

# INTRODUCTION À L'ANALYSE DE MALWARE

# LE PROF

- Laurent Clévy (@lorenzo2472)
- Informatique Forensique depuis 2013
- Coordination des analyses Forensique et Malware chez Thales depuis 2015
- Giac Certified Forensic Analyst ([GCFA](#)) depuis 2013, [GREM](#) (Malware reversing) depuis 2015
- Auteur de plusieurs articles MISC sur le forensic et l'analyse de malware
- Cours d'analyse forensique à l'AFTI (master 2) entre 2015 et 2019.

# ORGANISATION DU COURS

- 9h30-11h30 : cours
- 12h30-17h30 : TDs et évaluations
  - 12h30 : TD1
  - 13h50 : Evaluation TD1
  - 14h10 : pause de 30mn
  - 14h40 : TD2
  - 16h10 : Evaluation TD2
  - 16h30 : Synthèse
  - 17h30 : envoi de la synthèse

# AGENDA

- Principes de l'analyse malware
- Analyse de maldocs
  - Pdf
  - OLE2 et OpenXml
- Analyse statique exécutable Windows
  - Approche basique
  - Signatures
  - Caractéristiques du format PE
- Analyse dynamique

# PRINCIPES DE L'ANALYSE MALWARE

# MALWARE : MALICIOUS SOFTWARE

## Quelque chose qui s'exécute

- Parfois ce n'est pas censé le faire (prévisualisation des fichiers offices dans l'explorateur Windows, exécute les macros)

## Mais pas avec le comportement attendu

- Pourquoi un économiseur d'écran dump le process lsass.exe ?

# MALWARE : CARACTERISTIQUES

## Cache ses fonctions internes

- Chiffrement ou Packing (compression et encapsulation)
- Encodage des chaînes
- Chargement dynamique des DLL
- Encapsulation (poupées russes, multiples étages)

## Protections pour retarder l'analyse

- Offuscation
- Anti-debug
- Anti-VM (pour tromper les sandbox)
- Contournement des AV/EDR

# UNE MÉTHODE FACILE ? SANDBOX

Simule un environnement d'exécution pour faire « détoner » le malware, observer son comportement système et réseau, récolter des marqueurs / IOCs

Souvent couplé avec plusieurs antivirus

Mais:

- Facile pour un malware de comprendre qu'il est dans une VM et donc de ne pas dévoiler ces fonctions malveillantes, ou alors on attend une action de l'utilisateur... (mouvement de souris) qui n'arrive jamais dans une sandbox
- Et pour interpréter les résultats détaillés ou un verdict suspect, il faut connaître les bases de l'analyse de malware: vous n'avez pas le choix 😊



# OBJECTIFS DE L'ANALYSE DE MALWARE

Déterminer les fonctions internes / cachées

- Accès à distance
- Vol de données
- Écoute du clavier, activation de la caméra ou du micro ?
- Mouvement latéral ?

Extraire des IOCs

- Pour les chercher dans les logs, sur le parc, dans les bases CTI

Identifier l'étage suivant / bloquer son téléchargement

Bloquer le malware ! Créer des règles de détection

Reconnaitre le groupe d'attaquant

# QUELQUE CHOSE QUI S'EXECUTE

## Du code dans un document

- PDF
- Fichiers office (.doc, .xls, .ppt, .xlsx, ...)
- RTF
- ...

## Un script

- Powershell (.ps1)
- Javascript
- VBA (dans les documents Office)
- Shell
- Autolt
- .bat

## Binaires

- Win32/Win64
- ELF

# OFFUSCATION

## Rendre difficile la compréhension du code

- Possible quel que soit le langage

```
967427917; sleep -s 73;$nhi=Get-ItemProperty -path
("hk"+"cu:\sof"+"tw"+"are\mic"+"ros"+"oft\Phone\"+[Environment]::("use"+"rn"+"ame")+"0");for
($pph=0;$pph -le 738;$pph++){Try{$ul+=$nhi.$pph}Catch{}};$pph=0;while($true){ $pph++;
$ko=[math]::("sq"+"rt") ($pph); if($ko -eq 1000){break}}$fq=$ul.replace("#",$ko);
$sqx=[byte[]]::("ne"+"w") ($fq.Length/2); for($pph=0;$pph -lt
$fq.Length;$pph+=2){ $sqx[$pph/2]=[convert]::("ToB"+"yte")
($fq.Substring($pph,2),(2*8))}[reflection.assembly]::("Lo"+"ad") ($sqx);[Open]::("Te"+"st") ();683724585;
```

```
967427917;
sleep -s 73;
$nhi=Get-ItemProperty -path ("hk"+"cu:\sof"+"tw"+"are\mic"+"ros"+"oft\Phone\"+[Environment]::("use"+"rn"+"ame")+"0");
for ($pph=0;$pph -le 738;$pph++){
    Try{$ul+=$nhi.$pph}Catch{}
};
$pph=0;
while($true){
    $pph++;
    $ko=[math]::("sq"+"rt") ($pph);
    if($ko -eq 1000){break}
}
$fq=$ul.replace("#",$ko);
$sqx=[byte[]]::("ne"+"w") ($fq.Length/2);
for($pph=0;$pph -lt $fq.Length;$pph+=2){
    $sqx[$pph/2]=[convert]::("ToB"+"yte") ($fq.Substring($pph,2),(2*8))
}
[reflection.assembly]::("Lo"+"ad") ($sqx);[Open]::("Te"+"st") ();
683724585;
```

Lit le contenu d'une clé de  
registre spécifique au malware

Substitution  
et conversion

Exécution du résultat

IOC =

« HKCU:Software\Microsoft\Phone\[username] »

<https://redcanary.com/blog/gootloader/>

# DETECTER LE CHIFFREMENT OU LE PACKING

Calculer l'entropie

- Proche de 8.0 : chiffrement
- Entre 6.5 et 7.5 : compression

# MALDOC : MALICIOUS DOCUMENT

# MALDOC : MALICIOUS DOCUMENT

Il est possible d'exécuter du code dans un PDF ou un document Office à l'ouverture du document ou suite à une action utilisateur

- PDF : Javascript,
- XLS : Macro Excel<sub>4</sub>
- Doc, Xls, ppt : VBA

# PDF : PORTABLE DOCUMENT FORMAT

Depuis 1993

Basé sur postscript (format texte)

À base d'objets

Pour l'analyser: **pdfid** et **pdf-parser** de Didier Stevens

<https://blog.didierstevens.com/programs/pdf-tools/>  
<https://www.pdfa.org/resource/pdf-specification-index/>

# DOCUMENTS MICROSOFT OFFICE

2 formats (avant et après 2007)

- OLE2 : système de fichier, avec des « flux »

**Header Signature (8 bytes):** Identification signature for the compound file structure, and MUST be set to the value 0xD0, 0xCF, 0x11, 0xE0, 0xA1, 0xB1, 0x1A, 0xE1.

- OpenXML : fichiers XML dans une archive Zip

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00000000	50	4B	03	04	14	00	06	00	08	00	00	00	21	00	DF	2F	PK.....!.B/
00000010	EE	78	D3	01	00	00	79	0A	00	00	13	00	08	02	5B	43	isÛ...y.....[C
00000020	6F	6E	74	65	6E	74	5F	54	79	70	65	73	5D	2E	78	6D	ontent_Types].xm
00000030	6C	20	A2	04	02	28	A0	00	02	00	00	00	00	00	00	00	1 e..( .....

<https://www.decalage.info/python/oletools>

<https://www.sstic.org/user/plagadec/>



# OLETOOLS, POUR OLE<sub>2</sub> SURTOUT

## Tools in oletools:

### Tools to analyze malicious documents

- **oleid**: to analyze OLE files to detect specific characteristics usually found in malicious files.
- **olevba**: to extract and analyze VBA Macro source code from MS Office documents (OLE and OpenXML).
- **MacroRaptor**: to detect malicious VBA Macros
- **msodde**: to detect and extract DDE/DDEAUTO links from MS Office documents, RTF and CSV
- **pyxswf**: to detect, extract and analyze Flash objects (SWF) that may be embedded in files such as MS Office documents (e.g. Word, Excel) and RTF, which is especially useful for malware analysis.
- **oleobj**: to extract embedded objects from OLE files.
- **rtfobj**: to extract embedded objects from RTF files.

### Tools to analyze the structure of OLE files

- **olebrowse**: A simple GUI to browse OLE files (e.g. MS Word, Excel, Powerpoint documents), to view and extract individual data streams.
- **olemeta**: to extract all standard properties (metadata) from OLE files.
- **oletimes**: to extract creation and modification timestamps of all streams and storages.
- **oledir**: to display all the directory entries of an OLE file, including free and orphaned entries.
- **olemap**: to display a map of all the sectors in an OLE file.

À utiliser dans l'ordre : oleid, oleobj, olevba

# OLEDUMP (FORMAT OLE2)

Pour aller plus loin, analyser la structure et les flux OLE2  
Extraire un flux, le décoder

Plug-ins pour les vba, ppt, xls et msg

- Pour analyser un email office .msg sans l'ouvrir

<https://blog.didierstevens.com/programs/oledump-py/>

# EXECUTABLE (WINDOWS)

# EXECUTABLE

Fichier sur disque conçu pour **devenir un processus**

Conçu pour de multiples architectures (x32, x64, arm, ...)

Tâches à réaliser par le loader:

- Charger le code en mémoire
- Initialiser la pile et le tas
- Traduire les adresses dans le code
- **Charger les bibliothèques partagées** (.dll ou .so)
- Lier les adresses des fonctions externes à l'espace mémoire du processus

Windows: format Portable Executable (PE):

<https://learn.microsoft.com/en-us/windows/win32/debug/pe-format>

# WINDOWS PE: EN HEXA ?

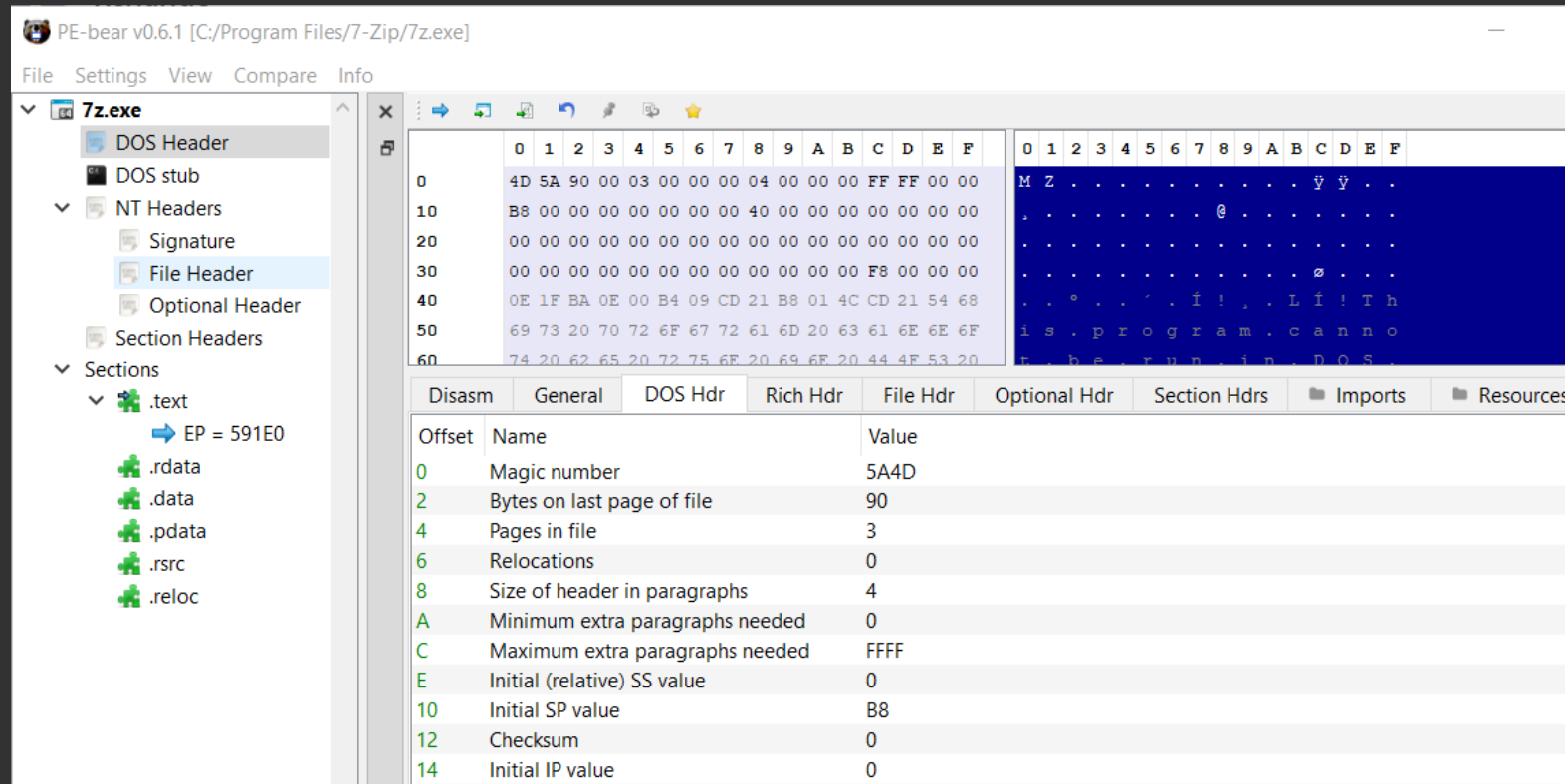
Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Texte Décodé
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ.....ÿÿ..
00000010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00	.....@.....
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000030	00	00	00	00	00	00	00	00	00	00	00	00	F8	00	00	00	.....ø....
00000040	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68	...°.í!..Lí!Th
00000050	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	is program canno
00000060	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20	t be run in DOS
00000070	6D	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00	mode....\$.....
00000080	56	89	6D	E2	12	E8	03	B1	12	E8	03	B1	12	E8	03	B1	Vmmâ.è.±.è.±.è.±
00000090	64	75	7E	B1	13	E8	03	B1	64	75	78	B1	19	E8	03	B1	du~±.è.±dux±.è.±
000000A0	12	E8	02	B1	8B	E8	03	B1	64	75	6D	B1	5C	E8	03	B1	.è.±<è.±dum±\è.±
000000B0	AB	9D	07	B0	13	E8	03	B1	64	75	6E	B1	02	E8	03	B1	«..°.è.±dun±.è.±
000000C0	8A	9A	00	B0	13	E8	03	B1	64	75	7F	B1	13	E8	03	B1	Šš.°.è.±du.±.è.±
000000D0	64	75	7B	B1	13	E8	03	B1	52	69	63	68	12	E8	03	B1	du{±.è.±Rich.è.±
000000E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000000F0	00	00	00	00	00	00	00	00	50	45	00	00	64	86	06	00	.....PE..dt.
00000100	60	75	C8	61	00	00	00	00	00	00	00	00	F0	00	22	00	`uÊa.....@.....
00000110	0B	02	08	00	00	9E	05	00	00	AE	02	00	00	00	00	00	.....ž...@.....
00000120	E0	9D	05	00	00	10	00	00	00	00	40	00	00	00	00	00	à.....@.....
00000130	00	10	00	00	00	02	00	00	04	00	00	00	00	00	00	00	.....
00000140	05	00	02	00	00	00	00	00	00	70	08	00	00	04	00	00	.....p.....
00000150	00	00	00	00	03	00	60	81	00	00	10	00	00	00	00	00	.....`.....
00000160	00	10	00	00	00	00	00	00	00	00	10	00	00	00	00	00	.....
00000170	00	10	00	00	00	00	00	00	00	00	00	00	10	00	00	00	.....
00000180	00	00	00	00	00	00	00	00	A4	9B	07	00	78	00	00	00	.....»...x...
00000190	00	50	08	00	E0	06	00	00	00	E0	07	00	8C	6D	00	00	.P..à....à..@m..
000001A0	00	00	00	00	00	00	00	00	00	60	08	00	60	07	00	00	.....`.....
000001B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000001C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000001D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000001E0	00	B0	05	00	40	04	00	00	00	00	00	00	00	00	00	00	..°.è.....
000001F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000200	2E	74	65	78	74	00	00	00	C6	9D	05	00	00	10	00	00	.text...E.....
00000210	00	9E	05	00	00	04	00	00	00	00	00	00	00	00	00	00	.z.....
00000220	00	00	00	00	20	00	00	60	2E	72	64	61	74	61	00	00	...`..rdata.
00000230	04	F9	01	00	00	B0	05	00	00	FA	01	00	00	A2	05	00	.ù...°.....

Commence par la valeur  
« MZ »

Plus loin, il y a la partie  
« PE »

Et les noms des sections,  
ici « .text », « .rdata »

# WIN PE : ENTÊTE DOS



# MZ est un header DOS historique

# WIN PE : ENTÊTE PE / NT

PE-bear v0.6.1 [C:/Program Files/7-Zip/7z.exe]

File Settings View Compare Info

7z.exe

- DOS Header
- DOS stub
- NT Headers
  - Signature
  - File Header
  - Optional Header
- Section Headers

Sections

- .text
  - EP = 591E0
- .rdata
- .data
- .pdata
- .rsrc
- .reloc

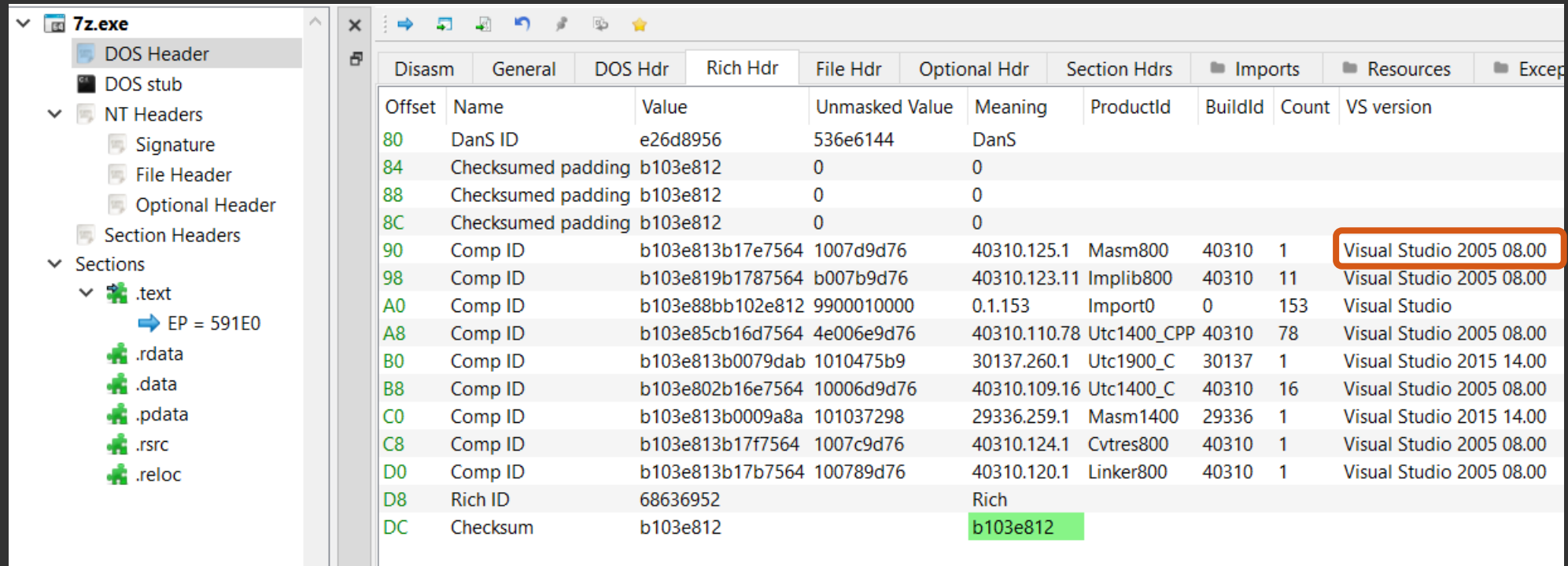
Offset	Name	Value	Meaning
FC	Machine	8664	AMD64 (K8)
FE	Sections Count	6	6
100	Time Date Stamp	61c87560	dimanche, 26.12.2021 14:00:00 UTC
104	Ptr to Symbol Table	0	0
108	Num. of Symbols	0	0
10C	Size of OptionalHeader	f0	240
10E	Characteristics	22	
		2	File is executable (i.e. no unresolved external references).
		20	App can handle >2gb addresses

En forensic on aime les timestamps!

Mais il peut être modifié sans traces par l'attaquant

PE ou NT\_header est le header principal

# WIN PE : ENTÊTE RICH

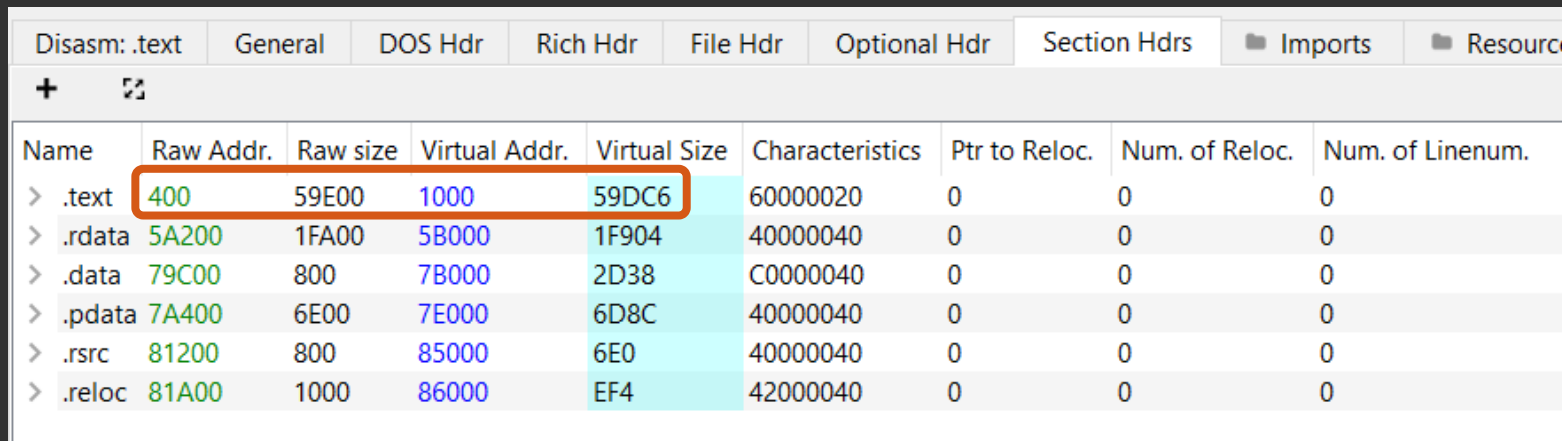


Offset	Name	Value	Unmasked Value	Meaning	ProductId	BuildId	Count	VS version
80	DanS ID	e26d8956	536e6144	DanS				
84	Checksummed padding	b103e812	0	0				
88	Checksummed padding	b103e812	0	0				
8C	Checksummed padding	b103e812	0	0				
90	Comp ID	b103e813b17e7564	1007d9d76	40310.125.1	Masm800	40310	1	Visual Studio 2005 08.00
98	Comp ID	b103e819b1787564	b007b9d76	40310.123.11	Implib800	40310	11	Visual Studio 2005 08.00
A0	Comp ID	b103e88bb102e812	9900010000	0.1.153	Import0	0	153	Visual Studio
A8	Comp ID	b103e85cb16d7564	4e006e9d76	40310.110.78	Utc1400_CPP	40310	78	Visual Studio 2005 08.00
B0	Comp ID	b103e813b0079dab	1010475b9	30137.260.1	Utc1900_C	30137	1	Visual Studio 2015 14.00
B8	Comp ID	b103e802b16e7564	10006d9d76	40310.109.16	Utc1400_C	40310	16	Visual Studio 2005 08.00
C0	Comp ID	b103e813b0009a8a	101037298	29336.259.1	Masm1400	29336	1	Visual Studio 2015 14.00
C8	Comp ID	b103e813b17f7564	1007c9d76	40310.124.1	Cvtres800	40310	1	Visual Studio 2005 08.00
D0	Comp ID	b103e813b17b7564	100789d76	40310.120.1	Linker800	40310	1	Visual Studio 2005 08.00
D8	Rich ID	68636952		Rich				
DC	Checksum	b103e812		b103e812				

Fournit les versions des fichiers objets composant l'exe (encodé avec une clé xor). Uniquement avec Visual Studio. Cela peut être utilisé comme signature de la chaîne de compilation de l'attaquant... ou pour tromper l'analyste



# WIN PE : SECTIONS



Name	Raw Addr.	Raw size	Virtual Addr.	Virtual Size	Characteristics	Ptr to Reloc.	Num. of Reloc.	Num. of Linenum.
> .text	400	59E00	1000	59DC6	60000020	0	0	0
> .rdata	5A200	1FA00	5B000	1F904	40000040	0	0	0
> .data	79C00	800	7B000	2D38	C0000040	0	0	0
> .pdata	7A400	6E00	7E000	6D8C	40000040	0	0	0
> .rsrc	81200	800	85000	6E0	40000040	0	0	0
> .reloc	81A00	1000	86000	EF4	42000040	0	0	0

Nom des sections (optionnel). « .text » pour le code,  
« .rdata » pour données / constantes initialisées, « .rsrc » pour les ressources (éléments GUI)

**RAW** addr et size = sur le disque

**Virtual** addr et size = en mémoire

Ici : le mapping **fichier** et **mémoire** est identique, aux alignements prêts : rien de bizarre

« Overlay » = données après la fin officielle du fichier. Plutôt inhabituel

# WIN PE : PROPRIÉTÉS DES SECTIONS

Name	Raw Addr.	Raw size	Virtual Addr.	Virtual Size	Characteristics
▼ .text	400	59E00	1000	59DC6	60000020
>	5A200	^	5B000	mapped: 5A000	r-x
▼ .rdata	5A200	1FA00	5B000	1F904	40000040
>	79C00	^	7B000	mapped: 20000	r--
▼ .data	79C00	800	7B000	2D38	C0000040
>	7A400	^	7E000	mapped: 3000	rw-
▼ .pdata	7A400	6E00	7E000	6D8C	40000040
>	81200	^	85000	mapped: 7000	r--
▼ .rsrc	81200	800	85000	6E0	40000040
>	81A00	^	86000	mapped: 1000	r--
▼ .reloc	81A00	1000	86000	EF4	42000040
>	82A00	^	87000	mapped: 1000	r--

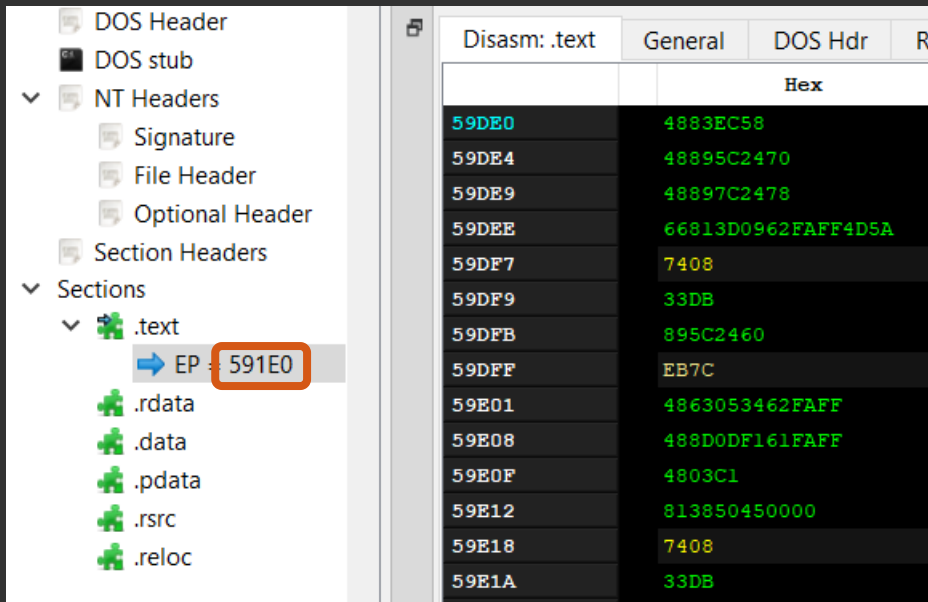
Remarques: le nom d'une section est un usage, aucune obligation. Le nombre classique de sections entre 4 et 7.

Seule la section « .text » devrait être « x » : **e**xécutable (droit de la portion mémoire)

Seule la section « .data » devrait être « w » : **w**ritable (variables locales)

Les autres sections sont normalement « r » : **r**ead only

# WIN PE : ENTRY POINT

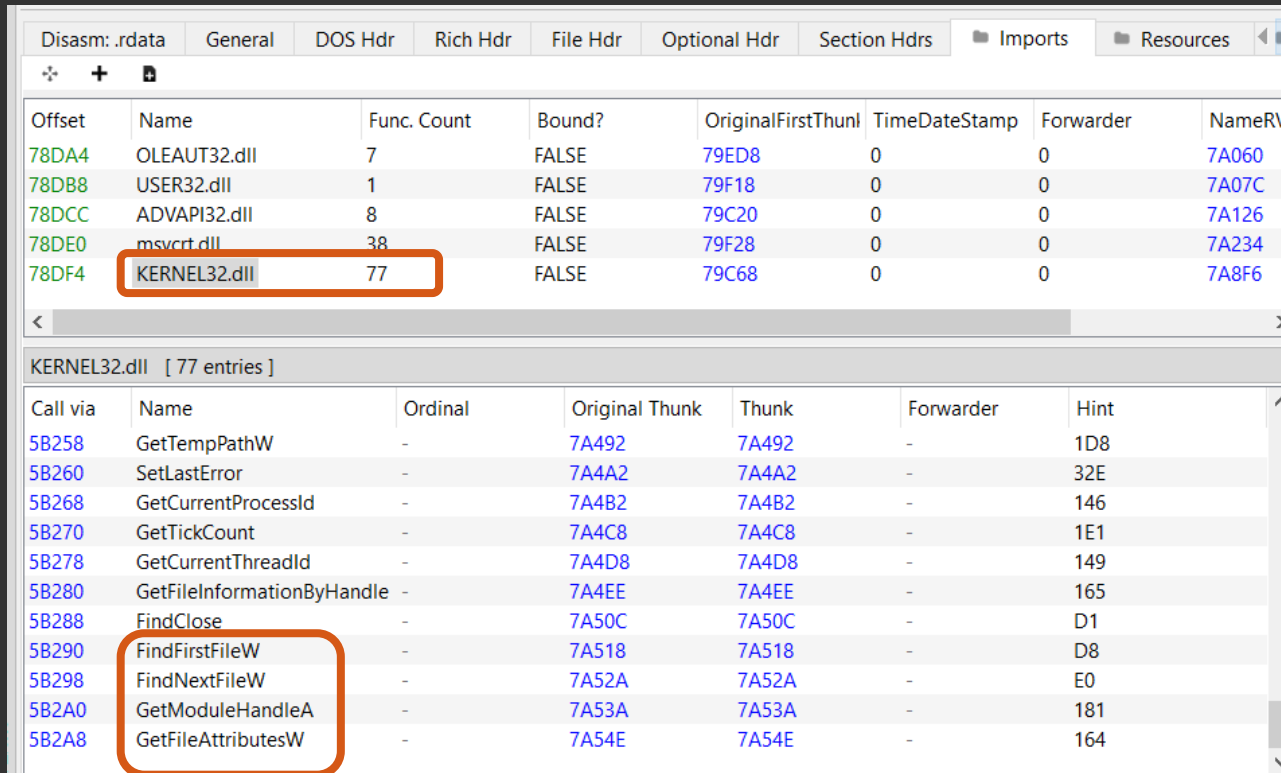


Name	Raw Addr.	Raw size	Virtual Addr.	Virtual Size	Characteristics
▼ .text	400	59E00	1000	59DC6	60000020
>	5A200	^	5B000	mapped: 5A000	r-x
▼ .rdata	5A200	1FA00	5B000	1F904	40000040
>	79C00	^	7B000	mapped: 20000	r--
▼ .data	79C00	800	7B000	2D38	C0000040
>	7A400	^	7E000	mapped: 3000	rw-
▼ .pdata	7A400	6E00	7E000	6D8C	40000040
>	81200	^	85000	mapped: 7000	r--
▼ .rsrc	81200	800	85000	6E0	40000040
>	81A00	^	86000	mapped: 1000	r--
▼ .reloc	81A00	1000	86000	EF4	42000040
>	82A00	^	87000	mapped: 1000	r--

Le **point d'entrée** se situe normalement dans la section code « .text ». Il s'agit de l'adresse où sera lancé le code du processus.

(**0x400** + 0xc00 = **0x1000**, 0x591e0 + 0xc00 = **0x59deo**)

# WIN PE : IMPORTS



Offset	Name	Func. Count	Bound?	OriginalFirstThunk	TimeDateStamp	Forwarder	NameRV
78DA4	OLEAUT32.dll	7	FALSE	79ED8	0	0	7A060
78DB8	USER32.dll	1	FALSE	79F18	0	0	7A07C
78DCC	ADVAPI32.dll	8	FALSE	79C20	0	0	7A126
78DE0	msvcrt.dll	38	FALSE	79F28	0	0	7A234
78DF4	KERNEL32.dll	77	FALSE	79C68	0	0	7A8F6

Call via	Name	Ordinal	Original Thunk	Thunk	Forwarder	Hint
5B258	GetTempPathW	-	7A492	7A492	-	1D8
5B260	SetLastError	-	7A4A2	7A4A2	-	32E
5B268	GetCurrentProcessId	-	7A4B2	7A4B2	-	146
5B270	GetTickCount	-	7A4C8	7A4C8	-	1E1
5B278	GetCurrentThreadId	-	7A4D8	7A4D8	-	149
5B280	GetFileInformationByHandle	-	7A4EE	7A4EE	-	165
5B288	FindClose	-	7A50C	7A50C	-	D1
5B290	FindFirstFileW	-	7A518	7A518	-	D8
5B298	FindNextFileW	-	7A52A	7A52A	-	E0
5B2A0	GetModuleHandleA	-	7A53A	7A53A	-	181
5B2A8	GetFileAttributesW	-	7A54E	7A54E	-	164

Imports: listes des DLL et fonctions utilisées

Ici *FindFirstFileW*,  
*FindNextFileW* et  
*GetFileAttributesW*

Fonctions légitimes pour 7zip  
= archiver une arborescence  
de fichiers

# WIN PE : MASQUER LES IMPORTS

Connaître l'API système permet de connaître les fonctions offertes par les librairies partagées (DLL ou .so)

L'import de telles DLL et l'usage de ces fonctions décrivent le fonctionnement interne d'un exécutable

C'est pour cela que les **malware cachent les imports**, et ré-implémentent un « loader » pour échapper à la surveillance sécurité.

# WIN PE : CHARGEMENT ET RÉOLUTION DYNAMIQUE

Plutôt que de laisser faire le loader Windows avec imports officiels (et visibles), les malware peuvent le faire eux-mêmes

Les **fonctions typiquement utilisées** sont:

- *VirtualAlloc/VirtualProtect* : allouer et changer les droits d'une portion mémoire
- *LoadLibraryA* pour y charger une DLL
- *GetProcAddress* pour résoudre / connecter les adresses des fonctions chargées avec le code de l'exécutable

Chargement et résolution dynamiques != malveillant. Contre exemple : plug-ins

# WIN PE : PACKER POUR MASQUER

Packer : Outil qui compresse ou chiffre un exécutable. L'opération inverse se réalise lors de l'exécution du binaire en version « packé ».

1- exe1 : entry point 1

2- exe2 : [ packer\_stub – exe1\_packé]

Lorsque l'on exécute exe2 [entry point2], le code « stub » alloue la mémoire, décompresse exe1 dans cet espace mémoire, reconstitue les imports, puis on exécute via l'entry point original (OEP=original entry point1)

<https://kindredsec.wordpress.com/2020/01/07/the-basics-of-packed-malware-manually-unpacking-upx-executables/>

# WIN PE : LANGUAGE D'ORIGINE ?

Quelques packers: WinRar, UPX, ASPack, ...

Un exécutable avec beaucoup d'imports est souvent basé sur un **interpréteur** : Autolt, PyInstaller, Delphi, .Net, Java, ... peut on retrouver le code source ? On ne voit plus les fonctions réellement utilisées.

Pour les langages **compilés**, quel est-il ? C, C++, Rust, Go ? Il faudra retrouver les fonctions, les structures de données.

Parfois des indices se trouvent dans la section « .rsrc »



# WINDOWS PE : RÉSUMÉ DES ANOMALIES POSSIBLES

- Le nombre, les noms et les propriétés (rwx) des sections sont-ils inhabituels ?
  - La section « .rsrc » est exécutable ?
  - Il y a plusieurs sections « writable » ?
  - « w » et « x », c'est louche !
  - Voir : <https://www.hexacorn.com/blog/2016/12/15/pe-section-names-re-visited/>
- Les tailles de chaque section sur disque et en mémoire sont-elles cohérentes ? Taille zéro en mémoire ?
- Le nombre de DLLs il est anormalement faibles ? <5 par exemple
- Le nombre de fonctions importées est-il anormalement faibles ? <5
- Les fonctions utilisées sont-elles compatibles avec le rôle de l'application ?
- Anomalies != malveillant

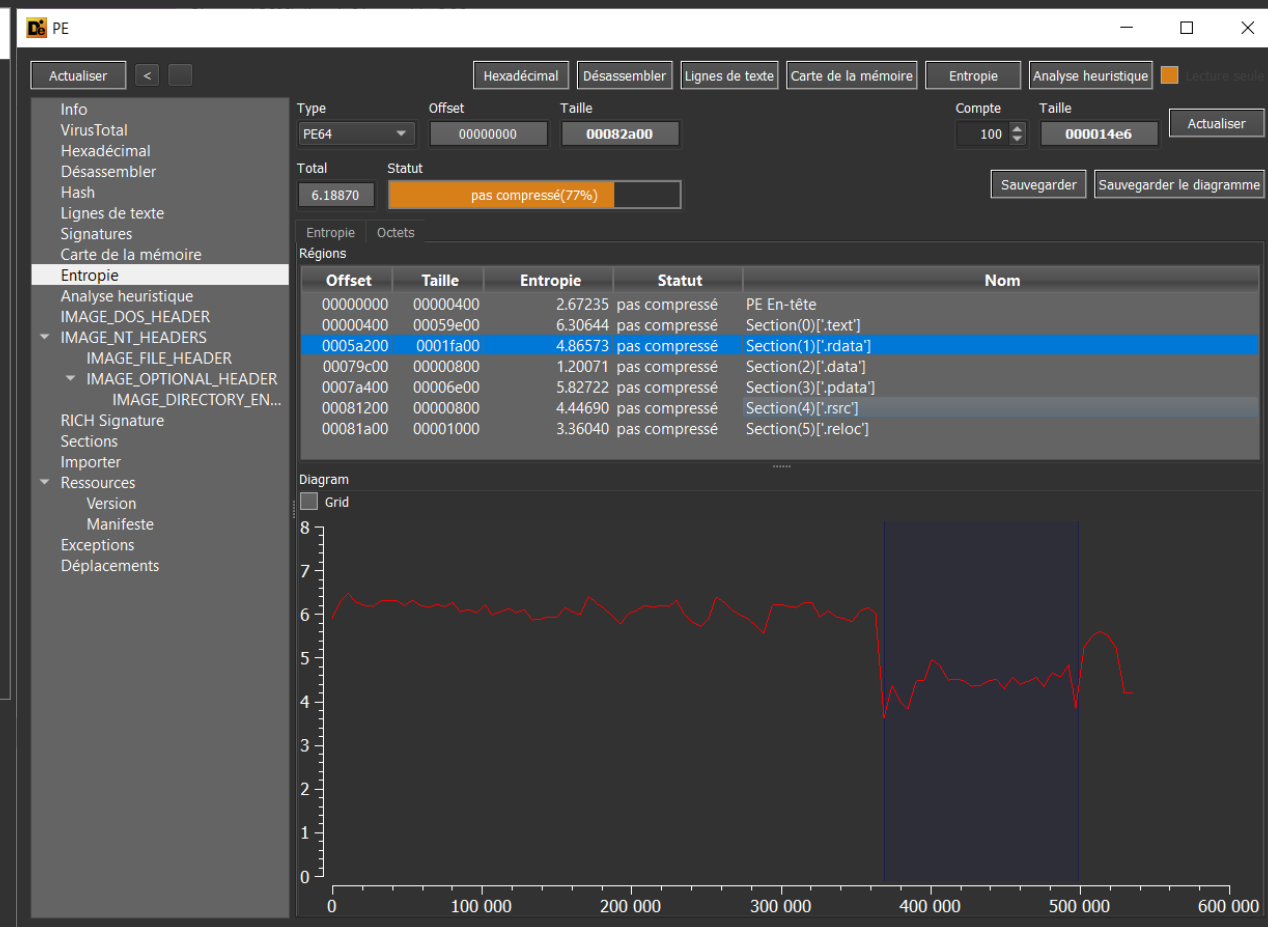
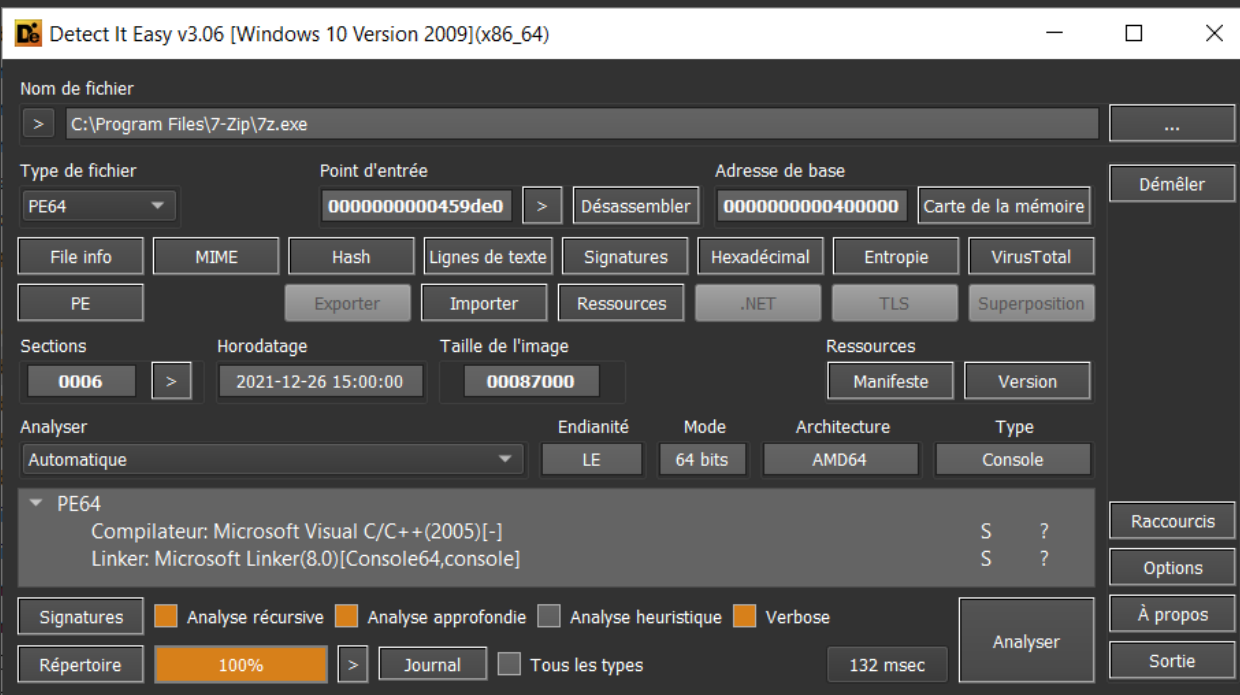
# WINDOWS PE : OUTILLAGE

## DiE (Detect It Easy)

- pour deviner les packers, analyser le PE, avec des signatures
- Versions console et GUI
- Hashes des différentes sections
- Calcul d'entropie
- Analyse de la structure PE

<https://github.com/horsicq/Detect-It-Easy>

# WINDOWS PE : DIE



# WINDOWS PE : PE-BEAR

Par @hasherezade

- pour explorer la structure et les propriétés d'un exécutable PE

<https://github.com/hasherezade/pe-bear>

<https://github.com/hasherezade/pe-bear/releases>

# ANALYSE STATIQUE : CAPA(BILITIES)

“extracts features from files, such as strings, disassembly, and control flow”. “Second, a logic engine finds combinations of features that are expressed in a common rule format. When the logic engine finds a match, capa reports on the capability described by the rule.”

<https://github.com/mandiant/capa>

Installer le binaire Windows: capa-v4.0.1-windows.zip

<https://github.com/mandiant/capa/releases>

# ANALYSE DYNAMIQUE : IDA FREEWARE

IDA Pro est l'outil de référence, surtout de par les possibilités d'extension (plug-ins) et d'automatisation (scripts).

Il existe une version gratuite, que nous allons utiliser en TD.

<https://hex-rays.com/ida-free/>