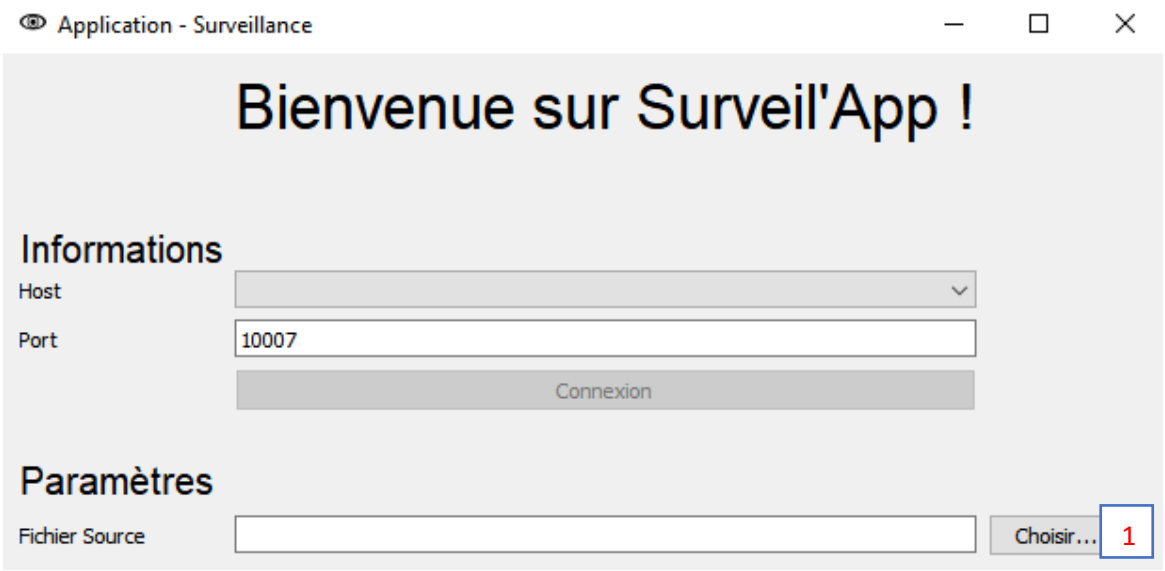
SAE3.02 – Surveil'App
Programmer documentation

I. Structure of the application

First of all, it seems interesting to show you the final structure of the application, here it is. You can see first a title, then an information section including the host and port and a Settings section, allowing to select the source file.

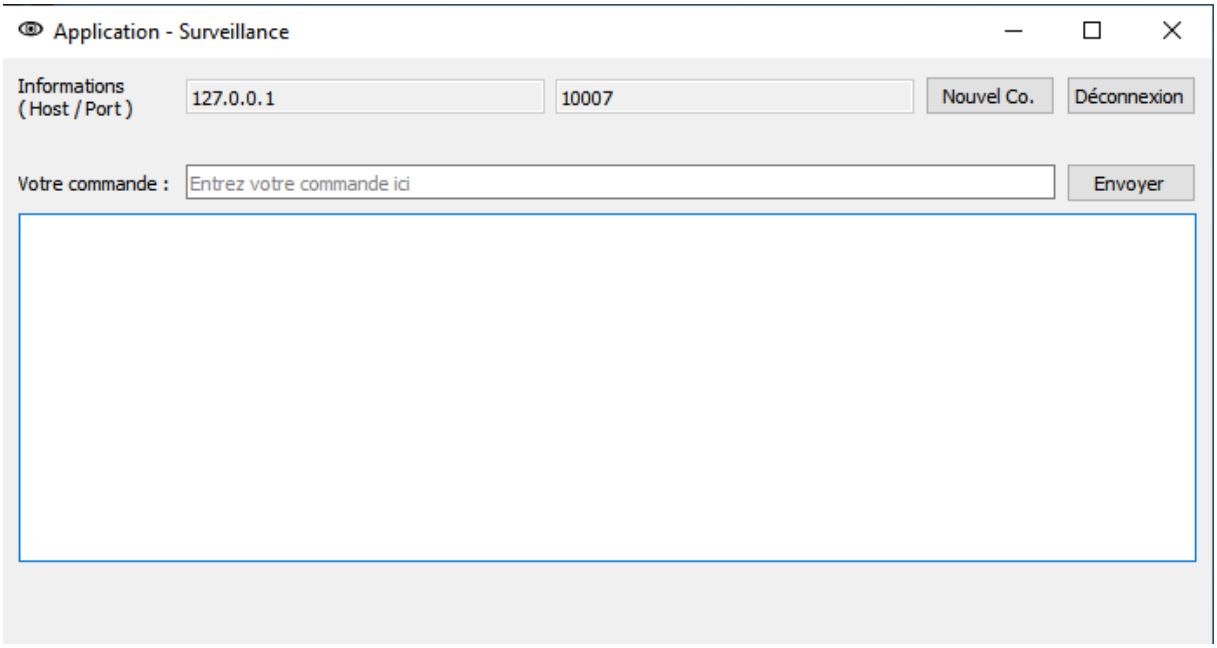
The most important thing on this home interface is the "Choisir" button (see 1), in fact it allows you to select, thanks to a file explorer, the .txt file you wish to link, in which there is the IP of your different machines.

Once selected, you will now see all the IPs in your file in the "Host" drop-down menu, and an "Ajout IP" feature has appeared in the settings section (see 2). Once connected, you can also connect using the "Connexion" button which is now declining.



Une fois le fichier sélectionné :

Once connected, this is what you see, the second part of the application's structure, the one that allows you to send remote commands. At the top, the connection information is displayed with two buttons next to it: New. Co. This allows you to connect to a second server and "Deconnexion" allows you to disconnect the client.



The second part is reserved for the management of orders, by being able to send them on the "Your order" part and then receiving the result sent in the area below. Now you understand the whole structure of Surveil'App.

II. Code Architecture

Below I will explain the architecture of my two codes, the client and the server. Let's start with the simplest one, the server.

First, it is necessary to import the necessary libraries, that's why this part is located at the top of the file.

```
import socket
import subprocess
import platform
import psutil
```

Next, we created a Server function. Inside this function, we initialize the value of the received message to empty, then we enter a loop, checking that the message is not kill and when it is, we logically leave the loop and close the server socket, since this is the specification:

```
msg = ""
while msg != "kill" :
                ...
                ...
```

```
print ("Connexion fermée")
server_socket.close()
print ("Serveur fermé")
```

Then we have two other loops, allowing us to manage the reset and disconnect commands, both of which are explained in the User documentation. Finally, the most interesting part is probably inside the last loop, since we find the interpretation of the commands and the return of their possible answer.

```
while msg != "kill" and msg != "reset":
                ...
                ...
        while msg != "kill" and msg != "reset" and msg != "disconnect":
```

Here is a more detailed explanation of the command management part, from the beginning, the command is stored in the variable msg. Then we split msg and store the result in the cmd variable in order to differentiate the OS specified in the command. As can be seen in the second screenshot, we then go through a series of loops allowing for some verification, such as writing the OS and compatibility, with the platform function. Finally, we use "subprocess.check_output" to obtain the result of the command entered as a parameter, the result of which is stored in the "reply" variable and sent using the "conn.send" function!

```
msg = conn.recv(1024).decode()
print ("Reçu du client : ", msg)
cmd = msg.split(':')
```

```
elif cmd[0].lower() == "powershell" and platform.platform()[:7] ==
"Windows":
    try:
        reply = subprocess.check_output(f"powershell.exe {cmd[1]}",
shell=True).decode('cp850').strip()
    except:
        reply = "[x] Commande Powershell non reconnue. Vérifiez la syntaxe"
        conn.send(reply.encode())
    else:
        conn.send(reply.encode())
```

We will now move on to the Client part. In the latter, we have made two classes to organise our code: one dedicated to message exchanges, called "Client" and a second dedicated to the graphical interface, called "MainWindow".
Firstly, in the Client class, we have 3 functions, two ensuring the connection of the client to the server and the last managing the message exchanges. The code used takes the basic functions of the sockets, with the presence of ".connect" as well as ".send" in particular.
As for the MainWindow class, we have the initialization of the interface in the __init__ function, as well as the creation, naming and placement of the different text zones, labels and buttons. Finally, a large part is also devoted to the hiding of certain labels/texts/other, since the home window and the one visible once connected are the same window, a choice made for questions of connection fluidity.
We then have the functions linked to the button, so we find "okcommand" allowing the sending and display of commands, "filename" allowing the link with a selected file, "newco" allowing the opening of a new window for multi-connection, "okconnexion" allows the

connection to the server and the transition between the two visuals, hence the large number of lines in this function, then we have the "add" function which manages the addition of host IPs in the connection menu and finally two functions which manage the disconnection, via the button and by sending the keyword "disconnect".

III. What has been done and not done

Finally, to conclude this documentation, I would like to point out that all the requested functionalities have been respected, one of the only points that could pose a problem is a connection to several servers, even if there should be no reason for malfunction. Also, the graphic aspect of the application is sober and basic, an improvement of this last one is envisaged during future updates!

That's it! You are now ready to improve and perform maintenance on Surveil'App, the first monitoring application for your remote servers! Thanks to you dear developer for installing Surveil'App and find our other applications on our website!