Virologie
2 - Techniques de rétroconception

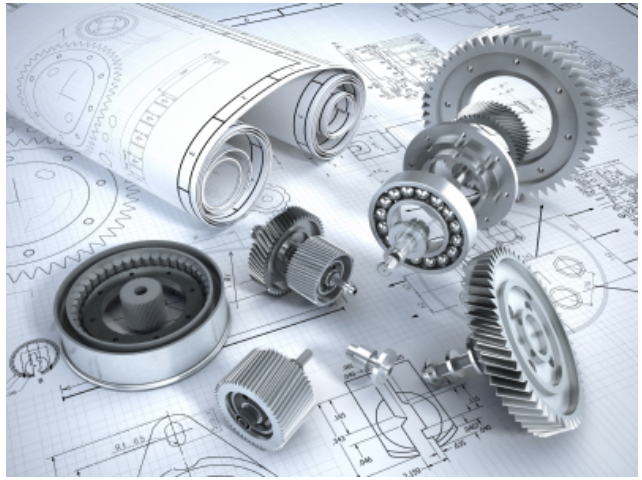
Jérémy Rubert

rubert.jeremy@gmail.com

22 & 24 octobre 2025

Plan

Introduction sur la rétroconception



Plan

Définitions

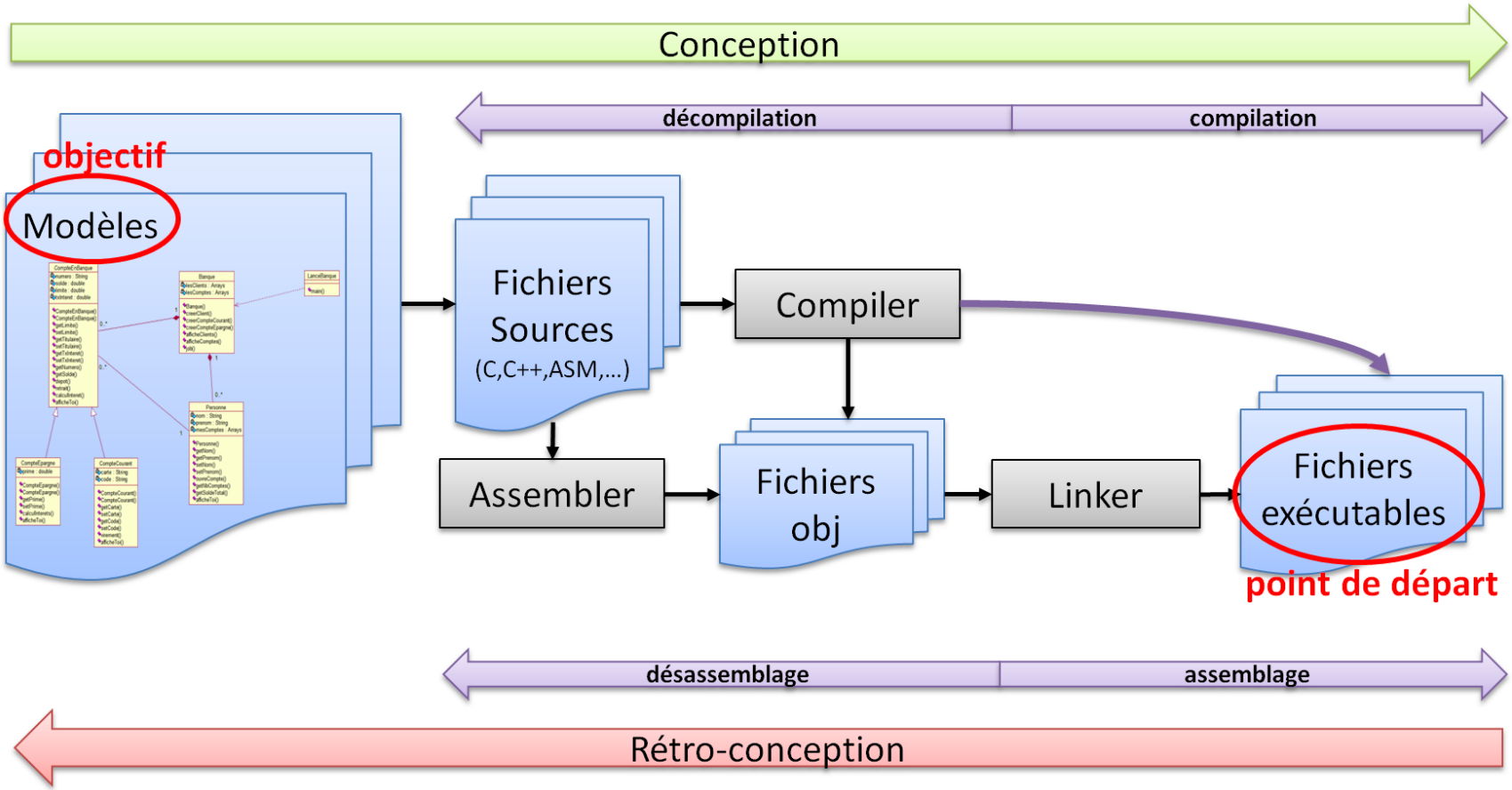
Rétroconception

Dans son acceptation large, la rétroconception consiste à étudier un objet afin d'en déterminer son fonctionnement interne ou la façon dont il a été conçu.

Pour un logiciel, il s'agit d'en comprendre son fonctionnement interne pour en extraire une **abstraction** de plus haut niveau.

- Processus inverse de celui de la conception ;
- Travail sur des binaires.

Objectif


Conception
décompilation
compilation
objectif
Modèles
Fichiers Sources
(C,C++,ASM,...)
Compiler
Assembler
Fichiers obj
Linker
Fichiers exécutables
point de départ
désassemblage
assemblage
Rétro-conception

Illustration

Attention

La rétroconception ne se limite pas au désassemblage, ni à la décompilation.

Désassemblage...

```
  push ebx
  push esi
  mov  eax, ecx
  mov  esi, 2
  xor  ebx, ebx
  mov  ecx, 1
Loop:
  xor  edx, edx
  div  esi
  test edx, edx
  jz   Next
  or   ebx, ecx
Next:
  shl  ecx, 1
  jnb  Loop
  mov  eax, ebx
  pop  esi
  pop  ebx
  retn
```

Illustration

Attention

La rétroconception ne se limite pas au désassemblage, ni à la décompilation.

Désassemblage...

```
  push ebx
  push esi
  mov  eax, ecx
  mov  esi, 2
  xor  ebx, ebx
  mov  ecx, 1
Loop:
  xor  edx, edx
  div  esi
  test edx, edx
  jz   Next
  or   ebx, ecx
Next:
  shl  ecx, 1
  jnb  Loop
  mov  eax, ebx
  pop  esi
  pop  ebx
  retn
```

Décompilation...

```
int __fastcall proc(int a1)
{
  v1 = a1;
  v2 = 0;
  v3 = 1;
  do {
    v4 = v1;
    v1 /= 2;
    if ( v4 % 2) v2 |= v3;
    v5 = __CFSHL__(v3, 1);
    v3 *= 2;
  } while ( v5 );
  return v2;
}
```

Illustration

Attention

La rétroconception ne se limite pas au désassemblage, ni à la décompilation.

Désassemblage...

```
  push ebx
  push esi
  mov  eax, ecx
  mov  esi, 2
  xor  ebx, ebx
  mov  ecx, 1
Loop:
  xor  edx, edx
  div  esi
  test edx, edx
  jz   Next
  or   ebx, ecx
Next:
  shl  ecx, 1
  jnb  Loop
  mov  eax, ebx
  pop  esi
  pop  ebx
  retn
```

Décompilation...

```
int __fastcall proc(int a1)
{
  v1 = a1;
  v2 = 0;
  v3 = 1;
  do {
    v4 = v1;
    v1 /= 2;
    if ( v4 % 2) v2 |= v3;
    v5 = __CFSHL__(v3, 1);
    v3 *= 2;
  } while ( v5 );
  return v2;
}
```

Rétroconception...

```
int __fastcall proc(int a1)
{
  return a1;
}
```

Approches d'analyse

Il existe deux contextes d'analyse de logiciels, le contexte :

- «**boîte blanche**», lorsqu'on peut observer le fonctionnement interne du programme d'intérêt (que l'on dispose des sources ou du binaire) ;
- «**boîte noire**», lorsqu'on accède uniquement aux entrées et sorties du programme. Ce dernier se comporte alors comme un oracle.

Contexte de rétroconception

Dans le cadre de ce cours, nous nous plaçons dans un contexte de ***«boîte blanche» sur le binaire***

Applications

Le domaine d'applications de la rétroconception est vaste :

- Validation du code généré par une chaîne de compilation
- Cracking de logiciel : suppression de protection
- Vol de propriété intellectuelle : récupération d'algorithme
- Interopérabilité avec un logiciel propriétaire (principalement pratiqué par l'univers Open Source)
- Analyse de malware
- Recherche de vulnérabilités

Applications : code généré

Exemple

Quel est le code assembleur x86 32-bits produit par un compilateur C pour la fonction suivante ?

```
__int64 __cdecl mod_ll(__int64 Value)
{
  return Value % 2;
}
```

```
push ebp
mov  ebp, esp
push ebx
sub  esp, 0Ch
mov  ecx, dword ptr [ebp+arg_0]
mov  ebx, dword ptr [ebp+arg_0+4]
mov  eax, ecx
mov  edx, ebx
mov  eax, edx
mov  edx, eax
sar  edx, 1Fh
sar  eax, 1Fh
mov  eax, edx
mov  edx, 0
shr  eax, 1Fh
add  eax, ecx
adc  edx, ebx
shrd eax, edx, 1
sar  edx, 1
mov  [ebp+var_10], eax
mov  [ebp+var_C], edx
mov  eax, [ebp+var_10]
mov  edx, [ebp+var_C]
shld edx, eax, 1
add  eax, eax
sub  ecx, eax
sbb  ebx, edx
mov  [ebp+var_10], ecx
mov  [ebp+var_C], ebx
mov  eax, [ebp+var_10]
mov  edx, [ebp+var_C]
add  esp, 0Ch
pop  ebx
pop  ebp
retn
```

Cadre légal

La règle d'or

La rétroconception logicielle est **illégale** exceptée dans le cadre très contrôlé de l'interopérabilité.

Problématiques

1. Compilation → perte d'information
   - Noms de symboles
   - Distinction code/données
   - Typage
   - Structure de contrôles de « haut niveau » (for, while, do, switch, etc.)
2. Assemblage → aucune structuration imposée
   - Pas forcément de fonctions (cadre de pile, call / ret, etc.)
   - Conventions d'appels propres
   - Code auto-modifiant
   - Modification de l'adresse de retour par l'appelé
   - Utilisation des registres propres (ESP, EBP, etc).
   - ...

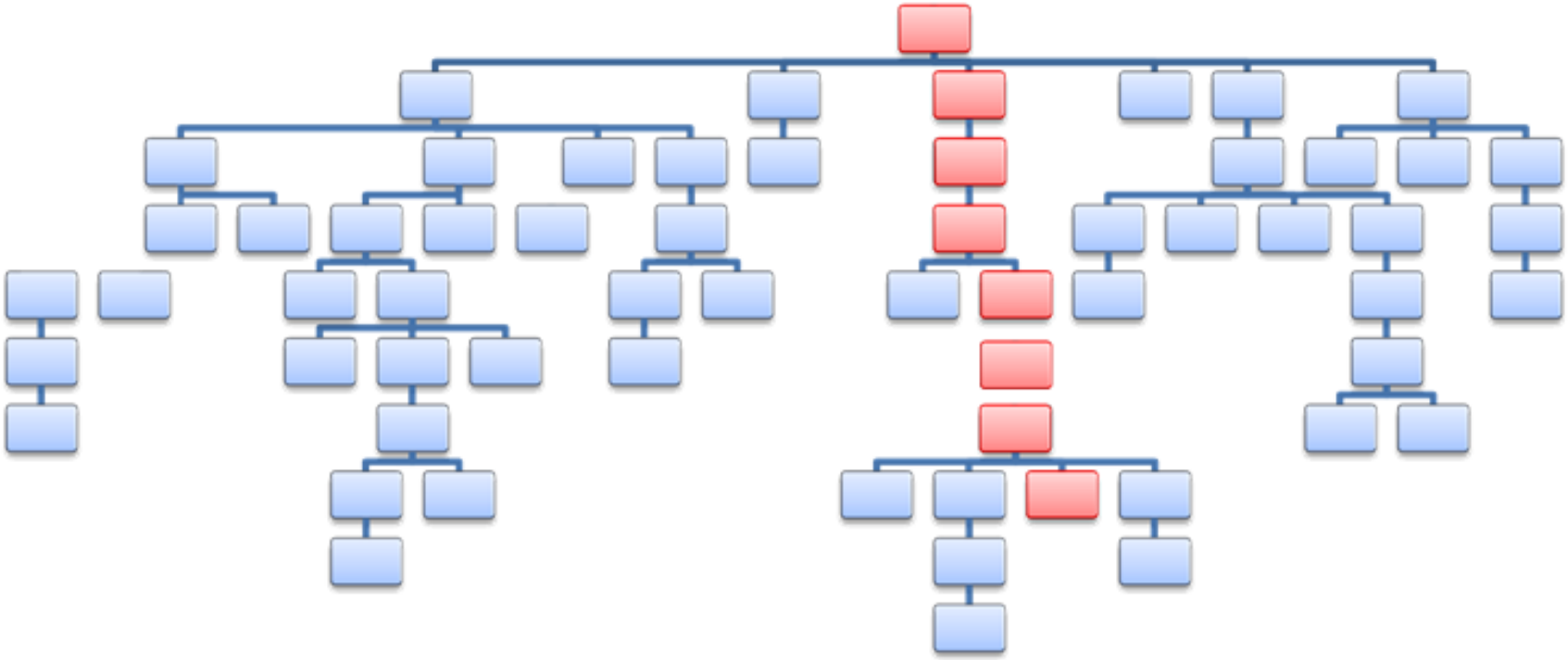
Attention

Il faut retrouver (en partie) l'information qui a disparu !

Approches

Rétro-conception boîte blanche sur du binaire :

- **Analyse statique**
  - Extraction d'information sur un programme sans recourir à son exécution → **analyse multi-chemins**
- **Analyse dynamique**
  - Suivie de l'exécution d'un programme → **analyse mono-chemin**



Plan

Différents types de langages

Il existe différent type de langages :

- Compilé
  - C/C++
  - Ocaml
- Interprété (script)
  - Python
  - VBS
  - PHP
- Emulé / virtualisé
  - Java
  - .Net

Différentes approches de rétroconception

Pour chaque langage les techniques et concepts de rétro-conceptions sont différents :

- Compilé
  - Analyse du code ASM après transformation par le compilateur
  - Maitrise de concepts liés au langage (vftable en C++)
  - etc.
- Interprété (script)
  - Lecture de code possible
  - Code souvent obfusqué dans le cadre de malware
- Emulé / virtualisé
  - Existence d'outils facilitant le passage du bytecode vers une représentation du code original
    - Moins de perte d'information entre le code source et le bytecode que dans les langages compilés

Différentes approches de rétroconception

Limite de ce cours

Apprentissage de la rétroconception de binaires compilés en C

Difficultés supplémentaires

Certains langages compilés, par leur fonctionnement, sont encore plus difficiles à analyser.
Exemple :

- Go
- Rust

Un hello world en Go c'est presque 1 800 fonctions (contre 15 en C) !

Plan

Analyse statique

Définition

Analyse d'un programme sans l'exécuter

Plusieurs étapes à réaliser :

1. Désassemblage
2. Propagation de données
3. Propagation de types
4. Détection de bibliothèques par signatures
5. Décompilation

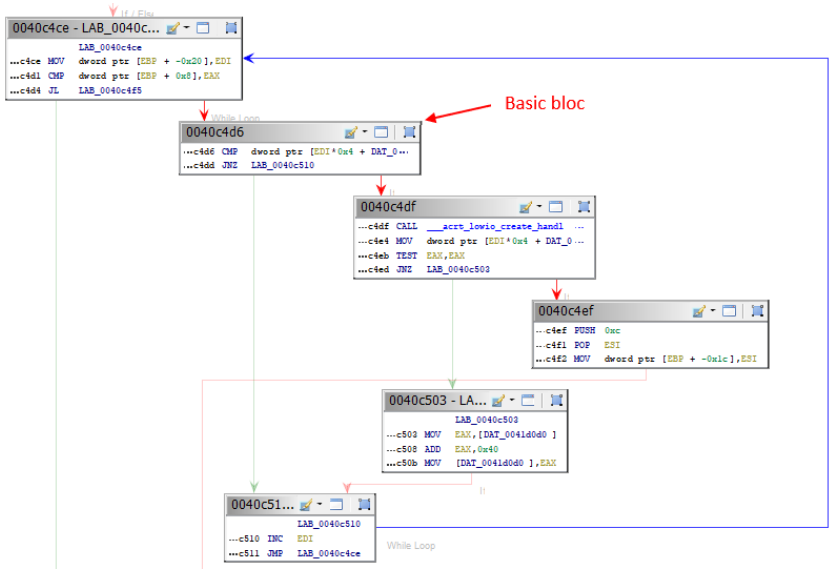
Attention

Les outils d'analyse effectuent une partie de chaque étape mais restent limités !

Flot de Contrôle (Control Flow)

Définition

Suivi des instructions exécutées en fonction des sauts et des appels de fonction

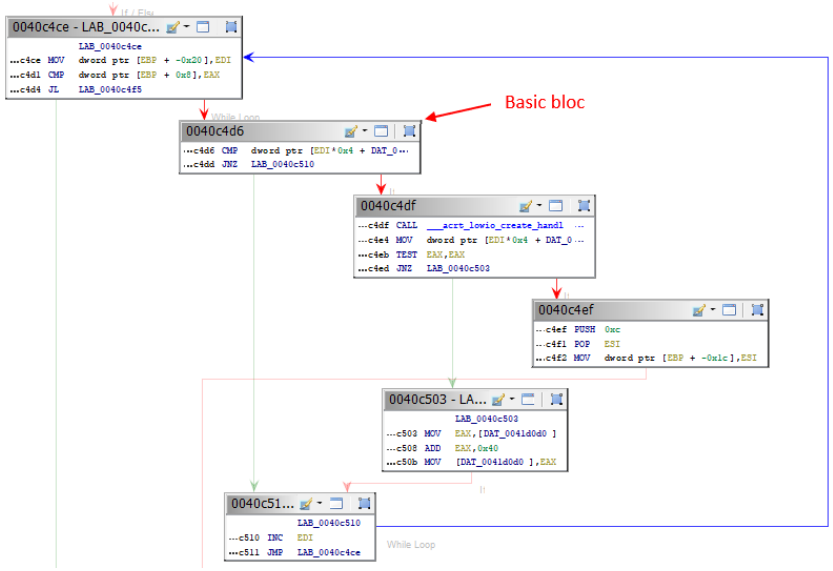


Vue graph

Flot de Contrôle (Control Flow)

Définition

Suivi des instructions exécutées en fonction des sauts et des appels de fonction



Vue graph

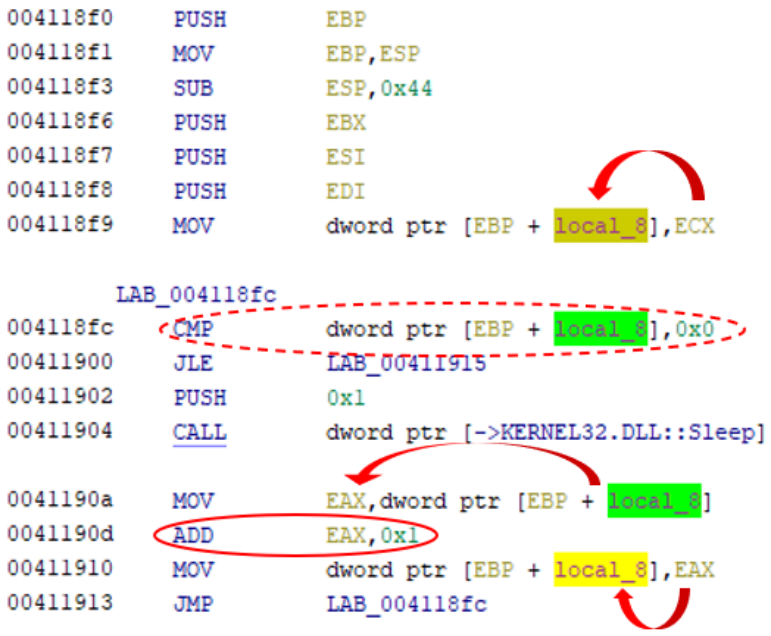
```
                        LAB_0040c4ce                                    XREF[1]:
0040c4ce 89 7d e0       MOV     dword ptr [EBP + -0x20],EDI
0040c4d1 39 45 08       CMP     dword ptr [EBP + 0x8],EAX
0040c4d4 7c 1f          JL      LAB_0040c4f5
0040c4d6 39 34 bd       CMP     dword ptr [EDI*0x4 + DAT_0041ced0],ESI
         d0 ce 41 00
0040c4dd 75 31          JNZ     LAB_0040c510
0040c4df e8 f5 fe       CALL    ___acrt_lowio_create_handle_array
         ff ff
0040c4e4 89 04 bd       MOV     dword ptr [EDI*0x4 + DAT_0041ced0],EAX
         d0 ce 41 00
0040c4eb 85 c0          TEST    EAX,EAX
0040c4ed 75 14          JNZ     LAB_0040c503
0040c4ef 6a 0c          PUSH    0xc
0040c4f1 5e             POP     ESI
0040c4f2 89 75 e4       MOV     dword ptr [EBP + -0x1c],ESI

                        LAB_0040c4f5                                    XREF[1]:
0040c4f5 c7 45 fc       MOV     dword ptr [EBP + -0x4],0xfffffffe
         fe ff ff ff
0040c4fc e8 15 00       CALL    FUN_0040c516
         00 00
0040c501 eb ac          JMP     LAB_0040c4af

                        LAB_0040c503                                    XREF[1]:
0040c503 a1 d0 d0       MOV     EAX,[DAT_0041d0d0]
         41 00
0040c508 83 c0 40       ADD     EAX,0x40
0040c50b a3 d0 d0       MOV     [DAT_0041d0d0],EAX
         41 00

                        LAB_0040c510                                    XREF[1]:
0040c510 47             INC     EDI
0040c511 eb bb          JMP     LAB_0040c4ce
```

Vue linéaire

Flot de données

Définition

Suivi des données à travers les registres et la mémoire



1 - Désassemblage : problématique

- **Linéaire** : séquentiel indépendamment du flot de contrôle
- **Récursif** : séquentiel jusqu'à une instruction de transfert
  - CALL/Jcc → appel récursif
  - JMP → continue à partir de l'adresse ciblée

1 - Désassemblage : Linéaire

Avantages

Capable de trouver du code indépendamment du flot de contrôle

Inconvénients

Sur-approximation du code en désassemblant de la donnée

Utilisé principalement par les débuggeurs (exemple WinDBG)

1 - Désassemblage : Récursif

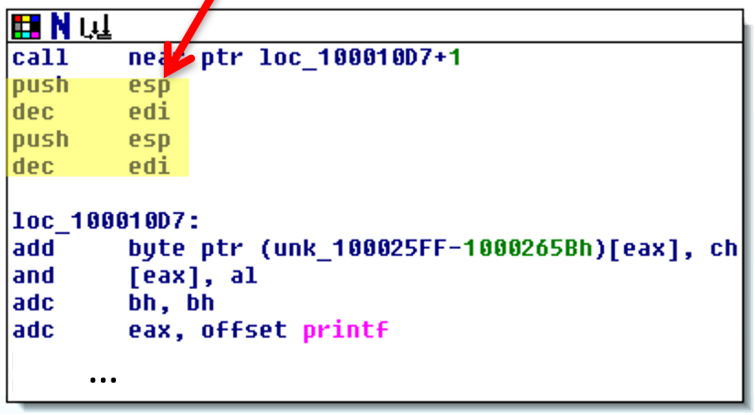
Avantages

Limite l'interprétation de données en tant que code

Inconvénients

Nécessite de connaître la destination exacte d'un branchement ainsi que son retour

Présupposition sur le retour du call



Code désassemblé

1 - Désassemblage : Récursif

Avantages

Limite l'interprétation de données en tant que code

Inconvénients

Nécessite de connaître la destination exacte d'un branchement ainsi que son retour

Présupposition sur le retour du call

```
call    near ptr loc_100010D7+1
push    esp
dec     edi
push    esp
dec     edi

loc_100010D7:
add     byte ptr (unk_100025FF-1000265Bh)[eax], ch
and     [eax], al
adc     bh, bh
adc     eax, offset printf
    ...
```

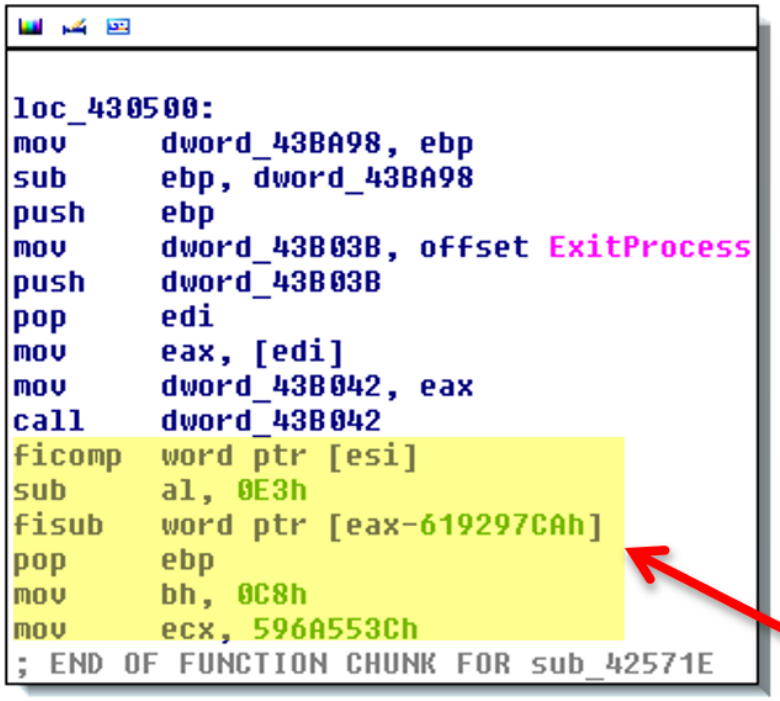
Code désassemblé

```
                call    loc_100010D8
; ---------------------------------------
aToto           db 'TOTO',0
; ---------------------------------------

loc_100010D8:
                push    offset aS
                call    ds:printf
                        ...
```

Code réel

2 - Propagation de données

Propagation des registres et des variables tant qu'ils ne sont pas redéfinis



Désassemblage incorrect

2 - Propagation de données

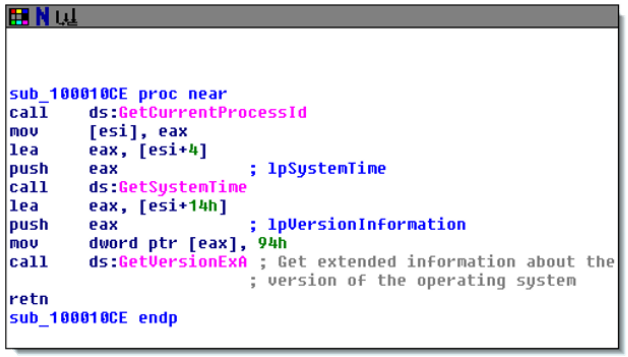
Propagation des registres et des variables tant qu'ils ne sont pas redéfinis

```
loc_430500:
mov     dword_43BA98, ebp
sub     ebp, dword_43BA98
push    ebp
mov     dword_43B03B, offset ExitProcess
push    dword_43B03B
pop     edi
mov     eax, [edi]
mov     dword_43B042, eax
call    dword_43B042
ficomp  word ptr [esi]
sub     al, 0E3h
fisub   word ptr [eax-619297CAh]
pop     ebp
mov     bh, 0C8h
mov     ecx, 596A553Ch
; END OF FUNCTION CHUNK FOR sub_42571E
```

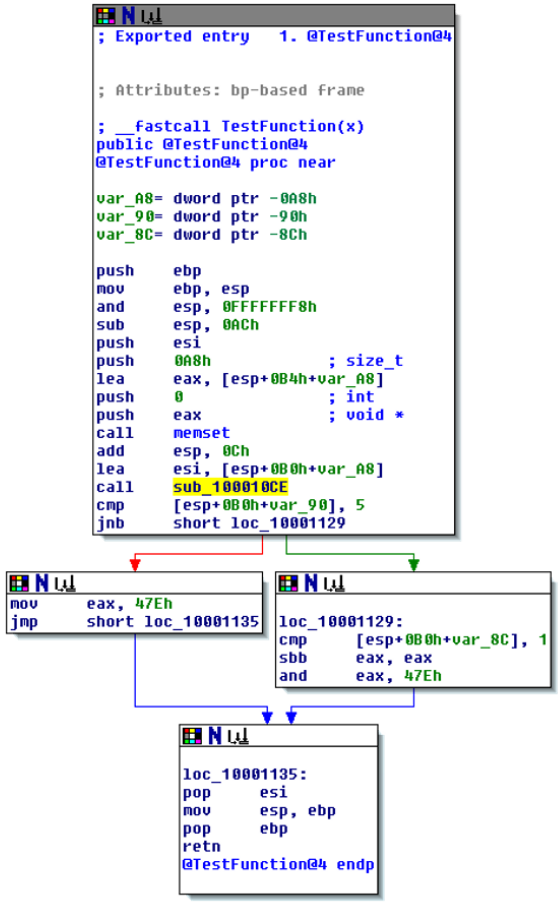
Désassemblage incorrect

```
loc_430500:
mov     dword_43BA98, ebp
sub     ebp, dword_43BA98
push    ebp
mov     dword_43B03B, offset ExitProcess
push    dword_43B03B
pop     edi
mov     eax, [edi]
mov     dword_43B042, eax
call    dword_43B042
```

3 - Propagation des types



```
sub_100010CE proc near
call    ds:GetCurrentProcessId
mov     [esi], eax
lea     eax, [esi+4]
push    eax             ; lpSystemTime
call    ds:GetSystemTime
lea     eax, [esi+14h]
push    eax             ; lpVersionInformation
mov     dword ptr [eax], 94h
call    ds:GetVersionExA ; Get extended information about the
                         ; version of the operating system
retn
sub_100010CE endp
```



```
; Exported entry   1. @TestFunction@4

; Attributes: bp-based frame

; __fastcall TestFunction(x)
public @TestFunction@4
@TestFunction@4 proc near

var_A8= dword ptr -0A8h
var_90= dword ptr -90h
var_8C= dword ptr -8Ch

push    ebp
mov     ebp, esp
and     esp, 0FFFFFFF8h
sub     esp, 0ACh
push    esi
push    0A8h            ; size_t
lea     eax, [esp+0B4h+var_A8]
push    0               ; int
push    eax             ; void *
call    memset
add     esp, 0Ch
lea     esi, [esp+0B0h+var_A8]
call    sub_100010CE
cmp     [esp+0B0h+var_90], 5
jnb     short loc_10001129
```

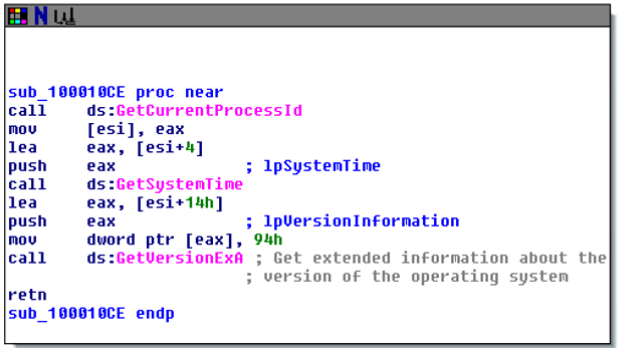
```
mov     eax, 47Eh
jmp     short loc_10001135
```

```
loc_10001129:
cmp     [esp+0B0h+var_8C], 1
sbb     eax, eax
and     eax, 47Eh
```

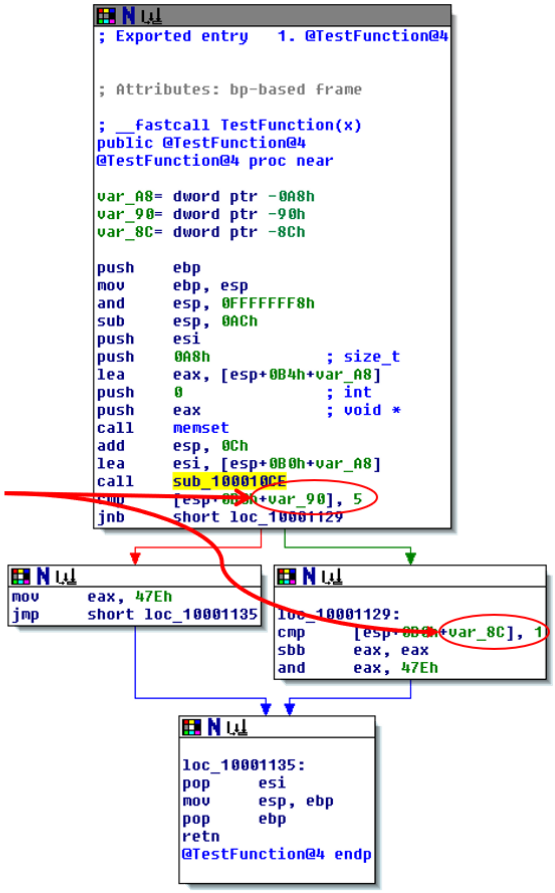
```
loc_10001135:
pop     esi
mov     esp, ebp
pop     ebp
retn
@TestFunction@4 endp
```

3 - Propagation des types

Le retour de fonction dépend de **var_90** et **Var_8C** qui ne sont pas **directement** définis !?



```
sub_100010CE proc near
call    ds:GetCurrentProcessId
mov     [esi], eax
lea     eax, [esi+4]
push    eax             ; lpSystemTime
call    ds:GetSystemTime
lea     eax, [esi+14h]
push    eax             ; lpVersionInformation
mov     dword ptr [eax], 94h
call    ds:GetVersionExA ; Get extended information about the
                         ; version of the operating system
retn
sub_100010CE endp
```



```
; Exported entry   1. @TestFunction@4


; Attributes: bp-based frame

; __fastcall TestFunction(x)
public @TestFunction@4
@TestFunction@4 proc near

var_A8= dword ptr -0A8h
var_90= dword ptr -90h
var_8C= dword ptr -8Ch

push    ebp
mov     ebp, esp
and     esp, 0FFFFFFF8h
sub     esp, 0ACh
push    esi
push    0A8h            ; size_t
lea     eax, [esp+0B4h+var_A8]
push    0               ; int
push    eax             ; void *
call    memset
add     esp, 0Ch
lea     esi, [esp+0B0h+var_A8]
call    sub_100010CE
cmp     [esp+0B0h+var_90], 5
jnb     short loc_10001129
```

```
mov     eax, 47Eh
jmp     short loc_10001135
```

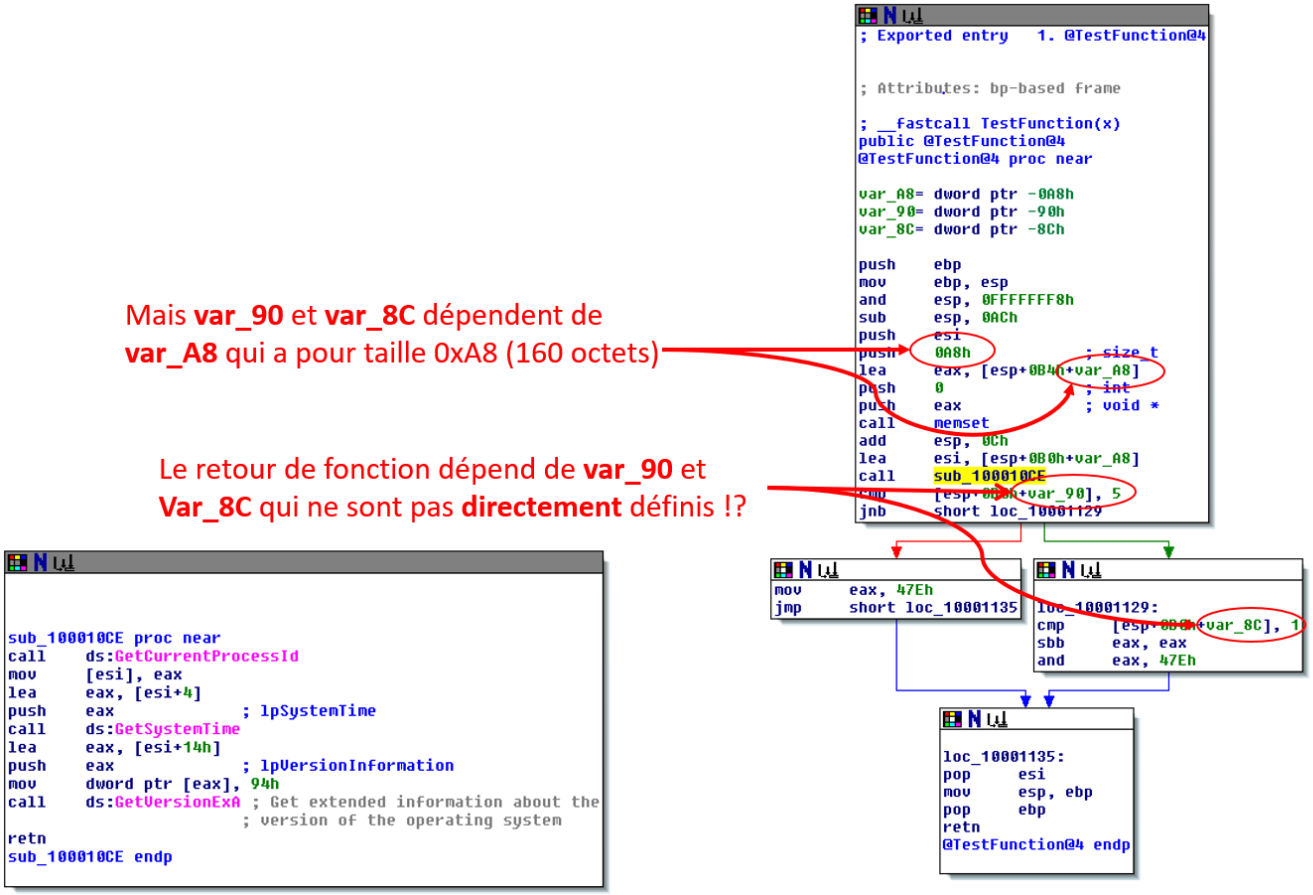
```
loc_10001129:
cmp     [esp+0B0h+var_8C], 1
sbb     eax, eax
and     eax, 47Eh
```

```
loc_10001135:
pop     esi
mov     esp, ebp
pop     ebp
retn
@TestFunction@4 endp
```

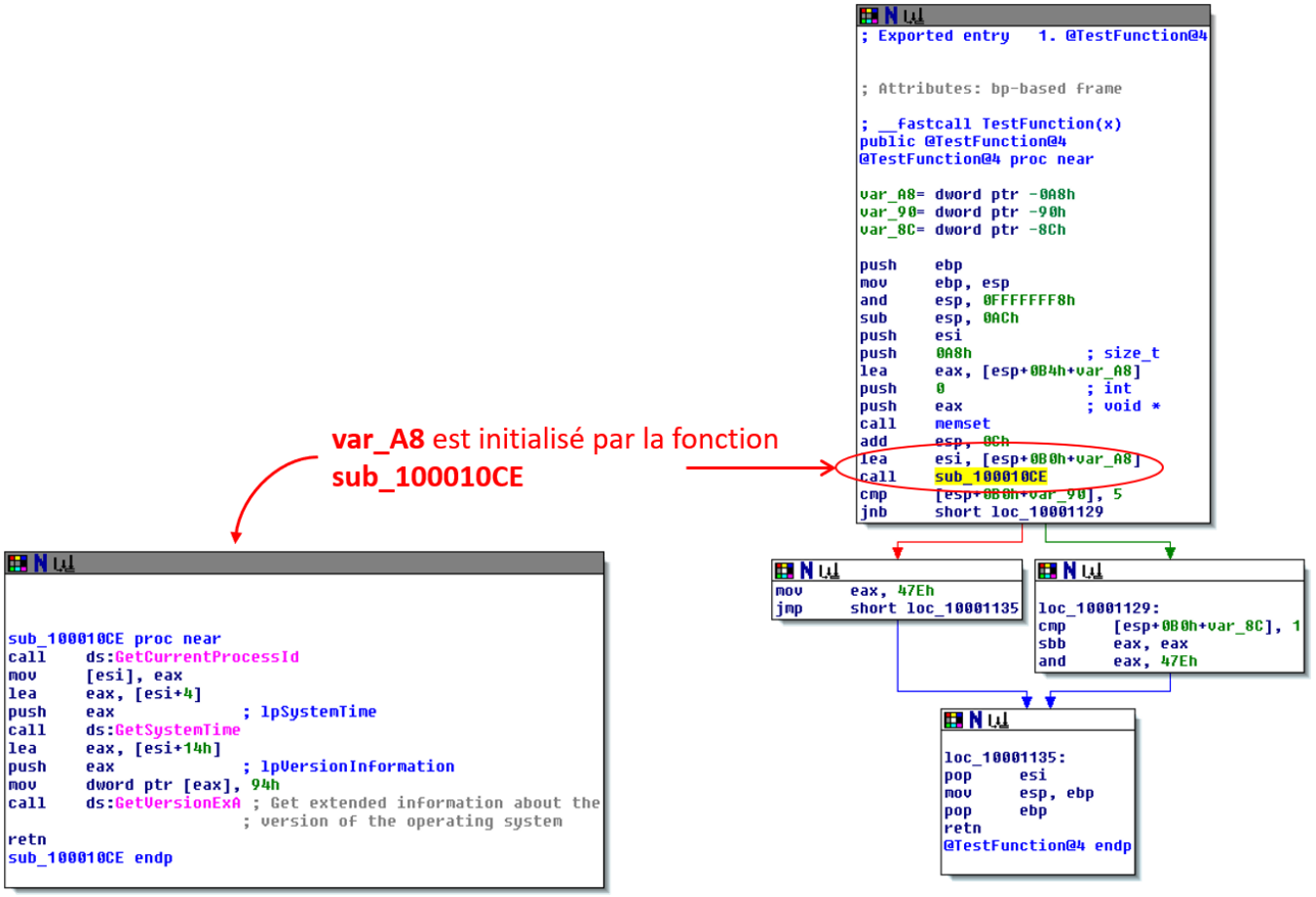
3 - Propagation des types

Mais **var_90** et **var_8C** dépendent de **var_A8** qui a pour taille 0xA8 (160 octets)

Le retour de fonction dépend de **var_90** et **Var_8C** qui ne sont pas **directement** définis !?



3 - Propagation des types



3 - Propagation des types

```
00000000 ; ---------------------------------------
00000000
00000000 Struct1_t       struc ; (sizeof=0xA8)
00000000 Pid             dd ?
00000004 SystemTime      SYSTEMTIME ?
00000014 OsVersionInfo   OSVERSIONINFO ?
000000A8 Struct1_t       ends
```

sub_100010CE permet de définir le type de **var_A8** (les **champs** et **leurs types**)

```
sub_100010CE proc near
call    ds:GetCurrentProcessId
mov     [esi], eax
lea     eax, [esi+4]
push    eax             ; lpSystemTime
call    ds:GetSystemTime
lea     eax, [esi+14h]
push    eax             ; lpVersionInformation
mov     dword ptr [eax], 94h
call    ds:GetVersionExA ; Get extended information about the
                         ; version of the operating system
retn
sub_100010CE endp
```

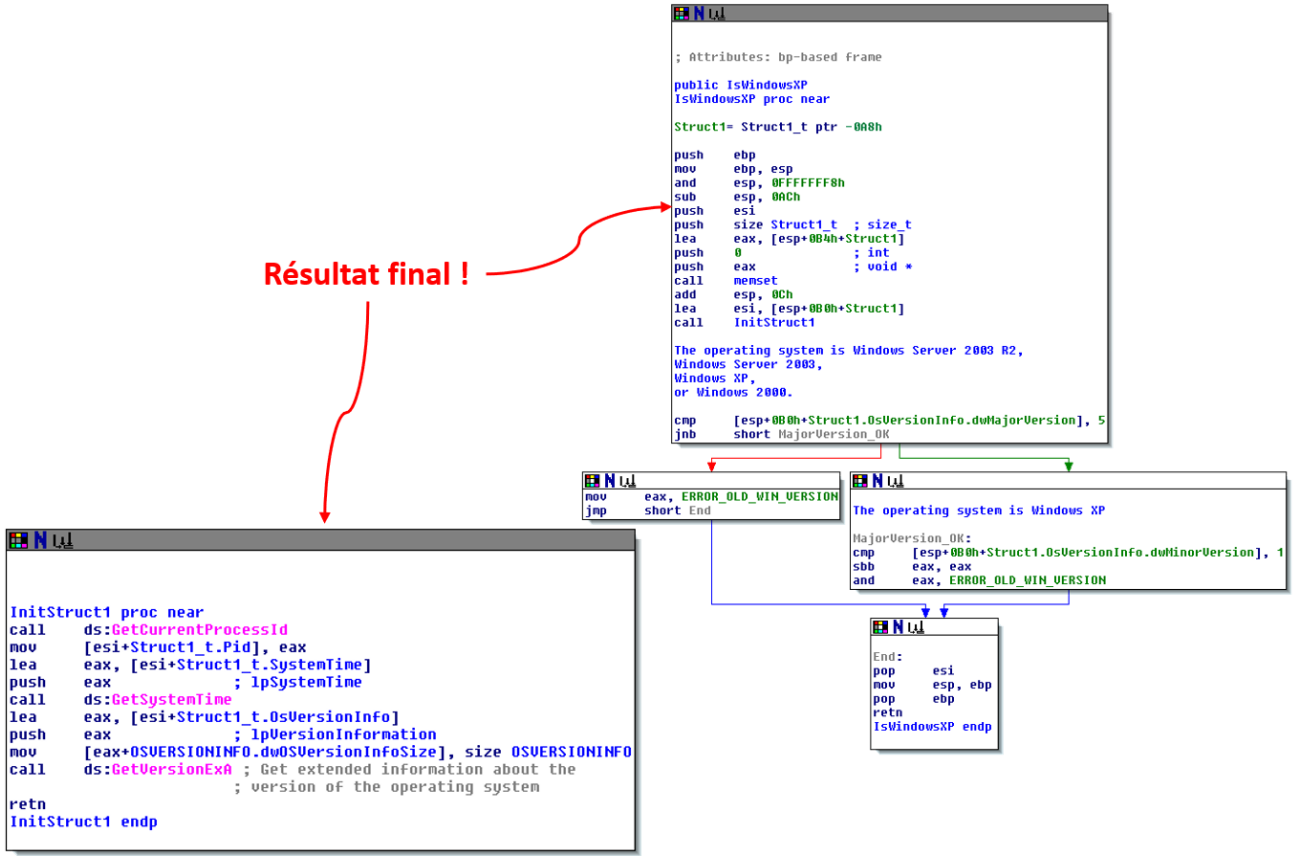
```
; Exported entry   1. @TestFunction@4

; Attributes: bp-based frame

; __fastcall TestFunction(x)
public @TestFunction@4
@TestFunction@4 proc near

var_A8= dword ptr -0A8h
var_90= dword ptr -90h
var_8C= dword ptr -8Ch

push    ebp
mov     ebp, esp
and     esp, 0FFFFFFF8h
sub     esp, 0ACh
push    esi
push    0A8h            ; size_t
lea     eax, [esp+0B4h+var_A8]
push    0               ; int
push    eax             ; void *
call    memset
add     esp, 0Ch
lea     esi, [esp+0B0h+var_A8]
call    sub_100010CE
cmp     [esp+0B0h+var_90], 5
jnb     short loc_10001129
```

```
mov     eax, 47Eh
jmp     short loc_10001135
```

```
loc_10001129:
cmp     [esp+0B0h+var_8C], 1
sbb     eax, eax
and     eax, 47Eh
```

```
loc_10001135:
pop     esi
mov     esp, ebp
pop     ebp
retn
@TestFunction@4 endp
```

3 - Propagation des types

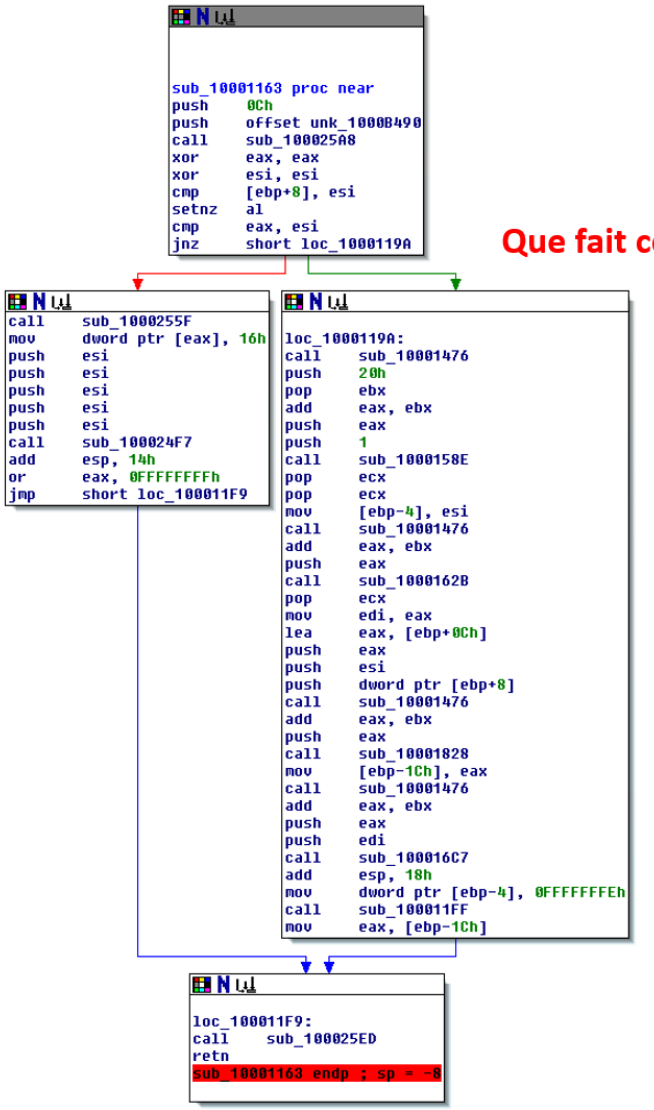


4 - Détection des signatures

- Permet de distinguer les bibliothèques (lib) liées statiquement du code « propre » à l'application
- Bibliothèques standards fournies avec tout bon désassembleur (IDA Pro, Ghidra, etc.)
- Peuvent être générées à partir de binaires pour être intégrées à un désassembleur

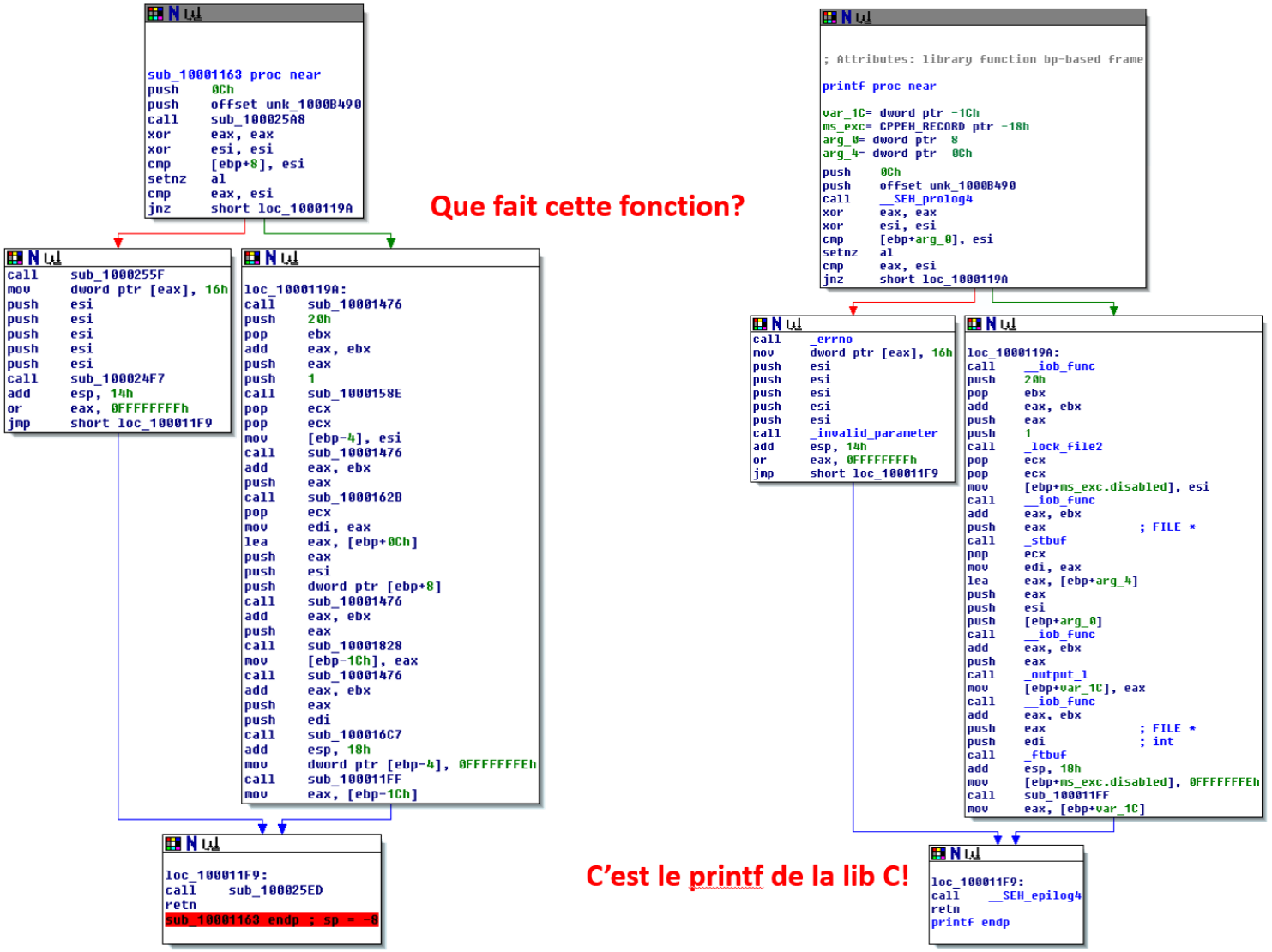
Intérêt ?

4 - Détection des signatures - Exemple



Que fait cette fonction?

4 - Détection des signatures - Exemple

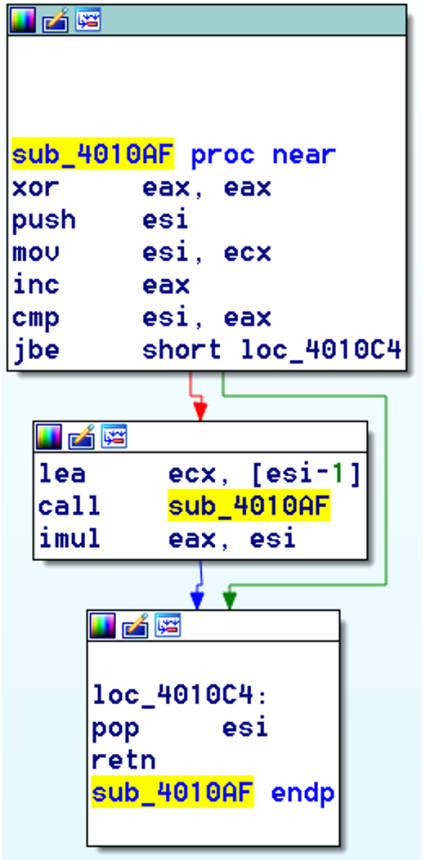


5 - Décompilation

Définition

Passer de l'assembleur à du pseudo code haut niveau

Que fait cette fonction ?

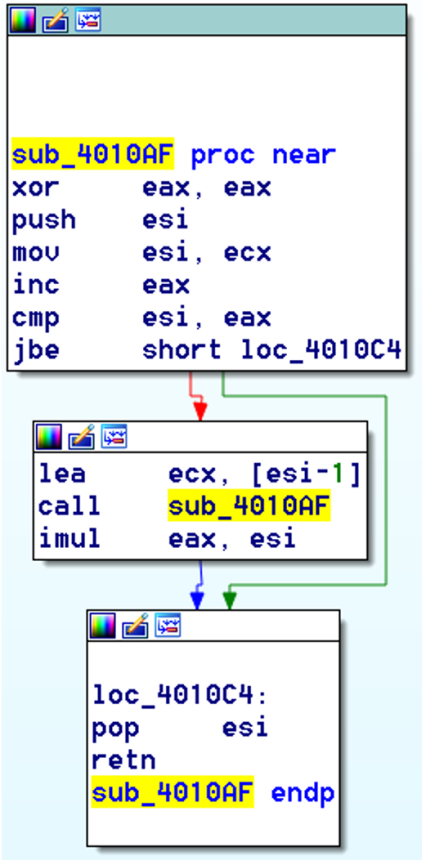


5 - Décompilation

Définition

Passer de l'assembleur à du pseudo code haut niveau

Que fait cette fonction ?




```
1 int __fastcall sub_4010AF(unsigned int a1)
2 {
3   int result; // eax@1
4
5   result = 1;
6   if ( a1 > 1 )
7     result = a1 * sub_4010AF(a1 - 1);
8   return result;
9 }
```

Analyse dynamique

Définition

Analyse du programme pendant son exécution

Permet de déterminer précisément le « comportement » d'un programme pour une entrée donnée

Différentes techniques :

- Monitoring
- **Debug**
- Tracing

Analyse dynamique : Monitoring

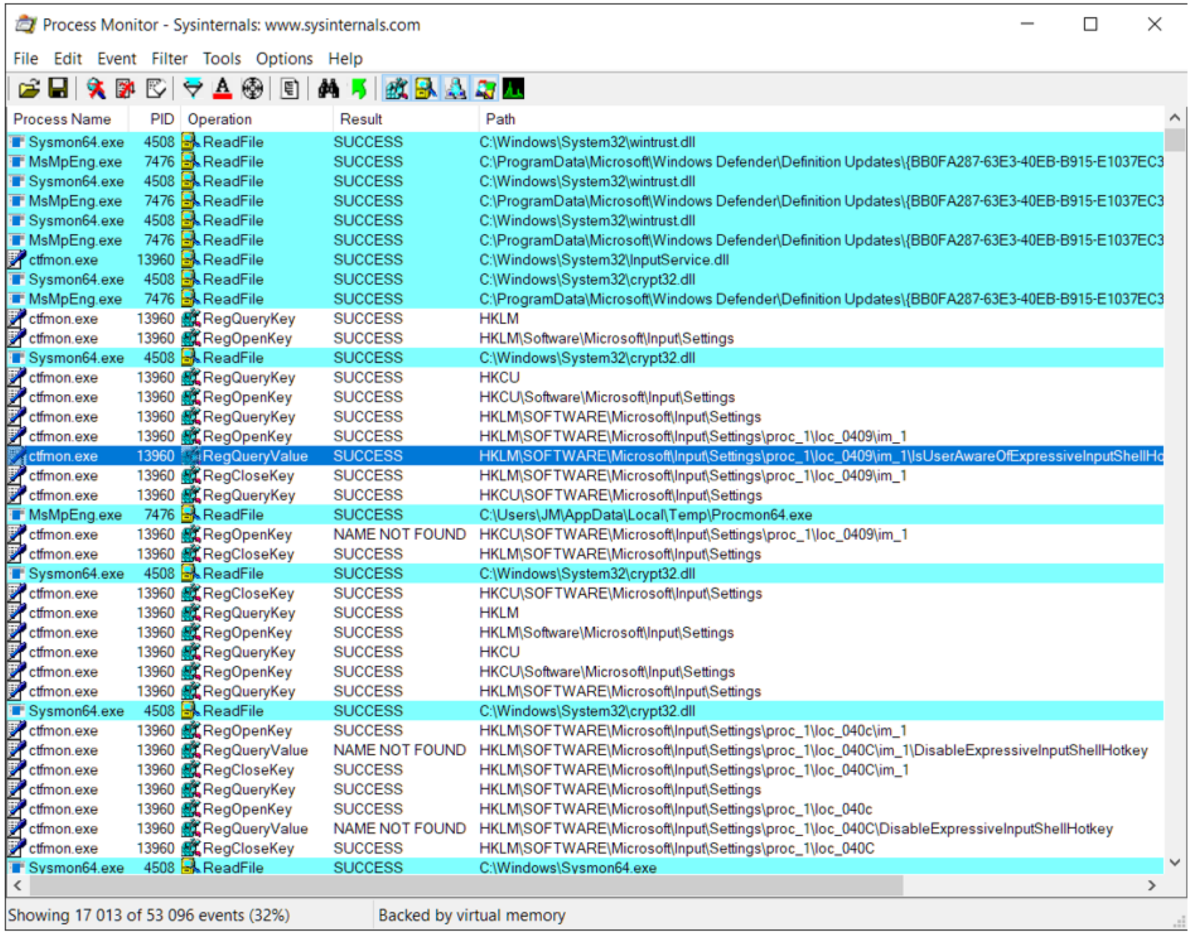
Monitoring des actions menées par un programme

- Actions de « haut niveau »
  - Lancement et terminaison de
    - Processus
    - Thread
  - Accès aux
    - Fichiers
    - Registres
  - Activité
    - Réseau
    - IPCs (InterProcess Communications)
  - etc.
- Actions de « bas niveau »
  - Appels à un API (WIN32, LibC,etc)
  - Appels système

Analyse dynamique : Monitoring

Exemple : procmon.exe (Sysinternals)

- FileSystem
- Registry
- Process
- Thread
- Network



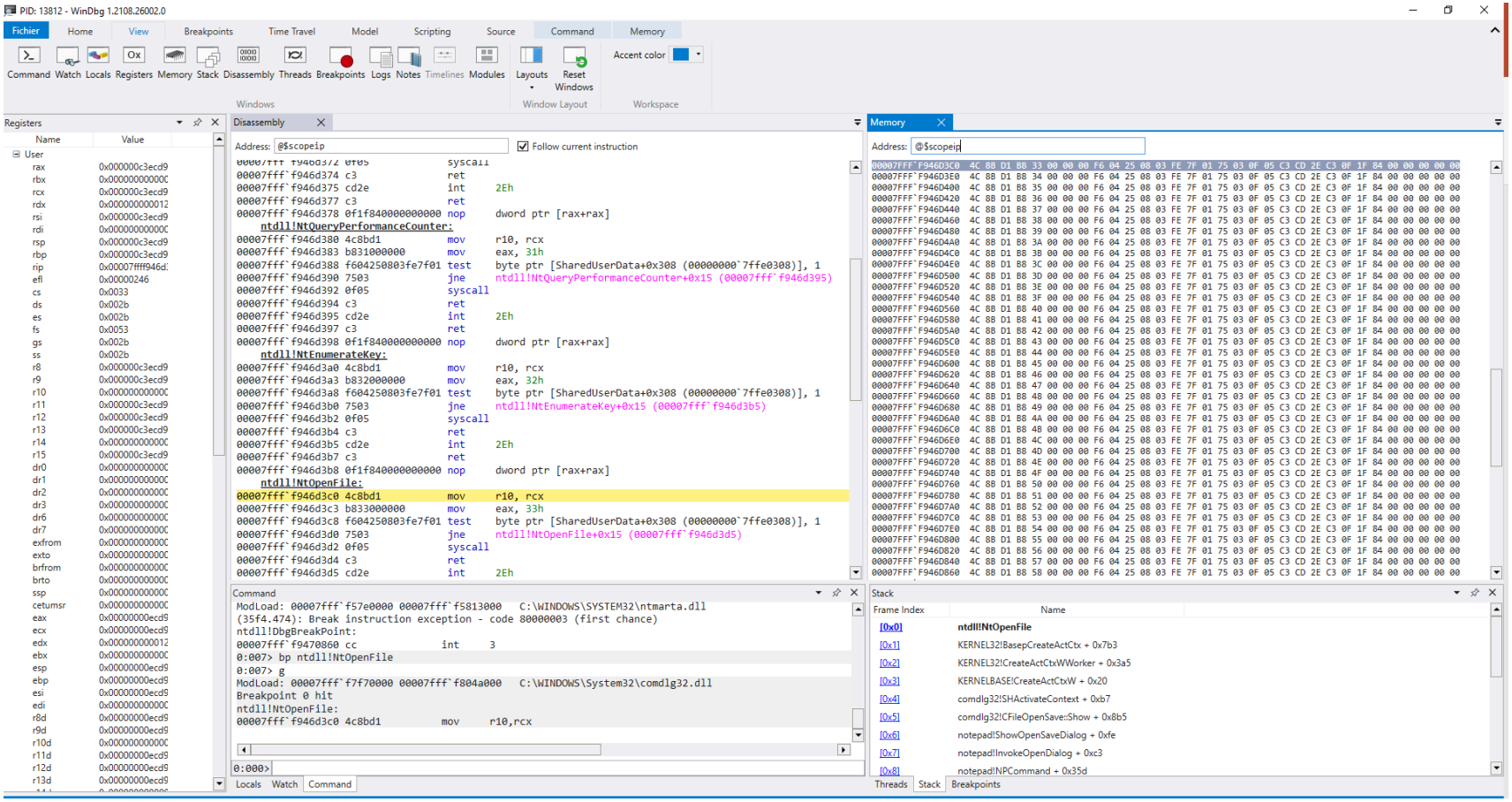
Analyse dynamique : Debug

Debug applicatif/noyau

- Permet de connaître l'état exact de la machine à un instant donné
  - Accès aux registres (RW)
  - Accès à la mémoire (RW)
  - Accès au code en cours d'exécution
  - Positionnement des points d'arrêts
    - Matériels : registres de debug (accès RWX)
    - Logiciels : patch du programme (0xCC = int 3, accès en exécution seulement)

Analyse dynamique : Debug

Exemple : WinDbg



Analyse dynamique : Tracing

Tracing

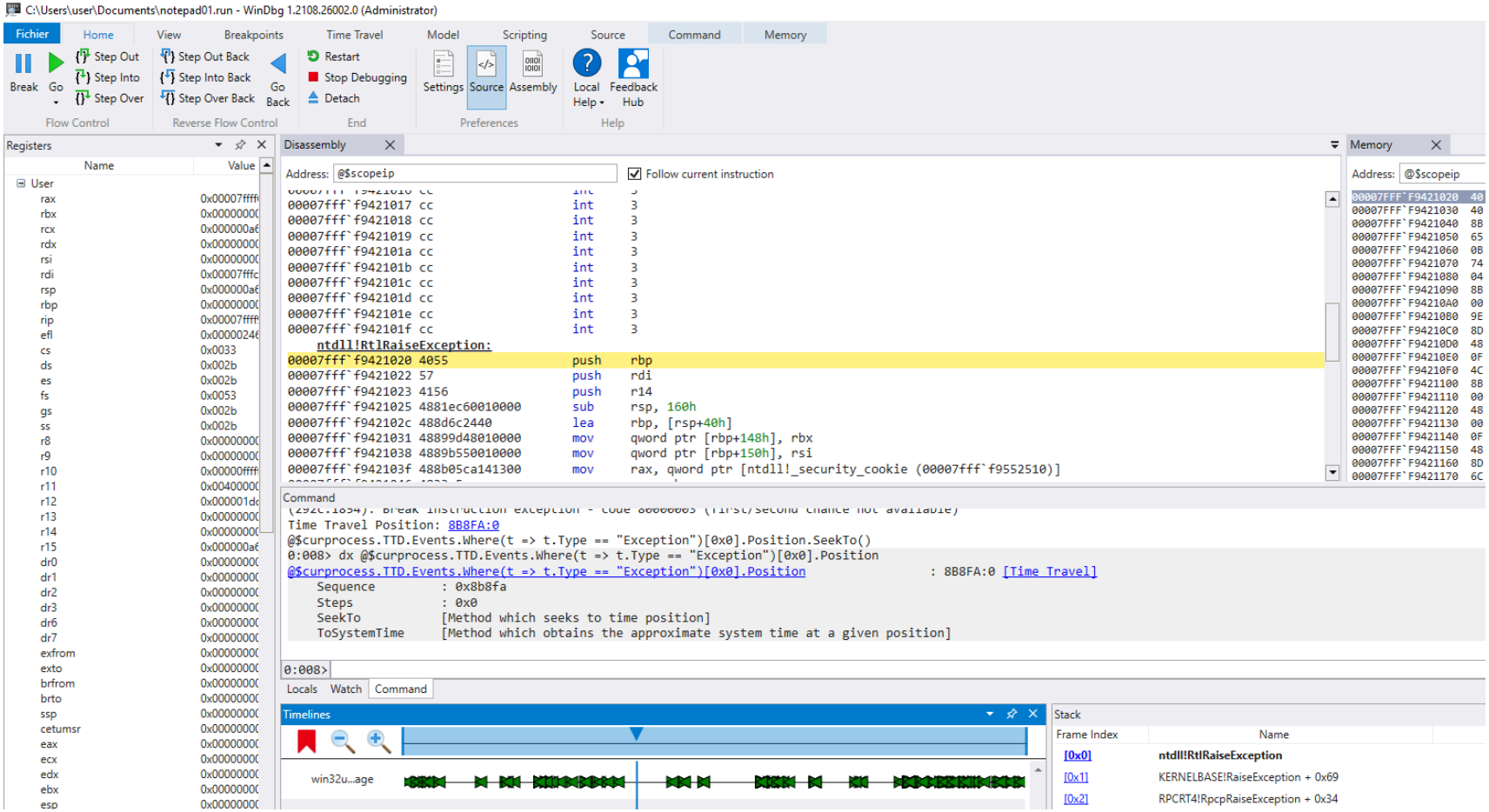
- Permet de connaître l'état exact d'un programme **à tout instant**
- Enregistrement de l'état du programme (registre/mémoire) à chaque instruction
- Navigation entre les états (en avant et en arrière)
- Possibilité de poser des breakpoints (code ou mémoire)

Inconvénients

- Long à enregistrer
- Prend beaucoup de place mémoire

Analyse dynamique : Tracing

Exemple : WinDbg Time Travel Debuging



Analyse dynamique - Utilité

- Connaître la destination d'un appel indirect

```
    sub_403D60 proc near

    var_90= byte ptr -90h
    var_3C= byte ptr -3Ch

000 sub     esp, 90h
090 push    esi
094 mov     esi, ecx
094 push    edi
098 mov     ecx, 15h
098 xor     eax, eax
098 lea     edi, [esp+98h+var_90]
098 rep stosd
098 mov     eax, [esi]
098 mov     ecx, esi
098 mov     dword ptr [esp+98h+var_90], 1132h
098 call    dword ptr [eax+44h]
098 test    eax, eax
098 jz      End
```

- « Prendre la vue » sur une action précise d'un programme (synchronisation)

```
    Recv1 proc near

    buf= dword ptr  4
    len= dword ptr  8

000 mov     eax, [esp+len]
000 mov     edx, [esp+buf]
000 push    0               ; flags
004 push    eax             ; len
008 mov     eax, [ecx+0A10h]
008 push    edx             ; buf
00C push    eax             ; s
010 call    ds:recv         ; Receive data from a socket
000 retn    8
    Recv1 endp
```

- Voir l'état instantané des registres ou de la mémoire

Analyse statique/dynamique

Utiliser correctement les deux approches

- Elles sont complémentaires
- Mais bien différentes :
  - **Analyse statique** = cartographie du programme = modèle
  - **Analyse dynamique** = instantané du programme = instance

Attention

NE PAS ABUSER DU DEBUG
Abuser plutôt de la vue statique !

Attention

Debuguer un programme dont on ne connait pas son comportement (exemple : malware)

Extraction du modèle d'intérêt

Identifier des éléments remarquables

- Constantes remarquables
  - Chaînes de caractères
  - Codes d'erreurs
  - Format de fichier (PE, ELF, etc) / nombres magiques
  - Etc.
- Structures de contrôles (boucles et idiomes) → algorithmie
- Fonctions d'une API → propagation des types
- Connaissances système
- Fonctions particulières → cryptographie
  - Symétrique (bloc/flux)
  - Asymétrique
  - Hachage

Structure de contrôle de base

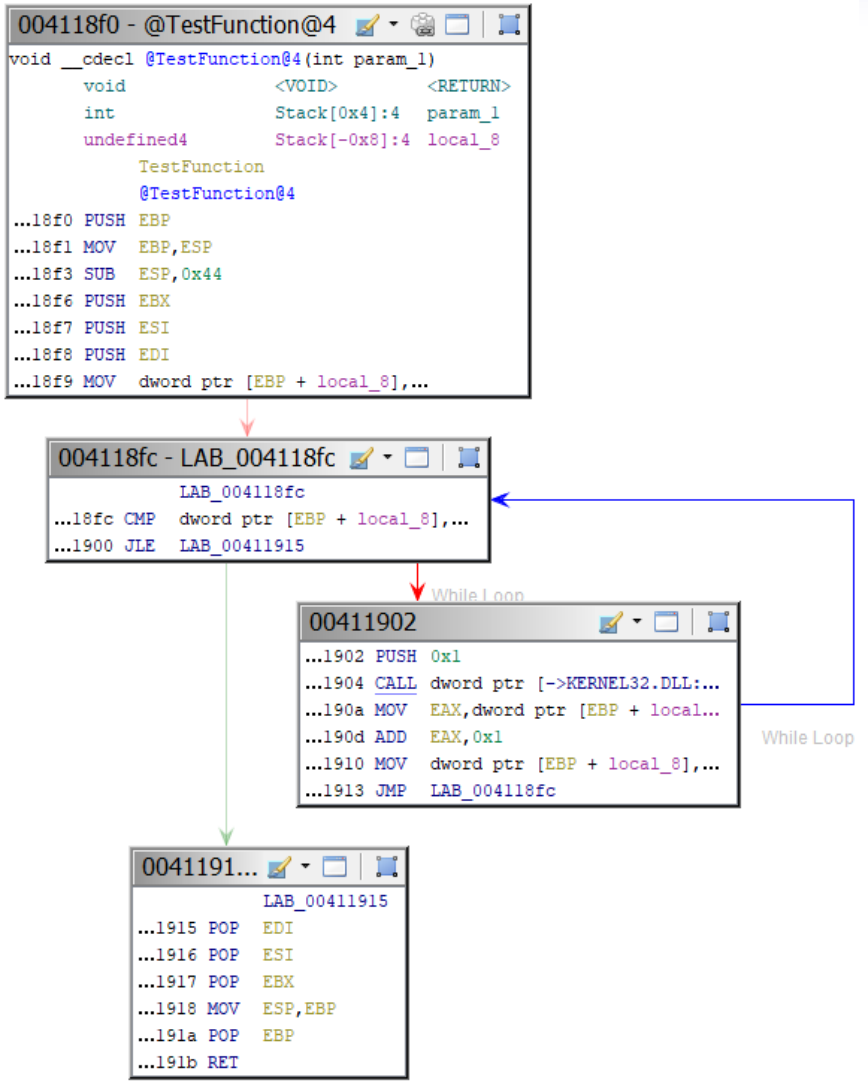
- Boucle : for, while, do, etc.
- Switch
- Idiomes

Structure de contrôle de base

Boucles :

- **for** (init ; condition ; loop) statement
- **while** (expression) statement
- **do** statement **while** (expression) ;

```
                    TestFunction
                    @TestFunction@4
004118f0    PUSH    EBP
004118f1    MOV     EBP,ESP
004118f3    SUB     ESP,0x44
004118f6    PUSH    EBX
004118f7    PUSH    ESI
004118f8    PUSH    EDI
004118f9    MOV     dword ptr [EBP + local_8],ECX

                    LAB_004118fc
004118fc    CMP     dword ptr [EBP + local_8],0x0
00411900    JLE     LAB_00411915
00411902    PUSH    0x1
00411904    CALL    dword ptr [->KERNEL32.DLL::Sleep]

0041190a    MOV     EAX,dword ptr [EBP + local_8]
0041190d    ADD     EAX,0x1
00411910    MOV     dword ptr [EBP + local_8],EAX
00411913    JMP     LAB_004118fc

                    LAB_00411915
00411915    POP     EDI
00411916    POP     ESI
00411917    POP     EBX
00411918    MOV     ESP,EBP
0041191a    POP     EBP
0041191b    RET
```



Structure de contrôle de base

Switch : « case » **consécutifs** et **peu** nombreux

Exemple

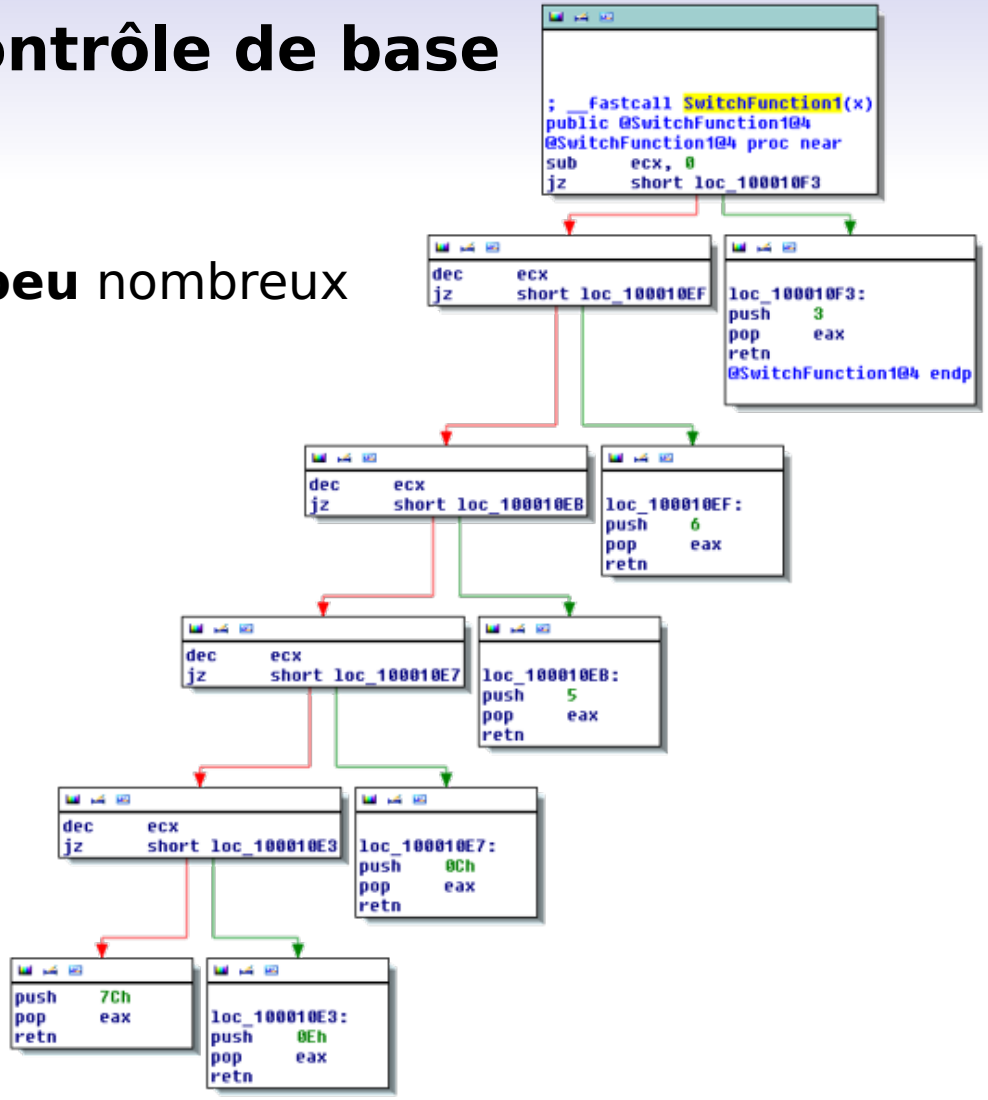
```
ULONG __fastcall Switch1(ULONG n)
{
    switch(n)
    {
        case 0    :return (3);
        case 1    :return (6);
        case 2    :return (5);
        case 3    :return (12);
        case 4    :return (14);
        default   :return (124);
    }
}
```

Structure de contrôle de base

Switch : « case » **consécutifs** et **peu** nombreux

Exemple

```
ULONG __fastcall Switch1(ULONG n)
{
    switch(n)
    {
        case 0    :return (3);
        case 1    :return (6);
        case 2    :return (5);
        case 3    :return (12);
        case 4    :return (14);
        default   :return (124);
    }
}
```



Structure arborescente

Structure de contrôle de base

Switch : « case » **consécutifs** et **plus** nombreux

Exemple

```
ULONG __fastcall Switch2(ULONG n)
{
    switch(n)
    {
        case 0    :return (3);
        case 1    :return (6);
        case 2    :return (5);
        case 3    :return (12);
        case 4    :return (14);
        case 5    :return (15);
        case 6    :return (16);
        case 7    :return (19);
        default   :return (124);
    }
}
```

Structure de contrôle de base

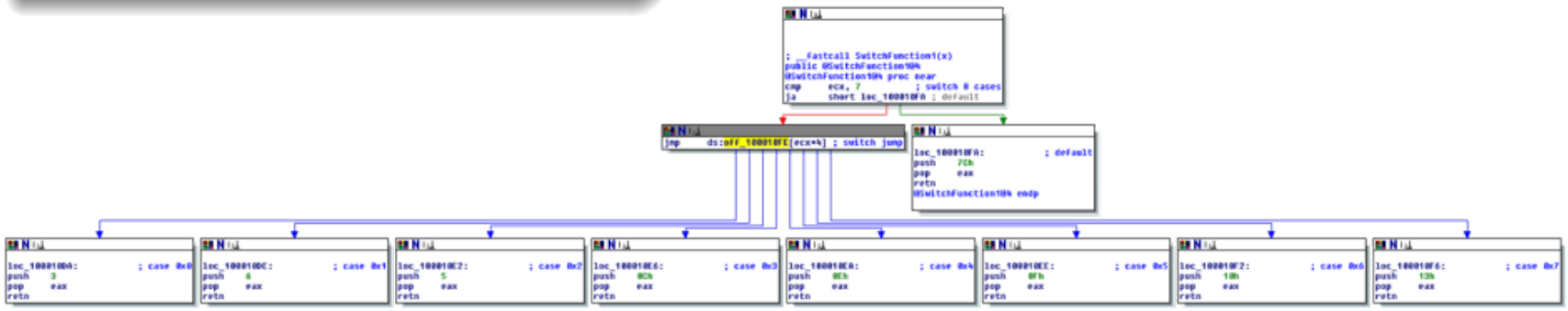
Switch : « case » **consécutifs** et **plus** nombreux

Exemple

```
ULONG __fastcall Switch2(ULONG n)
{
    switch(n)
    {
        case 0    :return (3);
        case 1    :return (6);
        case 2    :return (5);
        case 3    :return (12);
        case 4    :return (14);
        case 5    :return (15);
        case 6    :return (16);
        case 7    :return (19);
        default   :return (124);
    }
}
```

```
off_100010FE    dd offset loc_100010DA  ; DATA XREF: SwitchFunction1(x)+5↑r
                dd offset loc_100010DE  ; jump table for switch statement
                dd offset loc_100010E2
                dd offset loc_100010E6
                dd offset loc_100010EA
                dd offset loc_100010EE
                dd offset loc_100010F2
                dd offset loc_100010F6
```

Génération d'une table de « switch » (de traitements)



Structure de contrôle de base

Switch : « case » **non consécutifs** à **traitements independants**

Exemple

```
ULONG __fastcall Switch3(ULONG n)
{
  switch(n)
  {
      case 0    :return (3);
      case 12   :return (6);
      case 21   :return (5);
      case 35   :return (12);
      case 41   :return (14);
      case 57   :return (15);
      case 60   :return (16);
      case 73   :return (19);
      default   :return (124);
  }
}
```

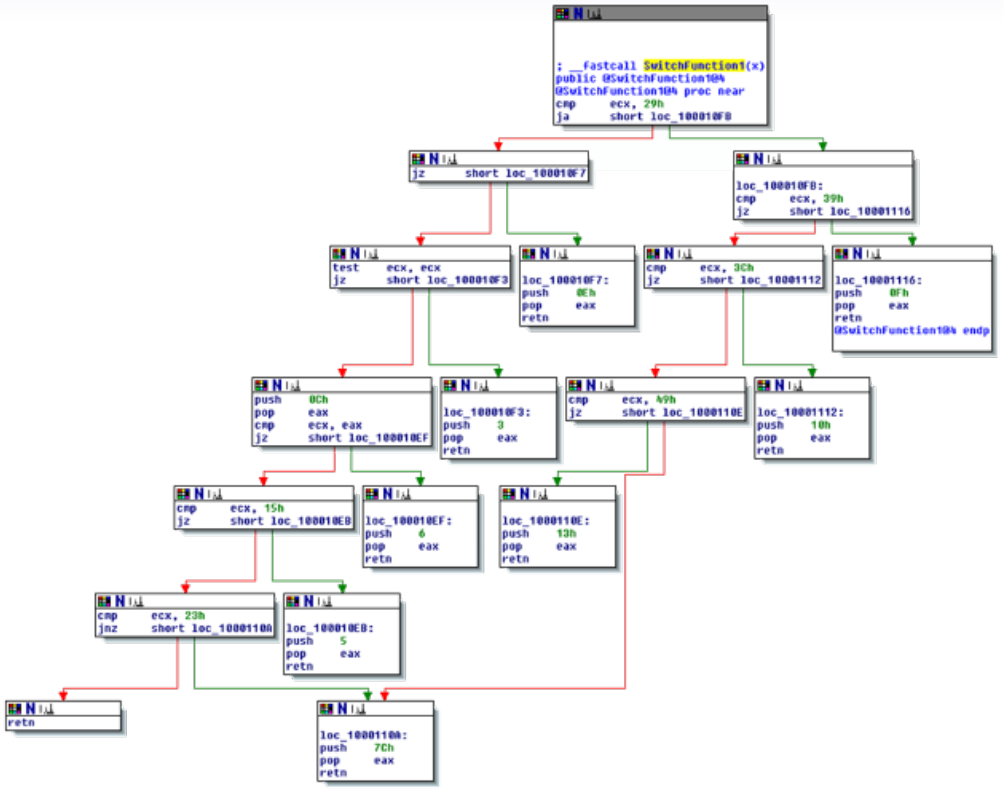
Structure de contrôle de base

Switch : « case » **non consécutifs** à **traitements independants**

Exemple

```
ULONG __fastcall Switch3(ULONG n)
{
  switch(n)
  {
      case 0    :return (3);
      case 12   :return (6);
      case 21   :return (5);
      case 35   :return (12);
      case 41   :return (14);
      case 57   :return (15);
      case 60   :return (16);
      case 73   :return (19);
      default   :return (124);
  }
}
```

Structure arborescente binaire équilibrée



Structure de contrôle de base

Switch : « case » **consécutifs** à **traitements factorisés**

Exemple

```
ULONG __fastcall Switch4(ULONG n)
{
  switch(n)
  {
    case 0 :
    case 1 :
    case 2 :
    case 3 :
    case 4 :
    case 5 : return (15);
    case 6 :
    case 7 :
    case 8 : return (19);
    [...]
    default: return (124);
  }
}
```

Structure de contrôle de base

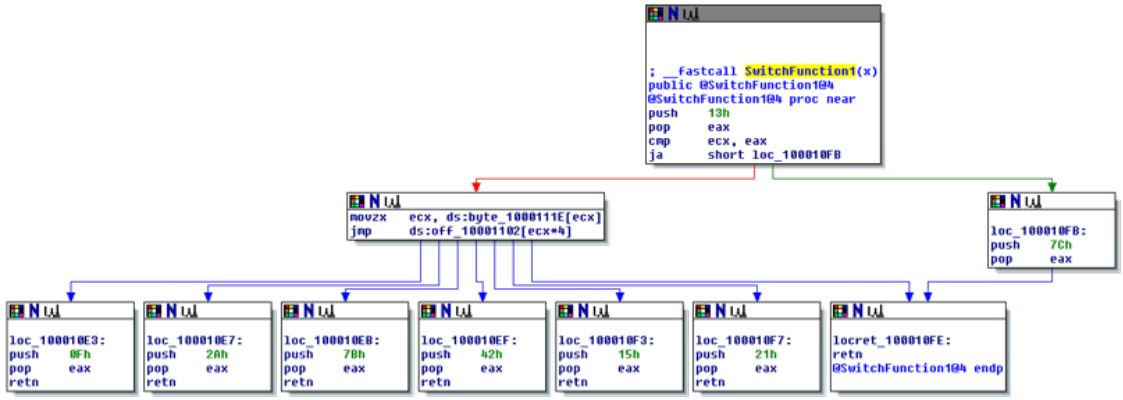
Switch : « case » **consécutifs** à **traitements factorisés**

Exemple

```
ULONG __fastcall Switch4(ULONG n)
{
  switch(n)
  {
    case 0 :
    case 1 :
    case 2 :
    case 3 :
    case 4 :
    case 5 : return (15);
    case 6 :
    case 7 :
    case 8 : return (19);
    [...]
    default: return (124);
  }
}
```

```
byte_1000111E   db 6 dup(0), 3 dup(1), 3 dup(2), 3, 2 dup(4), 2 dup(5)
                                        ; DATA XREF: SwitchFunction1(x)+7↑r
                db 3 dup(6)
off_10001102    dd offset loc_100010E3  ; DATA XREF: SwitchFunction1(x)+E↑r
                dd offset locret_100010FE
                dd offset loc_100010E7
                dd offset loc_100010EB
                dd offset loc_100010EF
                dd offset loc_100010F3
                dd offset loc_100010F7
```

Table d'index (de traitements) & Table de traitements



Structure de contrôle de base

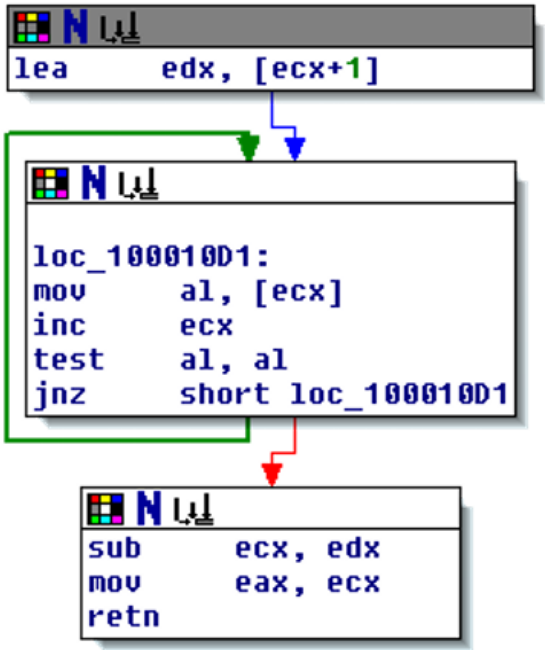
Idiomes remarquables

- Optimisation en taille des fonctions de manipulation de strings via les mnémoniques spécialisés :
  - **CMPS** : compare ES :[EDI] avec DS :[ESI]
  - **MOVS** : copie ES :[EDI] avec DS :[ESI]
  - **SCAS** : compare ES :[EDI] avec EAX
  - **STOS** : stocke EAX vers ES :[EDI]
- Utilisés avec le préfix REP/REPE/RPNE
  - Utilisation de ECX comme compteur
  - Arrêt en fonction de ZF

Structure de contrôle de base

Idiomes remarquables

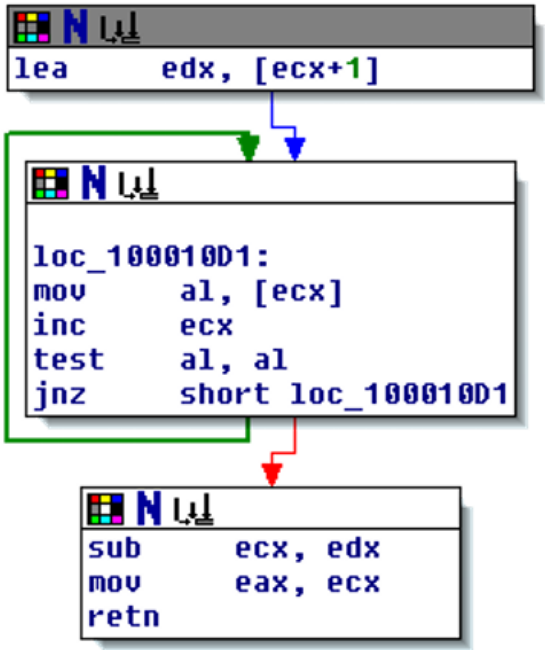
```
arg_0= dword ptr  8

push    edi
sub     ecx, ecx
mov     edi, [esp+arg_0]
not     ecx
sub     al, al
cld
repne scasb
not     ecx
pop     edi
lea     eax, [ecx-1]
retn
```



Structure de contrôle de base

Idiomes remarquables

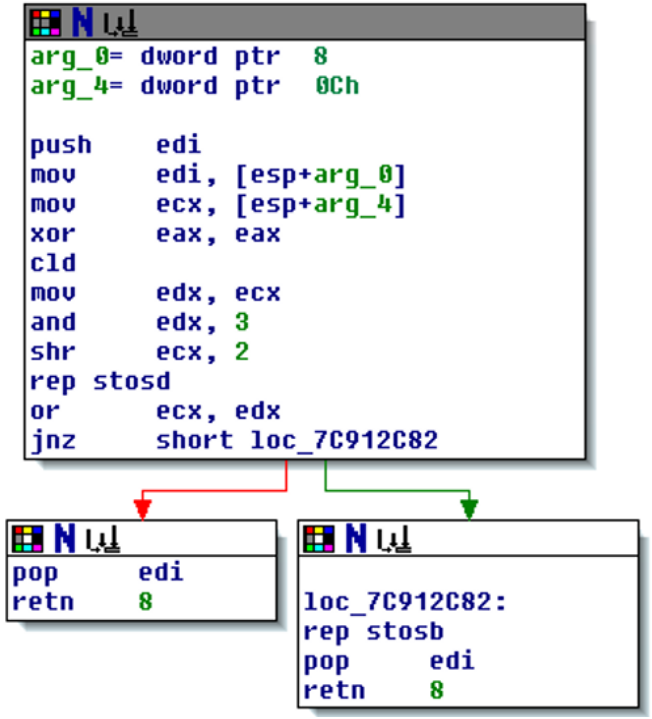
```
arg_0= dword ptr  8

push    edi
sub     ecx, ecx
mov     edi, [esp+arg_0]
not     ecx
sub     al, al
cld
repne scasb
not     ecx
pop     edi
lea     eax, [ecx-1]
retn
```



strlen

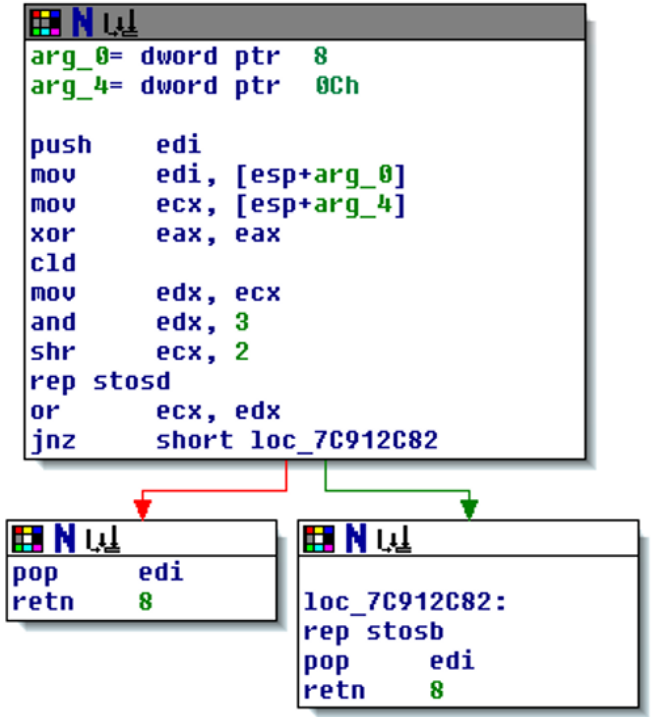
Structure de contrôle de base

Idiomes remarquables



Structure de contrôle de base

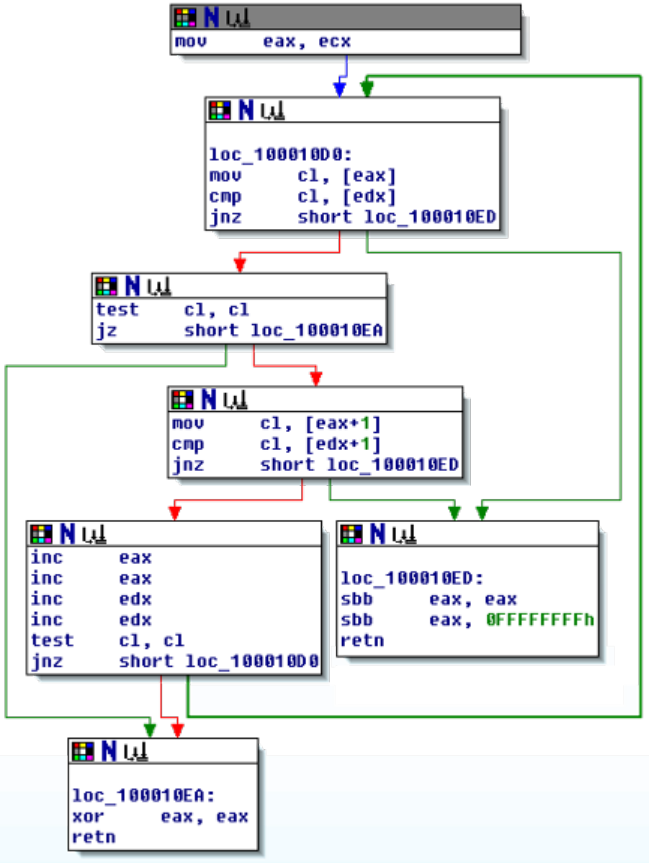
Idiomes remarquables



memset à 0

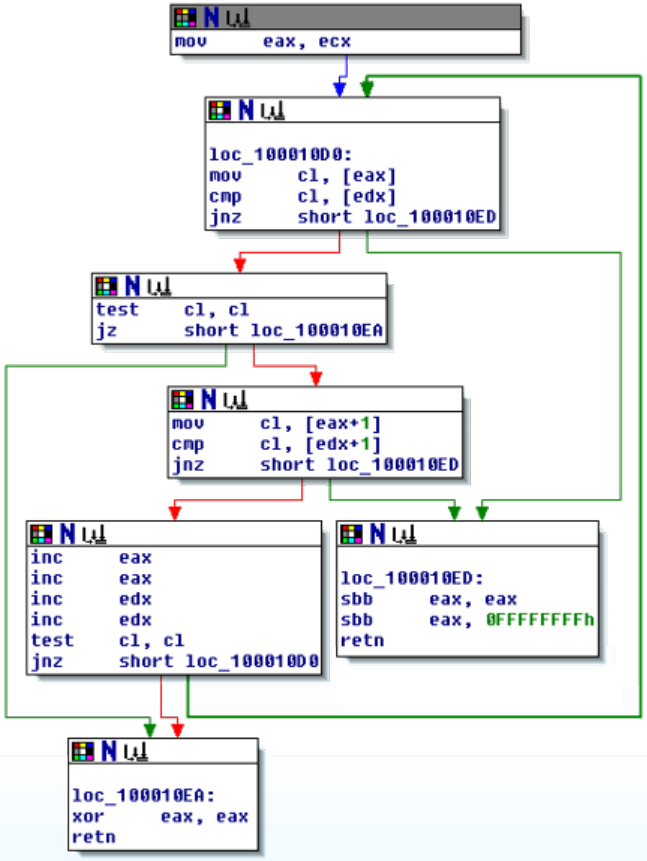
Structure de contrôle de base

Idiomes remarquables



Structure de contrôle de base

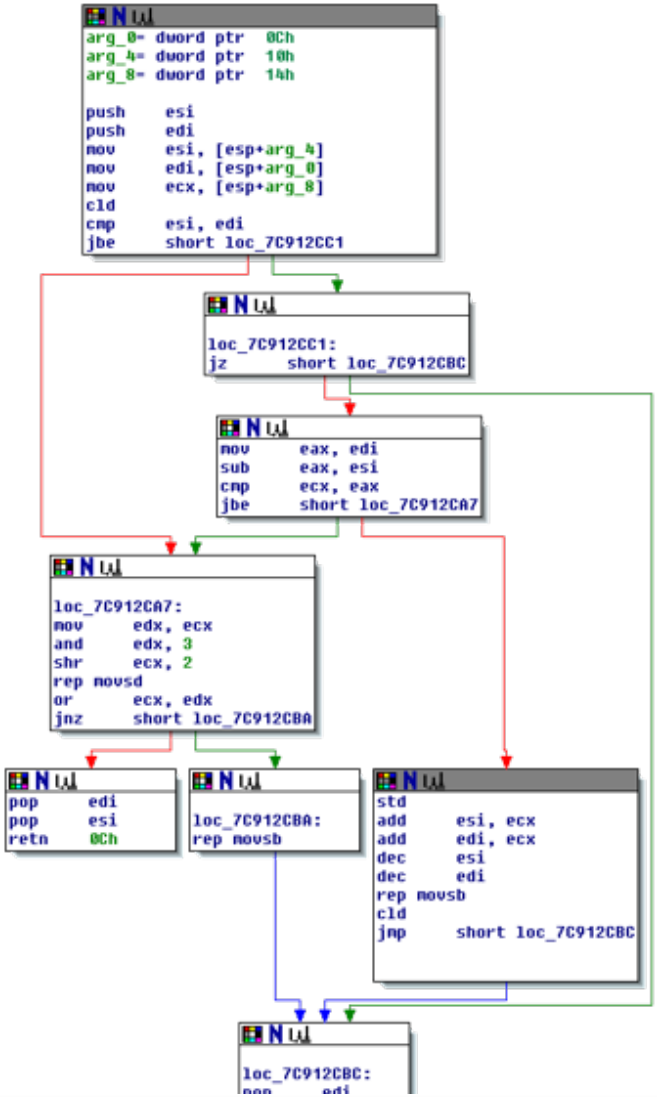
Idiomes remarquables



strcmp

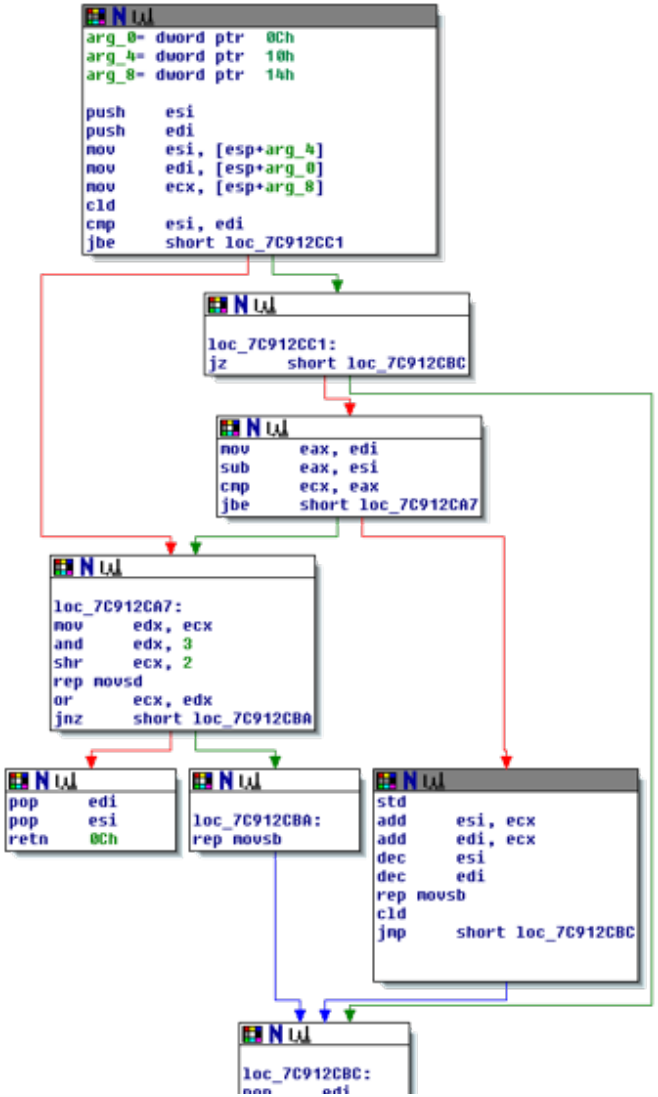
Structure de contrôle de base

Idiomes remarquables



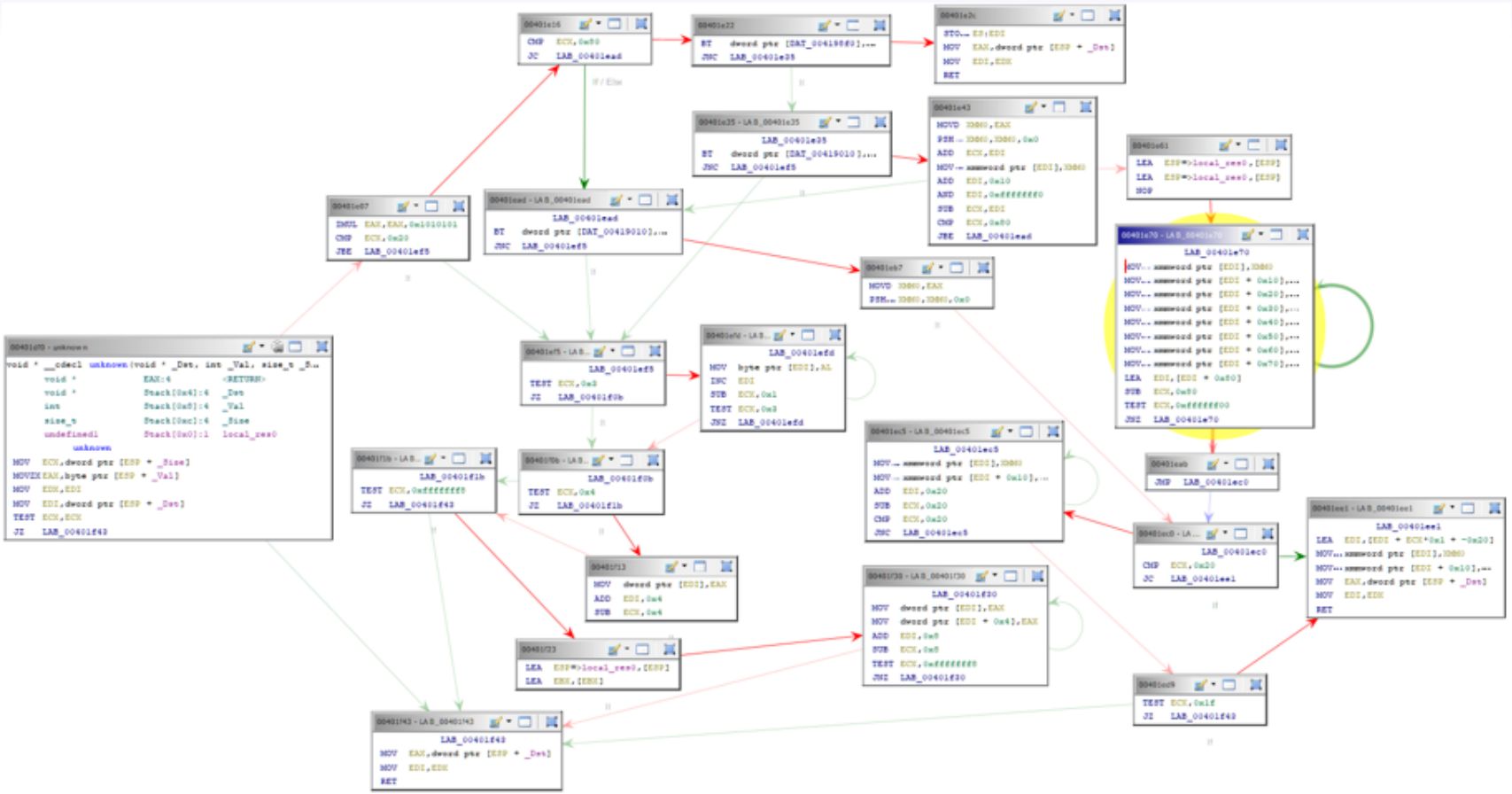
Structure de contrôle de base

Idiomes remarquables



memcpy

Structure de contrôle de base

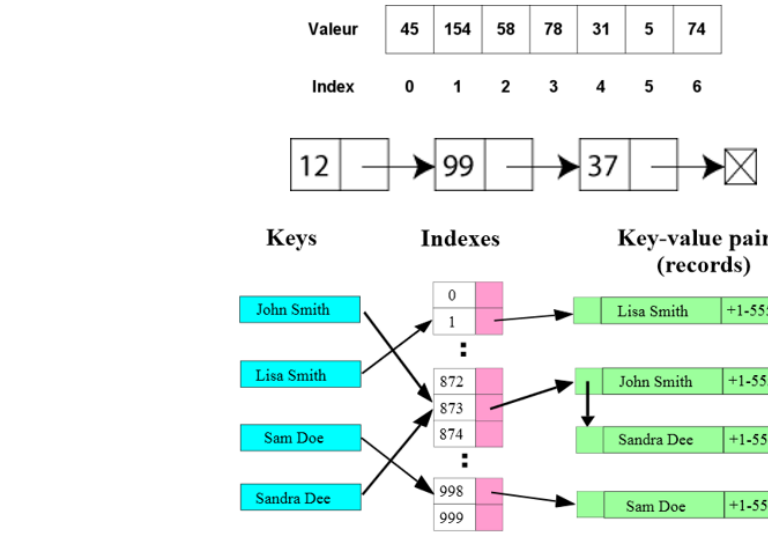


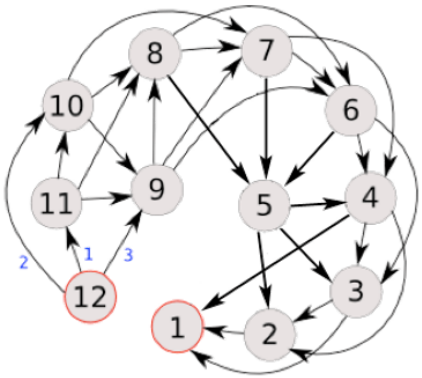
Attention

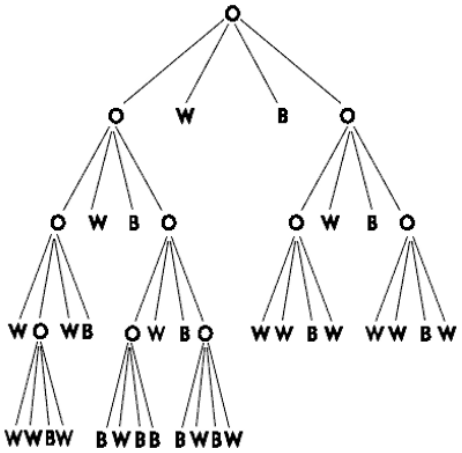
Optimisations des compilateurs (ici un memset)

Structures Algorithmiques

- Tableaux
- **Listes**
- Tables de hachage
- Arbres
- Graphes

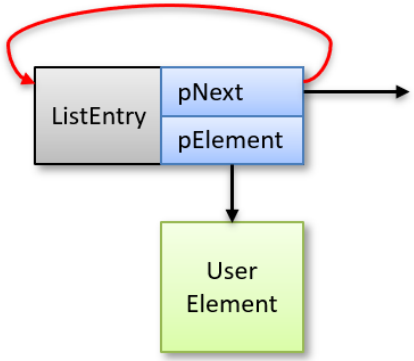






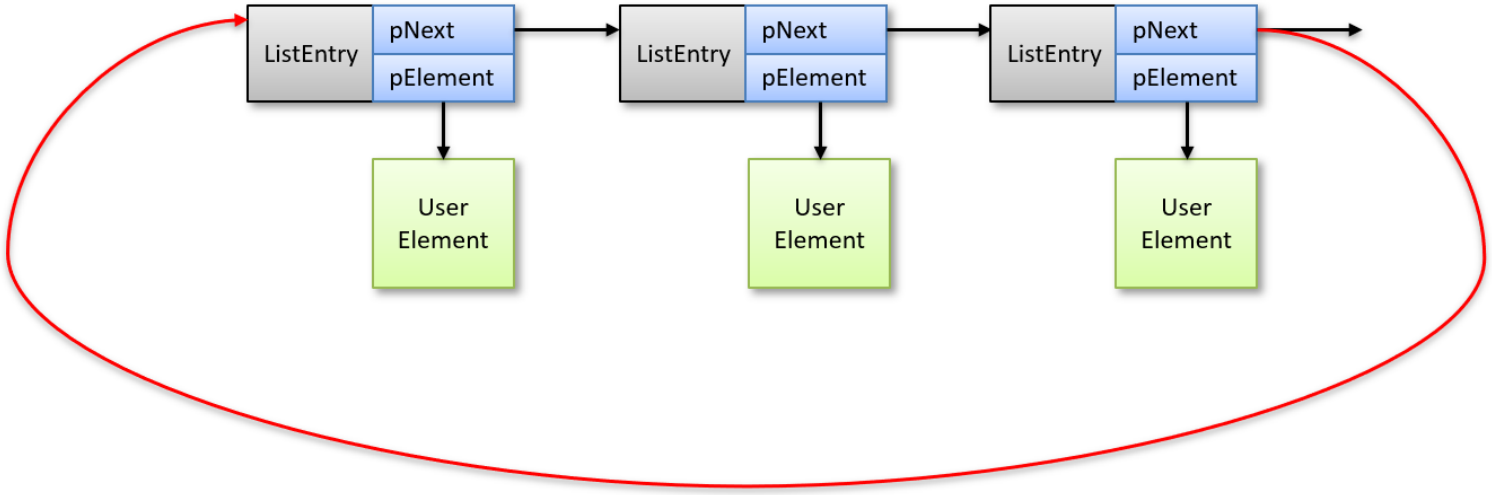
Les listes

Cyclique simplement chaînée Acyclique simplement chaînée



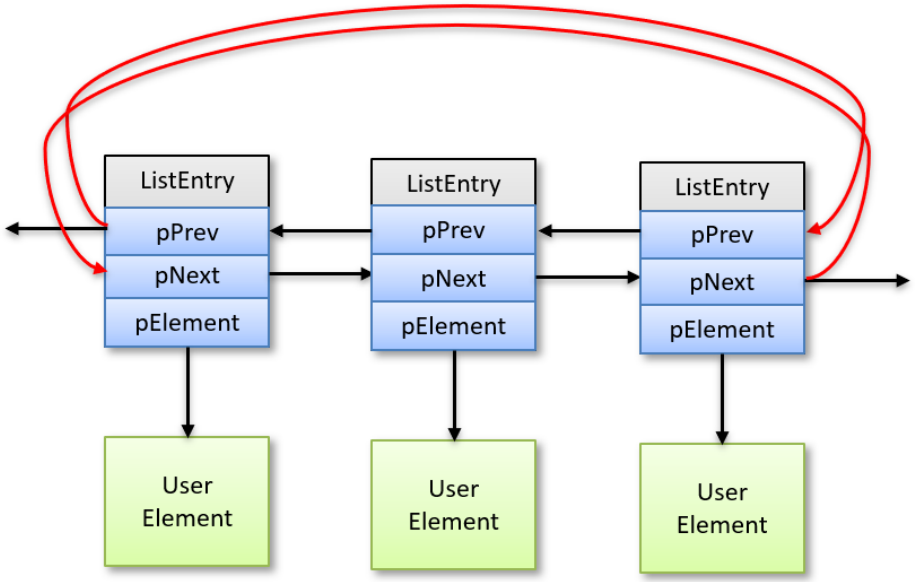
Les listes

Cyclique simplement chaînée Acyclique simplement chaînée



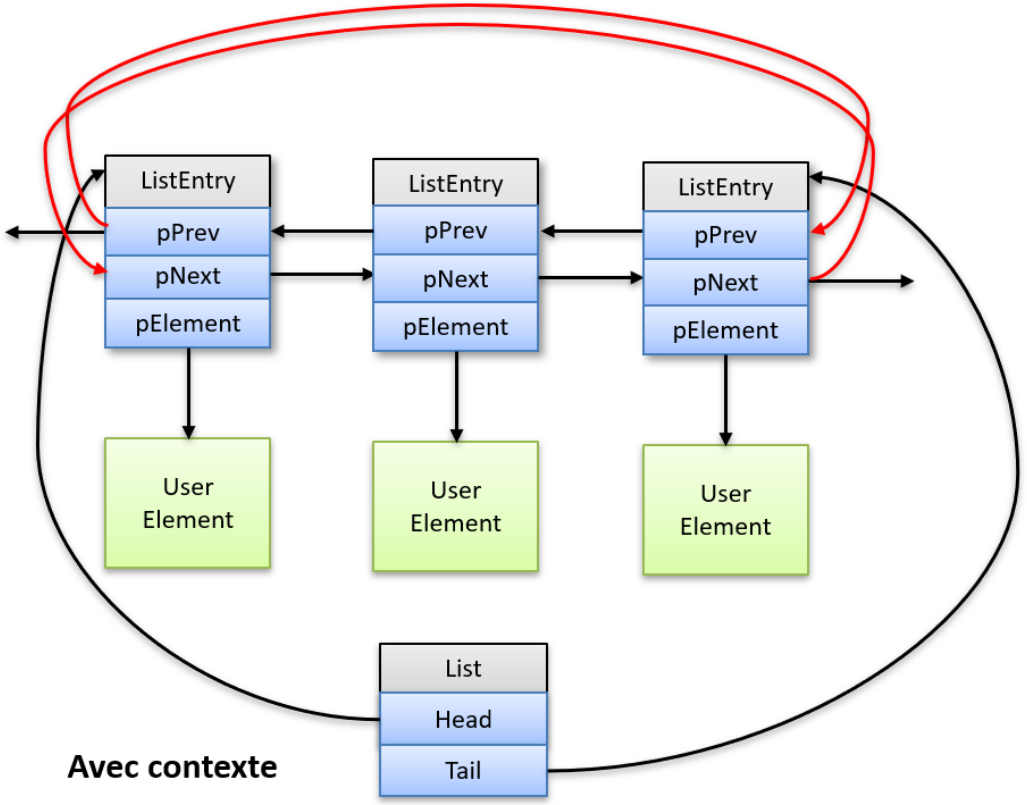
Les listes

Cyclique doublement chaînée Acyclique doublement chaînée



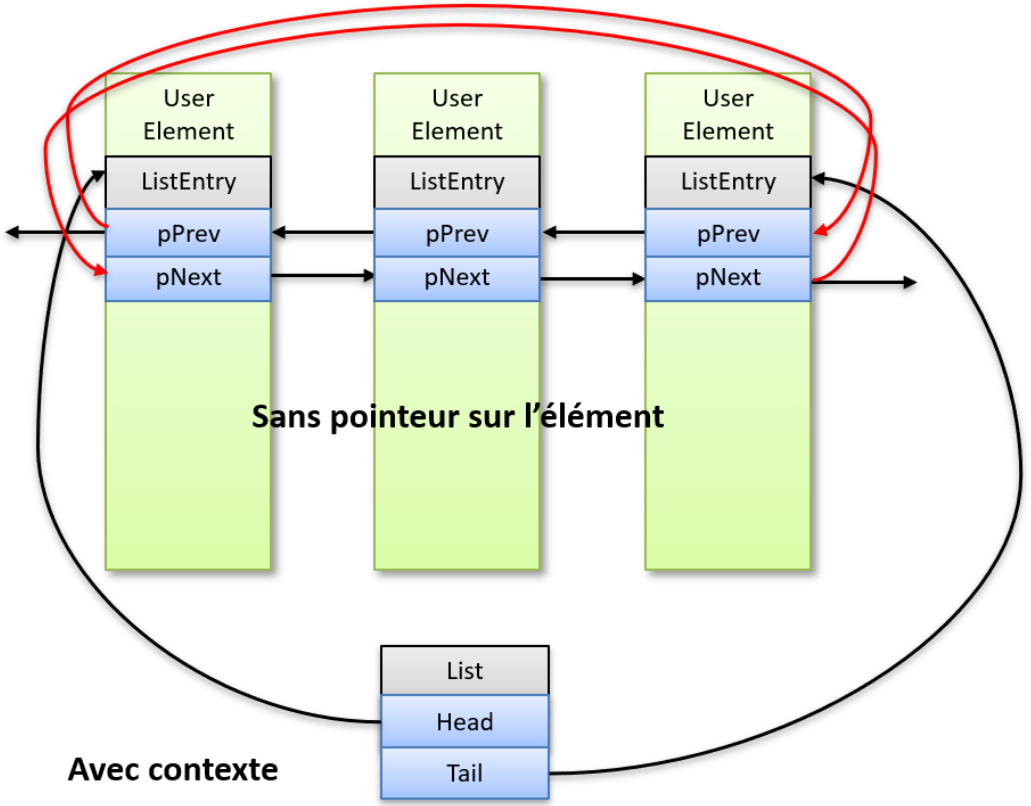
Les listes

Cyclique doublement chaînée Acyclique doublement chaînée



Les listes

Cyclique doublement chaînée Acyclique doublement chaînée



Program Slicing

Définition

*Une «**slice**» de programme est constituée par l'ensemble des instructions du programme qui affectent des valeurs à un point d'intérêt.*

Intérêt du Slicing :

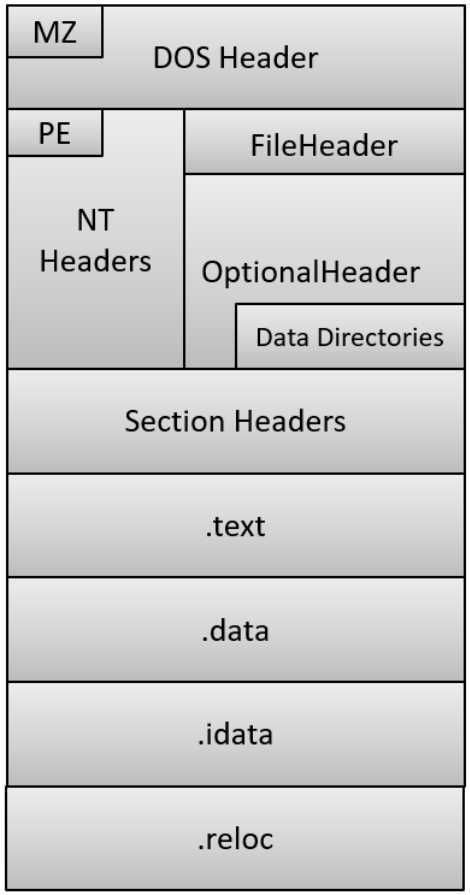
- Se concentrer uniquement sur la partie «utile» du programme
- Analyse d'une seule fonctionnalité à la fois
- Faire une hypothése de fonctionnement et se concentrer dessus

Exemple

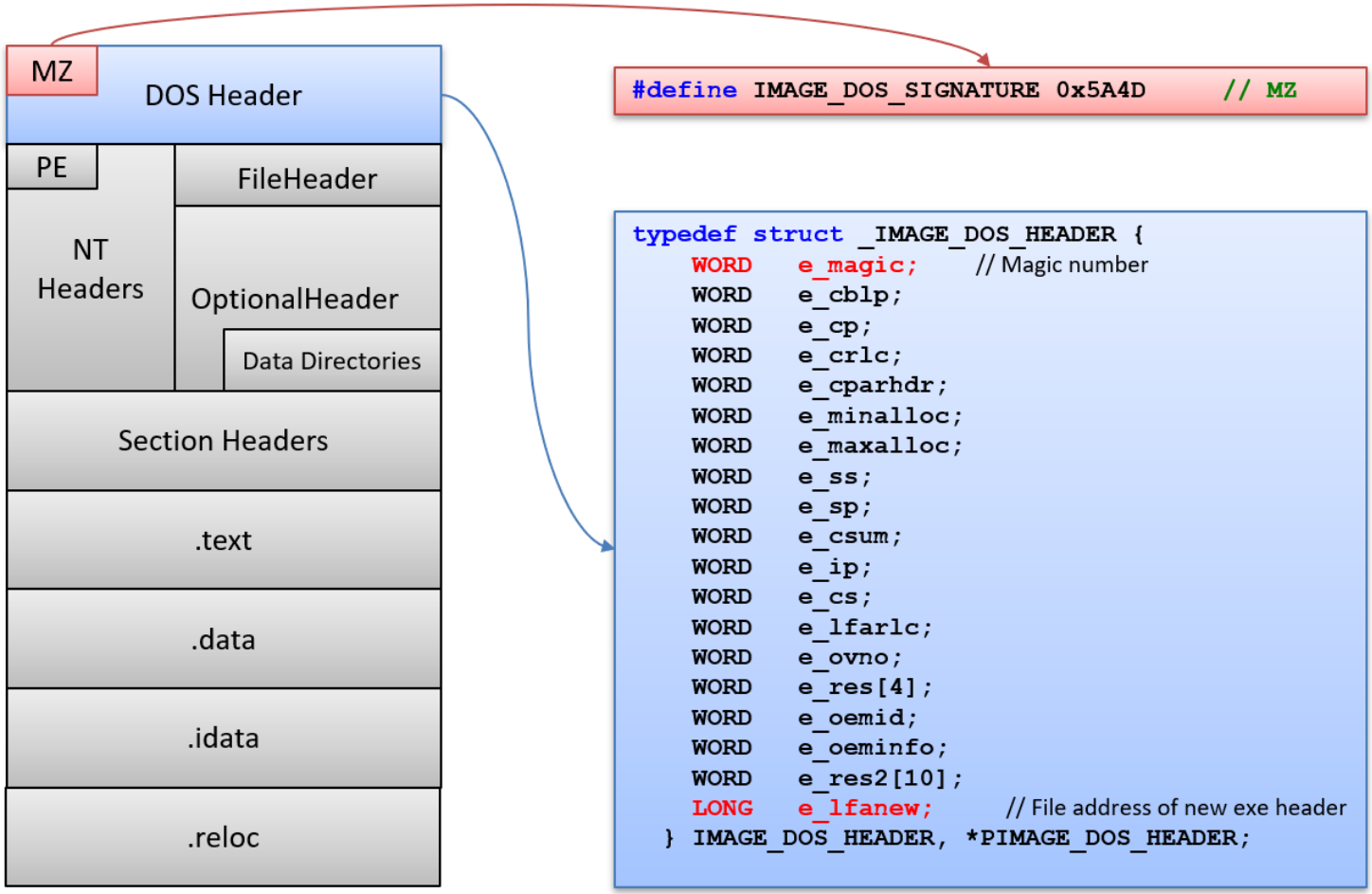
Rechercher la mise en place des points de persistance dans le virus

Plan

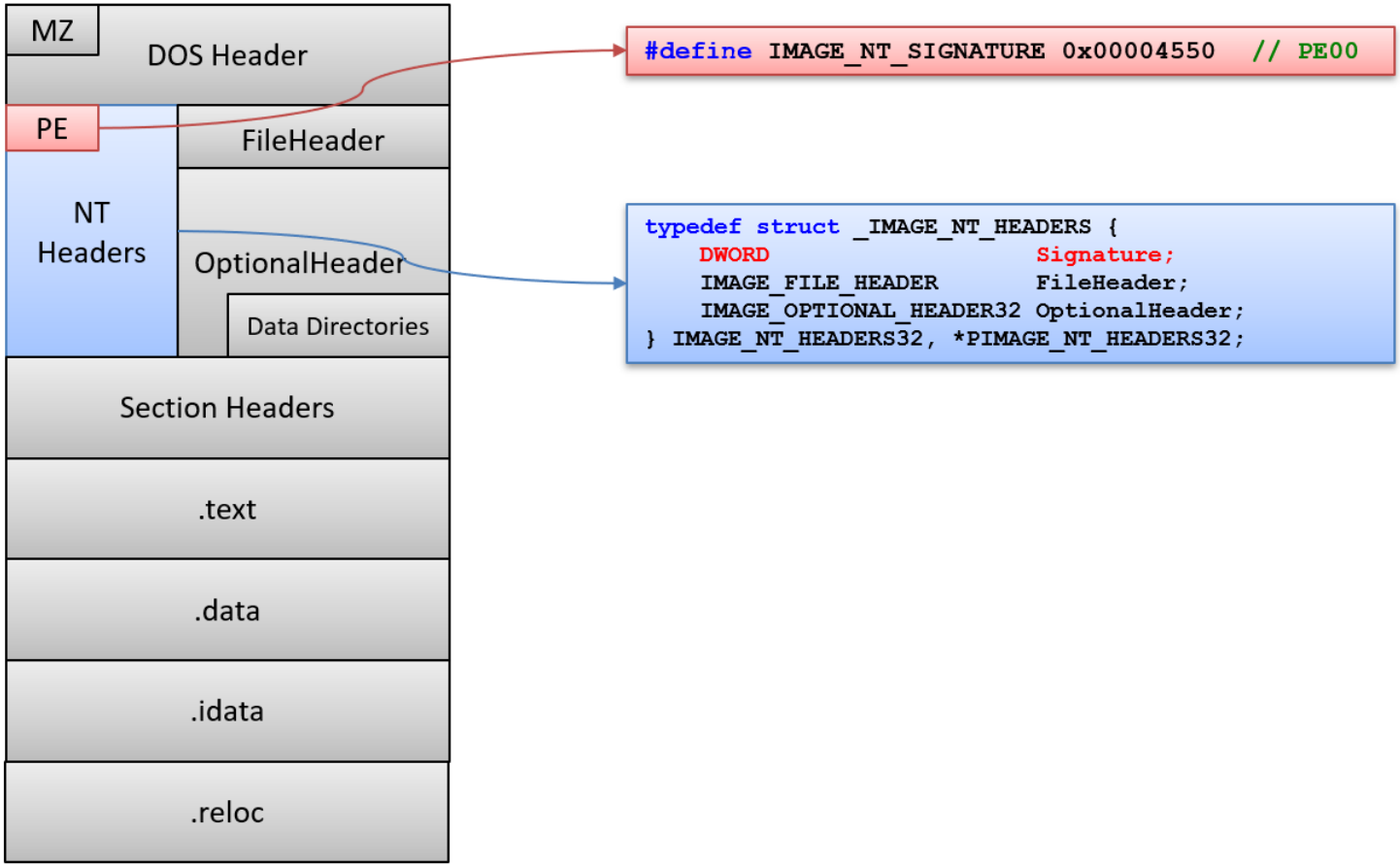
Le format PE


MZ
DOS Header
PE
FileHeader
NT Headers
OptionalHeader
Data Directories
Section Headers
.text
.data
.idata
.reloc

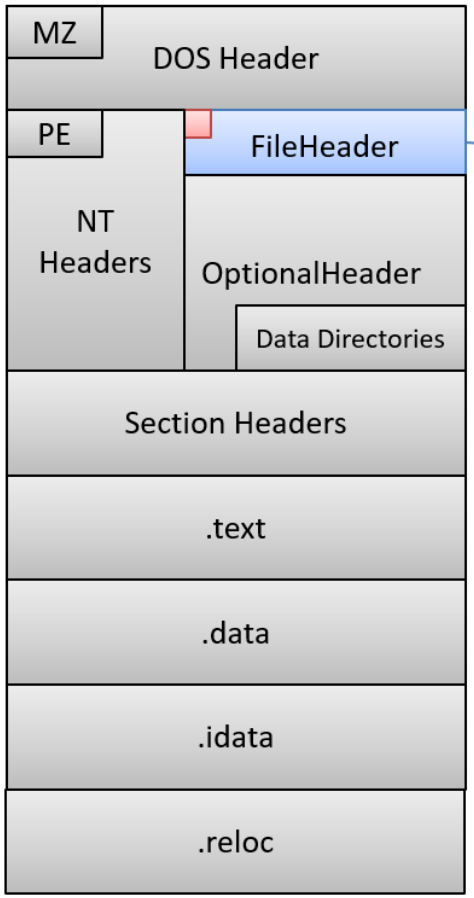
Le format PE


MZ
DOS Header
PE
FileHeader
NT Headers
OptionalHeader
Data Directories
Section Headers
.text
.data
.idata
.reloc
#define IMAGE_DOS_SIGNATURE 0x5A4D // MZ
typedef struct _IMAGE_DOS_HEADER {
WORD e_magic; // Magic number
WORD e_cblp;
WORD e_cp;
WORD e_crlc;
WORD e_cparhdr;
WORD e_minalloc;
WORD e_maxalloc;
WORD e_ss;
WORD e_sp;
WORD e_csum;
WORD e_ip;
WORD e_cs;
WORD e_lfarlc;
WORD e_ovno;
WORD e_res[4];
WORD e_oemid;
WORD e_oeminfo;
WORD e_res2[10];
LONG e_lfanew; // File address of new exe header
} IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;

Le format PE


MZ
DOS Header
PE
FileHeader
NT Headers
OptionalHeader
Data Directories
Section Headers
.text
.data
.idata
.reloc
#define IMAGE_NT_SIGNATURE 0x00004550 // PE00
typedef struct _IMAGE_NT_HEADERS {
DWORD Signature;
IMAGE_FILE_HEADER FileHeader;
IMAGE_OPTIONAL_HEADER32 OptionalHeader;
} IMAGE_NT_HEADERS32, *PIMAGE_NT_HEADERS32;

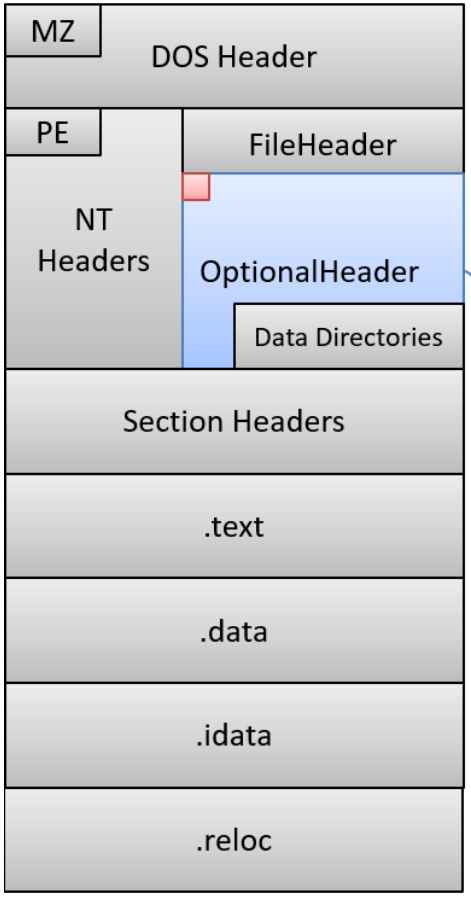
Le format PE




```
#define IMAGE_FILE_MACHINE_I386      0x014c  // Intel 386
#define IMAGE_FILE_MACHINE_IA64      0x0200  // Intel 64
#define IMAGE_FILE_MACHINE_AMD64     0x8664  // AMD64 (K8)
```

```
typedef struct _IMAGE_FILE_HEADER {
    WORD    Machine;
    WORD    NumberOfSections;
    DWORD   TimeDateStamp;
    DWORD   PointerToSymbolTable;
    DWORD   NumberOfSymbols;
    WORD    SizeOfOptionalHeader;
    WORD    Characteristics;
} IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;
```

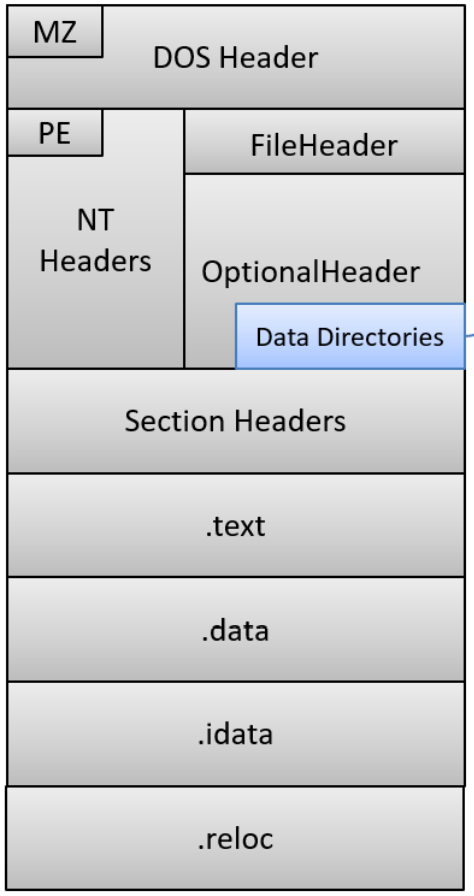
Le format PE




```
#define IMAGE_NT_OPTIONAL_HDR32_MAGIC      0x10b
#define IMAGE_NT_OPTIONAL_HDR64_MAGIC      0x20b
```

```
typedef struct _IMAGE_OPTIONAL_HEADER {
    WORD                 Magic;
    BYTE                 MajorLinkerVersion;
    BYTE                 MinorLinkerVersion;
    DWORD                SizeOfCode;
    DWORD                SizeOfInitializedData;
    DWORD                SizeOfUninitializedData;
    DWORD                AddressOfEntryPoint;
    DWORD                BaseOfCode;
    DWORD                BaseOfData;
    DWORD                ImageBase;
    DWORD                SectionAlignment;
    DWORD                FileAlignment;
    WORD                 MajorOperatingSystemVersion;
    WORD                 MinorOperatingSystemVersion;
    WORD                 MajorImageVersion;
    WORD                 MinorImageVersion;
    WORD                 MajorSubsystemVersion;
    WORD                 MinorSubsystemVersion;
    DWORD                Win32VersionValue;
    DWORD                SizeOfImage;
    DWORD                SizeOfHeaders;
    DWORD                CheckSum;
    WORD                 Subsystem;
    WORD                 DllCharacteristics;
    DWORD                SizeOfStackReserve;
    DWORD                SizeOfStackCommit;
    DWORD                SizeOfHeapReserve;
    DWORD                SizeOfHeapCommit;
    DWORD                LoaderFlags;
    DWORD                NumberOfRvaAndSizes;
    IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
} IMAGE_OPTIONAL_HEADER32, *PIMAGE_OPTIONAL_HEADER32;
```

Le format PE

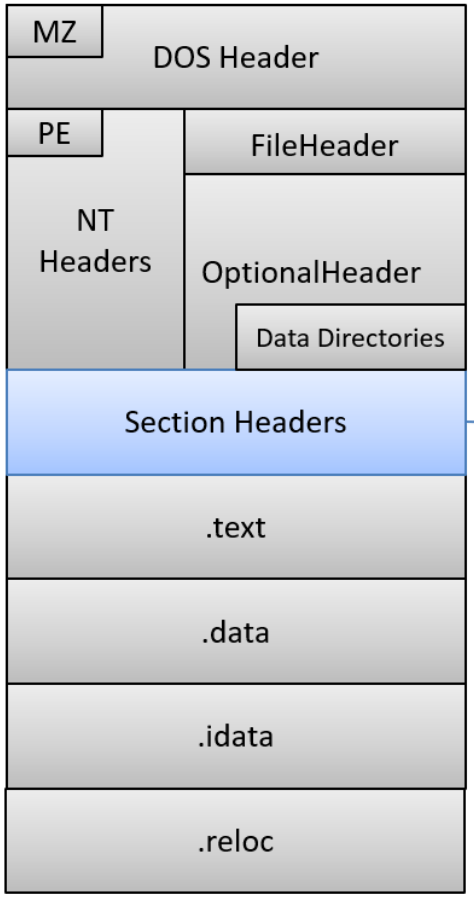



```
typedef struct _IMAGE_DATA_DIRECTORY {
    DWORD   VirtualAddress;         //RVA
    DWORD   Size;
} IMAGE_DATA_DIRECTORY, *PIMAGE_DATA_DIRECTORY;
```

```
#define IMAGE_DIRECTORY_ENTRY_EXPORT          0
#define IMAGE_DIRECTORY_ENTRY_IMPORT          1
#define IMAGE_DIRECTORY_ENTRY_RESOURCE        2
#define IMAGE_DIRECTORY_ENTRY_EXCEPTION       3
#define IMAGE_DIRECTORY_ENTRY_SECURITY        4
#define IMAGE_DIRECTORY_ENTRY_BASERELOC       5
#define IMAGE_DIRECTORY_ENTRY_DEBUG           6
#define IMAGE_DIRECTORY_ENTRY_ARCHITECTURE    7
#define IMAGE_DIRECTORY_ENTRY_GLOBALPTR       8
#define IMAGE_DIRECTORY_ENTRY_TLS             9
#define IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG    10
#define IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT   11
#define IMAGE_DIRECTORY_ENTRY_IAT            12
#define IMAGE_DIRECTORY_ENTRY_DELAY_IMPORT   13
#define IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR 14
```

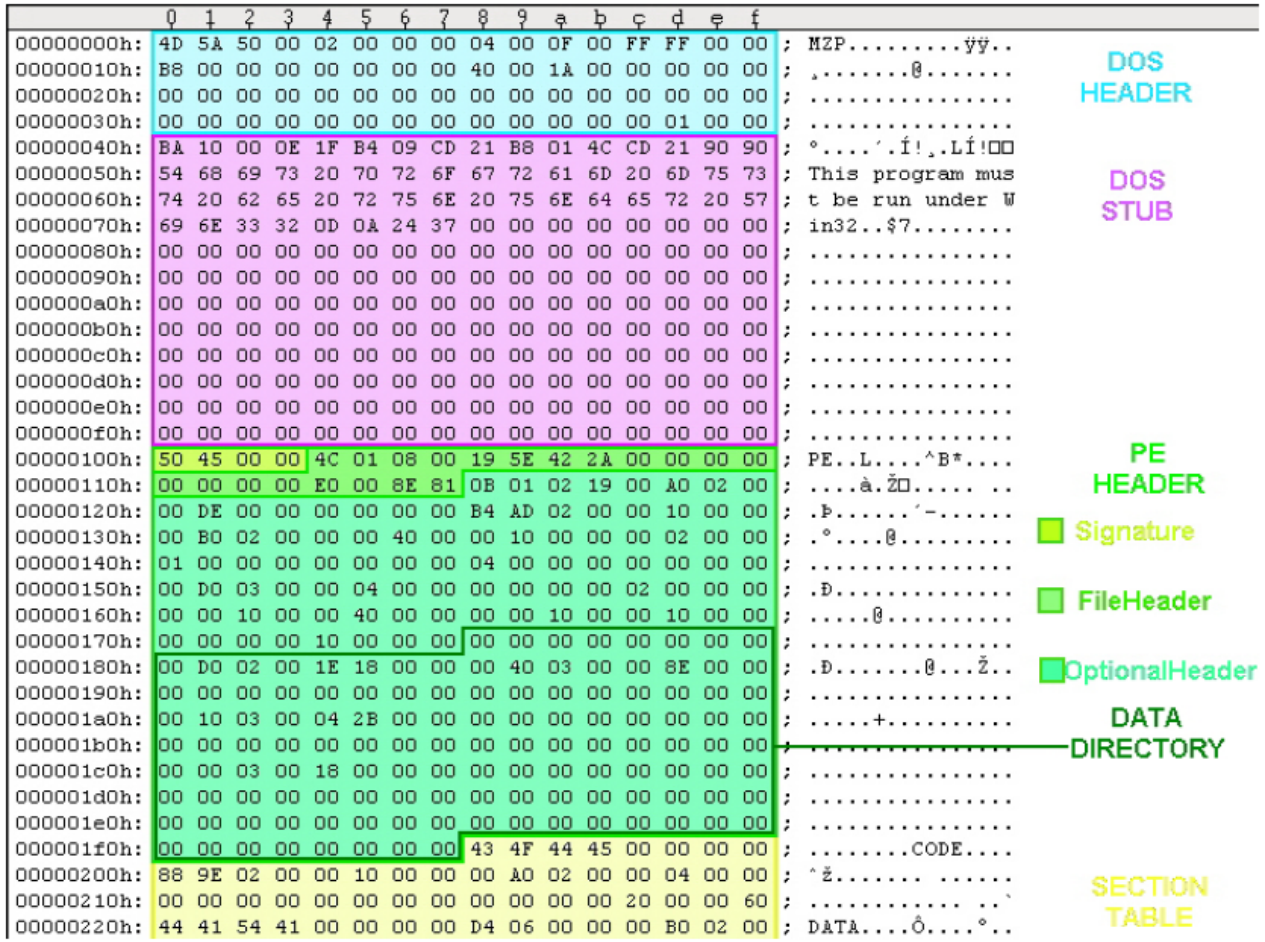
```
typedef struct _IMAGE_EXPORT_DIRECTORY {
    DWORD   Characteristics;
    DWORD   TimeDateStamp;
    WORD    MajorVersion;
    WORD    MinorVersion;
    DWORD   Name;
    DWORD   Base;
    DWORD   NumberOfFunctions;
    DWORD   NumberOfNames;
    DWORD   AddressOfFunctions;     // RVA from base of image
    DWORD   AddressOfNames;         // RVA from base of image
    DWORD   AddressOfNameOrdinals;  // RVA from base of image
} IMAGE_EXPORT_DIRECTORY, *PIMAGE_EXPORT_DIRECTORY;
```

Le format PE

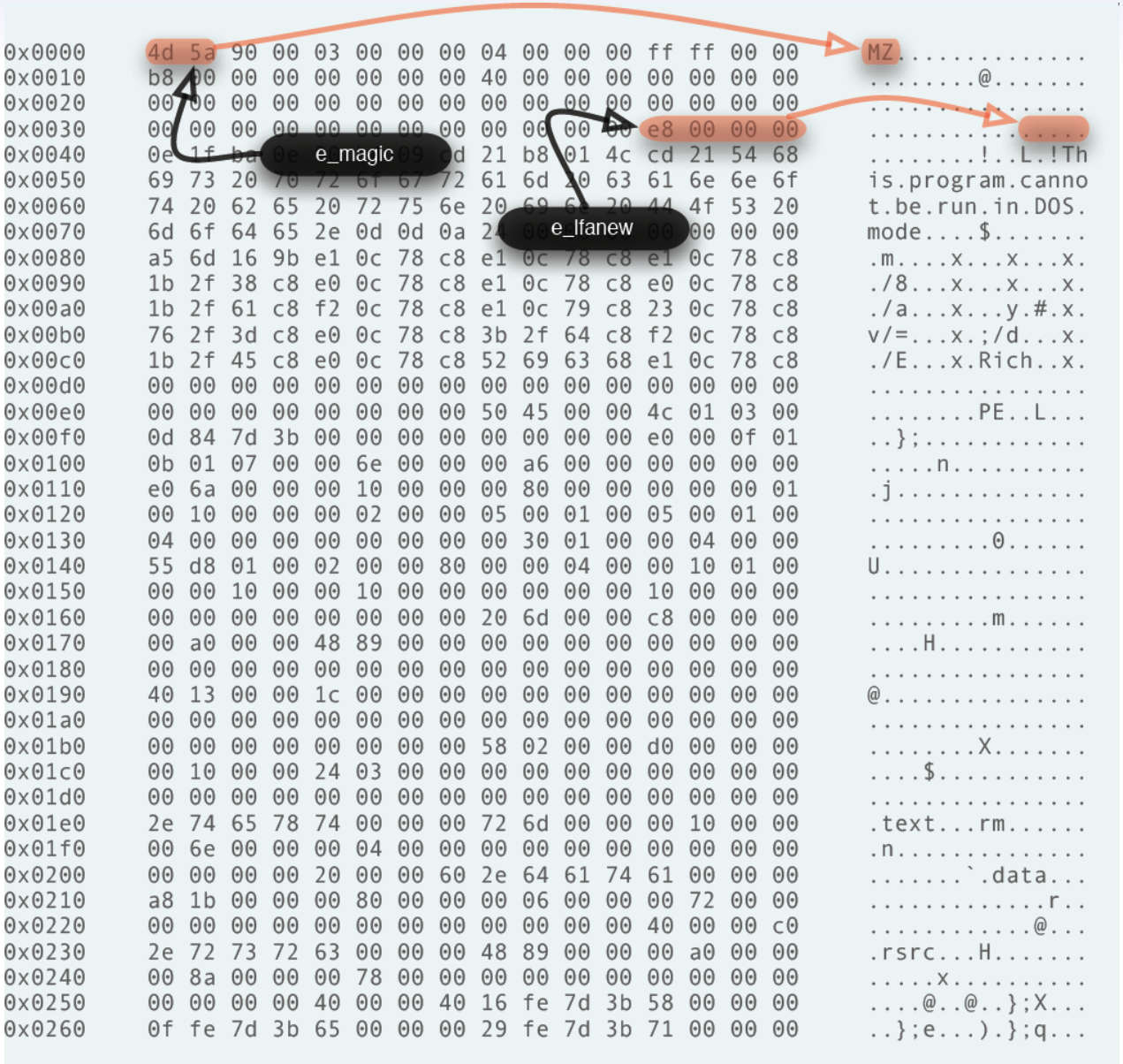



```
typedef struct _IMAGE_SECTION_HEADER {
    BYTE    Name[IMAGE_SIZEOF_SHORT_NAME];
    union {
            DWORD   PhysicalAddress;
            DWORD   VirtualSize;
    } Misc;
    DWORD   VirtualAddress;         // RVA
    DWORD   SizeOfRawData;
    DWORD   PointerToRawData;
    DWORD   PointerToRelocations;
    DWORD   PointerToLinenumbers;
    WORD    NumberOfRelocations;
    WORD    NumberOfLinenumbers;
    DWORD   Characteristics;
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

Format PE

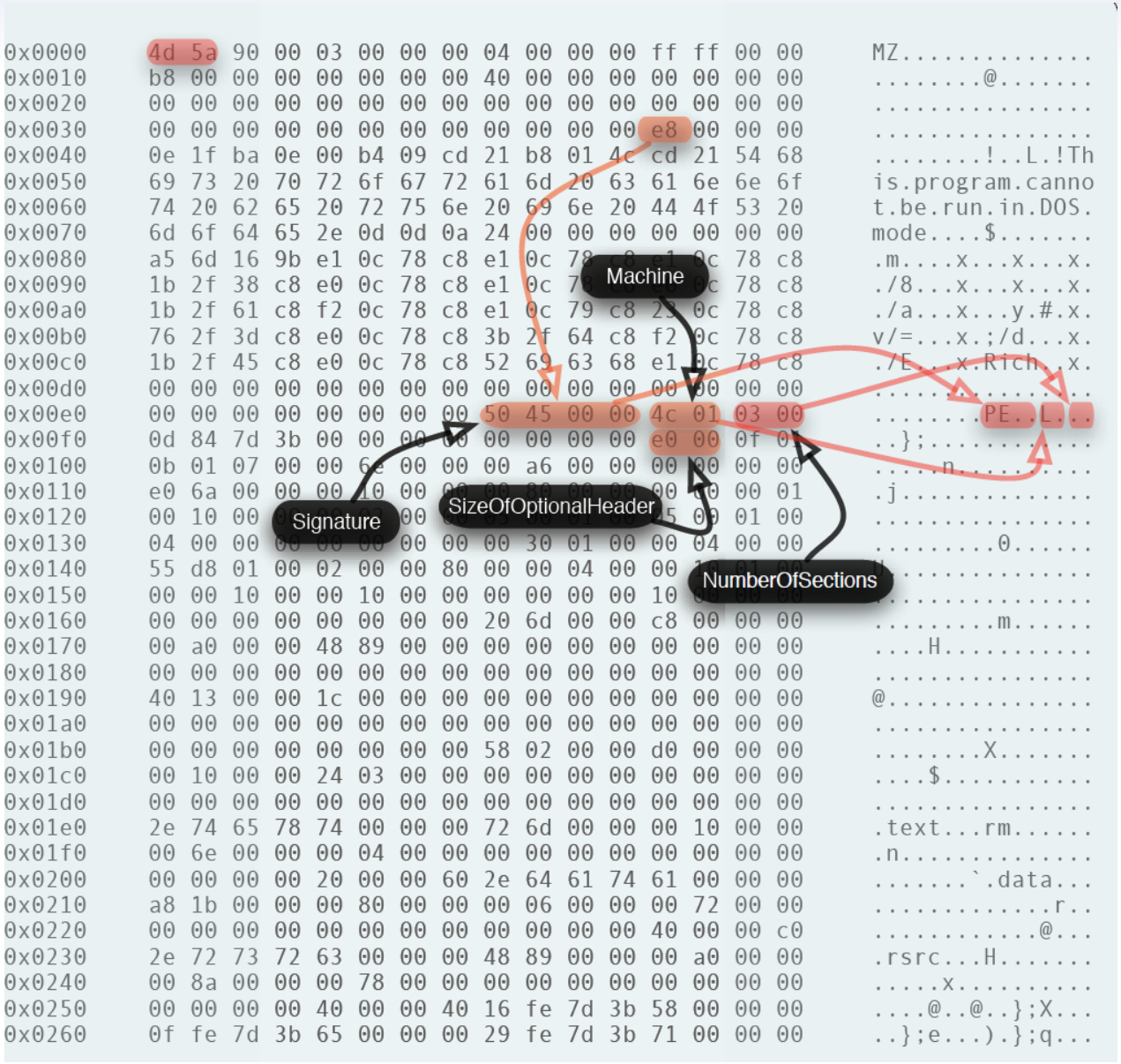

DOS HEADER
DOS STUB
PE HEADER
Signature
FileHeader
OptionalHeader
DATA DIRECTORY
SECTION TABLE

Format PE : Dos Header



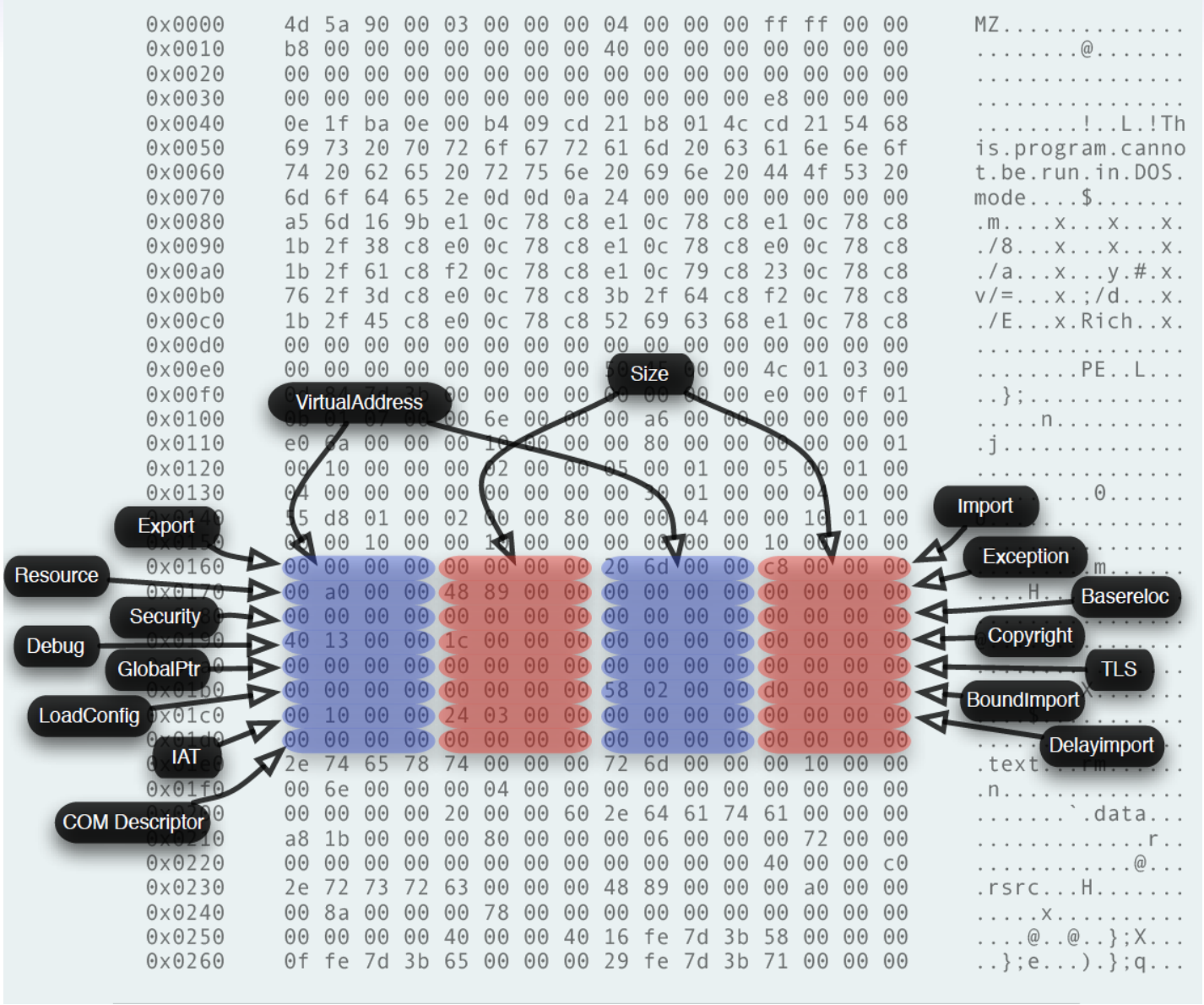

```
0x0000  4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00  MZ..............
0x0010  b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ........@.......
0x0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x0030  00 00 00 00 00 00 00 00 00 00 00 00 e8 00 00 00  ................
0x0040  0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c cd 21 54 68  ........!..L.!Th
0x0050  69 73 20 70 72 6f 67 72 61 6d 20 63 61 6e 6e 6f  is.program.canno
0x0060  74 20 62 65 20 72 75 6e 20 69 6e 20 44 4f 53 20  t.be.run.in.DOS.
0x0070  6d 6f 64 65 2e 0d 0d 0a 24 00 00 00 00 00 00 00  mode....$.......
0x0080  a5 6d 16 9b e1 0c 78 c8 e1 0c 78 c8 e1 0c 78 c8  .m....x...x...x.
0x0090  1b 2f 38 c8 e0 0c 78 c8 e1 0c 78 c8 e0 0c 78 c8  ./8...x...x...x.
0x00a0  1b 2f 61 c8 f2 0c 78 c8 e1 0c 79 c8 23 0c 78 c8  ./a...x...y.#.x.
0x00b0  76 2f 3d c8 e0 0c 78 c8 3b 2f 64 c8 f2 0c 78 c8  v/=...x.;/d...x.
0x00c0  1b 2f 45 c8 e0 0c 78 c8 52 69 63 68 e1 0c 78 c8  ./E...x.Rich..x.
0x00d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x00e0  00 00 00 00 00 00 00 00 50 45 00 00 4c 01 03 00  ........PE..L...
0x00f0  0d 84 7d 3b 00 00 00 00 00 00 00 00 e0 00 0f 01  ..};............
0x0100  0b 01 07 00 00 6e 00 00 00 a6 00 00 00 00 00 00  .....n..........
0x0110  e0 6a 00 00 00 10 00 00 00 80 00 00 00 00 00 01  .j..............
0x0120  00 10 00 00 00 02 00 00 05 00 01 00 05 00 01 00  ................
0x0130  04 00 00 00 00 00 00 00 00 30 01 00 00 04 00 00  .........0......
0x0140  55 d8 01 00 02 00 00 80 00 00 04 00 00 10 01 00  U...............
0x0150  00 00 10 00 00 10 00 00 00 00 00 00 10 00 00 00  ................
0x0160  00 00 00 00 00 00 00 00 20 6d 00 00 c8 00 00 00  .........m......
0x0170  00 a0 00 00 48 89 00 00 00 00 00 00 00 00 00 00  ....H...........
0x0180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x0190  40 13 00 00 1c 00 00 00 00 00 00 00 00 00 00 00  @...............
0x01a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x01b0  00 00 00 00 00 00 00 00 58 02 00 00 d0 00 00 00  ........X.......
0x01c0  00 10 00 00 24 03 00 00 00 00 00 00 00 00 00 00  ....$...........
0x01d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x01e0  2e 74 65 78 74 00 00 00 72 6d 00 00 00 10 00 00  .text...rm......
0x01f0  00 6e 00 00 00 04 00 00 00 00 00 00 00 00 00 00  .n..............
0x0200  00 00 00 00 20 00 00 60 2e 64 61 74 61 00 00 00  ....... `.data...
0x0210  a8 1b 00 00 00 80 00 00 00 06 00 00 00 72 00 00  .............r..
0x0220  00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 c0  ............@...
0x0230  2e 72 73 72 63 00 00 00 48 89 00 00 00 a0 00 00  .rsrc...H.......
0x0240  00 8a 00 00 00 78 00 00 00 00 00 00 00 00 00 00  .....x..........
0x0250  00 00 00 00 40 00 00 40 16 fe 7d 3b 58 00 00 00  ....@..@..};X...
0x0260  0f fe 7d 3b 65 00 00 00 29 fe 7d 3b 71 00 00 00  ..};e...).};q...
```

Format PE : NT Headers



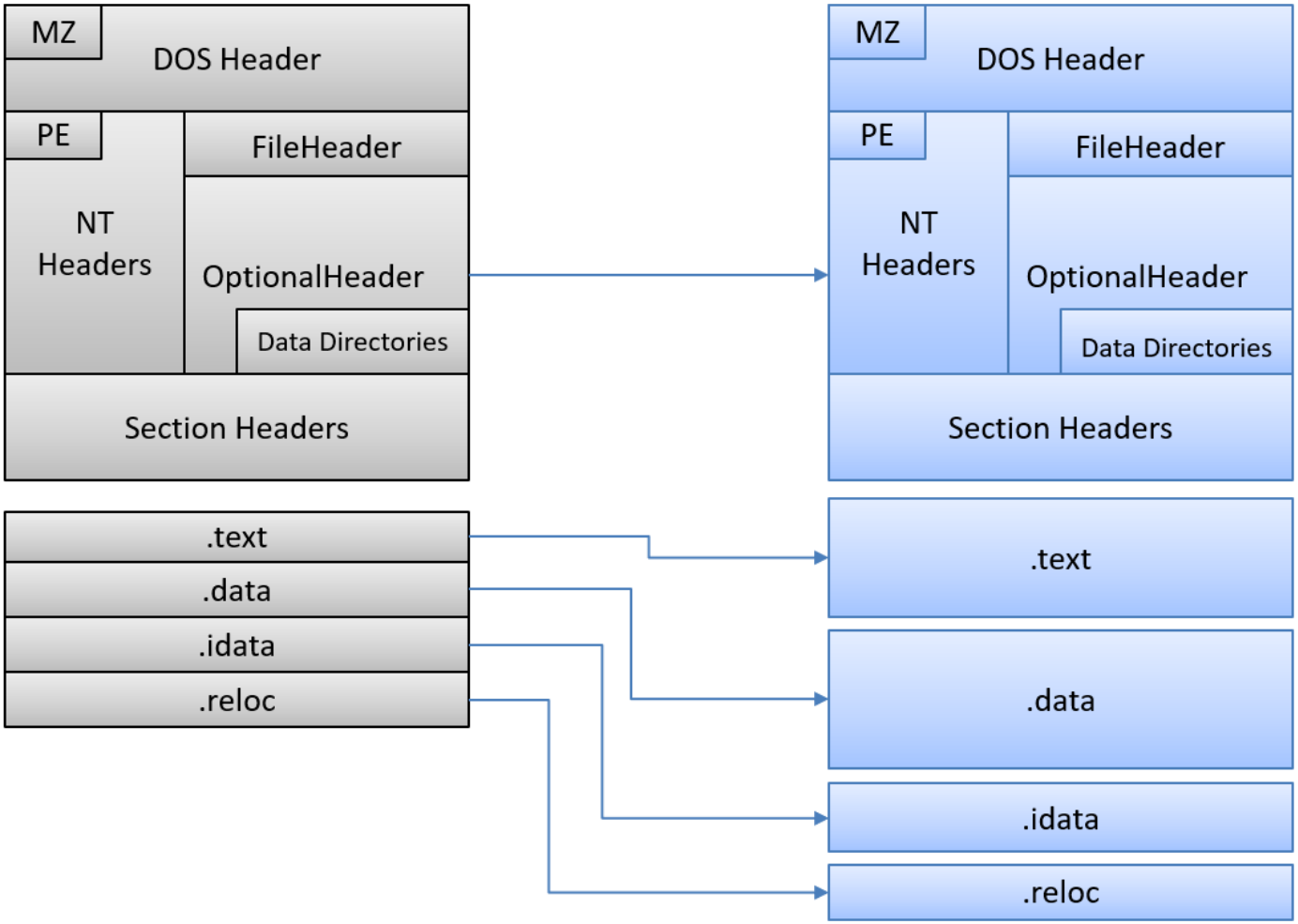
Format PE : Optional Header

Format PE : Data Directories



Format PE : Section Headers

Format PE : Chargement en mémoire



Plan

Fonctions de hachage

Définition

Du point de vue de la rétroconception, une fonction de hachage se caractérise par une ***taille d'entrée quelconque*** *et une* ***sortie de taille fixe****.*

Le calcul d'un hash se fait généralement en 3 temps :

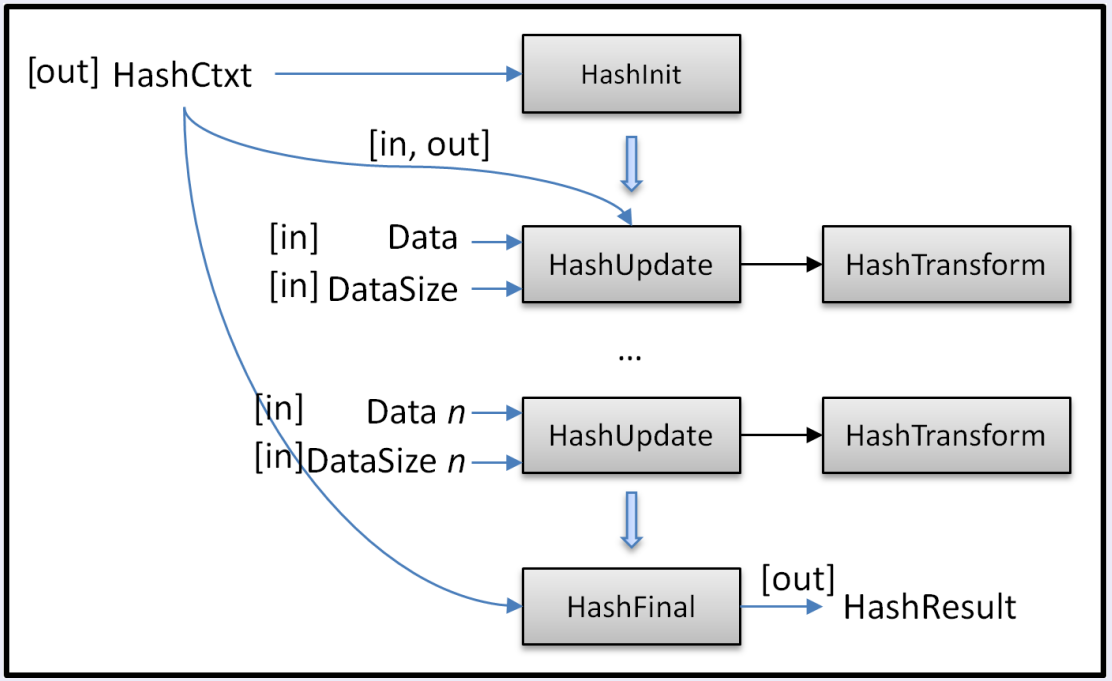
1. une fonction *HashInit* initialise un contexte propre à la fonction de hachage (généralement avec de constante caractéristique) ;
2. une fonction *HashUpdate* met à jour le contexte interne de la fonction de hachage en fonction de la donnée ;
3. une fonction *HashFinal* récupère le hash final à partir du contexte.

Fonctions de hachage

Les prototypes génériques des fonctions de hachage sont les suivants :

```
1. void HashInit(HashContext* pHashCtxt);
2. void HashUpdate(const char* pData, size_t dataSize, HashContext* pHashCtxt);
3. void HashFinal(char hashResult[], HashContext* pHashCtxt);
```

«Pattern» d'une fonction de hachage



Fonctions de hachage

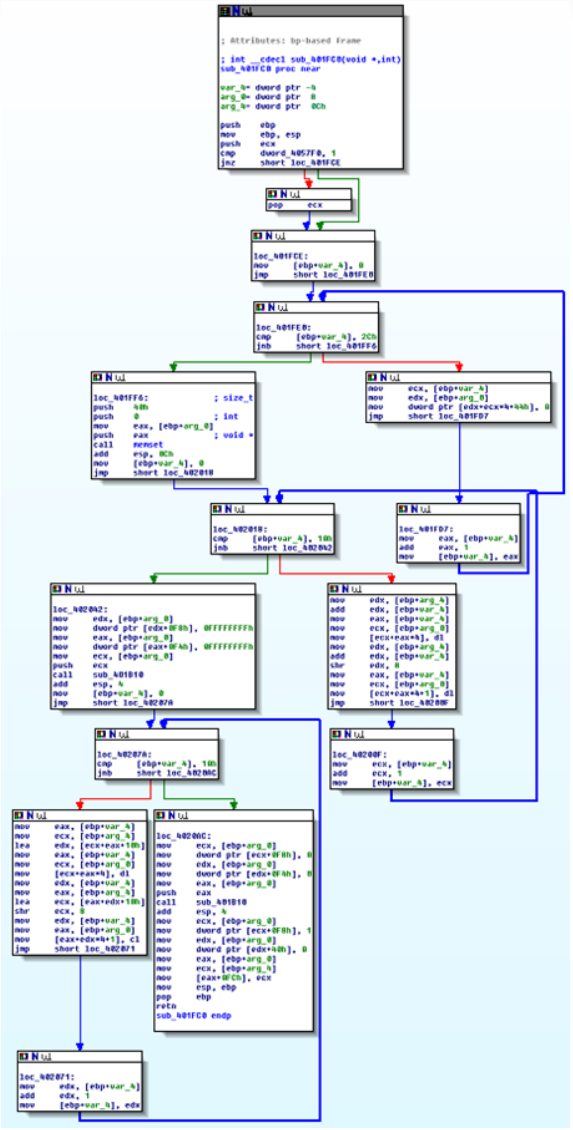
Exemple d'identification de constantes caractéristiques

```
RMD160Init      proc near
arg_0 = dword ptr  4
          mov     eax, [esp+arg_0]
          and     dword ptr [eax+18h], 0
          and     dword ptr [eax+1Ch], 0
          mov     dword ptr [eax], 67452301h
          mov     dword ptr [eax+4], 0EFCDAB89h
          mov     dword ptr [eax+8], 98BADCFEh
          mov     dword ptr [eax+0Ch], 10325476h
          mov     dword ptr [eax+10h], 0C3D2E1F0h
          mov     byte_10005010, 80h
          retn
RMD160Init endp
```

Fonctions de hachage

Attention

Tous les algorithmes de hachage n'ont pas forcément de constantes caractéristiques.
Dans ce cas, il faut revenir au «pattern» des fonctions de hachage.



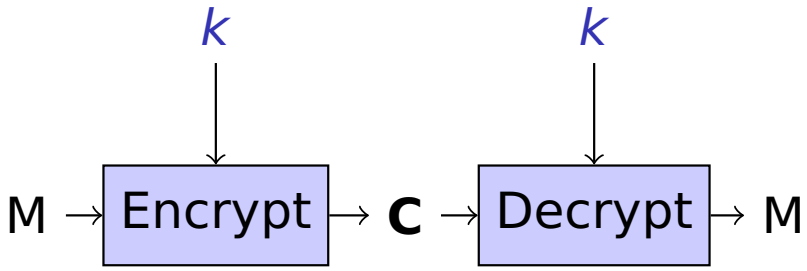
Cryptographie symétrique

Définition

La cryptographie symétrique se caractérise par la notion de clé secrète utilisée à la fois pour le chiffrement et le déchiffrement.

Il existe 2 types chiffrements symétriques :

- le chiffrement par **bloc** ;
- le chiffrement par **flots**.



Chiffrement par bloc

Définition

Du point de vue de la rétroconception, un fonction de chiffrement par bloc transforme un bloc (de taille fixe) de données en clair en un bloc (de même taille) de donnée chiffrées.

Le chiffrement symétrique par bloc se fait en 2 appels :

1. une fonction *InitKey* initialise une clé interne à l'algorithme de chiffrement à partir d'une clé fournie par l'utilisateur (dérivation de clé) ;
2. une fonction *EncryptBloc* permet de chiffrer un bloc d'entrée à partir de la clé interne.

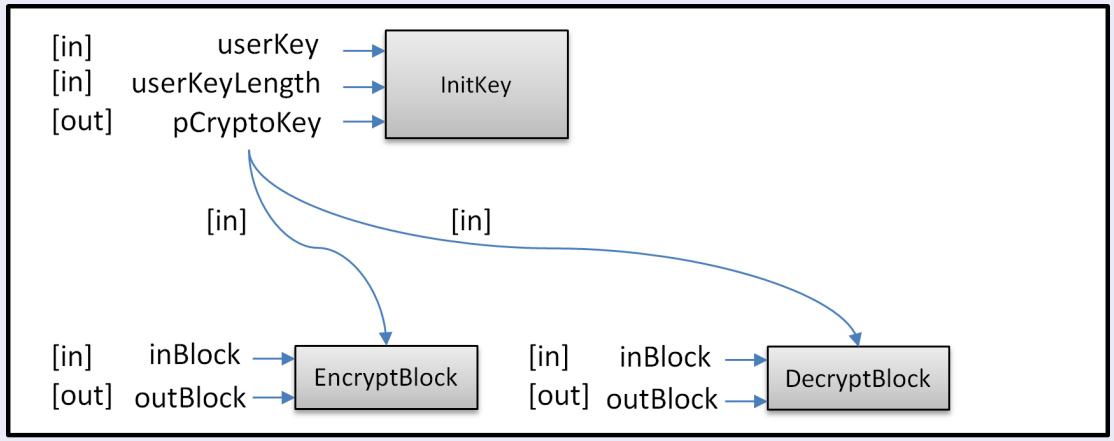
   De la même manière, une fonction *DecryptBloc* permet de déchiffrer un bloc d'entrée à partir de la clé interne.

Chiffrement par bloc

Les prototypes génériques des fonctions de chiffrement par bloc sont les suivants :

```
1. void InitKey(char userKey[], int userKeylen, char* pCryptoKey);
2. void EncryptBloc(const char* inBlock, char* outBlock, char* pCryptoKey);
2. void DecryptBloc(const char* inBlock, char* outBlock, char* pCryptoKey);
```

«Pattern» d'une fonction de chiffrement par bloc



Chiffrement par bloc

Mode d'opération

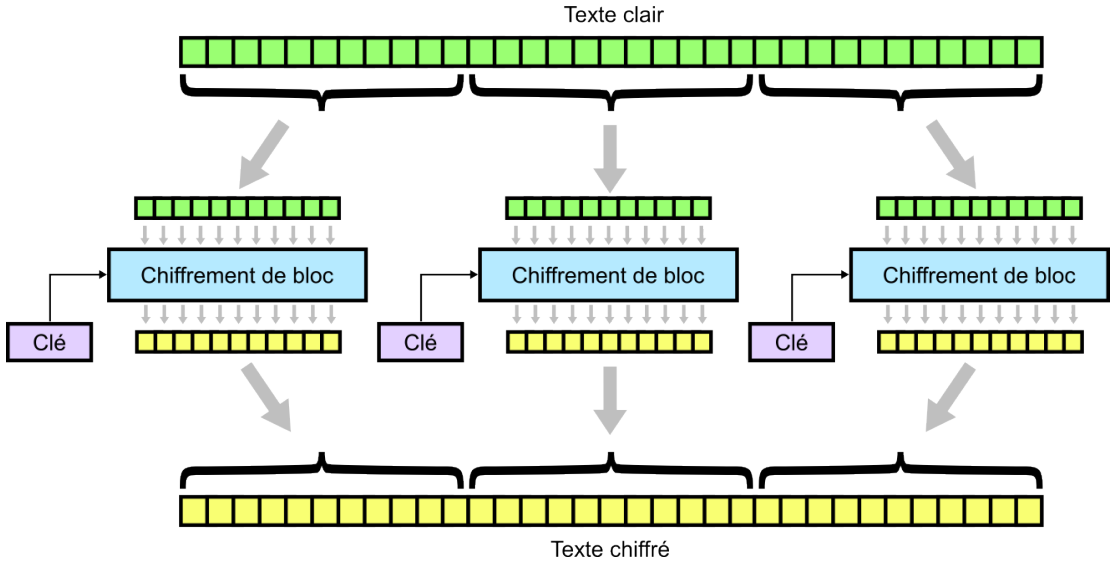
Le mode opératoire désigne une manière de traiter les blocs de données (claires et chiffrées) par un algorithme de chiffrement par bloc.

Les principaux modes d'opération sont :

- Dictionnaire de codes (ECB[1]) ;
- Enchaînement des blocs (CBC[2]) ;
- Chiffrement à rétroaction (CFB[3]) ;
- Chiffrement à rétroaction de sortie (OFB[4]) ;
- Chiffrement basé sur un compteur (CTR[5]) ;
- Chiffrement avec vol de texte (CTS[6]).

1. Electronic Code Book
2. Cipher Block Chaining
3. Cipher Feedback
4. Output Feedback
5. CounTeR
6. CipherText Stealing

Chiffrement par bloc : ECB

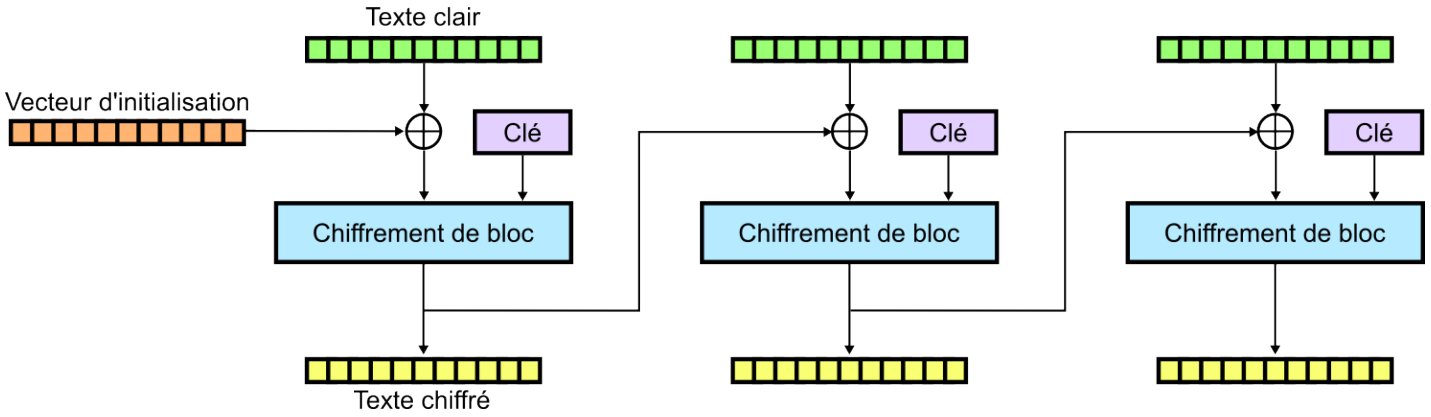


Attention

Ce mode est faible : dictionnaire de code, sensible au rejeu, etc.

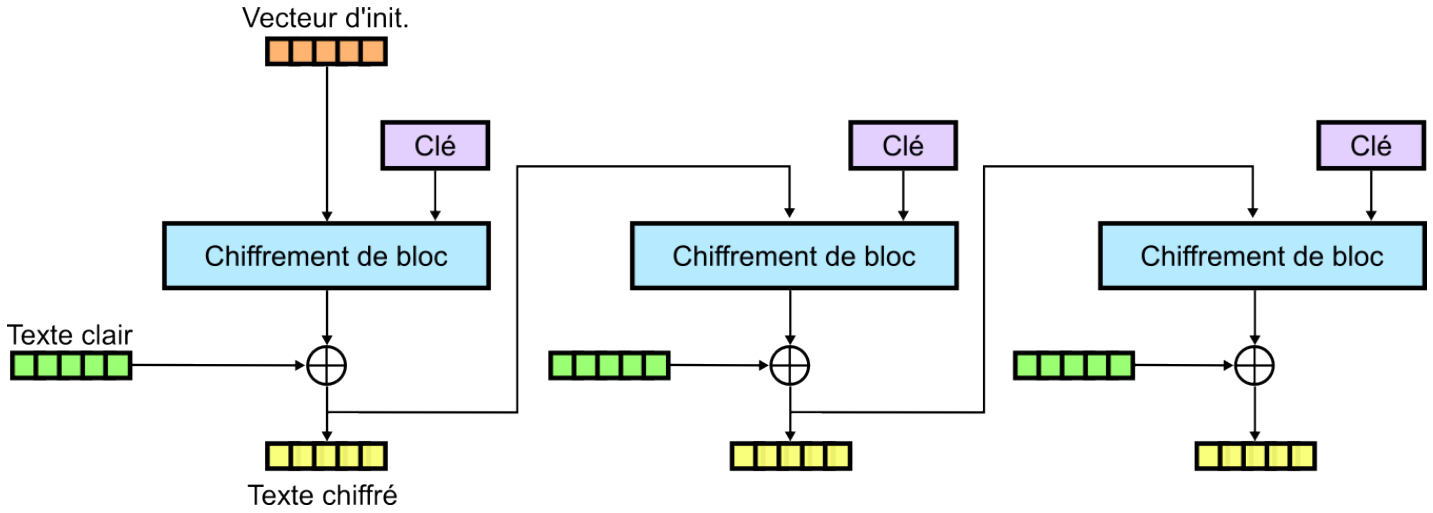
Chiffrement par bloc : CBC

CBC : **Cipher Block Chaining**



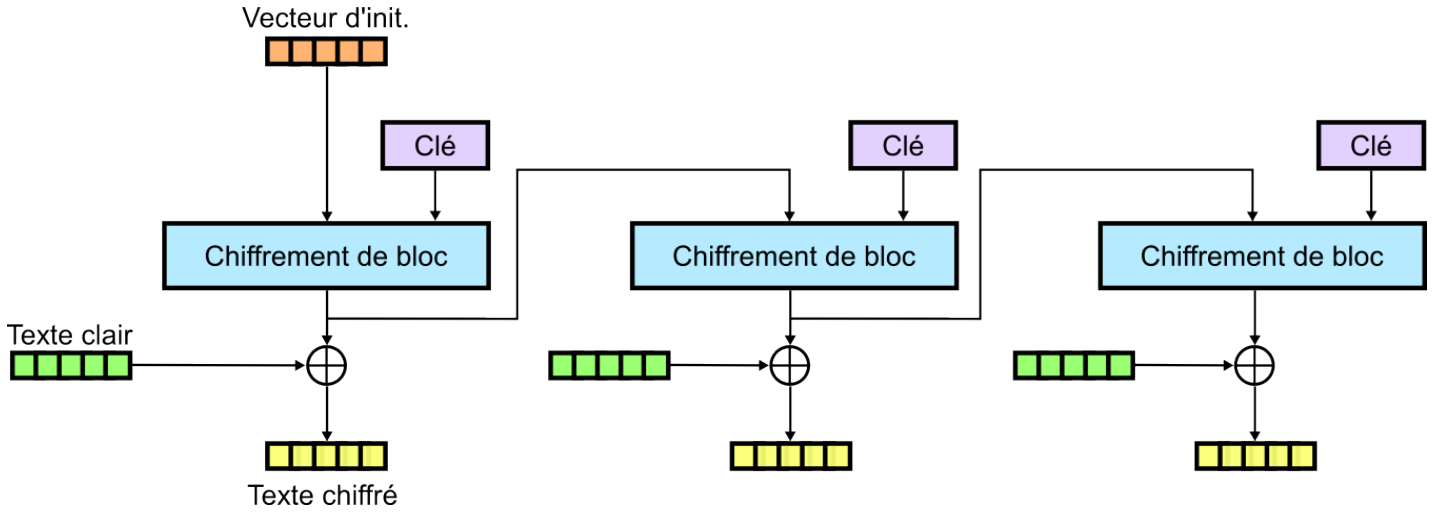
Chiffrement par bloc : CFB

CFB : **Cipher FeedBack**



Chiffrement par bloc : OFB

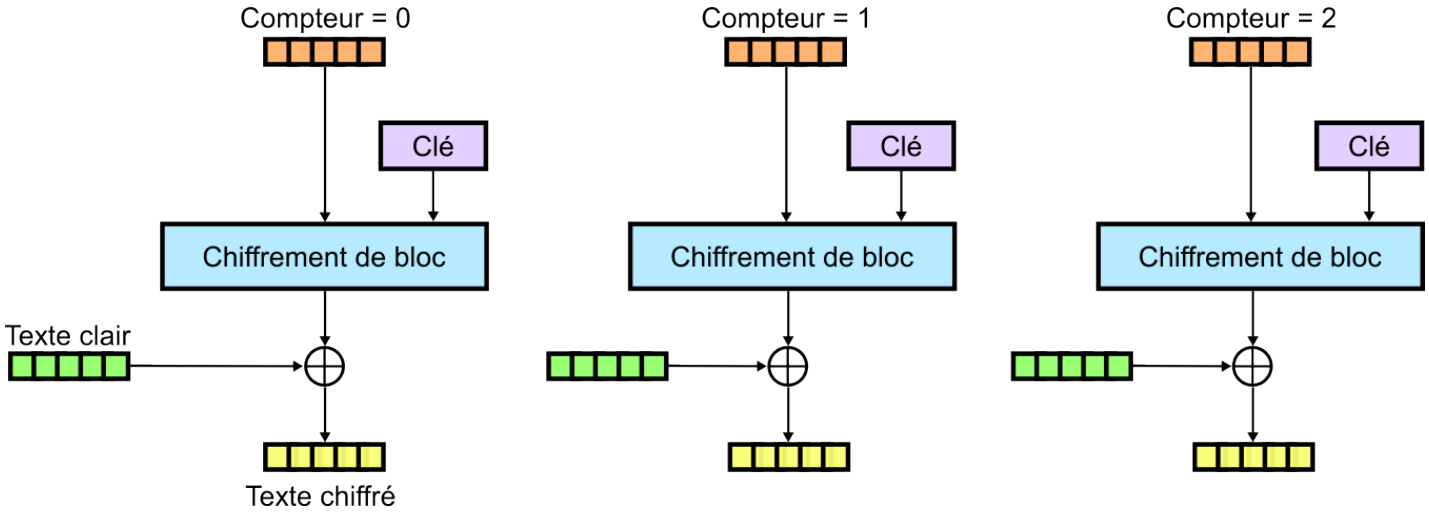
OFB : **Output FeedBack**



Chiffrement par bloc : CTR

CTR : **CounTeR**

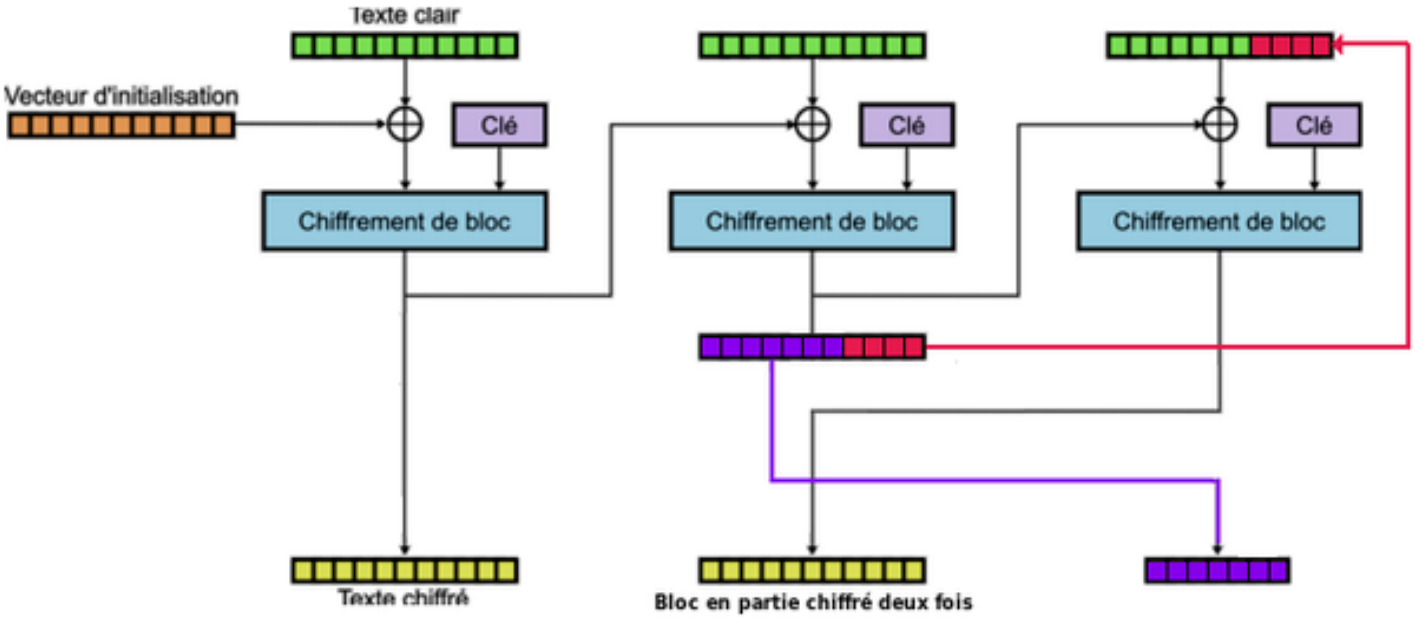
- Parallélisable et précalculable !



Chiffrement par bloc : CTS

CTS : **CipherText Stealing**

- Applicable après un des modes précédents



Chiffrement par flots

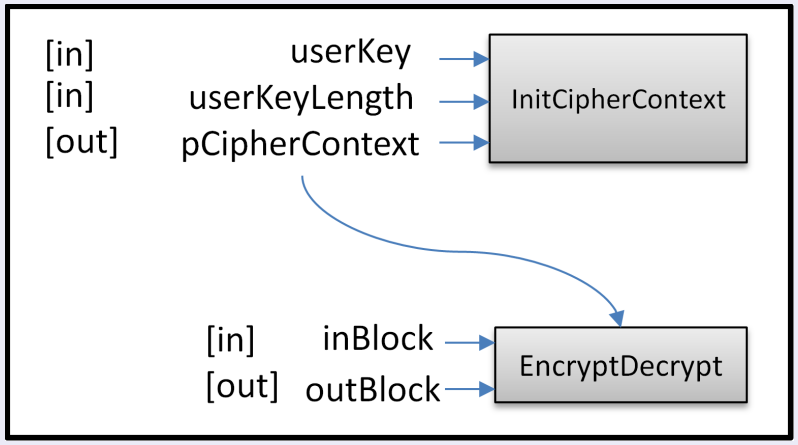
Définition

Du point de vue de la rétroconception, un fonction de chiffrement par flots transforme flux de données en clair en un flux de donnée chiffrées.

Le chiffrement symétrique par flots se fait en 2 appels :

1. une fonction *InitCipherContext* initialise une clé interne à l'algorithme de chiffrement à partir d'une clé fournie par l'utilisateur ;
2. une fonction *EncryptDecrypt* permet de chiffrer ou déchiffrer un flux d'entrée à partir de la clé interne.

«Pattern» d'une fonction de chiffrement par flots



Détection cryptographie symétrique

Outils d'aide à la détection de fonctions cryptographiques :

- Findcrypt2 (plugin IDA et Ghidra)
- Crypto Detector (`https://github.com/Wind-River/crypto-detector`)
- PEiD KANAL
- Kerckhoffs (`https://github.com/felixgr/kerckhoffs`)
- SnD Crypto Scanner (Olly/Immunity Plugin)

Cryptographie asymétrique

La cryptographie asymétrique repose sur l'arithmétique des grands nombres, c'est-à-dire des entiers dont la taille dépasse (et de loin) celle gérée nativement par le CPU.
Du point de vue de la rétro-conception, l'élément de base consiste à identifier une bibliothèque de gestion des grands nombres.

Example

- bignum de OpenSSL
- gmp The GNU Multiple Precision Arithmetic Library

La structure d'un grand nombre est généralement composée d' :

- un entier pour définir la taille
- un pointeur qui contient la valeur
- un signe pour les entiers signés

RSA

Création des clés

1. Choisir p et q, deux nombres premiers distincts ;
2. calculer leur produit $n = pq$, appelé **module de chiffrement** ;
3. calculer $\phi(n) = (p-1)(q-1)$;
4. choisir un entier naturel e premier avec $\phi(n)$ et strictement inférieur à $\phi(n)$, appelé **exposant de chiffrement** ;
5. calculer l'entier naturel d, inverse de e modulo $\phi(n)$, et strictement inférieur à $\phi(n)$, appelé exposant de déchiffrement ;

(n,e)** est la **clé publique

(n,d)** est la **clé privée

Chiffrement

M un message à chiffrer ($M < n$), on calule le chiffré C :

$$C \equiv M^e \pmod{n}$$

Déchiffrement

À partir de C, on calcule le clair M :

$$M \equiv C^d \pmod{n}$$

RSA

L'élément clé pour le RSA est l'exponentiation modulaire sur des grands nombres.

Algorithm 1 Exponentiation rapide

Input: M and $e = (e_t, \ldots, e_1, e_0)_2$.
Output: $M^e \pmod n$

```
1: S ← 1
2: for i from t to 0 do
3:     S ← S × S (mod n)
4:     if e_i == 1 then
5:         S ← S × M (mod n)
6:     end if
7: end for
8: return S
```

Example

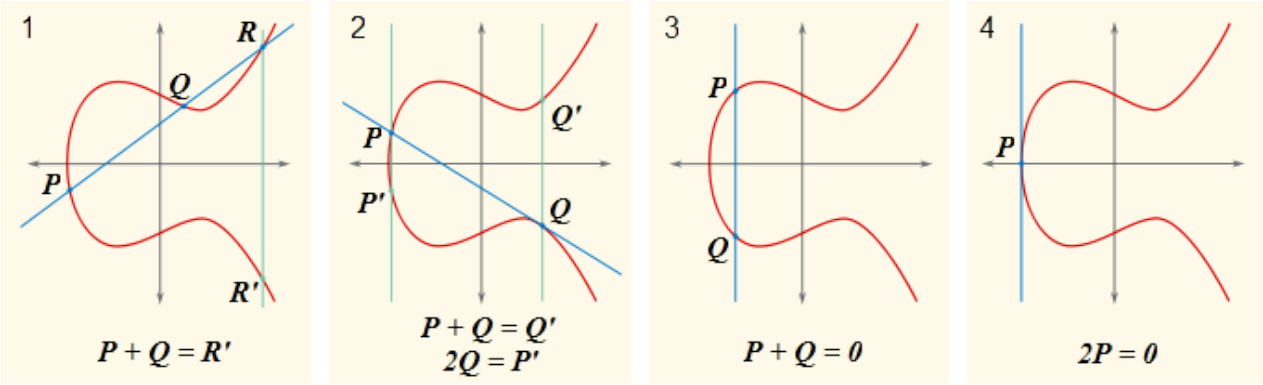
D'autres algorithmes d'exponentiation existent :

- algorithme NAF
- exponentiation de fenêtres fixes
- exponentiation de fenêtres glissantes

Cryptographie sur les courbes elliptiques

Définition

Une courbe elliptique $E(a, b, p)$ est de la forme :
$y^2 = x^3 + ax + b \bmod p$.



Conclusion

- La rétro-conception dans le domaine du logiciel est assez récente dans l'histoire mais s'est beaucoup développée et continue à l'être
- Problème formellement impossible (indécidabilité du désassemblage) et difficile dans la pratique (grande perte d'information durant la compilation)
- Les outils existants sont limités (désassembleurs, débuggeurs et décompilateurs)
- Nécessite une analyse humaine
- Travail long, fastidieux requérant une grande expérience (format de fichier, protocoles, API système, fonctionnement OS, etc)
- Compétences très recherchés dans le domaine mais pas de cursus qui forme à ce métier

Références

Bruce Dang, Alexandre Gazet, and Elias Bachaalany.
Practical Reverse Engineering : X86, X64, ARM, Windows Kernel, Reversing Tools, and Obfuscation.
John Wiley & Sons, 2014.

Chris Eagle.
The IDA pro book : the unofficial guide to the world's most popular disassembler.
No Starch Press, 2011.

Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone.
Handbook of applied cryptography.
CRC press, 1996.

Dennis Yurichev.
Reverse Engineering for Beginners.
2013.