# Virus : Détections Heuristiques en environnement Win32

Nicolas Brulez

(SiliconRealms)

## 1 Introduction

Depuis la première apparition des virus Win32, de nombreuses techniques d'infections ont été inventées. Nombreuses de ses techniques restent utilisées à l'heure actuelle, il est donc important de bien comprendre leur principe de fonctionnement.

Tout comme il existe des detections heuristiques pour les virus DOS,de Macros, VBS etc, il existe aussi des detections heuristiques pour les virus Win32. Les méthodes les plus classiques d'infections sont présentées dans ce papier, ainsi que leurs détections heuristiques.

Les techniques anti heuristiques utilisées par les virus modernes seront aussi présentées.

Finalement, je présenterai mon propre moteur de detection Heuristique et les resultats obtenus par celui ci.

## 2 Les différents types de virus

Je présenterai ici seulement les virus qui infectent véritablement leur cible. Seront donc écartés, les virus compagnons, ainsi que les virus Prependers, qui écrasent le fichier hote. Il ne s'agit pas de vrais infections, et n'ont donc par conséquent, que très peu d'interet.

### 2.1 Les virus : Appenders

En général, ce type de virus ajoute son code à la fin du programme infecté. Lors de l'exécution d'un fichier infecté,le virus s'exécute en premier, pour infecter de nouveaux fichiers, puis rend la main au fichier hote qui s'exécute normalement. Ce type de virus est completement invisible à l'utilisateur dans la majorité des cas.

**Virus Cryptés** Comme leur nom l'indique, ces virus voient leur corps cryptés. Il existe deux types de virus cryptés. Le premier type de virus cryptés est composé d'un algorithme de chiffrement et d'une clé unique. Le virus sera toujours chiffré de la même manière. Le seul intêret de cette technique est d'empêcher l'analyse directe du virus.

L'autre type de virus cryptés utilise une clé différente pour chaque fichier infecté. Cette technique est apparue il y a maintenant de longues années, pour contourner les Anti Virus. Cependant, l'algorithme de dechiffrement restant toujours le même, il est très simple de détecter ces virus.

**Oligomorphiques** Contrairement aux virus cryptés, les virus Oligomorphiques contiennent plusieurs variations du décrypteur. Ces virus combinent chaque parties différement afin de créer un nouvel algorithme. En modifiant l'odre de ses blocks, il est possible d'obtenir une centaine de variations, ce qui implique une plus grande difficulté de détection de ces virus. Il faut autant de signatures qu'il y a de variations possibles pour détecter le virus à 100% (ou alors avoir recours à l'Emulation).

**Polymorphiques** Contrairement aux virus cryptés, les virus polymorphiques génèrent un code de decryptage différent, et utilisent des clés différentes. Cela leur permet d'echapper aux détections par signatures. Il existe plusieurs types de virus polymorphes, les polymorphes lents, et rapides.

Un virus polymorphique lent, ne changera de décrypteur qu'une fois par jour par exemple, ou à chaque redemarrage du pc. La récuperation de variantes différentes prends donc du temps. Le virus polymorphique rapide quant à lui, change de décrypteurs à chaque infection.

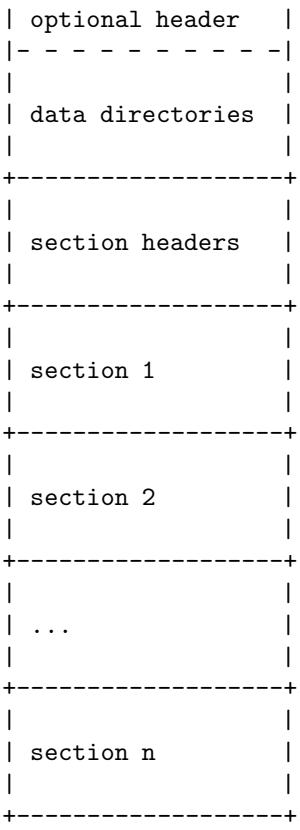
Le corps des virus reste inchangé, ce qui permet une detection du virus, une fois le virus décrypté. Les Anti virus utilisent en général leur émulateur pour décrypter le corps du virus, puis scan l'intérieur du virus decrypté, à la recherche d'une signature dans le corps en clair.

**Métamorphiques** Le métamorphisme peut être présenté comme un "polymorphisme du virus entier". Contrairement au polymorphisme classique, où seul le decrypteur est modifié, les virus metamorphiques ont une mutation de leur corps, ainsi que du decrypteur. Il est beaucoup plus compliqués de détecter toutes les variations d'un tel virus, car il n'a en théorie aucune partie constante. Les detections par statistiques peuvent être utilisées pour detecter ce genre de virus.

### 2.2 Présentation du format PE

Le format PE est le format des exécutables windows 32 bit. Je vais présenter rapidement son architecture, ses structures afin d'aider à la compréhension de cette présentation. Pour plus d'information, lire une documentation plus complète sur le format PE. Un fichier peut être représenté sous cette forme :

```
+--------------------+
| MZ Header      |
+--------------------+
| PE file-header    |
+--------------------+
```




**Le PE Header** Le PE Header débute en 0x00 par un MZ Header pour assurer une compatibilité MS-DOS. En effet, les exécutables win32 affichent un message d'erreur lorsqu'ils sont exécutés sous DOS : This programm cannot be run in DOS mode.

```
00000000 4D5A 9000 0300 0000 0400 0000 FFFF 0000 MZ..............
00000010 B800 0000 0000 0000 4000 0000 0000 0000 ........@.......
00000020 0000 0000 0000 0000 0000 0000 0000 0000 ................
00000030 0000 0000 0000 0000 0000 0000 8000 0000 ................
00000040 0E1F BA0E 00B4 09CD 21B8 014C CD21 5468 ........!..L.!Th
00000050 6973 2070 726F 6772 616D 2063 616E 6E6F is program canno
00000060 7420 6265 2072 756E 2069 6E20 444F 5320 t be run in DOS
00000070 6D6F 6465 2E0D 0D0A 2400 0000 0000 0000 mode....$.......
```

On trouve aussi l'adresse du PE file header dans le MZ Header. Celle-ci est placée dans le champ e_lfanew disponible en +3Ch.

**Le PE file header**

```
00000080 5045 0000 4C01 0600 8D54 F32F 0000 0000 PE..L....T./....
00000090 0000 0000 E000 0E01 ........
```

La structure du PE file header est :

```
PE File Header:
+0 DWORD PE\x00\x00
4 WORD Machine;
6 WORD NumberOfSections;
8 DWORD TimeDateStamp;
C DWORD PointerToSymbolTable;
10 DWORD NumberOfSymbols;
14 WORD SizeOfOptionalHeader;
16 WORD Characteristics;
```

### Le PE Optional Header

```
18 WORD Magic;
1a UCHAR MajorLinkerVersion;
1b UCHAR MinorLinkerVersion;
1c DWORD SizeOfCode;
20 DWORD SizeOfInitializedData;
24 DWORD SizeOfUninitializedData;
28 DWORD AddressOfEntryPoint;
2c DWORD BaseOfCode;
30 DWORD BaseOfData;
34 DWORD ImageBase;
38 DWORD SectionAlignment;
3c DWORD FileAlignment;
40 WORD MajorOperatingSystemVersion;
42 WORD MinorOperatingSystemVersion;
44 WORD MajorImageVersion;
46 WORD MinorImageVersion;
48 WORD MajorSubsystemVersion;
4a WORD MinorSubsystemVersion;
4c DWORD Reserved1;
50 DWORD SizeOfImage;
54 DWORD SizeOfHeaders;
58 DWORD CheckSum;
5c WORD Subsystem;
5e WORD DllCharacteristics;
60 DWORD SizeOfStackReserve;
64 DWORD SizeOfStackCommit;
68 DWORD SizeOfHeapReserve;
6c DWORD SizeOfHeapCommit;
70 DWORD LoaderFlags;
74 DWORD NumberOfRvaAndSizes;
```

Cette structure contient des informations très importantes telles que la RVA2 du point d'entrée. Le point d'entrée est l'adresse de la première instruction à exécuter. Nous verrons plus tard qu'elle a une importance capitale dans l'étude des protections PE.

Toutes les adresses contenues dans le PE header sont données sous la forme d'adresses relatives à l'image base (champ ImageBase). Pour obtenir l'adresse en mémoire, il faut ajouter ces RVA à l'image base d'un programme. Pour obtenir l'image base d'un programme, il suffit de lire la valeur contenue dans la structure, en +34h. Il s'agit de l'adresse à laquelle sera chargé l'exécutable en mémoire. En général, celle-ci vaut 0x400000.

**Le DataDirectory** Il s'agit d'un tableau de structures contenant des informations diverses tel que l'import table, l'export table, etc.

**Les Section Headers** Ils sont accessibles à l'adresse PE optional header + F8h. Chaque section de l'exécutable est décrite par la structure suivante :

```
+0 8byte ANSI name (Nom de la section)
+8 dword misc (taille de la section en mémoire)
+C dword virtual address (RVA de la section)
10 dword sizeofrawdata (Taille de la section sur le disque)
14 dword pointerToRawData (Offset de la section)
18 dword pointerToRelocations
1C dword PointerToLinenumbers
20 word NumberOfRelocations
22 word NumberOfLineNumbers
24 dword Characteristics (Caractéristiques de la section)
```

### 2.3 Principes du code relogeable

Un virus doit, pour fonctionner et infecter des programmes hôtes différents, pouvoir être lancé à n'importe quelle adresse mémoire, et être toujours fonctionnel. Pour se faire, les virus utilisent du code relogeable. Ce principe de code relogeable existait déja sous DOS.

Le principe est très simple, et consiste à utiliser un offset de référence pour accéder aux variables du virus.

Les données du virus se trouvent toujours à la même distance de l'offset de référence quelque soit l'adresse ou est chargé le virus, ce qui permet de retrouver les infos nécéssaires au bon fonctionnement du parasite. L'offset de référence est en général appelé "delta offset".

```
Exemple:
--------

virus_start:
```

```
call get_delta

get_delta:

pop ebp
sub ebp,offset get_delta
```

A partir de la, EBP contient un offset permettant d'accéder aux données du virus.

*Explications.*- L'instruction call pousse sur la pile l'adresse de retour, c'est à dire l'adresse qui précède le call. Les virus se servent de cette caractéristique pour récupérer l'adresse à laquelle ils ont été chargé grace à l'utilisation de l'instruction pop, qui va lire l'adresse de retour sur la pile.

Le virus soustrait ensuite à l'adresse ou il est chargé, l'offset du delta de la première génération. On obtient ainsi un delta offset pour accéder à nos données.

Dans la seconde génération du virus, c'est à dire une fois qu'un fichier est infecté, le virus soustrait à l'adresse poussée sur la pile, l'anciene adresse du pop, ce qui a pour effet de nous donner le déplacement exacte pour accéder à nos données.

Il suffit alors d'utiliser le registre "ebp" pour référencer nos données :

```
mov eax, dword ptr [ebp+kernel]
```

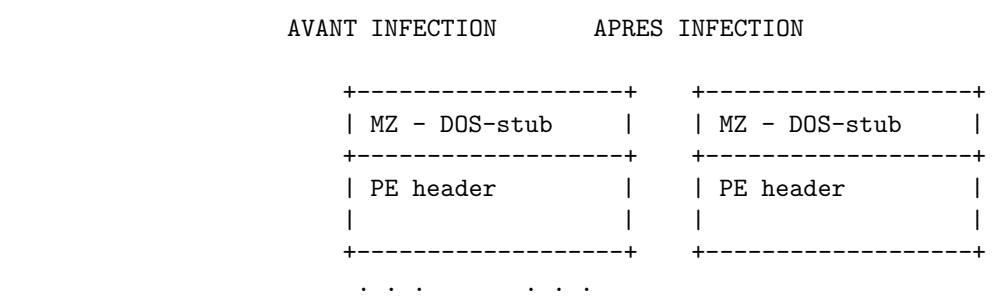
"kernel" est représenté par une adresse une fois compilé. La valeur dans le registre EBP (delta offset) permet d'ajuster l'adresse, et donc de retrouver les données du virus.

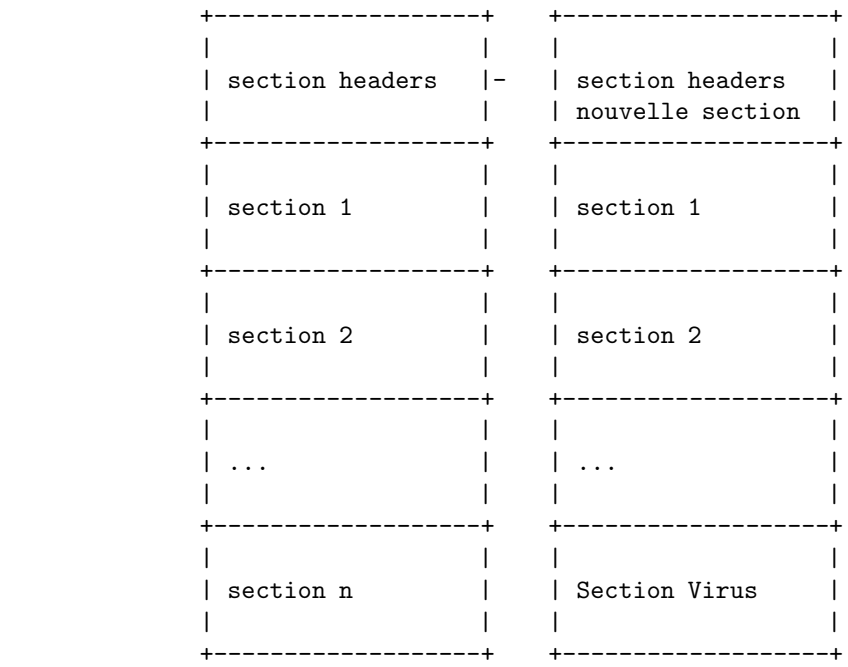
## 3 Présentation de quelques types d'infections Win32

### 3.1 Emplacement du virus

**Dernière section** La plus part des virus se copient à la fin de leur fichier hote, plus précisement dans la dernière section des exécutables PE. Il existe deux types d'infections de ce type.

– **Ajout de Section**.- Les premiers virus ajoutaient une nouvelle section aux fichiers infectés. Le premier virus pour Win95, Boza ajoutait une section du nom de ".vlad".





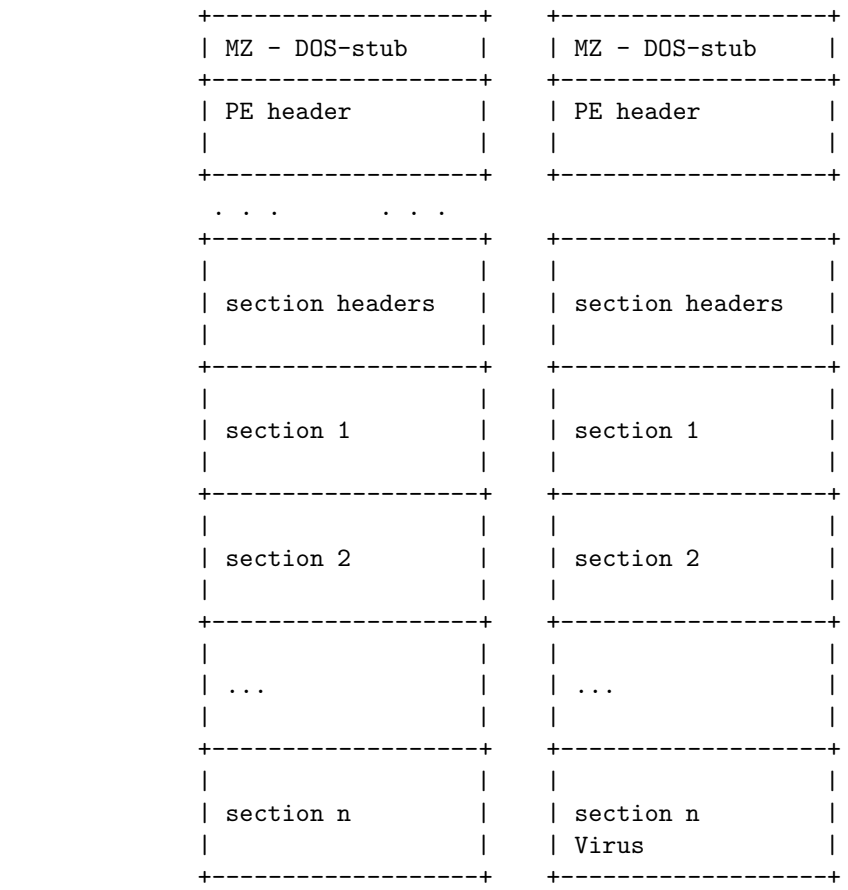

Ces virus modifient le PE Header afin d'ajouter une nouvelle section.

```
.qjxh:004106F6              public start
.qjxh:004106F6 start        proc near
.qjxh:004106F6
.qjxh:004106F6 arg_4        = dword ptr  8
.qjxh:004106F6 arg_8        = dword ptr  0Ch
.qjxh:004106F6
.qjxh:004106F6              jmp     $+5
.qjxh:004106FB
.qjxh:004106FB loc_4106FB:  ; CODE XREF: .qjxh:00407127j
.qjxh:004106FB              mov     ecx, 10000h
.qjxh:00410700              mov     ebp, offset unk_404200

etc
```

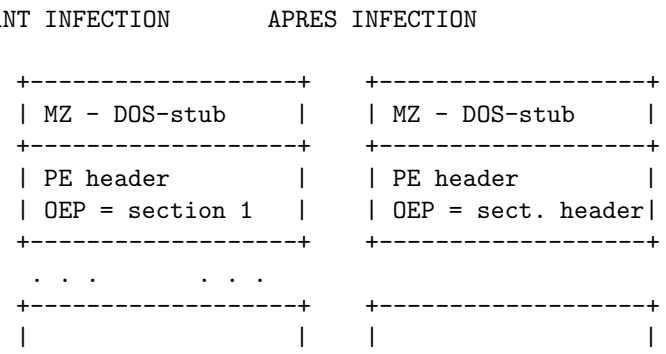
– **Agrandissement de la dernière section**.- Par la suite, de nouveaux virus sont apparus, ceux ci n'ajoutaient pas de nouvelle section, mais agrandissaient la dernière section pour s'y loger. Cette technique a plusieurs avantages.
Elle est plus facile à mettre en oeuvre : pas de grosse modification du PE header. Elle est aussi plus furtive. L'infection n'apparait pas d'un simple coup d'oeil a la vue du nom des sections.

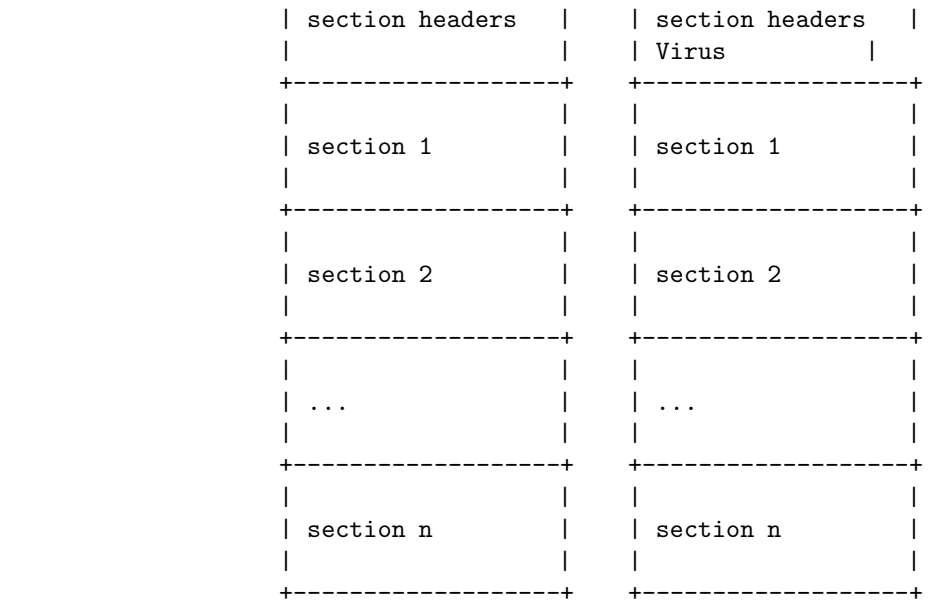
```
AVANT INFECTION       APRES INFECTION
```




Exemple de virus : Anxiety.

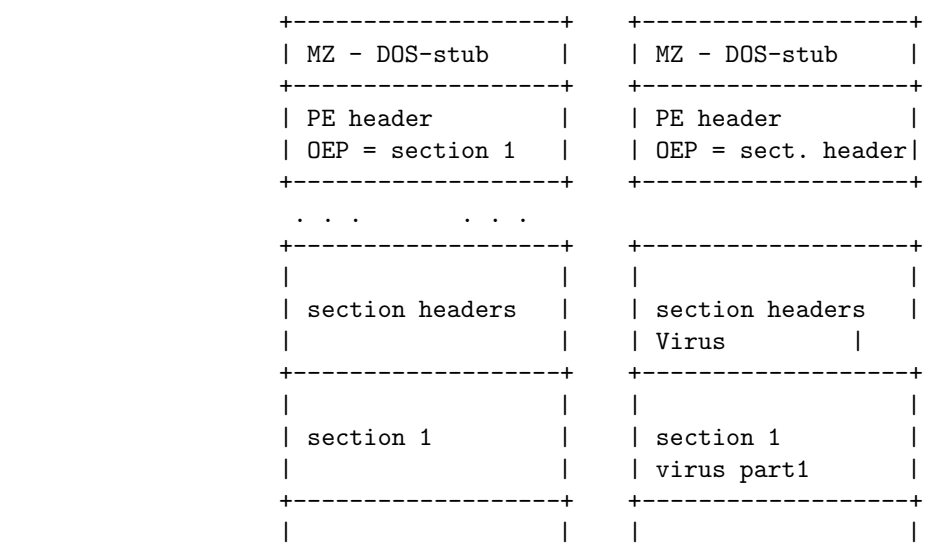
**Infection du Header** Sous windows 95, certains virus se copiaient juste après le section header, et juste avant la première section. Meme si le code du virus n'était dans aucune section, le loader de windows autorisait l'exécution du virus.

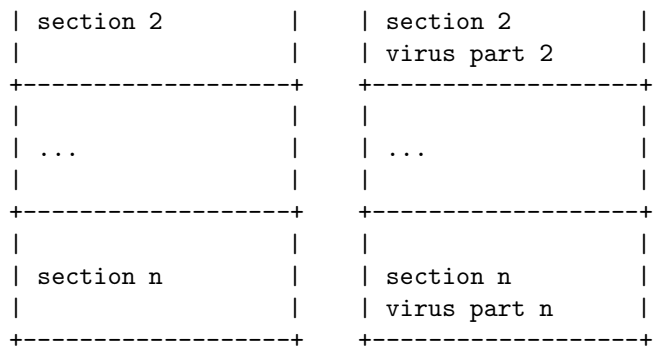





Exemple de virus : Murkry

**Cavité** Les infecteurs par cavités utilisent le padding des sections pour se loger. Le corps du virus est dissimulé en plusieurs parties dans le padding utilisé pour aligner les sections, et permet d'infecter un fichier sans modifier la taille de celui ci.



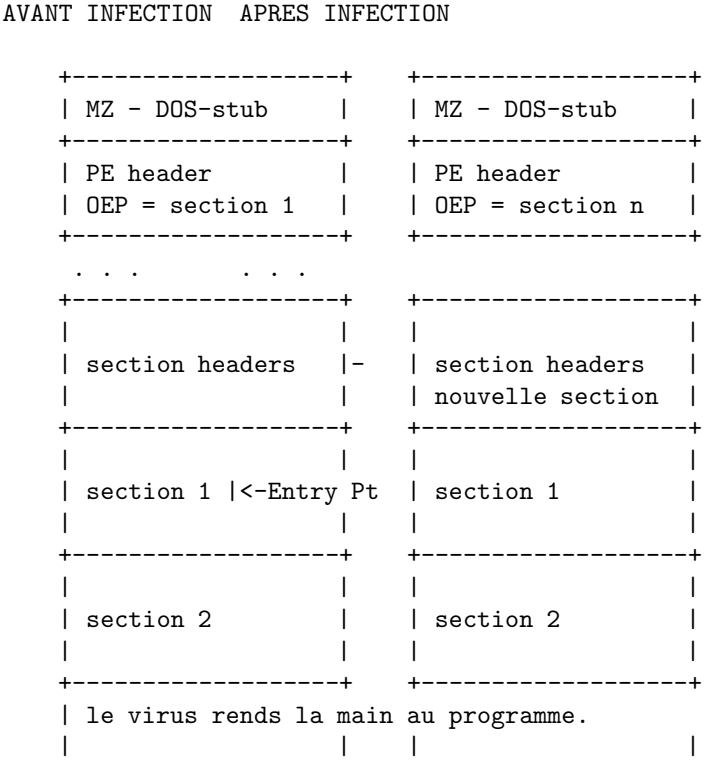


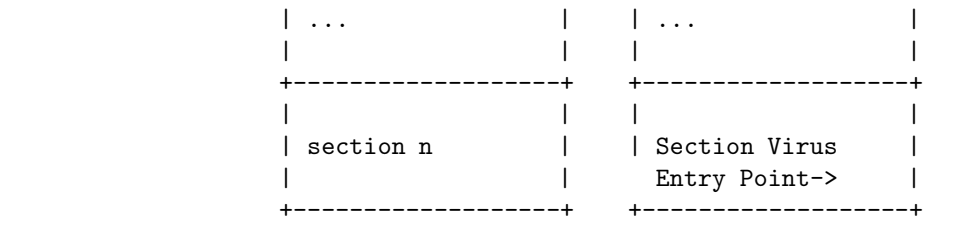

Exemple de virus : CIH

### 3.2 Point d'entrée

Les virus pour s'activer, doivent modifier le programme afin de prendre la main avant le programme hote. Différentes techniques existent, voici les principales :

#### Dans la dernière section



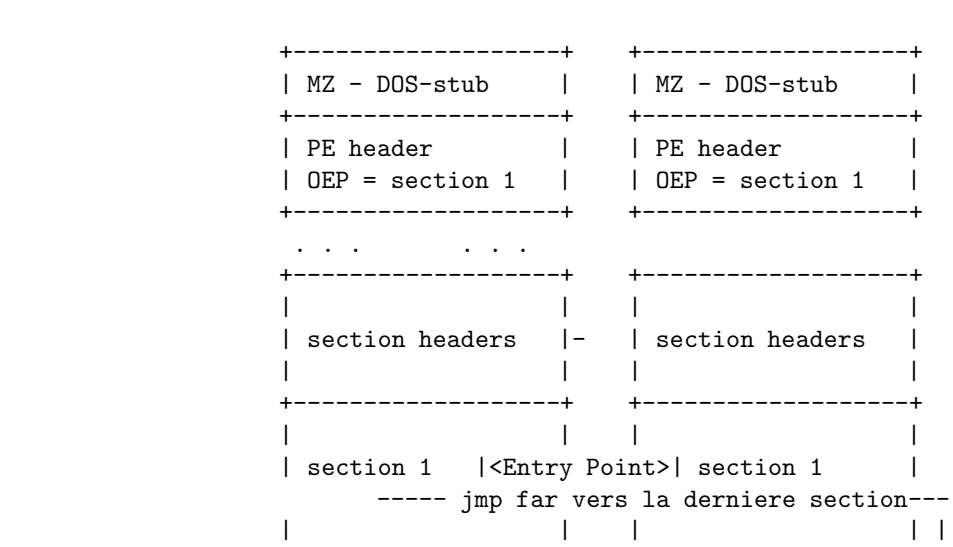


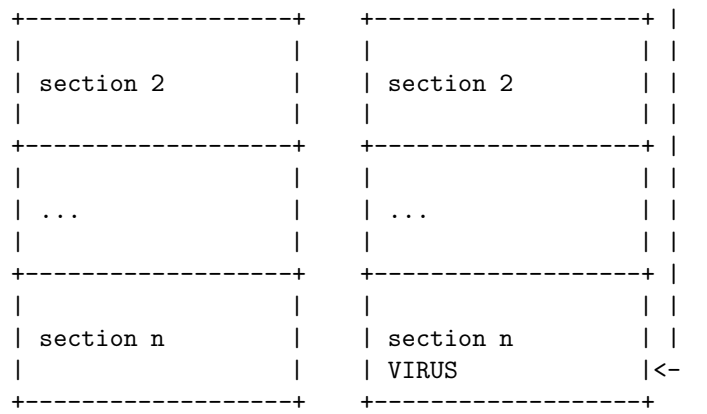

Le point d'entrée d'un fichier infecté pointe dans la derniere section. Le programme commence directement par le code du virus, qui après s'être reproduit, rends la main au fichier hote.

```
.reloc:00409FF4         public start
.reloc:00409FF4 start   proc near
.reloc:00409FF4         and     bl, 99h
.reloc:00409FF7         mov     edx, 1BB51A23h
.reloc:00409FFC         ror     edx, 0BCh
.reloc:00409FFF         movsx   ebp, ah
.reloc:0040A002         mov     ebp, 0EC58F5B4h
.reloc:0040A007         rol     ebp, 2Fh
.reloc:0040A00A         mov     ecx, ebp
etc
```

(Extrait du virus Crypto qui commence dans la dernière section. Ici section .reloc.)

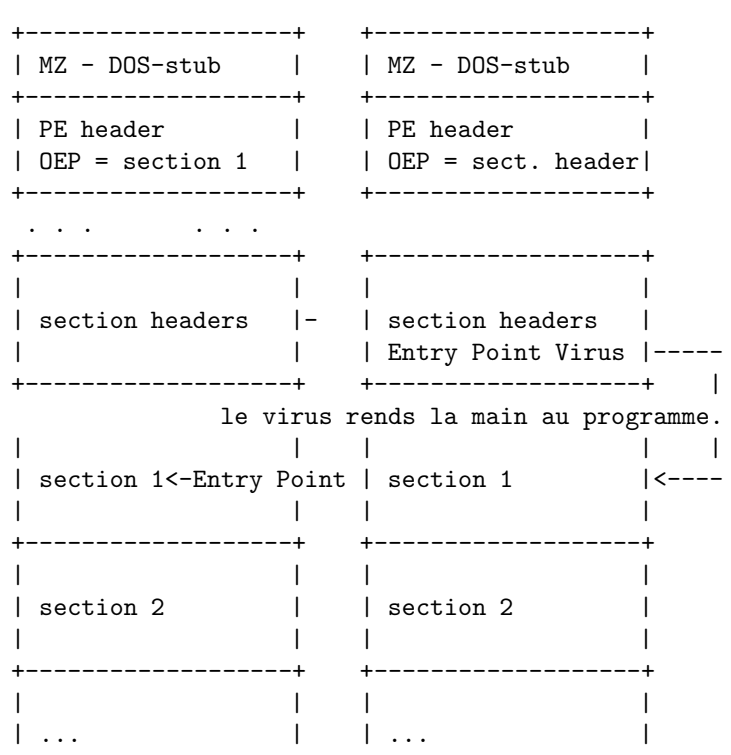
### Dans la première section

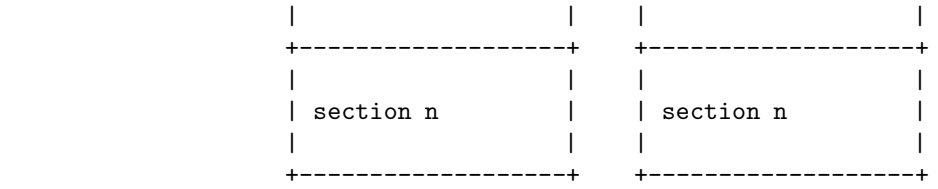





Une technique utilisée certaine fois, est l'insertion d'un saut vers le code du virus à l'entry point du programme hote afin de rediriger le fichier infecté sur le code du virus, d'une magnière plus furtive. Exemple de virus : Marburg.

**Avant la première section** L'entry point commence avant la premiere section :






Le virus au moment de l'infection modifie le PE header pour qu'il pointe non pas vers le debut du programme original, mais apres la section header. Cette section n'a pas de caracteristiques, mais microsoft a cru bon de la laisser exécutable, donc exploitable par un virus. Le virus se copie donc apres la table des sections et avant la premiere section.

IDA nous avertit d'ailleurs lorsqu'un fichier commence avant la premiere section.

```
HEADER:00400400 ;
HEADER:00400400 ; The code at 400000..401000
            is hidden from normal disassembly
HEADER:00400400 ; and was loaded because the
            user ordered to load it explicitly
HEADER:00400400 ;
HEADER:00400400 ;  <<<< IT MAY CONTAIN TROJAN
    HORSES, VIRUSES, AND DO HARMFUL THINGS >>>
HEADER:00400400 ;
HEADER:00400400 ;
HEADER:00400400
HEADER:00400400 ;
------- S U B R O U T I N E -----------------
HEADER:00400400
HEADER:00400400
HEADER:00400400             public start
HEADER:00400400 start       proc near
HEADER:00400400             xchg    eax, esi
HEADER:00400401             push    edi
HEADER:00400402             sidt    qword ptr [esp-2]
HEADER:00400407             pop     edi
HEADER:00400408             fild    qword ptr [edi]
....
```

### 3.3 Infection par modification de e_lfanew

L'infection par modification d'elfanew est la méthode d'infection la plus simple à mettre en oeuvre. En effet, ce type d'infection ne nécessite pas de code relogeable. Le virus se copie entiérement à la fin du fichier. Il s'agit d'un exécutable standard, avec ses propres imports. Pour infecter un fichier, le virus modifie le pointeur vers le PE header (elfanew), et le fait pointer sur le PE

Header du virus. Le loader de Windows ne vas donc voir que le code du virus et l'exécuter. Celui ci après avoir infecté d'autres fichiers, effectue une copie du fichier infecté et re-modifie elfanew pour pointer ver le programme hote. Le virus exécute ensuite le fichier "temporairement" désinfecté et attends que celui lui rende la main, puis efface le fichier temporaire, et le virus se ferme.

# 4 Les detections heuristiques Win32

## 4.1 Analyse de la structure PE

Les techniques de detections heuristiques en environnement win32 sont basées en grande partie sur l'analyse de la structure des exécutables PE. D'une manière générale, les fichiers sont compilés selon un même modèle.

**Point d'entrée dans la dernière section** Une fois compilés, les programmes ont pour point d'entrée la première section, soit la section code. Les premiers virus, ou les virus simples, modifient le point d'entrée, et le font pointer dans la dernière section, la ou le virus s'est copié. Un programme par defaut, n'aura jamais un point d'entrée pointant ailleurs que dans la première section.

**Point d'entrée avant la première section** Comme vu précédement, Le point d'entrée d'une application standard pointe dans la première section, ou du moins en aucun cas avant la première section. Il s'agit d'une caractèristique très suspecte et seulement utilisée par les virus.

**Caractéristiques des sections** Chaque section a des caractéristiques données, et en régle général, toutes caractéristiques différentes pour un type de section donnée doit être considéré comme suspect.

- **Dernière section exécutable**.- En régle général, seul la première section est exécutable, la dernière section est généralement utilisée pour les relocations, le debug, ou les ressources. Celles ci ne sont pas exécutables. Ce type de caractéristique est donc suspect. Cependant, les packers d'exécutables, utilisent eux aussi régulierement, la dernière section, pour contenir le code de décompression des fichiers packés. Il y a une risque de faux positifs.
- **Première section "writeable"**.- La première section a généralement, les attributs d'exécution et de lecture, mais pas d'écriture. Les virus modifient les caractéristiques de la première section en "writeable", lorsque ceux ci ont cryptés le code du fichier hote, ou alors quand le virus se copie dans la premiere section et qu'il doit mettre à jours ses propres variables, il a besoin des droits en écriture.

**Nom des sections et leurs caractéristiques** Certains virus renoment la derniere section avant de changer ses caractéristiques pour être moins visibles. Cependant certains ne renoment pas les sections, et se contentent de changer

leurs attributs. Une section ".reloc" ou ".rsrc" par exemple n'aura jamaisle flag "executable", et il s'agit très probablement d'un signe d'infection. Cette caractéristique est donc TRES suspecte.

**La "Virtual Size" est incorrect dans le PE header** La plus part des virus pour windows 9x n'alignent pas "SizeOfImage" avec l'alignement de section le plus proche. Sous les anciens windows, le loader n'y prete pas attention, et les fichiers s'exécutent, contrairement à Windows NT qui refuse d'exécuter l'application si l'alignement est incorrecte. Cela permet de detecter les anciens virus.

**PE header en fin de fichier** Dans le cas des virus qui infecent par modification de elfanew, le PE header se retrouve en fin de fichier. Le PE header n'est jamais au dela d'une centaine d'octets dans un fichier normal, il est donc possible de detecter ce type d'infection et de noter l'adresse très suspecte du PE header.

**Size Of Code incorrect** Beaucoup de virus ne mettent pas à jours le champ "SizeOfCode" lorsqu'ils ajoutent une section avec l'attribut exécutable activée. Si la somme des tailles des sections code est supérieure a la taille définie dans le champ "SizeOfCode", il est fort probable qu'une section ait été ajoutée, probablement par un virus. Suspect.

### 4.2 Analyse du code

Après l'analyse structurale d'un fichier PE, il est possible d'effectuer une analyse heuristique à partir du code de l'application.

**Instruction non standard au point d'entrée.** La majorité des applications compilés commencent par un stack frame, ou certaines instructions spécifiques, utilisées pour chaque fichier compilé par un compilateur donné. Si un fichier ne commence pas par ses instructions standards, le fichier peut avoir été infecté par un virus, peut etre packé, ou programmé en assembleur. Il n'est pas inutile de prendre en compte les instructions au point d'entrée du fichier, cela peut apporter une information supplémentaire si d'autres caractèristiques suspectes ont été identifiées dans un fichier. Moyenement suspect.

**Calcul du delta offset** Comme vu en introduction, les virus ont besoin d'avoir un code relogeable pour pouvoir fonctionner. Le code utilisé est généralement un "call instruction suivante", et sa représentation hexadécimale est la suivante : 0xE800000000. Certains virus utilisent des variations, il est donc possible d'utiliser des régles un peu moins restrictives pour eviter d'être trompé par du code comme celui ci :

```
call get_delta
db 0EBh ; octet inutilisé mais permettant
                ; de tromper une detection trop
                ; simpliste d'un delta offset.
get_delta:
pop ebp
```

**Redirection de code Suspect** Les virus ne modifiant pas le point d'entrée modifient le code au point d'entrée, pour donner la main au virus d'une manière plus discrete.

- **JMP Far**.- Soit en ajoutant un saut vers une autre section, c'est à dire vers le code du virus, détectable en scannant le point d'entrée à la recherche d'un "jmp far", il est possible d'utiliser un désassembleur pour obtenir un resultat plus précis.
- **PUSH / RET**.- Soit en ajoutant une combinaison d'instructions, tel qu'un "push adresse virus" suivis d'un RET. Ce type d'appel n'est jamais utilisé dans un programme normal, et surtout pas pour appeler le code d'une section différente de la section à laquelle se trouve ce code.
  Ex:

  ```
  406070 Push 414345
  406075 ret
  ```

Un virus peut modifier un programme d'une multitude de manières pour récuperer la main lors de l'exécution du fichier infecté, il peut donc être interressant d'ajouter d'autres type de code assez simple pour appeler le code d'une autre section.

**Recherche de fichiers PE** Un autre type de detection heuristique par analyse de code, est la detection de la recherche de fichiers PE. Un virus, pour pouvoir infecter un fichier, va d'abord vérifier s'il a bien affaire à un fichier PE, et donc un fichier infectable. On retrouve souvant dans les virus, le code suivant :

```
cmp word ptr [registre],'EP'
cmp word ptr [registre],'ZM'

Exemple dans un virus:

reloc:00404260                cmp     word ptr [esi], 'ZM'
```

Ce type de code est facilement détectable, surtout à l'aide d'un désassembleur intégré au moteur heuristique.

**Chaines de caractères particulieres** Certains virus contiennent des chaines de caractères en clair, surtout lors de la première génération, ou en cas de virus simples.

- "***.exe**".- Il n'est pas rare de trouver en clair, la chaine "*.exe" dans la section code d'un virus. Il s'agit du mask de recherche des fichiers qu'infecte le virus. Cette chaine permet de trouver de nouveaux fichiers hotes. Attention aux faux positifs, bien définir les regles de recherche : *.exe dans la derniere section par exemple si celle ci est différente de .data.
- **API et dll windows**.- Les premiers virus avaient dans leur section, le nom des fonctions d'API windows à charger en clair, ainsi que le nom des dlls à laquelle elles appartiennent. Certaines fonctions sont caractèristiques des virus Win32 (*FindFirstFileA, FindNextFileA, CreateFileA, CreateFileMapping, MapViewOfFile...* par exemple). Si ces chaines se retrouvent toutes en même temps dans la section supposée du virus, et que d'autres éléments suspects ont été detectés, il est fort probable qu'il s'agisse d'un virus simple, ou d'une première génération d'un virus qui n'a pas encore chiffré ses chaines par exemple.

**Utilisation du PEB pour récupérer des adresses systemes** Les virus fonctionnant seulement sous le systeme d'exploitation Windows NT peuvent récuperer l'adresse des fonctions de l'API Windows en utilisant le PEB (Process Environement Block). Il est possible de detecter ce code en désassemblant le point d'entrée d'un fichier.

Exemple concret d'un virus :

```
.rsrc:0040461B     mov     edx, fs:30h
.rsrc:00404621     mov     eax, [edx+0Ch]
.rsrc:00404624     mov     esi, [eax+1Ch]
.rsrc:00404627     mov     eax, [esi]
.rsrc:00404629     mov     esi, [eax+8]
.rsrc:0040462C     xchg    eax, esi
.rsrc:0040462D     mov     [ebp+401900h], eax
```

**Détection de code utilisant des adresses systemes "hardcodées"** Certains virus contiennent les adresses de certaines dll en dur dans leur code. En cas d'echec de la récupération de l'imagebase d'une dll par exemple, ou par exemple pour accéder au PEB (voir section précédente), en utilisant l'adresse du PEB directement. Celle ci est statique.

Exemple d'adresse hardcodée :

```
.reloc:00404630   mov     ebx, 0BFF70000h
```

L'adresse "BFF70000h" correspond à l'image base de la dll "kernel32.dll" dans les systemes d'exploitation de type Windows 98.

### 4.3 Emulation simple

Il est possible de coupler l'émulation à l'heuristique pour obtenir de meilleurs resultats.

**JMP Far** Comme nous l'avons vu plus haut, certains virus ne modifient pas le point d'entrée, et inserent un saut vers la section du virus. Il est possible d'émuler ce saut, et de continuer l'analyse du code dans la section du virus. Une fois au véritable point d'entrée du virus, la recherche d'indices qui nous permettent d'indiquer la présence eventuelle d'un virus peut continuer.

**PUSH / RET** Certains virus utilisent d'autres instructions pour se rendre au code du virus. Certains utilisent la combinaison PUSH / RET.

L'émulation de cette combinaison permet de trouver le véritable point d'entrée du virus, et de continuer l'analyse du code.

Il existe une multitude de possibilités, c'est pourquoi il est necéssaire d'émuler au minimum les instructions les plus classiques : PUSH, POP, RET, MOV, JMP, CALL, XOR, ADD, SUB, SHL, SHR...

**Emulation des décrypteurs** Les virus cryptés ou polymorphes utilisent des routines de décryptage, qui empechent l'analyse du code du virus, même si celui ci a pu être trouvé. Un émulateur de bonne qualité permet d'emuler les boucles de décryptages pour ensuite scanner l'interieur du virus, et donc de détecter les caractèristiques virales.

## 5 Techniques Anti Heuristiques

Les techniques de détections évoluants, les virus ont eux aussi évolués pour continuer les méthodes de détections heuristiques. Je vais vous présenter maintenant, différentes méthodes anti heuristiques.

### 5.1 Structure PE

**Non Modification des caractéristiques des sections** Certains virus ne modifient pas les sections de la dernière section. Ils utilisent la fonction de l'API Windows : VirtualProtect. Le virus change les caractéristiques de la "section" en mémoire, et peut donc exécuter du code, meme si sur le disque la section n'a pas l'attribut exécutable. Les moteurs heuristiques ne peuvent donc pas detecter l'exécution dans la derniere section.

**Ajout de plusieurs sections** En régle général, le virus se copie dans la dernière section. Cependant, certain virus ajoutent deux sections. La première des sections est exécutable et contient le code du virus, alors que la seconde n'est pas utilisée, et ne contient aucun attribut suspect tel que l'exécution ou l'écriture. Les anti virus heuristiques scannant la dernière section ne detectent rien d'anormal.

**Ajout d'un bout de code du virus dans la première section (point d'entrée tjs dans la section code)** Certains virus copient le code du programme original qu'ils écrasent, dans la derniere section, sans toucher les attributs de celle ci. Ils écrasent ensuite le code sauvegardé, et ajoutent un tout petit bout de code, permettant d'allouer de la mémoire pour copier ensuite le véritable code du virus en mémoire alloué, et ensuite d'exécuter celui ci. Le virus prends donc la main, et replace le code précédement écrasé et infecte d'autres fichiers, avant de rentre la main à l'application qui s'exécutera normalement.

Le principe de cette technique permet d'exécuter le code du virus, sans avoir à changer les attributs des sections, et donc de rester discret et de ne pas se faire detecter par les moteurs heuristiques.

**Packing de la section code et ajout du virus dans l'espace non utilisé** Cette technique est très similaire à celle précédement présentée, excepté que la section code est compressée pour prendre moins de place. Le virus se copie ensuite à l'endroit précédement utilisé par le code non compressé. Le comportement du virus est ensuite très similaire à l'explication précédente.

**Point d'Entrée Obscure** La technique du point d'entrée obscure est une évolution du simple saut inséré à l'entry point du fichier hote. De nombreuses techniques ont été inventées pour rendre la main au virus, sans modifier le point d'entrée du programme infecté.

- **Patch des appels aux API windows pour appeler le virus**.- Une des méthodes les plus utilisée est la modification d'un appel vers une fonction de l'API Windows. Les fonctions dans un programme sont appelées de la manière suivante :

```
.text:004012A4
; void __stdcall ExitProcess(UINT uExitCode)
.text:004012A4   ExitProcess     proc near
; CODE XREF: start+Fp
.text:004012A4 FF 25 04 20 40 00
               jmp     ds:__imp_ExitProcess
.text:004012A4
                  ExitProcess     endp
```

  Certains compilateurs utilisent un FF 15 (call dword ptr) pour appeler les fonctions.
  Voici le même appel patché par un virus :

```
.text:004012A4 sub_4012A4 proc near
 ; CODE XREF: start+Fp
.text:004012A4   call    loc_404600
.text:004012A9            nop
.text:004012A9 sub_4012A4 endp
```

  Concraitement, le virus a modifié l'appel à la fonction ExitProcess. Au lieu d'appeler cette fonction, le programme appelera le code du virus. Dans le cas de la fonction ExitProcess, le virus se déclanchera seulement quand on

ferme le programme. Les sandbox n'émulant que le début du programme ne détecteront pas ce type de virus par exemple.
Les moteurs heuristiques doivent scanner les appels aux fonctions d'API pour detecter ce type de détournement, et cela implique un scan plus long car tout le fichier doit être parcouru, surtout si la fonction patchée est choisie aléatoirement.

– **Patch du Stack Frame**.- Certains virus modifient l'initialisation d'une fonction pour y insérer un saut vers le code du virus. Ce type de point d'entrée obscure est plus difficile à detecter que le premier. Le premier peut être détecté en scannant l'espace du fichier contenant tout les appels aux fonctions d'API, il est très simple de trouver un détournement. Dans le cas du patch du stack frame, il n'y a aucun espace contenant tout les stack frames des fonctions.
L'anti virus doit scanner tout le fichier de façons intelligentes pour detecter le code du virus. Si le virus utilise un code différent pour chaque patch du Stack Frame, il peut être très difficile de détecter ce genre de virus, surtout s'ils sont en plus polymorphiques.

**Calcul du Checksum du fichier PE** Les virus anti heuristiques mettent à jour le checksum des fichiers PE qu'ils infectent, afin d'éviter le drapeau "checksum du fichier incorrect" dans les moteurs heuristiques.

**Renomage des sections existantes** Les virus élargissants la dernière section, modifient souvent les caractèristiques de la derniere section pour leur ajouter l'attribut exécutable. Les anti virus detectent ce genre de modification, et la suspicion est encore plus grande si le nom de la section est connue pour ne jamais avoir les attributs exécutables : .reloc par exemple. Certains virus modifient tout simplement le nom de la section après l'avoir modifiée.

A noté qu'il est possible de détecter ce genre de modification si le pointeur vers les relocations dans le PE header n'a pas été modifié. Une section de relocation n'ayant pas pour nom ".reloc" est très suspect.

**"Size of Code" est corrigé** Les virus anti heuristiques mettent à jour la size of Code pour éviter d'obtenir le drapeau lorsqu'ils sont scannés par les moteurs heuristiques.

### 5.2 Anti Emulation

Les moteurs utilisant l'émulation étant de plus en plus utilisés, des techniques anti émulations sont apparues.

**SEH - Structured Exception Handling.** Pour contourner les premiers Emulateurs, les virus utilisaient des Exceptions Handlers (Gestion des erreurs par le programme) afin de déstabiliser les émulateurs. Les virus créaient des erreurs

dans leur code, et s'occupaient eux même de gérer l'erreur pour continuer le code juste après. Certains virus utilisaient ce principe pour effectuer le saut final vers le fichier hote. De nos jours, tout les Emulateurs dignes de ce nom, supportent les Exceptions.

**Instructions du Co-Processeur** Les premiers Emulateurs ne pouvaient emuler les instructions du co-processeur tel que le FPU. Les auteurs de virus utilisaient donc ces instructions dans le code de leur virus pour déstabiliser les Emulateurs et donc empecher la detection de leur virus.

**MMX / SSE** Comme pour les instructions FPU, certains virus utilisent les instructions MMX et SSE pour decrypter leur "corps". Les emulateurs ne supportant pas ses instructions ne pouvaient donc pas decrypter le corps du virus, et donc continuer l'analyse.

Tout les Emulateurs actuels supportent ce type d'instructions de nos jours.

**Instructions non documentées** Les processeurs intel contiennent quelques instructions non documentées et les emulateurs ne supportant pas ses instructions ne pouvaient pas correctement emuler les virus les utilisant intelligement. De nos jours, ses instructions sont correctement émulées.

**Code Anti Machine Virtuelle** Certains virus utilisent du code pour empecher leur exécution dans une machine Virtuelle telle que Vmware ou Virtual PC. Il est généralement assez simple de detecter ses machines virtuelles, certains virus peuvent ensuite agir différement en fonction du type de machine détectée, ou alors cesser de fonctionner tout simplement dans une machine virtuelle.

**Couches de cryptage avec brute forcing** Des virus comme Crypto utilisent des couches de cryptage qui se brutent forcent jusqu'a trouver la clé permettant le décryptage de la suite du virus. L'interet de ce type de cryptage/decryptage et qu'il ralentie considérablement le scanning des fichiers, les Emulateurs étant plus lents qu'une exécution réelle d'un programme, l'émulation complete des layers de cryptage d'un virus utilisant ce type d'algorithme peut prendre plusieurs minutes par fichiers.

**Threads** Certains virus utilisent des threads pour exécuter certaines opérations, comme le decryptage du code du virus. Les Emulateurs ne supportant pas le multi threading ne peuvent pas emuler correctement ce type de virus.

### 5.3 Code anti heuristique

**Le delta offset est obtenu différement** Certains virus calculent le delta offset différement afin de ne pas se faire detecter par les anti virus heuristiques

qui effectuent de l'analyse de code. En général le code du delta offset est de la forme : E800000000 ou très similaire. Il est aussi suivit d'un pop registre.

Il est possible de programmer le calcul du delta offset de cette manière :

```
startvirus:

call calculatedelta
delta:
sub ebp, (offset delta1-start)

<code du virus ici>

calculatedelta:
pop ebp
jmp delta
```

De cette manière le code généré est différent des calculs de delta offsets standard, et permet donc d'echapper aux detections simples de calculs de delta offset. Il est possible de modifier ce code en ajoutant du code qui n'a aucune utilité entre chaque instructions, pour rendre les detections encore plus complexes.

**Le code pour la recherche de fichiers PE est obscurci** Certains moteurs detectent la recherche de fichiers PE. Pour contre carrer ca, les auteurs de virus peuvent employer des méthodes différentes :

```
mov eax, 'EP' xor 46h
; le resultat de "EP" xor 46h est stockée
; lors de la compilation.
xor eax,46h
; au lancement la valeur est mise dans EAX,
; puis ensuite xorée.
cmp byte ptr [edi],al
; on compare le premier octet pointé par EDI
; avec al qui contient "P"
jnz bad_pe_file
; si c'est pas bon, on sort.
cmp byte ptr [edi+1],ah
; sinon si le deuxième octet est égal à "ah",
; c'est à dire "E", alors c'est un fichier PE.
jnz bad_pe_file
; sinon on sort.
```

Une autre variation :

```
mov  eax,dword ptr [edi]
; EDI pointe sur PE\00
shl eax,3
```

```
; 00004550 << 3
cmp ax,2a80h
; si ax = 2a80 alors fichier PE
jnz bad_pe_file
; sinon mauvais fichier.
```

Il existe des centaines de possibilités pour programmer les detections différements, et ce genre de code contourne les analyses de code simples sans difficultés.

**Les API ne sont plus référencées directement (checksum)** Certains moteurs heuristiques recherchent la présence des chaines de caractères correspondantes au nom de fonctions de l'API Windows souvant utilisées par les infecteurs de fichiers PE, dans la section ou le virus est sensé se trouver. Pour contourner ce problème, la majorité des virus contiennent le CRC de la chaine de caractère pour identifier l'API à charger, et ils utilisent souvent un CRC maison, pour eviter la recherche des valeurs de CRC, si ceux si utilisent des algorithmes standards.

## 6 Présentation d'un moteur Heuristique Perso

J'ai programmé un moteur heuristique permettant la detection de virus Win32 connus et inconnus en se basant sur l'analyse de la structure PE, ainsi que du code de l'application.

### 6.1 Analyse de binaires standards : notepad, regedit, calc, Ms Pain, WordPad...

Les applications standards ne déclanchent aucun drapeau.

#### Notepad

```
Heuristic Engine v0.06
Filename: C:\WINNT\NOTEPAD.EXE

Nb Sections: 7
Size of Code: 38000
Entry Point: 2F9A9
Image Base: 1000000

.text Vsize:65CA RVA:1000 Psize:0
      offset:0 Flags:60000020
.data Vsize:1944 RVA:8000 Psize:0
      offset:0 Flags:C0000040
.text1 Vsize:30000 RVA:A000 Psize:2B000
      offset:1000 Flags:E0000020
```

```
.adata Vsize:10000 RVA:3A000 Psize:D000
      offset:2C000 Flags:E0000020
.data1 Vsize:20000 RVA:4A000 Psize:9000
      offset:39000 Flags:C0000040
.pdata Vsize:30000 RVA:6A000 Psize:27000
      offset:42000 Flags:C0000040
.rsrc Vsize:6000 RVA:9A000 Psize:4000
      offset:69000 Flags:40000040

This file looks normal :-)
```

### Regedit

```
Heuristic Engine v0.06
Filename: C:\WINNT\regedit.exe

Nb Sections: 3
Size of Code: AE00
Entry Point: 72E6
Image Base: 1000000

.text Vsize:ADBC RVA:1000 Psize:AE00
      offset:600 Flags:60000020
.data Vsize:48AF0 RVA:C000 Psize:200
      offset:B400 Flags:C0000040
.rsrc Vsize:8000 RVA:55000 Psize:7200
      offset:B600 Flags:40000040

This file looks normal :-)
```

### Taskman

```
Heuristic Engine v0.06
Filename: C:\WINNT\TASKMAN.EXE

Nb Sections: 3
Size of Code: 4C00
Entry Point: 28A0
Image Base: 1000000

.text Vsize:4A84 RVA:1000 Psize:4C00
      offset:600 Flags:60000020
.data Vsize:3938 RVA:6000 Psize:2400
      offset:5200 Flags:C0000040
.rsrc Vsize:2000 RVA:A000 Psize:1600
      offset:7600 Flags:40000040
```

```
This file looks normal :-)
```

### 6.2 Analyse de binaires Infectées : Virus polymorphes, Cryptés, Standard, EPO...

#### Virus XTC

```
Heuristic Engine v0.06
Filename: C:\reverse engineering\VIRUS\NOUVEAUX\XTC.VIR

Nb Sections: 5
Size of Code: 0
Entry Point: 9000
Image Base: 400000

.AVXencr Vsize:2000 RVA:1000 Psize:E00
         offset:400 Flags:C0000040
.AVXencr Vsize:2000 RVA:3000 Psize:400
         offset:1200 Flags:C0000040
.AVXencr Vsize:2000 RVA:5000 Psize:400
         offset:1600 Flags:C0000040
.rsrc Vsize:2000 RVA:7000 Psize:200
         offset:1A00 Flags:C0000040
.AVXencr Vsize:4000 RVA:9000 Psize:3400
         offset:1C00 Flags:C0000040

Instruction at entry point is not usually used
by a compiler. ASM ? ;-)
Execution starts in last section.
Last section is executable. (Its used by virus
and PE protector)
Suspicious Delta trick (used by injected code)
found at EP+14
Code section is writeable.

This file is probably infected :-/
```

#### Virus Cocaine

```
Heuristic Engine v0.06
Filename:
C:\reverse engineering\VIRUS\NOUVEAUX\COCAINE.VIR

Nb Sections: 4
Size of Code: 2C00
```

```
Entry Point: 3750
Image Base: 1020000

.text Vsize:2BFE RVA:1000 Psize:2C00
      offset:400 Flags:60000020
.data Vsize:9CC RVA:4000 Psize:A00
      offset:3000 Flags:C0000040
.rsrc Vsize:BAC RVA:5000 Psize:C00
      offset:3A00 Flags:40000040
.reloc Vsize:7000 RVA:6000 Psize:6200
      offset:4600 Flags:C0000040

Instruction at entry point is not usually used
by a compiler. ASM ? ;-)
Suspicious JMP FAR found at Entry point+9.Might
be some OEP obfuscation.
Suspicious difference between OEP RVA and
Section RVA.OEP obfuscation?

This file is probably infected :-/
```

### Virus Energy

```
Heuristic Engine v0.06
Filename:
C:\reverse engineering\VIRUS\NOUVEAUX\Energy.vir

Nb Sections: 5
Size of Code: 0
Entry Point: 6000
Image Base: 400000

9286742 Vsize:1000 RVA:1000 Psize:600
        offset:400 Flags:C0000040
5808079 Vsize:1000 RVA:2000 Psize:0
        offset:A00 Flags:C0000040
9079359 Vsize:2000 RVA:3000 Psize:200
        offset:A00 Flags:C0000040
.rsrc Vsize:1000 RVA:5000 Psize:200
        offset:C00 Flags:C0000040
.data Vsize:2000 RVA:6000 Psize:1C00
        offset:E00 Flags:C0000040

Instruction at entry point is not usually used
by a compiler. ASM ? ;-)
Execution starts in last section.
```

```
Last section is executable. (Its used by virus
and PE protector)
Suspicious Delta trick (used by injected code)
found at EP+14
Code section is writeable.

This file is probably infected :-/
```

### Virus Crypto

```
Heuristic Engine v0.07
Filename:
H:\reverse engineering\VIRUS\NOUVEAUX\CRYPTO.VIR

Nb Sections: 4
Size of Code: 200
Entry Point: 9FF4
Image Base: 400000

.text Vsize:DD5 RVA:1000 Psize:1000
      offset:1000 Flags:60000060
.idata Vsize:E4 RVA:2000 Psize:1000
      offset:2000 Flags:40000040
.rsrc Vsize:1000 RVA:3000 Psize:1000
      offset:3000 Flags:40000040
.reloc Vsize:8000 RVA:4000 Psize:8000
      offset:4000 Flags:E2000060

Instruction at entry point is not usually used
by a compiler. ASM ? ;-)
Execution starts in last section.
Last section is Writeable and Executable.
A section known not to have any code is executable!
Very suspicious.

This file is probably infected! :-/
```

### PEjacky

```
Heuristic Engine v0.06
Filename:
C:\reverse engineering\VIRUS\Souches InfectÚs
   (via vmware)\PEjacky\NO
EPAD.VIR

Nb Sections: 3
```

```
Size of Code: 6600
Entry Point: F800
Image Base: 1000000

.text Vsize:65CA RVA:1000 Psize:6600
      offset:600 Flags:60000020
.data Vsize:1944 RVA:8000 Psize:600
      offset:6C00 Flags:C0000040
.rsrc Vsize:6000 RVA:A000 Psize:5E00
      offset:7200 Flags:E0000040

Execution starts in last section.
Last section is executable. (Its used by virus
and PE protector)
Suspicious code looking for PE files. (could
be a virus or a PE tool
Suspicious difference between OEP RVA and
Section RVA.OEP obfuscation?

This file is probably infected :-/
```

### 6.3 Analyse de fichiers PE packés : UPX, Aspack, FSG...

**PeLocknt**

```
Heuristic Engine v0.06
Filename:
C:\reverse engineering\TASM5Plus\EXAMPLE\contemplate/
  \packers\pelocked.EXE

Nb Sections: 5
Size of Code: 600
Entry Point: 5000
Image Base: 400000

PELOCKnt Vsize:1000 RVA:1000 Psize:600
         offset:600 Flags:E0000020
PELOCKnt Vsize:1000 RVA:2000 Psize:600
         offset:C00 Flags:C0000040
PELOCKnt Vsize:1000 RVA:3000 Psize:200
         offset:1200 Flags:C0000040
PELOCKnt Vsize:1000 RVA:4000 Psize:200
         offset:1400 Flags:C0000040
PELOCKnt Vsize:2A00 RVA:5000 Psize:2A00
         offset:1600 Flags:C0000060
```

```
This file is protected by PE-Locknt 2.4
```

**PE Protect**

```
Heuristic Engine v0.06
Filename:
C:\reverse engineering\TASM5Plus\EXAMPLE\contemplate/
  \packers\pe-prot.EXE

Nb Sections: 5
Size of Code: 600
Entry Point: 5000
Image Base: 400000

CODE Vsize:1000 RVA:1000 Psize:600
     offset:600 Flags:E0000020
DATA Vsize:1000 RVA:2000 Psize:600
     offset:C00 Flags:C0000040
.idata Vsize:1000 RVA:3000 Psize:200
     offset:1200 Flags:C0000040
.reloc Vsize:1000 RVA:4000 Psize:200
     offset:1400 Flags:50000040
.PE-PROT Vsize:1000 RVA:5000 Psize:455
     offset:1800 Flags:C0000040

This file is protected by PE-Protect
```

## 7 Conclusion

Les techniques heuristiques Win32 permettent de détecter beaucoup de virus Win32 sans avoir recours à des signatures spécifiques. Cependant, des problèmes de faux positifs existent, notament avec les packers d'exécutables PE, qui fonctionnent grossièrement, de la même manière que les virus. (Code Relogeable, Delta Offset, etc) Il est important d'utiliser des réglès minutieusement définie, ainsi qu'un émulateur puissant, pour pouvoir détecter la plus grand majorité des infections d'exécutables Win32 PE.

Les moteurs heuristiques permettent toutes fois de detecter un exécutable "suspect", et ensuite de permettre l'analyse de ce fichier, par une personne compétente.