**Introduction**

Dans le cadre du homework, nous nous sommes intéressés à la réalisation d’un Latching mute [https://support.biamp.com/Tesira/Programming/Muting\_microphones\_with\_logic], en d’autres termes, le verrouillage muet (ou "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 (exécuté), et restera verrouillé dans cet état jusqu'à ce que ce toggle soit à nouveau réactivé.

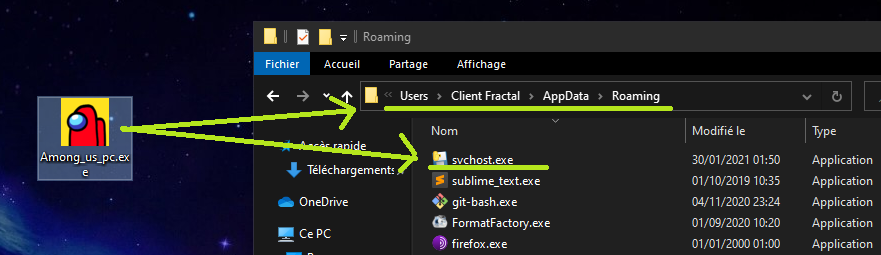
Dans ce rapport, nous présenterons notre solution type qui consiste en un Latching mute dans le but de répondre aux objectifs de l’énoncée du homework qui comprend :

* Procédure d’installation
* La prise de contrôle du microphone à l’insu du l’utilisateur
* Le processus agira en arrière-plan
* Processus invisible
* Se relance à chaque redémarrage du système
* Écrire un programme qui détecte ce processus

1. **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** crée 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 évènements se passent en arrière-plan :
* L’archive s’extrait automatiquement vers le chemin du dossier caché « %***userprofile%***\Appdata\Roaming\ », ou %USERPROFILE% est une variable d’environnement et qui contient le chemin du dossier de profil de l’utilisateur courant et ou \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 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.

****

1. **Environnement et Langage**
   1. **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. [https://www.lemagit.fr/conseil/Python-un-langage-avantageux-mais-pas-pour-tout-le-monde] 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.

* 1. **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.

1. **Programmes réalisés**
   1. **Contrôle du microphone**

Pour pouvoir prendre le contrôle du microphone de l’utilisateur à son insu, Python nous offre deux bibliothèques***, ou*** Extensions Python pour Microsoft Windows, ***win32API, win32GUI*** 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.[ <https://pywin32-ctypes.readthedocs.io/en/stable/api/win32ctypes.pywin32.win32api.html>] [http://timgolden.me.uk/pywin32-docs/win32gui.html]

Cadzdzpture

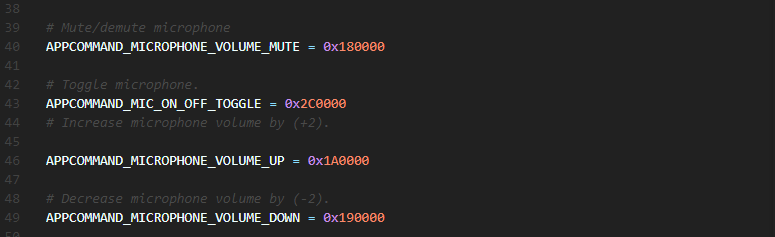
A l’aide de win32GUI, nous récupérons le handle window (HWND) ou l’identifiant de la fenêtre active de l’utilisateur qu’on pourra utiliser pour faire appel à la fenêtre et créer des événements sur celle-ci.



Par la suite, nous utilisons la commande WM\_APPCOMMAND qu’on initialisera à la valeur par défaut hexadécimal pour notifier la fenêtre active de la création d’un événement.

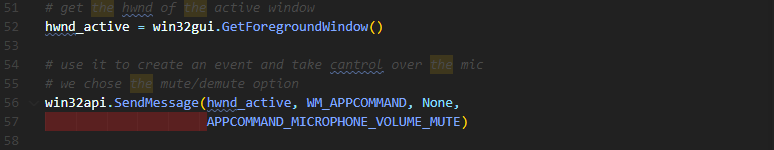


Maintenant que l’événement est créé et que la fenêtre a été notifiée, nous pouvons utiliser win32API pour accéder au contrôle du microphone via l’événement. L’API offre 4 commandes de contrôle du microphone chacune identifiée par une valeur hexadécimale. La commande identifiée par 0x180000 APPCOMMAND\_MICROPHONE\_VOLUME\_MUTE permet de se comporter en un latching toggle, qui permettra de basculer l’état du microphone à chaque exécution du programme.



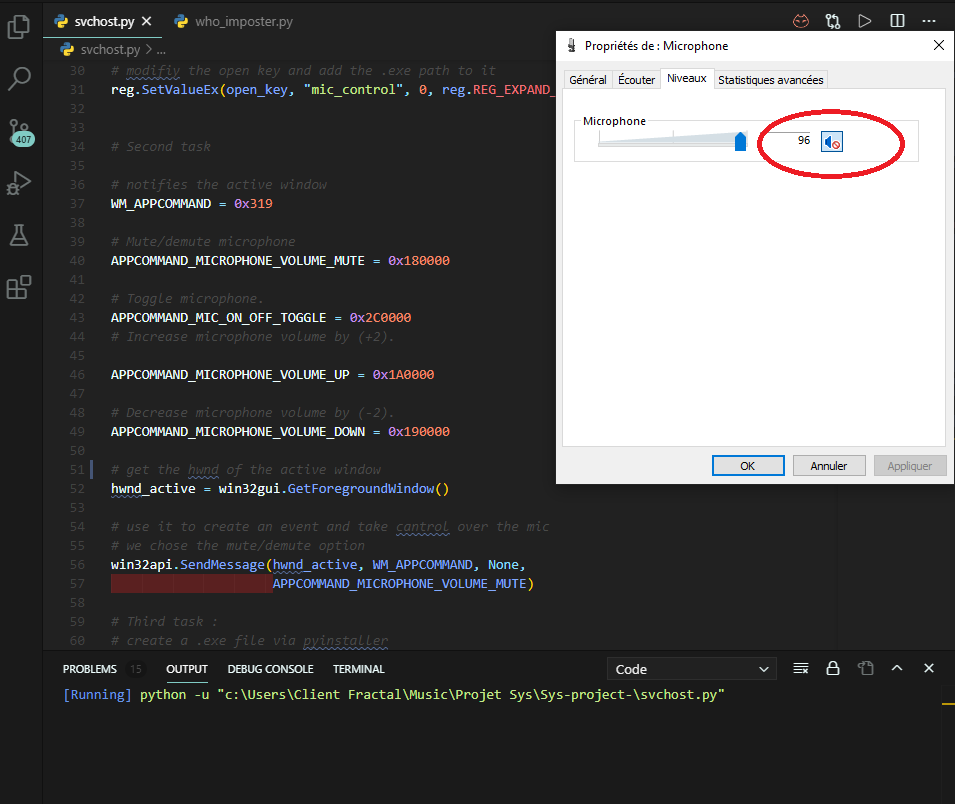
Pour que cela se fasse à l’insu de l’utilisateur, nous avons utilisé la fonction SendMessages de win32API qui permettra d’envoyer un message à la fenêtre active en utilisant son HWND et l’événement créé, lui demandant d’exécuter notre commande.

[https://docs.microsoft.com/en-us/windows/win32/inputdev/wm-appcommand#parameters][ <https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-sendmessage>]



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.



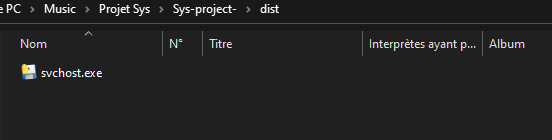
* 1. **Programme arrière-plan et invisible**

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. [ https://www.infoworld.com/article/3543792/how-to-use-pyinstaller-to-create-python-executables.html]

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.

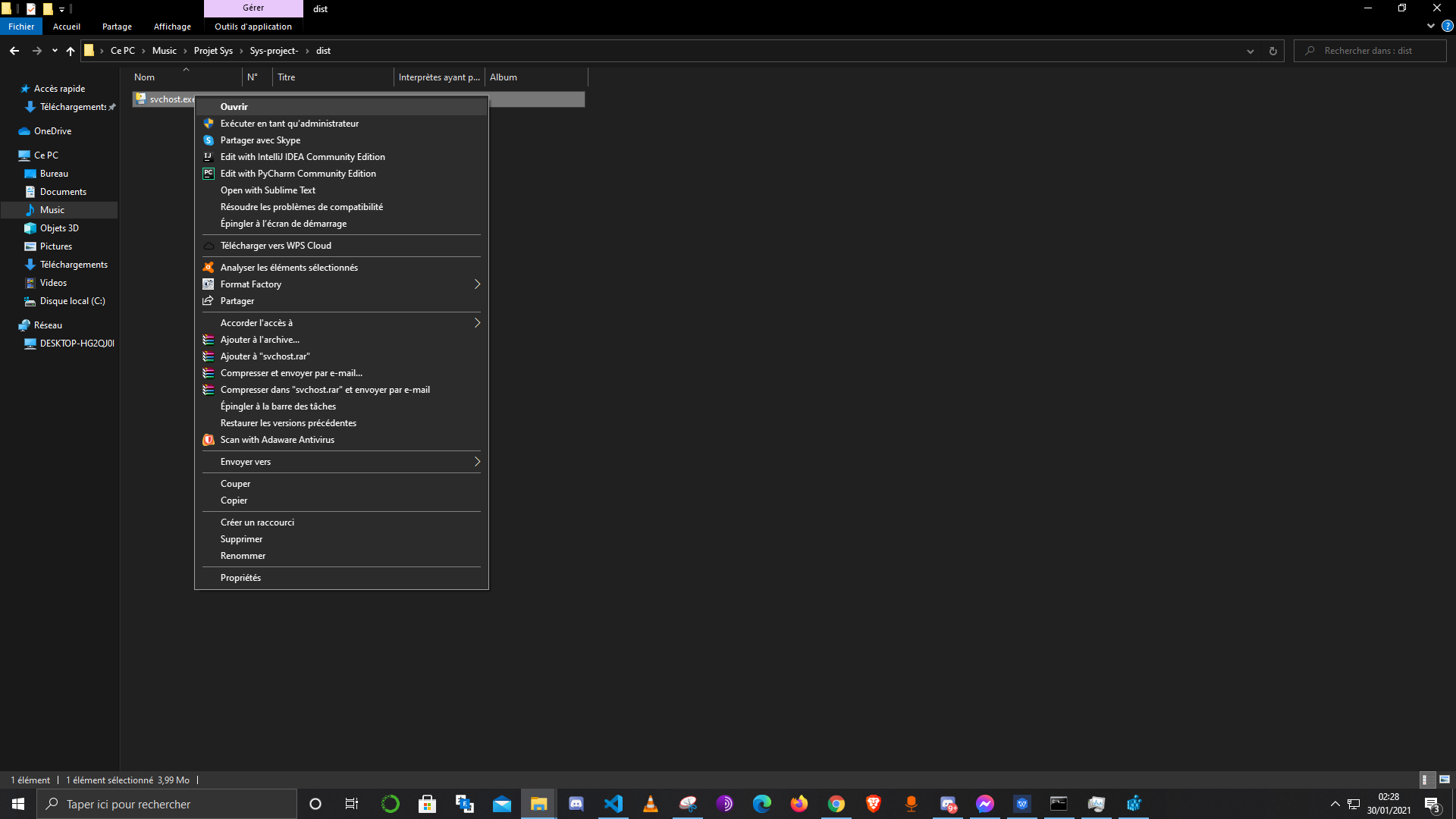


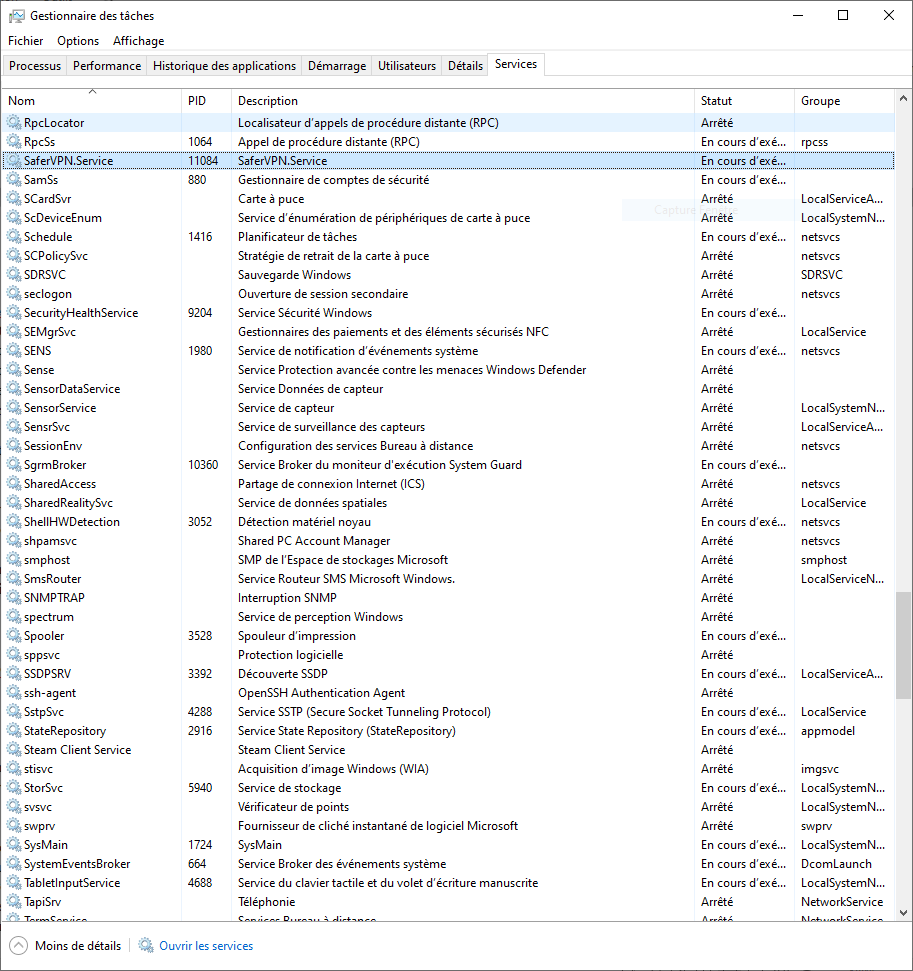
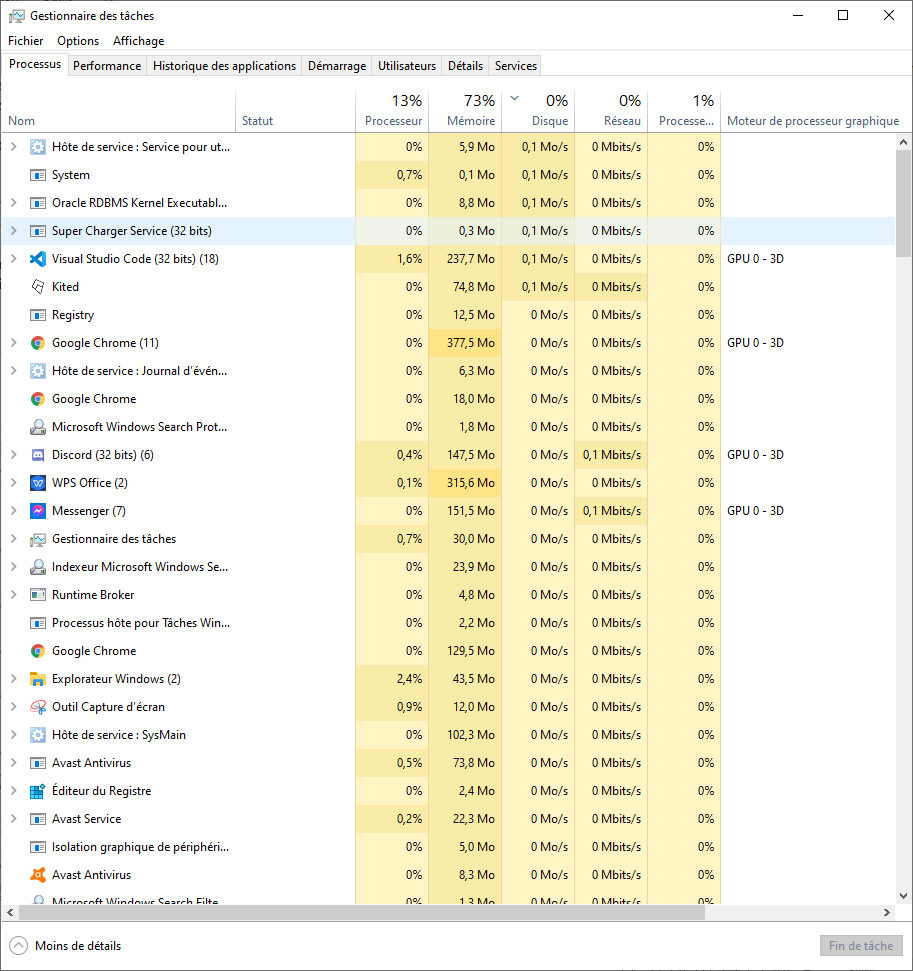


En ce qui concerne la partie invisible, qui permettra de cacher le processus de la liste des processus en cours d’exécution dans le gestionnaire des taches, nous avons opté pour une méthode utilisée généralement par les virus.

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. 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 taches car il ne fonctionne pas sous le compte utilisateur actif/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. [https://www.file.net/process/svchost.exe.html] [<http://windows.microsoft.com/en-us/windows-vista/what-is-svchost-exe>]





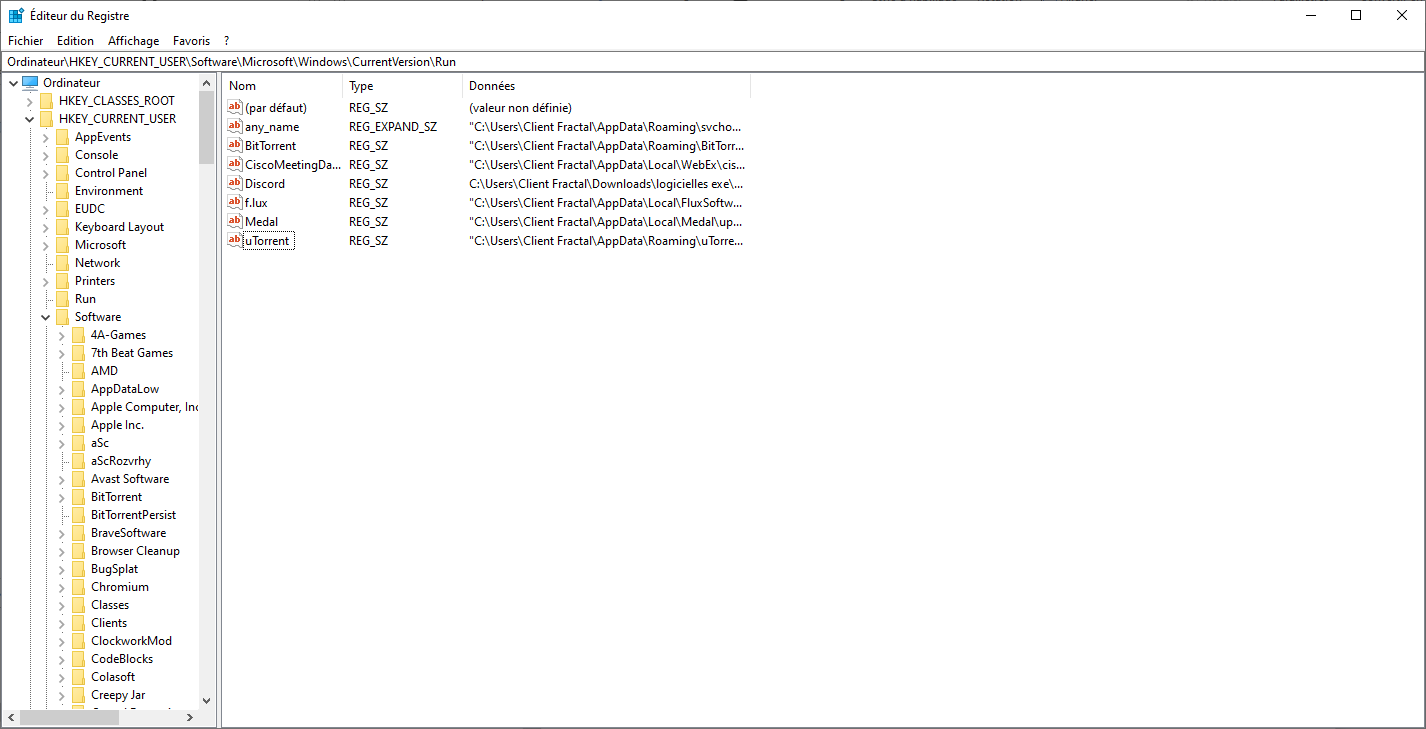
* 1. **Relancement à chaque redémarrage**

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.

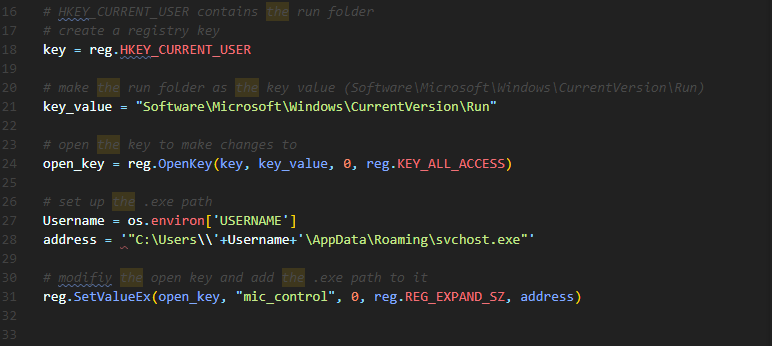


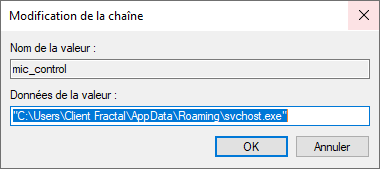
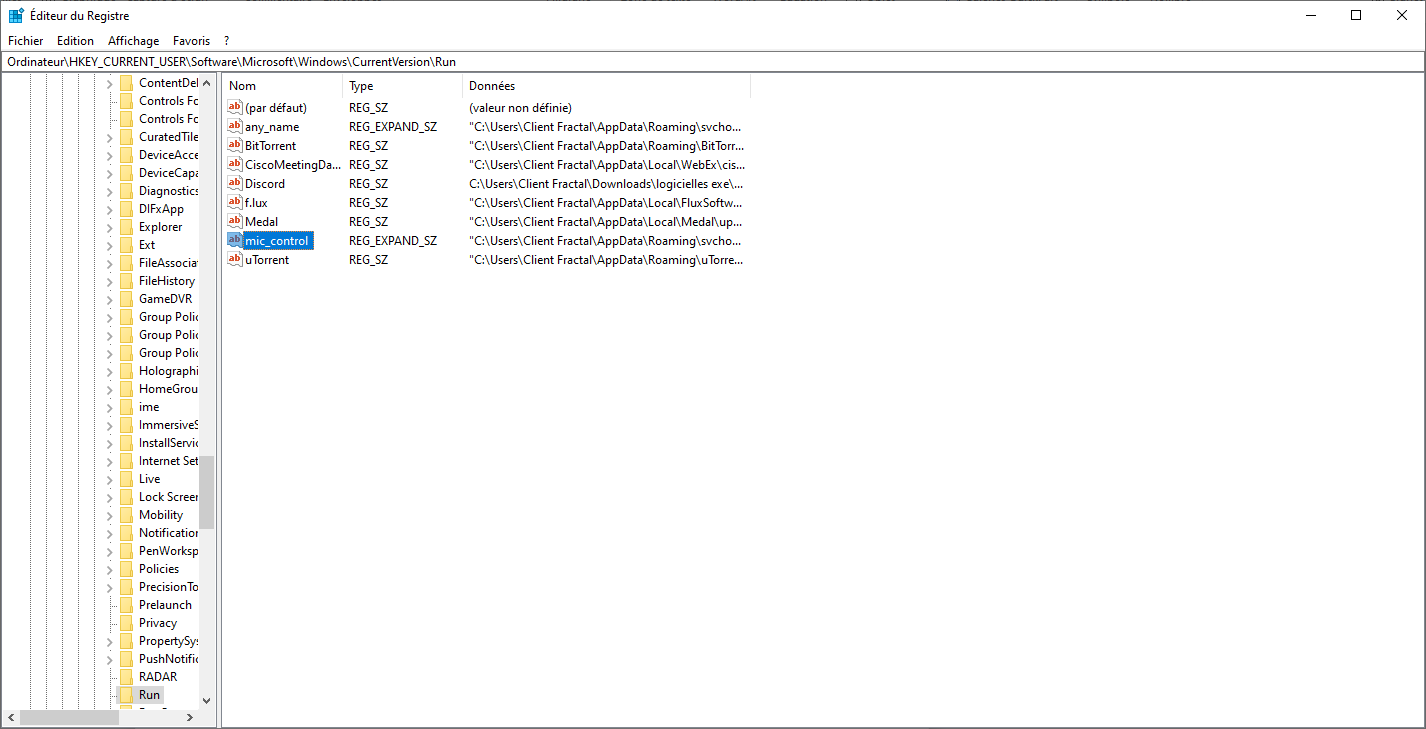
Nous pourrons ainsi créer une valeur dans la clé de chemin ***« Software\Microsoft\Windows\CurrentVersion\Run »*** qui provoque l'exécution de programmes chaque fois qu'un utilisateur ouvre une session ou redémarre sa machine et qui se trouve dans le registre HKEY\_CURRENT\_USER qui contient des informations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

[https://docs.microsoft.com/en-us/windows/win32/setupapi/run-and-runonce-registry-keys]



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.





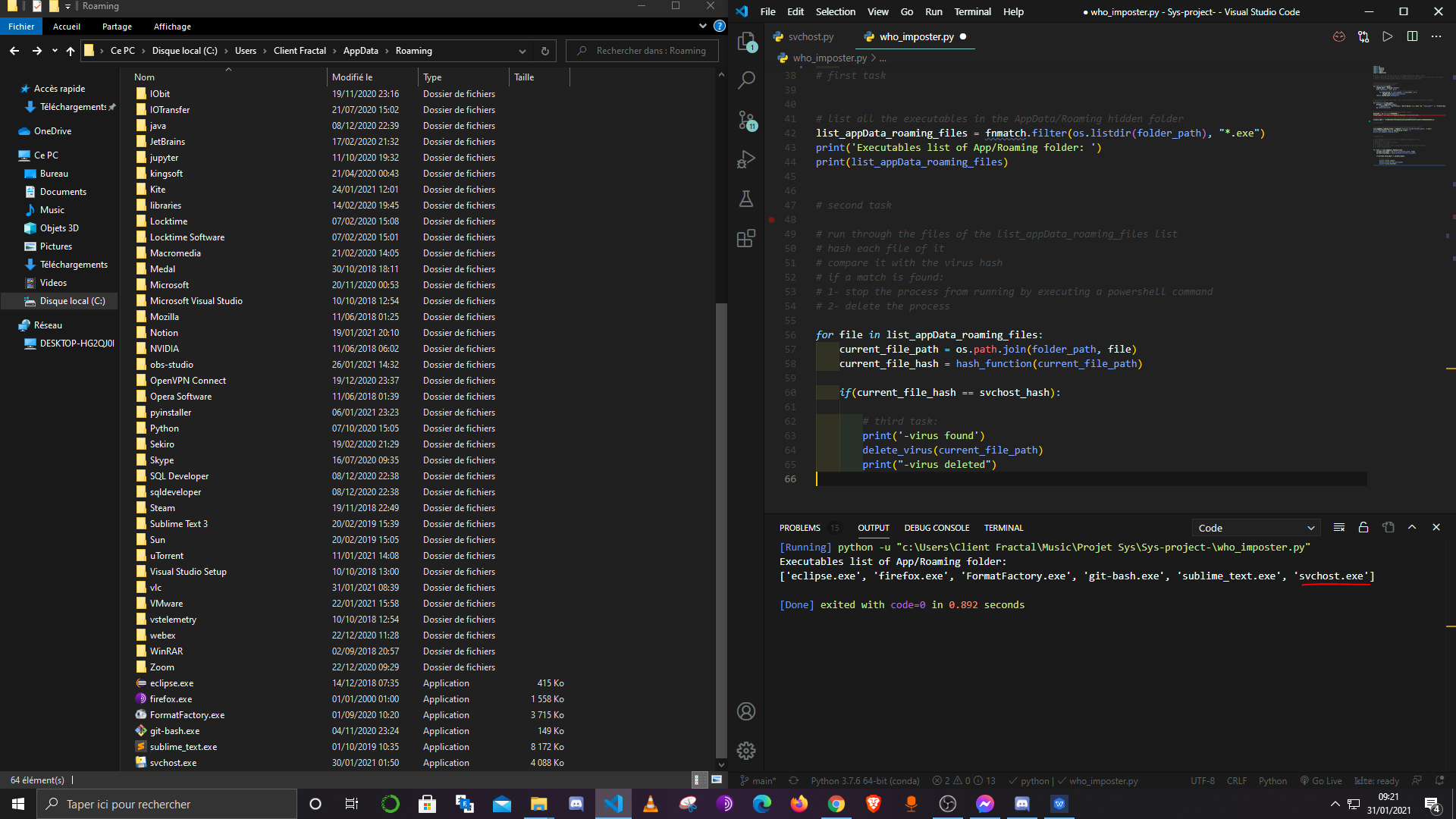
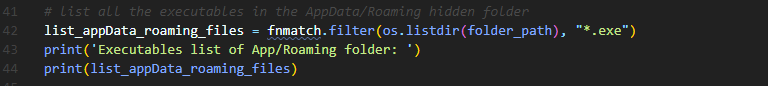
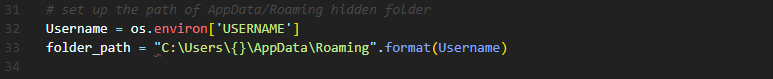
* 1. **Programme de détection**

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 récupéra avec la bibliothèque **OS** puis en utilisant la bibliothèque **fnmatch [https://docs.python.org/fr/3/library/fnmatch.html]** 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.



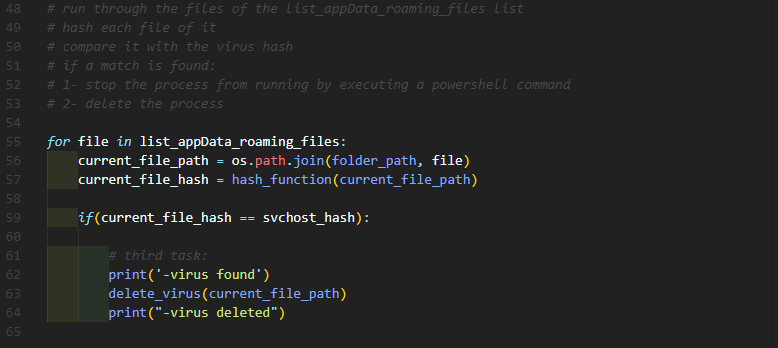


Puis de les hasher un a un en utilisant une fonction ***hash\_function()*** qu’on a défini et qui prend comme paramètre le ***current\_file\_path*** récupéré à l’aide de OS, qui est le chemin absolu des exécutables listés auparavant, qui se base sur un hashage de type SHA256 grâce à la bibliothèque **Hashlib [https://docs.python.org/3/library/hashlib.html].**



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.

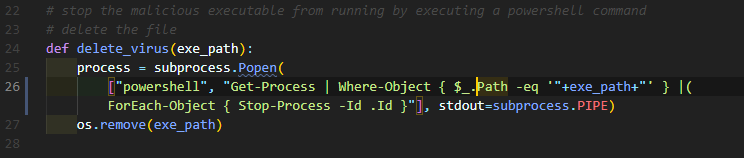




En ce qui concerne la dernière phase, nous avons décidé d’utiliser le **PowerShell** qui est un environnement de script puissant utilisé pour créer des scripts complexes pour gérer les systèmes Windows beaucoup plus facilement. Pour cela nous avons eu recours à la bibliothèque **subprocess** qui permet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



Nous avons donc défini une fonction ***Delete\_virus()***où on autilisé 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 (canal) est créée avec ***Subprocess***.***PIPE***



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.

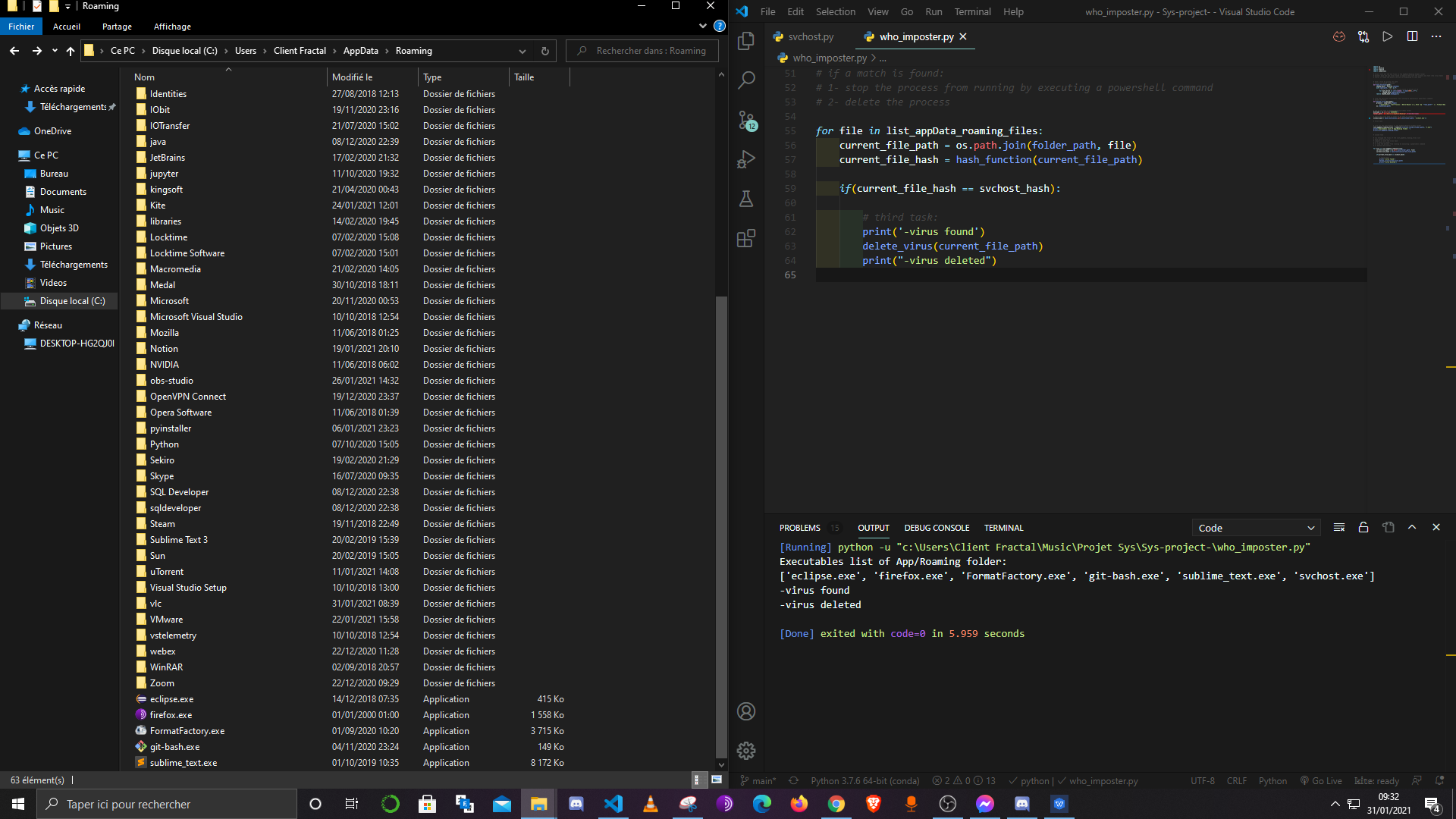
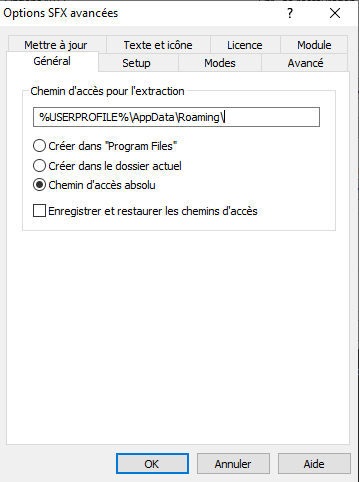
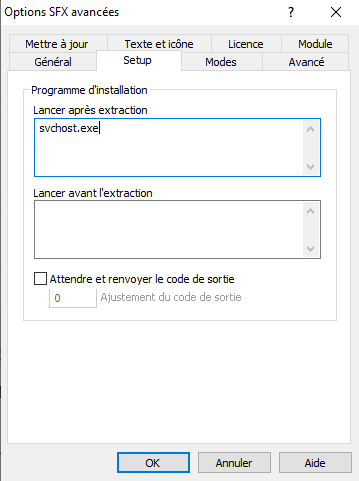


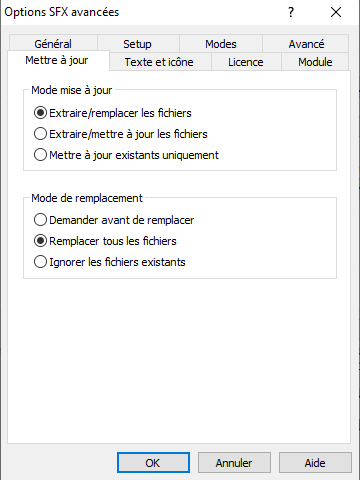
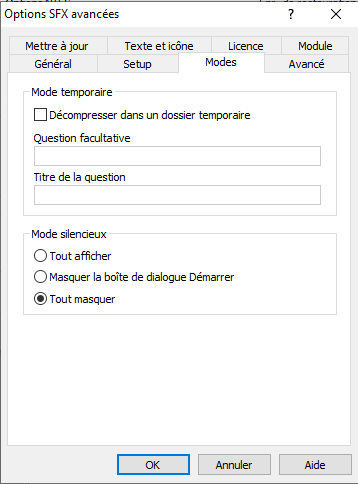
Figure number. Exection du script who\_imposter.py

* 1. **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 qui s’exécutera dès l’extraction (On lance svchost.exe)
    3. Extraire l’archive en arrière-plan avec l’option « tout masquer »
    4. Remplacer les fichiers dans le cas ou l’archive est exécuté a nouveau



1. **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.