Labo SSI Groupe 4

Valentin LESPINE | Clément HEBRARD

PROJET

USBVacuum

2023/2024

Sommaire

Sommaire **1**
I) Introduction **2**
II) Présentation du projet - USBVacuum **3**
1) Objectifs et enjeux 3
III) Analyse du besoin **4**
1) Description du problème à résoudre 4
2) Identification des utilisateurs cibles 5
IV) Architecture de l'application **6**
1) Choix technologiques 6
2) Schéma logique de l’application 7
V) Fonctionnalités **8**
1) Détection des USB 8
2) Afficher l'arborescence du périphérique USB sélectionné 11
3) Sélection du type de scan 13
4) Le scan 14
5) Affichage des fichiers infectés 17
6) Utilisation du multi-threading 18
VI) Axe d'amélioration pour le futur **20**
VII) Conclusion **21**
VIII) ANNEXE **22**
1) Sources 22

I) Introduction

Dans un monde de plus en plus connecté et interconnecté, les clés USB sont devenues des outils incontournables pour le stockage et le transfert de données. Cependant, avec leur utilisation répandue, les risques associés aux logiciels malveillants transportés par ces dispositifs sont devenus une préoccupation majeure pour les utilisateurs et les organisations.

Les clés USB sont en fait un vecteur idéal pour la propagation des logiciels malveillants. Une fois qu'une clé USB infectée est insérée dans un système, les logiciels malveillants qu'elle contient peuvent se propager rapidement, compromettant la sécurité du réseau et des données sensibles.

Notre projet de station blanche informatique “USBVacuum” vise à fournir une solution efficace et fiable pour nettoyer les clés USB des logiciels malveillants.
Cette station automatisée sera conçue pour détecter et éliminer les menaces potentielles.

Le choix de repartir de zéro malgré l'existence de projets préexistants revêt une importance capitale dans notre démarche. En optant pour cette approche, nous avons délibérément décidé de nous plonger dans le processus complet de développement, depuis la conception initiale jusqu'à la réalisation concrète. Cette décision découle de notre volonté d'acquérir une compréhension approfondie de tous les aspects impliqués dans la création d'une solution logicielle, de la définition des besoins à la mise en œuvre technique.

En partant d'une feuille blanche, nous avons pu explorer de manière exhaustive les différentes options et technologies disponibles, sans être limités par les choix préalables effectués dans d'autres projets. Cette approche nous a offert une liberté créative et nous a permis de concevoir une solution entièrement adaptée à nos besoins spécifiques, sans compromis.

Enfin, repartir de zéro nous a offert l'opportunité d'apprendre de nos erreurs, ce qui a enrichi notre expérience et renforcé notre expertise dans ce domaine.

II) Présentation du projet - USBVacuum

1) Objectifs et enjeux

La station blanche USBVacuum, conçue pour sécuriser l'utilisation des clés USB, joue un rôle crucial dans la gestion des risques associés au transfert de données :

- L'objectif principal est de stopper la diffusion de logiciels malveillants via les clés USB, un vecteur commun d'attaques informatiques, ce qui représente un enjeu majeur pour la sécurité des infrastructures informatiques.

- Offrir une solution simple et automatisée qui peut être utilisée sans connaissances techniques approfondies, ce qui est crucial pour son adoption par un large éventail d'utilisateurs.

- Garantir que l'utilisation de la station aide les organisations à se conformer aux normes de protection des données, comme le GDPR en Europe, un enjeu clé pour éviter les sanctions légales et renforcer la réputation.

- La station est capable de s'adapter aux nouvelles menaces et de se mettre à jour facilement avec de nouvelles définitions de virus, répondant ainsi à l'enjeu de la sécurité dynamique dans un paysage de menaces en évolution.

- Élever la sensibilisation aux risques liés à l'utilisation des supports amovibles et éduquer les utilisateurs sur les bonnes pratiques de sécurité, ce qui est essentiel pour renforcer la culture de la sécurité au sein des organisations.

III) Analyse du besoin

1) Description du problème à résoudre

L'évolution rapide des technologies de l'information et de la communication a donné lieu à un paysage numérique complexe, où la sécurité des données et des systèmes est devenue une priorité absolue.
Dans cet environnement en constante évolution, les logiciels malveillants continuent de représenter une menace majeure, avec les clés USB servant souvent de vecteur de propagation privilégié.
Pour répondre à ce défi croissant, diverses approches et solutions ont été développées, mais des lacunes subsistent, justifiant ainsi notre projet de station blanche informatique pour le nettoyage des clés USB.

Des exemples solutions existantes :

- Les logiciels antivirus classiques sont conçus pour détecter et éliminer les menaces sur les systèmes informatiques.
  Cependant, leur efficacité peut être limitée lorsqu'il s'agit de détecter les logiciels malveillants sur les clés USB, en raison de la nature discrète de ces dispositifs et de la variété des menaces potentielles.

- Ces outils surveillent le comportement des programmes en temps réel pour détecter les activités suspectes. Bien qu'ils puissent être efficaces pour détecter les logiciels malveillants, leur utilisation sur les clés USB nécessite souvent une intégration complexe avec les systèmes existants.

- Les environnements de sandboxing permettent d'exécuter des programmes dans un environnement isolé, réduisant ainsi le risque pour le système hôte en cas d'activité malveillante. Cependant, leur déploiement sur les clés USB peut être difficile en raison de contraintes de ressources et de performances.

Malgré ces avancées, il reste clair qu'il existe un besoin persistant de solutions spécifiquement adaptées au nettoyage des clés USB des logiciels malveillants.
Notre projet se positionne comme une réponse à ce besoin en offrant une approche automatisée et spécialisée pour analyser, détecter et éliminer les menaces, sans compromettre l'intégrité des données légitimes.

2) Identification des utilisateurs cibles

L’identification des utilisateurs cibles pour une station blanche revêt une importance capitale. Une station blanche est un poste de travail dédié à l'analyse des médias amovibles, tels que les clés USB ou les disques durs externes, afin de déterminer leur fiabilité avant leur utilisation sur le réseau opérationnel.

Les utilisateurs cibles de la station blanche sont variés et comprennent notamment :

1. **Les Employés et Utilisateurs du Réseau** : Les utilisateurs finaux, qu'ils soient employés d'une entreprise ou membres d'une organisation. Ils bénéficient indirectement de la sécurité assurée par la station blanche, car elle contribue à prévenir les infections par des logiciels malveillants et à protéger leurs données et leur système informatique.
2. **Les Responsables de la Sécurité de l'Information** : Ces professionnels ont pour mission de garantir la confidentialité, l'intégrité et la disponibilité des données au sein de l'organisation. La station blanche est un outil précieux dans leur arsenal de sécurité, leur permettant de détecter et de neutraliser les menaces potentielles dès leur apparition.

IV) Architecture de l'application

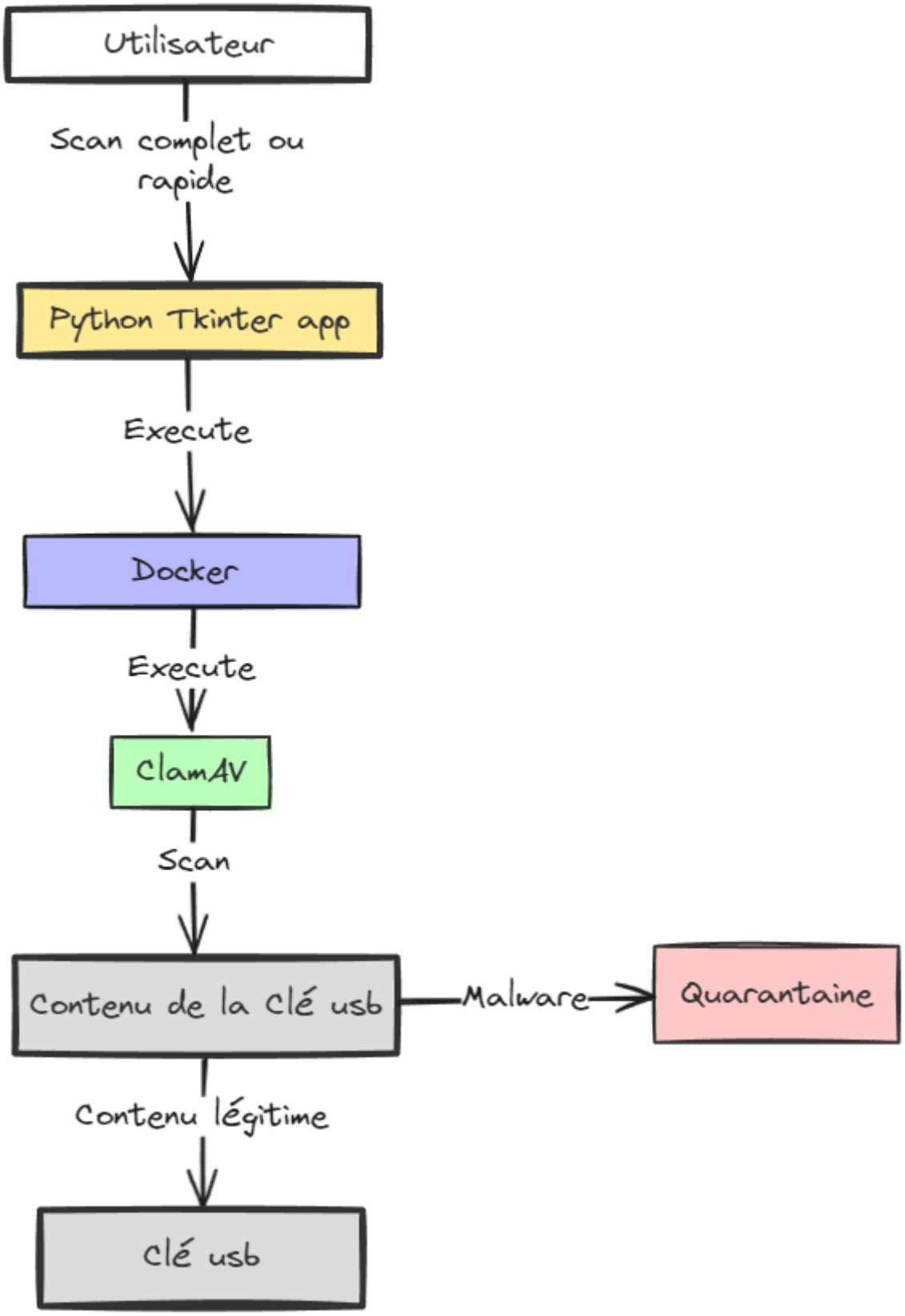
1) Choix technologiques

Notre approche technologique repose sur un ensemble de technologies puissantes soigneusement sélectionnées pour leur efficacité et leur adaptabilité. Parmi ces choix stratégiques, nous avons opté pour l'utilisation de Python, Figma, Docker, Tkinter et Tkinter Design chacune apportant une contribution significative à notre processus de développement et à l'expérience utilisateur finale.

Python	Avec sa syntaxe claire et sa grande polyvalence, est au cœur de notre écosystème de développement. Sa richesse en bibliothèques et son vaste écosystème de packages en font un choix naturel pour la construction d'applications robustes et évolutives.
Figma	Outil de prédilection pour la conception d'interfaces utilisateur. Sa facilité d'utilisation, ses fonctionnalités collaboratives et sa puissante capacité de prototypage nous permettent de créer des interfaces intuitives et esthétiques qui répondent aux besoins de nos utilisateurs.
Docker	Docker et Docker Compose jouent un rôle crucial. Docker Compose en particulier nous permet de simplifier et d'accélérer le processus de déploiement de notre application en orchestrant efficacement les conteneurs Docker nécessaires à son fonctionnement. Cette approche nous permet de maintenir un environnement de développement cohérent et reproductible, tout en facilitant les mises à jour et les déploiements sur différents environnements.
Tkinter	Bibliothèque graphique standard de Python, est notre choix pour la création d'interfaces graphiques utilisateur (GUI). Sa facilité d'utilisation en fait un outil idéal pour développer des interfaces utilisateur interactives et conviviales pour notre application.
Tkinter Design	Tkinter Design est un ensemble de principes, de techniques et de bonnes pratiques utilisés pour concevoir des interfaces utilisateur attrayantes et fonctionnelles à l'aide de la bibliothèque graphique Tkinter de Python. Conçu pour les développeurs d'applications Python, Tkinter Design propose une approche structurée pour créer des interfaces graphiques intuitives et esthétiques qui offrent une expérience utilisateur optimale. De plus, Tkinter Design peut être combiné avec des outils de conception comme Figma pour une collaboration transparente entre les concepteurs et les développeurs. En utilisant Figma pour créer des maquettes d'interface utilisateur, les développeurs peuvent facilement importer les conceptions dans Tkinter et les mettre en œuvre avec précision.

2) Schéma logique de l’application

Voici un schéma d’architecture du fonctionnement de notre application :

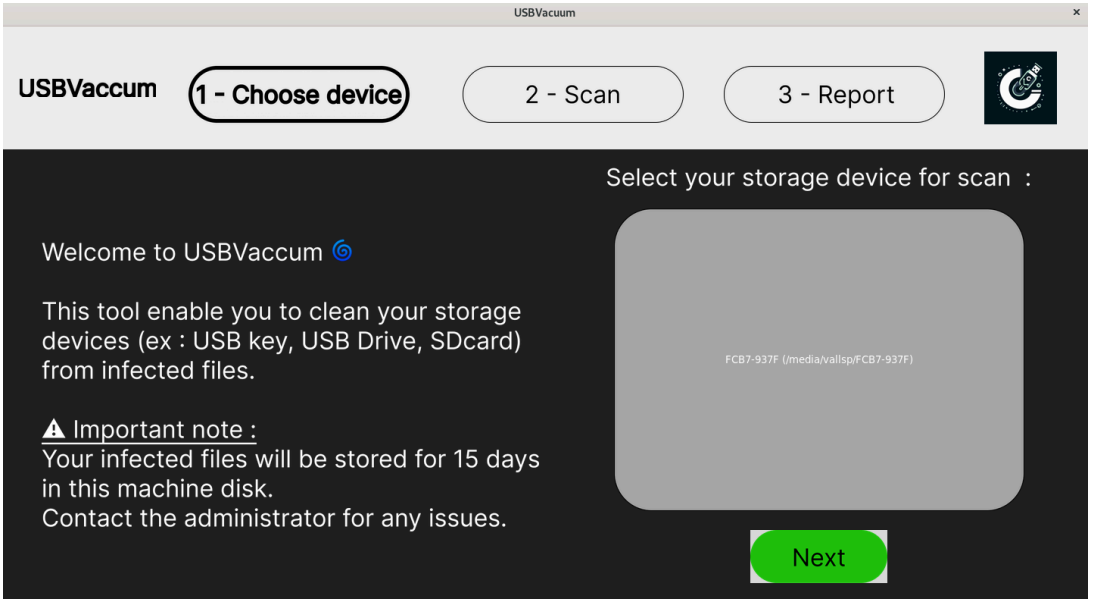


V) Fonctionnalités

1) Détection des USB

a) Explications

Cette fonctionnalité permet de détecter, afficher et sélectionner les périphériques USB connectés. Voici comment elle fonctionne :



Dans un premier temps la fonction “`display_usb_devices(canvas)`” efface les anciens textes des périphériques USB affichés sur un canvas et affiche les informations des périphériques USB actuellement connectés. Pour chaque périphérique USB détecté, elle crée un texte affichant le nom du volume et le chemin de montage sur le canvas. De plus, elle lie un événement de clic à chaque texte pour pouvoir interagir avec les périphériques USB sélectionnés.

```python
def display_usb_devices(canvas):
    canvas.delete("usb_text")  # Efface les anciens textes pour éviter de les empiler
    usb_devices = get_usb_devices()

    for i, (device_path, device_name, mount_path) in enumerate(usb_devices):
        x_position = 1082.0  # Choisissez la position x en fonction de votre mise en page
        y_position = 439.0 + i * 50  # Ajustez la position y en fonction de votre mise en page
        text = f"{device_name} ({mount_path})"
        # Créez le texte pour le périphérique USB avec un tag unique pour identifier l'événement de
clic
        canvas.create_text(x_position, y_position, text=text, fill="white", tags=f"usb_text")
        # Liez un événement de clic à ce texte
        canvas.tag_bind(f"usb_text", "<Button-1>", lambda event, mount_path=mount_path:
on_usb_device_click(mount_path, device_name))
```

Nous pouvons voir que la fonction appelle une autre pour récupérer les informations, qui ce nomme “`get_usb_devices()`” celle-ci récupère la liste des périphériques USB connectés à l'ordinateur.

```python
def get_usb_devices():
    devices = []

    try:
        partitions = psutil.disk_partitions(all=True)

        for partition in partitions:
            if is_usb_device(partition):
                volume_name = os.path.basename(partition.mountpoint)
                devices.append((partition.device, volume_name, partition.mountpoint))

    except Exception as e:
        logger.error(f"Error getting USB devices: {e}")


    return devices
```

Pour ce faire, elle utilise la fonction “ `psutil.disk_partitions(all=True)`” pour obtenir toutes les partitions, puis elle filtre celles qui sont des périphériques USB en utilisant la fonction “`is_usb_device(partition)`”. Les informations pertinentes (chemin du périphérique, nom du volume et chemin de montage) sont stockées dans une liste.

```python
def is_usb_device(partition):
    # Vérifie si la partition est probablement une clé USB plutôt qu'une partition système
    # Critère 1: Point de montage
    if '/media/' in partition.mountpoint:
        return True

    return False
```

Pour finir, La fonction “`on_usb_device_click(mount_path, device_name)`” est appelée lorsque l'utilisateur clique sur un périphérique USB affiché. Elle enregistre le chemin de montage et le nom du périphérique USB sélectionné dans des variables globales, puis affiche une boîte de dialogue d'information avec le nom du périphérique sélectionné.

```python
def on_usb_device_click(mount_path, device_name):
    global selected_usb_mount_path
    global selected_usb_device_name
    selected_usb_mount_path = mount_path
    selected_usb_device_name = device_name
    logger.info(f"USB device selected: {selected_usb_device_name}")
    logger.info(f"USB device mount path: {selected_usb_mount_path}")
    messagebox.showinfo("Info", f"USB device selected: {device_name}")
```

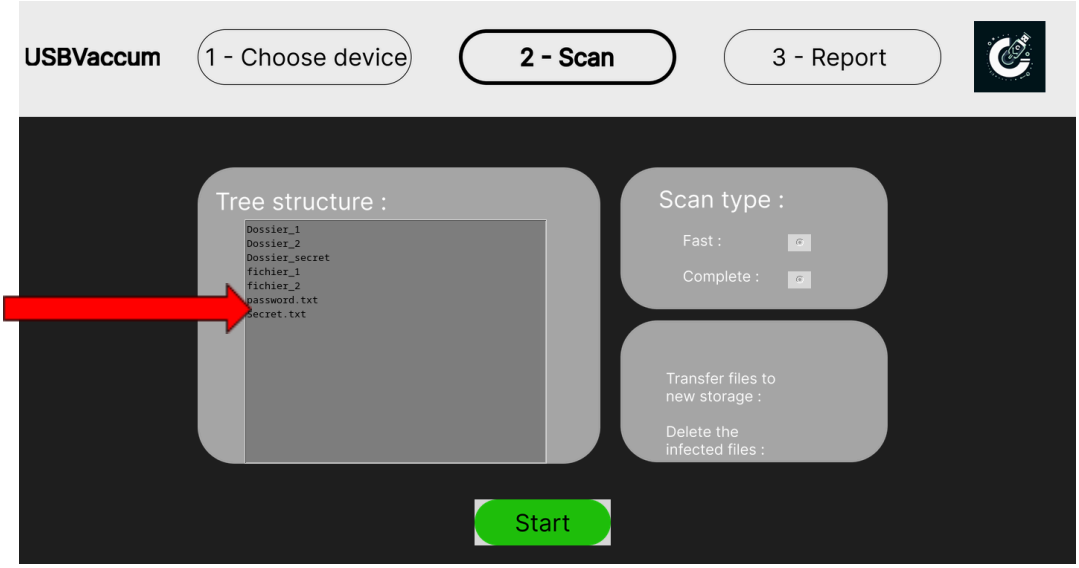
b) Problème rencontré

Nous avons rencontré quelques problèmes durant le développement de cette fonctionnalité, notamment par manque de connaissance des libraires qui existent et de comment elle fonctionne (psutil). Cependant, avec de la recherche dans les documentations nous avons pu résoudre nos problèmes.

2) Afficher l'arborescence du périphérique USB sélectionné

a) Explications

Nous avons implémenté une fonctionnalité pour afficher la structure des dossiers et des fichiers d'un répertoire donné sur une fenêtre graphique afin de vérifier si c'est le bon périphérique qui à été sélectionné.



Voici ce que fait chaque partie du code :

1. Nous avons utilisé deux modules Python : `os` pour travailler avec les fichiers et les répertoires du système d'exploitation, et `Text` de `tkinter` pour créer une zone de texte dans une interface graphique.
2. La fonction "`list_tree_structure()`" prend un le chemin de montage de la clé sélectionnée en entrée et affiche la structure de ce répertoire dans une zone de texte graphique.
3. Pour finir, la boucle `for` parcourt tous les dossiers et fichiers du répertoire spécifié et de ses sous-répertoires. Pour chaque itération de la boucle :
    - Les sous-dossiers sont ajoutés à la zone de texte.
    - Les noms de fichiers sont ajoutés à la zone de texte.

```
def list_tree_structure():
    text_widget = Text(width=50, height=17, bg='gray')  # Set the background color to gray
    text_widget.place(x=320, y=300)  # Place the Text widget at position (100, 100)

    for dirname,dirnames, filenames in os.walk(selected_usb_mount_path):
        # print path to all subdirectories first.
        for subdirname in dirnames:
            text_widget.insert(END, subdirname + '\n')  # Insert the subdirectory name into the Text
widget
        # print path to all filenames.
        for filename in filenames:
            text_widget.insert(END, filename + '\n')  # Insert the filename into the Text widget
```

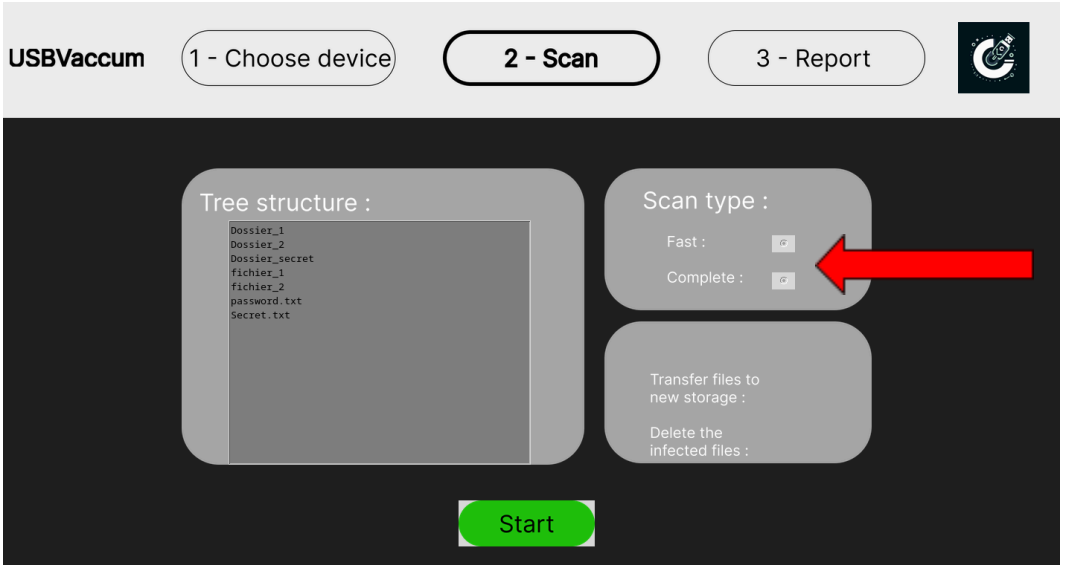
b) Problème rencontré

Nous avons rencontré quelques problèmes durant le développement de cette fonctionnalité, au début nous avons réussi seulement à afficher les dossiers avec une autre fonction, nous avons ensuite utilisé “`os.walk()`” qui nous a permis d’afficher les 2 à la fois.

3) Sélection du type de scan

a) Explications

Nous avons ajouté la possibilité de sélectionner le type de scan (entre un scan complet et un scan rapide) avec deux radiobuttons, voici comment cela fonctionne :



Tout d'abord, lorsque l'un des deux boutons radio est sélectionné, la fonction " update_scan_type() " est appelée pour définir le type de scan.

```
# Définition des Radiobuttons
        fast_scan_radio = Radiobutton(window, variable=scan_type_var, value=1, command=lambda:
update_scan_type("1"))
        fast_scan_radio.place(x=1050, y=320)

        complete_scan_radio = Radiobutton(window, variable=scan_type_var, value=2, command=lambda:
update_scan_type("2"))
        complete_scan_radio.place(x=1050, y=370)
```

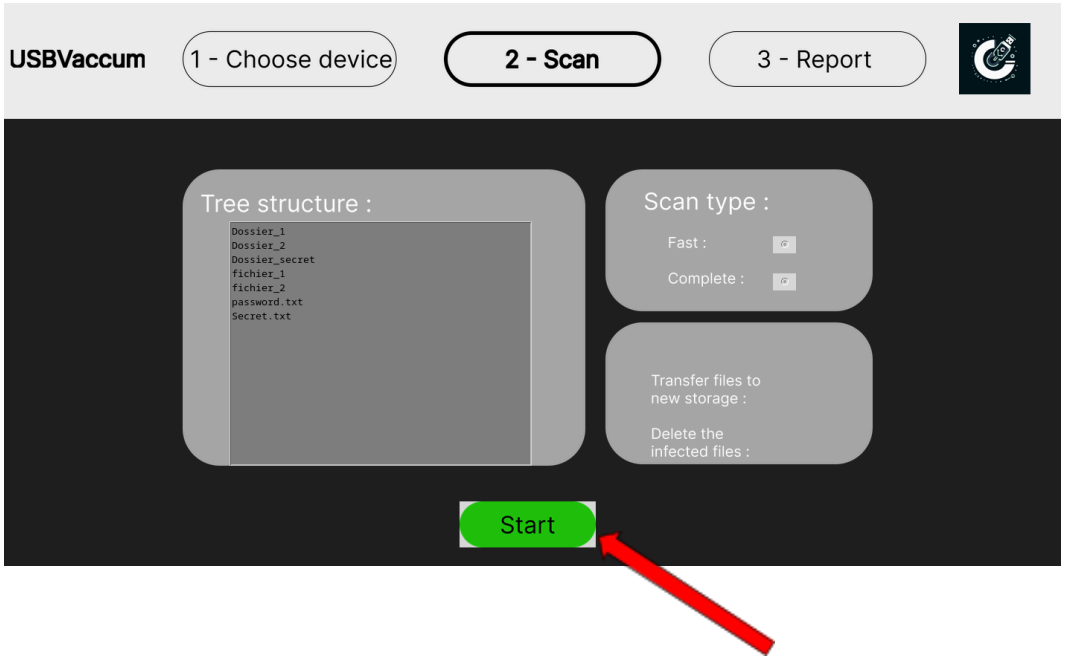
Voici la fonction " update_scan_type() " qui définit la variable globale "scan_type" soit en "fast" soit en "complète".

```
# Fonction pour mettre à jour la variable en fonction de l'état des radioButton
def update_scan_type(value):
    global scan_type
    if value == "1":
        scan_type = "fast"
    elif value == "2":
        scan_type = "complete"
    else:
        scan_type = None
```

4) Le scan

a) Explications

Une fois que le périphérique ainsi que le type de scan sont sélectionnés, la fonction “`do_scan()`” effectue une analyse antivirus sur le périphérique USB spécifié.



1. Lorsque le bouton Start est pressé, un répertoire de quarantaine est créé s'il n'existe pas déjà. Ce répertoire se trouve dans le répertoire `~/.clamav/quarantine`. S'il n'existe pas, il est créé.
2. Puis, selon le type d'analyse spécifié (“`scan_type`”), l'une des deux fonctions est appelée :” `complete_scan()`” pour une analyse complète ou “`fast_scan()`” pour une analyse rapide.
3. Un sous-répertoire spécifique pour ce périphérique est créé dans le répertoire de quarantaine.
4. Pour finir, si aucun type d'analyse n'est spécifié, un message d'erreur est enregistré dans les logs et une boîte de dialogue d'erreur est affichée.

```python
def do_scan(scan_type, usb_path):
    device_name = test_select_name()
    if not os.path.exists(f"{os.environ['HOME']}/.clamav/quarantine"):
        os.makedirs(f"{os.environ['HOME']}/.clamav/quarantine")
    if scan_type == 'complete':
        if not os.path.exists(f"{os.environ['HOME']}/.clamav/quarantine/{device_name}"):
            os.makedirs(f"{os.environ['HOME']}/.clamav/quarantine/{device_name}")
        logger.info(f"Scanning {usb_path} with {scan_type} scan type.")
        complete_scan(usb_path)
    elif scan_type == 'fast':
        if not os.path.exists(f"{os.environ['HOME']}/.clamav/quarantine/{device_name}"):
            os.makedirs(f"{os.environ['HOME']}/.clamav/quarantine/{device_name}")
        logger.info(f"Scanning {usb_path} with {scan_type} scan type.")
        fast_scan(usb_path)
    else:
        logger.error("No scan type selected.")
        messagebox.showerror("Error", "Please select a scan type.")
        return
```

Les fonctions `fast_scan() et complete_scan()` effectuent une analyse antivirus sur le périphérique USB sélectionné à l'aide du logiciel ClamAV qui est exécuté dans un conteneur Docker.

```python
def fast_scan(usb_path):
    device_name = test_select_name()
    try:
        os.system(f'docker run -it --rm --mount type=bind,source={usb_path},target=/scandir --mount
type=bind,source=$HOME/.clamav/quarantine/{device_name},target=/quarantinedir clamav/clamav:unstable
clamscan -r --move /quarantinedir --max-filesize=100M /scandir')
        logger.info(f"Fast scan finished.")
        scan_finished.set()
    except Exception as e:
        logger.error(f"Error in fast scan: {e}")
```

```python
def complete_scan(usb_path):
    device_name = test_select_name()
    try:
        os.system(f'docker run -it --rm --mount type=bind,source={usb_path},target=/scandir --mount
type=bind,source=$HOME/.clamav/quarantine/{device_name},target=/quarantinedir clamav/clamav:unstable
clamscan -r --move /quarantinedir /scandir')
        logger.info(f"Complete scan finished.")
    except Exception as e:
        logger.error(f"Error in complete scan: {e}")
    finally:
        # Définir l'Event pour signaler la fin de la numérisation
        scan_finished.set()
```

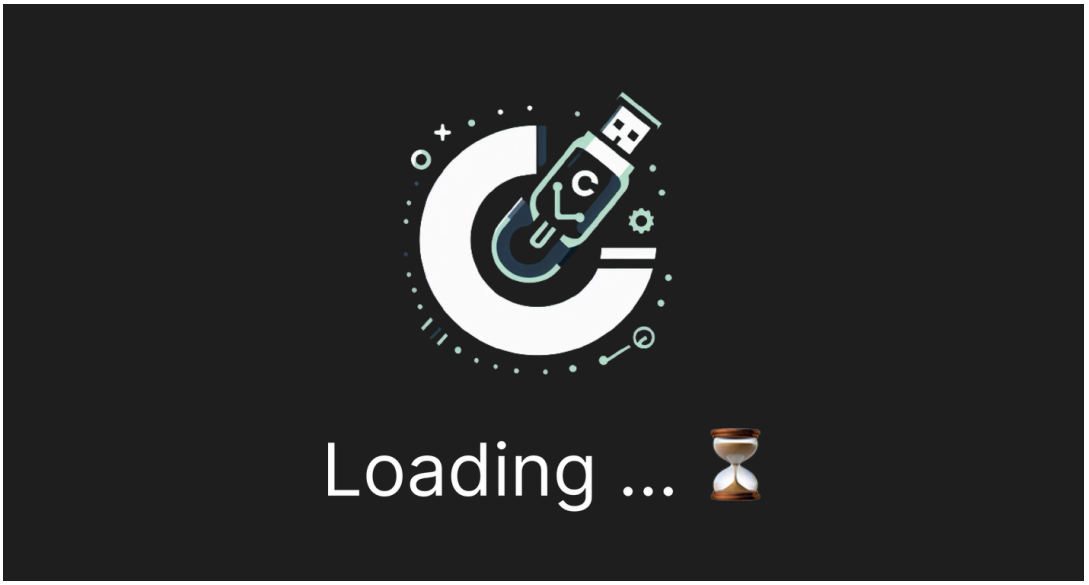
Voici ce que fait chaque partie des codes :

1. La commande `os.system()` est utilisée pour exécuter une commande shell. Cette commande crée un conteneur Docker avec l'image `clamav/clamav:unstable`, montant le répertoire du périphérique USB à analyser (`usb_path`) et le répertoire de quarantaine correspondant (`$HOME/.clamav/quarantine/{device_name}`) dans le conteneur, puis exécute `clamscan` sur le répertoire du périphérique USB (`/scandir`). Les fichiers infectés sont déplacés vers le répertoire de quarantaine.
2. Si l'analyse se termine avec succès, un message d'information est enregistré dans les logs indiquant que l'analyse rapide est terminée. De plus, une variable `scan_finished` est définie, ce qui peut indiquer à d'autres parties du code que l'analyse est terminée.
3. Si une erreur se produit pendant l'analyse, elle est capturée et enregistrée dans les logs, ce qui permet de diagnostiquer les problèmes éventuels.

Voici les différences entre les deux scans :

- **fast_scan()** : Utilise l'option `--max-filesize=100M`, ce qui signifie qu'il ne traitera que les fichiers de taille maximale de 100 Mo. Cela permet d'accélérer l'analyse en excluant les fichiers volumineux.
- **complete_scan()** : N'utilise pas l'option `--max-filesize`, ce qui signifie qu'il analysera tous les fichiers, quelle que soit leur taille. Cela garantit une analyse exhaustive de tous les fichiers présents sur le périphérique USB.

Pour finir, lors du chargement du scan l'utilisateur est redirigé vers une autre page l'indiquant que c'est en cours :



b) Problème rencontré:

Nous avons rencontré des difficultés lors de la mise en œuvre de l'analyse antivirus dans notre application. Au cours de nos tests, nous avons constaté que l'exécution de cette tâche bloquait l'interface utilisateur.

Après plusieurs recherches il est devenu évident que le blocage de l'interface utilisateur était dû au fait que l'analyse antivirus s'exécutent sur le même thread principal que celui utilisé pour gérer l'interface graphique. Cela signifie que toute opération, comme une analyse antivirus, empêchait l'interface utilisateur de répondre aux interactions de l'utilisateur.

Pour résoudre ce problème, nous avons adopté une approche à threads multiples. En exécutant l'analyse antivirus sur un thread séparé, nous permettrons à l'interface utilisateur de rester réactive et fonctionnelle pendant que la tâche d'analyse est en cours.

5) Affichage des fichiers infectés

a) Explications

La fonction `list_quarantine()` est conçue pour afficher le contenu du répertoire de quarantaine associé au périphérique USB sélectionné.

```
def list_quarantine():
    text_widget = Text(width=50, height=17, bg='gray')  # Set the background color to gray
    text_widget.place(x=525, y=300)  # Place the Text widget at position (100, 100)

    for dirname,dirnames, filenames in os.walk(f"{os.environ['HOME']}/.clamav/quarantine/
{selected_usb_device_name}"):
        # print path to all subdirectories first.
        for subdirname in dirnames:
            text_widget.insert(END, subdirname + '\n')  # Insert the subdirectory name into the Text
widget
        # print path to all filenames.
        for filename in filenames:
            text_widget.insert(END, filename + '\n')  # Insert the filename into the Text widget
```

Voici une explication du code :

1. Un widget de texte est créé avec une largeur de 50 caractères et une hauteur de 17 lignes. Sa couleur de fond est définie en gris. Ce widget de texte servira à afficher le contenu du répertoire de quarantaine.
2. Le widget de texte est positionné à l'emplacement `(525, 300)` dans la fenêtre où est le canevas graphique. Cela permet de contrôler où le texte sera affiché dans l'interface graphique.
3. Une boucle `for` va parcourir tous les dossiers et fichiers du répertoire de quarantaine associé au périphérique USB sélectionné. Pour chaque itération de la boucle :
    - Les sous-dossiers sont ajoutés au widget de texte.
    - Les noms de fichiers sont ajoutés au widget de texte.
4. La fonction `os.walk()` est utilisée pour parcourir récursivement tous les sous-répertoires et fichiers dans le répertoire de quarantaine. Cette fonction retourne les chemins des répertoires, les noms des sous-répertoires et les noms des fichiers.
5. Les noms de sous-répertoires et de fichiers sont insérés dans le widget de texte à l'aide de la méthode `insert()`.

6) Utilisation du multi-threading

a) Explications

Nous avons identifié un problème critique dans notre application lié à l'exécution de l'analyse antivirus, qui entraînait le blocage de l'interface utilisateur pendant que l'analyse était en cours. Pour remédier à cette situation, nous avons introduit une solution efficace qui permet à l'interface utilisateur de rester réactive même pendant les tâches intensives telles que l'analyse antivirus.

La solution consiste à utiliser des threads séparés pour exécuter l'analyse antivirus, tout en maintenant le thread principal dédié à la gestion de l'interface utilisateur. Ainsi, pendant que l'analyse est en cours dans un thread séparé, l'interface utilisateur reste libre de répondre aux interactions de l'utilisateur, assurant ainsi une expérience fluide et sans blocage.

Plus spécifiquement, la fonction `makeScan()` a été faite pour créer un nouveau thread qui exécute l'analyse antivirus en arrière-plan. Une fois que l'analyse est lancée dans ce thread séparé, la fonction `switch_to_interface()` est appelée pour passer à une autre interface graphique, permettant ainsi à l'utilisateur de continuer à utiliser l'application sans interruption.

```
def makeScan():
    if return_scan_type() == 'complete' or return_scan_type() == 'fast':
        global terminate_thread
        terminate_thread = False
        thread1 = threading.Thread(target=do_scan, args=(return_scan_type(),test_select_USB()))
        thread1.start()
        switch_to_interface("gui5")

    else:
        logger.error("No scan type selected.")
        messagebox.showerror("Error", "Please select a scan type.")
        return
```

Ainsi grâce au multi threads, nous avons pu gérer le changement d'interface et la détection de la fin du scan. Voici le code qui permet de faire cela :

Ce code introduit une nouvelle fonction `wait_switch_gui3()` et crée un nouveau thread `thread2` pour exécuter cette fonction :

1. Cette fonction attend que l'événement `scan_finished` se produise en utilisant la méthode `wait()`. Une fois que l'événement se produit, la fonction `switch_to_interface("gui3")` est appelée à l'aide de `window.after()` avec un délai de 0 millisecondes. Cela permet de passer à l'interface "gui3" dès que l'analyse est terminée.
2. Un nouveau thread `thread2` est créé en utilisant la classe `Thread` du module `threading`. Ce thread est configuré pour exécuter la fonction `wait_switch_gui3()`. Une fois que le thread est créé, il est démarré en appelant sa méthode `start()`.

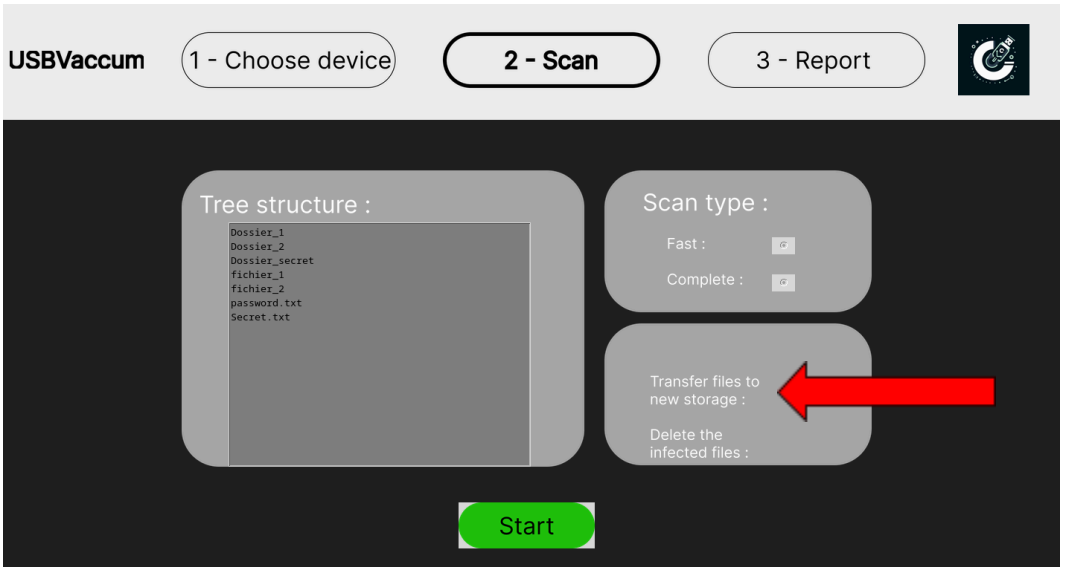
```
    def wait_switch_gui3():
            scan_finished.wait()
            window.after(0,switch_to_interface, "gui3")

    thread2 = threading.Thread(target=wait_switch_gui3)
    thread2.start()
```

VI) Axe d'amélioration pour le futur

Notre projet sera prochainement Open Source et nous allons mettre le dépôt en public sur GitHub. Notre objectif est de créer un environnement où chacun peut contribuer à l'amélioration continue du projet, enrichissant ainsi notre base de connaissances et renforçant sa pertinence et son utilité pour la communauté.

Il serait également possible de faire une intégration avec l'API de Virustotal, afin de ne plus être dépendant des résultats de qu'un seul antivirus (ClamAV).

Nous voulons aussi ajouter la possibilité de faire du transfert de fichier sur un autre périphérique contrôlé, pour l'instant la fonctionnalité est présente sur l'interface mais n'est pas fonctionnelle :



A noter également que la fonctionnalité qui permet de supprimer les fichiers infectés à été rajoutée, mais nous avons décidé de ne pas rajouter cette fonctionnalité, car ce n'est pas à l'utilisateur de gérer les fichiers infectés, mais plutôt l'administrateur de la station.

VII) Conclusion

En conclusion, ce projet a représenté une étape significative dans notre parcours, nous permettant d'explorer et d'appliquer diverses technologies et méthodologies pour atteindre nos objectifs. L'utilisation de Python, Docker et Tkinter Designer combiné à des outils de conception tels que Figma nous a permis de créer des interfaces utilisateur élégantes et fonctionnelles, offrant ainsi une expérience utilisateur optimale.

Au cours de ce projet, nous avons également eu l'occasion de développer nos compétences techniques et notre compréhension des bonnes pratiques de développement logiciel. La collaboration au sein de l'équipe a été essentielle pour surmonter les défis techniques. De plus, les conseils des dir labs nous ont aidés sur certains points ou nous avons bloqué.

En outre, l'adoption de technologies telles que Docker Compose pour le scan antivirus nous a permis de renforcer la sécurité de l’application.

Enfin, ce projet nous a permis de fournir une solution logicielle innovante. Nous sommes convaincus que les enseignements tirés de ce projet nous serviront de base solide pour nos futurs projets et contribueront à notre croissance en tant qu'équipe et en tant qu'étudiant.

VIII) ANNEXE

1) Sources

GitHub - ParthJadhav/Tkinter-Designer: An easy and fast way to create a Python GUI 🐍
Figma : l'outil de design à interface collaborative
Docker
Python.org
Visual Studio Code
ClamAV