Les protections logiciels

Les concevoir, les tester.

Pourquoi protéger son logiciel :

Les désassembleurs et les débogueurs permettent d'étudier un exécutable sans posséder sa source, il est donc possible de modifier le comportement du programme (pour éviter d'avoir a payer une License par exemple), de retrouver les algorithmes utilisé dans votre code, de coder un programme qui ait les mêmes propriétés .

Il est donc nécessaire de protéger votre application pour :

- Qu'il ne soit pas pirater
- Protéger les algorithmes qu'il contient contre l'espionnage industriel
- Eviter de perdre de l'argent en général

Comment le protéger :

- Juridiquement :
  - tout les logiciels sont protégés par le droit d'auteur
  - Dans certains cas grâce à des brevets
- En bloquant et en freinant l'analyse des crackers, des reverse engineer :
  - En modifiant le code de façon a le rendre incompréhensible par un humain directement (obfuscation)
  - En détectant et en bloquant les tentatives d'analyses de votre logiciel (anti-débogueurs, anti-désassembleur)
  - En utilisant des mesures préventives (listes de clients, vérification de la licence par un serveur, distribution de versions de démo incomplètes)

Pourquoi ce cours ?

1. Le coût du piratage aux éditeurs de logiciels
2. La mauvaise connaissance générale de la protection logiciel
3. La mauvaise implémentation des solutions commerciales actuelles.

Le coût du piratage dans le monde et en France*

Année	Taux de piratage (monde / France)	Manque à gagner (en M de $) (monde / France)
2003	36% / 45%	28 803 / 2 311
2004	35% / 45%	32 778 / 2 928
2005	35% / 47%	34 482 / 3 191
2006	35% / 45%	39 698 / 2 676
2007	38% / 42%	47 809 / 2 601

* Source : BSA and IDC Global Software Piracy Study

La méconnaissance du fonctionnement des protections logiciels.

- La protection logiciel est quelque chose de souvent obscure pour les programmeurs.
- Les éditeurs de protections commerciales abusent parfois de ce manque de connaissance et gonflent exagérément leurs prix.
- Les protections logiciels sont encore trop souvent vendues pour leur image plutôt que pour leur efficacité.

La mauvaise implémentation d'une protection logiciel

- Les protections actuelles sont généralement livrées avec un lot de fonctions utilisables par le développeur ainsi qu'une série d'options.

- Une protection dans sa version de base est rarement efficace contre les crackers, il n'est pas rare qu'un éditeur voit son logiciel piraté juste parce qu'il n'a pas pris le temps de configurer les options de protection.

- Une protection logiciel est d'autant plus efficace qu'elle est intimement liée avec le programme qu'elle protège, négliger les outils qu'elle met a votre disposition est une erreur que beaucoup de développeurs font.

Plan du cours

1. Les outils et techniques d'analyse de code :
   - Désassembleurs et analyse statique du code, introduction à IDA
   - Débogueurs et analyse dynamique, introduction à OllyDBG
2. Protections "maison" :
   - Numéro de série simple
   - Obfuscation du code
   - Chiffrement du code
   - Vérifications furtives
3. Protections commerciales :
   - Principe des *packers*
   - Redirections d'APIs win32 et anti-débogueurs
   - Techniques de fusion avec l'application
   - Quelques exemples de *packers*
4. Conclusion :
   - Récapitulatif
   - Le métier de Reverse-Engineer
   - Questions ?

Les outils et techniques d'analyse du code

- Désassembleurs et analyse statique du code
  - introduction à IDA

- Débogueurs et analyse dynamique
  - introduction a OllyDBG

Les désassembleurs

Un désassembleur permet à partir d'un binaire d'obtenir du code en assembleur, le code obtenu ne contient bien sur plus aucun commentaire ni labels.

Les désassembleurs les plus performants arrivent à retrouver le type des variables, permettent de les renommer et, si ils le peuvent, génèrent automatiquement un nom à partir de l'utilisation qui en est fait.

Le désassembleur le plus utilisé et le plus général en environnement win32 est IDA. Il existe d'autres désassembleur qui ciblent des exécutables générés par des compilateurs précis mais nous ne les aborderons pas dans ce cour.

L’analyse statique

Faire une analyse statique d’un exécutable va consister à en étudier le code désassemblé sans l’exécuter (d’où le ‘statique’).

On renomme les variables du programmes, ses fonctions etc. jusqu’à être capable de retrouver la logique du programme, capable de réécrire un code qui aura les mêmes propriétés.

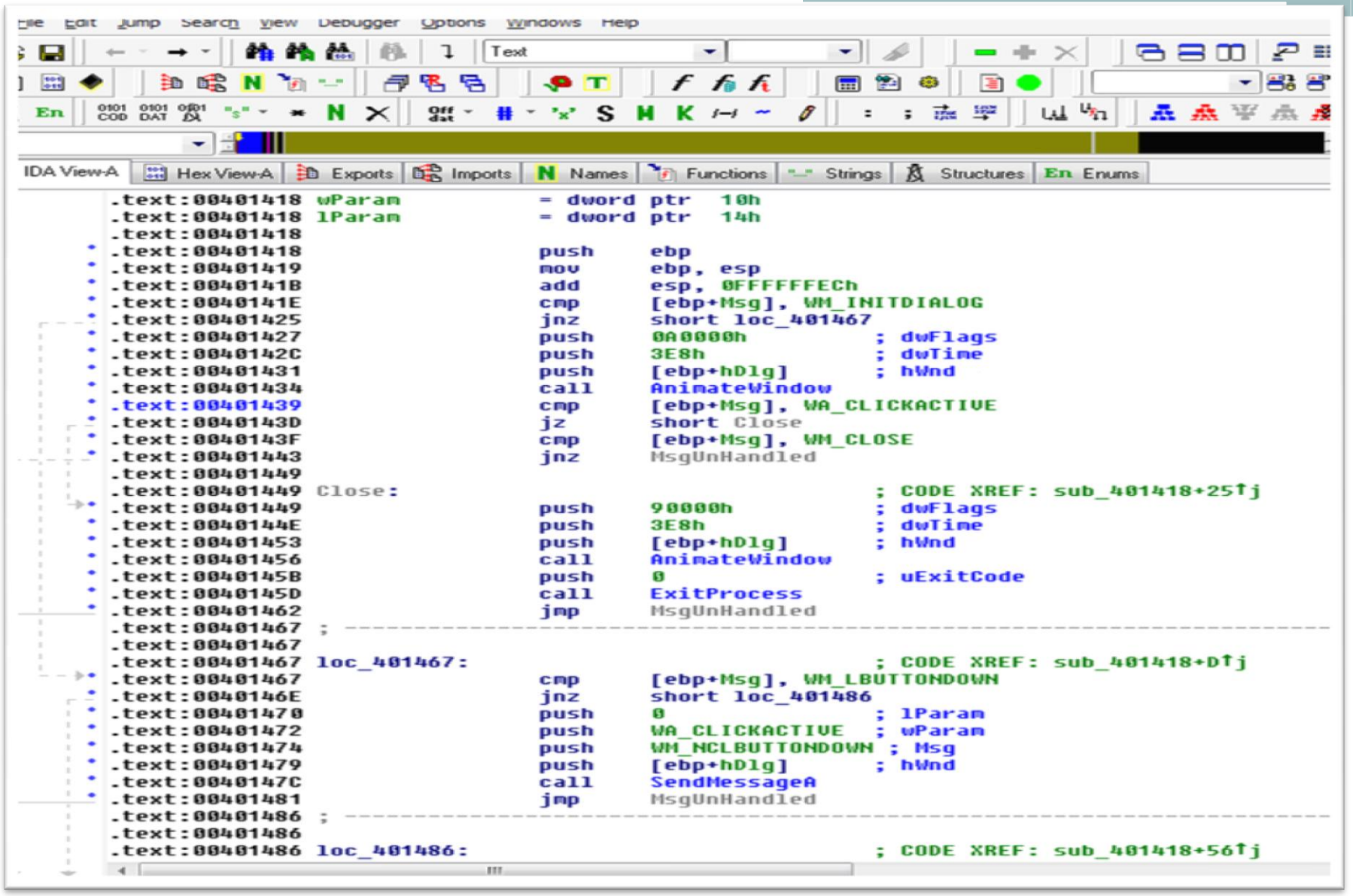
IDA

IDA est un désassembleur professionnel très puissant qui permet à l'utilisateur de s'approprier le code d'une application grâce à l'analyse très fine qu'il fait du code, de l'utilisation des registres et de la pile.

Cette analyse lui permet d'isoler les fonctions, leurs relations, leurs variables locales etc.

IDA a longtemps été développé par DataRescue mais est maintenant la propriété de Hex-Rays, la société fondée par Ilfak Guilfanov, principal développeur de IDA.

Nous travaillerons avec la version gratuite de IDA qui ne comporte pas certaines options tel que la représentation du code sous forme de graphiques.



IDA 4.9 : Version *Freeware*

Les débogueurs

Un débogueur permet, comme son nom l'indique, de déboguer un programme, c'est-à-dire de suivre son déroulement pas à pas, de contrôler son exécution grâce à des points d'arrêts, la possibilité de modifier le contenu des registres, de la mémoire etc.

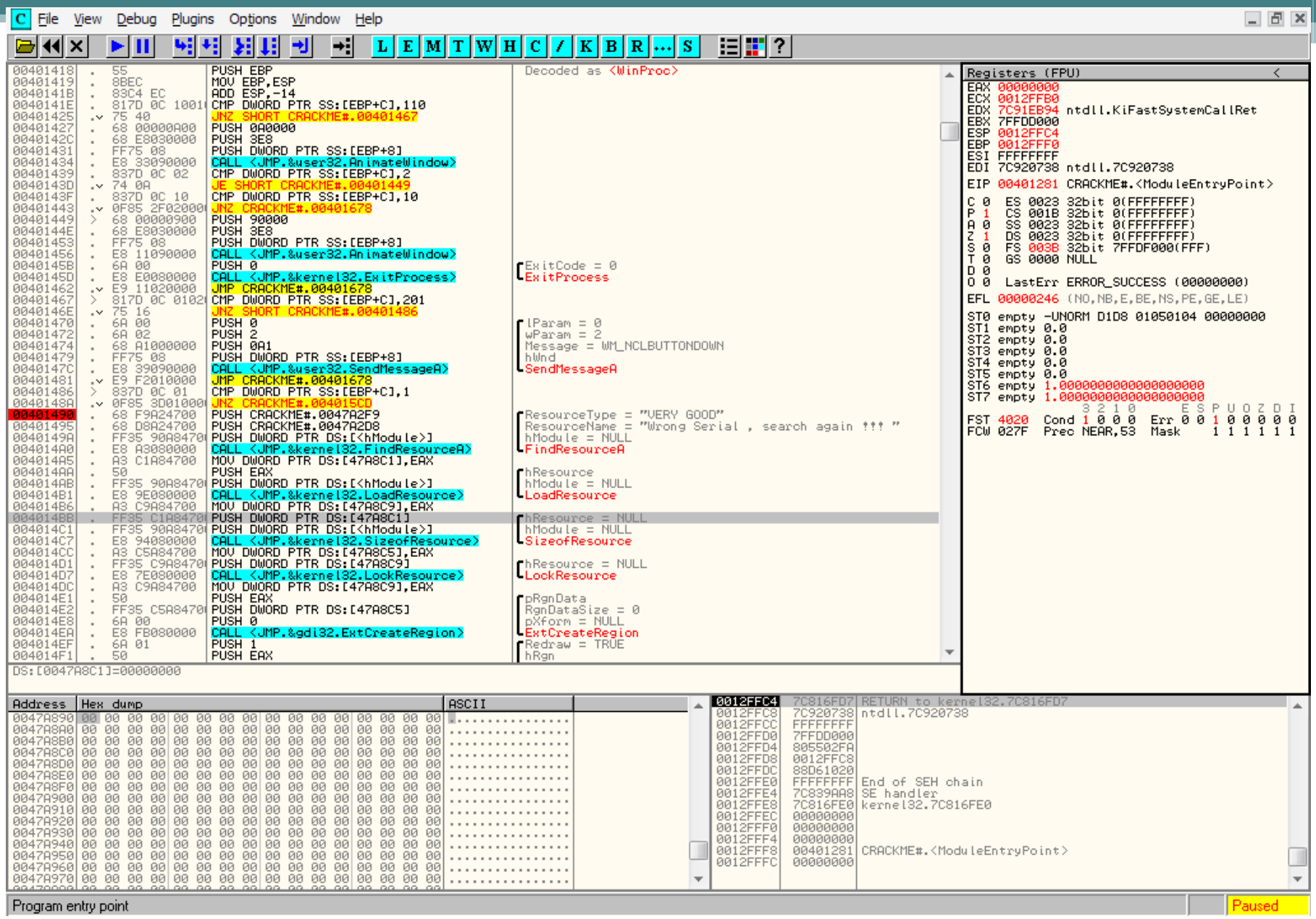
Les débogueurs sont associés à un moteur de désassemblage souvent moins puissant que les désassembleur 'purs' mais qui permettent de lire confortablement le code (coloration syntaxique), de le commenter et de définir quelques labels globaux .

Nous utiliserons OllyDBG, c'est un débogueur gratuit associé à un bon désassembleur qui facilite la programmation de modules.

L’analyse dynamique

L’analyse Dynamique est plus aisée que l’analyse statique, elle permet :

- Le suivit du déroulement du programme pas-à-pas
- La surveillance des variables et des registres en direct
- D’agir sur le déroulement du programme en modifiant les flags, le code encore une fois en direct
- De visualiser le fonctionnement du programme


File View Debug Plugins Options Window Help
00401418 . 55 PUSH EBP
00401419 . 8BEC MOV EBP,ESP
0040141B . 83C4 EC ADD ESP,-14
0040141E . 817D 0C 1001 CMP DWORD PTR SS:[EBP+C],110
00401425 .v 75 40 JNZ SHORT CRACKME#.00401467
00401427 . 68 00000A00 PUSH 0A0000
0040142C . 68 E8030000 PUSH 3E8
00401431 . FF75 08 PUSH DWORD PTR SS:[EBP+8]
00401434 . E8 33090000 CALL <JMP.&user32.AnimateWindow>
00401439 . 837D 0C 02 CMP DWORD PTR SS:[EBP+C],2
0040143D .v 74 0A JE SHORT CRACKME#.00401449
0040143F . 837D 0C 10 CMP DWORD PTR SS:[EBP+C],10
00401443 .v 0F85 2F020000 JNZ CRACKME#.00401678
00401449 > 68 00000900 PUSH 90000
0040144E . 68 E8030000 PUSH 3E8
00401453 . FF75 08 PUSH DWORD PTR SS:[EBP+8]
00401456 . E8 11090000 CALL <JMP.&user32.AnimateWindow>
0040145B . 6A 00 PUSH 0
0040145D . E8 E0080000 CALL <JMP.&kernel32.ExitProcess>
00401462 .v E9 11020000 JMP CRACKME#.00401678
00401467 > 817D 0C 0102 CMP DWORD PTR SS:[EBP+C],201
0040146E .v 75 16 JNZ SHORT CRACKME#.00401486
00401470 . 6A 00 PUSH 0
00401472 . 6A 02 PUSH 2
00401474 . 68 A1000000 PUSH 0A1
00401479 . FF75 08 PUSH DWORD PTR SS:[EBP+8]
0040147C . E8 39090000 CALL <JMP.&user32.SendMessageA>
00401481 .v E9 F2010000 JMP CRACKME#.00401678
00401486 > 837D 0C 01 CMP DWORD PTR SS:[EBP+C],1
0040148A .v 0F85 3D01000 JNZ CRACKME#.004015CD
00401490 . 68 F9A24700 PUSH CRACKME#.0047A2F9
00401495 . 68 D8A24700 PUSH CRACKME#.0047A2D8
0040149A . FF35 90A8470 PUSH DWORD PTR DS:[<hModule>]
004014A0 . E8 A3080000 CALL <JMP.&kernel32.FindResourceA>
004014A5 . A3 C1A84700 MOV DWORD PTR DS:[47A8C1],EAX
004014AA . 50 PUSH EAX
004014AB . FF35 90A8470 PUSH DWORD PTR DS:[<hModule>]
004014B1 . E8 9E080000 CALL <JMP.&kernel32.LoadResource>
004014B6 . A3 C9A84700 MOV DWORD PTR DS:[47A8C9],EAX
004014BB . FF35 C1A8470 PUSH DWORD PTR DS:[47A8C1]
004014C1 . FF35 90A8470 PUSH DWORD PTR DS:[<hModule>]
004014C7 . E8 94080000 CALL <JMP.&kernel32.SizeofResource>
004014CC . A3 C5A84700 MOV DWORD PTR DS:[47A8C5],EAX
004014D1 . FF35 C9A8470 PUSH DWORD PTR DS:[47A8C9]
004014D7 . E8 7E080000 CALL <JMP.&kernel32.LockResource>
004014DC . A3 C9A84700 MOV DWORD PTR DS:[47A8C9],EAX
004014E1 . 50 PUSH EAX
004014E2 . FF35 C5A8470 PUSH DWORD PTR DS:[47A8C5]
004014E8 . 6A 00 PUSH 0
004014EA . E8 FB080000 CALL <JMP.&gdi32.ExtCreateRegion>
004014EF . 6A 01 PUSH 1
004014F1 . 50 PUSH EAX
Decoded as <WinProc>
ExitCode = 0
ExitProcess
lParam = 0
wParam = 2
Message = WM_NCLBUTTONDOWN
hWnd
SendMessageA
ResourceType = "VERY GOOD"
ResourceName = "Wrong Serial , search again !!! "
hModule = NULL
FindResourceA
hResource
hModule = NULL
LoadResource
hResource = NULL
hModule = NULL
SizeofResource
hResource = NULL
LockResource
pRgnData
RgnDataSize = 0
pXform = NULL
ExtCreateRegion
Redraw = TRUE
hRgn
DS:[0047A8C1]=00000000
Registers (FPU)
EAX 00000000
ECX 0012FFB0
EDX 7C91EB94 ntdll.KiFastSystemCallRet
EBX 7FFDD000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdll.7C920738
EIP 00401281 CRACKME#.<ModuleEntryPoint>
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDF000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty -UNORM D1D8 01050104 00000000
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 1.0000000000000000000
ST7 empty 1.0000000000000000000
3 2 1 0 E S P U O Z D I
FST 4020 Cond 1 0 0 0 Err 0 0 1 0 0 0 0 0
FCW 027F Prec NEAR,53 Mask 1 1 1 1 1 1
Address Hex dump ASCII
0047A890 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0047A8A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0047A8B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0047A8C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0047A8D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0047A8E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0047A8F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0047A900 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0047A910 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0047A920 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0047A930 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0047A940 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0047A950 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0047A960 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0047A970 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012FFC4 7C816FD7 RETURN to kernel32.7C816FD7
0012FFC8 7C920738 ntdll.7C920738
0012FFCC FFFFFFFF
0012FFD0 7FFDD000
0012FFD4 805502FA
0012FFD8 0012FFC8
0012FFDC 88D61020
0012FFE0 FFFFFFFF End of SEH chain
0012FFE4 7C839AA8 SE handler
0012FFE8 7C816FE0 kernel32.7C816FE0
0012FFEC 00000000
0012FFF0 00000000
0012FFF4 00000000
0012FFF8 00401281 CRACKME#.<ModuleEntryPoint>
0012FFFC 00000000
Program entry point
Paused

OllyDBG 1.10

Protéger soit même son programme

- Numéro de série
  - Avantages / Défauts / Solutions
- Obfuscation du code
  - Principe / Réalisation
- Chiffrement du code
- Vérifications furtives
  - Comment mettre en place des vérifications cachées de façon efficace

Protection par numéro de série

Atouts

- Facilité d'implémentation
- Possibilité de reposer la génération du n° de série sur des caractéristiques matérielles
- Nombres d'algorithme de vérification et de génération de clefs infini
- Possibilité de vérifier le n° de série sur un serveur via internet

Défauts

- On peut, à partir du code de vérification, retrouver le code de génération du n° de série
- Il est toujours possible de modifier le code de l'application et fausser la vérification
- Nécessite une bonne maitrise en programmation et en maths pour espérer faire un algorithme solide

Routine de vérification de n° de série basique

```
.  6A 64          PUSH 64
.  68 E4204000    PUSH OFFSET <Nom>
.  68 4C040000    PUSH 44C
.  FF75 08        PUSH [ARG.1]
.  E8 932F0000    CALL <JMP.&USER32.GetDlgItemTextA>
.  A3 AC214000    MOV [<Longueur Nom>],EAX
.  85C0           TEST EAX,EAX
.v 75 15          JNZ SHORT 00401091
.  6A 00          PUSH 0
.  6A 00          PUSH 0
.  68 30204000    PUSH 00402030
.  6A 00          PUSH 0
.  E8 802F0000    CALL <JMP.&USER32.MessageBoxA>
.v E9 73000000    JMP <Fin>
>  6A 64          PUSH 64
.  68 48214000    PUSH OFFSET <n de serie>
.  68 4D040000    PUSH 44D
.  FF75 08        PUSH [ARG.1]
.  E8 612F0000    CALL <JMP.&USER32.GetDlgItemTextA>
.  A3 B0214000    MOV [<longueur n de serie>],EAX
.  85C0           TEST EAX,EAX
.v 75 15          JNZ SHORT 004010C3
.  6A 00          PUSH 0
.  6A 00          PUSH 0
.  68 3E204000    PUSH 0040203E
.  6A 00          PUSH 0
.  E8 4E2F0000    CALL <JMP.&USER32.MessageBoxA>
.v E9 41000000    JMP <Fin>
>  3905 AC214000  CMP [<Longueur Nom>],EAX
.v 0F85 3A000000  JNZ <Mauvais n de serie>
.  B9 00000000    MOV ECX,0
>  8A81 E4204000  MOV AL,[ECX+<Nom>]
.  04 0C          ADD AL,0C
.  3A81 48214000  CMP AL,[ECX+<n de serie>]
.v 0F85 21000000  JNZ <Mauvais n de serie>
.  41             INC ECX
.  3B0D AC214000  CMP ECX,[<Longueur Nom>]
.^ 75 E3          JNZ SHORT <loop>
.  6A 00          PUSH 0
.  68 10204000    PUSH 00402010
.  68 00204000    PUSH 00402000
.  6A 00          PUSH 0
.  E8 082F0000    CALL <JMP.&USER32.MessageBoxA>
>v E9 3A000000    JMP <Reprise du programme>
>  6A 00          PUSH 0
.  6A 00          PUSH 0
.  68 18204000    PUSH 00402018
.  6A 00          PUSH 0
.  E8 F32E0000    CALL <JMP.&USER32.MessageBoxA>
.v E9 25000000    JMP <Reprise du programme>
```

```
Count = 64 (100.)
Buffer = OFFSET <crackme.Nom>
ControlID = 44C (1100.)
hWnd
GetDlgItemTextA


Style = MB_OK|MB_APPLMODAL
Title = NULL
Text = "Entrez un nom"
hOwner = NULL
MessageBoxA

Count = 64 (100.)
Buffer = OFFSET <crackme.n de serie>
ControlID = 44D (1101.)
hWnd
GetDlgItemTextA


Style = MB_OK|MB_APPLMODAL
Title = NULL
Text = "Entrez un n° de sürie"
hOwner = NULL
MessageBoxA

Style = MB_OK|MB_APPLMODAL
Title = "Bravo !"
Text = "Bon n° de sürie"
hOwner = NULL
MessageBoxA

Style = MB_OK|MB_APPLMODAL
Title = NULL
Text = "Mauvais n° de sürie !!!"
hOwner = NULL
MessageBoxA
```

Comment pallier à ces défauts ?

Défauts

1. Reverse du code de vérification
2. Il est toujours possible de modifier le code de l'application et fausser la vérification
3. Nécessite une bonne maitrise en programmation et en maths pour espérer faire un algorithme solide

Solutions

1. Obfuscation du code, ajouts de vérifications cachée et additionnelles lors du déroulement du programme
2. Chiffrement du code, vérification d'intégrité du code
3. Utilisation d'algorithmes asymétriques, utiliser si possible une composante matérielle

L’obfuscation

L’obfuscation a pour but de rendre le code de l’application incompréhensible directement par l’homme.

Le code devra être traité avant de pouvoir être analysé et si l’obfuscation est bien faite, la programmation d’un outil qui éliminera entièrement et automatiquement l’obfuscation sera quasi impossible.

Différentes sortes d'obfuscation :

- Rendre le code le moins clair possible, utiliser des moyens détournés pour arriver au même résultat :

```
var1 = var2 + var3 -> var1 = (var1 + var2*const + var3*const – var 1) / const
```

- Insertion de code, de tests inutiles :

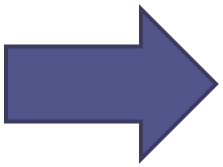
```
var1 += var2 [...] var1 -= var2
```

```
If (TRUE) { [...] } else { [...] <- ce code ne sera jamais exécuté }
```

- L'utilisation massive de sauts entre les instructions, ce qui rend illisible le code

Exemple d’obfuscation de code *

mov al, byte ptr [edi]
or al, al



```
0043B7AB    87EE            XCHG ESI,EBP
0043B7AD  v E9 A7040000     JMP 0043BC59
0043B7B2    B2 E9           MOV DL,0E9
0043B7B4    CB              RETF                        Far return
0043B7B5    0100            ADD [EAX],EAX
0043B7B7    002A            ADD [EDX],CH
0043B7B9    94              XCHG EAX,ESP
0043B7BA  v E9 E9050000     JMP 0043BDA8
0043B7BF    1BE9            SBB EBP,ECX
0043B7C1    3C 05           CMP AL,5
0043B7C3    0000            ADD [EAX],AL
0043B7C5    A0 B7762B44     MOV AL,[442B76B7]
0043B7CA    03E9            ADD EBP,ECX
0043B7CC    6903 0000EE36   IMUL EAX,[EBX],36EE0000
0043B7D2  v E9 D1050000     JMP 0043BDA8
0043B7D7    94              XCHG EAX,ESP
0043B7D8    45              INC EBP
0043B7D9  v E9 97010000     JMP 0043B975
0043B7DE    AE              SCAS BYTE PTR ES:[EDI]
0043B7DF  v E9 8A010000     JMP 0043B96E
0043B7E4  ^ 76 C6           JBE SHORT 0043B7AC
0043B7E6  v 0F87 C5020000   JA 0043BAB1
0043B7EC    A1 53E9B505     MOV EAX,[5B5E953]
0043B7F1    0000            ADD [EAX],AL
0043B7F3    50              PUSH EAX
0043B7F4  v E9 AF050000     JMP 0043BDA8
0043B7F9    6C              INS BYTE PTR ES:[EDI],DX    I/O command
0043B7FA  v E9 CC000000     JMP 0043B8CB
0043B7FF    53              PUSH EBX
0043B800  v E9 A3050000     JMP 0043BDA8
0043B805    D3E9            SHR ECX,CL
0043B807    14 03           ADC AL,3
0043B809    0000            ADD [EAX],AL
0043B80B    80E9 E6         SUB CL,0E6
0043B80E    0000            ADD [EAX],AL
0043B810    00B8 7615AEE9   ADD [EAX+E9AE1576],BH
0043B816    DA0400          FIADD DWORD PTR [EAX+EAX]
0043B819    0076 C6         ADD [ESI-3A],DH
0043B81C    0F8A 09020000   JPE 0043BA2B
0043B822  v E9 C9040000     JMP 0043BCF0
0043B827    61              POPAD
0043B828    F3:             PREFIX REP:                 Superfluous prefix
0043B829  v E9 7A050000     JMP 0043BDA8
0043B82E    28E9            SUB CL,CH
0043B830  v 74 05           JE SHORT 0043B837
0043B832    0000            ADD [EAX],AL
0043B834    26:0F84 6D0200( JE 0043BAA8                 Superfluous prefix
0043B83B    C3              RETN
0043B83C    DF              ???                         Unknown command
0043B83D  v 0F85 27030000   JNZ 0043BB6A
0043B843    3E:60           PUSHAD                      Superfluous prefix
0043B845  v E9 1C010000     JMP 0043B966
```

* Obfuscation du YO-bfuscator I

Autre exemple de code obfusqué *:

```
0040150B    AF              SCAS DWORD PTR ES:[EDI]
0040150C    44              INC ESP
0040150D    00F3            ADD BL,DH
0040150F    8D8D 11B84400   LEA ECX,[EBP+44B811]
00401515    F2:             PREFIX REPNE:                   Superfluous prefix
00401516    2BCF            SUB ECX,EDI
00401518    F3:             PREFIX REP:                     Superfluous prefix
00401519    000F            ADD [EDI],CL
0040151B    8037 D2         XOR BYTE PTR [EDI],0D2
0040151E    F2:             PREFIX REPNE:                   Superfluous prefix
0040151F    83EC 08         SUB ESP,8
00401522    F3:             PREFIX REP:                     Superfluous prefix
00401523    8D6424 04       LEA ESP,[ESP+4]
00401527    F2:             PREFIX REPNE:                   Superfluous prefix
00401528    893C24          MOV [ESP],EDI
0040152B    8BF8            MOV EDI,EAX
0040152D    F2:             PREFIX REPNE:                   Superfluous prefix
0040152E    2BFA            SUB EDI,EDX
00401530    F3:             PREFIX REP:                     Superfluous prefix
00401531    F2:             PREFIX REPNE:                   Superfluous prefix
00401532    83EC 08         SUB ESP,8
00401535    F3:             PREFIX REP:                     Superfluous prefix
00401536    8D6424 04       LEA ESP,[ESP+4]
0040153A    F2:             PREFIX REPNE:                   Superfluous prefix
0040153B    890424          MOV [ESP],EAX
0040153E    2BC2            SUB EAX,EDX
00401540    F2:             PREFIX REPNE:                   Superfluous prefix
00401541    8D41 FF         LEA EAX,[ECX-1]
00401544    E8 01000000     CALL 0040154A
00401549    E8 E8020000     CALL 00401836
0040154E    00CD            ADD CH,CL
00401550    2083 04240B83   AND [EBX+830B2404],AL
00401556    44              INC ESP
00401557    24 04           AND AL,4
00401559    13C3            ADC EAX,EBX
0040155B  - E9 8D7801F2     JMP F2418DED
00401560    83EC 08         SUB ESP,8
00401563    F3:             PREFIX REP:                     Superfluous prefix
00401564    8D6424 04       LEA ESP,[ESP+4]
00401568    F2:             PREFIX REPNE:                   Superfluous prefix
00401569    890C24          MOV [ESP],ECX
0040156C    1BCA            SBB ECX,EDX
0040156E    F2:             PREFIX REPNE:                   Superfluous prefix
0040156F    8D88 2D3A4400   LEA ECX,[EAX+443A2D]
00401575    8D0C02          LEA ECX,[EDX+EAX]
00401578    F3:             PREFIX REP:                     Superfluous prefix
00401579    03C9            ADD ECX,ECX
0040157B    B9 52604000     MOV ECX,00406052
00401580    F2:             PREFIX REPNE:                   Superfluous prefix
00401581    C1F8 00         SAR EAX,0                       Shift constant out of range 1..31
00401584  v EB 01           JMP SHORT 00401587
00401586  - E9 8B0C24F3     JMP F3642216
```

* layer obfsusqué du Kaine5

Exemple d'utilisation massive de sauts dans le code *

```
00781B8E  v E9 2CC30700     JMP 007FDEBF
00781B93    47              INC EDI
00781B94    81C3 F5010000   ADD EBX,1F5
00781B9A  - E9 6D1BCCFF     JMP 0044370C
00781B9F    1D 0BEDE9E1     SBB EAX,E1E9ED0B
00781BA4    2202            AND AL,[EDX]
00781BA6    0021            ADD [ECX],AH
00781BA8    8BEC            MOV EBP,ESP
00781BAA  ^ E9 5198FFFF     JMP 0077B400
00781BAF    53              PUSH EBX
00781BB0    6A 64           PUSH 64
00781BB2  ^ E9 F29BFFFF     JMP 0077B7A9
00781BB7    36:50           PUSH EAX                    Superfluous prefix
00781BB9  v 75 6C           JNZ SHORT 00781C27
00781BBB  v 73 61           JNB SHORT 00781C1E
00781BBD  v 72 00           JB SHORT 00781BBF
00781BBF  v E9 CD200200     JMP 007A3C91
00781BC4  v E3 FF           JECXZ SHORT 00781BC5
00781BC6    35 B9EE7700     XOR EAX,77EEB9
00781BCB  v E9 379C0B00     JMP 0083B807
00781BD0    8D6B 68         LEA EBP,[EBX+68]
00781BD3    61              POPAD
00781BD4  v 72 6E           JB SHORT 00781C44
00781BD6    65:74 68        JE SHORT 00781C41           Superfluous prefix
00781BD9    00E9            ADD CL,CH
00781BDB    0B0E            OR ECX,[ESI]
00781BDD    04 00           ADD AL,0
00781BDF    B6 00           MOV DH,0
00781BE1    0000            ADD [EAX],AL
00781BE3    00E9            ADD CL,CH
00781BE5    A0 51C8FFE7     MOV AL,[E7FFC851]
00781BEA  ^ E9 2662F4FF     JMP 006C7E15
00781BEF    59              POP ECX
00781BF0  - E9 7A10D7FF     JMP 004F2C6F
00781BF5    42              INC EDX
00781BF6  v E9 C1270000     JMP 007843BC
00781BFB  - E9 DAA6D1FF     JMP 0049C2DA
00781C00    84D8            TEST AL,BL
00781C02  ^ E9 6098FFFF     JMP 0077B467
00781C07    64:55           PUSH EBP                    Superfluous prefix
00781C09  - E9 6812CCFF     JMP 00442E76
00781C0E    1B42 65         SBB EAX,[EDX+65]
00781C11    61              POPAD
00781C12  v 74 72           JE SHORT 00781C86
```

* Obfuscation du crack-me Meloquynthe

Chiffrement du code

Au chargement du programme ou au début d'une fonction, le code va être déchiffrer avant d'être exécuté

Atouts

- Le code avant d'avoir été déchiffré est illisible
- Le code devant être déchiffré avant d'être exécuté, il n'est plus possible de le modifier directement
- Associé à l'obfuscation, le chiffrement est souvent efficace contre les crackers de niveau moyen

Défauts

- Il est difficile d'implémenter ce genre de protection, la programmation d'outils et souvent nécessaire
- Il est toujours possible d'enregistrer le code déchiffré et donc de le modifier ou de rajouter une fonction au programme qui se chargera de le modifier une fois déchiffré (*inline patching*)

Code chiffré :

```
00401000   B9 86010000     MOV ECX,186                    \
00401005   3089 19104000   XOR [ECX+401019],CL             |
0040100B   0089 19104000   ADD [ECX+401019],CL             |> routine de dechiffrement
00401011   80A9 19104000   SUB BYTE PTR [ECX+401019],12    |
00401018 ^ E2 EB           LOOPD SHORT 00401005           /
0040101A   CB              RETF                           Far return
0040101B   8013 0A         ADC BYTE PTR [EBX],0A
0040101E   0832            OR [EDX],DH
00401020   93              XCHG EAX,EBX
00401021   3F              AAS
00401022   1042 0C         ADC [EDX+C],AL
00401025   CA 8B3F         RETF 3F8B                      Far return
00401028   1C 52           SBB AL,52
0040102A   1017            ADC [EDI],DL
0040102C   F2:             PREFIX REPNE:                  Superfluous prefix
0040102D   FB              STI
0040102E   DEFE            FDIVP ST(6),ST
00401030   11E6            ADC ESI,ESP
00401032   E7 1E           OUT 1E,EAX                     I/O command
00401034   51              PUSH ECX
00401035   0C 63           OR AL,63
00401037   1E              PUSH DS
00401038 v E1 0E           LOOPDE SHORT 00401048
0040103A   2328            AND EBP,[EAX]
0040103C   335A 25         XOR EBX,[EDX+25]
0040103F   D8B8 3DAC2303   FDIVR DWORD PTR [EAX+323AC3D]
00401045   B5 BA           MOV CH,0BA
00401047   B9 D8987385     MOV ECX,857398D8
0040104C   25 A3A3CE75     AND EAX,75CEA3A3
00401051   8F              ???                            Unknown command
00401052   2F              DAS
00401053   AD              LODS DWORD PTR [ESI]
00401054 v 79 67           JNS SHORT 004010BD
00401056   44              INC ESP
00401057   2E:0151 26       ADD CS:[ECX+26],EDX
0040105B   CC              INT3
0040105C v 75 7C           JNZ SHORT 004010DA
0040105E   D4 53           AAM 53
00401060   C9              LEAVE
00401061   99              CDQ
00401062   03B3 0DB54B0A   ADD ESI,[EBX+A4BB50D]
00401068   D070 B6         SAL BYTE PTR [EAX-4A],1
0040106B   4E              DEC ESI
0040106C   B4 BC           MOV AH,0BC
0040106E   65:6279 89      BOUND EDI,GS:[ECX-77]          Superfluous prefix
00401072   8341 3F 74       ADD DWORD PTR [ECX+3F],74
00401076   DF              ???                            Unknown command
00401077   8F              ???                            Unknown command
00401078   21B0 5A2C151C   AND [EAX+1C152C5A],ESI
0040107E   88E5            MOV CH,AH
00401080   AF              SCAS DWORD PTR ES:[EDI]
00401081 v EB 7D           JMP SHORT 00401100
00401083   EE              OUT DX,AL                      I/O command
```

Code déchiffré :

```
00401000    B9 86010000     MOV ECX,186                      \
00401005    3089 19104000   XOR [ECX+401019],CL               |
0040100B    0089 19104000   ADD [ECX+401019],CL              |
00401011    80A9 19104000   SUB BYTE PTR [ECX+401019],12     |
00401018  ^ E2 EB           LOOPD SHORT 00401005             |
0040101A    B9 72010000     MOV ECX,172                      |
0040101F    2889 2D104000   SUB [ECX+40102D],CL              |
00401025    C081 2D104000 ( ROL BYTE PTR [ECX+40102D],5      |
0040102C  ^ E2 F1           LOOPD SHORT 0040101F             |  > routines successives de dechiffrement
0040102E    B9 5D010000     MOV ECX,15D                      |
00401033    C089 42104000 ( ROR BYTE PTR [ECX+401042],6      |
0040103A    8081 42104000 ( ADD BYTE PTR [ECX+401042],34     |
00401041  ^ E2 F0           LOOPD SHORT 00401033             |
00401043    B9 49010000     MOV ECX,149                      |
00401048    3089 56104000   XOR [ECX+401056],CL              |
0040104E    80B1 56104000 ! XOR BYTE PTR [ECX+401056],56     |
00401055  ^ E2 F1           LOOPD SHORT 00401048             /
00401057    6A 00           PUSH 0                           -> debut du code dechiffre
00401059    E8 BA2F0000     CALL <JMP.&KERNEL32.GetModuleHandleA>
0040105E    A3 E0204000     MOV [4020E0],EAX
00401063    6A 00           PUSH 0
00401065    68 7E104000     PUSH 0040107E
0040106A    6A 00           PUSH 0
0040106C    68 E8030000     PUSH 3E8
00401071    50              PUSH EAX
00401072    E8 892F0000     CALL <JMP.&USER32.DialogBoxParamA>
00401077    6A 00           PUSH 0
00401079    E8 A02F0000     CALL <JMP.&KERNEL32.ExitProcess>
0040107E    55              PUSH EBP
0040107F    89E5            MOV EBP,ESP
00401081    8B45 0C         MOV EAX,[EBP+C]
00401084    3D 11010000     CMP EAX,111
00401089  v 0F84 08000000   JE 00401097
0040108F    31C0            XOR EAX,EAX
00401091    89EC            MOV ESP,EBP
00401093    5D              POP EBP
00401094    C2 1000         RETN 10
00401097    8B45 10         MOV EAX,[EBP+10]
0040109A    2D E9030000     SUB EAX,3E9
0040109F  v 0F84 10000000   JE 004010B5
004010A5    48              DEC EAX
004010A6  v 0F84 D5000000   JE 00401181
004010AC    48              DEC EAX
004010AD  v 0F84 C1000000   JE 00401174
004010B3  ^ EB DA           JMP SHORT 0040108F
004010B5    6A 64           PUSH 64
004010B7    68 E4204000     PUSH 004020E4
004010BC    68 4C040000     PUSH 44C
004010C1    FF75 08         PUSH DWORD PTR [EBP+8]
004010C4    E8 3D2F0000     CALL <JMP.&USER32.GetDlgItemTextA>
004010C9    A3 AC214000     MOV [4021AC],EAX
004010CE    85C0            TEST EAX,EAX
004010D0  v 75 15           JNZ SHORT 004010E7
004010D2    6A 00           PUSH 0
```

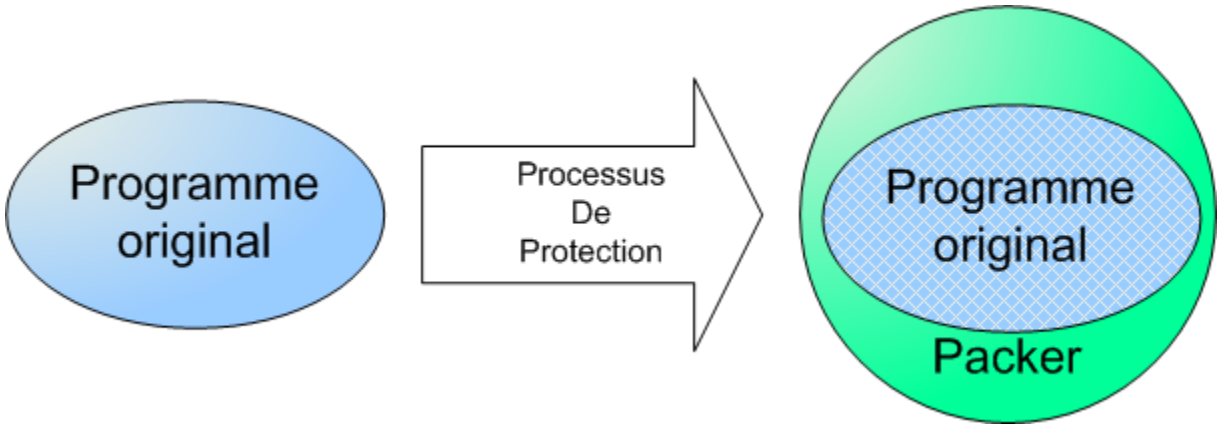
Vérifications furtives

- Faire des vérifications sur le n° de série à des endroits critiques du programme
- Utiliser différents algorithmes de vérifications (sur différentes parties du n° par exemple)
- Faire des vérification d'intégrité du code des parties sensible de façon aléatoire dans le temps
- Les conséquences d'une détection d'un piratage doivent être handicapantes pour l'utilisateur mais difficiles à localiser pour le cracker (plantage de l'application, modification des résultats attendus, menus inactifs une fois sur deux etc.)
- Les possibilités sont infinies, vous n'êtes bridés que par votre inventivité.

Les protections commerciales

- Principe des *packers*
  - Comment fonctionnent-ils
  - Quels sont leurs intérêt
- Redirections d'APIs win32 et anti-débogueurs
  - Comment les *packers* protègent vos applications contres la modification, l'etude
- Techniques de fusion avec l'application
  - Comment les packers fusionnent avec vos applications pour un maximum d'efficacité.
- Quelques exemples de fonctionnalités de packers
  - Les triggers de SecuROM©
  - La virtualisation de VMProtect©

Schéma simplifié du fonctionnement d'un *packer*



Redirections d'APIs

Pour fonctionner une application a besoin d'utiliser certaines fonctions extérieures a son propre code pour, par exemple, afficher quelque chose a l'écran

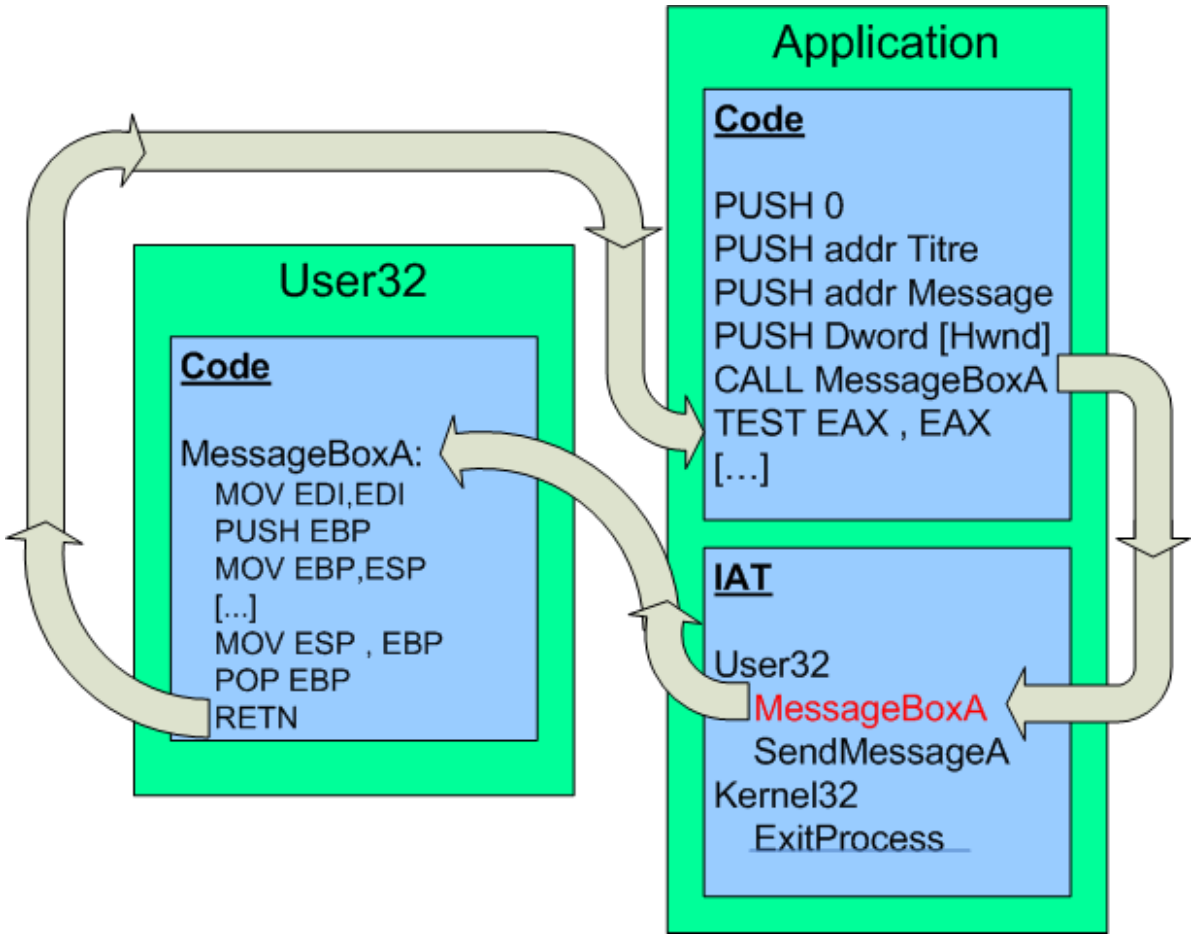
Les adresses de ces fonctions sont situés dans l'IAT (*Import Address Table*) du programme qui est remplie par Windows© à l'initialisation du processus

Les *packers* détruisent cette IAT et remplacent les adresses des fonctions par des adresses pointant sur leur propre code qui se chargera de rediriger le programme sur la bonne API.

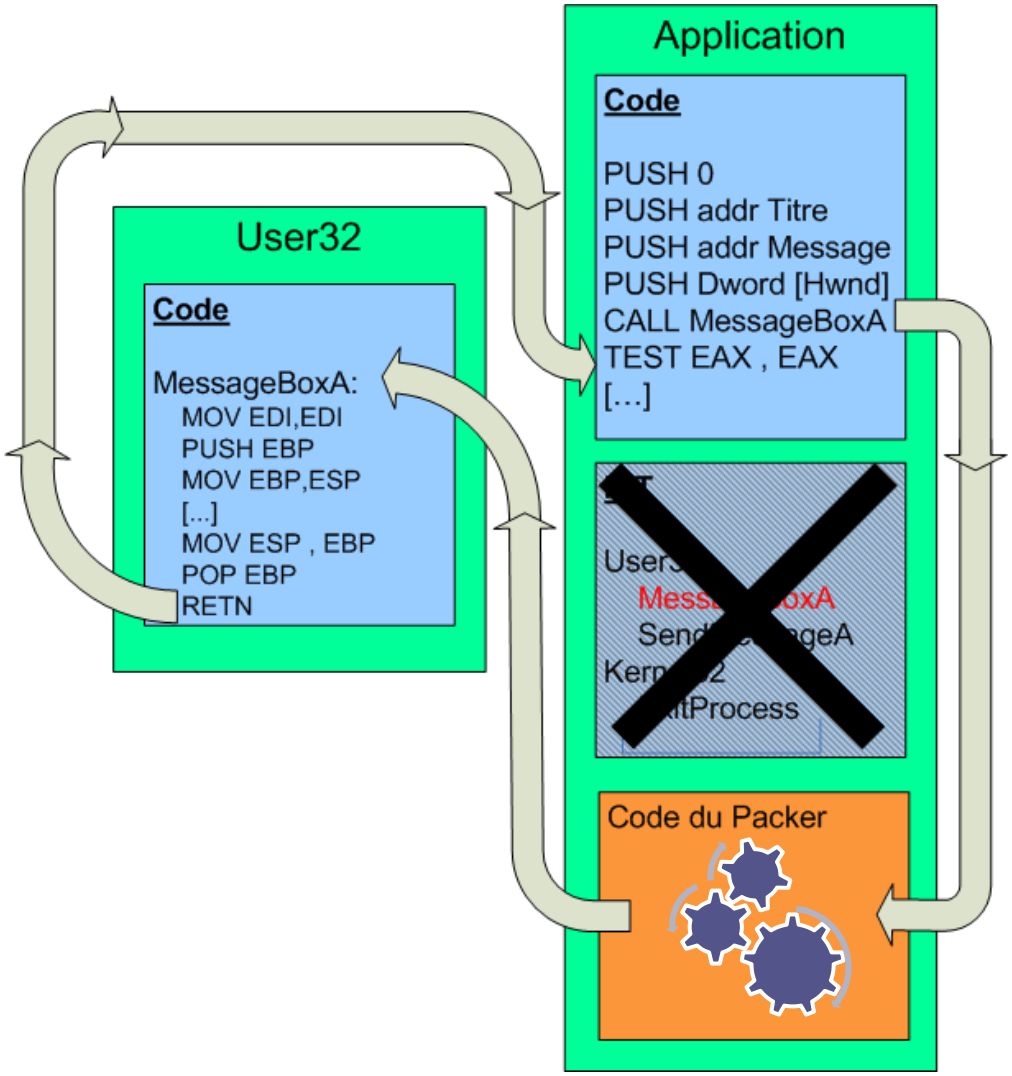
Pour enlever la protection offerte par le *packer* il faudra donc reconstruire cette table

Il sera généralement nécessaire de coder des outils différents pour automatiser cette reconstruction suivant les *packers*.

Application originale



Redirections d'APIs win32
Application protégée



Les anti-débogueurs

Atouts

- Détectent les débogueurs et donc une tentative de piratage
- Freinent l'analyse
- Fonctionnements très variés
- Ils peuvent cibler tout les débogueurs …
- … ou des outils précis (il existe ainsi des 'anti-OllyDBG')
- Il peuvent être utilisé dans des vérifications furtives

Défauts

- Certains outils sont spécialisés dans l'amélioration de la 'furtivité' du débogueur
- Le débogueur ayant le contrôle de l'application, il peut neutraliser les effets de l'anti-débogueur et empêcher sa détection
- Il sont parfois très intrusifs et peuvent rendre le système instable

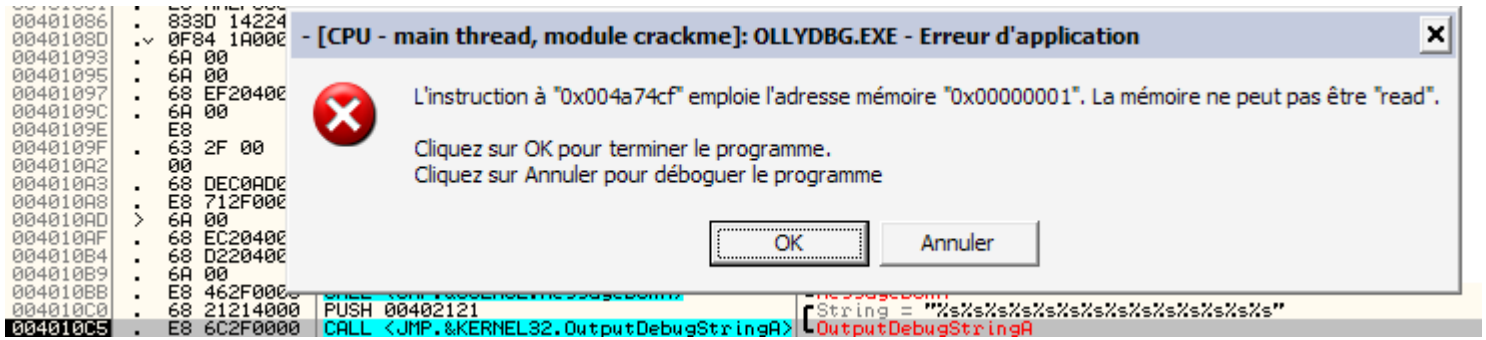
Exemples simples d'anti-débogueurs

Anti-débogueurs génériques :

```
00401069  >  E8 B62F0000   CALL <JMP.&KERNEL32.IsDebuggerPresent>      IsDebuggerPresent
0040106E  .  85C0          TEST EAX,EAX
00401070  .v 0F84 1A000000 JE <Pas de debugger>
00401076  .  6A 00         PUSH 0                                     Style = MB_OK|MB_APPLMODAL
00401078  .  6A 00         PUSH 0                                     Title = NULL
0040107A  .  68 EF204000   PUSH 004020EF                              Text = "Debugger Detectü !!!♪◘Le programme va se terminer"
0040107F  .  6A 00         PUSH 0                                     hOwner = NULL
00401081  .  E8 802F0000   CALL <JMP.&USER32.MessageBoxA>             MessageBoxA
00401086  .  68 DEC0AD0B   PUSH 0BADC0DE                              ExitCode = BADC0DE
0040108B  .  E8 8E2F0000   CALL <JMP.&KERNEL32.ExitProcess>           ExitProcess
00401090  >  6A 00         PUSH 0                                     Style = MB_OK|MB_APPLMODAL
00401092  .  68 EC204000   PUSH 004020EC                              Title = "OK"
00401097  .  68 D2204000   PUSH 004020D2                              Text = "Pas de debugger dütüctü !"
0040109C  .  6A 00         PUSH 0                                     hOwner = NULL
0040109E  .  E8 632F0000   CALL <JMP.&USER32.MessageBoxA>             MessageBoxA
```

```
00401069  >  E8 B62F0000   CALL <JMP.&KERNEL32.GetCurrentProcessId>   GetCurrentProcessId
0040106E  .  50            PUSH EAX                                   ProcessId
0040106F  .  6A 00         PUSH 0                                     Inheritable = FALSE
00401071  .  68 FF0F1F00   PUSH 1F0FFF                                Access = PROCESS_ALL_ACCESS
00401076  .  E8 AF2F0000   CALL <JMP.&KERNEL32.OpenProcess>           OpenProcess
0040107B  .  68 14224000   PUSH 00402214
00401080  .  50            PUSH EAX
00401081  .  E8 AA2F0000   CALL <JMP.&KERNEL32.CheckRemoteDebugger
00401086  .  833D 1422400( CMP DWORD PTR [402214],0
0040108D  .v 0F84 1A000000 JE <Pas de debugger>
```

Anti-débogueur exploitant une faille de OllyDBG :



Les anti-désassembleurs

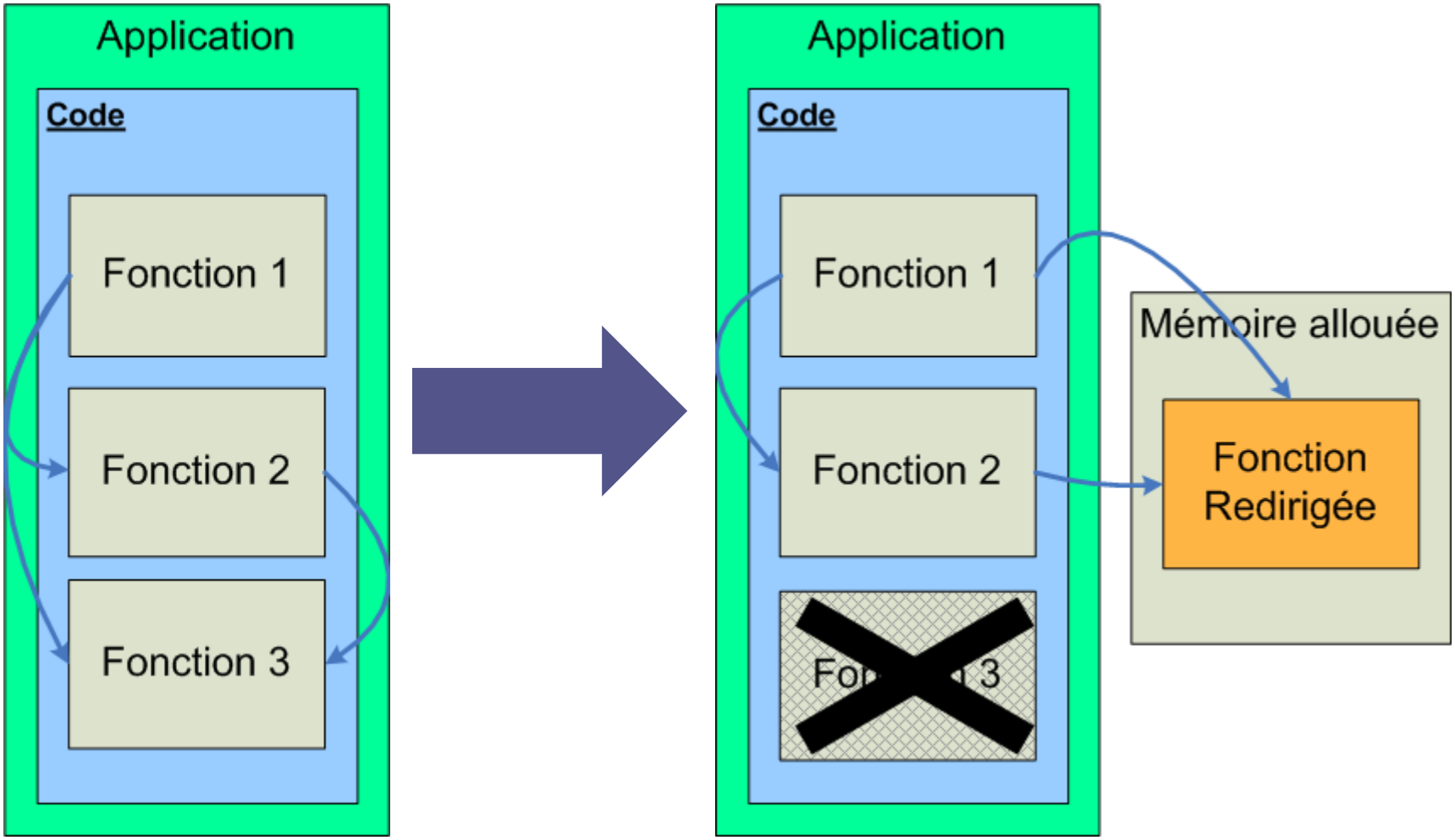
Ils cherchent à dérouter l'analyse du debugger.
Quelques anti-désassembleurs :

- Le code changeant d'apparence : c'est un code qui va s'auto modifier au cour de son exécution, le désassembleur ne pouvant pas prévoir les changements dans le code, l'analyse s'arrête.
- L'utilisation des registres et de l'instruction *retn* pour se déplacer dans le code, le désassembleur ne pouvant pas prédire la valeur des registres, il ne peut plus analyser le flot de code
- L'utilisation d'instructions non standards ou non documentées

Fusion avec l’application

- Redirection de code
- Utilisation d’un SDK de la protection

Redirection de code



Les SDKs des protections

Les protections mettent à la disposition des développeurs des fonctions préprogrammées qui permettent de mettre en place des vérifications cachées, des contrôles du code, des protections de certaines fonctions sensibles du programme.

Exemples :

- Les triggers de SecuROM©
- Les fonctions de déchiffrements pour les protections par *dongles*
- Les fonctions qui permettent de gérer les différentes options des licences
- etc.

Les *triggers* de SecuROM©

SecuROM est une *packer* utilisé dans la protection des jeux vidéos, il a beaucoup fait parler de lui en 2002 avec une nouvelle forme de protection anti-cracking : les triggers.

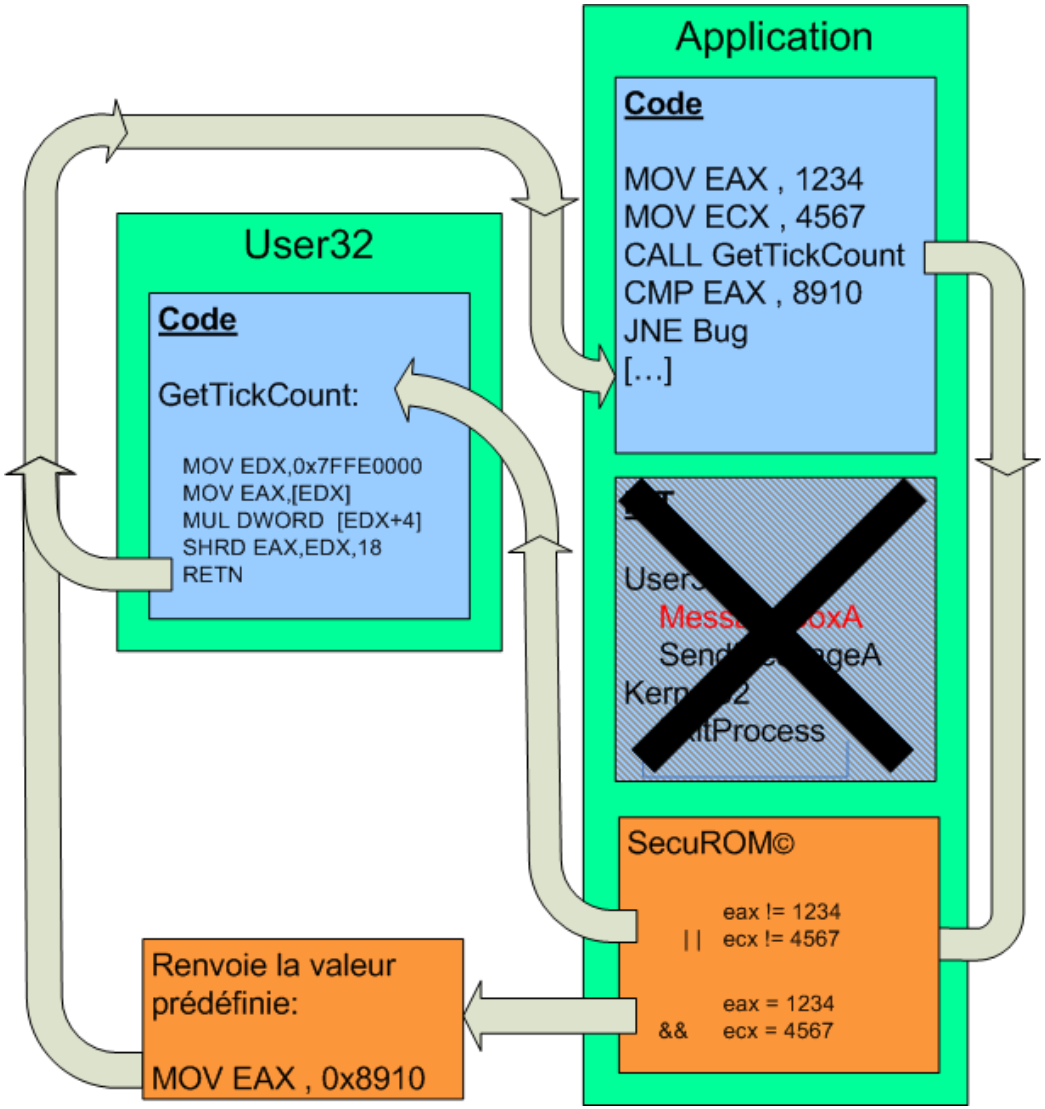
Les triggers sont une fonctionnalité offerte par SecuROM qui permet aux développeurs d'utiliser les APIs Windows© pour réaliser des vérifications furtives du code pour s'assurer que la protection du jeux n'a pas été enlevée.

Il suffit au développeur de déclarer que si telle API est appelée avec tels arguments alors elle doit renvoyer une valeur prédéterminée.

Par exemple, si les registre eax et ecx contiennent les valeurs 1234 et 4567 lors d'un appel a GetTickCount alors cette API devra renvoyer 8910.

L'API GetTickCount sera alors filtrée par SecuROM qui se chargera de renvoyer la valeur prédéfinie si eax et ecx contiennent 1234 et 4567

Schéma simplifié du fonctionnement des triggers :

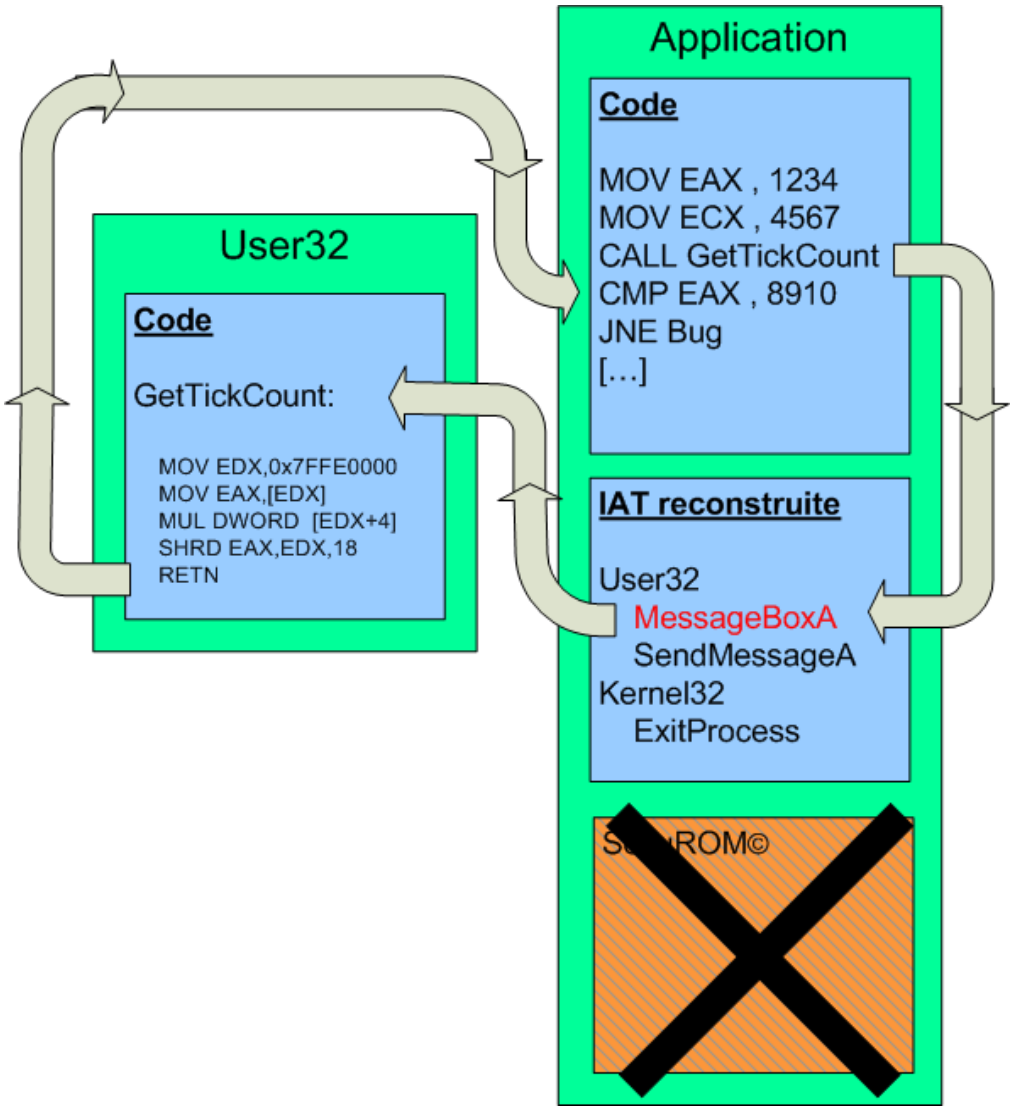


Si la protection est supprimée, l'IAT reconstruite, la redirection des APIs n'est plus assurée par SecuROM©, le programme appelle directement les APIs

Comme les APIs ne sont redirigées vers le code de SecuROM, leurs valeur de retour ne seront plus contrôlées non plus

Le développeur est totalement libre et peut choisir lui-même l'API qui servira à mettre en place une vérification furtive, il choisit aussi les conséquences de la détection d'une tentative de piratage ainsi que la valeur de retour que doit renvoyé le trigger.

Application (mal) déprotégée :

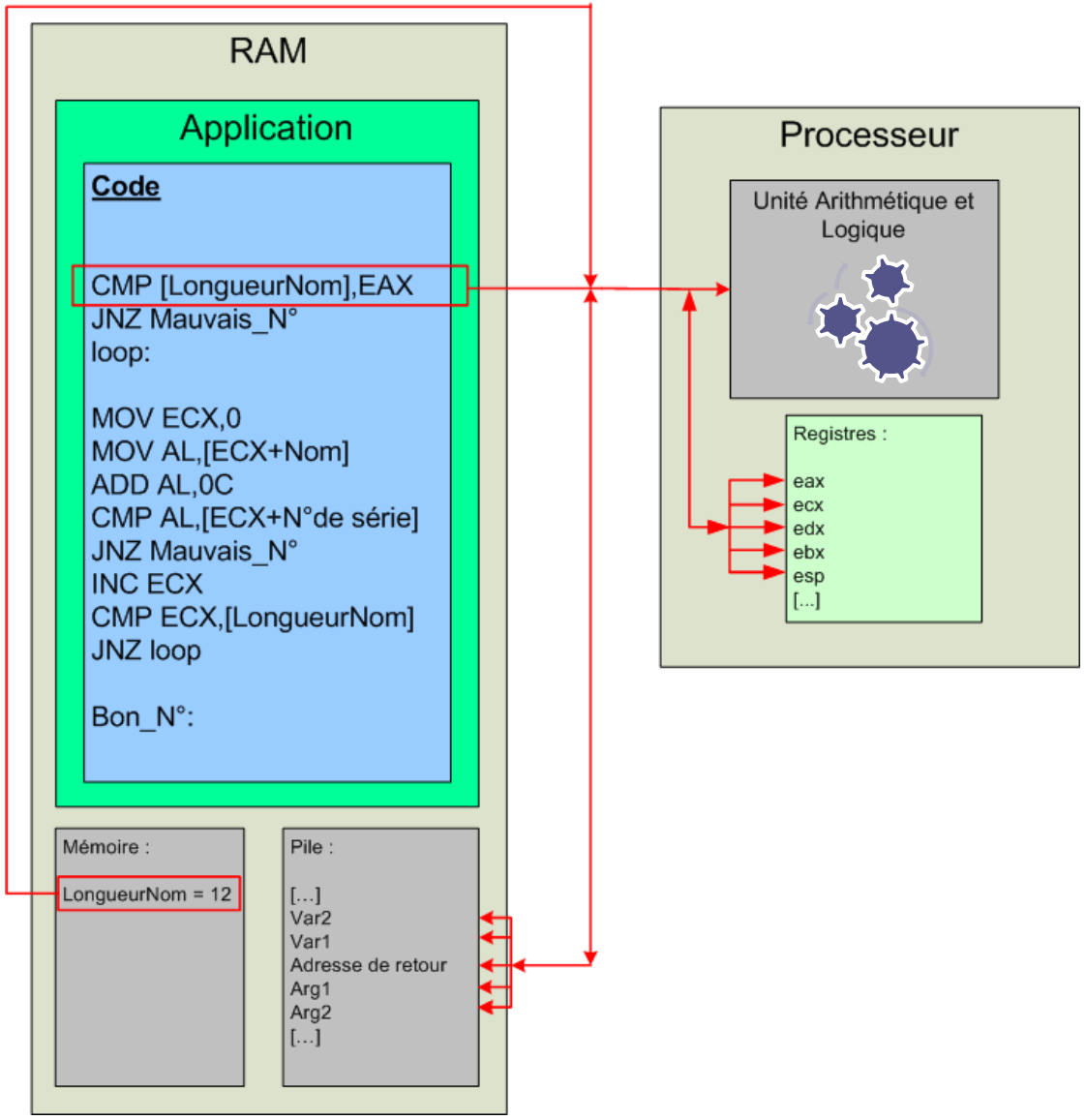


Exemple de bug graphique engendré suite à la détection de la dé protection d'un jeux :

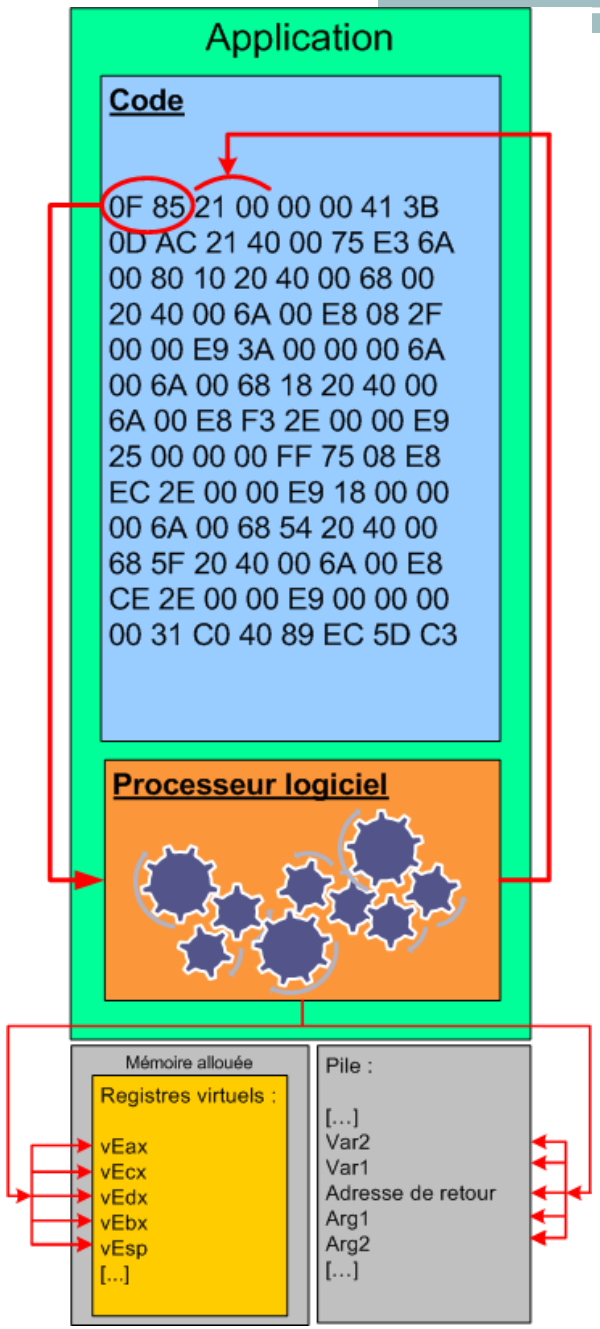
VMProtect© : virtualisation de code

- VMProtect va virtualiser le code de l'application c'est-à-dire créer un processeur virtuel logiciel qui interprétera les instructions qui auront été transformé au depart
- Le code ainsi transformer ne pourra pas être désassemblé par les désassembleur x86 et ne pourra pas être débogué directement, en effet la syntaxe des instructions interprétées par le processeur logiciel est différente de la syntaxe des processeurs matériels.
- La virtualisation est utilisée dans toutes les protections récentes de qualité
- Elle bloque le cracker qui devra étudier le processeur logiciel et créer un outils qui fera la relation instruction du processeur logiciel -> instruction x86
- Associé à une bonne obfuscation, la virtualisation bloquera la plupart des crackers

Fonctionnement normal d'un programme



Programme virtualisé



Conclusion

- Récapitulatif
- Le métier de Reverse-Engineer
- Questions ?

Récapitulatif

- Développer sois-mêmes une protection, nécessite beaucoup de temps et d'investissement, mais est très enrichissant
- Si vous choisissez d'utiliser un *packer* commercial :
  - N'hésitez pas à vous renseigner sur son efficacité, à chercher des outils, des techniques permettant de le contourner
  - Utilisez toutes les options qu'il vous offre
  - Utilisez au maximum son SDK
  - N'hésitez pas à rajouter d'autres vérifications par vous-mêmes, cela freinera le cracker qui cherchera d'abord des protections
- Il est nécessaire d'observer une bonne politique de sécurité
  - Eviter de distribuer des versions d'évaluation 100% fonctionnelles, même protégées
  - Garder un fichier des clients
  - Utiliser les options permettant de calculer le n° de série à partir d'une variable matérielle (adresse MAC, n° de série du disque dur ...)

Le métier de Reverse Engineer

Il consiste à étudier des exécutables de toutes sortes, des protection logicielles, des malwares, des formats etc.

Le reverse engineer développe des outils, des protections et fait de la recherche.

Les salaires peuvent varier entre 50 000 euros et 100 000 euros brut par an. Mais une personne faisant seulement du reverse aura en général un salaire moins élevé qu'une personne faisant du reverse , ainsi que de la recherche et du développement.

En France, les postes de reverse engineer sont rares et souvent moins bien payé que dans les autres pays.

Questions ?