



# WINDOWS MALWARE HUNTER HANDBOOK

## Concepts et implementations

### PAGE DE SERVICE

**Référence :** n/a

**Plan de classement :** cybersecurity|malware|sysinternals

**Niveau de confidentialité :** corporate

#### Mises à jour

Version	Date	Auteur	Description du changement
1.0.0	19-01-2024	BOTTON Gaël	Création
1.1.0	30-01-2024	BOTTON Gaël	Handles   ProcessExplorer   ProcessMonitor
1.2.0	01-02-2024	BOTTON Gaël	Threads   Cas n°1
2.0.0	22-02-2024	BOTTON Gaël	Ajout de la V2
3.0.0	10-03-2024	BOTTON Gaël	Ajout de la V3
4.0.0	29-03-2024	BOTTON Gaël	Ajout de la V4

#### Validation

Version	Date	Nom	Rôle

#### Diffusion

Version	Date	Nom	Rôle

# SOMMAIRE

1	Rappel du contexte	3
2	Objectifs	3
3	Les process	3
3.1	Introduction des processus	3
3.2	Process Explorer	4
3.2.1	Surveillance des processus	5
3.2.2	Observation de la mémoire	6
3.2.3	Voir les ressources d'un processus	7
3.2.3.1	LES HANDLES	7
3.2.3.2	LES THREADS	8
3.3	Process Monitor	8
3.3.1	Méthode d'utilisation de Process Monitor	9
3.3.2	Cas d'utilisation n°1 - Home page Firefox	10
3.3.2.1	Identification de l'événement à capturer	10
3.3.2.2	Pose du piège	10
3.3.2.3	Capture du malware	11
3.3.3	Cas d'utilisation n°2 - DNS empoisonne	12
3.3.3.1	Identification de l'événement à capturer	12
3.3.3.2	Pose du piège	14
3.3.3.3	Capture du malware	14
4	Les malwares	15
4.1	Introduction sur les malwares	15
4.2	Introduction aux antivirus	16
4.3	Présentation de VirusTotal	16
4.4	Les API	17
4.5	Faire face aux malwares	18
4.5.1	Expression du besoin	18
4.5.2	Analyse de conception	18
4.5.3	Conception du script d'analyse	18
4.6	Déploiement	23
4.7	Test de script	25
5	Signature de code	27
5.1	Concepts	27
5.2	architecture et infrastructure (PKI)	27
5.2.1	Architecture de la PKI :	27
5.2.2	Infrastructure de la PKI :	28
5.3	Le cas particulier des PE sous Windows	28
5.4	Les outils en ligne de commande	29
5.4.1	SigCheck du SysInternals	29
5.4.2	SignTool du Windows SDK	30
5.5	Les outils en interface graphique	31
5.5.1	La MMC - Microsoft Management Console	31
5.5.2	Process explorer	32

5.6	OID	33
5.7	Par script Python avec PEFile	33
5.7.1	Analyse et conception	33
5.7.2	Conception du script	34
5.7.3	Test	36
5.7.4	Ajout des méthodes au robot d'analyse	37
6	Analyse de fichier	40
6.1	Fonctionnement d'un système de stockage	40
6.2	Les outils en ligne de commande	41
6.1.1	DISKPART	41
6.3	HXD - autopsie d'une clé USB FAT	42
6.4	Payload dans un Alternate Data Stream NTFS.	44
6.4.1	Ce qu'il faut retenir d'un ADS (Alternate Data Stream)	45
6.5	Ajout d'un système de vérification des ADS d'un fichier dans le robot d'analyse	46
7	La séquence de boot	48
7.1	Boot de la machine	48
7.1.1	BIOS vs UEFI	49
7.1.2	MBR - Master Boot RecordTable des partitions dans le MBR / GPT - Globally Unique	50
	Identifier Partition Table	50
7.1.3	BS - Boot Sector (BS)	51
7.2	La séquence de boot de Windows	52
7.2.1	BCDEdit	52
7.2.2	MSConfig	54
7.2.3	Autoruns	57
8	Bibliographie	59
9	Table des illustrations	59

## 1 RAPPEL DU CONTEXTE

---

NetWorking Solutions Inc. (NSI) est une Société de Services du Numérique (ESN1) qui conçoit, réalise et maintient les infrastructures matérielles et logicielles de ses clients. NSI a besoin d'un document permettant à ses employés de comprendre le fonctionnement interne d'une machine et de comprendre comment faire face à des malwares. De plus, nous verrons le fonctionnement d'un système de stockage, l'analyse de fichier et d'un boot machine.

## 2 OBJECTIFS

---

De nos jours la grande majorité des utilisateurs utilise des antivirus, mais nombreux restent victimes d'action malveillante. Dans ce document nous allons voir des procédures permettant de surveiller notre environnement et de contrer les virus pour ainsi limiter les comportements malveillants.

1. Comprendre le fonctionnement d'un processus
2. Prendre connaissance du logiciel Process Explorer
3. Apprendre à surveiller les processus
4. Prendre connaissance du logiciel Process Monitor
5. Localiser des comportements malveillants
6. Comprendre ce qu'est un malware.
7. Comprendre ce qu'est un antivirus et comment il fonctionne.
8. Comprendre ce qu'est VirusTotal.
9. Comprendre le fonctionnement d'une API et ses différents types.
10. Mettre en place une solution de neutralisation des fichiers indésirables.
11. Comprendre le fonctionnement d'un système de stockage et d'un boot machine.

## 3 LES PROCESS

---

### 3.1 INTRODUCTION DES PROCESSUS

Un processus du latin pro « vers l'avant » et cessus « aller » est un programme, une application en cours d'exécution. Plus précisément, c'est un ensemble ordonné d'instructions à exécuter, répondant à un certain schéma et aboutissant à un résultat. Ils sont chargés depuis la mémoire de masse (HDD<sup>1</sup>, SSD<sup>2</sup>) vers la mémoire vive pour être exécutés par le processeur. Chaque processus reste initialisé dans la mémoire vive jusqu'à son arrêt.

La mémoire vive (RAM<sup>3</sup>) offre un accès rapide aux données nécessaires pour l'exécution des processus. Une fois les instructions chargées dans la mémoire vive, le processeur peut les traiter et aboutir au résultat attendu.

Durant le fonctionnement de la machine, le processeur exécute plusieurs processus en parallèle. Pour se faire il réveille un processus pour venir l'exécuter un certain temps et ensuite venir le rendormir jusqu'à ce qu'il le réveille, pour ainsi laisser la place à l'exécution d'un autre processus ; il switch entre les processus initialisés. En d'autres termes chaque processus attend son tour pour être exécuté par le processeur.

Un processus peut créer et exécuter des fils d'exécution en parallèle de l'exécution du processus parent par le processeur. C'est ce que l'on appelle un Thread, c'est une unité d'exécution : entité capable d'exécuter des instructions de manière indépendante.

---

<sup>1</sup> Hard Disk Drive

<sup>2</sup> Solid State Drive

<sup>3</sup> Random Access Memory

De plus un processus possède des Handles. Les handles sont des références numériques qui permettent au processus d'interagir avec diverses ressources système (fichier, objets du registre, variable, socket réseau, etc).

Le schéma ci-dessous illustre de manière simplifiée, le cycle de vie d'un processus.

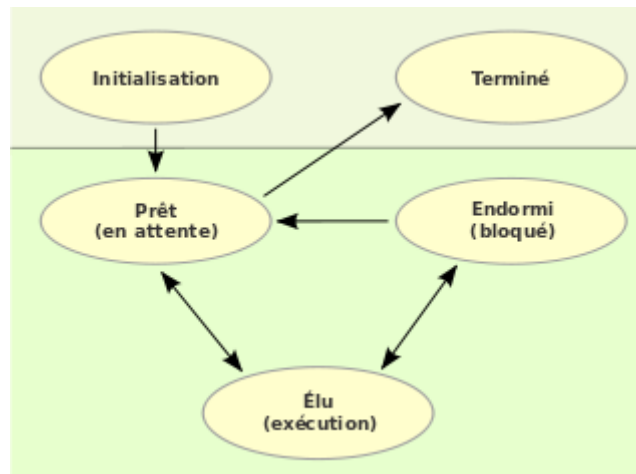


Figure 1 : Diagramme d'état d'un processus

Le fonctionnement d'un processus consiste en son initialisation (quand l'utilisateur clique sur l'exécutable de celui-ci) suivit de la transition vers l'état « prêt » qui vient ensuite passé par son exécution (un certain temps) puis être endormir par le processeur et attendre qu'il le réveil, le processus sort de cette boucle et va dans l'état « terminé » quand il est arrêté par le processeur.

### 3.2 PROCESS EXPLORER

Process Explorer fait partie d'une suite de logiciel « SysInternalsSuite » fourni gratuitement par Microsoft. Cette suite nous permet d'accéder à des informations clés pour la surveillance du système d'exploitation.

Process Explorer est un logiciel qui nous permet de surveiller les processus en temps réel. Il peut nous donner accès à des informations telles que le PID (Process Identifier), le nombre de Threads, de switch, la taille de la mémoire virtuelle, privée et beaucoup d'autres informations personnalisables.

Voici une image qui montre les informations renvoyées par Process Explorer :

Process	CPU	PID	Threads	Handles	Context Switches	Private Bytes	Working Set	Virtual Size	I/O Reads	I/O Writes	I/O Read Bytes	I/O Write Bytes	Verified Signer	Virus Total	Company Name	Description
Interrupts	1.50	n/a	0	0	113201681	0 K	0 K	0 K	0	0	0	0				Hardware Interrupts and DPCs
System Idle Process	89.58	0	32	0	21577463	60 K	8 K	8 K	0	0	0	0				
System	3.52	4	418	7919	6045253	52 K	152 K	4100 K	0	0	0	0				
Secure System	Suspended	172	0	0	0	184 K	50560 K	2352 K	0	0	0	0				
Registry		212	4	0	1613	16384 K	42812 K	188208 K	0	0	0	0				
smss.exe		592	2	58	823	1104 K	1088 K	2151718744 K	0	0	0	0				
csrss.exe		728	14	909	12753	2360 K	5280 K	2151775336 K	0	0	0	0				
svchost.exe		748	3	167	60	1452 K	6852 K	2151747720 K	0	0	0	0			Microsoft Corpor...	Processus hôte pour les services Windows
svchost.exe		752	13	491	5981	10100 K	24596 K	2151820688 K	16147	440	63.2 MB	889.8 KB			Microsoft Corpor...	Processus hôte pour les services Windows
SystemSettings.exe	Suspended	808	44	1432	5839	120404 K	1764 K	2165075732 K	248	4	8.9 MB	220 B			Microsoft Corpor...	Paramètres
wininit.exe		1152	2	145	287	1444 K	6280 K	2151748128 K	0	0	0	0				
csrss.exe	< 0.01	1172	16	974	297644	9716 K	6436 K	2151838980 K	0	0	0	0				
services.exe		1236	6	979	1110	6820 K	14992 K	2151773348 K	0	0	0	0				
lsass.exe		1264	1	61	40	1276 K	3312 K	4244884 K	0	0	0	0				
winlogon.exe		1284	3	308	749	2888 K	13144 K	2151830212 K	0	0	0	0				
lsass.exe		1300	11	1954	12502	12744 K	27812 K	2151827608 K	0	0	0	0			Microsoft Corpor...	Local Security Authority Process
svchost.exe		1408	15	456	2990	6444 K	14800 K	2151799564 K	0	0	0	0			Microsoft Corpor...	Processus hôte pour les services Windows
svchost.exe	< 0.01	1468	24	1436	10855	11872 K	31308 K	2151830528 K	0	0	0	0			Microsoft Corpor...	Processus hôte pour les services Windows
fontdrvhost.exe		1496	5	40	2265	1748 K	2956 K	2151747424 K	0	0	0	0				
fontdrvhost.exe		1500	5	40	22969	4940 K	8564 K	2151866780 K	0	0	0	0				
WUDFHost.exe		1580	7	251	905	1912 K	6380 K	2151749912 K	0	0	0	0				
svchost.exe		1636	12	1562	6695	8608 K	16076 K	2151767984 K	0	0	0	0			Microsoft Corpor...	Processus hôte pour les services Windows
svchost.exe		1652	9	488	8333	2988 K	9896 K	2151768904 K	0	0	0	0			Microsoft Corpor...	Processus hôte pour les services Windows
svchost.exe	< 0.01	1688	5	363	16290	3108 K	8504 K	2151768832 K	0	0	0	0			Microsoft Corpor...	Processus hôte pour les services Windows
dmn.exe	0.09	1776	29	1558	662314	174708 K	107292 K	2152367896 K	0	0	0	0				
svchost.exe		1856	1	110	57	1144 K	5048 K	2151743452 K	0	0	0	0			Microsoft Corpor...	Processus hôte pour les services Windows
svchost.exe		1860	7	338	1364	4428 K	12848 K	2151778312 K	0	0	0	0			Microsoft Corpor...	Processus hôte pour les services Windows
svchost.exe		1872	11	181	23067	6644 K	9344 K	2151757156 K	0	0	0	0			Microsoft Corpor...	Processus hôte pour les services Windows
svchost.exe		1888	3	123	70	1228 K	4936 K	2151745784 K	0	0	0	0			Microsoft Corpor...	Processus hôte pour les services Windows
svchost.exe		1904	2	207	55	1956 K	7900 K	2151753256 K	0	0	0	0			Microsoft Corpor...	Processus hôte pour les services Windows
svchost.exe		1928	3	235	53	2796 K	11996 K	2151770556 K	0	0	0	0			Microsoft Corpor...	Processus hôte pour les services Windows
svchost.exe		1972	7	306	1098	3620 K	13340 K	2151768100 K	0	0	0	0			Microsoft Corpor...	Processus hôte pour les services Windows
svchost.exe		2128	5	408	2093	6316 K	15728 K	2151767840 K	0	0	0	0			Microsoft Corpor...	Processus hôte pour les services Windows

Figure 2 : Capture d'écran du Logiciel Process Explorer

### Détail de chaque colonne :

- Process : Nom du processus.
- CPU : Ce chiffre représente le pourcentage d'utilisation de l'unité centrale par le processus sélectionné.
- PID : Identifiant du processus
- Threads : Le nombre de threads en cours d'exécution dans le processus sélectionné.
- Handles : Cette valeur indique le nombre de handles dont dispose le processus sélectionné.
- Context Switches : Nombre de fois que le processus a été endormi puis réveillé.
- Private Bytes : Quantité de mémoire allouée exclusivement au processus sélectionné et qui ne peut être partagée avec d'autres processus.
- Working Set : L'ensemble de travail est la partie de la mémoire d'un processus qui se trouve actuellement dans la RAM.
- Virtual Size : La quantité totale d'espace d'adressage virtuel réservée au processus sélectionné. Il comprend à la fois la RAM et l'espace de fichier de page.
- I/O Read/Write : Nombre d'opérations d'entrée/sortie (lecture ou écriture) effectuées par le processus sélectionné.
- I/O Read/Write Bytes : Quantité de données lues ou écrites sur le disque par le processus sélectionné.
- Verified Signer : Indique si le fichier exécutable du processus possède une signature numérique vérifiée.
- VirusTotal : Ce champ indique la note que VirusTotal a assignée à ce processus suivant sa malveillance.
- Compagny Name : Nom de l'entreprise qui a créé le processus.
- Description : Description du processus.

### **3.2.1 SURVEILLANCE DES PROCESSUS**

Lançons le logiciel « CalculatorApp.exe » fourni par Windows.

Process	CPU	PID	Threads	Handles	Context Switches	Private Bytes	Working Set	Virtual Size	I/O Reads	I/O Writes	I/O Read Bytes	I/O Write Bytes	Verified Signer	VirusTotal	Company Name	Description
CalculatorApp.exe	< 0.01	5296	16	577	142	32672 K	74796 K	5033112 K	234	1	178.4 KB	8 B			Microsoft Corpor...	Calculator

*Figure 3 : Informations du processus*

Ici, nous pouvons voir les informations du processus tel que son nom « CalculatorApp.exe », sont PID « 5296 » le nombre de Thread qu'il utilisé « 16 », etc.

Avec ces informations nous pouvons voir le comportement d'un processus dans le système d'exploitation et ainsi limité l'utilisation de certaines ressources ou bien faire des analyses d'utilisation pour voir potentiellement des processus qui utiliseraient de façons anormales certaines ressources.

### 3.2.2 OBSERVATION DE LA MEMOIRE

L'observation de la mémoire vive « RAM » peut-être très utilise pour repérer des processus gourmand en mémoire et comprendre comment ils impactent les performances du système. Via la surveillance de la mémoire, nous pouvons détecter des fuites de mémoire où un processus n'est pas correctement libéré, entraînant une augmentation constante de la consommation de celle-ci. L'observation de celle-ci permet aussi l'optimisation des ressources du système, qui peut ainsi limiter les utilisations anormales de ressource.

Cette image illustre les propriétés mémoires d'un processus en cours d'exécution :

Virtual Memory	
Private Bytes	38 008 K
Peak Private Bytes	38 328 K
Virtual Size	5 049 292 K
Page Faults	21 368
Page Fault Delta	0
Physical Memory	
Memory Priority	5
Working Set	80 548 K
WS Private	28 100 K
WS Shareable	52 416 K
WS Shared	52 416 K
Peak Working Set	80 992 K

Figure 4 : Propriété de La mémoire du processus

Peak Private Bytes : C'est la valeur maximale atteinte par la quantité de mémoire privée utilisée par un processus depuis le démarrage.

Page Faults : Un page fault se produit lorsqu'un programme tente d'accéder à une page de mémoire qui n'est pas actuellement en mémoire physique.

Page Fault Delta : Il représente le changement dans le nombre de défauts de page depuis la dernière mise à jour de l'outil de surveillance.

Physical Memory : La quantité de mémoire RAM physique installée sur l'ordinateur.

WS Private : La quantité de mémoire physique dans l'ensemble de travail « Working Set » d'un processus.

WS Shareable : La mémoire dans l'ensemble de travail qui peut être partagée avec d'autres processus.

WS Shared : La mémoire qui est partagée ave d'autre processus.

### 3.2.3 VOIR LES RESSOURCES D'UN PROCESSUS

#### 3.2.3.1 LES HANDLES

Lorsqu'un processus est en cours d'exécution, il peut mobiliser différentes ressources pour accomplir ses tâches. Ces ressources incluent des répertoires (directories), des fichiers, des bibliothèques de liens dynamiques (DLL), des clés de registre, et bien d'autres. Chaque fois qu'un processus ou l'un de ses threads souhaite accéder à l'une de ces ressources, le système d'exploitation attribue un handle à cette ressource spécifique.

Par exemple, lorsqu'un processus souhaite lire un fichier sur le disque dur, le système d'exploitation lui attribue un handle de fichier. Ce handle devient une référence abstraite au fichier, permettant au processus de lire, écrire ou effectuer d'autres opérations sur le fichier sans avoir à gérer directement les détails complexes du système de fichiers, un handle de répertoire permettrait l'accès à la structure d'un dossier spécifique, tandis qu'un handle de clé de registre fournirait un moyen de manipuler les paramètres système, les handles de DLL permettent aux processus de charger et d'utiliser des bibliothèques dynamiques, étendant ainsi leurs fonctionnalités.

En conclusion, les handles offrent une abstraction puissante qui simplifie l'interaction des processus et de leurs threads avec une variété de ressources système. Leur utilisation appropriée contribue à la robustesse des applications et des systèmes d'exploitation, en assurant une gestion efficace des ressources mobilisées.

Cette image illustre les propriétés des handles du processus en cours d'exécution :

Handles	
Handles	2 202
Peak Handles	2 202
GDI Handles	456
USER Handles	216

Figure 5 : Propriété des handles d'un processus

**Peak Handles** : Il s'agit du nombre maximal de handles que le processus a eu en même temps depuis son démarrage.

**GDI Handles** : Les Graphics Devices Interface (GDI) Handles représentent le nombre de handles alloués pour gérer les objets graphiques, tels que les fenêtres, les polices, les bitmaps, etc.

**USER Handles** : Les USER Handles représentent le nombre de handles utilisés pour gérer les objets d'interface utilisateur, tels que les fenêtres, les menus, les icônes, etc.

En résumé, Peak Handles donne une vision du point le plus élevé atteint par le nombre total de handles, tandis que GDI Handles et USER Handles se concentrent respectivement sur les handles liés aux objets graphiques et à l'interface utilisateur.

Dans cette seconde image prise sur le logiciel Process Explorer, j'ai sélectionné un processus pour afficher le détail de ses handles.

Type	Name	Handle
File	C:\Users\ixne\AppData\Local\Microsoft\Windows\Explorer\iconcache_idx.db	0x00000AC0
File	C:\Program Files\WindowsApps\Microsoft.LanguageExperiencePackfr-FR_22621.47.192.0...	0x00000B78
File	C:\Users\ixne\AppData\Local\Microsoft\Windows\Explorer\iconcache_32.db	0x00000BC0
Key	HKLM\SYSTEM\ControlSet001\Control\Session Manager	0x00000090
Key	HKLM\SYSTEM\ControlSet001\Control\Nls\Sorting\Versions	0x0000009C
Key	HKLM	0x000000D4

Figure 6 : détail des handles d'un processus



Comme dit précédemment quand un processus a besoin d'accéder à un fichier que ce soit en lecture ou en écriture, le système d'exploitation lui attribue un handles qui sera une référence à ce fichier. Sur l'image ci-dessus nous pouvons voir que le processus a des handles de type fichier (ce sont des références pour l'accès au fichier) et des handles de type key qui sont des références à des clés de registre.

### 3.2.3.2 LES THREADS

Comme expliqué précédemment, un processus peut créer et exécuter des fils d'exécution en parallèle de l'exécution du processus parent par le processeur. C'est ce que l'on appelle un Thread, c'est une unité d'exécution : entité capable d'exécuter des instructions de manière indépendante.

Il existe deux types principaux de threads : les threads utilisateur et les threads noyau. Les threads utilisateur sont gérés par la bibliothèque de threads au niveau de l'application, tandis que les threads noyau sont gérés par le noyau du système d'exploitation.

Les avantages des threads incluent l'amélioration des performances, la réactivité des applications et la simplification de la programmation concurrente.

Sur cette image prise encore une fois sur Process Explorer, nous pouvons voir les détails de chaque thread d'un processus.

State	Wait Reason	TID	User Time	Kernel Time	CPU	CPU Time	Start Time	Start Address	Base Pri
Waiting	WrUserRequest	15088	00:00:03	00:00:12	< 0.01	00:00:16	02/01/24 11:...	procexp64.exe+0xe1d78	13
Waiting	UserRequest	18336	00:00:19	00:00:02	< 0.01	00:00:22	02/01/24 11:...	procexp64.exe+0xf7a78	13
Running		1936	00:00:00	00:00:00	< 0.01	00:00:00	02/01/24 11:...	ntdll.dll!RtlClearThreadW...	13
Waiting	WrQueue	5112	00:00:00	00:00:00		00:00:00	02/01/24 11:...	ntdll.dll!RtlClearThreadW...	13
Waiting	WrQueue	16280	00:00:00	00:00:00		00:00:00	02/01/24 11:...	ntdll.dll!RtlClearThreadW...	13
Waiting	UserRequest	3948	00:00:00	00:00:00		00:00:00	02/01/24 11:...	CorperfmonExt.dll!Close...	13
Waiting	UserRequest	11204	00:00:00	00:00:00		00:00:00	02/01/24 11:...	procexp64.exe+0xf7a78	13
Waiting	WrQueue	8268	00:00:00	00:00:00		00:00:00	02/01/24 11:...	ntdll.dll!RtlClearThreadW...	13

Figure 7 : détail des threads d'un processus

Nous pouvons voir si un thread est en attente ou alors en cours d'utilisation, la priorité qui lui est accordée, son utilisation processeur, la raison pour laquelle il a été mis en attente, et bien d'autres informations complémentaires.

## 3.3 PROCESS MONITOR

Le logiciel Process Monitor est lui aussi un logiciel de la suite « SysInternalsSuites ». Il permet d'attraper les informations de ce que fait chaque processus (à quel endroit il écrit, lis, etc.). Il permet de surveiller l'activité des processus afin de vérifier si un processus est malveillant ou non.

Voici une image qui illustre les informations renvoyées par Process Monitor :

Time of Day	Process Name	PID	Operation	Path	Result	Detail
08:25:08,4011603	svchost.exe	2708	RegQueryKey	HKLM\System\CurrentControlSet\Services\Tcpip6\...	SUCCESS	Query: HandleTags, HandleTags: 0x0
08:25:08,4011685	svchost.exe	2708	RegOpenKey	HKLM\System\CurrentControlSet\Services\Tcpip6\...	SUCCESS	Desired Access: Read
08:25:08,4011777	svchost.exe	2708	RegCloseKey	HKLM\System\CurrentControlSet\Services\Tcpip6\...	SUCCESS	
08:25:08,4011868	svchost.exe	2708	RegOpenKey	HKLM	SUCCESS	Desired Access: Maximum Allowed, Granted Ac...
08:25:08,4011951	svchost.exe	2708	RegQueryKey	HKLM	SUCCESS	Query: HandleTags, HandleTags: 0x0
08:25:08,4012032	svchost.exe	2708	RegOpenKey	HKLM\System\CurrentControlSet\Services\Tcpip\...	REPARSE	Desired Access: Read

Figure 8 : Capture d'écran du Logiciel Process Monitor

### Détail de chaque colonne :

- Time of Day : L'heure à laquelle l'événement a été enregistré.
- Process Name : Nom du processus.
- PID : Identifiant du processus.
- Operation : Il spécifie le type d'opération effectuée par le processus, par exemple, lecture de fichier, écriture, etc.
- Path : Chemin du fichier ou de la ressource sur laquelle l'opération a été effectuée.
- Result : Il donne le résultat de l'opération.
- Detail : fournis des informations supplémentaires sur l'opération ou des détails spécifiques liés à l'événement.

### 3.3.1 METHODE D'UTILISATION DE PROCESS MONITOR

Pour surveiller un processus avec Process Monitor, il faut appliquer certains types de filtres sur un processus précis pour afficher uniquement les actions effectuées par celui-ci souhaitées.

#### Capturer des actions spécifiques à un processus :

J'ai créé un simple script en Python compilé en .exe « Exécutable », qui toute les une seconde va venir écrire dans le fichier « n.txt » la valeur 0 de type «string (chaîne de caractère)».

```
from time import sleep

while True:
    with open("./n.txt", "w") as f:
        f.write("0")
    sleep(1)
```

Figure 9 : capture d'écran du script Python

Dans un premier lieu, nous allons créer un filtre de type « Process Name » et « is » en spécifiant le nom du processus « python\_script.exe » et « include ».

Display entries matching these conditions:

Process Name	is	python_script.exe	then	Include
--------------	----	-------------------	------	---------

Figure 10 : Ajout d'un filtre de type Process Name

Ensuite, nous allons créer un second filtre, cette fois-ci de type « opération » et « is » en spécifiant le type d'opération « WriteFile » et « include », car nous savons que notre processus écrit dans un fichier.

Display entries matching these conditions:

Operation	is	WriteFile	then	Include
-----------	----	-----------	------	---------

Figure 11 : Ajout d'un filtre de type Opération

Enfin, nous pouvons lancer le processus et voir que Process Monitor capture bien les opérations de type « WriteFile » du processus python\_script.exe.

Time of Day	Process Name	PID	Operation	Path	Result	Detail
09:21:19.1706965	python_script...	18720	WriteFile	C:\Users\ixine\AppData\Local\Temp\_MEI187202\...	SUCCESS	Offset: 958464, Length: 4096
09:21:19.1707068	python_script...	18720	WriteFile	C:\Users\ixine\AppData\Local\Temp\_MEI187202\...	SUCCESS	Offset: 962560, Length: 4096
09:21:19.1707286	python_script...	18720	WriteFile	C:\Users\ixine\AppData\Local\Temp\_MEI187202\...	SUCCESS	Offset: 966656, Length: 4096
09:21:19.1707378	python_script...	18720	WriteFile	C:\Users\ixine\AppData\Local\Temp\_MEI187202\...	SUCCESS	Offset: 970752, Length: 4096
09:21:19.1707569	python_script...	18720	WriteFile	C:\Users\ixine\AppData\Local\Temp\_MEI187202\...	SUCCESS	Offset: 974848, Length: 4096
09:21:19.1707665	python_script...	18720	WriteFile	C:\Users\ixine\AppData\Local\Temp\_MEI187202\...	SUCCESS	Offset: 978944, Length: 4096
09:21:19.1707869	python_script...	18720	WriteFile	C:\Users\ixine\AppData\Local\Temp\_MEI187202\...	FAST IO DISALLOWED	Offset: 983040, Length: 4096
09:21:19.1708050	python_script...	18720	WriteFile	C:\Users\ixine\AppData\Local\Temp\_MEI187202\...	SUCCESS	Offset: 983040, Length: 4096, Priority: Normal
09:21:19.1708427	python_script...	18720	WriteFile	C:\Users\ixine\AppData\Local\Temp\_MEI187202\...	SUCCESS	Offset: 987136, Length: 4096
09:21:19.1708733	python_script...	18720	WriteFile	C:\Users\ixine\AppData\Local\Temp\_MEI187202\...	SUCCESS	Offset: 991232, Length: 4096
09:21:19.1708827	python_script...	18720	WriteFile	C:\Users\ixine\AppData\Local\Temp\_MEI187202\...	SUCCESS	Offset: 995328, Length: 4096
09:21:19.1708998	python_script...	18720	WriteFile	C:\Users\ixine\AppData\Local\Temp\_MEI187202\...	SUCCESS	Offset: 999424, Length: 4096
09:21:19.1709405	python_script...	18720	WriteFile	C:\Users\ixine\AppData\Local\Temp\_MEI187202\...	SUCCESS	Offset: 1003520, Length: 4096
09:21:19.1709514	python_script...	18720	WriteFile	C:\Users\ixine\AppData\Local\Temp\_MEI187202\...	SUCCESS	Offset: 1007616, Length: 4096
09:21:19.1709937	python_script...	18720	WriteFile	C:\Users\ixine\AppData\Local\Temp\_MEI187202\...	SUCCESS	Offset: 1011712, Length: 4096
09:21:19.1710040	python_script...	18720	WriteFile	C:\Users\ixine\AppData\Local\Temp\_MEI187202\...	SUCCESS	Offset: 1015808, Length: 4096
09:21:19.1710309	python_script...	18720	WriteFile	C:\Users\ixine\AppData\Local\Temp\_MEI187202\...	SUCCESS	Offset: 1019904, Length: 4096
09:21:19.1710556	python_script...	18720	WriteFile	C:\Users\ixine\AppData\Local\Temp\_MEI187202\...	SUCCESS	Offset: 1024000, Length: 4096

Figure 12 : Capture des opérations d'un processus

### 3.3.2 CAS D'UTILISATION N°1 – HOME PAGE FIREFOX

#### 3.3.2.1 IDENTIFICATION DE L'ÉVÉNEMENT À CAPTURER

Dans ce cas d'utilisation, un client créer un ticket support, car la page d'accueil de son navigateur Web Firefox a été modifié intentionnellement et il pense que c'est une page de phishing qui imite la vraie page d'accueil pour faire croire à son utilisateur qu'elle est légitime et ainsi récupérer ses identifiants de connexion.

On sait que Firefox est la victime, mais on ne sait pas qui est le responsable. Pour ça nous allons capturer des événements qui vont nous amener vers le processus responsable.

Il y a donc deux événements à capturer dans un premier temps :

- Type = process\_name, Relation = is, Value = FireFox.exe
- Type = operation, Relation = is, Value = ReadFile

Avec ces deux filtres d'événements, on va pouvoir capturer tous les événements de lecture de fichier du processus FireFox.exe et savoir où se situe le fichier de préférence utilisateur de Firefox.

Une fois que nous aurons trouvé le fichier de préférence utilisateur de Firefox, nous pourrions capturer les événements des processus qui modifient ce fichier et ainsi trouver le processus malveillant.

Événement à capturer dans un second temps :

- Type = operation, Relation = is, Value = WriteFile
- Type = path, Relation = is, Value = chemin d'accès du fichier de préférence utilisateur de Firefox

#### 3.3.2.2 POSE DU PIEGE

En premiers lieux nous allons mettre les filtres :

- Type = process\_name, Relation = is, Value = FireFox.exe
- Type = operation, Relation = is, Value = ReadFile

Pour capturer toutes les lectures de fichier du processus, Firefox.exe est trouvé où est situé le fichier de préférence utilisateur de Firefox.

Cette image illustre les filtres qui viennent d'être ajoutés.



Column	Relation	Value	Action
<input checked="" type="checkbox"/>  Process Name	is	firefox.exe	Include
<input checked="" type="checkbox"/>  Operation	is	ReadFile	Include

Figure 13 : Ajout des filtres

On peut voir qu'au lancé de Firefox il lit un fichier nommé « pref.js ». C'est un fichier de type « JavaScript » qui contient les informations de préférence de l'utilisateur (donc sa page d'accueil).

Cette image illustre le chemin d'accès du fichier préférence utilisateur de Firefox obtenu lors de la capture.



12:07:08,2136339  firefox.exe 11492  ReadFile C:\Users\Vixne\AppData\Roaming\Mozilla\Firefox\Profiles\x3s3qkzj.default-release\prefs.js

Figure 14 : Screenshot de la capture de Lecture du fichier de préférence utilisateur de Firefox

Si on ouvre le fichier « pref.js » on peut voir que l'on a bien une valeur (URL du site web accueil) qui est associée à la valeur « browser.startup.homepage » qui est la page d'accueil de démarrage. Actuellement l'URL de la page d'accueil de démarrage est [www.google.com](http://www.google.com).

Cette image illustre les valeurs de la page d'accueil

`user_pref("browser.startup.homepage", "www.google.com");`

Figure 15 : Valeur de La page d'accueil de démarrage

Une fois que nous savons quel est le fichier qui contient les préférences de la page d'accueil, nous pouvons poser le piège pour capturer le processus malveillant.

Nous allons supprimer tous nos filtres et venir en créer deux nouveaux qui vont permettre d'enregistrer tous les événements d'écriture sur le fichier « pref.js », donc potentiellement de modification des valeurs de la page d'accueil.

Cette image illustre les filtres qui viennent d'être ajoutés.



Column	Relation	Value	Action
<input checked="" type="checkbox"/>  Operation	is	WriteFile	Include
<input checked="" type="checkbox"/>  Path	is	C:\Users\Vixne\AppData\...	Include

Figure 16 : Ajout des filtres

### 3.3.2.3 CAPTURE DU MALWARE

Dans notre cas le processus malveillant sera un programme python qui viendra modifier la valeur de la page d'accueil.

Cette image montre le code du script python qui modifiera l'URL de la page d'accueil.

```
import re

new_value = 'www.kollabsound.com'

with open("C:/Users/lixne/AppData/Roaming/Mozilla/Firefox/Profiles/x3s3qkzj.default-release/prefs.js", "r") as file:
    content = file.read()

new_content = re.sub(r'user_pref\(("browser.startup.homepage", "(.*?)");', f'user_pref("browser.startup.homepage", "{new_value}");', content)

with open("C:/Users/lixne/AppData/Roaming/Mozilla/Firefox/Profiles/x3s3qkzj.default-release/prefs.js", "w") as file:
    file.write(new_content)
```

Figure 17 : Script python responsable de la modification de l'URL de la page d'accueil

Cette image montre l'événement de modification du fichier « pref.js » capturé par Process Monitor.

Time of Day	Process Name	PID	Operation	Path
12:57:58,9435691	malware.exe	21564	WriteFile	C:\Users\lixne\AppData\Roaming\Mozilla\Firefox\Profiles\x3s3qkzj.default-release\prefs.js

Figure 18 : Événement de modification du fichier "pref.js"

Après avoir exécuté le malware, on peut voir que notre logiciel de capture Process Monitor a capturé un événement d'écriture sur le fichier « pref.js » par un processus nommé « malware.exe » avec le PID 21564. Donc on peut en déduire que le malware qui a compromis la page d'accueil de Firefox est « malware.exe ».

### 3.3.3 CAS D'UTILISATION N°2 – DNS EMPOISONNE

#### 3.3.3.1 IDENTIFICATION DE L'ÉVÉNEMENT À CAPTURER

Dans ce cas d'utilisation, un client crée un ticket support, car il pense que son DNS par défaut a été modifié par un programmeur malveillant pour être remplacé par un DNS empoisonné.

Nous ne savons pas où sont stockées les informations de configuration des interfaces réseau. Est-ce qu'elles se trouvent dans le registre, dans les fichiers système ?

Pour ça nous allons faire une capture d'événement avec ces filtres sur le logiciel Process Monitor :

- Type = Event Class, Relation = is, Value = File System
- Type = Event Class, Relation = is, Value = Registre
- Type = Category, Relation = is, Value = Write

Ces trois filtres vont permettre d'effectuer une capture sur tous les événements d'écriture dans les fichiers du système et dans le registre.

Cette image illustre les filtres qui viennent d'être ajoutés :

Column	Relation	Value	Action
<input checked="" type="checkbox"/> Event Class	is	File System	Include
<input checked="" type="checkbox"/> Event Class	is	Registry	Include
<input checked="" type="checkbox"/> Category	is	Write	Include

Figure 19 : Ajout des filtres

Ensuite nous allons aller dans l'outil « ncpa.cpl » et venir modifier le DNS de notre interface réseau.

Cette image illustre une modification effectuée sur le DNS de l'interface réseau.

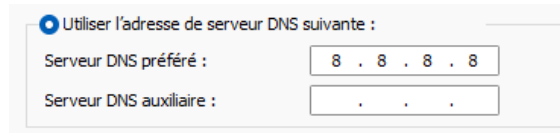


Figure 20 : Modification du DNS

Cette image illustre le résultat de la capture de la modification du DNS.

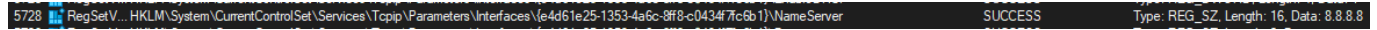


Figure 21 : Événement de modification du DNS

Nous pouvons voir que Process Monitor a bien capturé un événement de type « RegSetValue » de la clé HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\Interfaces\{e4d61e25-1353-4a6c-8ff8-c0434f7fc6b1}\NameServer et que la valeur vient d'être modifiée par « 8.8.8.8 ».

Avec cette information on sait que les configurations des interfaces réseau sont stockées dans le registre à la clé :

HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\Interfaces

Chaque machine possède un registre (un registre est une base de données hiérarchique utilisée par les systèmes d'exploitation. Le registre stocke des informations essentielles sur la configuration du système, les applications installées et les utilisateurs).

Cette clé contient toutes les interfaces réseau que possède notre machine (carte Wi-Fi, Ethernet, Virtuelle, etc.) sous la forme d'un UUID4 (Universally unique identifier, version 4).

Cette image illustre les clés contenues dans la clé « Interfaces » du registre Windows.

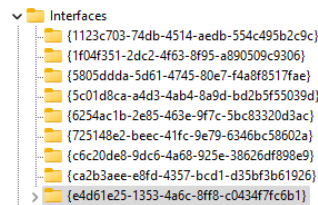


Figure 22 : Interface réseau du registre Windows

Maintenant , si nous allons voir la valeur de la clé NameServer de l'interface réseau qui vient d'être modifiée, on peut s'apercevoir que sa valeur est bien « 8.8.8.8 ».

Cette image illustre les valeurs d'une configuration d'une interface réseau dans le registre.

Nom	Type	Données
(par défaut)	REG_SZ	(valeur non définie)
AddressType	REG_DWORD	0x00000000 (0)
DhcpConnForce...	REG_DWORD	0x00000000 (0)
DhcpDefaultGat...	REG_MULTI_SZ	192.168.1.1
DhcpDomain	REG_SZ	Cisco
DhcpGatewayH...	REG_BINARY	c0 a8 01 01 06 00 00 d8 67 d9 d1 b4 2a
DhcpGatewayH...	REG_DWORD	0x00000001 (1)
DhcpInterfaceO...	REG_BINARY	fc 00 00 00 00 00 00 00 00 00 00 00 00 00 fa d8 ...
DhcpIPAddress	REG_SZ	192.168.1.211
DhcpNameServer	REG_SZ	172.31.1.4 172.31.1.6
DhcpNetworkHint	REG_SZ	7596669f5394f4f5554594
DhcpServer	REG_SZ	192.168.1.1
DhcpSubnetMask	REG_SZ	255.255.255.0
DhcpSubnetMas...	REG_MULTI_SZ	255.255.255.0
Domain	REG_SZ	
EnableDHCP	REG_DWORD	0x00000001 (1)
IsServerNapAware	REG_DWORD	0x00000000 (0)
Lease	REG_DWORD	0x00015180 (86400)
LeaseObtainedTL...	REG_DWORD	0x65bdcdaaf (1706875567)
LeaseTerminates...	REG_DWORD	0x65bdc2cf (1706961967)
NameServer	REG_SZ	8.8.8.8
RegisterAdapter...	REG_DWORD	0x00000000 (0)
RegistrationEna...	REG_DWORD	0x00000001 (1)
T1	REG_DWORD	0x65bd7533 (1706915123)
T2	REG_DWORD	0x65bd93c3 (1706947523)

Figure 23 : Valeur de l'interface réseau Wi-Fi

Pour trouver le malware qui est à l'origine de ces actions malveillantes, nous allons avec le logiciel Process Monitor, venir capturer les événements qui viennent modifier la valeur de cette clé de registre.

### 3.3.3.2 POSE DU PIEGE

La pose du piège consiste à paramétrer des filtres spécifiques sur Process Monitor, pour capturer les processus qui écrivent des valeurs sur cette clé.

Pour ce faire nous allons utiliser ces filtres :

- Type = Event Class, Relation = is, Value = Registry
- Type = Operation, Relation = is, Value = RegSetValue
- Type = Path, Relation = more than, Value = HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\Interfaces

Cette image illustre les filtres qui viennent d'être ajoutés.

Column	Relation	Value	Action
Operation	is	RegSetValue	Include
Path	more than	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\Interfaces\	Include
Event Class	is	Registry	Include

Figure 24 : Ajout des filtres

Avec ces filtres nous allons capturer tous les événements qui viennent modifier une valeur de la configuration des interfaces réseau dans le registre.

### 3.3.3.3 CAPTURE DU MALWARE

Pour la démonstration, je vais utiliser un script python qui fera office de malware et qui va venir modifier la valeur du DNS d'une configuration d'une interface réseau du registre Windows.

Cette image illustre le code Python du malware.

```
import subprocess

registry_path = r'HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\Interfaces\{e4d61e25-1353-4a6c-8ff8-c0434f7fc6b1}'
value_name = 'NameServer'
value_type = 'REG_SZ'
new_value = '1.1.1.1'

command = f'reg add "{registry_path}" /v {value_name} /t {value_type} /d {new_value} /f'

subprocess.run(command, shell=True)
```

Figure 25 : Script du malware Python

Quand le script va être exécuté, il va remplacer la valeur du DNS de l'interface spécifiée (8.8.8.8) par « 1.1.1.1 ».

Cette image illustre l'événement de modification de la valeur de la clé du DNS de l'interface réseau capturée par Process Monitor.

14.07.05.9725748	poisonned_d_...	24404	RegSetValue	HKLM\System\CurrentControlSet\Services\Tcpip\Parameters\Interfaces\{e4d61e25-1353-4a6c-8ff8-c0434f7fc6b1}\NameServer	SUCCESS	Type: REG_BINARY, Length: 24, Data: A4 4
14.07.06.0247370	reg.exe	24452	RegSetValue	HKLM\System\CurrentControlSet\Services\Tcpip\Parameters\Interfaces\{e4d61e25-1353-4a6c-8ff8-c0434f7fc6b1}\NameServer	SUCCESS	Type: REG_SZ, Length: 16, Data: 1.1.1.1

Figure 26 : Capture de La modification du DNS

On peut voir que Process Monitor a capturé un événement de modification de la clé qui contient la valeur du DNS. Dans le premier événement, on peut voir un processus nommé « poisonned\_dns.exe » (le malware python compilé) qui exécute une invite de commande « cmd.exe » puis un deuxième événement où un processus nommé « reg.exe » vient modifier la valeur de la clé NameServer (DNS) par « 1.1.1.1 ». On peut donc en déduire que le script python à exécuter la commande « reg add » via l'invite de commande qui a donc fait appel au processus « reg.exe » pour venir modifier la valeur. Le processus « poisonnd\_dns.py » est à l'origine de cette modification, donc c'est le malware.



## 4 LES MALWARES

---

### 4.1 INTRODUCTION SUR LES MALWARES

Un « malware » est un terme générique qui fait référence à un logiciel ou à un script conçu pour endommager, perturber, voler des données ou prendre le contrôle d'un système informatique sans l'autorisation du propriétaire de la machine.

Dans les malwares nous retrouvons sept grandes catégories.

**Virus** : Un virus est un programme malveillant qui s'attache à des fichiers exécutables ou à des secteurs de démarrage et se propage en infectant d'autres fichiers lorsqu'ils sont exécutés.

**Vers** : Contrairement aux virus, les vers sont des programmes autonomes qui se propagent d'un ordinateur à un autre via des réseaux, souvent en exploitant des failles de sécurité.

**Cheval de Troie (Trojan)** : Les chevaux de Troie sont des programmes qui semblent être légitimes, mais contiennent en réalité un code malveillant. Ils peuvent permettre à un attaquant d'accéder à distance à un système, de voler des informations sensibles ou de causer d'autres dommages.

**Logiciel espion (Spyware)** : Les logiciels espions sont conçus pour collecter des informations sur les activités d'un utilisateur sans son consentement, comme les habitudes de navigation sur Internet ou les données personnelles.

**Logiciel publicitaire (Adware)** : Les logiciels publicitaires affichent des publicités non sollicitées sur un système, souvent dans le but de générer des revenus publicitaires pour les créateurs de logiciel malveillant.

**Ransomware** : Ce type de malware crypte les fichiers d'un utilisateur et demande ensuite une rançon pour les déchiffrer, généralement en utilisant des cryptomonnaies.

**Rootkit** : Un rootkit est un ensemble de programmes ou de scripts conçus pour cacher et maintenir l'accès non autorisé à un ordinateur, souvent en modifiant le système d'exploitation ou d'autres logiciels.



## 4.2 INTRODUCTION AUX ANTIVIRUS

Un antivirus est un logiciel conçu pour détecter, prévenir et éliminer les logiciels malveillants (malwares) sur un ordinateur ou un réseau informatique. Son objectif principal est de protéger les utilisateurs contre les menaces potentielles telles que les virus, les vers, les chevaux de Troie, les logiciels espions, les logiciels publicitaires et les ransomwares.

Ces logiciels de détections de virus se basent en général sur des virus à signature numérique connue.

Voici comment cela fonctionne :

**Détection des signatures** : Les antivirus utilisent des bases de données de signatures de malware. Ces signatures sont essentiellement des empreintes digitales uniques associées à des logiciels malveillants connus. Lorsque l'antivirus analyse les fichiers sur votre système, il compare les empreintes digitales des fichiers avec celles de sa base de données pour détecter les malwares connus.

**Correspondance des signatures** : Si un fichier sur votre système correspond à une signature de malware répertoriée dans la base de données de l'antivirus, il est identifié comme étant malveillant.

**Action de l'antivirus** : Une fois qu'un malware est détecté, l'antivirus prend généralement des mesures pour le neutraliser. Cela peut inclure la mise en quarantaine du fichier infecté, la suppression complète du malware ou d'autres actions pour empêcher sa propagation et son exécution.

**Mises à jour de la base de données** : Les bases de données de signatures des antivirus sont constamment mises à jour pour inclure de nouvelles menaces à mesure qu'elles sont découvertes, cela permet à l'antivirus de rester efficace contre les nouveaux malwares.

## 4.3 PRESENTATION DE VIRUSTOTAL

VirusTotal est un service en ligne gratuit qui fournit une plateforme pour l'analyse de fichiers et d'URL afin de détecter les malwares. Fondé en 2004 par Hispasec Sistemas, une société de sécurité informatique basée en Espagne, VirusTotal a été acquis par Google en 2012. Il est désormais géré par Google Chronicle, une filiale de Google Cloud.

### **Caractéristiques principales de VirusTotal :**

**Analyse multi-moteurs** : VirusTotal utilise plusieurs moteurs antivirus et anti-malwares pour analyser les fichiers et les URL soumis par les utilisateurs. Cela permet d'obtenir une vue d'ensemble de la détection des malwares par différents fournisseurs.

**Interface utilisateur conviviale** : VirusTotal propose une interface utilisateur simple et intuitive, ce qui le rend facile à utiliser même pour les utilisateurs moins expérimentés.

**Analyse de fichiers et d'URL** : Les utilisateurs peuvent soumettre des fichiers exécutables, des documents, des archives compressées et des URL à VirusTotal pour analyse. Le service vérifie si ces éléments sont infectés par des malwares connus.

**Rapports détaillés** : Une fois l'analyse terminée, VirusTotal fournit des rapports détaillés indiquant les résultats de chaque moteur antivirus utilisé. Les utilisateurs peuvent consulter ces rapports pour obtenir des informations sur la détection des malwares et les signatures associées.

**API** : VirusTotal propose une API publique qui permet aux développeurs d'intégrer les fonctionnalités d'analyse de malwares dans leurs propres applications ou services.

## 4.4 LES API

Dans la suite du document, nous allons employer le mot API.

Voici une explication pour une compréhension claire de ce qu'est une API.

Une API, ou Interface de Programmation d'application (en anglais, Application Programming Interface) est un ensemble de règles, de protocoles et d'outils permettant à différents logiciels de communiquer entre eux. En d'autres termes, une API définit les méthodes et les interactions disponibles pour accéder aux fonctionnalités d'un logiciel ou d'un service depuis un autre logiciel, souvent de manière programmable.

Points clés pour comprendre ce qu'est une API :

**Interface** : Comme son nom l'indique, une API est une interface qui définit la manière dont deux systèmes informatiques interagissent entre eux. Elle expose un ensemble de fonctionnalités ou de services d'un logiciel, permettant à d'autres logiciels de les utiliser de manière contrôlée et sécurisée.

**Abstraction** : Une API permet d'abstraire la complexité interne d'un système. Plutôt que de devoir comprendre les détails techniques et les implémentations sous-jacentes, les développeurs peuvent utiliser l'API pour accéder aux fonctionnalités souhaitées de manière simplifiée.

**Réutilisabilité** : Les API favorisent la réutilisabilité du code. Plutôt que de réinventer la roue à chaque fois qu'une fonctionnalité est nécessaire, les développeurs peuvent utiliser des API existantes pour intégrer des fonctionnalités éprouvées dans leurs propres applications.

**Interopérabilité** : Les API permettent à des systèmes développés sur des plateformes différentes, utilisant des langages de programmation différents, voire fonctionnant sur des matériels différents, de communiquer entre eux de manière standardisée.

Une API agit en tant qu'intermédiaire entre deux logiciels, permettant à l'un d'accéder aux fonctionnalités ou aux données de l'autre de manière contrôlée et sécurisée. Elle offre un ensemble de méthodes et d'interfaces standardisées qui permettent aux développeurs d'intégrer facilement les fonctionnalités d'un logiciel dans un autre, ou de permettre à différents logiciels de communiquer entre eux de manière efficace. En somme, une API facilite l'interopérabilité et la communication entre les systèmes informatiques.

Il existe plusieurs types d'API.

**API Locales (ou natives)** : Ces API sont spécifiques à une plateforme ou à un langage de programmation particulier. Elles permettent aux programmes informatiques de communiquer avec les ressources locales de l'ordinateur, telles, les que le système de fichiers périphériques matériels, etc. Exemples : les API Windows pour les applications Windows, les API POSIX pour les systèmes Unix/Linux, les API Android pour les applications mobiles Android.

**API RESTful (Representational State Transfer)** : Basées sur l'architecture REST, ces API utilisent les méthodes standard du protocole HTTP (GET, POST, PUT, DELETE, etc.) pour accéder et manipuler des ressources / données. Elles sont généralement utilisées pour créer des services web et des applications web. Elles sont largement utilisées en raison de leur simplicité, de leur évolutivité et de leur compatibilité avec le web. Exemples : L'API Twitter, GitHub, Google Maps, VirusTotal, etc. Toutes ces API permettent de manipuler des données de ces plateformes.

**API SOAP (Simple Object Access Protocol)** : Ces API utilisent le protocole SOAP pour définir la structure des messages échangés entre le client et le serveur. Elles sont souvent utilisées dans les environnements d'entreprise pour intégrer des systèmes informatiques hétérogènes et pour la communication entre applications. Elles sont plus complexes que les API RESTful en raison de leur utilisation de XML et de WSDL (Web Services Description Language). Exemples : les services web basés sur SOAP dans les environnements .NET, les services web Apache Axis, etc.

Ces types d'API sont les plus courants, mais il en existe d'autres, utilisés dans des contextes spécifiques et très précis.

## 4.5 FAIRE FACE AUX MALWARES

### 4.5.1 EXPRESSION DU BESOIN

Nous allons créer un script d'analyse autonome. Ce script permettra d'analyser un ou plusieurs répertoires ciblés, de A à Z, pour vérifier qu'il ne contient pas de fichier malveillant. Le/les répertoires à analysés seront configurable manuellement par l'administrateur dans un fichier appelé « config.cfg » que le script lira pour effectuer une analyse sur tous les répertoires présents dans celui-ci. Si un fichier malveillant est détecté par notre script, alors le fichier sera déplacé dans un répertoire de quarantaine et un SMS sera envoyé à l'administrateur pour lui faire part des résultats de l'analyse qui vient d'être effectuée. De plus pour chaque analyse, le script enregistrera dans un dossier « data » un fichier nommé « nom-du-fichier-jour-mois-année-heure-min-sec.dat », le résultat complet de l'analyse. Enfin à chaque fois qu'un fichier sera détecté comme malveillant et déplacé en quarantaine, tout sera enregistré dans fichier log (Journal d'activités).

### 4.5.2 ANALYSE DE CONCEPTION

Pour réaliser ce script, nous utiliserons le langage de programmation Python, réputé pour sa praticité dans ce type de programme en raison de sa grande communauté et de la disponibilité de nombreuses bibliothèques pour les développeurs, ainsi que de sa puissance d'analyse.

En plus du langage Python, nous allons utiliser l'API RESTful de VirusTotal qui nous permettra d'envoyer des requêtes d'analyse de fichier au serveur de VirusTotal. De plus nous utiliserons l'API RESTful de « vonage » qui nous servira à envoyer des SMS à l'administrateur en cas de détection de fichier malveillant. Nous utiliserons aussi le module « os » qui est une API de type locale faisant le lien entre notre application et le système d'exploitation pour gérer des fichiers, des répertoires, etc.

Pour ce faire nous allons développer le script en « programmation fonctionnelle », c'est-à-dire que chaque fonction aura une tâche spécifique et agira comme un bloc de construction modulaire dans notre script. Plutôt que d'écrire un script qui s'exécute de manière linéaire, nous allons définir des fonctions qui peuvent être appelées indépendamment les unes des autres, favorisant ainsi la réutilisabilité du code et la clarté de la logique de programmation.

### 4.5.3 CONCEPTION DU SCRIPT D'ANALYSE

Dans un premier temps nous allons installer le module vt (API de VirusTotal) à l'aide de la commande « pip install vt-py » dans le terminal.

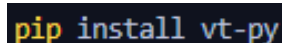


Figure 27 : installation de la librairie vt-py

Dans un second temps nous allons installer le module vonage (API de Vonage) à l'aide de la commande « pip install vonage » dans le terminal.

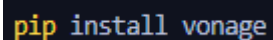


Figure 28 : installation de la librairie vonage

Une fois le module installé nous allons importer tous les modules que nous aurons besoin pour la réalisation de ce script. Cela comprend le module vt, os (pour la manipulation du système d'exploitation), shutil (pour la manipulation de fichier), datetime (pour la gestion du temps) et vonage.

```
#Import
import vt, os, shutil, datetime, vonage
```

Figure 29 : importation des modules

Une fois tous les modules importés dans notre script nous allons créer une catégorie « config » qui comprendra quatre constantes de type string. Ces constantes stockent le chemin d'accès du fichier de configuration et les chemins d'accès de tous les répertoires qui seront utilisés par le script.

```
#Config
CONFIG_FILE = "./config/config.cfg"
QUARANTINE_DIR = "./quarantine"
DATA = "./data"
LOGS = "./logs"
```

Figure 30 : définition des constantes de configuration du script

Après avoir créé les constantes de configuration, nous allons créer une constante de type string nommée « API\_KEY » qui stockera notre clé d'API. Comme nous utilisons une API nous avons besoin d'une clé d'authentification qui peut être récupérée sur le site web de VirusTotal dans la catégorie API après s'être créé un compte. Cette clé d'authentification est unique à chaque utilisateur de l'API. Elle permet d'authentifier la requête HTTP envoyée à l'API. Une fois notre clé API stockée dans notre constante, nous allons créer le client avec la méthode « Client » de la class « vt » et notre clé d'API.

```
#VirusTotal config
API_KEY = "e232de4f4e17fd9cbd34dc2f5c4e107df6d77c8c0c1d797c59da95669353cd52"
client = vt.Client(API_KEY)
```

Figure 31 : création du client de VirusTotal

Ensuite nous ferons pareil pour le module « vonage » avec cette fois-ci une clé et une clé secrète (aussi récupérable sur le site web). Nous créerons en premier une constante qui contiendra le client vonage avec ces deux clés et ensuite une variable « sms » qui contiendra la méthode de la classe « vonage » pour envoyer des SMS.

```
#Vonage config
VONAGE = vonage.Client(key="da361fa0", secret="M7u5V3E4jm5CI5kI")
sms = vonage.Sms(VONAGE)
```

Figure 32 : création du client Vonage

Avant de commencer notre fonction principale, nous allons créer trois autres fonctions nécessaires au bon fonctionnement de celle-ci.

En premier nous allons créer la fonction « check\_suspicious\_files » qui prend en argument notre analyse sous la forme d'un dictionnaire, notre liste de virus trouvés, et le chemin d'accès de notre élément. Ensuite nous allons créer une condition qui va vérifier deux valeurs de notre analyse (malicious et suspicious). Si l'une d'entre elles est supérieure à 0 alors notre fichier est potentiellement indésirable, donc on l'ajoute dans notre liste de virus trouvés et on affiche un message dans la console qui dit qu'on vient de trouver un virus avec le nom du virus trouvé.

```
def check_suspicious_files(analysis_json : dict, scan_elements : list, element_path : str):
    if analysis_json['attributes']['stats']['malicious'] > 0 or analysis_json['attributes']['stats']['suspicious'] > 0:
        scan_elements.append(element_path)
        print("Suspicious file found : " + element_path)
```

Figure 33 : création de la fonction check\_suspicious\_files

Ensuite nous allons créer une fonction qui va lire notre fichier de configuration. Cette fonction renvoie une liste qui contient chaque chemin d'accès référencer dans notre fichier de configuration (les chemins d'accès à surveiller).

```
def read_config() -> list:
    with open(CONFIG_FILE, "r") as cfg:
        return cfg.readlines()
```

Figure 34 : création de la fonction read\_config

Enfin nous allons créer une fonction « make\_dir » qui va permettre de créer nos répertoires principaux (date, quarantine, logs) s'ils ne sont pas existants. De plus cette fonction prend un argument, car on veut différencier la création du répertoire « logs » des autres, car celui doit avoir un fichier « log.log » précréé. Si c'est 0 c'est que c'est tous les autres répertoires donc on crée juste le répertoire, sinon c'est que c'est le répertoire « logs » donc il faut que le script crée le fichier « log.log ».

```
def make_dir(path : str, type = 0):
    if os.path.exists(path) != True:
        os.mkdir(path)

    if type != 0:
        open(LOGS + "/log.log", "w").close()
```

Figure 35 : création de la fonction make\_dir

Une fois nos clients et notre configuration effectués, nous allons créer une fonction « virus\_check » qui prendra en paramètre notre fichier de configuration lu ligne par ligne de type liste (ce paramètre conviendra tous les chemins d'accès ciblé dans une liste). Cette fonction parcourra tous les répertoires présents dans notre paramètre pour vérifier de manière récursive, c'est-à-dire tous les fichiers et dossiers d'un dossier qui peut lui-même avoir des sous-dossiers et fichiers qui peuvent eux-mêmes avoir des fichiers et sous-dossiers, etc. Pour vérifier s'il y a un fichier malveillant.

En premiers lieux ont créé deux variables, une qui va contenir la liste des virus qui ont été trouvés durant l'analyse « scan\_elements » et une qui va contenir le résultat complet de l'analyse.

```
scan_elements = []
analysis_json = ""
```

Figure 36 : création de la fonction des variables d'initialisation

Ensuite nous allons créer une boucle de type « for » qui va itérer élément par élément dans notre paramètre (donc, itérer chemin d'accès par chemin d'accès). Pour chaque chemin d'accès, nous allons remplacer les « \ » du chemin d'accès par des « / » pour une meilleure compatibilité.

```
for e in config:
    e = e.replace("\\", "/")
```

Figure 37 : boucle for pour les chemins d'accès

Ensuite nous allons créer une autre boucle de type « for » dans la boucle for précédente qui va itérer sur tous les éléments (fichier et répertoire) avec la fonction « listdir » du module « os » qui retourne une liste de tous les éléments d'un répertoire (fichier, dossier), du chemin d'accès itérer dans le paramètre. Ensuite pour chaque élément itéré nous allons stocker le chemin d'accès absolu de celui-ci et vérifier si c'est fichier.

```
for element in os.listdir(e):
    element_path = os.path.join(e, element)
    if os.path.isfile(element_path):
```

Figure 38 : boucle for sur tous les éléments du chemin d'accès itéré

Si c'est un fichier alors nous allons ouvrir le fichier avec la fonction « open » en passant le chemin d'accès absolu de l'élément et le mode de lecture qui est « rb » (Read bytes) pour lire le fichier en binaire. Une fois le fichier lu en binaire, nous allons créer une variable « analysis » qui va appeler la méthode « scan\_file » de notre client VirusTotal précédemment créer en lui passant notre fichier lu en binaire et en le paramètre « wait\_for\_completion » de la méthode sur « true » pour attendre que l'analyse soit terminée pour continuer le programme.

```
with open(element_path, "rb") as file:
    analysis = client.scan_file(file, wait_for_completion=True)
```

Figure 39 : ouverture du fichier itéré

Une fois l'analyse finie nous allons créer une variable « analysis\_json » qui va convertir notre résultat d'analyse en un dictionnaire avec des valeurs correspondantes à des clés. Ensuite, nous appeller notre fonction « check\_suspicious\_files » créer précédemment en lui passant notre dictionnaire « analysis\_json » la liste qui contient nos virus et le chemin de notre fichier analysé. Enfin le script va écrire le résultat complet de l'analyse de ce fichier dans le dossier « data » en créant un fichier nommé « nom-du-fichier-jour-mois-année-heure-min-sec.dat ».

```
analysis_json = analysis.to_dict()
check_suspicious_files(analysis_json, scan_elements, file_path)
```

Figure 40 : analyse du résultat du fichier analysé

Si ce n'est pas un fichier, mais un répertoire, alors on utilise le générateur walk du module « os » pour aller dans le sous-répertoire suivant et récupérer tous ses éléments pour les scanners si c'est un fichier, sinon avancer d'un pour rentrer du sous-répertoire suivant et récupérer tous ses éléments et les scanner si c'est un fichier et ainsi de suite et si tous les éléments ont été parcourus dans ce sous-répertoire et qu'il n'y en a pas d'autre alors on recule de 1 pour retourner à la racine de celui-ci, etc. C'est le système récursif ». Pour chaque analyse d'élément, le script va écrire le résultat complet de l'analyse de ce fichier dans le dossier « data » en créant un fichier nommé « nom-du-fichier-jour-mois-année-heure-min-sec.dat ».

```
else:
    for root, dirs, files in os.walk(element_path):
        for file in files:
            file_path = os.path.join(root, file)
            with open(file_path, "rb") as file:
                analysis = client.scan_file(file, wait_for_completion=True)
            analysis_json = analysis.to_dict()
            check_suspicious_files(analysis_json, scan_elements, file_path)
            with open(DATA + "/" + element + "-" + str(datetime.datetime.now()).replace(" ", "-").replace(":", "-") + ".dat", "w") as data:
                data.write(str(analysis_json))
```

Figure 41 : système récursif

Une fois tous les éléments de tous les répertoires analysés nous pouvons fermer le client (clôturer la connexion avec l'API) puis appeler notre fonction « make\_dir » crée précédemment en lui passant notre constante qui contient notre chemin vers le répertoire de quarantaine et pareil pour le répertoire des data et des logs en lui passant l'argument 1 pour celui-ci pour qu'il crée directement le fichier « log.log ».

```
client.close()

make_dir(QUARANTINE_DIR)
make_dir(DATA)
make_dir(LOGS, 1)
```

Figure 42 : création des répertoires généraux et fermeture du client VirusTotal

Ensuite nous allons créer une boucle de type « for » qui va parcourir tous les éléments dans la liste « scan\_elements », donc les virus trouvés, pour ensuite faire appelle à la fonction « move » du module « shutil » qui va déplacer le virus dans le répertoire de quarantaine et écrire un message dans le fichier de log pour indiquer son déplacement.

```
for element in scan_elements:
    shutil.move(element, QUARANTINE_DIR)
    with open(LOGS + "/log.log", "a") as log:
        log.write(str(datetime.datetime.now()) + " : " + element + " -> " + os.path.abspath(QUARANTINE_DIR) + "\n")
```

Figure 43 : déplacement des virus dans le dossier de quarantaine et écriture des logs

Une fois tous les virus déplacer en quarantaine et enregistrés dans le fichier log le script va vérifier s'il y a au moins un virus dans la liste des virus trouvés, si c'est le cas il tente d'envoyer un message à l'administrateur pour lui faire part de l'analyse. Pour ce faire ont défini un dictionnaire qui va contenir des clés dont le numéro de tel du destinataire, de l'envoyeur, et le message.

```
if len(scan_elements) > 0:
    try:
        message = "Scanning to detect one or more malicious files.\nThey have been quarantined.\nFiles : \n- " + "\n- ".join(scan_elements)
        responseData = sms.send_message(
            {
                "from": "Vonage APIs",
                "to": "33761476039",
                "text": message
            }
        )
```

Figure 44 : envoi du message

Ensuite on vérifie toujours dans le bloc « try » si le résultat de l'envoi est bien égal à 0, si c'est le cas, c'est qu'il n'y a aucune erreur, donc on peut afficher que le message a été envoyé, sinon on affiche que le message a rencontré une erreur avec l'erreur précise qui est survenue.

```
if responseData["messages"][0]["status"] == "0":
    print("Message sent successfully.")
else:
    print(f"Message failed with error: {responseData['messages'][0]['error-text']}")
```

Figure 45 : vérification de l'envoi du message



En exception du bloc, « try » ont « pass » (on dit au script de ne rien faire, de continuer).  
En revanche s'il n'y a aucun virus dans la liste des virus trouvé, on affiche un message expliquant que l'analyse s'est correctement finie et qu'aucuns virus n'ont été trouvés.

```
else:
    print("No suspicious files found.\nAnalysis completed.")
```

Figure 46 : s'il n'y a aucuns virus

Pour finir notre script on utilise la condition « if \_\_name\_\_ == "\_\_main\_\_" » pour vérifie si le script est bien lancé indépendamment (donc de lui-même) et pas appelé par un autre script. S'il est bien lancé indépendamment alors, on affiche un message et on lance la fonction principale « virus\_check ».

```
if __name__ == "__main__":
    print("Analysis in progress...")
    virus_check(read_config())
```

Figure 47 : vérification du lancement du script

## 4.6 DEPLOIEMENT

Pour le déploiement nous pouvons par exemple faire tourner notre script sur un serveur de notre entreprise qui analysera toutes les heures, jours, ou autre les répertoires référencés dans notre fichier de configuration.

Comme la représentation schématique ci-dessous.

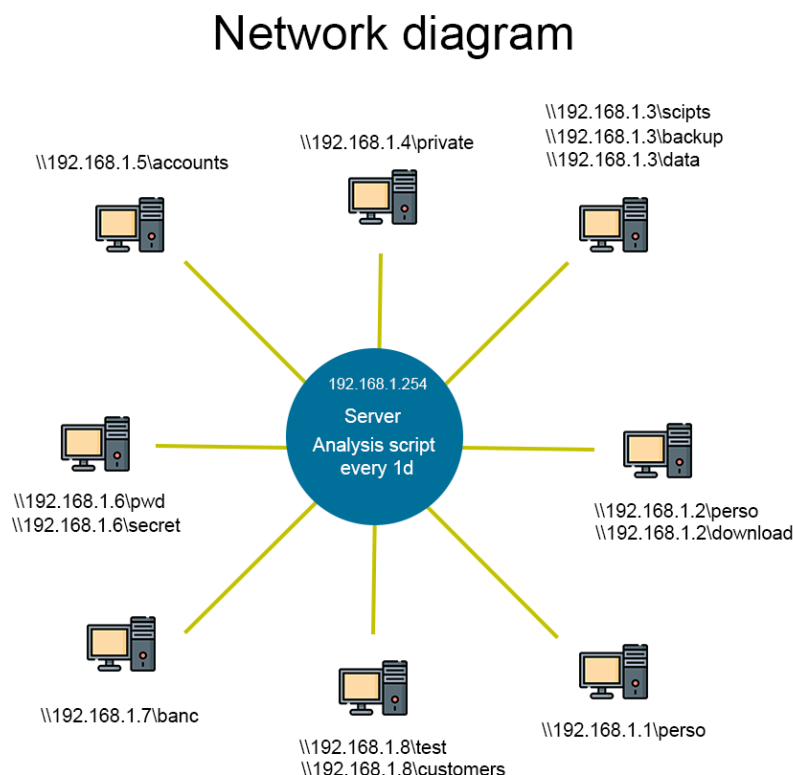


Figure 48 : network diagram



Dans ce schéma nous pouvons voir que notre script est au centre, hébergé sur un serveur, et analyse tous les jours les répertoires de ces machines pour vérifier qu'il n'y a pas de fichier malveillant.

Ou alors nous pouvons passer par un déploiement, machine par machine en passant par le planificateur de tâche sous Windows en exécutant Win + r et en tapant taskschd.msc.

Une fois dans le planificateur de tâche, nous pouvons créer une nouvelle tâche, lui donner un nom.

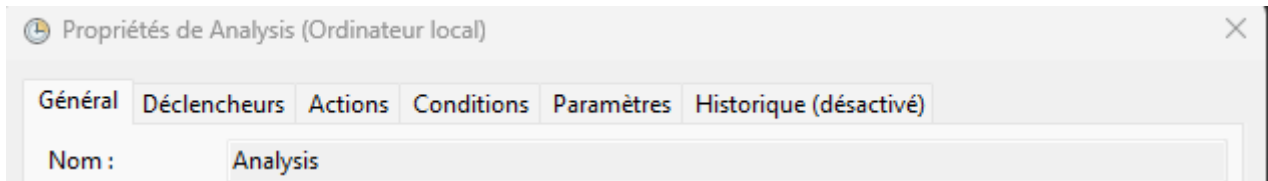


Figure 49 : onglet général du planificateur d'une nouvelle tâche

Ensuite nous pouvons aller dans déclencher, faire nouveau et par exemple choisir « au moment de la connexion à une session utilisateur », le faire pour tous les utilisateurs, et sur depuis l'ordinateur local.

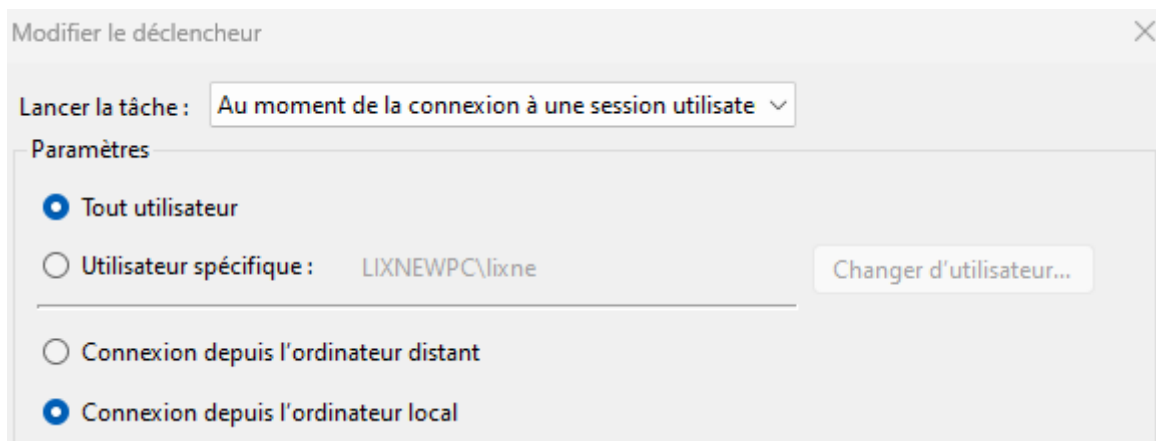


Figure 50 : onglet déclencheur de la nouvelle tâche

Ensuite dans les actions nous pouvons faire « nouveau » et « démarrer un programme » puis référencer le chemin d'accès de notre script à démarrer.

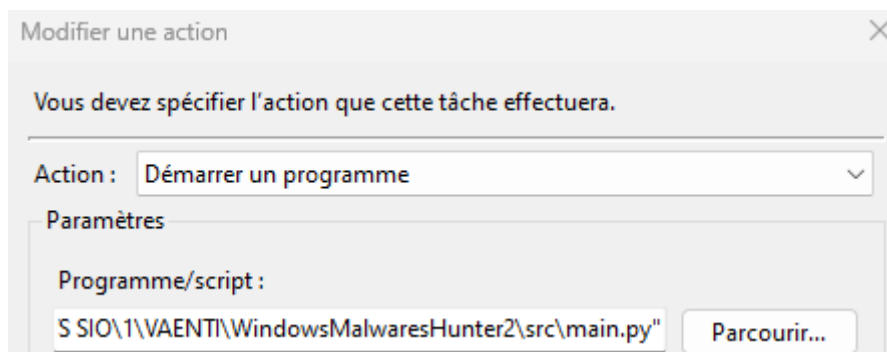


Figure 51 : onglet action de la nouvelle tâche

Maintenant à chaque connexion d'un utilisateur, sur sa machine le script sera lancé et analysera les répertoires référencer dans le fichier de configuration du script.

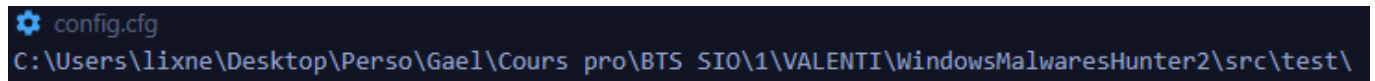
## 4.7 TEST DE SCRIPT

Pour réaliser le test du script, je vais le faire localement sur ma machine. Je me suis muni d'un jeu d'essai comportant une architecture précise :

```
- test
|
- access
|
--- keys.txt
- files
|
- docs
|
- secret
|
- data
|
--- data1.txt
- txt
|
--- embed.txt
```

Les deux virus sont « embed.txt » et « data1.txt ».

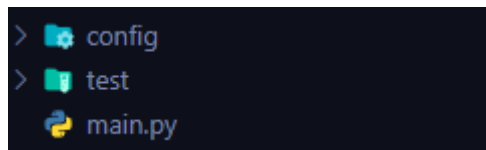
Dans mon fichier « config.cfg », je vais référencer le chemin d'accès du répertoire à surveiller.



```
config.cfg
C:\Users\lixne\Desktop\Perso\Gael\Cours pro\BTS SIO\1\VALENTI\WindowsMalwaresHunter2\src\test\
```

Figure 52 : configuration du fichier de configuration

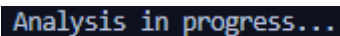
Je n'ai actuellement aucun dossier (data, quarantine, logs) créer car je n'ai pas effectué d'analyse.



```
> config
> test
main.py
```

Figure 53 : environnement avant le lancement de l'analyse

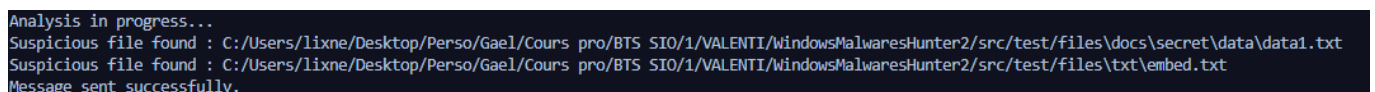
Après avoir bien configuré mon fichier de configuration, je peux lancer le script d'analyse.



```
Analysis in progress...
```

Figure 54 : lancement de l'analyse

Une fois que l'analyse effectuée, le script envoie un message dans la console, indiquant qu'il a trouvé deux virus (qui sont bien ce du jeu d'essai) de plus il a envoyé un SMS à l'administrateur.



```
Analysis in progress...
Suspicious file found : C:/Users/lixne/Desktop/Perso/Gael/Cours pro/BTS SIO/1/VALENTI/WindowsMalwaresHunter2/src/test/files/docs/secret/data/data1.txt
Suspicious file found : C:/Users/lixne/Desktop/Perso/Gael/Cours pro/BTS SIO/1/VALENTI/WindowsMalwaresHunter2/src/test/files/txt/embed.txt
Message sent successfully.
```

Figure 55 : message du script de fin d'analyse

Dans ce SMS nous avons un message nous indiquant que l'analyse a détecté des fichiers suspects et nous affiche le chemin d'accès de chaque fichier suspect détectés.

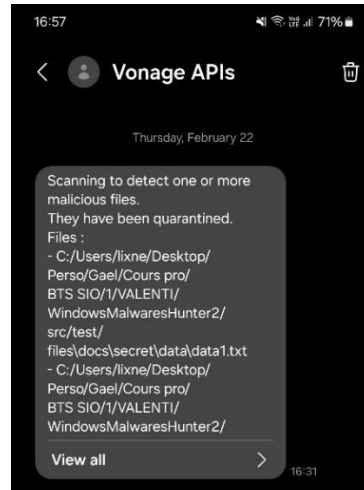


Figure 56 : SMS envoyé par le script

Si nous regardons cette fois-ci au niveau de l'environnement des tests nous pouvons voir que dans le répertoire « test » nous n'avons plus nos deux fichiers indésirables.

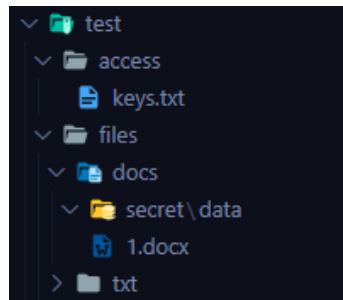


Figure 57 : répertoire test après analyse

En revanche dans le répertoire « quarantine » nous retrouvons nos deux fichiers.

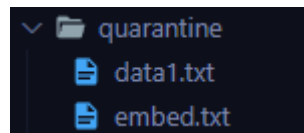


Figure 58 : répertoire quarantine après analyse

Dans le répertoire log, nous avons bien fichier « log.log » qui contient tous les logs des virus déplacés vers le répertoire de quarantaine.

```
2024-02-22 16:31:04.717899 : C:/Users/lixne/Desktop/Perso/Gael/Cours pro/BTS SIO/1/VALENTI/WindowsMalwaresHunter2/src/test/files/docs\secret\data\data1.txt -> C:/Users/lixne/Desktop/Perso/Gael/Cours pro/BTS SIO/1/VALENTI/WindowsMalwaresHunter2/quarantine/data1.txt
2024-02-22 16:31:04.718908 : C:/Users/lixne/Desktop/Perso/Gael/Cours pro/BTS SIO/1/VALENTI/WindowsMalwaresHunter2/src/test/files/docs\secret\data\data1.txt -> C:/Users/lixne/Desktop/Perso/Gael/Cours pro/BTS SIO/1/VALENTI/WindowsMalwaresHunter2/quarantine/data1.txt
```

Figure 59 : fichier de log après analyse

Enfin dans le répertoire « data » nous retrouvons, tous les résultats d'analyse de chaque fichier (virus ou non) analysé.

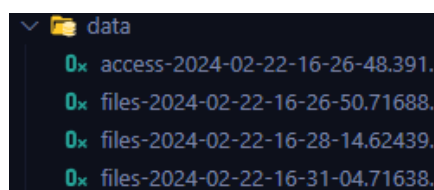


Figure 60 : répertoire data après analyse

## 5 SIGNATURE DE CODE

---

### 5.1 CONCEPTS

La signature de code est un processus de sécurité qui consiste à apposer une signature numérique à un logiciel ou à un script pour garantir son authenticité et son intégrité.

**Signature numérique** : Une signature numérique est une empreinte cryptographique générée à partir du logiciel ou du script. Elle est unique pour chaque fichier et permet de vérifier l'identité de l'auteur du code.

**Certificat numérique** : Un certificat numérique est un fichier électronique qui lie une identité à une clé publique. Il est délivré par une autorité de certification (CA) de confiance et atteste que la clé publique appartient à l'entité mentionnée dans le certificat.

**Clé privée et clé publique** : Dans le processus de signature numérique, une clé privée est utilisée pour générer la signature, tandis que la clé publique correspondante est utilisée pour vérifier la signature.

**Autorité de certification (CA)** : Une autorité de certification est une entité de confiance qui émet des certificats numériques après avoir vérifié l'identité du demandeur.

**Chemin de certification** : Le chemin de certification est une séquence de certificats qui relie le certificat utilisé pour signer le logiciel ou le script à un certificat racine de confiance. Il permet de vérifier que le certificat de signature a été émis par une autorité de certification de confiance.

En résumé, la signature de code permet aux utilisateurs de vérifier l'authenticité et l'intégrité des logiciels et des scripts en associant une signature numérique à un certificat numérique délivré par une autorité de certification de confiance. Cela contribue à prévenir les attaques telles que l'injection de code malveillant ou les modifications non autorisées du logiciel.

### 5.2 ARCHITECTURE ET INFRASTRUCTURE (PKI)

L'architecture et l'infrastructure de la PKI (Infrastructure à Clé Publique) sont essentielles pour la mise en œuvre réussie de la signature de code.

#### 5.2.1 ARCHITECTURE DE LA PKI :

**Autorité de Certification (CA)** : La CA est l'entité centrale de la PKI. Elle est responsable de la délivrance, de la révocation et de la gestion des certificats numériques.

**Autorité d'Enregistrement (RA)** : La RA est responsable de vérifier l'identité des entités demandant des certificats numériques avant de transmettre les demandes de certificats à la CA.

**Annuaire** : L'annuaire est une base de données qui stocke les informations sur les certificats délivrés, y compris les clés publiques et les informations d'identification des entités.

**Entités Utilisatrices** : Ce sont les utilisateurs finaux ou les applications qui utilisent les certificats numériques pour l'authentification, la signature et le chiffrement.

## 5.2.2 INFRASTRUCTURE DE LA PKI :

**Clés Publiques et Privées :** Chaque entité dispose d'une paire de clés cryptographiques, une clé publique qui peut être partagée publiquement et une clé privée qui doit rester confidentielle.

**Certificats Numériques :** Les certificats numériques contiennent des informations sur l'entité, sa clé publique et d'autres métadonnées. Ils sont signés par une CA pour attester de leur authenticité.

**Protocoles de Communication :** Les protocoles comme le protocole X.509 sont utilisés pour l'échange sécurisé des certificats et des données entre les différentes entités de la PKI.

**Sécurité Physique et Logique :** Des mesures de sécurité physiques et logiques sont mises en place pour protéger les composants de l'infrastructure de la PKI contre les accès non autorisés et les attaques.

En résumé, l'architecture et l'infrastructure de la PKI fournissent une sécurité élevée pour la délivrance, la gestion et l'utilisation des certificats numériques, ce qui permet de garantir l'authenticité, l'intégrité et la confidentialité des communications.

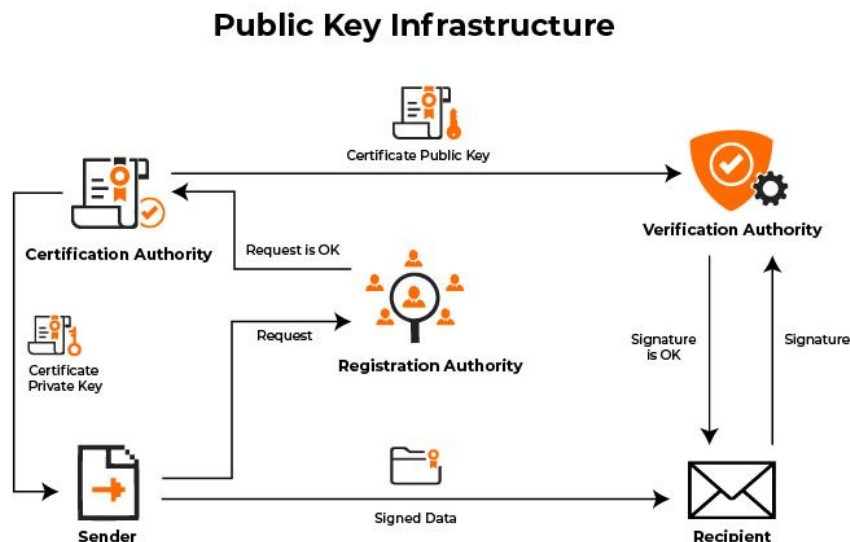


Figure 61 : infrastructure PKI

Ce schéma montre l'infrastructure du système PKI présenté précédemment.

## 5.3 LE CAS PARTICULIER DES PE SOUS WINDOWS

Le format de fichier PE (Portable Exécutable) est un format de fichier regroupant les exécutables et les bibliothèques d'un programme sur Windows 32bits et 64bits. Le format a été adopté par Microsoft suite à l'introduction de Windows NT en 1993 : Windows NT (New technologie) désigne la série de systèmes d'exploitation multitâches préemptif, multi-utilisateur, multiprocesseur et ne reposant pas sur le système MS-DOS.

Ce format a été créé par Microsoft pour créer un format de fichier avec une structure de fichier qui puisse s'adapter aux différentes machines utilisant Windows NT.

Un fichier exécutable PE est structuré de cette manière :



Figure 62 : Structure d'un fichier PE

Ce schéma illustre la structure d'un fichier au format PE.

L'en-tête MZ-DOS permet au système d'exploitation de reconnaître le fichier comme étant un exécutable valide dans le cas où celui-ci sera lancé depuis le MS-DOS.

Le segment DOS est exécuté lorsque Windows ne connaît pas le fichier comme étant au format PE, ou s'il est exécuté sous MS-DOS.

L'en-tête PE est un ensemble de structures, regroupées dans une même et unique structure nommée IMAGE\_NT\_HEADER.

La Table de Section est située juste derrière l'en-tête PE. C'est un tableau contenant plusieurs structures. Ces structures contiennent les informations sur les sections binaires devant être chargées en mémoire.

En somme, le format de fichier PE a été un élément crucial dans l'évolution des systèmes d'exploitation Windows, permettant une compatibilité et une exécution efficace des applications sur une gamme variée de matériel.

## 5.4 LES OUTILS EN LIGNE DE COMMANDE

### 5.4.1 SIGCHECK DU SYSINTERNALS

L'outil SignCheck, développé par Microsoft, est un utilitaire de la Suite SysInternals conçu pour vérifier les signatures numériques des fichiers sur les systèmes Windows.

#### Fonctionnalités de l'outil SignCheck :

- Vérification des signatures numériques : SignCheck permet aux utilisateurs de vérifier rapidement si les fichiers sur leur système sont signés numériquement.
- Identification des éditeurs : En plus de vérifier la présence de signatures numériques, SignCheck peut extraire et afficher des informations sur les éditeurs des fichiers, y compris le nom de l'entreprise et d'autres détails pertinents. Cela permet aux utilisateurs de savoir qui a publié le fichier.

- **Détection des fichiers non signés :** En identifiant les fichiers qui ne sont pas signés numériquement, SignCheck peut aider les administrateurs système à repérer les fichiers potentiellement dangereux ou non fiables sur leur système.
- **Intégration dans des scripts et des processus automatisés :** Comme SignCheck est utilisation en ligne de commande, il peut être utilisé dans des scripts et des processus automatisés pour effectuer des vérifications de sécurité régulières.

En outre, SignCheck offre aux administrateurs système un moyen efficace de vérifier les signatures numériques des fichiers sur les systèmes Windows, ce qui contribue à renforcer la sécurité et la fiabilité des environnements informatiques.

Vérification de la signature numérique d'un fichier avec SignCheck en utilisant la commande « `.\sigcheck64.exe [chemin_accès_fichier]` » :

```
Sigcheck v2.90 - File version and signature viewer
Copyright (C) 2004-2022 Mark Russinovich
Sysinternals - www.sysinternals.com

f:\visualstudiocode\microsoft vs code\Code.exe:
  Verified:      Signed
  Signing date:  16:49 08/03/2024
  Publisher:     Microsoft Corporation
  Company:       Microsoft Corporation
  Description:   Visual Studio Code
  Product:       Visual Studio Code
  Prod version:  1.87.2
  File version:  1.87.2
  MachineType:   64-bit
```

Figure 63 : exécution de signCheck

On peut voir que l'outil nous renvoie si le fichier est vérifié en indiquant « Signed » si c'est le cas, la date à laquelle le fichier a été signé numériquement, le éditeur, le nom de l'entreprise, la version de fichier, etc.

#### 5.4.2 SIGNTOOL DU WINDOWS SDK

L'outil Signtool est un utilitaire inclus dans le Windows Software Development Kit (SDK) fourni par Microsoft. Cet outil est principalement utilisé pour signer numériquement des fichiers et des programmes sur les systèmes d'exploitation Windows.

##### Fonctionnalités de l'outil Signtool :

- **Signature numérique des fichiers :** L'outil Signtool permet aux développeurs et aux éditeurs de logiciels de signer numériquement leurs fichiers exécutables, bibliothèques de liens dynamiques (DLL), fichiers de configuration, fichiers d'installation, etc.
- **Certificats numériques :** Signtool utilise des certificats numériques pour signer les fichiers. Ces certificats sont émis par des autorités de certification (CA) et servent de preuve d'identité pour l'éditeur du logiciel.

En résumé, Signtool est un outil essentiel pour les développeurs Windows, leur permettant de signer numériquement leurs fichiers afin d'assurer leur authenticité et leur intégrité.

Signature d'un fichier avec signtool en utilisant la commande « `.\signtool.exe sign /fd [algorithme_de_chiffrement] /f [chemin_accès_certificat] /p [mot_de_passe] [chemin_accès_fichier_a_signer]` »

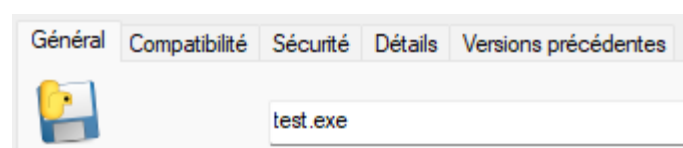


Figure 64 : fichier non signé

Avant d'exécuter la commande pour signer le fichier on peut bien voir qu'il n'est pas encore signé, car il ne possède pas l'onglet « Signature numérique ».

```
Done Adding Additional Store
Successfully signed: C:\Users\lixne\Desktop\test.exe
```

Figure 65 : signature effectuée

Après que la commande a été effectuée correctement, la console nous retourner un message nous informant que le certificat a été ajouté au magasin de certificat et que le fichier a été signé avec succès.

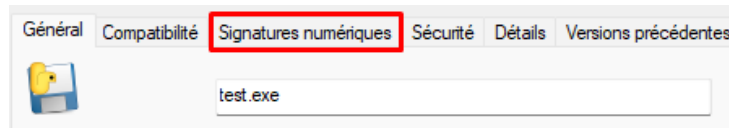


Figure 66 : fichier singé

Une fois la signature effectuée notre fichier possède bien l'onglet « Signature numérique ».

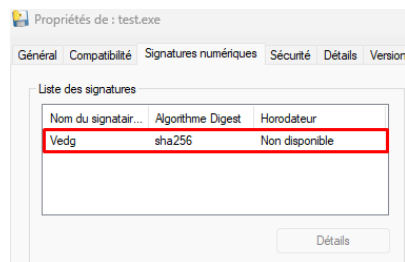


Figure 67 : vérification de La signature numérique

En allant dans celle-ci, nous pouvons voir les informations du certificat avec lequel il a été signé.

## 5.5 LES OUTILS EN INTERFACE GRAPHIQUE

### 5.5.1 LA MMC – MICROSOFT MANAGEMENT CONSOLE

La MMC ou Microsoft Management Console est une application de gestion intégrée dans le système Windows qui fournit une interface graphique utilisateur (GUI) pour l'administration des certificats et signature numériques.

#### Fonctionnalités de la MMC :

- **Gestion des certificats :** La MMC permet d'accéder au magasin de certificats de l'ordinateur local ou de l'utilisateur actuel. Elle permet d'importer, d'exporter, d'afficher, et de supprimer des certificats.
- **Signature numérique :** La MMC peut être utilisée pour afficher et gérer les propriétés des certificats numériques, y compris les informations sur les signatures numériques associées à un fichier ou à une application. Elle permet également de vérifier l'authenticité des fichiers signés numériquement.
- **Gestion des autorités de certification :** La MMC peut aussi gérer les autorités de certification de confiance et les autorités de certification intermédiaires. Cela comprend l'ajout, la suppression et la modification des autorités de certification dans les magasins de certificats.

En résumé la MMC permet de gérer les certificats / signature numérique enregistrés dans notre environnement informatique, en offrant diverse fonctionnalité telle que des fonctionnalités de CRUD (Create, Read, Update, Delete) et de sécurité.



### 5.5.2 PROCESS EXPLORER

L'outil process explorer permet aussi aux utilisateurs d'accéder aux informations des certificats des processus en cours d'exécution sur leur système, ce qui leur permet de vérifier l'authenticité des processus et de valider leurs signatures numériques. Cela peut être particulièrement utile pour détecter les processus malveillants ou non autorisés sur le système. En examinant les propriétés d'un processus dans Process Explorer, les utilisateurs peuvent accéder aux détails du certificat associé au fichier exécutable du processus, tels que l'éditeur du logiciel, l'autorité de certification et la validité du certificat.

On peut par exemple vérifier si un processus est signé en se rendant dans ses propriétés et en faisant « vérifié ».

Pour un processus qui est signé et vérifié, Process Explorer nous retourne le nom du signataire.

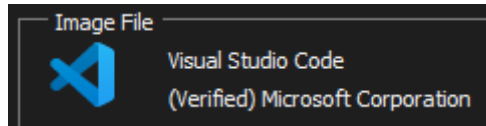


Figure 68 : processus signé

Pour un processus qui est signé, mais pas vérifié, Process Explorer nous affiche un message nous expliquant qu'une chaîne de certificat a été trouvée dans le processus, mais que le certificat racine (parent) n'est pas un certificat approuvé par le fournisseur d'approbation.

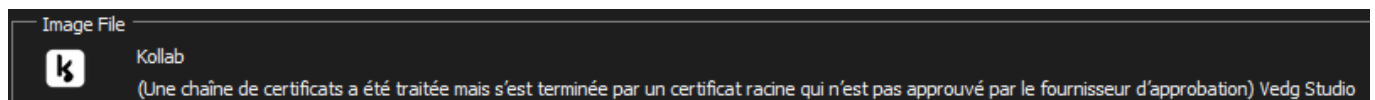


Figure 69 : processus signé, mais pas vérifié

En revanche sur un processus qui n'est pas signé il nous affiche un message nous expliquant qu'aucune signature numérique n'a été trouvée pour ce processus.

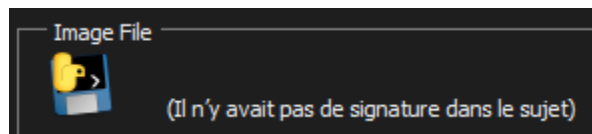


Figure 70 : processus non signé

Nous pouvons aussi avoir une indication sur le signataire du processus si la colonne « Vérified Signer » est activée. Pour un processus possédant une signature d'un signataire vérifié la colonne nous retourne le nom du signataire.

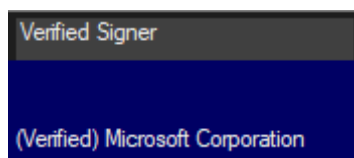


Figure 71 : signataire vérifié

## 5.6 OID

Un OID (Object Identifier) est une séquence numérique qui identifie de manière unique le certificat. Les OIDs sont utilisés pour identifier les types de clés publiques, les algorithmes de chiffrement, les extensions de certificat, etc. Ils permettent d'assurer une interopérabilité entre les différents systèmes et applications.

Un OID se trouve dans les détails de la signature du certificat du fichier en question. Par exemple voici l'OID du certificat de « sigcheck64.exe » : 1.3.6.1.4.1.311.2.1.12

Détails de la signature :

Champ	Valeur
Numéro de série	33000002528b33aaf895f339db0000000
Algorithme Digest	sha256
Algorithme de chiffremen...	RSA
Attributs authentifiés	
Type de contenu	06 0a 2b 06 01 04 01 82 37 02 01 04
1.3.6.1.4.1.311.2.1.11	30 0c 06 0a 2b 06 01 04 01 82 37 02 01
Résumé du message	04 20 2e 62 d8 a8 39 92 37 75 a2 e6 21.
1.3.6.1.4.1.311.2.1.12	30 34 a0 12 80 10 00 53 00 69 00 67 00.

Figure 72 : détail d'une signature numérique

Si nous entrons un OID sur un site proposant une base de données complète d'OID comme le site web <http://oid-info.com/>, il nous retournera des informations comme son algorithme de chiffrement, le nombre de signatures, etc.

OID:	{iso(1) member-body(2) us(840) ansi-x962(10045) signatures(4) ecdsa-with-SHA2(3) ecdsa-with-SHA256(2)}	(ASN.1 notation)
	1.2.840.10045.4.3.2	(dot notation)
	/ISO/Member-Body/US/10045/4/3/2	(OID-IRI notation)
<b>Description:</b> Elliptic Curve Digital Signature Algorithm (DSA) coupled with the Secure Hash Algorithm 256 (SHA256) algorithm		
<b>Information:</b> See IETF RFC 5480 and RFC 5758.		

Figure 73 : détail d'un OID

## 5.7 PAR SCRIPT PYTHON AVEC PEFILE

Dans cette partie nous allons créer un programme permettant d'interagir avec les fichiers au format PE et les signatures numériques.

### 5.7.1 ANALYSE ET CONCEPTION

Pour illustrer notre compréhension des PE et des signatures numériques, nous allons créer un programme permettant de vérifier si un fichier est un fichier au format PE, s'il est signé et de le signer numériquement.

Pour créer ce programme, nous allons utiliser le langage de programmation Python et la librairie PEFile ainsi que l'outil de la Suite SysInternals « signCheck » et celui du SDK de Windows « signTool ».

## 5.7.2 CONCEPTION DU SCRIPT

Avant de créer le script principal, nous allons créer un script qui contiendra toutes les erreurs pouvant arriver durant l'exécution du script principal.

Pour chaque erreur nous créons une classe qui aura pour nom le type de l'erreur et qui héritera de la classe « Exception ». Le constructeur de la classe contiendra un paramètre nommé « file\_path » de type « str » et initialisé à « " " », soit vide, par défaut. Ensuite nous initialiserons le message que nous souhaitons que l'erreur renvoie quand elle est générée.

```
class AlreadySignedError(Exception):
    def __init__(self, file_path : str = ""):
        self.message = "File already signed : " + str(file_path)
        super().__init__(self.message)

class NotPEFileError(Exception):
    def __init__(self, file_path : str = ""):
        self.message = "File is not a PE file : " + str(file_path)
        super().__init__(self.message)

class FileError(Exception):
    def __init__(self, file_path : str = ""):
        self.message = "File error : " + str(file_path)
        super().__init__(self.message)

class SignatureError(Exception):
    def __init__(self, file_path : str = ""):
        self.message = "Error while signing file : " + str(file_path)
        super().__init__(self.message)

class SignCheck(Exception):
    def __init__(self, file_path : str = ""):
        self.message = "Error while checking signature : " + str(file_path)
        super().__init__(self.message)
```

Figure 74 : création des erreurs

Dans un premier temps nous allons importer notre librairie « pefile » et deux autres supplémentaires. Une qui se nomme « subprocess » qui permet d'exécuter des commandes dans le MS-DOS et une autre que ce nomme « errors » que j'ai fait, qui servira à gérer les erreurs.

```
import pefile, subprocess, libs.errors as ers
```

Figure 75 : importation des librairies

Ensuite nous allons créer une classe nommée « FILEValidator » qui contiendra les méthodes pour vérifier si un fichier est un fichier PE, s'il est signé et pour le signer. Le constructeur de cette classe prend deux « str ». Un qui sera le chemin d'accès du « signtool » et l'autre du « sigcheck ».

```
class FILEValidator:
    def __init__(self, SIGCHECK, SIGNTOOL):
        self.SIGCHECK = SIGCHECK
        self.SIGNTOOL = SIGNTOOL
```

Figure 76 : création de La classe FILEValidator

Cette classe contiendra trois méthodes. La première nommée « is\_pe » et qui prends un paramètre de type « str » qui est le chemin d'accès du fichier de l'utilisateur, cette méthode retournera un booléen (true / false) si le fichier est un fichier au format PE ou non. La deuxième nommée « is\_signed » qui n'a aussi qu'un seul paramètre qui est chemin d'accès du fichier de l'utilisateur, cette méthode retournera un booléen (true / false) si le fichier est signé ou non. Enfin la dernière (la troisième) nommée « sign\_file » prend quatre paramètres de type « str », une pour le chemin du fichier à signer, un second pour le chemin du certificat, le troisième pour le mot de passe du certificat, et le dernier pour le type d'algorithme de chiffrement du certificat (par défaut initialisé à « SHA256 », cette méthode permettra de signer un fichier et ne retourne rien.

```
def is_pe(self, file_path : str) -> bool: ...
def is_signed(self, file_path : str) -> bool: ...
def sign_file(self, file_path : str, certificate_path : str, certificate_pdw : str, encryption_algorithm : str = "SHA256") -> None: ...
```

Figure 77 : création des prototypes des méthodes

La méthode « is\_pe » créer une instance (un objet) de la classe PE de la librairie PEFile avec le chemin d'accès du fichier à vérifier. On passe cette initialisation d'instance dans un bloc « try/except », car c'est un code critique : code qui peut interrompre le programme. On vérifie si c'est un PE avec notre objet créé précédemment et la méthode « is\_exe » de la classe PE. SI c'est un PE, on retourne « true » soit vrai, sinon « false » soit faux. De plus s'il se passe une erreur inattendue durant la vérification, la méthode passera automatiquement dans le bloc « except » et créera une erreur de type « FileNotFoundError » pour indiquer à l'utilisateur qu'il s'est passé quelque chose d'anormal.

```
def is_pe(self, file_path : str) -> bool:
    try:
        pe = pefile.PE(file_path)
        return True if pe.is_exe else False
    except:
        raise ers.FileError(file_path)
```

Figure 78 : création de la méthode is\_pe

La méthode « is\_signed » crée aussi une instance de la classe PE avec le chemin d'accès du fichier à vérifier. La variable « result » va stocker le résultat de la commande de vérification de signature avec l'outil « signcheck ». La variable « output » va lire le résultat de sortie stocké dans la variable « result » et l'encoder en « latin-1 »<sup>4</sup>, ensuite elle va venir supprimer les retours à la ligne du résultat. Une fois notre « output » préparer, la méthode va parcourir toutes les lignes de l'output à la recherche d'une ligne qui possède le mot « Verified », si la ligne possède le mot, elle va alors vérifier sur la même ligne s'il y a un mot « signed », si c'est le cas, elle retourne true, sinon false, puis si la méthode ne parvient pas à trouver le premier mot qui est « Verified » alors elle retournera false par défaut. Si une erreur se passe durant la vérification, alors la méthode créera une erreur de type « SignCheck ».

```
def is_signed(self, file_path : str) -> bool:
    try:
        result = subprocess.Popen(self.SIGCHECK + " " + "'" + file_path + "'", shell=False, stdout=subprocess.PIPE)
        output = result.stdout.read().decode('latin-1')
        output = output.split("\n")

        for line in output:
            if "Verified:" in line:
                return True if "Signed" in line else False
        return False
    except:
        raise ers.SignCheck(file_path)
```

Figure 79 : création de la méthode is\_signed

<sup>4</sup> Latin-1 est un type d'encodage de caractères.



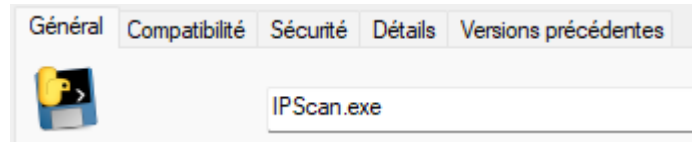


Figure 83 : fichier à tester

Après avoir exécuté le programme, on peut voir dans la console que le fichier est bien un fichier au format PE, et qu'il n'est pas signé. Comme le fichier est un fichier au format PE et qu'il n'est pas déjà signé la méthode va le signer et retourner que le fichier a été signé.

```
Is PE : True
Is signed : False
IPScan.exe signed !
```

Figure 84 : résultat du test unitaire

Si on va voir le fichier après l'exécution du script, on peut voir que le fichier a bien été signé, car il possède l'onglet « Signatures numériques ».

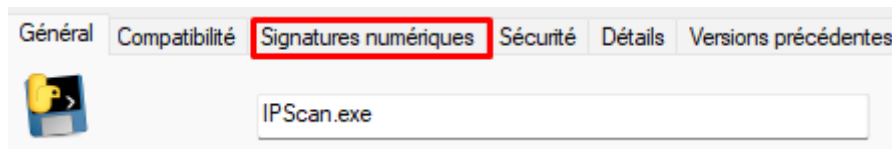


Figure 85 : vérification de la signature numérique du fichier de test

Nom du signataire...	Algorithme Digest	Horodateur
Vedg	sha256	Non disponible

Figure 86 : vérification du certificat

#### 5.7.4 AJOUT DES METHODES AU ROBOT D'ANALYSE

Dans cette dernière partie, nous allons ajouter nos méthodes dans notre robot d'analyse de virus. Quand notre robot effectuera une analyse il vérifiera si le fichier itérer est un fichier signé, si c'est le cas il continue l'analyse sur les autres fichiers à analyser. Si un fichier est non signé et au format PE, il l'ajoute directement à la liste de virus potentiels, enfin s'il n'est pas signé, mais que ce n'est pas un PE, il va envoyer une requête à l'api de Virus Total pour vérifier s'il n'est pas malveillant.

##### Résumé :

Si signé -> BYPASS (on saute l'analyse).

Si pas signé et PE (exécutable) -> ajout à la liste de virus potentiels.

Si pas signé et pas PE (txt, docx, autre qu'exécutable) -> envoie requête à l'API Virus Total.

```
import libs.file_validator as fv
```

Figure 87 : import des modules

Une fois nos modules import nous allons créer une instance de la classe « FileValidator » avec le chemin d'accès vers nos outils.

```
#File validator config
SIGCHECK = "./V3/tools/sigcheck64.exe"
SIGNTOOL = "./V3/tools/signtool.exe"
FV = fv.FILEValidator(SIGCHECK, SIGNTOOL)
```

Figure 88 : initialisation de l'instance

Ensuite nous allons créer une fonction « check\_file », pour englober nos conditions et limiter la duplication de code.

```
def checkFile(element_path : str, scan_elements : list):
    if FV.is_signed(element_path) != True:
        if FV.is_pe(element_path):
            scan_elements.append(element_path)
            print("Suspicious file found : " + element_path)
            return True
        else:
            return False
    else:
        return True
```

Figure 89 : création de la fonction de check

Nous allons appeler cette fonction pour chaque élément itéré. Si la fonction renvoie « True » alors on passe au fichier suivant.

```
if checkFile(element_path, scan_elements):
    continue
```

Figure 90 : ajout de la condition avec la fonction de check

Pour tester cette intégration à notre robot d'analyse, nous allons utiliser le jeu d'essai réaliser sur la partie de Virus Total du handbook (il contient des fichiers non PE et non signé (embed.txt, 1.docx, data.txt, key.txt) et deux autres fichiers, un fichier PE signé (audacity.exe), et un fichier PE non signé (kollab.exe)).

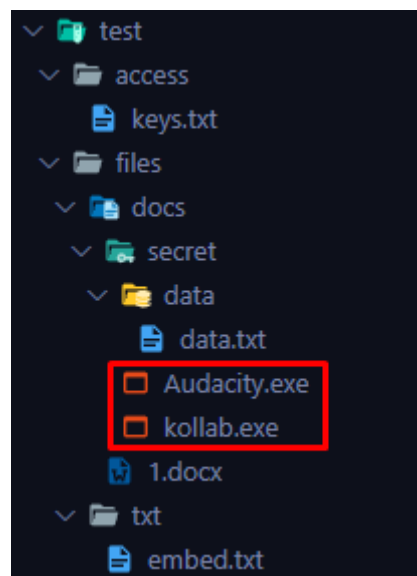


Figure 91 : jeu d'essai avant analyse

Si on lance le robot d'analyse on peut bien voir que tous les fichiers qui sont signés n'ont pas bougés de leurs emplacements, les fichiers non signés et PE ont été déplacés en quarantaine et les fichiers non signés et non PE ont été analysés et s'ils sont analysés par l'API comme malveillant, ils sont mis en quarantaine. Comme le fichier « kollab.exe » est un PE non signé, il est bien en quarantaine, comme « data.txt » et « embed.txt » sont des fichiers non PE et non signés (mais des EICAR) ils ont été analysés comme malveillant par l'API, donc ils ont été mis en quarantaine. Tous les autres fichiers n'ont pas bougé, car soit signés, soit non malveillant.

```
Analysis in progress...
Suspicious file found : C:/Users/lixne/Desktop/Perso/Gael/cours_pro/BTS_SIO/1/VALENTI/WindowsMalwaresHunter/src/V2/test/files/docs/secret/kollab.exe
Suspicious file found : C:/Users/lixne/Desktop/Perso/Gael/cours_pro/BTS_SIO/1/VALENTI/WindowsMalwaresHunter/src/V2/test/files/docs/secret/data/data.txt
Suspicious file found : C:/Users/lixne/Desktop/Perso/Gael/cours_pro/BTS_SIO/1/VALENTI/WindowsMalwaresHunter/src/V2/test/files/txt/embed.txt
Message sent successfully.
```

Figure 92 : résultat de l'analyse

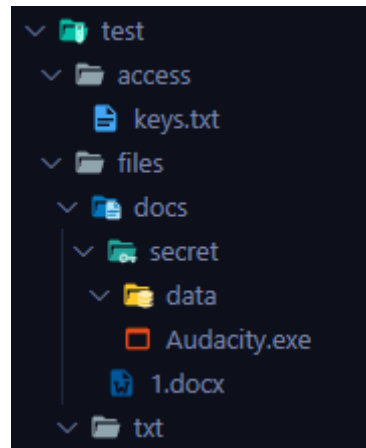


Figure 93 : jeu essai après analyse

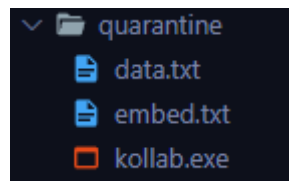


Figure 94 : dossier de quarantaine après analyse



## 6 ANALYSE DE FICHER

---

### 6.1 FONCTIONNEMENT D'UN SYSTEME DE STOCKAGE

Dans un système de stockage, les fichiers sont écrits et stockés en utilisant une structure complexe qui comprend des secteurs, des clusters et des métadonnées, etc.

- **Secteurs** : Un disque dur est divisé en petites unités de stockage appelées secteurs. Chaque secteur contient une quantité fixe de données, généralement de 512 octets à 4 Ko (4096 octets).
- **Clusters** : Les secteurs sont regroupés en clusters, qui sont des groupes de secteurs. Les clusters constituent l'unité de base pour l'allocation de l'espace disque. La taille d'un cluster dépend du système de fichiers utilisé.
- **Métadonnées** : Chaque système de fichiers maintient des métadonnées pour suivre l'emplacement et les attributs des fichiers. Les métadonnées comprennent des informations telles que le nom du fichier, la taille, les autorisations d'accès, les dates de création et de modification, ainsi que des pointeurs vers les clusters où se trouvent les données réelles du fichier.
- **Allocation de l'espace** : Lorsqu'un fichier est créé, le système de fichiers alloue un certain nombre de clusters pour stocker ses données. Ces clusters peuvent être situés de manière contiguë sur le disque ou dispersés à travers plusieurs emplacements, selon la fragmentation du disque.
- **Écriture des données** : Lorsque des données sont écrites dans un fichier, le système d'exploitation les divise en blocs de la taille du cluster et les écrit dans les clusters alloués pour le fichier. Si les données dépassent la taille d'un cluster, le système de fichiers alloue automatiquement plus de clusters pour stocker les données supplémentaires.
- **Actualisation des métadonnées** : Après l'écriture des données, les métadonnées du fichier sont mises à jour pour refléter les modifications, y compris la taille du fichier, la date de modification et les nouveaux clusters alloués.
- **Fragmentation** : Au fil du temps, les fichiers peuvent être fragmentés, ce qui signifie que leurs clusters sont dispersés sur le disque non contiguëment. La fragmentation peut ralentir l'accès aux fichiers car le disque doit parcourir plusieurs emplacements pour lire toutes les parties du fichier.

## 6.2 LES OUTILS EN LIGNE DE COMMANDE

Il existe des outils en ligne de commande permettant d'obtenir des informations sur l'état d'une mémoire de masse comme un HDD, SSD, clé USB, etc. Ces outils sont très utiles et très puissants pour détecter et réparer des problèmes au sein d'un système de stockage comme :

- **Erreur de secteur défectueux** : Un secteur défectueux est un petit bloc de données sur le disque qui est endommagé physiquement ou logiquement. Cela peut être dû à des chocs physiques, à l'usure normale du disque, à des interférences électromagnétiques, etc.
- **Erreur de lecture/écriture** : Ces erreurs se produisent lorsqu'il y a des problèmes lors de la lecture ou de l'écriture de données sur le périphérique de stockage. Cela peut être dû à des problèmes matériels tels que des défaillances du contrôleur, des têtes de lecture/endommagées ou des erreurs logicielles telles que des conflits de fichiers ou des secteurs défectueux.
- **Erreur de formatage** : L'erreur de formatage se produit lorsque le système de fichiers sur le périphérique de stockage est corrompu ou endommagé. Lorsqu'un périphérique de stockage est mal formaté, il peut devenir inaccessible ou inutilisable.
- **Erreur de fragmentation** : La fragmentation se produit lorsqu'un fichier est divisé en plusieurs parties dispersées sur le disque plutôt que d'être stocké de manière continue. Elle peut entraîner des temps d'accès plus longs aux fichiers et des performances réduites du système.

Une autre utilité aux outils de commande peut-être, l'intégration dans des applications. Comme ces outils sont utilisables via un système DOS, nous pouvons les implémenter facilement dans des scripts pour par exemple automatiser leurs utilisations.

### 6.1.1 DISKPART

Des dizaines d'outils en ligne de commande permettant de gérer la mémoire de masse existent, mais un en particulier est très puissant et est directement installé sur le système d'exploitation Windows. Cet outil a été créé par Microsoft et se nomme « Diskpart ».

```
DISKPART> ?
Microsoft DiskPart version 10.0.22621.1

ACTIVE          - Indiquer la partition sélectionnée comme étant active.
ADD             - Ajouter un miroir à un volume simple.
ASSIGN          - Assigner une lettre de lecteur ou un point de montage au volume
                 sélectionné.
ATTRIBUTES      - Manipuler les attributs de volume ou de disque.
ATTACH          - Attache un fichier de disque virtuel.
AUTOMOUNT       - Activer et désactiver le montage automatique des volumes de base.
BREAK          - Détruire un jeu de miroir.
CLEAN           - Effacer les informations de configuration ou toutes les
                 informations du disque.
COMPACT         - Tente de réduire la taille physique du fichier.
CONVERT         - Convertir à différents formats de disque.
CREATE          - Créer un volume, une partition ou un disque virtuel.
DELETE          - Supprimer un objet.
DETAIL          - Fournir des détails concernant un objet.
DETACH          - Détache un fichier de disque virtuel.
EXIT            - Quitter DiskPart.
EXTEND          - Étendre un volume.
EXPAND          - Étend la taille maximale disponible sur un disque virtuel.
FILESYSTEMS     - Affiche les systèmes de fichiers en cours et pris en charge sur le volume.
FORMAT          - Formater la partition ou le volume actif.
GPT             - Assigne des attributs à la partition GPT sélectionnée.
HELP           - Imprimer une liste de commandes.
IMPORT          - Importer un groupe de disques.
INACTIVE        - Marquer la partition sélectionnée comme inactive.
LIST            - Afficher une liste des objets.
MERGE           - Fusionne un disque enfant avec ses parents.
ONLINE          - Mettre en ligne un objet qui est actuellement marqué comme hors connexion.
OFFLINE         - Mettre hors connexion un objet actuellement marqué comme en ligne.
RECOVER         - Actualise l'état de tous les disques dans le jeu de disques sélectionné.
                 Tente une récupération sur les disques situés dans le jeu de
                 disques non valide, et resynchronise les volumes en miroir et les
                 volumes RAID5 qui possèdent des données de plex ou de parité
                 périmées.
REM             - Ne fait rien. Utilisé pour commenter des scripts.
REMOVE          - Supprimer une lettre de lecteur ou l'attribution d'un point de
                 montage.
REPAIR          - Réparer un volume RAID-5 avec un membre défectueux.
RESCAN         - Analyser à nouveau l'ordinateur à la recherche de disques et de
                 volumes.
RETAIN          - Placer une partition nominale sous un volume simple.
SAN            - Afficher ou définir la stratégie SAN pour le système d'exploitation actuellement démarré.
SELECT          - Déplacer le focus vers un objet.
SETID           - Changer le type de partition.
SHRINK          - Réduisez la taille du volume sélectionné.
UNIQUEID        - Affiche ou définit l'identificateur de la table de partition GUID (GPT) ou
                 signature de l'enregistrement de démarrage principal (MBR) d'un disque.
```

Figure 95 : commandes de l'outil DISKPART

Quand nous exécutons la commande « DISKPART ? », la console nous renvoie la page d'aide de l'outil. Cet outil possède beaucoup d'arguments qui ont tous une utilité différente.

Par exemple, l'utilisation de l'argument « FORMAT » va permettre de formater un volume spécifié. L'argument « REMOVE » va quant à lui, supprimer une lettre d'un lecteur spécifié. L'argument « CREATE » va créer une partition ou un disque virtuel. L'argument « LIST DISK » qui permet de lister tous les systèmes de stockage présents sur la machine. Etc, etc.

### 6.3 HxD – AUTOPSIE D'UNE CLE USB FAT

HxD est un éditeur hexadécimal utilisé pour visualiser et modifier des fichiers binaires, tels que des fichiers exécutables, des fichiers de données ou des secteurs de disque.

- **Visualisation hexadécimale :** HxD permet de visualiser le contenu d'un fichier sous forme hexadécimale. Chaque octet est affiché en hexadécimal, ce qui est utile pour inspecter les données brutes d'un fichier.
- **Édition de fichiers binaires :** Possibilité de modifier directement le contenu d'un fichier binaire en utilisant HxD.
- **Comparaison de fichiers :** HxD permet de comparer deux fichiers binaires pour repérer les différences entre eux. Cette fonctionnalité est souvent utilisée pour vérifier l'intégrité des fichiers ou pour identifier les changements entre différentes versions d'un fichier.
- **Analyse de structure de fichier :** HxD peut être utilisé pour examiner la structure interne d'un fichier, notamment les en-têtes de fichier, les tables de données et autres structures.

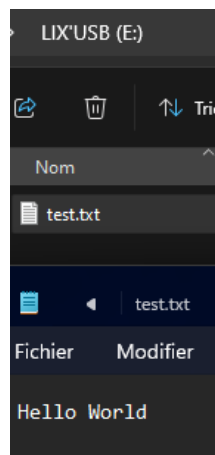


Figure 96 : création d'un fichier .txt de test

J'ai par exemple créé un fichier « test.txt » qui contient la chaîne de caractère « Hello World » dans une clé USB au système de fichier « format » « FAT32 ».

001C880800 48 65 6C 6C 6F 20 57 6F 72 6C 64 00 00 00 00 00 Hello World..... Secteur 9340916

Figure 97 : valeurs hexadécimales du secteur qui contient le fichier .txt de test

On peut voir dans HxD que le fichier « test.txt » est écrit dans le secteur 934916. De plus les valeurs hexadécimales « 48, 65, 6C, 6C, 6F, etc » que nous voyons sont les valeurs hexadécimales correspondantes aux caractères que nous avons insérées dans le fichier. Par exemple si on regarde le premier caractère de notre fichier qui est H et le premier octet du secteur, on peut voir que sa valeur en hexadécimale est « 48 », car la valeur de H en hexadécimale correspond à « 48 », ceci est exactement pareil pour tout ce qui est contenu

dans le fichier, le deuxième octet à une valeur de « 65 », car la valeur de e en hexadécimale est « 65 ».

Un caractère (symbole, lettre, chiffre) est codé sur 8bits soit 1 octet. Donc si nous écrivons un message tel que « Hello World », notre fichier aura une taille de 11 octets et chaque caractère sera codé sur un octet du secteur auquel le fichier est écrit. Donc notre secteur n'aura que 11 octets utilisés sur 512.

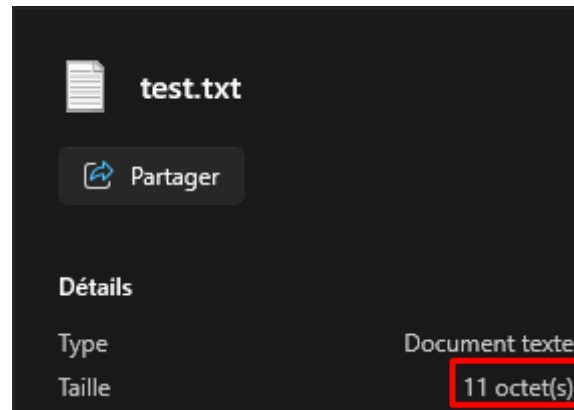


Figure 98 : taille du fichier de test (1)

Si on vérifie la taille du fichier « test.txt », on peut voir qu'il a bien une taille de 11 octets.

Cette fois si, si nous venons rajouter un « ! » après le texte, le fichier devrait faire 12 octets et la valeur hexadécimale du douzième octet devrait correspondre à « 21 » (c'est la valeur hexadécimale d'un « ! »).

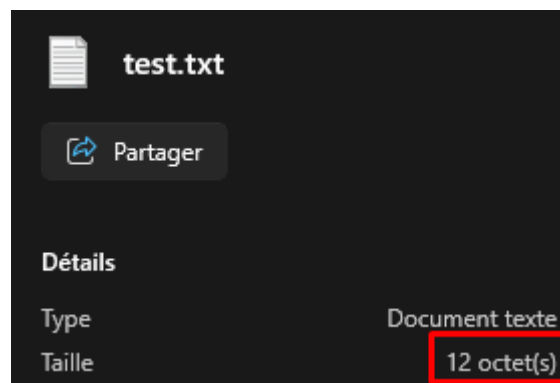


Figure 99 : taille du fichier de test (2)

```
001C880800 48 65 6C 6C 6F 20 57 6F 72 6C 64 21 00 00 00 00 Hello World!.... Secteur 9340916
```

Figure 100 : valeurs hexadécimales du secteur qui contient le fichier de test, après modification

En résumé, le logiciel HxD est un éditeur hexadécimal puissant et polyvalent qui permet d'afficher et de modifier le contenu des secteurs d'un système de stockage, ainsi que le contenu de fichiers de divers formats tels que les fichiers exécutables (PE), les fichiers texte (TXT), les feuilles de calcul (XLSX), les fichiers PDF, les fichiers batch (BAT), et bien d'autres encore.

## 6.4 PAYLOAD DANS UN ALTERNATE DATA STREAM NTFS.

Un ADS (Alternate Data Stream), ou flux de données alternatif en français, est une fonctionnalité spécifique au système de fichier NTFS permettant d'associer des données supplémentaires à un fichier existant, sans modifier le contenu principal du fichier. Cela signifie qu'un fichier peut contenir plusieurs flux de données, dont un principal et plusieurs alternatifs.

Les ADS peuvent être utilisés par des personnes malveillantes pour cacher des scripts, programme malveillant derrière un fichier et que ce programme soit exécuté dès que le flux principal est exécuté. Quand l'utilisateur va ouvrir le fichier, il ne verra que le contenu légitime, mais un script se sera exécuté en font sans que l'utilisateur ne s'en soit aperçus.

Par exemple le fichier test.txt contient la chaîne de caractère « Hello World! ». Via le DOS nous pouvons ajouter un flux supplémentaire au flux principal qui contiendra « test hello world ».

Avant de créer un ADS, vérifions que notre fichier ne contienne pas d'ADS, avec la commande « dir /r » qui permet d'afficher tous les flux supplémentaires des fichiers contenus dans un répertoire.

```
29/03/2024 17:46                12 test.txt
```

Figure 101 : vérification des ADS dans Le fichier de test

On peut voir que le fichier « test.txt » ne contient pas de flux supplémentaire et que sa taille est bien de 12 octets comme nous avons 12 caractères.

Maintenant, créons un flux supplémentaire. Pour ce faire nous avons qu'a exécuté la commande « echo [text] > [targer\_file]:[ads\_name] ». Cette commande va permettre de venir créer un flux supplémentaire dans le fichier cible. Dans ce cas, nous allons ajouter du texte au flux supplémentaire.

```
C:\Users\lixne\Desktop>echo test hello world > test.txt:ads
```

Figure 102 : création d'un ADS dans Le fichier de test

Nous venons de créer un flux supplémentaire appelé « ads » et qui contient le texte « test hello world ».

```
30/03/2024 14:13                12 test.txt
                               19 test.txt:ads:$DATA
```

Figure 103 : vérification de L'ADS dans Le fichier de test

Si on exécute à nouveau la commande « dir /r », on peut voir que notre fichier « test.txt » contient bien un ADS d'une taille de 19 octets.

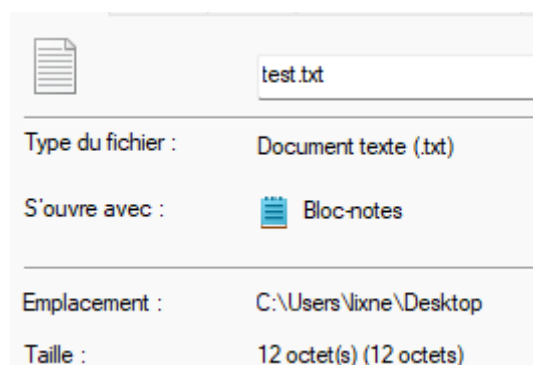
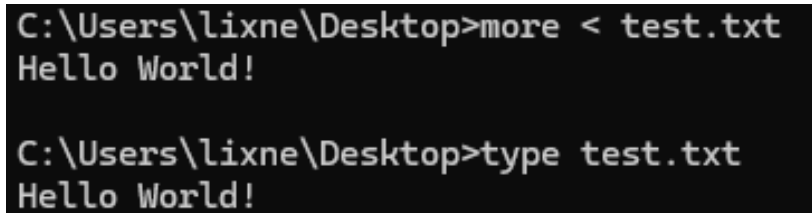


Figure 104 : taille du fichier de test via Les propriétés

En revanche si nous faisons « clique droit -> propriété » sur notre fichier, sa taille n'est que de 12 octets alors qu'avec un flux supplémentaire de 19 octets, elle devrait être de 31 octets. Il en est ainsi, car son ou ses ADS ne sont pas pris en compte puisque nous examinons les propriétés du flux principal et non du ou des flux supplémentaires. Le problème étant que le seul fichier apparent que nous pouvons analyser est le flux principal, mais plusieurs flux supplémentaires peuvent se cacher derrière sans éveiller le moindre soupçon.

Un des grands moyens pour vérifier si un fichier cache des flux supplémentaires est d'exécuter la commande « dir /r [file\_path] ».

**Pour finir, voyons comment afficher le contenu d'un flux supplémentaire d'un fichier.**

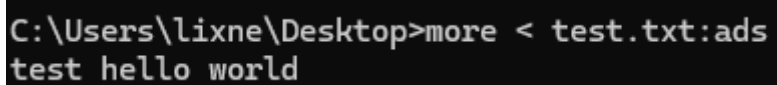


```
C:\Users\lixne\Desktop>more < test.txt
Hello World!

C:\Users\lixne\Desktop>type test.txt
Hello World!
```

*Figure 105 : affichage du contenu du fichier de test*

Pour afficher le flux principal d'un fichier, il nous suffit de taper la commande « type [file\_path] » ou « more < [file\_path] ».



```
C:\Users\lixne\Desktop>more < test.txt:ads
test hello world
```

*Figure 106 : affichage du contenu de l'ADS du fichier ce test*

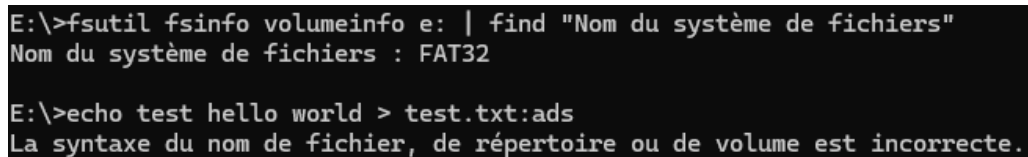
Pour afficher un des flux supplémentaires d'un fichier, il nous suffit d'exécuter la commande « more < [file\_path]:[ads\_name] ». Nous pouvons voir que les données contenues dans le flux supplémentaire sont bien différentes de celles du flux principal.

Cet exemple d'ADS ne fait lien avec aucune intention malveillante, mais est simplement là pour montrer qu'un fichier peut contenir des données supplémentaires cachées et exécutables à tout moment par une personne malveillante.

#### **6.4.1 CE QU'IL FAUT RETENIR D'UN ADS (ALTERNATE DATA STREAM)**

- Il permet d'associer des données supplémentaires à un fichier sans altérer son contenu principal.
- Un ADS est une fonctionnalité spécifique aux systèmes de fichiers NTFS utilisés par Windows :

Création d'un ADS sur un système FAT32 →



```
E:\>fsutil fsinfo volumeinfo e: | find "Nom du système de fichiers"
Nom du système de fichiers : FAT32

E:\>echo test hello world > test.txt:ads
La syntaxe du nom de fichier, de répertoire ou de volume est incorrecte.
```

*Figure 107 : exemple de création d'ADS sur FAT32*

Création d'un ADS sur un système NTFS →

```
C:\Users\lixne\Desktop>fsutil fsinfo volumeinfo c: | find "Nom du système de fichiers"
Nom du système de fichiers : NTFS

C:\Users\lixne\Desktop>echo test hello world > test.txt:ads

C:\Users\lixne\Desktop>
```

Figure 108 : exemple de création d'ADS sur NTFS

- Les ADS peuvent être exploités par des personnes malveillantes pour cacher des scripts ou des programmes malveillants derrière des fichiers, les exécutant lorsque le fichier principal est ouvert.
- Lorsque l'on examine les propriétés d'un fichier, seules les données du flux principal sont prises en compte, pas celles des flux supplémentaires.

## 6.5 AJOUT D'UN SYSTEME DE VERIFICATION DES ADS D'UN FICHIER DANS LE ROBOT D'ANALYSE

Dans cette partie, nous allons rajouter au robot d'analyse une condition qui va vérifier si le fichier à analyser possède des ADS. Après analyse du fichier, si le fichier n'est pas détecté malveillant par l'API Virus Total, notre robot va vérifier s'il contient des ADS, si c'est le cas, il ne sera pas mis en quarantaine, mais un message WARNING pour ce fichier sera écrit dans les logs.

```
def check ADS(file_path):
```

Figure 109 : fonction de vérification des ADS

Pour ce faire nous allons créer une nouvelle fonction appelée « check\_ads » qui prendra en paramètre le chemin d'accès du fichier à tester.

```
file_dir = os.path.dirname(file_path)
file_name = os.path.basename(file_path)
os.chdir(file_dir)
```

Figure 110 : variables principales de la fonction "check\_ads"

On crée une variable « file\_dir » qui récupère le dossier qui contient le fichier en question, une variable « file\_name » qui contient le nom du fichier et son extension et enfin on utilise le module « os » pour changer de dossier courant.

```
streams = []
```

Figure 111 : liste de streams du fichier

On crée ensuite une liste « streams » qui va venir stocker les streams trouvés du fichier à tester.

```
try:
    result = subprocess.Popen(["cmd", "/c", f"dir /r {file_name}"], shell=False, stdout=subprocess.PIPE)
    output = result.stdout.read().decode('latin-1')
    output = output.split("\n")

    for info in output:
        if "$DATA" in info:
            stream = info.replace("\r", "").replace(" ", "")
            streams.append(stream.split(":")[1])
except:
    pass
```

Figure 112 : exécution de la commande "dir /r [file\_path]" via le module subprocess

Puis dans un bloc « try », on vient exécuter la commande « dir /r [file\_path] » et itérer sur les lignes du résultat pour ajouter à notre liste qui contient les streams, le nom des streams du fichier.

```
if len(streams) != 0:
    log_content = str(datetime.datetime.now()) + " : " + "WARNING : " + file_path + " as " + str(len(streams)) + "Streams ! | Name of its Streams : " + ",".join(streams) + "\n"
    with open(LOGS + "/log.log", "a") as log:
        log.write(log_content)
    print("WARNING ! ADS found in file : " + file_path)
```

Figure 113 : écriture des logs dans Le fichier de logs et affichage de message dans la console

Ensuite si la liste de streams n'est pas vide (le fichier contient des streams), on écrit le message de log en WARNING dans le fichier de logs puis on affiche dans la console que le fichier contient des ADS.

```
if analysis_json['attributes']['stats']['malicious'] > 0 or analysis_json['attributes']['stats']['suspicious'] > 0:
    scan_elements.append(element_path)
    print("Suspicious file found : " + element_path)
else:
    check_ADS(element_path)
```

Figure 114 : ajout de l'appel de la fonction « check\_ads » dans la fonction "check\_suspicious\_file"

Enfin on vient ajouter un « else » à notre condition qui vérifie si le fichier est malveillant ou non dans notre fonction « check\_suspicious\_file ». Dans le cas où le fichier n'est pas malveillant alors on vérifie quand même s'il possède des ADS ou pas.

```
30/03/2024 19:03          10 ads.txt
                        23 ads.txt:secondary_stream:$DATA
                        11 ads.txt:third_stream:$DATA
```

Figure 115 : vérification des ADS du fichier de test

Pour le test j'ai créé un fichier « ads.txt » avec deux ADS, le premier nommé « secondary\_stream » et le second nommé « third\_stream ».

```
Analysis in progress...
WARNING ! 2 ADS found in file : C:/Users/lixne/Desktop/Person/Gael/cours_pro/BTS_SIO/1/VALENTI/WindowsMalwaresHunter/src/V2/test/files/ads/ads.txt
No suspicious files found.
Analysis completed.
```

Figure 116 : console après analyse

On peut bien voir après analyse qu'un message d'avertissement concernant le fichier en question a été affiché dans la console.

```
2024-03-30 19:03:52.142473 : WARNING : ads.txt as 2 Streams ! | Name of its Streams : secondary_stream, third_stream
```

Figure 117 : fichier logs après analyse d'ADS dans Le fichier test

Dans le fichier de logs, on retrouve aussi l'avertissement concernant ce fichier avec les noms de tous les ADS qu'il possède.



## 7 LA SEQUENCE DE BOOT

### 7.1 BOOT DE LA MACHINE

Le "boot" d'une machine fait référence au processus de démarrage de l'ordinateur ou de tout autre appareil électronique. Lorsqu'on allume un appareil, le processus de boot commence, impliquant généralement l'exécution d'un programme appelé "boot loader" ou "gestionnaire de démarrage". Ce programme charge le système d'exploitation (comme Windows, macOS, Linux, etc.) en mémoire et initialise les composants matériels essentiels de l'ordinateur, permettant ainsi à l'appareil de fonctionner et d'être opérationnel pour l'utilisateur. Le processus de boot comprend généralement plusieurs étapes, telles que la vérification du matériel, la configuration des paramètres système et le chargement du système d'exploitation.

Voici un diagramme présentant une séquence de boot d'une machine.

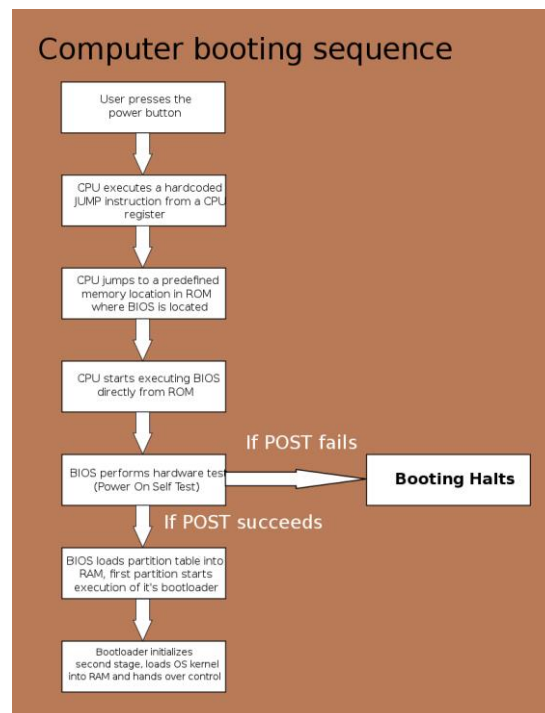


Figure 118 : diagramme de séquence de boot

Une fois l'appareil sous tension, l'alimentation envoie un signal « power-good ou power-ok » à la carte mère qui lance l'initialisation du processeur. Ensuite le processeur exécute une instruction JUMP codée en dur à partir du registre du **CPU**<sup>5</sup>. Après cette étape le CPU saute à un emplacement mémoire prédéfini dans la **ROM**<sup>6</sup> où se trouve le **BIOS**<sup>7</sup>. Le CPU vient exécuter le BIOS directement depuis la ROM. Ensuite le bios vient effectuer un test matériel, s'il y a une erreur durant ce test cela provoque l'arrêt du démarrage. Si les tests ont correctement été effectués, le BIOS va charger la table de partition dans la **RAM**<sup>8</sup> et la première partition

<sup>5</sup> Central processing unit ou Unité centrale de traitement en français : ensemble qui comprend l'unité arithmétique et logique (ALU), le registre qui stocke temporairement les données en cours de traitement et l'unité de contrôle qui interprète les instructions et coordonne le flux de données.

<sup>6</sup> Read-only Memory ou Mémoire en lecture seule en français : type de mémoire qui stocke de donnée qui ne doivent pas être modifiée ou effacée (comme le bios).

<sup>7</sup> Basic Input / Output System ou Système d'entrée/sortie de base en français : programme stocké sur une puce ROM sur la carte mère pour effectuer des fonctions essentielles nécessaires à l'initialisation de l'appareil.

<sup>8</sup> Random access Memory ou Mémoire à Accès aléatoire en français : type de mémoire volatile utilisée par tout autre appareil électronique pour stocker des données temporaires et des instructions nécessaires au fonctionnement des programmes en cours d'exécution.

exécute le **BOOTLOADER**<sup>9</sup>. Enfin le bootloader initialise la deuxième étape, charger le **Kernel**<sup>10</sup> du système d'exploitation dans la RAM et transféré le contrôle au kernel.

### **7.1.1 BIOS vs UEFI**

Le BIOS (Basic Input/Output System) et l'UEFI (Unified Extensible Firmware Interface) sont tous deux des interfaces logicielles utilisées au démarrage d'un ordinateur pour initialiser le matériel et charger le système d'exploitation.

En revanche, ils présentent des différences significatives en termes de fonctionnalités, de performances et de compatibilité.

#### **BIOS (Basic Input/Output System) :**

- **Interface** : Le BIOS est une interface logicielle plus ancienne, basée sur une architecture héritée, souvent textuelle et avec des options de configuration limitées.
- **Limitations** : Le BIOS est limité en termes de capacités de démarrage, de support matériel et de sécurité. Il utilise un système de partitionnement de disque hérité (MBR) et ne peut pas démarrer à partir de disques durs de grande capacité (supérieure à 2 To) ni de dispositifs de stockage plus modernes tels que les disques SSD NVMe.
- **Compatibilité** : Le BIOS est largement compatible avec les anciens systèmes d'exploitation et le matériel hérité.

#### **UEFI (Unified Extensible Firmware Interface),**

- **Interface** : L'UEFI est une interface plus moderne, graphique et interactive, offrant une expérience utilisateur améliorée avec plus d'options de configuration et de sécurité.
- **Fonctionnalités** : L'UEFI offre des fonctionnalités avancées telles que le support de l'interface graphique, le démarrage sécurisé (Secure Boot), la prise en charge des disques de grande capacité, la compatibilité avec les disques GPT (GUID Partition Table), le réseau et les protocoles de gestion à distance.
- **Performances** : L'UEFI peut offrir des temps de démarrage plus rapides et une meilleure gestion de l'énergie que le BIOS.
- **Compatibilité** : L'UEFI est conçu pour prendre en charge les nouveaux systèmes d'exploitation et matériels, mais il peut également être compatible avec les anciens systèmes d'exploitation grâce à la prise en charge du mode de compatibilité.

En résumé, bien que le BIOS soit une technologie éprouvée et largement utilisée, l'UEFI offre des fonctionnalités plus avancées, une meilleure compatibilité avec les périphériques modernes et une sécurité renforcée.

---

<sup>9</sup> BOOTLOADER ou chargeur d'amorçage en français : programme logiciel qui s'exécute au démarrage d'un appareil électronique et qui a pour but de démarrer le système d'exploitation.

<sup>10</sup> Kernel ou Noyau en français : agit comme une couche intermédiaire entre le matériel et les logiciels applicatifs, permettant la communication et la gestion des ressources matérielles de l'appareil.

### 7.1.2 MBR - MASTER BOOT RECORD

#### TABLE DES PARTITIONS DANS LE MBR / GPT - GLOBALLY UNIQUE IDENTIFIER PARTITION TABLE

Le MBR (Master Boot Record) et le GPT (GUID Partition Table) sont deux méthodes de partitionnement de disque utilisées pour diviser et organiser l'espace de stockage sur un disque dur ou un SSD.

#### MBR (Master Boot Record),

- **Taille maximale du disque :** Le MBR utilise un schéma de partitionnement hérité qui limite la taille maximale du disque à 2 To (téraoctets).
- **Structure :** Le MBR contient une table de partitionnement de 64 octets située dans le premier secteur du disque dur. Il peut prendre en charge jusqu'à 4 partitions primaires, ou 3 partitions primaires et une partition étendue pouvant contenir plusieurs partitions logiques.
- **Compatibilité :** Le MBR est largement pris en charge par les anciens systèmes d'exploitation et les BIOS hérités. Il est couramment utilisé sur les systèmes plus anciens et les disques de petite taille.
- **Limitations :** Outre la limitation de taille du disque, le MBR ne prend pas en charge certaines fonctionnalités modernes telles que le démarrage sécurisé (Secure Boot), la prise en charge de plus de quatre partitions primaires sans une partition étendue, etc.

#### GPT (GUID Partition Table),

- **Taille maximale du disque :** Le GPT utilise un schéma de partitionnement modernisé qui prend en charge des disques de grande capacité, allant jusqu'à plusieurs pétaoctets (PB).
- **Structure :** Le GPT contient une table de partitionnement plus robuste avec une capacité de stockage de 128 partitions par défaut (et jusqu'à 18,4 millions de milliards de partitions possibles). Il prend en charge à la fois les partitions primaires et les partitions de secours (backup).
- **Compatibilité :** Le GPT nécessite un système d'exploitation 64 bits et un firmware UEFI pour démarrer. Il est de plus en plus utilisé sur les nouveaux systèmes et est compatible avec les systèmes d'exploitation modernes tels que Windows 10, macOS, Linux, etc.
- **Fonctionnalités :** Le GPT offre des fonctionnalités avancées telles que le démarrage sécurisé (Secure Boot), la prise en charge de disques de grande capacité, la redondance des tables de partitionnement pour la fiabilité des données, etc.

Le GPT est une solution plus moderne et plus performante offrant une plus grande capacité de stockage, une meilleure fiabilité des données et des fonctionnalités avancées pour les systèmes modernes.

### 7.1.3 BS - BOOT SECTOR (BS)

Le terme "BS" ou "Boot Sector" en anglais, "secteur de démarrage" en français est une petite zone située au début d'un périphérique de stockage, tel qu'un disque dur, un SSD, etc., utilisé pour démarrer le système d'exploitation lors du processus de démarrage de l'ordinateur.

Le BS contient un code de démarrage, parfois appelé code amorce (boot code). Ce code est exécuté par le processeur dès que l'ordinateur est mis sous tension. Son rôle est d'initialiser le processus de démarrage en chargeant le chargeur de démarrage (BOOTLOADER). Le secteur de démarrage (BS) peut également contenir des informations sur la table de partitions du disque comme des détails sur les partitions présentes sur le disque, tel que leurs emplacements, leurs tailles, leurs types, etc.

À la fin d'un secteur de démarrage, il y a généralement une signature spéciale qui indique que ce secteur est un secteur de démarrage valide. Cette signature est utilisée par le BIOS ou L'UEFI pour s'assurer que le secteur de démarrage est authentique et peut être utilisé pour amorcer le système d'exploitation.

Lorsque l'ordinateur démarre, le BIOS ou L'UEFI recherche le secteur de démarrage du périphérique de stockage désigné comme périphérique de démarrage. Une fois trouvé, le BIOS ou l'UEFI charge le code de démarrage à partir du secteur de démarrage dans la mémoire et le transfère au CPU pour exécution. Le code de démarrage peut ensuite charger le BOOTLOADER, qui à son tour charge le système d'exploitation, permettant ainsi à l'ordinateur de démarrer complètement.

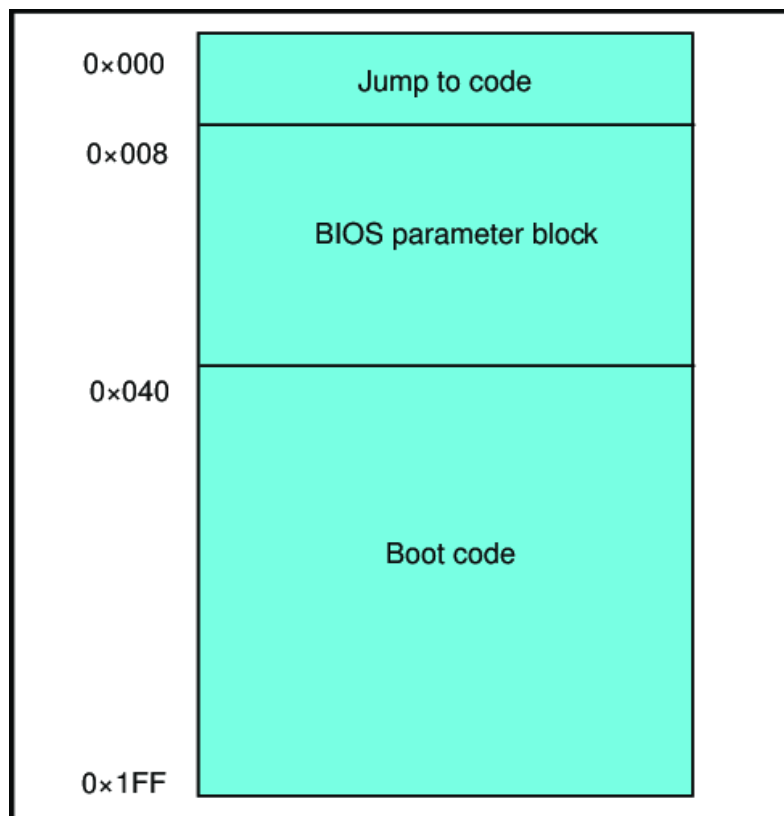


Figure 119 : diagramme du BS

Ce schéma montre la structure typique d'un secteur de démarrage.

## 7.2 LA SEQUENCE DE BOOT DE WINDOWS

### 7.2.1 BCDEDIT

BCDEDIT est une commande utilisée dans les OS Windows. Elle permet de manipuler les données de configuration du magasin de démarrage du système (Boot Configuration Data - BCD). Ces données incluent les informations sur les systèmes d'exploitation installés, les options de démarrage avancées, les paramètres de démarrage, etc. En utilisant BCDEDIT, les utilisateurs peuvent modifier les paramètres de démarrage de leur système d'exploitation, comme les options de démarrage, les paramètres de débogage, etc. C'est un outil avancé principalement utilisé par les administrateurs système ou les utilisateurs avancés pour gérer le démarrage des machines.

En lançant la console en tant qu'Administrateur, et en entrant la commande « bcdedit », la console nous renvoie des informations sur le Gestionnaire de démarrage de Windows et le Chargeur de démarrage de Windows.

```
C:\Windows\System32>bcdedit

Gestionnaire de démarrage Windows
-----
identificateur      {bootmgr}
device              partition=\Device\HarddiskVolume3
path                \EFI\MICROSOFT\BOOT\BOOTMGFW.EFI
description          Windows Boot Manager
locale              fr-FR
inherit              {globalsettings}
default              {current}
resumeobject         {1074bb20-6dce-11ee-bd89-e0cf29a72c67}
displayorder         {current}
toolsdisplayorder    {memdiag}
timeout              30

Chargeur de démarrage Windows
-----
identificateur      {current}
device              partition=C:
path                \Windows\system32\winload.efi
description          Windows 11
locale              fr-FR
inherit              {bootloadersettings}
recoverysequence     {1074bb22-6dce-11ee-bd89-e0cf29a72c67}
displaymessageoverride Recovery
recoveryenabled      Yes
isolatedcontext      Yes
allowedinmemorysettings 0x15000075
osdevice             partition=C:
systemroot           \Windows
resumeobject         {1074bb20-6dce-11ee-bd89-e0cf29a72c67}
nx                   OptOut
bootmenupolicy        Standard
hypervisorlaunchtype Off
```

Figure 120 : exécution de la commande bcdedit

Dans le Gestionnaire de démarrage Windows, nous pouvons voir des informations telles que :

- **Identificateur** : Identifiant unique attribué à chaque entrée de démarrage dans le Gestionnaire de démarrage Windows. Cet identifiant est utilisé pour référencer spécifiquement une entrée lors de l'exécution de commandes BCDEDIT.
- **Device** : cette option indique le périphérique physique ou logique à partir duquel le système d'exploitation doit être chargé. Il spécifie l'emplacement du volume contenant les fichiers nécessaires au démarrage du système.

- **Path** : il s'agit du chemin d'accès au fichier qui contient les informations de démarrage pour cette entrée. Il indique où se trouvent les fichiers nécessaires au démarrage du système sur le périphérique spécifié par l'option "Device".
- **Description** : C'est un champ textuel qui décrit l'entrée de démarrage.
- **Locale** : Cette option spécifie les paramètres régionaux utilisés pour l'entrée de démarrage. Elle indique la langue et les paramètres régionaux utilisés lors du chargement du système d'exploitation.
- **Inherit** : il s'agit d'une option qui indique si cette entrée hérite des paramètres globaux du Gestionnaire de démarrage Windows ou si elle possède ses propres paramètres spécifiques.
- **Default** : c'est un indicateur qui indique si cette entrée est l'option de démarrage par défaut. Le système démarrera automatiquement à partir de cette entrée s'il n'y a pas d'intervention de l'utilisateur.
- **Resumeobjet** : Cette option est utilisée pour spécifier l'objet de reprise qui est utilisé lorsque le système d'exploitation est mis en veille prolongée.
- **Displayorder** : Il s'agit de l'ordre dans lequel les entrées de démarrage sont affichées dans le menu de démarrage. Cet ordre détermine l'ordre dans lequel les options de démarrage sont présentées à l'utilisateur.
- **Toolsdisplayorder** : C'est similaire à "Displayorder", mais spécifique aux outils de récupération ou aux options avancées de démarrage.
- **Timeout** : c'est la durée, en secondes, pendant laquelle le menu de démarrage est affiché avant que le système ne démarre automatiquement à partir de l'option par défaut.

Enfin le chargeur de démarrage Windows nous renvoie ces informations :

- **Identificateur** : C'est un identifiant unique attribué à chaque entrée de démarrage dans le Chargeur de démarrage Windows. Cet identifiant est utilisé pour référencer spécifiquement une entrée lors de l'exécution de commandes ou d'opérations de gestion du démarrage.
- **Device** : cette option indique le périphérique physique ou logique à partir duquel le système d'exploitation est chargé.
- **Path** : il s'agit du chemin d'accès au fichier qui contient les informations de démarrage pour cette entrée.
- **Description** : C'est un champ textuel qui décrit l'entrée de démarrage. Il peut s'agir du nom du système d'exploitation ou d'une description de l'option de démarrage.
- **Locale** : Cette option spécifie les paramètres régionaux utilisés pour l'entrée de démarrage. Elle indique la langue et les paramètres régionaux utilisés lors du chargement du système d'exploitation.
- **Inherit** : il s'agit d'une option qui indique si cette entrée hérite des paramètres globaux du Chargeur de démarrage Windows ou si elle possède ses propres paramètres spécifiques.
- **Recoverysequence** : Cet élément spécifie la séquence de récupération à utiliser en cas de problème de démarrage.
- **Displaymessageoverride** : Cet élément permet de remplacer le message d'affichage par défaut lors du démarrage.

- **Recoveryenabled** : Cette option indique si la fonction de récupération automatique du système est activée ou désactivée pour cette entrée de démarrage.
- **Isolatedcontext** : Il s'agit d'une option qui indique si le contexte de démarrage est isolé ou partagé avec d'autres entrées de démarrage.
- **Allowedinmemorysettings** : cette option spécifie les paramètres de mémoire autorisés pour cette entrée de démarrage.
- **Osdevice** : Il s'agit du périphérique contenant le système d'exploitation principal.
- **Systemroot** : C'est le chemin d'accès au répertoire racine du système d'exploitation.
- **Resumeobject** : Cette option spécifie l'objet de reprise à utiliser lorsque le système d'exploitation est mis en veille prolongée.
- **Nx** : Cet élément indique si la protection d'exécution interdite (NX) est activée.
- **Bootmenupolicy** : Cette option spécifie le type de menu de démarrage à utiliser.
- **Hypervisorlaunchtype** : Il s'agit de l'option qui spécifie le type de lancement de l'hyperviseur.

### 7.2.2 MSConfig

MSConfig « Microsoft System Configuration Utility », est un utilitaire système intégré dans les versions de Windows qui permet aux utilisateurs de gérer divers aspects de la configuration du système

Voici un aperçu des fonctionnalités principales de MSConfig :

- **Configuration du démarrage** : MSConfig permet de gérer les programmes et les services qui se lancent automatiquement au démarrage de Windows.
- **Onglet Services** : Cet onglet permet de gérer les services système qui s'exécutent en arrière-plan sur un ordinateur.
- **Outils supplémentaires** : MSConfig comprend également plusieurs outils supplémentaires, tels que le Gestionnaire de démarrage, qui permet de modifier les paramètres de démarrage avancés, et l'Éditeur du Registre, qui permet d'accéder et de modifier les clés de registre système, etc.
- **Diagnostic de démarrage** : Il offre une option de démarrage sélectif qui vous permet de démarrer Windows avec un ensemble minimal de pilotes et de services.
- **Gestionnaire de démarrage** : Cela permet de modifier les paramètres de démarrage avancés, tels que le délai d'affichage du menu de démarrage, les options de démarrage sécurisé, etc.
- **Informations système** : MSConfig fournit des informations détaillées sur le matériel, les logiciels, les pilotes et les services installés sur votre système.

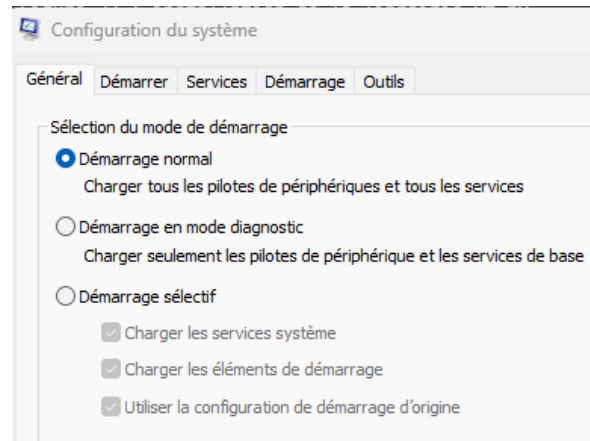


Figure 121 : Onglet "Général" de L'outil MSConig

Dans le menu "Général", on peut paramétrer si on veut lancer le système d'exploitation en mode diagnostique, ou alors en mode sélectif.

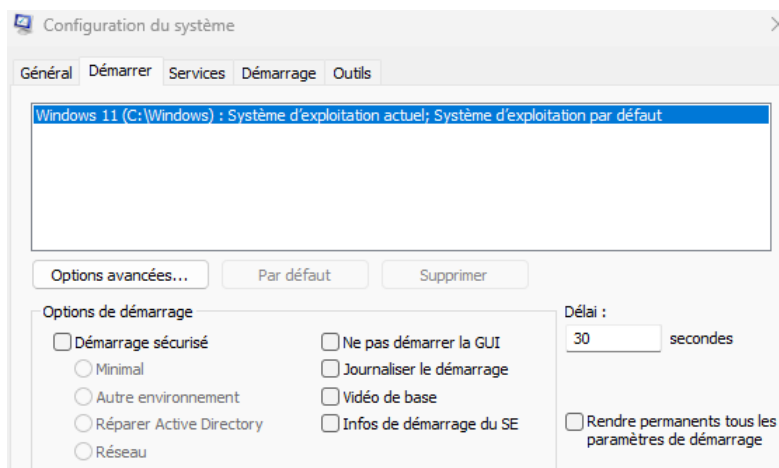


Figure 122 : Onglet "Démarrer" de L'outil MSConig

Dans le menu Démarrer on peut paramétrer plusieurs options pour le démarrage de notre système d'exploitation. Comme désactivé le démarrage du GUI, faire des logs du démarrage, diminuer ou augmenter le délai de démarrage.

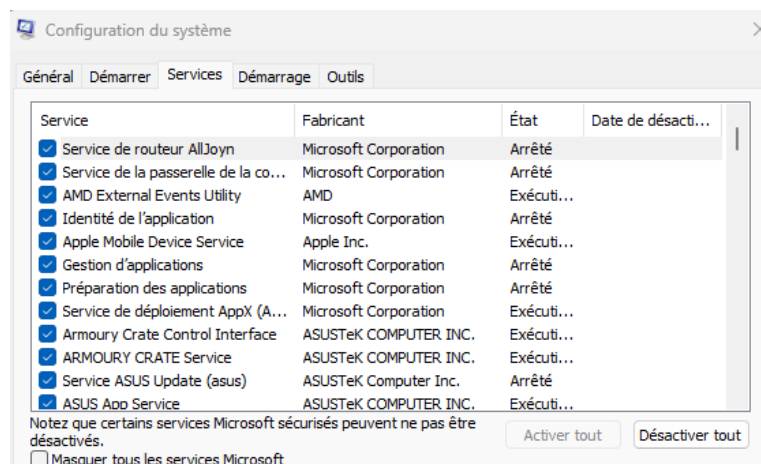


Figure 123 : Onglet "Services" de L'outil MSConig

Dans l'onglet Services on peut venir désactiver certains services du système d'exploitation, que ce soient des services de Microsoft ou des services tierces.



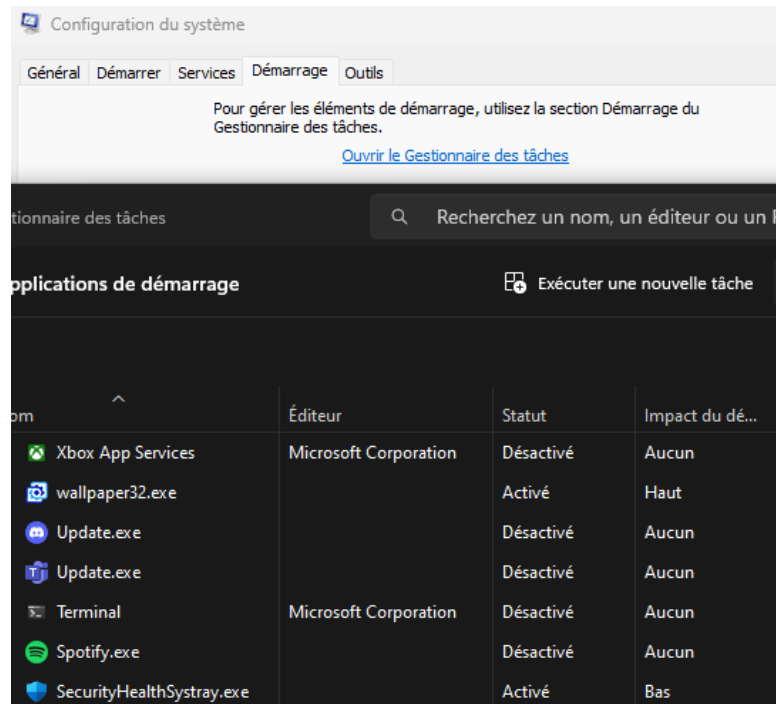


Figure 124 : Onglet "Démarrage" de l'outil MSConfig

Via l'onglet Démarrage on peut ouvrir le Gestionnaire des tâches pour gérer les processus qui se lancent au démarrage de notre système d'exploitation.

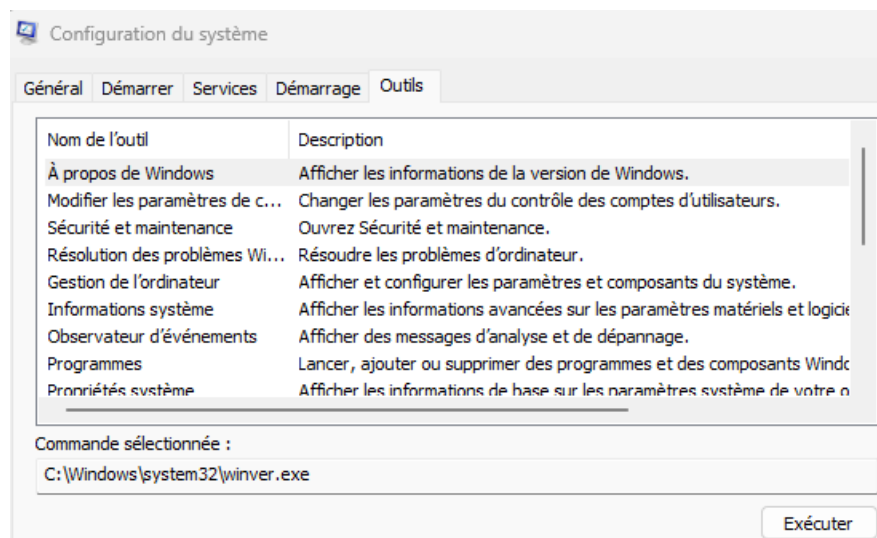


Figure 125 : Onglet "Outils" de l'outil MSConfig

Dans l'onglet outils on peut voir tous les outils Windows avec leurs chemins d'accès comme « regedit, cmd, msconfig, etc »

Pour finir pour ce qui est du boot, MSConfig offre une très grande plage de fonctionnalités permettant de paramétrer le démarrage de notre machine. Il permet de gérer la Gestion des programmes au démarrage, contrôler les services système, diagnostic de démarrage sélectif, gestionnaire de démarrage avancé, diagnostics système complets, enregistrement des logs lors du démarrage du système.

### 7.2.3 AUTORUNS

Autoruns est un outil complet et extrêmement utile, développé par Microsoft et fourni dans la suite d'outils SysInternals, conçu pour permettre aux utilisateurs de visualiser et de contrôler les programmes, les pilotes, les tâches planifiées, les services, les modules complémentaires de navigateur et bien plus encore, qui se lancent automatiquement au démarrage de Windows et lors de la connexion de l'utilisateur.

**Visualisation complète des éléments de démarrage :** Autoruns offre une vue de tous les endroits où des programmes et des composants peuvent être configurés pour se lancer automatiquement au démarrage de Windows.

**Contrôle précis sur les éléments de démarrage :** Les utilisateurs peuvent facilement activer ou désactiver les éléments de démarrage individuels à l'aide d'une interface conviviale. Cela permet de réduire la charge de démarrage de Windows, d'accélérer le temps de démarrage et d'améliorer les performances globales du système.

**Détection des menaces :** Autoruns est également un outil puissant pour détecter les rootkits et d'autres logiciels malveillants qui tentent de se cacher dans les processus de démarrage de Windows.

**Les assets :** Les assets font référence aux différents éléments de démarrage que Windows utilise pour initialiser son environnement. Autoruns permet de visualiser ces assets de manière détaillée, offrant un aperçu complet de la manière dont le système est configuré pour démarrer.

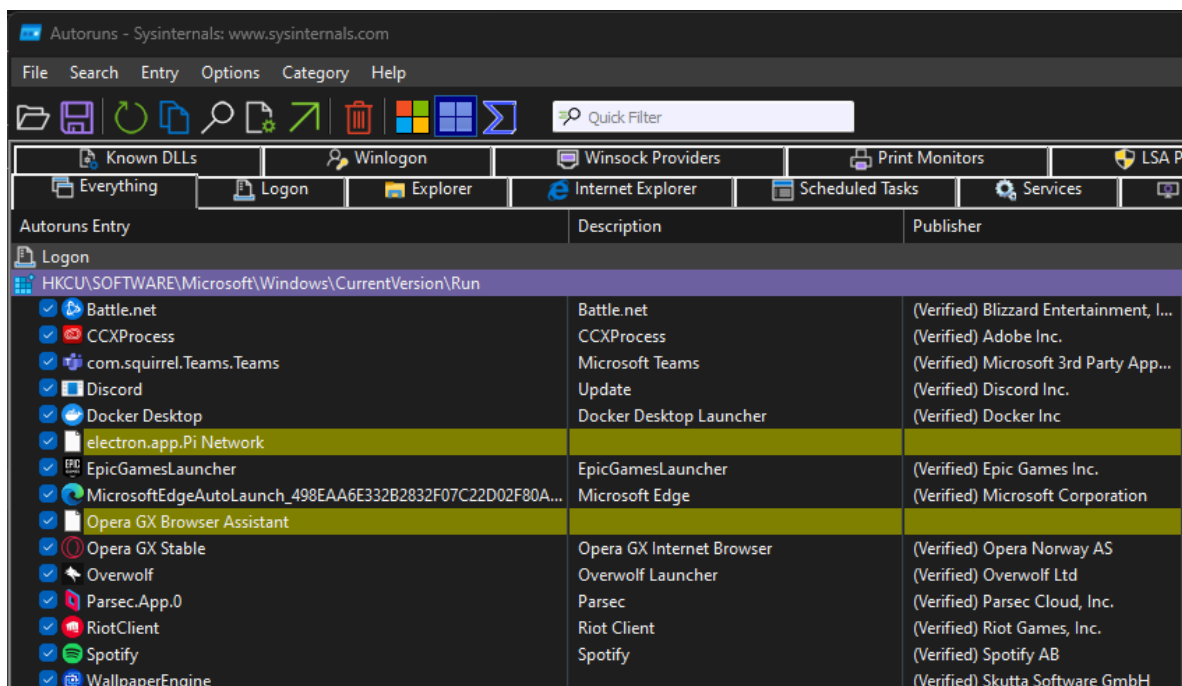


Figure 126 : interface du logiciel Autorun

Sur l'interface d'Autorun nous pouvons voir tous les processus qui ont été chargés par le système d'exploitation, que ce soient des processus / des services / des DLL /, etc. de Microsoft ou autre assets tiers.

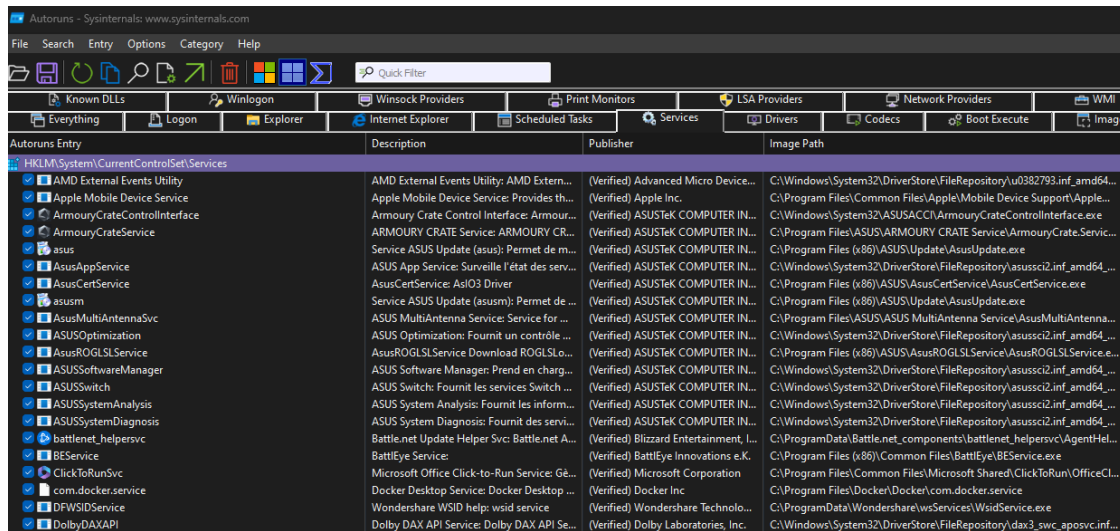


Figure 127 : Onglet "Services" du logiciel Autorun

Par exemple dans l'onglet « Services » nous pouvons voir tous les services qui ont été chargés au boot ou au login, depuis quel endroit dans la machine, s'ils sont vérifiés (signé) pour plus facilement repérer un virus et quand est-ce qu'ils ont été chargés avec le timestamp .

En résumé Autorun est un outil puissant pour analyser une grande partie du système d'exploitation et filtrer de manière efficace certain assets pour ainsi vérifier d'où ils proviennent, par qui, quand, comment, etc. Il est notamment très puissant et efficace pour trouver des malwares qui se lanceraient au lancement de l'OS ou lors d'une connexion à la machine.

## 8 BIBLIOGRAPHIE

<https://chat.openai.com>  
<https://www.microsoft.com>  
<https://fr.wikipedia.org>

## 9 TABLE DES ILLUSTRATIONS

Figure 1 : Diagramme d'état d'un processus .....	4
Figure 2 : Capture d'écran du logiciel Process Explorer .....	4
Figure 3 : Informations du processus .....	5
Figure 4 : Propriété de la mémoire du processus .....	6
Figure 5 : Propriété des handles d'un processus .....	7
Figure 6 : détail des handles d'un processus .....	7
Figure 7 : détail des threads d'un processus .....	8
Figure 8 : Capture d'écran du logiciel Process Monitor .....	8
Figure 9 : capture d'écran du script Python .....	9
Figure 10 : Ajout d'un filtre de type Process Name .....	9
Figure 11 : Ajout d'un filtre de type Opération .....	9
Figure 12 : Capture des opérations d'un processus .....	10
Figure 13 : Ajout des filtres .....	11
Figure 14 : Screenshot de la capture de lecture du fichier de préférence utilisateur de Firefox .....	11
Figure 15 : Valeur de la page d'accueil de démarrage .....	11
Figure 16 : Ajout des filtres .....	11
Figure 17 : Script python responsable de la modification de l'URL de la page d'accueil ....	12
Figure 18 : Événement de modification du fichier "pref.js" .....	12
Figure 19 : Ajout des filtres .....	12
Figure 20 : Modification du DNS .....	13
Figure 21 : Événement de modification du DNS .....	13
Figure 22 : Interface réseau du registre Windows .....	13
Figure 23 : Valeur de l'interface réseau Wi-Fi .....	13
Figure 24 : Ajout des filtres .....	14
Figure 25 : Script du malware Python .....	14
Figure 26 : Capture de la modification du DNS .....	14
Figure 27 : installation de la librairie vt-py .....	18
Figure 28 : installation de la librairie vonage .....	18
Figure 29 : importation des modules .....	19
Figure 30 : définition des constantes de configuration du script .....	19
Figure 31 : création du client de VirusTotal .....	19
Figure 32 : création du client Vonage .....	19
Figure 33 : création de la fonction check_suspicious_files .....	19
Figure 34 : création de la fonction read_config .....	20
Figure 35 : création de la fonction make_dir .....	20
Figure 36 : création de la fonction des variables d'initialisation .....	20
Figure 37 : boucle for pour les chemins d'accès .....	20
Figure 38 : boucle for sur tous les éléments du chemin d'accès itéré .....	21
Figure 39 : ouverture du fichier itéré .....	21
Figure 40 : analyse du résultat du fichier analysé .....	21
Figure 41 : système récursif .....	21
Figure 42 : création des répertoires généraux et fermeture du client VirusTotal .....	22
Figure 43 : déplacement des virus dans le dossier de quarantaine et écriture des logs ....	22
Figure 44 : envoi du message .....	22
Figure 45 : vérification de l'envoi du message .....	22
Figure 46 : s'il n'y a aucuns virus .....	23
Figure 47 : vérification du lancement du script .....	23
Figure 48 : network diagram .....	23
Figure 49 : onglet général du planificateur d'une nouvelle tâche .....	24
Figure 50 : onglet déclencheur de la nouvelle tâche .....	24
Figure 51 : onglet action de la nouvelle tâche .....	24
Figure 52 : configuration du fichier de configuration .....	25

Figure 53 : environnement avant le lancement de l'analyse .....	25
Figure 54 : lancement de l'analyse .....	25
Figure 55 : message du script de fin d'analyse .....	25
Figure 56 : SMS envoyé par le script .....	26
Figure 57 : répertoire test après analyse .....	26
Figure 58 : répertoire quarantaine après analyse .....	26
Figure 59 : fichier de log après analyse .....	26
Figure 60 : répertoire data après analyse .....	26
Figure 61 : infrastructure PKI .....	28
Figure 62 : Structure d'un fichier PE .....	29
Figure 63 : exécution de signCheck .....	30
Figure 64 : fichier non signé .....	30
Figure 65 : signature effectuée .....	31
Figure 66 : fichier singé .....	31
Figure 67 : vérification de la signature numérique .....	31
Figure 68 : processus signé .....	32
Figure 69 : processus signé, mais pas vérifié .....	32
Figure 70 : processus non signé .....	32
Figure 71 : signataire vérifié .....	32
Figure 72 : détail d'une signature numérique .....	33
Figure 73 : détail d'un OID .....	33
Figure 74 : création des erreurs .....	34
Figure 75 : importation des librairies .....	34
Figure 76 : création de la classe FILEValidator .....	34
Figure 77 : création des prototypes des méthodes .....	35
Figure 78 : création de la méthode is_pe .....	35
Figure 79 : création de la méthode is_signed .....	35
Figure 80 : création de la méthode sign_file .....	36
Figure 81 : importation des librairies .....	36
Figure 82 : création du script de test unitaire .....	36
Figure 83 : fichier à tester .....	37
Figure 84 : résultat du test unitaire .....	37
Figure 85 : vérification de la signature numérique du fichier de test .....	37
Figure 86 : vérification du certificat .....	37
Figure 87 : import des modules .....	37
Figure 88 : initialisation de l'instance .....	38
Figure 89 : création de la fonction de check .....	38
Figure 90 : ajout de la condition avec la fonction de check .....	38
Figure 91 : jeu d'essai avant analyse .....	38
Figure 92 : résultat de l'analyse .....	39
Figure 93 : jeu essai après analyse .....	39
Figure 94 : dossier de quarantaine après analyse .....	39
Figure 95 : commandes de l'outil DISKPART .....	41
Figure 96 : création d'un fichier .txt de test .....	42
Figure 97 : valeurs hexadécimales du secteur qui contient le fichier .txt de test .....	42
Figure 98 : taille du fichier de test (1) .....	43
Figure 99 : taille du fichier de test (2) .....	43
Figure 100 : valeurs hexadécimales du secteur qui contient le fichier de test, après modification .....	43
Figure 101 : vérification des ADS dans le fichier de test .....	44
Figure 102 : création d'un ADS dans le fichier de test .....	44
Figure 103 : vérification de l'ADS dans le fichier de test .....	44
Figure 104 : taille du fichier de test via les propriétés .....	44
Figure 105 : affichage du contenu du fichier de test .....	45
Figure 106 : affichage du contenu de l'ADS du fichier ce test .....	45
Figure 107 : exemple de création d'ADS sur FAT32 .....	45
Figure 108 : exemple de création d'ADS sur NTFS .....	46
Figure 109 : fonction de vérification des ADS .....	46
Figure 110 : variables principales de la fonction "check_ads" .....	46
Figure 111 : liste de streams du fichier .....	46
Figure 112 : exécution de la commande "dir /r [file_path]" via le module subprocess .....	46
Figure 113 : écriture des logs dans le fichier de logs et affichage de message dans la console .....	47

Figure 114 : ajout de l'appel de la fonction « check_ads » dans la fonction "check_suspicious_file" .....	47
Figure 115 : vérification des ADS du fichier de test .....	47
Figure 116 : console après analyse .....	47
Figure 117 : fichier logs après analyse d'ADS dans le fichier test .....	47
Figure 118 : diagramme de séquence de boot .....	48
Figure 119 : diagramme du BS .....	51
Figure 120 : exécution de la commande bcdedit .....	52
Figure 121 : Onglet "Général" de l'outil MSConig .....	55
Figure 122 : Onglet "Démarrer" de l'outil MSConig .....	55
Figure 123 : Onglet "Services" de l'outil MSConig .....	55
Figure 124 : Onglet "Démarrage" de l'outil MSConig .....	56
Figure 125 : Onglet "Outils" de l'outil MSConig .....	56
Figure 126 : interface du logiciel Autorun .....	57
Figure 127 : Onglet "Services" du logiciel Autorun .....	58