit de faire un clic droit sur la section à récupérer et de choisir « **Save section to disk** ». L'opération est à faire 4 fois.

2) On choisit un binaire cible qui va héberger nos 4 sections récupérées. J'ai choisi pour ma part de façon arbitraire une simple messageBox. Pour coller les nouvelles sections, il suffit de faire un clic droit sur l'une des sections du binaire, de sélectionner « **Load section from disk** ». L'opération est à effectuer 4 fois . On obtient alors un nouveau binaire semblable à celui-ci :

Name	VOffset	Vsize	ROffset	RSize	Flags

.text	1000	166	400	200	E0000040
.rdata	2000	F6	600	200	E0000040
.data	3000	2E	800	200	E0000040
_text	4000	2000	A00	1980	E0000040
_rdata	6000	1000	2400	80	E0000040
_INIT	7000	1000	2600	C0	E0000040
_reloc	8000	22000	2800	$213^{E}0$	E0000040

Vous voyez les nouvelles sections ajoutées avec leurs nouvelles Virtual Offset. Durant le debuggage, on prendra bien soin de surveiller le code et de remplacer les anciennes RVAs par les nouvelles. Par exemple, si on voit dans le binaire une adresse qui vaut 280h, on doit la convertir comme ceci :

Nouvelle Adresse = Ancienne Adresse – 260h + 4000h

soit

4020h = 280h – 260h + 4000h.

3) Pensez à changer l'EP de ce binaire pour qu'il pointe vers l'EP du driver. Attention au changement d'adresse ! A partir de là, on peut « débugger » le driver .

10.4 Fonctionnement de Protect.sys

10.4.1 Schéma du driver

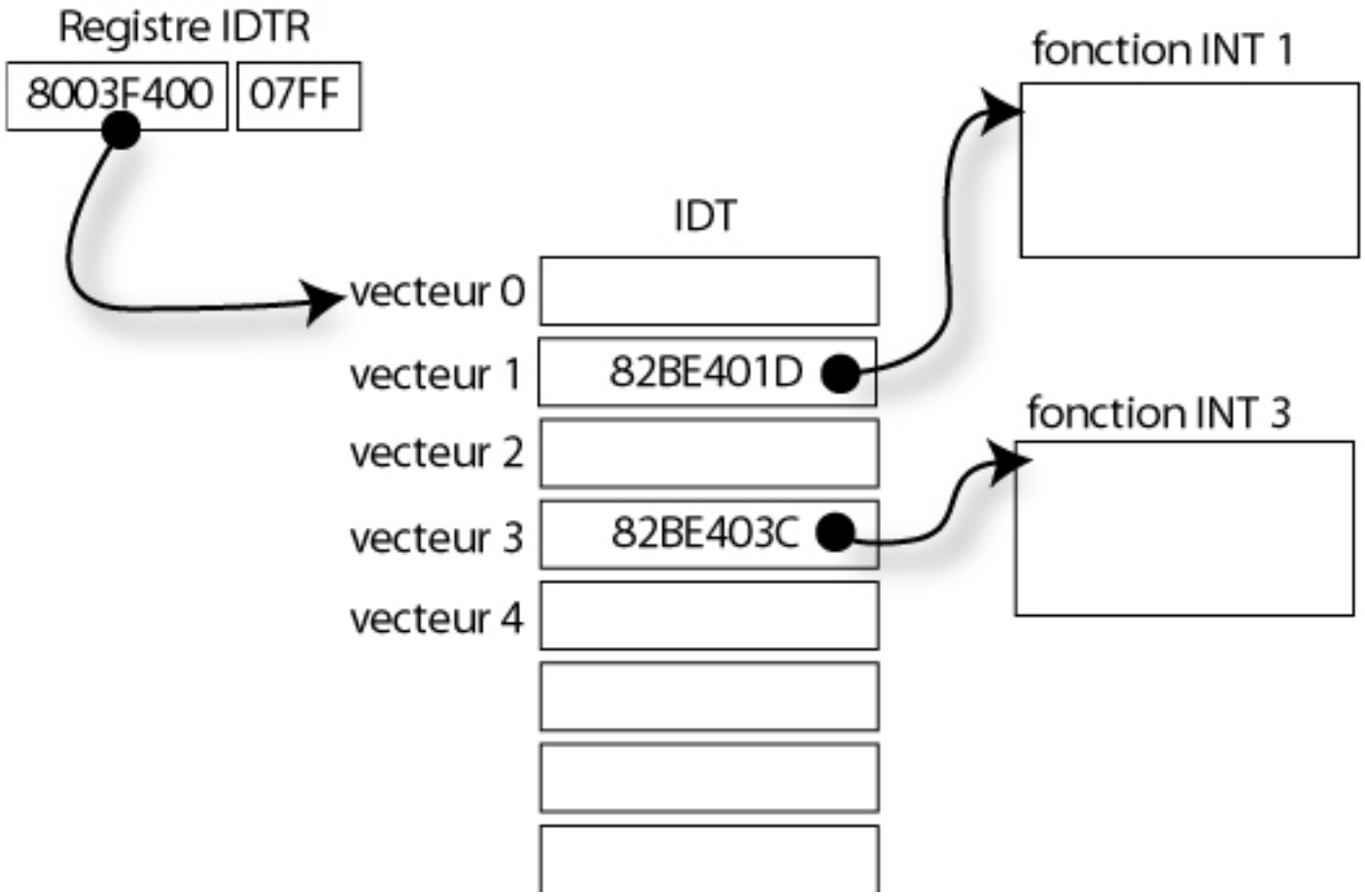
Pour avoir une vue d'ensemble du driver, je vous propose ce petit schéma très synthétique :

Layers cryptés	☐	environ 200 layers identiques à ceux de K5
Décompression	☐	Puis, le driver décompresse son code à l'aide de l'ApLib
fixer des adresses	☐	Puis, le driver fixe un certain nombre d'adresses non relogeables
hooker INT 3	☐	Remplace la fonction de INT 3
hokker INT 1	☐	Remplace la fonction de INT 1

10.4.2 Hooker des interruptions

Nous allons ici attaquer le cœur du problème. Ce driver a été conçu pour décrypter des layers en hookant deux interruptions logicielles : INT1 et INT3.

Les interruptions sont accessibles via une table appelée l'IDT (Interrupt Descriptor Table) située dans la partie supérieure des 4Go du process (en ring0). Cette table est composée d'adresses qui pointent vers des routines qui sont chargées de répondre à l'appel de l'interruption. On accède à ces routines en connaissant le vecteur de l'interruption qui se comporte comme un indice. Voyez plutôt le schéma suivant en guise d'exemple :



On accède à l'IDT via l'IDTR (Interrupt Descriptor Table Register). Le premier dword de ce registre indique l'adresse de l'IDT. Le Word suivant indique la taille de l'IDT. Les valeurs que vous pouvez voir sont celles que j'ai récupérées sur ma machine. L'IDT est une table de DWORDs. Le vecteur 1 se situe donc en 8h et le vecteur 3 en 18h. Les deux DWORDs que vous voyez dans l'IDT sont des pointeurs qui indiquent l'adresse des fonctions qui seront exécutées si les interruptions INT1 et INT3 sont invoquées.

Protect.sys va remplacer les deux fonctions INT1 et INT3 par ses propres routines. Voici le code commenté (remarquez les adresses qui sont des adresses fictives) :

Hooker INT1 et INT3

```
00405838  PUSH EBP
00405839  MOV EBP,ESP
0040583B  ADD ESP,-8
0040583E  MOV DWORD PTR SS:[EBP-4],C0000182
00405845  LEA EAX,DWORD PTR SS:[EBP-8]
00405848  PUSH EAX
00405849  PUSH 0
0040584B  PUSH 0
0040584D  PUSH 22
```

```
0040584F  PUSH decrypte.004059B8
00405854  PUSH 0
00405856  PUSH DWORD PTR SS:[EBP+8]
00405859  CALL IoCreateDevice
0040585E  OR EAX,EAX
00405860  JNZ SHORT decrypte.004058DC
00405862  PUSH decrypte.004059B8
00405867  PUSH decrypte.004059E8
0040586C  CALL IoCreateSymbolicLink
00405871  OR EAX,EAX
00405873  JNZ SHORT decrypte.004058CD
00405875  MOV EAX,DWORD PTR SS:[EBP+8]
00405878  MOV DWORD PTR DS:[EAX+38],decrypte.00405909
0040587F  MOV DWORD PTR DS:[EAX+40],decrypte.00405909
00405886  MOV DWORD PTR DS:[EAX+70],decrypte.00405921
0040588D  MOV DWORD PTR DS:[EAX+34],decrypte.004058E3
00405894  MOV DWORD PTR SS:[EBP-4],0
0040589B  SIDT FWORD PTR DS:[404000]          Charge l'adresse de l'IDTR
004058A2  CLI
004058A3  MOV EAX,CR0
004058A6  AND EAX,FFFEFFFF
004058AB  MOV CR0,EAX
004058AE  CALL decrypte.00404034              Hooker INT 3
004058B3  CALL decrypte.0040408F              Hooker INT 1
004058B8  MOV EAX,CR0
004058BB  OR EAX,10000
004058C0  MOV CR0,EAX
004058C3  STI
004058C4  MOV EAX,DWORD PTR SS:[EBP-4]
004058C7  LEAVE
004058C8  RETN 8
```

Hooker INT3

```
00404034  PUSHAD                              EBX = Base de l'IDT
00404035  MOV EBX,DWORD PTR DS:[404002]
0040403B  ADD EBX,18                          EBX = pointe sur le vecteur 3
0040403E  MOV DX,WORD PTR DS:[EBX+6]          EDX = adresse de la fonction de INT3
00404042  SHL EDX,10
00404045  MOV DX,WORD PTR DS:[EBX]
00404048  MOV DWORD PTR DS:[404018],EDX
0040404E  MOV EBX,decrypte.0040401C           Sauvegarde les 6 premiers octets de la fonction à
00404053  MOV EAX,DWORD PTR DS:[EDX]          remplacer
00404055  MOV DWORD PTR DS:[EBX],EAX
00404057  MOV AX,WORD PTR DS:[EDX+4]
0040405B  MOV WORD PTR DS:[EBX+4],AX

0040405F  MOV BYTE PTR DS:[EDX],0E8           Remplacer la fonction par :
00404062  MOV EAX,decrypte.00404D61
00404067  SUB EAX,EDX                         call 404D61
00404069  SUB EAX,5                           iretd
0040406C  MOV DWORD PTR DS:[EDX+1],EAX
0040406F  MOV BYTE PTR DS:[EDX+5],0CF
00404073  POPAD
00404074  RETN
```

Hooker INT1

```
0040408F  PUSHAD                              EBX = Base de l'IDT
```

```
00404090  MOV EBX,DWORD PTR DS:[404002]
00404096  ADD EBX,8                          EBX = pointe sur le vecteur 1
00404099  MOV DX,WORD PTR DS:[EBX+6]         EDX = adresse de la fonction de INT3
0040409D  SHL EDX,10
004040A0  MOV DX,WORD PTR DS:[EBX]
004040A3  MOV DWORD PTR DS:[404014],EDX

004040A9  MOV EBX,decrypte.00404024          Sauvegarde les 16 premiers octets de la fonction
004040AE  MOV EAX,DWORD PTR DS:[EDX]         à remplacer
004040B0  MOV DWORD PTR DS:[EBX],EAX
004040B2  MOV EAX,DWORD PTR DS:[EDX+4]
004040B5  MOV DWORD PTR DS:[EBX+4],EAX
004040B8  MOV EAX,DWORD PTR DS:[EDX+8]
004040BB  MOV DWORD PTR DS:[EBX+8],EAX
004040BE  MOV EAX,DWORD PTR DS:[EDX+C]
004040C1  MOV DWORD PTR DS:[EBX+C],EAX

004040C4  MOV DWORD PTR DS:[EDX],2464819C    Remplacer la fonction par :
004040CA  MOV DWORD PTR DS:[EDX+4],FFFEFF0C
004040D1  MOV BYTE PTR DS:[EDX+8],0FF        pushfd
004040D5  MOV BYTE PTR DS:[EDX+9],0E8        and dword ptr [esp+C], FFFFFEFF
004040D9  LEA EDX,DWORD PTR DS:[EDX+9]       call 404117
004040DC  MOV EAX,decrypte.00404117          popfd
004040E1  SUB EAX,EDX                        iretd
004040E3  SUB EAX,5
004040E6  MOV DWORD PTR DS:[EDX+1],EAX
004040E9  MOV BYTE PTR DS:[EDX+5],9D
004040ED  MOV BYTE PTR DS:[EDX+6],0CF
004040F1  POPAD
004040F2  RETN
```

Voilà donc nos deux fonctions remplacées. Pour INT3, il s'agit juste d'un appel à une routine qui sera chargée de décrypter les deux adresses eax et edx. Pour INT1, avant d'appeler une routine similaire, la fonction va désarmer le TRAP FLAG à l'aide d'un and.

10.4.3 Décryptage des layers de Kaine#5.exe

Une fois que Protect.sys a hooké ces deux interruptions, il redonne la main à K5 et attend un appel à Int1 ou Int3. Dans le cas où INT3 est appelé, la routine située (ici) en 404D61 est exécutée. Après quelques obfuscations, on arrive sur ce petit bout de code :

```
00404D83  PUSHFD
00404D84  PUSH ESI
00404D85  PUSH ECX
00404D86  MOV ESI,ECX
00404D88  SHR ESI,10
00404D8B  CMP SI,0C0DE
00404D90  JNZ SHORT decrypte.00404DA0
00404D92  AND ECX,0F
00404D95  PUSH EBX
00404D96  MOV EBX,DWORD PTR DS:[ECX*4+404D63]
00404D9D  CALL EBX
00404D9F  POP EBX
```

```
00404DA0  POP ECX
00404DA1  POP ESI
00404DA2  POPFD
00404DA3  RETN
```

Cette routine commence par récupérer la valeur de ECX transmise par K5. Il s'agit en fait d'un code. Si ECX commence par « 0C0DE », il s'agit alors bien d'un appel de K5 et on continue le travail. A partir de là, seul le premier octet de ECX nous intéresse. Il sert en fait d'indice pour accéder à une table qui contient des adresses de routines qui seront chargées de décrypter les valeurs EAX et EDX.

Voici un exemple :
K5 effectue l'opération suivante :

appel de Protect.sys

```
mov eax, 2CE4DC25h
mov edx, 6A83D5B6h
mov ecx, C0DECCC0
INT 3
```

A ce moment là, Protect.sys réagit et la valeur 0 de ECX (C0DECCC**0**) permet d'atteindre une routine de décryptage :

routine de décryptage

```
add eax, D35B739Ah
add edx, 95BC7B55h
```

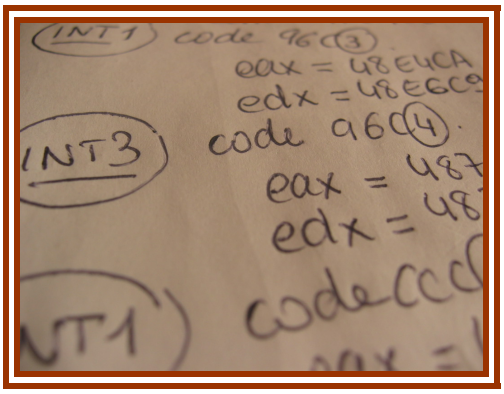
Contrairement aux apparences, la routine de décryptage ne se résume pas à deux simples opérations arithmétiques. Les routines sont giganstesques et constituées de junk code à ne pas en finir. De prime abord, il n'est pas si simple de retrouver le code « utile » dans cette jungle d'instructions. On obtient alors de nouvelles valeurs pour K5 : EAX = 404FBFh (adresse valide) et EDX = 40510Bh (valide également). Ces deux adresses sont alors utilisées pour décrypter un layer.

Voici maintenant la table complète de décryptage des valeurs de EAX et EDX suivant les valeurs de ECX :

INT 3			
Si ECX se termine par :	Protect.sys appelle une routine située en (exemple vu sous SI) :	La routine modifie EAX	La routine modifie EDX
0	F0881004	add eax, D35B739Ah	add edx, 95BC7B55h
1	F08811A3	add eax, 4AC6205Bh	add edx, E596AE97h
2	F08812E8	add eax, 3A54FEA3h	add edx, 643876ADh
3	F088142D	add eax, 6CB2895Dh	add edx, A78A2B63h
4	F088158E	add eax, 953AB431h	add edx, 12345678h
5	F08816EF	add eax, 0BD7EC86h	add edx, ECC8ECC9h
6	F0881837	add eax, 7BC6AE8Ah	add edx, 6A008BECh
7	F088193F	add eax, 864283A2h	add edx, 75ABCDEFh

On dispose du même principe de décryptage pour INT1. Voici la table coorespondante :

INT 1			
Si ECX se termine par :	Protect.sys appelle une routine située en (exemple vu sous SI) :	La routine modifie EAX	La routine modifie EDX
0	F08803BA	add eax, C86345ADh	add edx, 42685D1Dh
1	F088054C	add eax, 00112234h	add edx, CB97DEAFh
2	F08806E8	add eax, 825371FFh	add edx, EF45A2AAh
3	F088082D	add eax, 6CB2895Dh	add edx, A78A2B63h
4	F08809CA	add eax, D6C81FF2h	add edx, 98765432h
5	F0880B2C	add eax, 7BC6AE8Ah	add edx, 95FF7414h
6	F0880C72	add eax, 5C5C5C5Dh	add edx, AAAAAAAAh
7	F0880E24	add eax, 934D76A3h	add edx, 12345678h



JOUR 7 : Vendredi 27 mai 2005 : 7h40

J'ai compris ! Protect.sys permet de décrypter des layers de Kaine#5.. je commence le travail à la main…bon sang de bois ! il y en a des centaines !! Ras le bol de ce crackme ! il faut que j'automatise la tâche… je viens d'apprendre sur IRC que Kharneth a réussi à choper le driver sans avoir tracé jusqu'au startservice !! hmm… ça ne présage rien de bon ça ! Allez, Bea, tu es trop lent !!

Connaître ces tables n'est pas suffisant pour décrypter les layers qui les utilisent. Imaginez en fait qu'il y a des centaines de layers qui utilisent ce mode de décryptage. On doit donc automatiser le travail. J'ai opté pour la facilité et j'ai choisi de remplacer le driver Protect.sys par un handler de SEH.

Si vous observez le schéma du K5, vous constatez qu'après le lancement de Protect.sys, on tombe sur ces fameux layers. On réalité, ils sont protégés par un seh qui affiche un message d'erreur s'il est déclenché. Pour pouvoir franchir ces layers, j'ai donc empêché le driver de démarrer et j'ai remplacé le handler déjà installé par mon propre handler qui va faire office de décrypteur de layers. En fait, dès qu'un INT1 ou un INT3 est rencontré, le handler est exécuté. Il suffit alors de tester la valeur de ECX et de décrypter EAX et EDX à l'aide des valeurs adéquates avant de rendre la main. On peut ainsi franchir ces layers en une fraction de seconde.

Construction du handler / Décrypteur :

J'ai donc codé le handler comme ceci :

handler pour décrypter

```
.386
.model flat,stdcall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32.lib
.data

.code
start:
mov eax, dword ptr [esp+4]
```

```
mov ecx, dword ptr [esp+0Ch]
mov eax, dword ptr [eax]
cmp eax, 080000004h
je SINGLE_STEP
cmp eax, 080000003h
je BREAK_POINT

; ************************* INT3

BREAK_POINT:
add dword ptr [ecx+0B8h], 1
mov eax, dword ptr [ecx+0B0h]
mov edx, dword ptr [ecx+0A8h]
push ecx
mov ecx, dword ptr [ecx+0ACh]
and ecx, 0fh

cmp cl, 0
jne @F
add eax, 0d35b739ah
add edx, 095bc7b55h
jmp fin

@@:
cmp cl, 1
jne @F
add eax, 04ac6205bh
add edx, 0e596ae97h
jmp fin

@@:
cmp cl, 2
jne @F
add eax, 03a54fea3h
add edx, 0643876adh
jmp fin

@@:
cmp cl, 3
jne @F
add eax, 06cb2895dh
add edx, 0a78a2b63h
jmp fin

@@:
cmp cl, 4
jne @F
add eax, 0953ab431h
add edx, 012345678h
jmp fin

@@:
cmp cl, 5
jne @F
add eax, 0bd7ec86h
add edx, 0ecc8ecc9h
jmp fin

@@:
cmp cl, 6
jne @F
```

```
add eax, 07bc6ae8ah
add edx, 06a008bech
jmp fin

@@:
cmp cl, 7
jne @F
add eax, 0864283a2h
add edx, 075abcdefh
@@:
jmp fin

SINGLE_STEP:
; ************************* INT1

add dword ptr [ecx+0B8h], 0
mov eax, dword ptr [ecx+0B0h]
mov edx, dword ptr [ecx+0A8h]
push ecx
mov ecx, dword ptr [ecx+0ACh]
and ecx, 0fh
nop
nop
nop
cmp cl, 0
jne @F
add eax, 0c86345adh
add edx, 042685d1dh
jmp fin

@@:
cmp cl, 1
jne @F
add eax, 000112234h
add edx, 0cb97deafh
jmp fin

@@:
cmp cl, 2
jne @F
add eax, 0825371ffh
add edx, 0ef45a2aah
jmp fin

@@:
cmp cl, 3
jne @F
add eax, 06cb2895dh
add edx, 0a78a2b63h
jmp fin

@@:
cmp cl, 4
jne @F
add eax, 0d6c81ff2h
add edx, 098765432h
jmp fin

@@:
cmp cl, 5
jne @F
```

```
add eax, 07bc6ae8ah
add edx, 095ff7414h
jmp fin

@@:
cmp cl, 6
jne @F
add eax, 05c5c5c5dh
add edx, 0aaaaaaaah
jmp fin

@@:
cmp cl, 7
jne @F
add eax, 0934d76a3h
add edx, 012345678h
jmp fin

@@:
fin:
pop ecx
mov dword ptr [ecx+0B0h], eax
mov dword ptr [ecx+0A8h], edx
xor eax, eax
ret



end start
```

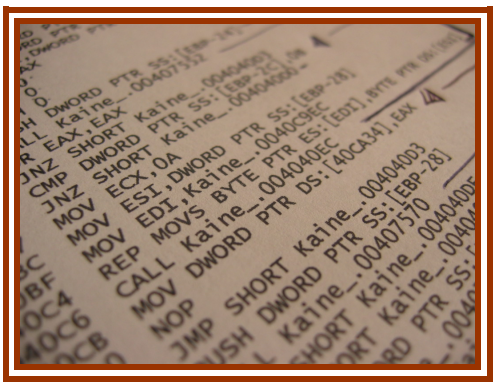
J'ai compilé ce code comme un exe et j'ai copié le code obtenu dans une zone non utilisée par K5. J'ai choisi en fait l'une des zones allouées pour les redirections. J'aurais très bien pu faire une injection de code automatique dans le process en recherchant le process, en l'ouvrant et en copiant le handler. On a tout aussi vite fait de faire un simple **Binary Copy / Binary Paste** avec OllyDbg.

Ensuite, il faut interdire l'utilisation du driver Protect.sys. Pour ça, j'ai donc breaké sur la redirection de CloseHandle (comme tout à l'heure), et à ce moment là, j'ai remplacé le driver Protect.sys par un autre driver (que j'ai aussi appelé Protect.sys) qui ne fait qu'un simple bip ! Il s'agit en fait de l'un des drivers de démonstration du KMDKIT appelé beeper.sys. Ainsi, je laisse le K5 exécuter gentiment « son » driver sans intervenir sur le binaire.

Je contourne alors le test de CreateFileA qui vérifie que le driver est bien chargé en mémoire.

Ensuite, il suffit de modifier l'adresse du SEH de façon à le faire pointer vers mon handler et le tour est joué !

Ainsi, on parvient à déjouer le premier driver en le neutralisant et en le remplaçant par un simple SEH. On peut alors atteindre le lancement du second driver, responsable cette fois de la vérification du serial.



JOUR 8 : Samedi 28 mai 2005 : 9h55

Gagné !! J'ai réussi à franchir les layers protégés par Protect.sys et je suis tombé hier soir sur un second driver Kaine.sys. Pour le moment, je continue à tracer le binaire… Je viens d'apprendre sur IRC que Kharneth a réussi encore une fois à choper le second driver et qu'il essaie de l'unpacker en ce moment même !! Pire… je viens de parler à Kaine qui m'a dit que Kharneth avait de l'avance sur moi !! IMPOSSIBLE ! J'ai du mal à le croire… il semblerait que Kharneth ait compris l'esprit du crackme…tssss. Qu'a-t-il compris ??

11. LE DRIVER KAINE.SYS

11.1 Schéma de Kaine.sys

Je commence d'abord par vous donner une vue d'ensemble de ce driver :

Layers cryptés	environ 200 layers protégés par Protect.sys
Décompression	Puis, le driver décompresse son code à l'aide de l'ApLib
fixer des adresses	Puis, le driver fixe un certain nombre d'adresses non relogeables
ouvrir License.key et récupérer le serial	utilisation de fonctions de ntoskernel.exe
Lancer la VM et tester le serial	Si le serial est valide, le driver renvoie eax=1 au K5. En cas d'échec, eax=0 lui est envoyé !

11.2 Décryptage des layers de Kaine.sys

Les layers sont identiques à ceux vus dans K5. Ils exécutent des INT3 et INT1 hookés par Protect.sys. J'ai donc utilisé un handler similaire au précédent. J'ai utilisé le SEH par défaut pour rediriger l'interruption vers mon handler.

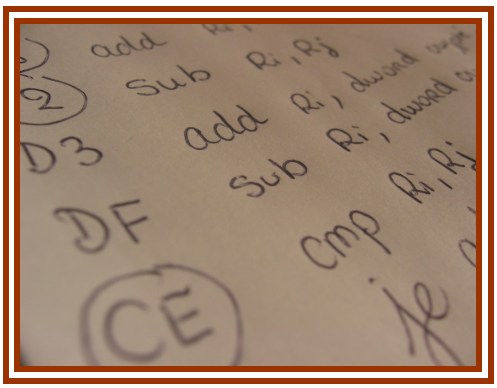
11.3 Fonctionnement de Kaine.sys

Comme vous pouvez le voir sur le petit schéma, Kaine.sys va commencer par ouvrir le fichier License.key et copier son contenu (le serial) dans une zone allouée. Puis, il lance une VM

qui va exécuter un pcode qui va vérifier la validité du serial. Si le serial est valide, la VM renvoie 1 dans eax. Si le serial est bidon, la VM renvoie 0 dans eax. Voici le code de Kaine.sys commenté :

```
00404000  PUSH EBP
00404001  MOV EBP,ESP
00404003  ADD ESP,-44
00404006  PUSHAD
00404007  MOV DWORD PTR DS:[40CA34],0
00404011  LEA ECX,DWORD PTR SS:[EBP-18]
00404014  MOV DWORD PTR DS:[ECX],18
0040401A  AND DWORD PTR DS:[ECX+4],0
0040401E  MOV DWORD PTR DS:[ECX+C],240
00404025  AND DWORD PTR DS:[ECX+10],0
00404029  MOV DWORD PTR DS:[ECX+8],Kaine_.0040C932
00404030  AND DWORD PTR DS:[ECX+14],0
00404034  PUSH 20
00404036  PUSH 7
00404038  LEA EAX,DWORD PTR SS:[EBP-20]
0040403B  PUSH EAX
0040403C  LEA EAX,DWORD PTR SS:[EBP-18]
0040403F  PUSH EAX
00404040  PUSH 100001
00404045  LEA EAX,DWORD PTR SS:[EBP-24]
00404048  PUSH EAX
00404049  CALL Kaine_.0040755E   ; ================= ZwQueryInformationFile
0040404E  OR EAX,EAX
00404050  JNZ Kaine_.004040E9
00404056  PUSH 5
00404058  PUSH 18
0040405A  LEA EAX,DWORD PTR SS:[EBP-44]   ; ====== "C:\License.key"
0040405D  PUSH EAX
0040405E  LEA EAX,DWORD PTR SS:[EBP-20]   ; ====== DesiredAccess
00404061  PUSH EAX
00404062  PUSH DWORD PTR SS:[EBP-24] ; =========== Handle

00404065  CALL Kaine_.00407558    ; ================== ZwOpenFile
0040406A  OR EAX,EAX
0040406C  JNZ SHORT Kaine_.004040DF
0040406E  MOV EAX,DWORD PTR SS:[EBP-3C]
00404071  INC EAX
00404072  MOV DWORD PTR SS:[EBP-2C],EAX
00404075  PUSH DWORD PTR SS:[EBP-2C]
00404078  PUSH 1
0040407A  CALL Kaine_.00407576   ; ==================== ExAllocatePool
0040407F  OR EAX,EAX
00404081  JE SHORT Kaine_.004040DD
00404083  MOV DWORD PTR SS:[EBP-28],EAX
00404086  PUSH DWORD PTR SS:[EBP-2C]
00404089  PUSH DWORD PTR SS:[EBP-28]
0040408C  CALL Kaine_.0040756A    ; ================ RtlZeroMemory
00404091  PUSH 0
00404093  PUSH 0
00404095  PUSH DWORD PTR SS:[EBP-2C]
00404098  PUSH DWORD PTR SS:[EBP-28]
0040409B  LEA EAX,DWORD PTR SS:[EBP-20]
0040409E  PUSH EAX
0040409F  PUSH 0
004040A1  PUSH 0
004040A3  PUSH 0
```

```
004040A5  PUSH DWORD PTR SS:[EBP-24]

004040A8  CALL Kaine_.00407552    ; ================ ZwReadFile (récupère le serial)
004040AD  OR EAX,EAX
004040AF  JNZ SHORT Kaine_.004040D3
004040B1  CMP DWORD PTR SS:[EBP-2C],0B
004040B5  JNZ SHORT Kaine_.004040D0
004040B7  MOV ECX,0A
004040BC  MOV ESI,DWORD PTR SS:[EBP-28]
004040BF  MOV EDI,Kaine_.0040C9EC
004040C4  REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]  ; ===== copie le serial

004040C6  CALL Kaine_.004040EC  ; ======================== Appel de la VM

004040CB  MOV DWORD PTR DS:[40CA34],EAX  ; ============ doit être égal à 1
004040D0  NOP
004040D1  JMP SHORT Kaine_.004040D3
004040D3  PUSH DWORD PTR SS:[EBP-28]
004040D6  CALL Kaine_.00407570   ;====================== ExFreePool
004040DB  JMP SHORT Kaine_.004040DF
004040DD  JMP SHORT Kaine_.004040DF
004040DF  PUSH DWORD PTR SS:[EBP-24]
004040E2  CALL Kaine_.00407564  ; ====================== ZwClose
004040E7  JMP SHORT Kaine_.004040E9
004040E9  POPAD
004040EA  LEAVE
004040EB  RETN
```



JOUR 9 : Dimanche 29 mai 2005 : 3h15 !

Week end ou pas…je dois terminer le premier ! j'ai enfin chopé le pcode de la VM de Kaine.sys…il faut le comprendre !! mais là-dessus, je dois avoir une demi journée de retard !Je galère sur certains opcodes… ils utilisent des paramètres dont je ne saisis pas l'utilité…

11.3.1 La VM

La VM est donc le cœur de ce crackme. Elle exécute un pseudo-code qui teste la validité du serial contenu dans le fichier C:\License.key. Kaine.sys est très obfusqué et la VM n'en réchappe pas. Pire, le pseudo-code est également ofusqué !! Commençons par voir le fonctionnement de la VM.

Pour pouvoir contrôler des instructions, la VM doit disposer de registres pour manipuler des adresses. Ici , la VM dispose de ce que l'on peut appeler un petit CONTEXT dont voici la structure :

+ 0	R0
+ 4	R1
+ 8	R2

+C	R3
+10	R4
+14	R5
+18	R6
+1C	R7
+20	R8
+24	
+28	
+2C	
+30	
+34	EIP pointeur d'instruction
+38	ESP pointeur de pile
+3C	COMPARISON FLAG (armé s'il y a égalité lors d'un cmp)
+3D	ZERO FLAG (armé si le registre utilisé passe à zéro)
+3E	Début de la pile
+A2	Fin de la pile

Les registres Ri sont utilisés pour stocker des valeurs. Dans la suite, j'utiliserai les notations Ri(8) et Ri(16) pour désigner le BYTE ou le WORD et Ri pour désigner le DWORD.

Avant de proposer le code de la VM en clair, je vais vous présenter les différents opcodes dans l'ordre d'arrivée dans la VM . Ils sont au nombre de 20 et recouvre la majorité des instructions de base de l'asm :

OPCODE 17 : (mov Reg, dword)

syntaxe : 17 0i DWORD (avec i, numéro du registre à utiliser)

exemple : 17 01 4C 68 59 88

Voici ce que fait cette instruction :

```
17 01 4C 68 59 88
mov R1, 8859684Ch
add R1, 77E72856h
```

Cette instruction affecte le dword spécifié au registre et décrypte le dword à l'aide d'une addition.

OPCODE 56 : (mov reg, reg)

syntaxe : 56 xx 0i yy 0j (avec i et j, numéros des registres à utiliser)

exemple : 56 01 05 04 02

Voici ce que fait cette instruction :

```
56 01 05 04 02
mov BYTE PTR [R5], R2(8)
```

Cette instruction gére les adressages directs et indirects. Les indices xx et yy permettent de préciser quel type d'adressage utiliser sur le registre.

Si xx = 04, on utilise Ri(8)
Si xx = 05, on utilise Ri(16)
Si xx = 00, on utilise Ri

Si xx = 01, on utilise BYTE PTR [Ri]
Si xx = 02, on utilise WORD PTR [Ri]
Si xx = 03, on utilise DWORD PTR [Ri]

OPCODE 89 : (push reg)

syntaxe : 89 0i (avec i, numéro du registre à utiliser)

exemple : 89 01
Voici ce que fait cette instruction :

OPCODE 33 : (pop reg)

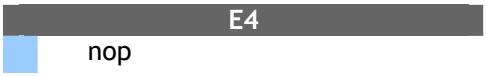
syntaxe : 33 0i (avec i, numéro du registre à utiliser)

exemple : 33 01
Voici ce que fait cette instruction :

OPCODE E4 : (nop)

syntaxe : E4

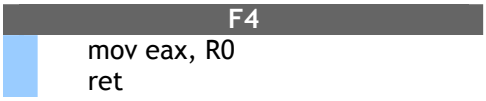
Voici ce que fait cette instruction :



OPCODE F4 : (ret)

syntaxe : F4

Voici ce que fait cette instruction :



OPCODE A0 : (xor reg, reg)

syntaxe : A0 xx 0i yy 0j (avec i et j, numéros des registres à utiliser)

exemple : A0 00 01 00 02
Voici ce que fait cette instruction :

xor R1,R2

Effectue l'opération arithmétique xor sur la valeur cible suivant le mode d'adressage choisi à l'aide de xx et yy.

OPCODE 65 : (inc reg)

syntaxe : 65 0i (avec i, numéro du registre à utiliser)

exemple : 65 01

Voici ce que fait cette instruction :

89 01
inc R1

Arme le ZERO FLAG en cas d'annulation de R1

OPCODE 99 : (dec reg)

syntaxe : 99 0i (avec i, numéro du registre à utiliser)

exemple : 99 01

Voici ce que fait cette instruction :

99 01
dec R1

Arme le ZERO FLAG en cas d'annulation de R1

OPCODE 97 : (jne)

syntaxe : 97 adresse-dword

exemple : 97 16 EB 51 DE

Voici ce que fait cette instruction :

97 16 EB 51 DE
jne

Teste le ZERO FLAG. S'il n'est pas armé, saute à l'adresse indiquée après l'avoir décrypté (en y ajoutant 21AE13F7h).

OPCODE 96 : (je)

syntaxe : 96 adresse-dword

exemple : 96 16 EB 51 DE

Voici ce que fait cette instruction :

96 16 EB 51 DE
je

Teste le ZERO FLAG. S'il est armé, saute à l'adresse indiquée après l'avoir décrypté (en y ajoutant BF6ADB9Ah).

OPCODE E8 : (add reg, reg)

syntaxe : E8 xx 0i yy 0j (avec i et j, numéros des registres à utiliser)

exemple : E8 00 01 00 02

Voici ce que fait cette instruction :

```
E8 00 01 00 02
    add R1,R2
```

Effectue l'opération arithmétique add sur la valeur cible suivant le mode d'adressage choisi à l'aide de xx et yy.

OPCODE F2 : (sub reg, reg)

syntaxe : F2 xx 0i yy 0j (avec i et j, numéros des registres à utiliser)

exemple : F2 00 01 00 02

Voici ce que fait cette instruction :

```
F2 00 01 00 02
    sub R1,R2
```

Effectue l'opération arithmétique sub sur la valeur cible suivant le mode d'adressage choisi à l'aide de xx et yy.

OPCODE D3 : (add reg, dword)

syntaxe : D3 0i DWORD (avec i, numéro du registre à utiliser)

exemple : D3 01 A9 D5 F8 31

Voici ce que fait cette instruction :

```
D3 01 A9 D5 F8 31
    add R1, 31F8D5A9h
    sub R1, 31F8D5A3h
```

Ajoute la valeur spécifiée à Ri et retranche à cette valeur le nombre 31F8D5A3h.

OPCODE DF : (sub reg, dword)

syntaxe : DF 0i DWORD (avec i, numéro du registre à utiliser)

exemple : DF 04 4E A9 5E 28

Voici ce que fait cette instruction :

```
DF 04 4E A9 5E 28
    sub R4, 285EA94Eh
    add R1, D7A156E2h
```

Retranche la valeur spécifiée à Ri et ajoute à cette valeur le nombre D7A156E2h.

OPCODE CE : (cmp reg, reg)

syntaxe : CE xx 0i yy 0j (avec i, numéro du registre à utiliser)

exemple : CE 04 00 04 04

Voici ce que fait cette instruction :

```
CE 04 00 04 04
```

```
cmp R0(8), R4(8)
```

Effectue une comparaison entre deux registres. Le mode d'adressage est défini par xx et yy. S'il y a égalité, le ZERO FLAG est armé.

OPCODE 57 : (je)

syntaxe : 57 adresse-dword (avec i, numéro du registre à utiliser)

exemple : 57 78 A4 0C 8B

Voici ce que fait cette instruction :

```
57 78 A4 0C 8B
mov ecx, 8B0CA478h
add ecx, 74F364Cah
je ecx
```

Teste le COMPARISON FLAG. S'il est armé, saute à l'adresse indiquée par le dword.(en y ajoutant le dword 74F364CAh)

OPCODE 67 : (jne)

syntaxe : 67 adresse-dword (avec i, numéro du registre à utiliser)

exemple : 67 78 A4 0C 8B

Voici ce que fait cette instruction :

```
67 78 A4 0C 8B
mov ecx, 8B0CA478h
add ecx, C846EFDAh
je ecx
```

Teste le COMPARISON FLAG. S'il n'est pas armé, saute à l'adresse indiquée par le dword.(en y ajoutant le dword C846EFDAh)

OPCODE 78 : (mul reg, dword)

syntaxe : 78 0i dword (avec i, numéro du registre à utiliser)

exemple : 78 08 66 45 62 15

Voici ce que fait cette instruction :

```
78 08 66 45 62 15
mov eax, 15624566h
sub eax, 15624562h
mul R8,eax
```

Effectue une multiplication du contenu d'un registre par un dword (décrypté au préalable)

OPCODE CC : (jmp)

syntaxe : CC dword

exemple : CC 05 00 00 00

Voici ce que fait cette instruction :

```
CC 05 00 00 00
jmp $+5
```

Voici maintenant la VM de Kaine.sys sans les obfuscations. J'ai rippé la majorité des instructions. Il se peut aussi qu'il y ait quelques différences mineures. Cela dit, ce code fonctionne exactement comme celui utilisé dans le driver Kaine.sys. Vous remarquerez la phase d'initialisation suivie d'une série de 20 tests pour identifier l'opcode en cours.

Virtual Machine de Kaine.sys

```
.586
.Model Flat ,StdCall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32.lib


.Data

; ******************************** CONTEXT de la VM de Kaine.sys
R0      DWORD   0
R1      DWORD   0
R2      DWORD   0
R3      DWORD   0
R4      DWORD   0
R5      DWORD   0
R6      DWORD   0
R7      DWORD   0
R8      DWORD   0
EIP_    DWORD   0
ESP_    DWORD   0
CmpF    BYTE    0
ZF_     BYTE    0
; ******************************** pile de 64 bytes de la VM

STACK_  BYTE   64 DUP(0)

; ******************************** Programme en pseudo-code (exemple : Junk code 1 issu de Kaine.sys)

PCODE   BYTE     89h, 00h
        BYTE     17h, 00h, 04h, 8Ch, 2Ch, 30h
        BYTE    0E8h, 00h, 00h, 00h, 007h
        BYTE     89h, 03h
        BYTE     17h, 03h, 64h, 35h, 4Dh, 81h
        BYTE    0A0h, 00h, 03h, 00h, 02h
        BYTE     33h, 03h
        BYTE    0F2h, 00h, 00h, 00h, 04h
        BYTE    0D3h, 00h,0EFh,0CFh, 41h, 95h
        BYTE     33h, 00h

.Code
Main:
 pushad
 mov eax, offset PCODE
 mov EIP_, eax
 mov eax, offset STACK_
 add eax,64h
 mov ESP_, eax

Test_opcodes:
 mov edi, EIP_
 cmp BYTE PTR [EDI], 17h
 jne @F
```

```
call OPCODE_17
test eax, eax
je ERROR
jmp Test_opcodes
@@:
cmp BYTE PTR [EDI], 56h
jne @F
call OPCODE_56
test eax, eax
je ERROR
jmp Test_opcodes

@@:
cmp BYTE PTR [EDI], 89h
jne @F
call OPCODE_89
test eax, eax
je ERROR
jmp Test_opcodes
@@:
cmp BYTE PTR [EDI], 33h
jne @F
call OPCODE_33
test eax, eax
je ERROR
jmp Test_opcodes
@@:
cmp BYTE PTR [EDI], 0E4h
jne @F
call OPCODE_E4
test eax, eax
je ERROR
jmp Test_opcodes
@@:
cmp BYTE PTR [EDI], 0F4h
jne @F
call OPCODE_17
test eax, eax
je ERROR
jmp Test_opcodes
@@:
cmp BYTE PTR [EDI], 0A0h
jne @F
call OPCODE_A0
test eax, eax
je ERROR
jmp Test_opcodes
@@:
cmp BYTE PTR [EDI], 65h
jne @F
call OPCODE_65
test eax, eax
je ERROR
jmp Test_opcodes
@@:
cmp BYTE PTR [EDI], 99h
jne @F
call OPCODE_99
test eax, eax
je ERROR
jmp Test_opcodes
```

```
@@:
cmp BYTE PTR [EDI], 97h
 jne @F
 call OPCODE_97
 test eax, eax
 je ERROR
 jmp Test_opcodes
@@:
cmp BYTE PTR [EDI], 96h
 jne @F
 call OPCODE_96
 test eax, eax
 je ERROR
 jmp Test_opcodes
@@:
cmp BYTE PTR [EDI], 0E8h
 jne @F
 call OPCODE_E8
 test eax, eax
 je ERROR
 jmp Test_opcodes
@@:
cmp BYTE PTR [EDI], 0F2h
 jne @F
 call OPCODE_F2
 test eax, eax
 je ERROR
 jmp Test_opcodes
@@:
cmp BYTE PTR [EDI], 0D3h
 jne @F
 call OPCODE_D3
 test eax, eax
 je ERROR
 jmp Test_opcodes
@@:
cmp BYTE PTR [EDI], 0DFh
 jne @F
 call OPCODE_DF
 test eax, eax
 je ERROR
 jmp Test_opcodes
@@:
cmp BYTE PTR [EDI], 0CEh
 jne @F
 call OPCODE_CE
 test eax, eax
 je ERROR
 jmp Test_opcodes
@@:
cmp BYTE PTR [EDI], 57h
 jne @F
 call OPCODE_57
 test eax, eax
 je ERROR
 jmp Test_opcodes
@@:
cmp BYTE PTR [EDI], 67h
 jne @F
 call OPCODE_67
 test eax, eax
```

```
 je ERROR
 jmp Test_opcodes
@@:
cmp BYTE PTR [EDI], 78h
 jne @F
 call OPCODE_78
 test eax, eax
 je ERROR
 jmp Test_opcodes
@@:
cmp BYTE PTR [EDI], 0CCh
 jne @F
 call OPCODE_CC
 test eax, eax
 je ERROR
 jmp Test_opcodes

@@:
 ; ************************* fin du pseudo-programme
 push 0
 call ExitProcess

ERROR:
 push 0
 call ExitProcess



; ************************************************************* OPCODE 17

OPCODE_17 PROC
 pushad
 inc edi
 movzx ecx, BYTE PTR [edi]
 cmp ecx, 0
 jl @F
 cmp ecx, 8
 jg @F
 lea esi, DWORD PTR [ecx*4 + offset R0]
 inc edi
 mov eax, DWORD PTR [edi]
 add eax, 77E72856h
 add edi,4
 mov dword ptr [esi], eax
 mov EIP_, edi
 popad
 mov eax, 1
 ret
@@:
 popad
 xor eax, eax
 ret
OPCODE_17 ENDP

; ************************************************************* OPCODE 56


OPCODE_56 PROC
 pushad
 inc edi
 movzx eax, byte ptr [edi]
```

```
 movzx ecx, byte ptr [edi+1]
 cmp eax, 0
 jl exit_56
 cmp eax, 5
 jg exit_56
 cmp ecx, 0
 jl exit_56
 cmp ecx, 8
 jg exit_56
 lea esi, dword ptr [ecx*4+offset R0]
 add edi,2
 movzx edx, byte ptr [edi]
 movzx ecx, byte ptr [edi+1]
 cmp edx, 0
 jl exit_56
 cmp edx, 5
 jg exit_56
 cmp ecx, 0
 jl exit_56
 cmp ecx, 8
 jg exit_56
 lea edi, dword ptr [ecx*4+offset R0]
 xchg esi, edi
; ***************************** tester eax et edx : soit eax=edx, soit |eax-edx| = 3
 cmp eax, edx
 jb @F
 cmp eax, edx
 je TESTER_EAX
 sub eax, edx
 cmp eax, 3
 je TESTER_EAX
 jmp exit_56
@@:
 cmp eax, edx
 je TESTER_EAX
 sub edx, eax
 cmp edx,3
 je TESTER_EAX
 jmp exit_56
; ************************** Tester le type de registre utilisé : DWORD, WORD ou BYTE
TESTER_EAX:
 push edx
 pop ecx
 cmp eax, 0
 je DWORD_56
 cmp eax, 1
 je BYTE_56
 cmp eax, 2
 je WORD_56
 cmp eax, 3
 je DWORD_56
 cmp eax, 4
 je BYTE_56
 jmp WORD_56
; ************************************ gestion des dword
DWORD_56:
 cmp eax, 0
 jne dword_xx_indirect
dword_xx_direct:
 cmp ecx, 0
 jne dword_xx_direct_yy_indirect
```

```
dword_xx_direct_yy_direct:
 push edi
 pop edx
 mov eax, dword ptr [esi]          ; mov Ri, Rj
 mov dword ptr [edx], eax
 jmp fin_56

dword_xx_direct_yy_indirect:
 push edi
 pop edx
 mov eax, dword ptr [esi]          ; mov Ri, dword ptr [Rj]
 mov eax, dword ptr [eax]
 mov dword ptr [edx], eax
 jmp fin_56

dword_xx_indirect:
 cmp ecx, 0
 jne dword_xx_indirect_yy_indirect
dword_xx_indirect_yy_direct:
 push edi
 pop eax
 mov esi, dword ptr [esi]          ; mov dword ptr [Ri], Rj
 mov eax, dword ptr [eax]
 mov dword ptr [eax], esi
 jmp fin_56
dword_xx_indirect_yy_indirect:
 push edi
 pop eax
 mov eax, dword ptr [eax]          ; mov dword ptr [Ri], dword ptr [Rj]
 mov esi, dword ptr [esi]
 mov esi, dword ptr [esi]
 mov dword ptr [eax], esi
 jmp fin_56
; ************************************ gestion des byte

BYTE_56:
 cmp eax, 4
 jne byte_xx_indirect
byte_xx_direct:
 cmp ecx, 4
 jne byte_xx_direct_yy_indirect
byte_xx_direct_yy_direct:
 push edi
 pop edx
 mov al, byte ptr [esi]            ; mov Ri(8), Rj(8)
 mov byte ptr [edx], al
 jmp fin_56

byte_xx_direct_yy_indirect:
 push edi
 pop edx
 mov eax, dword ptr [esi]          ; mov Ri(8), byte ptr [Rj]
 mov al, byte ptr [eax]
 mov byte ptr [edx], al
 jmp fin_56

byte_xx_indirect:
 cmp ecx, 4
 jne byte_xx_indirect_yy_indirect
byte_xx_indirect_yy_direct:
```

```
 push edi
 pop eax
 mov bl, byte ptr [esi]            ; mov byte ptr [Ri], Rj(8)
 mov eax, dword ptr [eax]
 mov byte ptr [eax], bl
 jmp fin_56
byte_xx_indirect_yy_indirect:
 push edi
 pop eax
 mov eax, dword ptr [eax]          ; mov byte ptr [Ri], byte ptr [Rj]
 mov esi, dword ptr [esi]
 mov bl, byte ptr [esi]
 mov byte ptr [eax], bl
 jmp fin_56


; ************************************ gestion des word

WORD_56:
 cmp eax, 5
 jne word_xx_indirect
word_xx_direct:
 cmp ecx, 5
 jne word_xx_direct_yy_indirect
word_xx_direct_yy_direct:
 push edi
 pop edx
 mov ax, word ptr [esi]            ; mov Ri(8), Rj(8)
 mov word ptr [edx], ax
 jmp fin_56
word_xx_direct_yy_indirect:
 push edi
 pop edx
 mov eax, dword ptr [esi]          ; mov Ri(8), byte ptr [Rj]
 mov ax, word ptr [eax]
 mov word ptr [edx], ax
 jmp fin_56
word_xx_indirect:
 cmp ecx, 5
 jne word_xx_indirect_yy_indirect
word_xx_indirect_yy_direct:
 push edi
 pop eax
 mov bx, word ptr [esi]            ; mov byte ptr [Ri], Rj(8)
 mov eax, dword ptr [eax]
 mov word ptr [eax], bx
 jmp fin_56
word_xx_indirect_yy_indirect:
 push edi
 pop eax
 mov eax, dword ptr [eax]          ; mov byte ptr [Ri], byte ptr [Rj]
 mov esi, dword ptr [esi]
 mov bx, word ptr [esi]
 mov word ptr [eax], bx
 jmp fin_56

fin_56:
 add EIP_, 5
 popad
 mov eax,1
 ret
```

```
exit_56:
 popad
 xor eax, eax
 ret
OPCODE_56 ENDP

; ************************************************************* OPCODE 89

OPCODE_89 PROC
 pushad
 sub ESP_, 4
 cmp ESP_, offset STACK_
 jb @F
 mov eax, ESP_
 movzx ecx, byte ptr [edi+1]
 add edi, 2
 mov EIP_, edi
 cmp ecx, 0
 jl @F
 cmp ecx, 8
 jg @F
 lea esi, dword ptr [ecx*4 + offset R0]
 mov esi, dword ptr [esi]
 mov dword ptr [eax], esi
 popad
 mov eax, 1
 ret
@@:
 popad
 xor eax, eax
 ret
OPCODE_89 ENDP

; ************************************************************* OPCODE 33

OPCODE_33 PROC
 pushad
 sub ESP_, 4
 cmp ESP_, offset STACK_
 jb @F
 mov eax, ESP_
 movzx ecx, byte ptr [edi+1]
 add edi, 2
 mov EIP_, edi
 cmp ecx, 0
 jl @F
 cmp ecx, 8
 jg @F
 lea esi, dword ptr [ecx*4 + offset R0]
 mov eax, dword ptr [eax]
 mov dword ptr [esi], eax
 add ESP_, 4
 mov eax, offset STACK_
 add eax, 64h
 cmp ESP_, eax
 ja @F
 popad
 mov eax, 1
 ret
@@:
 popad
```

```
 xor eax, eax
 ret
OPCODE_33 ENDP

; ************************************************************* OPCODE E4

OPCODE_E4 PROC
 add EIP_, 1
 ret
OPCODE_E4 ENDP

; ************************************************************* OPCODE F4


OPCODE_F4 PROC

 popad
 pop ebp
 mov eax, R0
 ret
OPCODE_F4 ENDP

; ************************************************************* OPCODE A0

OPCODE_A0 PROC
 pushad
 inc edi
 movzx eax, byte ptr [edi]
 movzx ecx, byte ptr [edi+1]
 cmp eax, 0
 jl exit_A0
 cmp eax, 5
 jg exit_A0
 cmp ecx, 0
 jl exit_A0
 cmp ecx, 8
 jg exit_A0
 lea esi, dword ptr [ecx*4+offset R0]
 add edi,2
 movzx edx, byte ptr [edi]
 movzx ecx, byte ptr [edi+1]
 cmp edx, 0
 jl exit_A0
 cmp edx, 5
 jg exit_A0
 cmp ecx, 0
 jl exit_A0
 cmp ecx, 8
 jg exit_A0
 lea edi, dword ptr [ecx*4+offset R0]
 xchg esi, edi
; ***************************** tester eax et edx : soit eax=edx, soit |eax-edx| = 3
 cmp eax, edx
 jb @F
 cmp eax, edx
 je TESTER_EAX_A0
 sub eax, edx
 cmp eax, 3
 je TESTER_EAX_A0
 jmp exit_A0
@@:
```

```
 cmp eax, edx
 je TESTER_EAX_A0
 sub edx, eax
 cmp edx,3
 je TESTER_EAX_A0
 jmp exit_A0
; ************************** Tester le type de registre utilisé : DWORD, WORD ou BYTE
TESTER_EAX_A0:
 push edx
 pop ecx
 cmp eax, 0
 je DWORD_A0
 cmp eax, 1
 je BYTE_A0
 cmp eax, 2
 je WORD_A0
 cmp eax, 3
 je DWORD_A0
 cmp eax, 4
 je BYTE_A0
 jmp WORD_A0
; ************************************ gestion des dword
DWORD_A0:
 cmp eax, 0
 jne dword_xx_indirect_a0
dword_xx_direct_a0:
 cmp ecx, 0
 jne dword_xx_direct_yy_indirect_a0

dword_xx_direct_yy_direct_a0:
 push edi
 pop edx
 mov eax, dword ptr [esi]          ; xor Ri, Rj
 xor dword ptr [edx], eax
 jmp fin_A0

dword_xx_direct_yy_indirect_a0:
 push edi
 pop edx
 mov eax, dword ptr [esi]          ; xor Ri, dword ptr [Rj]
 mov eax, dword ptr [eax]
 xor dword ptr [edx], eax
 jmp fin_A0

dword_xx_indirect_a0:
 cmp ecx, 0
 jne dword_xx_indirect_yy_indirect_a0
dword_xx_indirect_yy_direct_a0:
 push edi
 pop eax
 mov esi, dword ptr [esi]          ; xor dword ptr [Ri], Rj
 mov eax, dword ptr [eax]
 xor dword ptr [eax], esi
 jmp fin_A0
dword_xx_indirect_yy_indirect_a0:
 push edi
 pop eax
 mov eax, dword ptr [eax]          ; xor dword ptr [Ri], dword ptr [Rj]
 mov esi, dword ptr [esi]
 mov esi, dword ptr [esi]
 xor dword ptr [eax], esi
```

```
 jmp fin_A0
; ************************************ gestion des byte

BYTE_A0:
 cmp eax, 4
 jne byte_xx_indirect_a0
byte_xx_direct_a0:
 cmp ecx, 4
 jne byte_xx_direct_yy_indirect_a0
byte_xx_direct_yy_direct_a0:
 push edi
 pop edx
 mov al, byte ptr [esi]            ; xor Ri(8), Rj(8)
 xor byte ptr [edx], al
 jmp fin_A0

byte_xx_direct_yy_indirect_a0:
 push edi
 pop edx
 mov eax, dword ptr [esi]          ; xor Ri(8), byte ptr [Rj]
 mov al, byte ptr [eax]
 xor byte ptr [edx], al
 jmp fin_A0

byte_xx_indirect_a0:
 cmp ecx, 4
 jne byte_xx_indirect_yy_indirect_a0
byte_xx_indirect_yy_direct_a0:
 push edi
 pop eax
 mov bl, byte ptr [esi]            ; xor byte ptr [Ri], Rj(8)
 mov eax, dword ptr [eax]
 xor byte ptr [eax], bl
 jmp fin_A0
byte_xx_indirect_yy_indirect_a0:
 push edi
 pop eax
 mov eax, dword ptr [eax]          ; xor byte ptr [Ri], byte ptr [Rj]
 mov esi, dword ptr [esi]
 mov bl, byte ptr [esi]
 xor byte ptr [eax], bl
 jmp fin_A0



; ************************************ gestion des word

WORD_A0:
 cmp eax, 5
 jne word_xx_indirect_a0
word_xx_direct_a0:
 cmp ecx, 5
 jne word_xx_direct_yy_indirect_a0
word_xx_direct_yy_direct_a0:
 push edi
 pop edx
 mov ax, word ptr [esi]            ; xor Ri(8), Rj(8)
 xor word ptr [edx], ax
 jmp fin_A0
word_xx_direct_yy_indirect_a0:
 push edi
```

```
 pop edx
 mov eax, dword ptr [esi]          ; xor Ri(8), byte ptr [Rj]
 mov ax, word ptr [eax]
 xor word ptr [edx], ax
 jmp fin_A0
word_xx_indirect_a0:
 cmp ecx, 5
 jne word_xx_indirect_yy_indirect_a0
word_xx_indirect_yy_direct_a0:
 push edi
 pop eax
 mov bx, word ptr [esi]            ; xor byte ptr [Ri], Rj(8)
 mov eax, dword ptr [eax]
 xor word ptr [eax], bx
 jmp fin_A0
word_xx_indirect_yy_indirect_a0:
 push edi
 pop eax
 mov eax, dword ptr [eax]          ; xor byte ptr [Ri], byte ptr [Rj]
 mov esi, dword ptr [esi]
 mov bx, word ptr [esi]
 xor word ptr [eax], bx
 jmp fin_A0

fin_A0:
 add EIP_, 5
 popad
 mov eax,1
 ret
exit_A0:
 popad
 xor eax, eax
 ret
OPCODE_A0 ENDP

; ************************************************************** OPCODE 65

OPCODE_65 PROC
 pushad
 movzx ecx, byte ptr [edi+1]
 cmp ecx,0
 jl @F
 cmp ecx,8
 jg @F
 lea esi, dword ptr [ecx*4+ offset R0]
 inc dword ptr [esi]
 mov eax, dword ptr [esi]
 xor ebx, ebx
 test eax, eax
 sete bl
 mov ZF_, bl
 add EIP_, 2
 popad
 mov eax, 1
 ret
@@:
 popad
 xor eax, eax
 ret
OPCODE_65 ENDP
```

```
; ************************************************************* OPCODE 99

OPCODE_99 PROC
 pushad
 movzx ecx, byte ptr [edi+1]
 cmp ecx,0
 jl @F
 cmp ecx,8
 jg @F
 lea esi, dword ptr [ecx*4+ offset R0]
 dec dword ptr [esi]
 mov eax, dword ptr [esi]
 xor ebx, ebx
 test eax, eax
 sete bl
 mov ZF_, bl
 add EIP_, 2
 popad
 mov eax, 1
 ret
@@:
 popad
 xor eax, eax
 ret
OPCODE_99 ENDP


; ************************************************************* OPCODE 97

OPCODE_97 PROC
 pushad
 cmp ZF_, 0
 jnz @F
 mov eax, edi
 mov ecx, DWORD PTR [edi+1]
 add ecx, 21AE13F7h
 add eax, ecx
 mov EIP_, eax
@@:
 popad
 mov eax,1
 ret
OPCODE_97 ENDP

; ************************************************************* OPCODE 96

OPCODE_96 PROC
 pushad
 cmp ZF_, 1
 jnz @F
 mov eax, edi
 mov ecx, DWORD PTR [edi+1]
 sub ecx, 0BF6ADB9Ah
 add eax, ecx
 mov EIP_, eax
@@:
 popad
 mov eax,1
 ret
OPCODE_96 ENDP
; ************************************************************* OPCODE E8
```

```
OPCODE_E8 PROC
 pushad
 inc edi
 movzx eax, byte ptr [edi]
 movzx ecx, byte ptr [edi+1]
 cmp eax, 0
 jl exit_E8
 cmp eax, 5
 jg exit_E8
 cmp ecx, 0
 jl exit_E8
 cmp ecx, 8
 jg exit_E8
 lea esi, dword ptr [ecx*4+offset R0]
 add edi,2
 movzx edx, byte ptr [edi]
 movzx ecx, byte ptr [edi+1]
 cmp edx, 0
 jl exit_E8
 cmp edx, 5
 jg exit_E8
 cmp ecx, 0
 jl exit_E8
 cmp ecx, 8
 jg exit_E8
 lea edi, dword ptr [ecx*4+offset R0]
 xchg esi, edi
; ***************************** tester eax et edx : soit eax=edx, soit |eax-edx| = 3
 cmp eax, edx
 jb @F
 cmp eax, edx
 je TESTER_EAX_E8
 sub eax, edx
 cmp eax, 3
 je TESTER_EAX_E8
 jmp exit_E8
@@:
 cmp eax, edx
 je TESTER_EAX_E8
 sub edx, eax
 cmp edx,3
 je TESTER_EAX_E8
 jmp exit_E8
; ************************** Tester le type de registre utilisé : DWORD, WORD ou BYTE
TESTER_EAX_E8:
 push edx
 pop ecx
 cmp eax, 0
 je DWORD_E8
 cmp eax, 1
 je BYTE_E8
 cmp eax, 2
 je WORD_E8
 cmp eax, 3
 je DWORD_E8
 cmp eax, 4
 je BYTE_E8
 jmp WORD_E8
; ************************************* gestion des dword
DWORD_E8:
```

```
 cmp eax, 0
 jne dword_xx_indirect_e8
dword_xx_direct_e8:
 cmp ecx, 0
 jne dword_xx_direct_yy_indirect_e8

dword_xx_direct_yy_direct_e8:
 push edi
 pop edx
 mov eax, dword ptr [esi]          ; add Ri, Rj
 add dword ptr [edx], eax
 jmp fin_E8

dword_xx_direct_yy_indirect_e8:
 push edi
 pop edx
 mov eax, dword ptr [esi]          ; add Ri, dword ptr [Rj]
 mov eax, dword ptr [eax]
 add dword ptr [edx], eax
 jmp fin_E8

dword_xx_indirect_e8:
 cmp ecx, 0
 jne dword_xx_indirect_yy_indirect_e8
dword_xx_indirect_yy_direct_e8:
 push edi
 pop eax
 mov esi, dword ptr [esi]          ; add dword ptr [Ri], Rj
 mov eax, dword ptr [eax]
 add dword ptr [eax], esi
 jmp fin_E8
dword_xx_indirect_yy_indirect_e8:
 push edi
 pop eax
 mov eax, dword ptr [eax]          ; add dword ptr [Ri], dword ptr [Rj]
 mov esi, dword ptr [esi]
 mov esi, dword ptr [esi]
 add dword ptr [eax], esi
 jmp fin_E8
; ************************************ gestion des byte

BYTE_E8:
 cmp eax, 4
 jne byte_xx_indirect_e8
byte_xx_direct_e8:
 cmp ecx, 4
 jne byte_xx_direct_yy_indirect_e8
byte_xx_direct_yy_direct_e8:
 push edi
 pop edx
 mov al, byte ptr [esi]            ; add Ri(8), Rj(8)
 add byte ptr [edx], al
 jmp fin_E8

byte_xx_direct_yy_indirect_e8:
 push edi
 pop edx
 mov eax, dword ptr [esi]          ; add Ri(8), byte ptr [Rj]
 mov al, byte ptr [eax]
 add byte ptr [edx], al
 jmp fin_E8
```

```
byte_xx_indirect_e8:
 cmp ecx, 4
 jne byte_xx_indirect_yy_indirect_e8
byte_xx_indirect_yy_direct_e8:
 push edi
 pop eax
 mov bl, byte ptr [esi]            ; add byte ptr [Ri], Rj(8)
 mov eax, dword ptr [eax]
 add byte ptr [eax], bl
 jmp fin_E8
byte_xx_indirect_yy_indirect_e8:
 push edi
 pop eax
 mov eax, dword ptr [eax]          ; add byte ptr [Ri], byte ptr [Rj]
 mov esi, dword ptr [esi]
 mov bl, byte ptr [esi]
 add byte ptr [eax], bl
 jmp fin_E8


; ************************************ gestion des word

WORD_E8:
 cmp eax, 5
 jne word_xx_indirect_e8
word_xx_direct_e8:
 cmp ecx, 5
 jne word_xx_direct_yy_indirect_e8
word_xx_direct_yy_direct_e8:
 push edi
 pop edx
 mov ax, word ptr [esi]            ; add Ri(8), Rj(8)
 add word ptr [edx], ax
 jmp fin_E8
word_xx_direct_yy_indirect_e8:
 push edi
 pop edx
 mov eax, dword ptr [esi]          ; add Ri(8), byte ptr [Rj]
 mov ax, word ptr [eax]
 add word ptr [edx], ax
 jmp fin_E8
word_xx_indirect_e8:
 cmp ecx, 5
 jne word_xx_indirect_yy_indirect_e8
word_xx_indirect_yy_direct_e8:
 push edi
 pop eax
 mov bx, word ptr [esi]            ; add byte ptr [Ri], Rj(8)
 mov eax, dword ptr [eax]
 add word ptr [eax], bx
 jmp fin_E8
word_xx_indirect_yy_indirect_e8:
 push edi
 pop eax
 mov eax, dword ptr [eax]          ; add byte ptr [Ri], byte ptr [Rj]
 mov esi, dword ptr [esi]
 mov bx, word ptr [esi]
 add word ptr [eax], bx
 jmp fin_E8
```

```
fin_E8:
 add EIP_, 5
 popad
 mov eax,1
 ret
exit_E8:
 popad
 xor eax, eax
 ret



OPCODE_E8 ENDP
; ************************************************************* OPCODE F2

OPCODE_F2 PROC
 pushad
 inc edi
 movzx eax, byte ptr [edi]
 movzx ecx, byte ptr [edi+1]
 cmp eax, 0
 jl exit_F2
 cmp eax, 5
 jg exit_F2
 cmp ecx, 0
 jl exit_F2
 cmp ecx, 8
 jg exit_F2
 lea esi, dword ptr [ecx*4+offset R0]
 add edi,2
 movzx edx, byte ptr [edi]
 movzx ecx, byte ptr [edi+1]
 cmp edx, 0
 jl exit_F2
 cmp edx, 5
 jg exit_F2
 cmp ecx, 0
 jl exit_F2
 cmp ecx, 8
 jg exit_F2
 lea edi, dword ptr [ecx*4+offset R0]
 xchg esi, edi
; ***************************** tester eax et edx : soit eax=edx, soit |eax-edx| = 3
 cmp eax, edx
 jb @F
 cmp eax, edx
 je TESTER_EAX_F2
 sub eax, edx
 cmp eax, 3
 je TESTER_EAX_F2
 jmp exit_F2
@@:
 cmp eax, edx
 je TESTER_EAX_F2
 sub edx, eax
 cmp edx,3
 je TESTER_EAX_F2
 jmp exit_F2
; ************************** Tester le type de registre utilisé : DWORD, WORD ou BYTE
TESTER_EAX_F2:
 push edx
```

```
 pop ecx
 cmp eax, 0
 je DWORD_F2
 cmp eax, 1
 je BYTE_F2
 cmp eax, 2
 je WORD_F2
 cmp eax, 3
 je DWORD_F2
 cmp eax, 4
 je BYTE_F2
 jmp WORD_F2
; ************************************ gestion des dword
DWORD_F2:
 cmp eax, 0
 jne dword_xx_indirect_f2
dword_xx_direct_f2:
 cmp ecx, 0
 jne dword_xx_direct_yy_indirect_f2

dword_xx_direct_yy_direct_f2:
 push edi
 pop edx
 mov eax, dword ptr [esi]          ; sub Ri, Rj
 sub dword ptr [edx], eax
 jmp fin_F2

dword_xx_direct_yy_indirect_f2:
 push edi
 pop edx
 mov eax, dword ptr [esi]          ; sub Ri, dword ptr [Rj]
 mov eax, dword ptr [eax]
 sub dword ptr [edx], eax
 jmp fin_F2

dword_xx_indirect_f2:
 cmp ecx, 0
 jne dword_xx_indirect_yy_indirect_f2
dword_xx_indirect_yy_direct_f2:
 push edi
 pop eax
 mov esi, dword ptr [esi]          ; sub dword ptr [Ri], Rj
 mov eax, dword ptr [eax]
 sub dword ptr [eax], esi
 jmp fin_F2
dword_xx_indirect_yy_indirect_f2:
 push edi
 pop eax
 mov eax, dword ptr [eax]          ; sub dword ptr [Ri], dword ptr [Rj]
 mov esi, dword ptr [esi]
 mov esi, dword ptr [esi]
 sub dword ptr [eax], esi
 jmp fin_F2
; ************************************ gestion des byte

BYTE_F2:
 cmp eax, 4
 jne byte_xx_indirect_f2
byte_xx_direct_f2:
 cmp ecx, 4
 jne byte_xx_direct_yy_indirect_f2
```

```
byte_xx_direct_yy_direct_f2:
 push edi
 pop edx
 mov al, byte ptr [esi]            ; sub Ri(8), Rj(8)
 sub byte ptr [edx], al
 jmp fin_F2

byte_xx_direct_yy_indirect_f2:
 push edi
 pop edx
 mov eax, dword ptr [esi]          ; sub Ri(8), byte ptr [Rj]
 mov al, byte ptr [eax]
 sub byte ptr [edx], al
 jmp fin_F2

byte_xx_indirect_f2:
 cmp ecx, 4
 jne byte_xx_indirect_yy_indirect_f2
byte_xx_indirect_yy_direct_f2:
 push edi
 pop eax
 mov bl, byte ptr [esi]            ; sub byte ptr [Ri], Rj(8)
 mov eax, dword ptr [eax]
 sub byte ptr [eax], bl
 jmp fin_F2
byte_xx_indirect_yy_indirect_f2:
 push edi
 pop eax
 mov eax, dword ptr [eax]          ; sub byte ptr [Ri], byte ptr [Rj]
 mov esi, dword ptr [esi]
 mov bl, byte ptr [esi]
 sub byte ptr [eax], bl
 jmp fin_F2


; ************************************ gestion des word

WORD_F2:
 cmp eax, 5
 jne word_xx_indirect_f2
word_xx_direct_f2:
 cmp ecx, 5
 jne word_xx_direct_yy_indirect_f2
word_xx_direct_yy_direct_f2:
 push edi
 pop edx
 mov ax, word ptr [esi]            ; sub Ri(8), Rj(8)
 sub word ptr [edx], ax
 jmp fin_F2
word_xx_direct_yy_indirect_f2:
 push edi
 pop edx
 mov eax, dword ptr [esi]          ; sub Ri(8), byte ptr [Rj]
 mov ax, word ptr [eax]
 sub word ptr [edx], ax
 jmp fin_F2
word_xx_indirect_f2:
 cmp ecx, 5
 jne word_xx_indirect_yy_indirect_f2
word_xx_indirect_yy_direct_f2:
 push edi
```

```
 pop eax
 mov bx, word ptr [esi]            ; sub byte ptr [Ri], Rj(8)
 mov eax, dword ptr [eax]
 sub word ptr [eax], bx
 jmp fin_F2
word_xx_indirect_yy_indirect_f2:
 push edi
 pop eax
 mov eax, dword ptr [eax]          ; sub byte ptr [Ri], byte ptr [Rj]
 mov esi, dword ptr [esi]
 mov bx, word ptr [esi]
 sub word ptr [eax], bx
 jmp fin_F2

fin_F2:
 add EIP_, 5
 popad
 mov eax,1
 ret
exit_F2:
 popad
 xor eax, eax
 ret


OPCODE_F2 ENDP

; ************************************************************* OPCODE D3

OPCODE_D3 PROC
 pushad
 movzx ecx, byte ptr [edi+1]
 cmp ecx,0
 jl @F
 cmp ecx,8
 jg @F
 lea ebx, dword ptr [ecx*4+ offset R0]
 mov eax, dword ptr [edi+2]
 sub eax, 31F8D5A3h
 add dword ptr [ebx], eax
 add EIP_, 6
 popad
 mov eax, 1
 ret
@@:
 popad
 xor eax, eax
 ret
OPCODE_D3 ENDP


; ************************************************************* OPCODE DF

OPCODE_DF PROC
 pushad
 movzx ecx, byte ptr [edi+1]
 cmp ecx,0
 jl @F
 cmp ecx,8
 jg @F
 lea ebx, dword ptr [ecx*4+ offset R0]
```

```
 mov eax, dword ptr [edi+2]
 add eax, 0D7A156E2h
 sub dword ptr [ebx], eax
 add EIP_, 6
 popad
 mov eax, 1
 ret
@@:
 popad
 xor eax, eax
 ret

OPCODE_DF ENDP

; ************************************************************* OPCODE CE


OPCODE_CE PROC
 pushad
 inc edi
 movzx eax, byte ptr [edi]
 movzx ecx, byte ptr [edi+1]
 cmp eax, 0
 jl exit_CE
 cmp eax, 5
 jg exit_CE
 cmp ecx, 0
 jl exit_CE
 cmp ecx, 8
 jg exit_CE
 lea esi, dword ptr [ecx*4+offset R0]
 add edi,2
 movzx edx, byte ptr [edi]
 movzx ecx, byte ptr [edi+1]
 cmp edx, 0
 jl exit_CE
 cmp edx, 5
 jg exit_CE
 cmp ecx, 0
 jl exit_CE
 cmp ecx, 8
 jg exit_CE
 lea edi, dword ptr [ecx*4+offset R0]
 xchg esi, edi
; ************************** tester eax et edx : soit eax=edx, soit |eax-edx| = 3
 cmp eax, edx
 jb @F
 cmp eax, edx
 je TESTER_EAX_CE
 sub eax, edx
 cmp eax, 3
 je TESTER_EAX_CE
 jmp exit_CE
@@:
 cmp eax, edx
 je TESTER_EAX_CE
 sub edx, eax
 cmp edx,3
 je TESTER_EAX_CE
 jmp exit_CE
; ************************** Tester le type de registre utilisé : DWORD, WORD ou BYTE
```

```
TESTER_EAX_CE:
 push edx
 pop ecx
 cmp eax, 0
 je DWORD_CE
 cmp eax, 1
 je BYTE_CE
 cmp eax, 2
 je WORD_CE
 cmp eax, 3
 je DWORD_CE
 cmp eax, 4
 je BYTE_CE
 jmp WORD_CE
; ************************************ gestion des dword
DWORD_CE:
 cmp eax, 0
 jne dword_xx_indirect_ce
dword_xx_direct_ce:
 cmp ecx, 0
 jne dword_xx_direct_yy_indirect_ce

dword_xx_direct_yy_direct_ce:
 push edi
 pop edx
 mov eax, dword ptr [esi]          ; cmp Ri, Rj
 cmp dword ptr [edx], eax
 jmp fin_CE

dword_xx_direct_yy_indirect_ce:
 push edi
 pop edx
 mov eax, dword ptr [esi]          ; cmp Ri, dword ptr [Rj]
 mov eax, dword ptr [eax]
 cmp dword ptr [edx], eax
 jmp fin_CE

dword_xx_indirect_ce:
 cmp ecx, 0
 jne dword_xx_indirect_yy_indirect_ce
dword_xx_indirect_yy_direct_ce:
 push edi
 pop eax
 mov esi, dword ptr [esi]          ; cmp dword ptr [Ri], Rj
 mov eax, dword ptr [eax]
 cmp dword ptr [eax], esi
 jmp fin_CE
dword_xx_indirect_yy_indirect_ce:
 push edi
 pop eax
 mov eax, dword ptr [eax]          ; cmp dword ptr [Ri], dword ptr [Rj]
 mov esi, dword ptr [esi]
 mov esi, dword ptr [esi]
 cmp dword ptr [eax], esi
 jmp fin_CE
; ************************************ gestion des byte

BYTE_CE:
 cmp eax, 4
 jne byte_xx_indirect_ce
byte_xx_direct_ce:
```

```
 cmp ecx, 4
 jne byte_xx_direct_yy_indirect_ce
byte_xx_direct_yy_direct_ce:
 push edi
 pop edx
 mov al, byte ptr [esi]            ; cmp Ri(8), Rj(8)
 cmp byte ptr [edx], al
 jmp fin_CE

byte_xx_direct_yy_indirect_ce:
 push edi
 pop edx
 mov eax, dword ptr [esi]          ; cmp Ri(8), byte ptr [Rj]
 mov al, byte ptr [eax]
 cmp byte ptr [edx], al
 jmp fin_CE

byte_xx_indirect_ce:
 cmp ecx, 4
 jne byte_xx_indirect_yy_indirect_ce
byte_xx_indirect_yy_direct_ce:
 push edi
 pop eax
 mov bl, byte ptr [esi]            ; cmp byte ptr [Ri], Rj(8)
 mov eax, dword ptr [eax]
 cmp byte ptr [eax], bl
 jmp fin_CE
byte_xx_indirect_yy_indirect_ce:
 push edi
 pop eax
 mov eax, dword ptr [eax]          ; cmp byte ptr [Ri], byte ptr [Rj]
 mov esi, dword ptr [esi]
 mov bl, byte ptr [esi]
 cmp byte ptr [eax], bl
 jmp fin_CE


; ************************************ gestion des word

WORD_CE:
 cmp eax, 5
 jne word_xx_indirect_ce
word_xx_direct_ce:
 cmp ecx, 5
 jne word_xx_direct_yy_indirect_ce
word_xx_direct_yy_direct_ce:
 push edi
 pop edx
 mov ax, word ptr [esi]            ; cmp Ri(8), Rj(8)
 cmp word ptr [edx], ax
 jmp fin_CE
word_xx_direct_yy_indirect_ce:
 push edi
 pop edx
 mov eax, dword ptr [esi]          ; cmp Ri(8), byte ptr [Rj]
 mov ax, word ptr [eax]
 cmp word ptr [edx], ax
 jmp fin_CE
word_xx_indirect_ce:
 cmp ecx, 5
 jne word_xx_indirect_yy_indirect_ce
```

```
word_xx_indirect_yy_direct_ce:
 push edi
 pop eax
 mov bx, word ptr [esi]            ; cmp byte ptr [Ri], Rj(8)
 mov eax, dword ptr [eax]
 cmp word ptr [eax], bx
 jmp fin_CE
word_xx_indirect_yy_indirect_ce:
 push edi
 pop eax
 mov eax, dword ptr [eax]          ; cmp byte ptr [Ri], byte ptr [Rj]
 mov esi, dword ptr [esi]
 mov bx, word ptr [esi]
 cmp word ptr [eax], bx
 jmp fin_CE

fin_CE:
 sete bl
 mov ZF_, bl
 ja @F
 mov CmpF, 0
 jmp fin_CE2
@@:
 mov CmpF, 1
fin_CE2:
 add EIP_, 5
 popad
 mov eax,1
 ret
exit_CE:
 popad
 xor eax, eax
 ret
OPCODE_CE ENDP

; ************************************************************** OPCODE 57

OPCODE_57 PROC
 pushad
 mov eax, EIP_
 mov ecx, DWORD PTR [eax+1]
 cmp CmpF, 0
 jnz @F
 add EIP_, 5
 jmp exit_57
@@:
 add ecx, 74F364CAh
 add eax, ecx
 mov EIP_, eax
exit_57:
 popad
 mov eax, 1
 ret
OPCODE_57 ENDP

; ************************************************************** OPCODE 67

OPCODE_67 PROC
pushad
 mov eax, EIP_
 mov ecx, DWORD PTR [eax+1]
```

```
 cmp CmpF, 0
 je @F
 add EIP_, 5
 jmp exit_67
@@:
 add ecx, 74F364CAh
 add eax, ecx
 mov EIP_, eax
exit_67:
 popad
 mov eax, 1
 ret
OPCODE_67 ENDP


; ************************************************************ OPCODE 78



OPCODE_78 PROC
 pushad
 movzx ecx, byte ptr [edi+1]
 cmp ecx,0
 jl @F
 cmp ecx,8
 jg @F
 lea esi, dword ptr [ecx*4+ offset R0]
 mov ecx, dword ptr [edi+2]
 sub ecx, 15624562h
 mov eax, dword ptr [esi]
 mul ecx
 mov dword ptr [esi], eax
 add EIP_, 6
 popad
 mov eax, 1
 ret
@@:
 popad
 xor eax, eax
 ret
OPCODE_78 ENDP

; ************************************************************ OPCODE CC

OPCODE_CC PROC
 pushad
 mov eax, EIP_
 mov ecx, dword ptr [edi+1]
 add EIP_, ecx
 popad
 mov eax,1
 ret
OPCODE_CC ENDP


End Main
```

11.3.2 Le pseudo-code / routine d'enregistrement

Pour étudier le pcode de K5, il n'était pas nécessaire d'étudier la VM aussi profondément. Une étude comportementale en runtime du driver et une étude statique du pseudo-code suffisaient pour découvrir le rôle des instructions. Dans un premier temps, je vous propose le pseudo-code « brut » comme on peut le voir dans Kaine.sys . Je vous indique également en parallèle le rôle de chaque jeu d'instructions.

PSEUDO-CODE	
89 00 17 00 04 8C 2C 30 E8 00 00 00 07 89 03 17 03 64 35 4D 81 A0 00 03 00 02 33 03 F2 00 00 00 04 D3 00 EF CF 41 95 33 00	Junk code 1
17 01 4C 68 59 88	mov R1, 4090A2h
89 04 DF 04 FA A3 1D D6 56 04 04 04 06 89 01 17 01 4C 50 2B 2B E8 00 01 00 04 33 01 17 04 57 34 C8 20 E8 00 04 00 00 33 04	junk code 2
56 00 00 00 01	mov R0,R1
89 08 89 07 89 06 56 00 06 00 07 E8 00 06 00 08 D3 07 D4 BF 7D 31 F2 00 08 00 07 33 06 33 07 33 08	junk code 3
56 00 02 03 00 89 02	mov R2, DWORD PTR [R0] push R2
89 04 DF 04 FA A3 1D D6 56 04 04 04 06 89 01 17 01 4C 50 2B 2B E8 00 01 00 04	junk code 2

```
33 01
17 04 57 34 C8 20
E8 00 04 00 00
33 04

17 00 AA D7 18 88       mov R0, 0
89 00
17 00 04 8C 2C 30
E8 00 00 00 07
89 03
17 03 64 35 4D 81
A0 00 03 00 02
33 03
F2 00 00 00 04
D3 00 EF CF 41 95
33 00
17 03 B4 D7 18 88       mov R3, A
56 04 00 01 01          mov R0(8), BYTE PTR [R1]
89 04
DF 04 FA A3 1D
D6 56 04 04 04 06
89 01
17 01 4C 50 2B 2B
E8 00 01 00 04
33 01
17 04 57 34 C8 20
E8 00 04 00 00
33 04
17 06 EA E9 6D FF       mov R6,77551240
89 08
89 07
89 06
56 00 06 00 07
E8 00 06 00 08
D3 07 D4 BF 7D 31
F2 00 08 00 07
33 06
33 07
33 08
56 04 04 04 06          mov R4(8), BYTE PTR [R6]
89 00
17 00 04 8C 2C 30
E8 00 00 00 07
89 03
17 03 64 35 4D 81
A0 00 03 00 02
33 03
F2 00 00 00 04
D3 00 EF CF 41 95
33 00
17 07 0E D8 18 88       mov R7,64
```

```
CE 04 00 04 04          cmp R0(8),R4(8)
67 4F F9 46 C8          jne exit
89 04
DF 04 FA A3 1D
D6 56 04 04 04 06
89 01
17 01 4C 50 2B 2B
E8 00 01 00 04
33 01
17 04 57 34 C8 20
E8 00 04 00 00
33 04
CE 04 00 04 07          cmp R0(8), R7(8)
57 78 A4 0C 8B          je exit
65 01                   inc R1
89 00
17 00 04 8C 2C 30
E8 00 00 00 07
89 03
17 03 64 35 4D 81
A0 00 03 00 02
33 03
F2 00 00 00 04
D3 00 EF CF 41 95
33 00
99 03                   dec R3
97 16 EB 51 DE          jne boucle
89 00
17 00 04 8C 2C 30
E8 00 00 00 07
89 03
17 03 64 35 4D 81
A0 00 03 00 02
33 03
F2 00 00 00 04
D3 00 EF CF 41 95
33 00
17 01 4C 68 59 88       mov R1,4090A2
89 04
DF 04 FA A3 1D D6
56 04 04 04 06
89 01
17 01 4C 50 2B 2B
E8 00 01 00 04
33 01
17 04 57 34 C8 20
E8 00 04 00 00
33 04
17 08 89 68 59 88       mov R8,4090DF
17 03 B4 D7 18 88       mov R3,A
89 00
```

```
17 00 04 8C 2C 30
E8 00 00 00 07
89 03
17 03 64 35 4D 81
A0 00 03 00 02
33 03
F2 00 00 00 04
D3 00 EF CF 41 95
33 00
17 02 AA D7 18 88      mov R2,0
17 07 56 68 59 88      mov R7,4090AC
89 00
17 00 04 8C 2C 30
E8 00 00 00 07
89 03
17 03 64 35 4D 81
A0 00 03 00 02
33 03
F2 00 00 00 04
D3 00 EF CF 41 95
33 00
17 00 DA D7 18 88      mov R0,30
56 04 02 01 01         mov R2(8), BYTE PTR [R1]
89 04
DF 04 FA A3 1D D6
56 04 04 04 06
89 01
17 01 4C 50 2B 2B
E8 00 01 00 04
33 01
17 04 57 34 C8 20
E8 00 04 00 00
33 04
DF 02 5F A9 5E 28      sub R2,41
56 00 06 00 07         mov R6,R7
89 00
17 00 04 8C 2C 30
E8 00 00 00 07
89 03
17 03 64 35 4D 81
A0 00 03 00 02
33 03
F2 00 00 00 04
D3 00 EF CF 41 95
33 00
E8 00 06 00 02         add R6,R2
89 08
89 07
89 06
56 00 06 00 07
E8 00 06 00 08
```

```
D3 07 D4 BF 7D 31
F2 00 08 00 07
33 06
33 07
33 08
56 04 04 01 06      mov R4(8), BYTE PTR [R6]
E8 00 04 00 00      add R4,R0
89 00
17 00 04 8C 2C 30
E8 00 00 00 07
89 03
17 03 64 35 4D 81
A0 00 03 00 02
33 03
F2 00 00 00 04
D3 00 EF CF 41 95
33 00
56 01 08 04 04      mov BYTE PTR [R8], R4(8)
65 08               inc R8
89 00
17 00 04 8C 2C 30
E8 00 00 00 07
89 03
17 03 64 35 4D 81
A0 00 03 00 02
33 03
F2 00 00 00 04
D3 00 EF CF 41 95
33 00
65 01               inc R1
89 08
89 07
89 06
56 00 06 00 07
E8 00 06 00 08
D3 07 D4 BF 7D 31
F2 00 08 00 07
33 06
33 07
33 08
99 03               dec R3
97 C4 EA 51 DE      jne boucle
17 04 AA D7 18 88   mov R4,0
89 00
17 00 04 8C 2C 30
E8 00 00 00 07
89 03
17 03 64 35 4D 81
A0 00 03 00 02
33 03
F2 00 00 00 04
```

```
D3 00 EF CF 41 95
33 00
17 03 AE D7 18 88      mov R3,4
17 01 89 68 59 88      mov R1,4090DF
89 04
DF 04 FA A3 1D D6
56 04 04 04 06
89 01
17 01 4C 50 2B 2B
E8 00 01 00 04
33 01
17 04 57 34 C8 20
E8 00 04 00 00
33 04
D3 01 A9 D5 F8 31      add R1,6
89 00
17 00 04 8C 2C 30
E8 00 00 00 07
89 03
17 03 64 35 4D 81
A0 00 03 00 02
33 03
F2 00 00 00 04
D3 00 EF CF 41 95
33 00
17 02 AA D7 18 88      mov R2,0
89 08
89 07
89 06
56 00 06 00 07
E8 00 06 00 08
D3 07 D4 BF 7D 31
F2 00 08 00 07
33 06
33 07
33 08
56 04 04 01 01         mov R4(8), BYTE PTR [R1]
65 01                  inc R1
DF 04 4E A9 5E 28      sub R4,30
89 00
17 00 04 8C 2C 30
E8 00 00 00 07
89 03
17 03 64 35 4D 81
A0 00 03 00 02
33 03
F2 00 00 00 04
D3 00 EF CF 41 95
33 00
56 00 08 00 02         mov R8,R2
78 08 66 45 62 15      mul R8,4
```

```
89 08
89 07
89 06
56 00 06 00 07
E8 00 06 00 08
D3 07 D4 BF 7D 31
F2 00 08 00 07
33 06
33 07
33 08
E8 00 02 00 08          add R2,R8
89 00
17 00 04 8C 2C 30
E8 00 00 00 07
89 03
17 03 64 35 4D 81
A0 00 03 00 02
33 03
F2 00 00 00 04
D3 00 EF CF 41 95
33 00
78 02 64 45 62 15       mul R2,2
E8 00 02 00 04          add R2,R4
89 08
89 07
89 06
56 00 06 00 07
E8 00 06 00 08
D3 07 D4 BF 7D 31
F2 00 08 00 07
33 06
33 07
33 08
56 04 04 01 01          mov R4(8), BYTE PTR [R1]
89 00
17 00 04 8C 2C 30
E8 00 00 00 07
89 03
17 03 64 35 4D 81
A0 00 03 00 02
33 03
F2 00 00 00 04
D3 00 EF CF 41 95
33 00
65 01                   inc R1
99 03                   dec R3
97 22 EB 51 DE          jne ...
17 05 04 15 19 88       mov R5,3D5A
89 00
17 00 04 8C 2C 30
E8 00 00 00 07
```

```
89 03
17 03 64 35 4D 81
A0 00 03 00 02
33 03
F2 00 00 00 04
D3 00 EF CF 41 95
33 00
D3 02 61 F4 F8 31      add R2,1EBE
89 08
89 07
89 06
56 00 06 00 07
E8 00 06 00 08
D3 07 D4 BF 7D 31
F2 00 08 00 07
33 06
33 07
33 08
CE 00 02 00 05         cmp R2,R5
97 2F F1 51 DE         jne ...
17 06 89 68 59 88      mov R6,4090DF
89 00
17 00 04 8C 2C 30
E8 00 00 00 07
89 03
17 03 64 35 4D 81
A0 00 03 00 02
33 03
F2 00 00 00 04
D3 00 EF CF 41 95
33 00
17 08 AA D7 18 88      mov R8,0
89 08
89 07
89 06
56 00 06 00 07
E8 00 06 00 08
D3 07 D4 BF 7D 31
F2 00 08 00 07
33 06
33 07
33 08
56 04 08 01 06         mov R8(8),BYTE PTR [R6]
89 04
DF 04 FA A3 1D D6
56 04 04 04 06
89 01
17 01 4C 50 2B 2B
E8 00 01 00 04
33 01
17 04 57 34 C8 20
```

```
E8 00 04 00 00
33 04
DF 08 41 A9 5E 28       sub R8,23
89 00
17 00 04 8C 2C 30
E8 00 00 00 07
89 03
17 03 64 35 4D 81
A0 00 03 00 02
33 03
F2 00 00 00 04
D3 00 EF CF 41 95
33 00
17 04 F2 D7 18 88       mov R4,48
CE 00 08 00 04          cmp R8,R4
97 6C F0 51 DE          jne ...
89 08
89 07
89 06
56 00 06 00 07
E8 00 06 00 08
D3 07 D4 BF 7D 31
F2 00 08 00 07
33 06
33 07
33 08
65 06                   inc R6
89 00
17 00 04 8C 2C 30
E8 00 00 00 07
89 03
17 03 64 35 4D 81
A0 00 03 00 02
33 03
F2 00 00 00 04
D3 00 EF CF 41 95
33 00
17 08 AA D7 18 88       mov R8, 0
89 08
89 07
89 06
56 00 06 00 07
E8 00 06 00 08
D3 07 D4 BF 7D 31
F2 00 08 00 07
33 06
33 07
33 08
56 04 08 01 06          mov R8(8), BYTE PTR [R6]
89 04
DF 04 FA A3 1D D6
```

```
56 04 04 04 06
89 01
17 01 4C 50 2B 2B
E8 00 01 00 04
33 01
17 04 57 34 C8 20
E8 00 04 00 00
33 04
D3 08 F0 D5 F8 31      add R8,4D
89 00
17 00 04 8C 2C 30
E8 00 00 00 07
89 03
17 03 64 35 4D 81
A0 00 03 00 02
33 03
F2 00 00 00 04
D3 00 EF CF 41 95
33 00
17 07 58 D8 18 88      mov R7,AE
CE 00 08 00 07         cmp R8,R7
97 8C EF 51 DE         jne ...
65 06                  inc R6
17 02 AA D7 18 88      mov R2,0
89 00
17 00 04 8C 2C 30
E8 00 00 00 07
89 03
17 03 64 35 4D 81
A0 00 03 00 02
33 03
F2 00 00 00 04
D3 00 EF CF 41 95
33 00
56 04 02 01 06         mov R2(8), BYTE PTR [R6]
17 04 4F D8 18 88      mov R4,A5
A0 00 02 00 04         xor R2,R4
89 00
17 00 04 8C 2C 30
E8 00 00 00 07
89 03
17 03 64 35 4D 81
A0 00 03 00 02
33 03
F2 00 00 00 04
D3 00 EF CF 41 95
33 00
17 08 76 D8 18 88      mov R8,CC
89 08
89 07
89 06
```

```
56 00 06 00 07
E8 00 06 00 08
D3 07 D4 BF 7D 31
F2 00 08 00 07
33 06
33 07
33 08
CE 04 02 04 08          cmp R2,R8
97 F1 EE 51 DE          jne ...
D3 06 A4 D5 F8 31       add R6,1
89 08
89 07
89 06
56 00 06 00 07
E8 00 06 00 08
D3 07 D4 BF 7D 31
F2 00 08 00 07
33 06
33 07
33 08
17 07 AA D7 18 88       mov R7,0
89 00
17 00 04 8C 2C 30
E8 00 00 00 07
89 03
17 03 64 35 4D 81
A0 00 03 00 02
33 03
F2 00 00 00 04
D3 00 EF CF 41 95
33 00

56 04 07 01 06          mov R7(8), BYTE PTR [R6]
DF 07 63 A9 5E 28       sub R7,45
89 04
DF 04 FA A3 1D D6
56 04 04 04 06
89 01
17 01 4C 50 2B 2B
E8 00 01 00 04
33 01
17 04 57 34 C8 20
E8 00 04 00 00
33 04
17 01 6D D8 18 88       mov R1,C3
89 00
17 00 04 8C 2C 30
E8 00 00 00 07
89 03
17 03 64 35 4D 81
A0 00 03 00 02
```

```
33 03
F2 00 00 00 04
D3 00 EF CF 41 95
33 00
A0 04 07 04 01          xor R7,R1
DF 07 07 AA 5E 28       sub R7,E9
89 00
17 00 04 8C 2C 30
E8 00 00 00 07
89 03
17 03 64 35 4D 81
A0 00 03 00 02
33 03
F2 00 00 00 04
D3 00 EF CF 41 95
33 00
99 07                   dec R7
97 FD ED 51 DE          jne …
D3 06 A4 D5 F8 31       add R6,1
17 03 AA D7 18 88       mov R3,0
89 00
17 00 04 8C 2C 30
E8 00 00 00 07
89 03
17 03 64 35 4D 81
A0 00 03 00 02 33 03
F2 00 00 00 04
D3 00 EF CF 41 95
33 00
56 04 03 01 06          mov R3(8), BYTE PTR [R6]
DF 03 7E A9 5E 28       sub R3,60
89 08
89 07
89 06
56 00 06 00 07
E8 00 06 00 08
D3 07 D4 BF 7D 31
F2 00 08 00 07
33 06
33 07
33 08
78 03 77 45 62 15       mul R3,15
89 00
17 00 04 8C 2C 30
E8 00 00 00 07
89 03
17 03 64 35 4D 81
A0 00 03 00 02
33 03
F2 00 00 00 04
D3 00 EF CF 41 95
```

```
33 00
DF 03 86 A9 5E 28      sub R3,68
99 03                  dec R3
97 60 ED 51 DE         jne ...
65 06                  inc R6
89 08
89 07
89 06
56 00 06 00 07
E8 00 06 00 08
D3 07 D4 BF 7D 31
F2 00 08 00 07
33 06
33 07
33 08
17 00 AA D7 18 88      mov R0,0
89 00
17 00 04 8C 2C 30
E8 00 00 00 07
89 03
17 03 64 35 4D 81
A0 00 03 00 02
33 03
F2 00 00 00 04
D3 00 EF CF 41 95
33 00
56 04 00 01 06         mov R0(8),BYTE PTR [R6]
89 04
DF 04 FA A3 1D D6
56 04 04 04 06
89 01
17 01 4C 50 2B 2B
E8 00 01 00 04
33 01
17 04 57 34 C8 20
E8 00 04 00 00
33 04
17 08 71 68 59 88      mov R8, 4090C7h
89 08
89 07
89 06
56 00 06 00 07
E8 00 06 00 08
D3 07 D4 BF 7D 31
F2 00 08 00 07
33 06
33 07
33 08
DF 00 4E A9 5E 28      sub R0,30
89 00
17 00 04 8C 2C 30
```

```
E8 00 00 00 07
89 03
17 03 64 35 4D 81
A0 00 03 00 02 33 03
F2 00 00 00 04
D3 00 EF CF 41 95
33 00
78 00 66 45 62 15        mul R0,4
89 08
89 07
89 06
56 00 06 00 07
E8 00 06 00 08
D3 07 D4 BF 7D 31
F2 00 08 00 07
33 06
33 07
33 08
E8 00 08 00 00           add R8,R0
89 08
89 07
89 06
56 00 06 00 07
E8 00 06 00 08
D3 07 D4 BF 7D 31
F2 00 08 00 07
33 06
33 07
33 08
56 00 00 03 08           mov R0, BYTE PTR [R8]
89 04
DF 04 FA A3 1D D6
56 04 04 04 06
89 01
17 01 4C 50 2B 2B
E8 00 01 00 04
33 01
17 04 57 34 C8 20
E8 00 04 00 00
33 04
F4                       ret
89 04
DF 04 FA A3 1D D6
56 04 04 04 06
89 01
17 01 4C 50 2B 2B
E8 00 01 00 04
33 01
17 04 57 34 C8 20
E8 00 04 00 00
33 04
```

	17 00 AA D7 18 88 F4	mov R0,0 ret

ANALYSE DU PSEUDO-CODE :

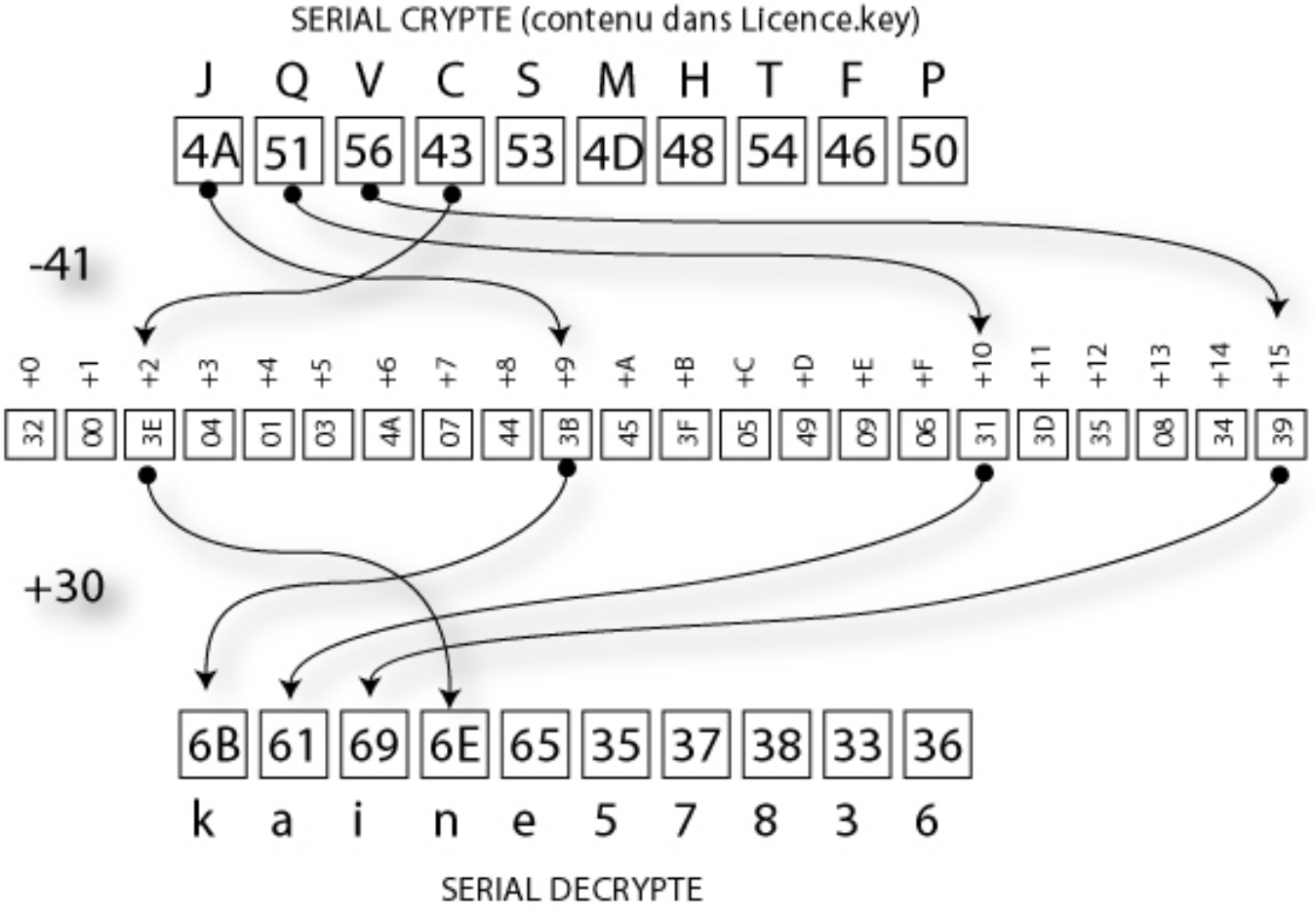
Maintenant, nous allons passer aux explications de cette « petite » routine. Tout d'abord, nous ferons évidemment abstraction du junk code. Maintenant, il faut en fait voir que cette routine est constituée de plusieurs boucles qui travaillent sur le serial les unes après les autres. Chacune d'elle a un rôle précis dans la vérification.

Voici tout d'abord un dump des zones mémoires utilisées :

Address	Hex dump	ASCII
004090A2	00 00 00 00 00 00 00 00 00 00 32 00 3E 04 01 03	..........2.>♦☺♥
004090B2	4A 07 44 3B 45 3F 05 49 09 06 31 3D 35 08 34 39	J•D;E?♣I.♠1=5◘49
004090C2	43 33 47 02 00 00 00 00 00 00 00 00 00 00 00 00	C3G☻............
004090D2	00 00 00 00 00 00 00 00 00 01 00 00 00 00 00 00	.........☺......
004090E2	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
004090F2	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
00409102	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
00409112	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....

Première boucle : La routine commence par vérifier si les 10 lettres du serial (situés en 4090A2) sont bien des caractères dont les valeurs ascii sont comprises entre 40h et 64h. Dans le cas contraire, la routine s'arrête.

Deuxième boucle : Cette deuxième routine va décrypter le serial à l'aide d'une table (située en 4090AC) caractère par caractère. Le nouveau serial décrypté (situé en 4090DF) sera celui qui permettra de vérifier la validité du serial de départ. Le cryptage se fait de la façon suivante :



Pour décrypter le premier caractère « J », soit **4Ah** en ascii , on procède de la façon suivante :
1) On retranche 41h ce qui donne 4Ah – 41h = 9h. Ceci est un indice qui permet d'accéder à la neuvième colonne d'une table de bytes.
2) On récupère la valeur située à la 9ème position, soit 3Bh et on lui ajoute 30h : 3Bh + 30h = **6Bh**. Ceci est la valeur décryptée que l'on place dans l'espace réservé au serial décrypté.

Troisième boucle (4 derniers caractères): Cette routine et les suivantes vont tester la validité du serial décrypté. Ici, la boucle va récupérer la chaîne « 7836 » (les quatre derniers caractères du serial décrypté) et va convertir ce « nombre » en hexadécimal. Le nombre 7836d devient 1E9Ch. La somme de ce nombre et de 1EBEh doit être égale à 3D5Ah (ici, c'est bien le cas) :

1E9Ch + 1EBEh = 3D5Ah

Quatrième boucle (1er caractère): On teste maintenant le premier caractère du serial décrypté. Il doit vérifier une condition très simple : (nommons xxh la valeur ascii de ce char)

xxh – 23h = 48h

On voit clairement que xxh = 6Bh.

Cinquième boucle (2ème caractère): On teste maintenant le second caractère du serial décrypté. Il doit vérifier une condition très simple : (nommons xxh la valeur ascii de ce char)

xxh + 4Dh = AEh

On voit clairement que xxh = 61h.

Sixième boucle (3ème caractère): On teste maintenant le troisième caractère du serial décrypté. Il doit vérifier une condition très simple : (nommons xxh la valeur ascii de ce char)

xxh xor A5h = CCh

On voit clairement que xxh = 69h.

Septième boucle (4ème caractère): On teste maintenant le quatrième caractère du serial décrypté. Il doit vérifier la condition suivante : (nommons xxh la valeur ascii de ce char)

(xxh - 45h) xor C3h – E9h - 1 = 0

On voit que xxh = 6Eh.

Huitième boucle (5ème caractère): On teste maintenant le cinquième caractère du serial décrypté. Il doit vérifier la condition suivante : (nommons xxh la valeur ascii de ce char)

(xxh - 60h) x 15h – 68h - 1 = 0

On voit que xxh = 65h.

Neuvième boucle (6ème caractère): On teste maintenant le sixième caractère du serial décrypté. Il doit vérifier la condition suivante : (nommons xxh la valeur ascii de ce char)

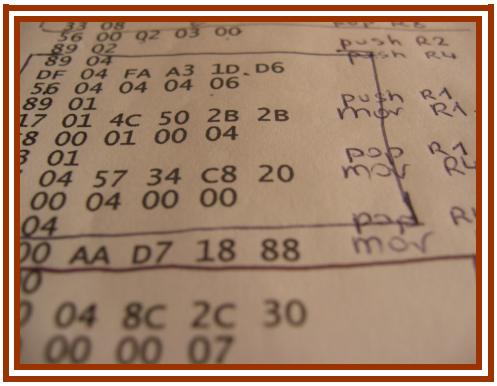
(xxh - 30h) x 4h = indice

Cet indice pointe dans une table (située en 4090C7) remplie de zéros et un 1 ! En fait, l'indice doit pointer vers ce nombre 1 (nous allons voir après pourquoi). Pour cela , **indice = 14h.** Ceci nous donne donc comme condition :

(xxh - 30h) x 4h = 14h

On voit donc que xxh = 35h

Une fois que la VM a terminé son travail, K5 va récupérer la valeur de eax fournie par Kaine.sys. Puis, il va faire une simple comparaison sur eax. Si eax = 1, K5 va afficher le message de succès. SI eax = 0, K5 affichera le fameux « try again man ».



JOUR 10 : Lundi 30 mai 2005 : 13h15

Je viens de passer la pire nuit de ma vie ! impossible de fermer l'œil… bon sang ! il me manque le 6ème caractère du serial !

***17h 40** : ça se joue à la minute peut-être avec Kharneth ! Mais ce n'est pas possible ! Je suis sûr de mon serial cette fois mais le crackme ne veut rien savoir ! Que se passe-t-il ?*

20h15** : Flûte ! le keyfile s'appelle License.key et pas Licence.key !! ça marche cette fois…j'ai FINI !! je retiens ma respiration… Kharneth a fini ? apparemment non ☺* ***Victoire !

Pour la petite histoire, Kharneth finira le surlendemain, mercredi 1er juin dans la matinée… ;)

12. LE « K5 Tour 2005 » par BeatriX.

Pour finir en beauté, quoi de mieux qu'une petite démarche pour essayer de tracer le K5 gentiment. J'ai horreur de faire ce genre de trucs mais il s'agit bien de la partie « presse bouton » de mon article. Je vous propose ici ma démarche pour tracer et atteindre certaines parties du crackme sans trop d'encombres… normalement, vous pouvez atteindre le ExitThread final en 5 minutes montre en main ☺.

PREPARATIFS :

Tout d'abord, il faut préparer son debugger favori. Il faut modifier le nom de la dll HideDebugger.dll pour éviter un bsod. Si vous disposez de isdebug.dll, modifiez son nom également. Ensuite avant de débugger K5, utilisez HideDebugger pour contourner IsDebuggerPresent et FindWindow tricks. Munissez vous du driver beeper.sys (du KMDKIT) que vous renommerez Protect.sys et Kaine.sys. (vous avez donc deux copies de beeper.sys). Pour finir, compilez le code du handler que je vous propose dans la section concernant le décryptage de layers effectué à l'aide de Protect.sys. Nous aurons besoin d'injecter ce code dans le process du K5.

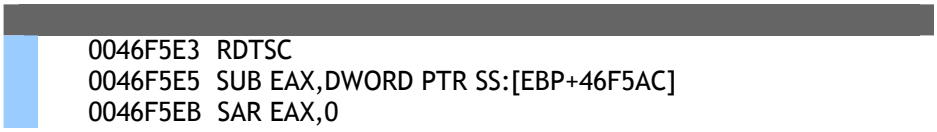
PROMENADE DANS LE K5 :

A partir de là , on peut commencer le debuggage.
0) Si OllyDbg vous propose d'analyser le K5 au moment de l'ouverture de celui-ci, REFUSEZ ! Autre remarque : Ne lancez le K5 (SHIFT+F9) que lorsque je le précise explicitement !

1) Ouvrez le K5 avec OllyDbg. Posez un BPX à l'intérieur de VirtualAlloc (sur l'appel de VirtualAllocEx) et lancez K5 (SHIFT + F9).
2) Il est possible que vous stoppiez sur des exceptions. Ignorez les en cochant les bonnes cases.

ESCALE 0 : TRICK DU CLOSEHANDLE+RDTSC

3) Il est possible que le K5 stoppe sur un INVALID_HANDLE. Tracez avec F8 jusqu'à ceci :



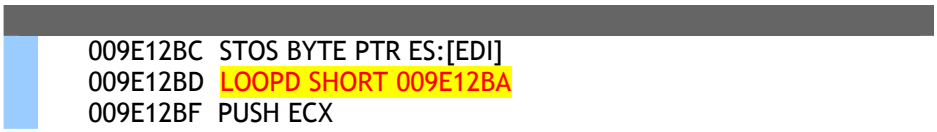
Dès que vous êtes en 46F5EB, mettez eax à 0.Lancez le K5.(SHIFT+F9)

ESCALES 1 et 2 :

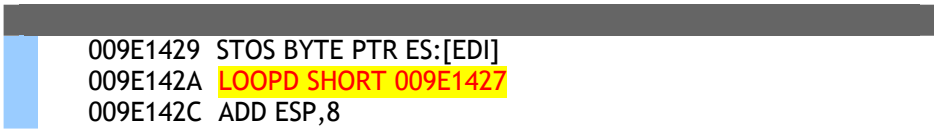
4) **Vous allez breaker sur VirtualAlloc**. Relancez le K5. Vous allez breaker une deuxième fois sur VirtualAlloc. Retirez **absolument** le BPX après le second break. EAX pointe à ce moment précis vers la zone virtuelle qui s'occupe de lancer les drivers. Retenez cette adresse qui sera utile par la suite. Pour mon exposé, eax = 960010h.

ESCALE 3 : éviter le trick du ZwQueryInformationProcess

5) Posez un BP memory on write sur 960010h + 9A5D3h = 9FA5E3h.(faites votre addition avec votre adresse virtuelle). Lancez le K5.
6) Vous breakez sur du code qui ressemble à ceci :



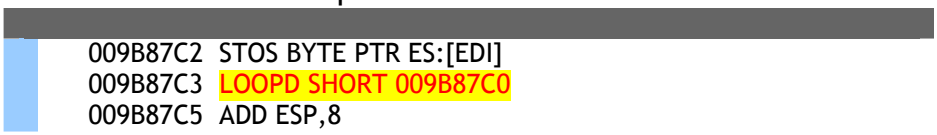
Vous devez vous placer sur le push ecx et changer l'EIP (CTRL + ALT GR + *). Poser tout de suite après un BP memory on write sur 960010h + 9761Ch = 9F762Ch. Lancez le K5.
7) Vous breakez sur du code qui ressemble à ceci :



Changez à nouveau l'EIP de la même façon sur le add esp, 8. Lancez le K5.

ESCALE 4 : éviter le trick du PEB.

8) vous breakez à nouveau sur du code qui ressemble à ceci :



Changez à nouveau l'EIP. Retirez le BP on memory write (clic droit + BP + Remove memoryBP). Posez un BPX à l'intérieur de VirtualAlloc (comme à l'étape 1). Lancez le K5.

ESCALE 5 : Secondes redirections.

9) Vous breakez sur VirtualAlloc. EAX indique l'adresse de la fonction LoadLibraryA. Lancez K5. Vous breakez à nouveau sur VirtualAlloc mais EAX indique cette fois l'adresse de la fonction

MessageBoxA. Recommencez cette opération jusqu'à ce que EAX indique l'adresse de la fonction OpenServiceA.

ESCALE 6 : Lancement de Protect.sys

10) Posez un BPX à l'intérieur de la fonction StartServiceA sur le premier call :

```
77DADF0E  PUSH 10
77DADF10  PUSH ADVAPI32.77DFF050
77DADF15  CALL ADVAPI32.77DA1674
```

Lancez le K5

11) Vous breakez sur StartServiceA. Remplacez le fichier Protect.sys (situé dans C:\WINDOWS\System32\ ) par votre copie. Posez également un BPX à l'intérieur de la fonction CreateFileA. Lancez le K5.

ESCALE 7 : Layers cryptés et protégés par Protect.sys.

12) BIP ! Vous avez entendu le driver beeper.sys démarrer. Vous breakez sur CreateFileA. Tracez avec F8 jusqu'à revenir dans K5. Vous tombez sur un *cmp eax, -1*. Changez la valeur de eax par 0. Tracez avec F7 jusqu'à 960010h + 99111h = 9F91C1h.

13) Lancez un second OllyDbg, ouvrez le handler que vous avez compilé. Dans la fenêtre de dump, sélectionnez tout le code du handler et faites un Binary Copy. Revenez dans le OllyDbg d'origine, choisissez l'une des zones du process (celle que vous voulez qui soit inscriptible et exécutable) et copiez (Binary Paste) le handler dedans. Modifiez l'adresse du handler du SEH en indiquant la nouvelle adresse que vous venez de fixer. Lancez le K5.

ESCALE 8 : Troisième groupe de redirections.

14) Vous breakez sur VirtualAlloc. Lancez K5 jusqu'à breaker sur StartServiceA (normalement, vous n'avez pas oté le BPX déjà posé tout à l'heure). Vous allez sans doute en cours de route breaker sur CreatFileA.

15) Vous breakez sur StartServiceA. Remplacez le driver Kaine.sys par votre copie. Tracez à présent le K5 pour voir la fin du crackme. Vous allez tomber sur un *cmp eax, 1* qui est le test du serial. Les messages affichés sont issus de la fonction MessageBoxA.

13. REMERCIEMENTS

Nous voilà donc à la fin de ce travail fastidieux ! Je dois avouer que je ne pensais pas faire un tel article. Néanmoins, K5 est une œuvre gigantesque qui ne peut pas se résumer en quelques pages d'un tutorial baclé. Je pense avoir fait honneur au travail de Kaine en réalisant un tel article.

Je commence évidemment par remercier **Kaine**, auteur de ce défi, qui s'est surpassé pour nous offrir ce binaire magnifique. J'affiche publiquement mon admiration pour le travail de ce jeune garçon ☺, « bravo mon Kainou !! ». En plus, (ça reste entre nous), c'est un beau gosse…il a tout pour lui ce salaud !! héhé ☺

Je le remercie aussi pour avoir accepté de faire la relecture complète de cet article et pour avoir corrigé quelques imprecisions.

Merci également à **Neitsa** pour avoir proposé le trick du PEB (extra ce truc !) , pour avoir également accepté de relire cet article fastidieux et avoir testé le « K5 Tour 2005 »…et j'en profite pour dire que nous avons là encore un reverser de grande qualité…sympa comme tout en plus ☺. Au fait Neitsa, reste avec nous ! On ne veut pas que tu partes !!

Merci à **Kharneth** pour m'avoir fait trembler durant 10 jours (espèce de gros vilain !). L'émulation entre reversers est finalement très bonne !! Je te promets, tu auras ta revanche ☺

Merci à **TTO** pour ses discussions sur IRC (notamment à propos de Soft Ice). Allez TTO, ton traceur aura raison de K5 un jour ☺ !

Merci à tous les membres actifs de #UCT et notamment à Genaytyk (alors ce compilo, il avance !?), aTa, meow, Kryshaam, lautheking, Holyview pour leur bonne humeur .

Merci aux membres de la FRET (Neitsa, ++Meat, DarkPhoenix, eedy31, Requiem, _TWG_) pour leur compétence et leur sympathie.

Merci aux membres de FC sans qui je n'aurais jamais pu faire tout le chemin que j'ai déjà parcouru…

Merci également à mon ami **Proton** qui a eu la patience et la gentillesse de m'expliquer comment gérer certains paramètres pour faire un pdf qui ressemble à quelque chose ☺

Info de dernière minute (avant la publication de ce tutorial) : eedy31 vient de réussir à son tour le K5 ☺ Bravo à lui !

14. SOURCES / DOCUMENTATIONS

Documentation utilisée :

La documentation officielle INTEL

Le MSDN

Jeremy Gordon (**Win32 Exception handling for assembler programmers)**

Matt Pietrek (**Under the Hood**)

Gary Nebett (**Windows NT2000 Native API)**

Peter Szor **(Virus research and defense)**

Peter Szor **(VIRUS BULLETIN NOVEMBER 1997 “Copying with Cabanas”)**

Four-F **(Kmd Tut) traduit en français par Neitsa**

BeatriX, le samedi 25 juin 2005 – 16h38 (1 mois après le début des hostilités)