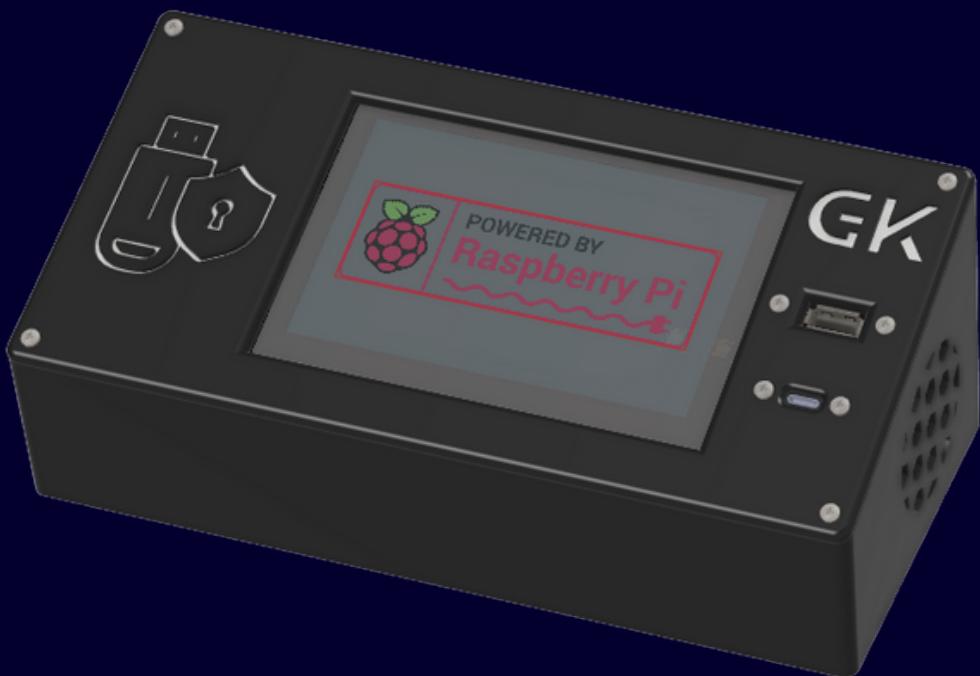


Rapport Projet GateKeopr

du 10 mai 2023 au 22 juin 2023



Théo Lefèvre, Valentin Lallier,
Camille Lefebvre, Mathis Lasson



<https://github.com/Skorthis/GateKeopr>

TABLE DES MATIÈRES



TABLE DES MATIÈRES

I_ Introduction.....	3
II_ Conception du schéma 3D.....	7
III_ Détection et premiers outils permettant de filtrer les périphériques (USBGuard).....	13
IV_ Outil de scan de la clé et identifier des potentiels virus, mise en quarantaine et suppression(ClamAV).....	21
V_ Keylogger.....	29
VI_ Interface graphique (PyQt).....	37
VII_ Différents tests.....	49
IIX_ Sécurité du code.....	53
IX_ Conclusion.....	57
X_ Remerciements.....	59
XI_ Bibliographie.....	60

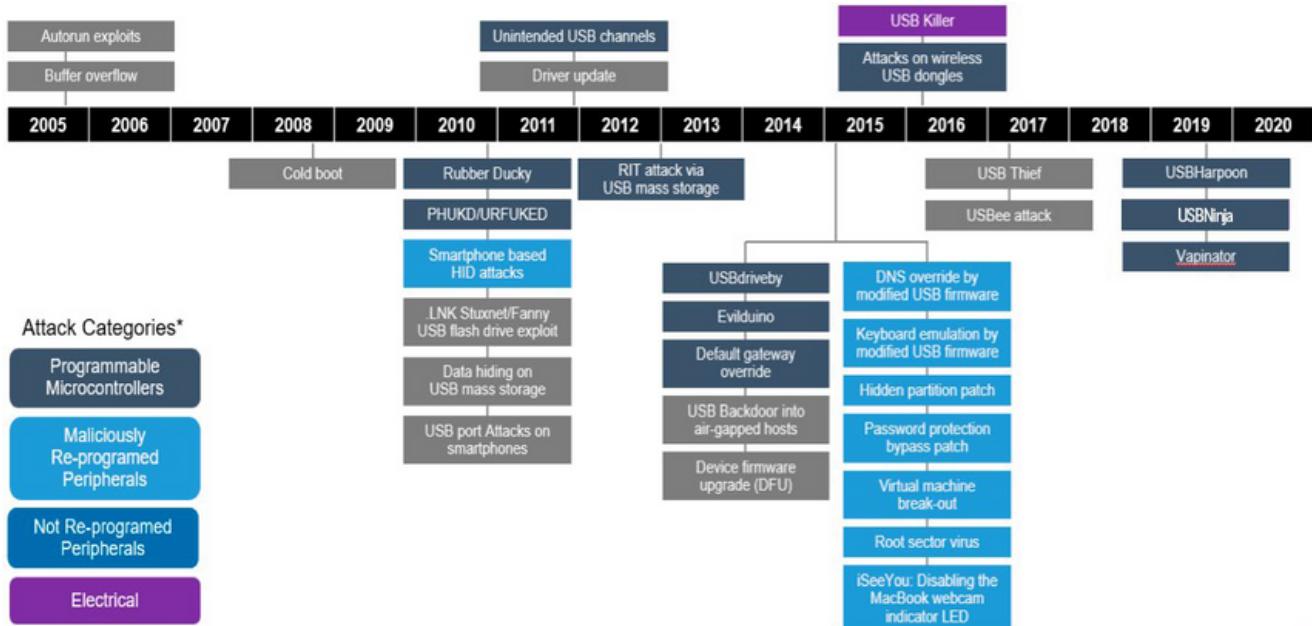


INTRODUCTION

Dans un monde où l'interconnectivité se généralise de manière exponentielle, la sécurité informatique s'impose désormais comme une préoccupation majeure et incontournable. En effet, les menaces se multiplient de façon alarmante, parmi lesquelles l'utilisation de périphériques USB corrompus s'avère être l'une des méthodes d'attaque les plus couramment employées. On sait que 52% des virus connus sont conçus pour être déployer sous USB [1]. Les cybercriminels, experts dans leur domaine, déploient des techniques d'infiltration d'une sophistication remarquable, allant de l'insertion d'un simple périphérique de stockage USB dissimulant un logiciel malveillant, jusqu'à l'usurpation de l'identité de certains périphériques ou l'usage de câbles infectés. On peut notamment illustré la facilité et le dangerosité de ces attaques en mentionnant une étude menée en 2016 par des chercheurs de Google et de deux universités américaines [2] dans laquelle 297 clés USB ont été déposées sur un campus universitaire. Les résultats ont montré qu'au moins 45% de ces clés ont été branchées sur un ordinateur, 68 % des étudiants l'ont fait par altruisme et 18 % avouent l'avoir fait par curiosité [3]. Ce qu'il faut retenir de cela est que seulement 13 % des étudiants sondés admettent avoir pris des précautions, ce qui montre à quel point les attaques par clé USB sont méconnues et de la facilité déconcertante pour un attaquant ou une organisation à réussir une cyberattaque. De plus, les vecteurs d'attaques sont multiples et le plus souvent invisibles car il existe de fausses clés USB comme la fameuse Rubber Ducky ou la Bash Bunny mais aussi des claviers, des souris, des câbles et même des cigarettes électroniques comme le Vapinator [4] capables de faire presque tout ce qu'il est possible de s'imaginer en exfiltrant par exemple des données, en installant des logiciels espions, en exécutant des scripts ou encore en créant des backdoors [5].



INTRODUCTION



Pourtant, face à cette menace grandissante, le volet sécuritaire des dispositifs USB a été étonnamment négligé, et peu nombreux sont les dispositifs capables d'offrir une protection véritablement efficace à un prix abordable. De plus, environ 70 % des ordinateurs utilisés dans le monde fonctionnent avec le système d'exploitation Windows [6] sur lequel il n'existe presque pas de solutions de sécurité pour certains types d'attaque comme les injections de commandes. C'est précisément dans ce contexte préoccupant que nous avons entrepris la conception d'une station de détection de menaces USB novatrice. L'objectif de ce dispositif est de combler cette lacune inquiétante en fournissant une solution globale pouvant être utilisée tant par les entreprises que par les particuliers soucieux de préserver l'intégrité de leurs systèmes.

Cette station sera équipée d'une panoplie d'outils sophistiqués permettant de filtrer, d'inspecter minutieusement et d'effectuer des analyses approfondies à la recherche de virus et autres programmes malveillants sur les périphériques de stockage de masse. L'utilisateur disposera ainsi de la capacité de supprimer ou de mettre en quarantaine les fichiers potentiellement dangereux, afin d'éradiquer toute menace avérée. Quant aux claviers, ils feront l'objet d'une analyse des frappes, détectant ainsi toute tentative d'envoi de commandes automatiques. Grâce à cette solution de pointe, nous aspirons à conférer à nos utilisateurs une tranquillité d'esprit et une confiance dans l'utilisation des périphériques USB, leur offrant ainsi une protection proactive face aux attaques qui prolifèrent à grande vitesse.

INTRODUCTION



Ce projet a été une expérience enrichissante pour notre équipe, nous apportant une meilleure compréhension de l'organisation et de la gestion de projet, grâce par exemple à l'utilisation d'un diagramme de Gantt pour planifier et suivre les différentes étapes du projet, ce qui nous a permis de visualiser et de gérer efficacement les délais et les tâches à accomplir.



La répartition des tâches entre les membres du groupe a été soigneusement effectuée, en tenant compte des compétences et des intérêts de chacun. Cela nous a permis de tirer parti des forces individuelles et de travailler de manière synergique pour atteindre nos objectifs. Nous avons maintenu un horaire de travail régulier de 9h30 à 16h tous les jours, en étant présents sur place pour favoriser la collaboration et la communication. Cela nous a permis de rester engagés et de progresser de manière constante tout au long du projet.

Pour faciliter le partage de fichiers et la collaboration, nous avons utilisé un Google Drive où nous avons regroupé l'entièreté des fichiers pertinents, des ébauches d'affiches au rapport éthique en passant par les codes sources et les schémas 3D. Cela nous a permis d'accéder facilement aux informations nécessaires et de travailler de manière synchronisée, que ce soit sur place ou depuis chez nous.

Dossiers	Nom
Vidéo de Présen...	⋮
Soutenance	⋮
Rapport Final	⋮
Planning	⋮
Logs	⋮
logos	⋮
Conception 3D	⋮
Code	⋮

INTRODUCTION



Afin de garder une trace de chaque action effectuée, nous avons rempli des fichiers logs quotidiens sur ce même Drive. Cela nous a permis de documenter les points évoqués durant les réunions, les étapes clés, les problèmes rencontrés et les solutions mises en œuvre. Ces logs ont été précieux pour analyser notre progression, garder trace de chaque action et anticiper d'éventuels retours en arrière.

Mercredi 17 mai :

Création d'un fichier bash qui permet de scanner la clé à l'insertion
→ Problème sur le scan, le démon scan qu'un fichier et non récursivement
Ligne a mettre dans le fichier 80-local.rules dans le répertoire /etc/udev/rules.d :
`SUBSYSTEM=="block", ACTION=="add", RUN+="/usr/local/bin/trigger.sh"`
`ACTION=="add", KERNEL == "sd[b-z]", RUN+="/usr/local/bin/trigger.sh"`
Fichier a créer dans le répertoire /usr/local/bin de nom trigger.sh (fichier qui sera exécuté lorsque la clé sera branché :
`/usr/bin/date >> /tmp/udev.log (a changer par la commande qui scan)`

→ Le script est exécuté deux fois, dans le fichier log on obtient deux fois la date.
Début de la modélisation 3D du boîtier.

Jeudi 18 mai : (jour férié)

Avancement chacun de notre côté sur diverses tâches.
→ Commencement de l'étude du temps d'analyse en fonction de la taille et du nombre de fichiers.

→ Suite de la modélisation 3D (boîtier principal terminé)

Plusieurs contretemps dans chaque équipe.

Reunion du 31 mai 2023 :

Var log sys
Bloquer tout commençant par alt ou Windows, CTRL

- Pour USBGuard, il faut trouver un moyen d'attendre que le démon se soit mis à jour en utilisant un script bash pour attendre ou en utilisant la commande linux lsusb -v qui donne beaucoup d'informations. Par la suite, il faudrait étudier la gestion des périphériques par USBGuard pour voir s'il est capable de détecter lorsqu'un périphérique USB change de classe et dans ce cas s'assurer qu'il applique les règles correctement. On peut aussi regarder dans les logs générés par le système d'exploitation qui se trouve dans var/log/syslog.

Choses à effectuer pour la semaine prochaine

Trouver une solution pour l'affichage des informations des périphériques USB insérés

L'organisation de réunions hebdomadaires avec notre tuteur nous a aussi permis de faire le point sur l'avancement du projet, partager nos résultats et discuter des éventuelles difficultés rencontrées. Ce à quoi nous avons rajouté des points réguliers au sein du groupe qui ont favorisé la communication, nous permettant de résoudre plus efficacement les problèmes grâce aux avis et connaissances de chacun.

Dans l'ensemble, ce projet nous a apporté une expérience précieuse en matière de planification, d'organisation et de gestion de projet. Nous avons acquis une meilleure compréhension des processus collaboratifs et de la nécessité de maintenir une communication efficace au sein de l'équipe. Ces compétences nous seront bénéfiques dans nos projets futurs et contribueront à notre développement professionnel.



CONCEPTION SCHÉMA 3D

Pour ce projet, nous avions besoin d'un objet physique car l'idée derrière GateKeepr était d'avoir une station sur laquelle les utilisateurs pourraient brancher leurs périphériques USB afin de vérifier s'ils étaient sécurisés. Nous avons donc décidé de concevoir un boîtier dans lequel se trouverait un ordinateur avec des ports USB ainsi qu'un écran pour permettre d'afficher des informations et offrir une interaction entre le système et l'utilisateur.

La première étape de création de notre station a été de choisir les composants adéquats afin qu'ils satisfassent à la fois les exigences techniques mais aussi financières car pour mener à bien ce projet, nous disposons d'un budget de 150 €. Le cœur de notre station est un nano-ordinateur, le Raspberry Pi 4 Model B dans sa version 4 Go de mémoire RAM avec son alimentation officielle. Nous avons fait ce choix car comme son nom l'indique, le Raspberry Pi est un des ordinateurs les plus petits au monde et il a de nombreux avantages comme son prix qui est inférieur à 100 €, ses performances, sa capacité à prendre en charge diverses distributions Linux ainsi que sa grande communauté [7]. Nous avions envisagé d'autres possibilités comme la Banana Pi, la Rock Pi ou encore la carte Asus Tinker Board mais aucune d'elles n'arrivent à rivaliser avec la Raspberry Pi. La version 4 Go de RAM que nous avons réussi à emprunter durant cette période de pénurie s'est imposée d'elle-même car pour répondre aux exigences techniques de l'antivirus que nous voulions utiliser, il nous fallait au moins 3 Go de RAM [8]. Ensuite, nous avions besoin de trouver un moyen de refroidir notre système et surtout notre carte Raspberry Pi car cette dernière dépasse généralement les 50 °C en fonctionnement normal et peut atteindre les 90 °C lorsque le CPU est complètement utilisé, or passé le seuil de 85 °C, la carte se met en sécurité et ralentie [9].



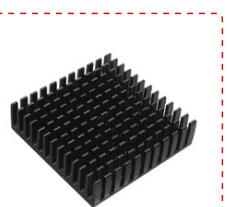
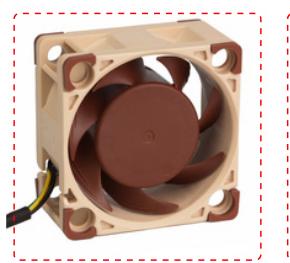
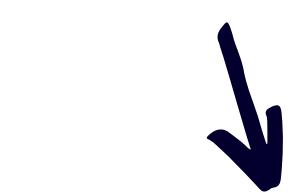
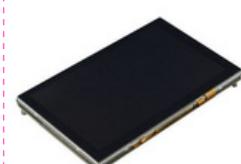
CONCEPTION SCHÉMA 3D

Pour conserver de bonnes performances, nous avons donc opté pour deux types de refroidissement, l'un passif avec un dissipateur thermique en aluminium qui vient se coller par-dessus le processeur, la RAM et le module WIFI et l'autre actif avec un ventilateur Noctua de 40 mm 5V qui offre un flux d'air constant dans le boîtier. Le choix avait été fait de mettre les ports USB pour insérer les périphériques à analyser en façade du boîtier ce qui nous a obligé à déporter les ports. Afin de résoudre ce problème, nous avons utilisé deux raccords USB mâle vers USB femelle, l'un en USB type A 3.0 et l'autre en USB type C. L'utilisation de la norme USB 3.0 est plus pertinente dans notre cas, car la Raspberry Pi dispose de deux ports USB 3.0, c'est la nouvelle norme USB et les vitesses de transfert sont dix fois plus rapides qu'avec la norme 2.0, ce qui améliore la rapidité globale de notre système. Enfin, le dernier élément que l'on a voulu intégrer à notre boîtier a été un écran tactile de 5" compatible avec notre carte et qui a la particularité de faire passer le flux vidéo via un port DSI (Display Serial Interface) ce qui nous permet de relier l'écran à la Raspberry Pi avec une nappe DSI et on a donc supprimé la contrainte d'avoir un écran inséré sur la carte. La fin de cette première étape a été marquée par la commande des composants auprès des différents fournisseurs comme Conrad, DigiKey ou encore GoTronic.

Raspberry Pi 4 B 4 Go



Affichage



Refroidissement



Connecteurs USB



La deuxième étape a été de modéliser un boîtier en 3D pour qu'il puisse ensuite être imprimé. Nous avons fait le choix d'utiliser le logiciel de CAO professionnel Autodesk Fusion 360 qui offre un large panel de fonctionnalités et qui a l'avantage d'être gratuit pour les étudiants. Pour expliquer simplement la façon dont nous avons procédé, nous avons commencé par faire un dessin en 2D de chacune des pièces aussi appelé esquisse. Ensuite, nous avons transformé ce dessin en un volume grâce aux différents outils comme l'extrusion, le perçage ou encore la révolution. À cette étape nous avions une pièce en 3D brute sur laquelle il fallait ajouter de nouvelles esquisses pour personnaliser les zones des pièces qui sont apparues suite à la mise en volume comme par exemple les parois du boîtier. L'addition de toutes les esquisses et les mises en volume nous a permis d'obtenir les pièces finales.



Concernant la partie principale du boîtier :

- nous avons placé quatre plots au fond pour y fixer la Raspberry Pi ainsi que des trous dans chaque coin pour pouvoir visser le capot,
- une grille d'aération en forme d'essaim d'abeilles sur la paroi droite pour permettre un flux d'air et limiter la vision sur l'intérieur du boîtier,
- une grille de ventilateur sur la paroi gauche pour que le ventilateur puisse extraire l'air du boîtier et pour éviter que quelqu'un ne mette ses doigts dans les pâles,
- et enfin sur la face arrière nous avons un trou aligné avec le connecteur type C de la carte pour pouvoir brancher l'alimentation et un cache de secours en face d'un des ports micro HDMI qu'il est possible de retirer pour avoir un accès à l'interface graphique du Raspberry Pi si l'écran venait à tomber en panne.

La partie sur laquelle se fixe le capot a volontairement été inclinée de 20° pour améliorer l'ergonomie du système et ainsi permettre aux utilisateurs de mieux voir l'écran.



Le capot quant à lui dispose de :

- deux emplacements à droite pour pouvoir y visser les raccords USB,
- un trou central pour accueillir l'écran
- et nous avons mis notre logo à gauche ainsi qu'en haut à droite.

La face inférieure du capot possède des supports qui permettent de fixer l'écran car malheureusement l'écran que nous avions ne pouvait se visser que par l'arrière. Pour finaliser le tout, nous avons ajouté des pieds à placer sous notre boîtier afin de le surélever et d'améliorer l'efficacité de la grille d'aération située en dessous. Pour chacune des pièces conçues, nous avons essayé le plus possible de tenir compte des contraintes de taille, d'impression, de sécurité, de température, d'ergonomie et de réparabilité. La dernière phase de la conception consiste à assembler toutes les pièces sur le logiciel pour s'assurer de leur conformité puis de générer un fichier au format STL utilisable pour l'impression 3D. Le logiciel nous a également permis de générer l'image au centre de notre poster ainsi que les animations présentes dans notre vidéo.

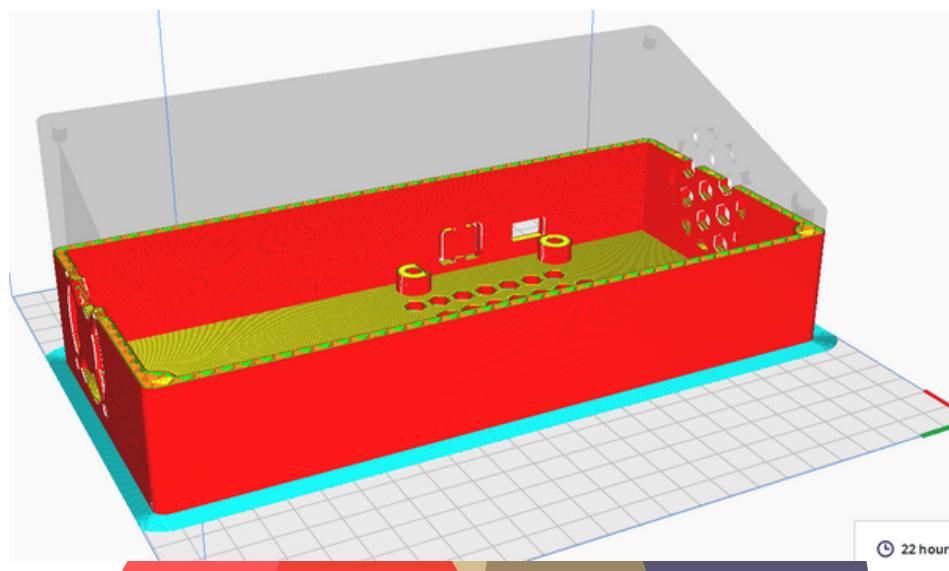
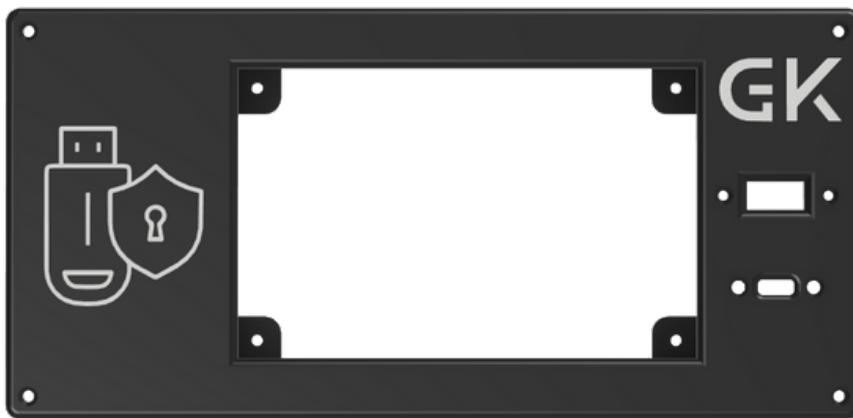
La troisième étape a été d'imprimer en filament PLA toutes les pièces en 3D. Pour cela, on a utilisé un slicer, c'est-à-dire un logiciel qui permet de convertir des fichiers 3D en une suite d'instructions compréhensibles par une imprimante et offrant un paramétrage complet de l'impression et de l'imprimante comme la qualité de l'impression ou le remplissage des pièces. En fonction des imprimantes, on a utilisé le logiciel open-source Cura ou le logiciel propriétaire FlashPrint. Nous avons eu beaucoup de mal à imprimer le capot et le boîtier car ce sont des pièces de grande taille qui demandent plusieurs heures voire des dizaines d'heures d'impression et qui par conséquent sont sujettes à un phénomène de décollement appelé "warping" qui entraîne une courbure des pièces. C'est pourquoi nous avons imprimé avec diverses imprimantes dont une professionnelle, la FlashForge Creator 3 Pro avec laquelle nous avons obtenu des pièces de bonne qualité.





CONCEPTION SCHÉMA 3D

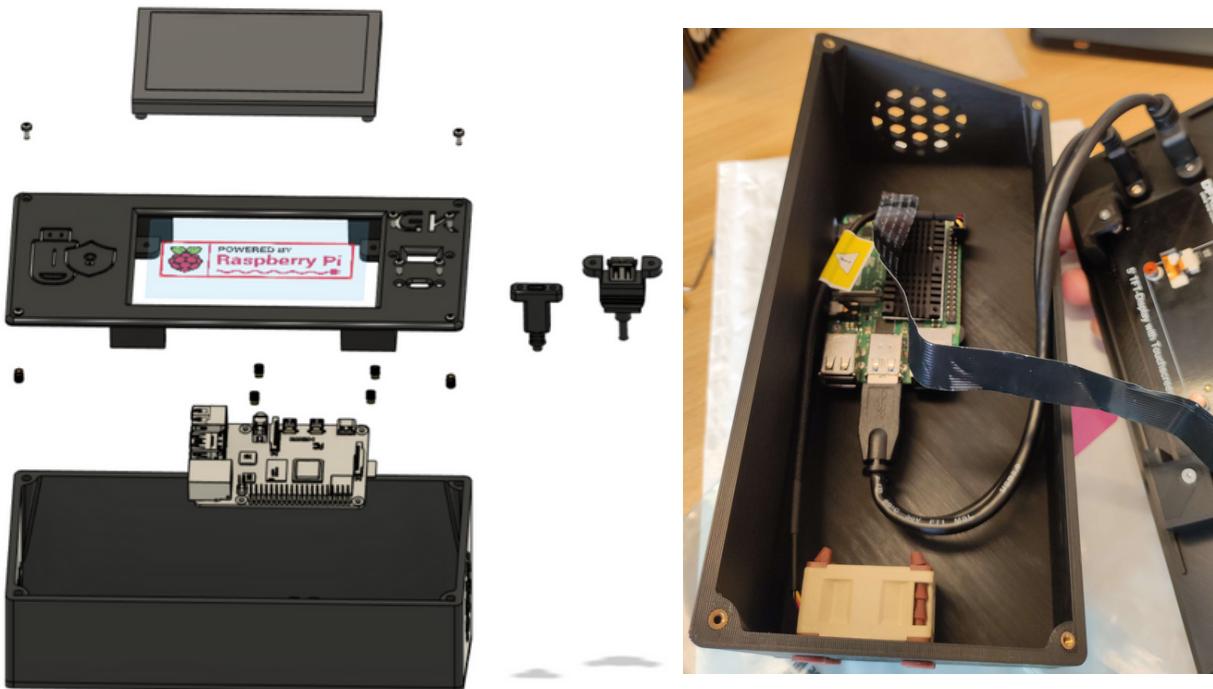
Une fois les pièces imprimées, il est souvent nécessaire de faire un post-traitement en enlevant avec une pince les supports imprimés pour maintenir les parties des pièces qui étaient dans le vide, en supprimant avec un cutter les fils et défauts d'impression et en coloriant au marqueur noir les zones avec un manque de matière. Nous avons également réalisé sur un des nombreux capots imprimés un lissage de la face avant avec une technique à base d'enduit à bois que l'on a ensuite poncé puis peint avec de la sous-couche universelle blanche et enfin nous avons appliqué une couche de peinture noire mais cette technique nous a pris beaucoup de temps et le résultat n'a pas été à la hauteur.





CONCEPTION SCHÉMA 3D

Enfin, à la réception des différentes pièces nous avons fait un pré-montage pour vérifier que toutes les pièces s'intégraient comme il faut. Malheureusement, les raccords USB que nous avions commandés étaient trop longs et surtout trop rigides pour pouvoir être installés correctement et ne pas gêner le fonctionnement du ventilateur. Nous avons donc dû modifier la taille du boîtier puis réimprimer toutes les pièces. Tout cela est dû au fait que nous avons réalisé la conception 3D avant d'avoir les pièces, il nous manquait donc des informations importantes pour que tout s'insère correctement. Une fois les nouvelles pièces récupérées, nous avons utilisé un fer à souder avec une panne plate pour insérer dans le boîtier des inserts filetés en laiton qui servent à visser la carte et le capot. Ensuite, nous avons fixé le ventilateur à l'aide de tiges en silicone conçues exprès pour limiter les vibrations et le bruit et nous avons inversé certains fils du connecteur du ventilateur pour pouvoir le brancher sur les pins GPIO 4 (5V power) et 6 (Ground). Après cela, nous avons vissé avec des vis M2.5 et M3 la Raspberry Pi au fond du boîtier ainsi que les deux connecteurs USB et l'écran sur le capot. Nous avons terminé par brancher les deux câbles USB sur la carte ainsi que la nappe DSI et nous avons collé à l'aide d'un pistolet à colle les quatre pieds avec leur patin antidérapant. Pour terminer nous avons vissé le capot sur le boîtier avec des vis M2.5.





DÉTECTION ET PREMIERS OUTILS PERMETTANT DE FILTRER LES PÉRIPHÉRIQUES (USBCGUARD)

Notre projet comporte deux grandes parties, une matérielle et une autre logicielle, or nous avions une seule carte Raspberry Pi pour 4. C'est pourquoi nous avons fait le choix d'utiliser des machines virtuelles pour pouvoir développer et tester les programmes et les applications directement sur nos ordinateurs puis après vérification du bon fonctionnement des travaux de chacun, nous avons tout mis en commun pour l'implémenter sur la carte. Pour mettre en place les VMs, nous avons utilisé Oracle VM VirtualBox puis nous avons téléchargé le système d'exploitation officiel de Raspberry Pi dans sa version Desktop [10], que l'on a installé en suivant un tutoriel [11]. Le système d'exploitation de la carte appelé Raspberry Pi OS est un système d'exploitation optimisé et personnalisé provenant de la distribution Linux Debian 11 (bullseye).

Dans cette première partie nous allons expliquer le travail réalisé pour gérer les différents périphériques USB insérés sur la station et informer les utilisateurs sur le type d'appareil détecté. Nous nous sommes donc documentés sur internet à la recherche de projets ou de logiciels fonctionnant sous linux et remplissant la fonction de firewall USB, c'est-à-dire des programmes capables de manager les périphériques insérés en fonction des règles imposées par le gestionnaire du système. En premier, nous avons trouvé un projet open-source sur Github appelé USBWall permettant un filtrage dynamique des périphériques USB en fonction d'une liste centralisée de périphériques autorisés par l'utilisateur [12]. Malheureusement, nous avons essayé de l'installer sur la machine virtuelle mais nous n'y sommes pas parvenus. De plus, le projet était faiblement documenté et il n'avait pas été mis à jour depuis plusieurs années rendant encore plus compliqué son installation et sa configuration.



USBGuard

Malgré ses fonctionnalités intéressantes, nous avons dû chercher un autre outil et nous avons découvert USBGuard [13], un logiciel open-source et communautaire développé depuis 2015 spécifiquement pour linux afin d'offrir une protection contre les périphériques USB malveillants basée sur les propriétés des périphériques insérés. Pour commencer, nous avons mis à jour le système et installé USBGuard avec les commandes ci-dessous dans un terminal [14] :

```
● ● ●  
1 $ sudo apt-get update  
2 $ sudo apt-get -y install usbguard
```

Il est important de noter que USBGuard fonctionne avec un démon, c'est-à-dire qu'il utilise un processus qui tourne en arrière-plan sur le système d'exploitation dans l'attente d'évènements générés par le système lorsqu'un périphérique est inséré et il a pour particularité de ne pas pouvoir être contrôlé directement par l'utilisateur. Après avoir installé USBGuard, on a ensuite démarré le démon avec la commande système *start* puis on a utilisé la commande *enable* pour faire en sorte qu'il démarre à chaque démarrage du système :

```
● ● ●  
1 $ sudo systemctl start usbguard.service  
2 $ sudo systemctl enable usbguard.service
```

Ensuite, il est nécessaire de configurer USBGuard car ce dernier fonctionne grâce à un système de règles qui sont définies par l'utilisateur et qui permettent d'accepter ou de bloquer les périphériques en fonction d'un certain nombre d'attributs comme l'ID du vendeur, l'ID du produit, le nom du produit, le type d'interface utilisé et bien d'autres. Il est important de noter que par défaut, tous les périphériques USB insérés sont bloqués par USBGuard.



La première chose à faire est d'autoriser les périphériques internes et les drivers qui permettent l'utilisation des ports USB de la carte. Pour cela, nous avons utilisé la commande système `lsusb` ainsi que la commande `usbguard list-devices` qui permettent de lister tous les composants utilisant la norme USB sur la carte et nous avons obtenu trois interfaces :

- Deux contrôleurs xHCI (eXtensible Host Controller Interface) qui sont des composants matériels et logiciels internes responsables de la gestion des périphériques USB connectés à un contrôleur USB xHCI, fournissant une infrastructure de communication nécessaire au transfert de données à grande vitesse entre le système hôte et les périphériques USB qui prennent en charge les normes USB 3.0 ou ultérieures [15].
- Un hub USB 2.0 qui est un dispositif matériel qui augmente le nombre de ports USB disponibles sur la carte pour en avoir jusqu'à 4 qui fonctionnent en même temps. Il permet ainsi de connecter simultanément plusieurs périphériques USB et offre un moyen pratique de connecter et de déconnecter les périphériques USB.

On a alors pu les ajouter dans les règles du démon pour permettre le bon fonctionnement des ports de la carte soit en utilisant une commande USBCGuard mais cela fut assez fastidieux, soit en ouvrant le fichier de configuration des règles `rules.conf` contenu dans `/etc/usbguard` et en faisant un copier-coller des informations retournées par la commande `usbguard list-devices` dans le fichier (voir l'image ci-dessous). Après cela, il est nécessaire de redémarrer le démon pour qu'il prenne en compte les nouvelles règles en utilisant cette commande :

```
● ● ●
1 $ sudo systemctl restart usbguard.service

● ● ●
1 # Ajout via la commande allow-device
2 sudo usbguard allow-device '12d1:157c serial 0123456789ABCDEF' {-p}
3
4 # Ajout direct dans le fichier rules.conf
5 allow id 1d6b:0002 serial "0000:01:00.0" name "xHCI Host Controller"
6 hash "WLLK8WnDcOZsmC13ldzWRmahoyPD7Y+TLn1uJ4TB2GU="
7 parent-hash "MvRvALwPgWJIPJ4ZsIUCR65FuMV4N8+5X1/ZylmE9z4="
8 with-interface 09:00:00 with-connect-type ""
```



Puis, il faut configurer les règles pour gérer les autres périphériques, ceux qui seront insérés par les utilisateurs de notre système. Dans notre cas, nous ne voulons accepter que deux types de périphériques : les clés USB ou périphériques de stockage de masse pour vérifier qu'ils ne contiennent pas de virus et les claviers pour s'assurer que ce ne sont pas des injecteurs de commandes malveillantes. Pour écrire les règles liées aux périphériques cités précédemment, il faut comprendre comment ils sont identifiés. Une des spécificités de la norme USB est qu'elle identifie chaque périphérique avec 6 nombres hexadécimaux comme suit [16] :

- Les deux premiers nombres hexa correspondent à un attribut de classe qui identifie la fonctionnalité générale ou l'objectif du dispositif USB. Il regroupe les périphériques en grandes catégories en fonction de l'usage auquel ils sont destinés, comme par exemple 01 pour les périphériques audio ou 03 pour les périphériques de stockage de masse.
- Les deux nombres suivants correspondent à un attribut de sous-classe qui affine l'attribut de classe en fournissant des informations plus spécifiques sur la fonctionnalité ou l'implémentation de l'appareil. Il permet de différencier les dispositifs au sein d'une classe particulière, ainsi les périphériques audio avec une sous-classe en 01 vont correspondre à un contrôleur audio tandis que ceux en 02 vont correspondre à des haut-parleurs.
- Les deux derniers nombres correspondent eux à un attribut de protocole qui fournit des détails encore plus précis sur le protocole de communication utilisé par le périphérique USB. Il définit la manière dont les données sont échangées entre le périphérique et le système hôte comme RS-232 ou Ethernet pour les périphériques de communication [17].

Class	Description
1_1_0	Audio Control
1_2_0	Audio Streaming
1_3_0	MIDI Streaming



Class	Description
3	HID Class device (Human Interface device)
3_1_1	Keyboard
3_1_2	Mouse
3_1_2	Tablet
3_1_2	Touch screen
3	Joystick
3	Barcode reader



Une fois les périphériques identifiés, il suffit de faire la même chose que précédemment en ajoutant de nouvelles règles dans le fichier `rules.conf`. Ici, nous utilisons le mot-clé `allow` pour accepter les périphériques puis nous utilisons `with-interface equals {xx:xx:xx}` pour préciser l'identité du périphérique dans le format expliqué précédemment :

```
● ● ●  
1 allow with-interface {08:***}  
2 allow with-interface {03:01:01}  
3 allow with-interface {03:00:01}  
4 allow with-interface {03:01:01 03:***}  
5 reject with-interface all-of {08:*** 03:01:01}
```

L'utilisation de “`*`” permet d'autoriser absolument tous les périphériques de stockage de masse (08), simplification qui n'a pas pu être possible avec certaines règles qui sont presque identiques car nous avons remarqué que par exemple, les claviers pouvaient avoir de légères variations au niveau de la sous-classe ou ils pouvaient même avoir une double interface. Pour les interfaces utilisateur comme le clavier, nous n'avons pas utilisé d'astérisque pour simplifier les règles car il existe plusieurs autres périphériques comme le joystick, la souris ou la liseuse de code-barres qui possèdent presque la même interface et que nous ne voulons pas accepter sur notre système. La configuration d'USBGuard est donc terminée, l'étape suivante est d'utiliser les fonctionnalités de ce firewall USB pour obtenir des informations sur le type de périphérique inséré afin de renseigner l'utilisateur. Nous avons alors conçu un système en 4 étapes. La première étape correspond au déclenchement d'un script pour récupérer les informations d'USBGuard et pour ça, nous avons besoin de savoir à quel moment un dispositif USB est branché. Nous avons donc utilisé udev qui est un gestionnaire de périphériques pour les systèmes Linux et qui gère dynamiquement les périphériques dans le répertoire `/dev` [18]. Il est responsable de la détection et de la gestion des événements liés aux périphériques, tels que les connexions ou les déconnexions, de la création ou de la suppression des périphériques et permet de mettre en place un système de règles pour déclencher des actions lors de certains événements.



Notre règle udev se présente comme ceci :



```
1 ACTION=="add",SUBSYSTEM=="usb",ENV{DEVTYPE}=="usb_device",RUN+="/bin/systemctl restart display.service"
```

Le mot-clé ACTION suivi de *add* permet de lire la règle lors de l'insertion d'un périphérique, les mots-clés SUBSYSTEM et ENV suivis de *usb* et *usb_device* permettent de spécifier qu'il faut que le périphérique inséré possède la norme USB et enfin si toutes les conditions précédentes ont été remplies, le mot-clé RUN va permettre d'exécuter la commande qui redémarre le démon appelé *display*. Dans la deuxième étape, on va donc exécuter le démon *display* qui permet à son redémarrage d'exécuter un script bash qui stocke les informations sur le périphérique inséré :

```
1 [Unit]
2 Description=Service pour exécuter le script display.sh
3
4 [Service]
5 ExecStart=/var/GateKeepr/Securite/display.sh
6
7 [Install]
8 WantedBy=default.target
```

Les explications concernant l'utilisation de ce démon seront abordées dans la partie sur ClamAV. Ensuite dans la troisième étape, un script bash [19] appelé *display.sh* va s'exécuter et va permettre de créer un fichier de log *USBGuard_logs.txt* dans le répertoire */var/log* où se trouve tous les logs du système puis il va être rempli avec le retour de la commande *usbguard list-devices* qui permet d'obtenir une liste de l'ensemble des périphériques USB branchés avec des informations comme l'ID, le nom, le numéro de port ou encore le code d'interface. Au départ, nous avions utilisé la commande *usbguard watch* qui affichait un journal des événements USBGuard mais son fonctionnement en continu empêchait l'exécution de la suite du script.



À noter qu'en premier lieu, nous effaçons le fichier avec les logs même si ce dernier n'existe pas encore pour être sûr de ne pas avoir des informations dupliquées. La dernière ligne du script, quant à elle, permet d'exécuter un script python qui servira à analyser les logs et à produire un affichage. Au début du script se trouve un shebang (!#) [20] dont le rôle est d'indiquer au système d'exploitation que ce n'est pas un fichier binaire mais un script et de préciser l'interpréteur du script qui ici est bash.

```
● ● ●  
1 #!/bin/bash  
2 ### BEGIN INIT INFO  
3 # Short-Description: affichage des logs générés par USBCGuard  
4 ### END INIT INFO  
5  
6 rm /var/log/USBCGuard_logs.txt  
7 touch /var/log/USBCGuard_logs.txt  
8 usbguard list-devices >> /var/log/USBCGuard_logs.txt  
9 python3 /var/GateKeepr/Securite/display.py
```

La dernière étape est alors l'exécution du script python *display.py* dont le but est d'ouvrir le fichier *USBCGuard_logs.txt*, de le lire, d'isoler la dernière ligne correspondant au périphérique inséré puis de faire une analyse de cette dernière. L'analyse permet d'obtenir des informations importantes comme la classe, la sous-classe, le protocole, le statut (autorisé/rejeté) du périphérique et s'il possède une interface multiple. En fonction de ces différentes informations et d'un dictionnaire composé des classes et de leur signification, on est capable de générer une chaîne de caractères qui va être écrite dans un autre fichier appelé *GateKeepr.log* qui servira à l'affichage sur l'écran. Des cas spécifiques ont été créés pour mieux renseigner l'utilisateur sur les périphériques dangereux comme dans le cas d'un dispositif bloqué avec une interface multiple ou pour pouvoir lancer des analyses plus approfondies comme pour les claviers et les dispositifs de stockage. Le dictionnaire permettant de définir le type d'appareil en fonction de la classe du périphérique a été créé à partir d'un site internet [15] et du fichier *usb.ids* contenu dans */var/lib/usbutils/* qui donne les types de classe supportés par notre système d'exploitation.



```

1  try:
2      file = open(file_path, 'r', encoding="utf-8")
3  except:
4      print("Le fichier contenant les logs n'a pas pu être trouvé !!! \n")
5  else:
6      logs = file.readlines() # lecture des informations provenant d'USBGuard
7      for i in range (len(logs)):
8          if i == len(logs)-1:
9              USB_device = logs[i].split(" ") # analyse et découpage des logs
10         if "block" in USB_device:
11             blocked = 1
12         if USB_device[USB_device.index("with-interface") + 1] == '{': # gestion des interfaces multiples
13             if (USB_device.index("{") - USB_device.index("{")) > 2:
14                 USB_code = USB_device[USB_device.index("with-interface") + 2].split(":")
15                 multiple_interface = 1
16             else:
17                 USB_code = USB_device[USB_device.index("with-interface") + 2].split(":")
18             else:
19                 USB_code = USB_device[USB_device.index("with-interface") + 1].split(":")
20             USB_class = USB_code[0]
21             USB_subClass = USB_code[1] # code de classe, sous-classe et protocole du périphérique inséré
22             USB_protocol = USB_code[2]
23             # dictionnaire permettant de transformer le code de classe en une information compréhensible
24             USB_classification = {"00":"un dispositif ayant une classification USB non connue",
25                     "01":"un périphérique audio",
26                     "02":"un périphérique de communication (téléphone, modem, contrôleur ATM, contrôleur ethernet,...)",
27                     "03":"un périphérique d'interface utilisateur",
28                     "05":"un périphérique physique",
29                     "06":"un périphérique de capture d'images",
30                     "07":"un périphérique d'impression",
31                     "08":"un périphérique de stockage de masse",
32                     "09":"un hub USB",
33                     "0a":"un périphérique de données CDC (modem USB ou communication série)",
34                     "0b":"une carte à puce",
35                     "0d":"un périphérique de sécurité du contenu",
36                     "0e":"un périphérique vidéo",
37                     "ef":"un périphérique multiple",
38                     "fe":"un périphérique à application spécifique",
39                     "ff":"un périphérique d'un vendeur spécifique"}
40             # génération des informations pour l'affichage et gestion des cas particuliers
41             if blocked == 1 and multiple_interface == 1 and (USB_class!="03" and USB_protocol != "02"):
42                 write_file("/var/Log/GateKeepr.log","Le périphérique que vous venez d'insérer est reconnu comme une combinaison de périphériques. Nous vous conseillons de ne pas l'utiliser car il est potentiellement dangereux.")
43             elif USB_class == "03" and USB_subClass == "01" and USB_protocol == "01":
44                 write_file("/var/Log/GateKeepr.log","Le système a détecté que vous avez branché un clavier. Une analyse des frappes de clavier va être lancée pour déterminer le niveau de menace de ce périphérique.\n3")
45                 terminal_command="lxterminal"
46                 path_file="/var/GateKeepr/Securite/keylogger.py"
47                 os.putenv("DISPLAY",":0")
48                 os.system(path_file) # déclenchement de l'analyseur de frappes
49             elif USB_class == "03" and USB_subClass == "01" and USB_protocol == "02":
50                 write_file("/var/log/GateKeepr.log","Le système a détecté que vous avez branché une souris. Êtes-vous sûr qu'il s'agit du bon dispositif USB ?")
51             elif USB_class == "08":
52                 write_file("/var/log/GateKeepr.log","Le système a détecté que vous avez branché " + USB_classification.get(USB_class) +
53 ". Votre périphérique va être scanné à la recherche de virus.\n2")
54                 path_file="/var/GateKeepr/Securite/usb_id.sh"
55                 os.system(f"sudo bash {path_file}") # déclenchemenent de l'analyse antivirus
56             else:
57                 write_file("/var/log/GateKeepr.log","Le système a détecté que vous avez branché " + USB_classification.get(USB_class) +
58 ". Êtes-vous sûr qu'il s'agit du bon dispositif USB ?")
59     finally:
60         file.close()

```

Vous pouvez remarquer que la dernière partie du code ne sert pas à générer uniquement les informations pour l'affichage. En effet, grâce à la librairie python `os` qui offre la possibilité d'interagir avec le système d'exploitation, nous pouvons lancer des analyses plus approfondies sur certains périphériques. C'est le cas lorsque le périphérique est un clavier, ce qui va déclencher l'exécution du script python `keylogger.py` qui sera traité plus loin et c'est aussi le cas pour les périphériques de stockage de masse qui vont déclencher un script bash pour démarrer une analyse antivirus qui elle sera traitée dans la prochaine partie.

OUTIL DE SCAN DE LA CLÉ ET IDENTIFIER DES POTENTIELS VIRUS, MISE EN QUARANTINE ET SUPPRESSION (CLAMAV)



Maintenant que nous savons reconnaître quel type de périphérique vient d'être branché sur le boîtier, il faut que dans le cas d'une clé USB nous puissions scanner cette dernière à la recherche de logiciels malveillants. Il est important que cette mesure de protection soit efficace et complète et pour cela nous allons utiliser l'outil ClamAV et son utilitaire Clamscan [21][22].

ClamAV est un logiciel open source qui est spécialisé dans la détection et la suppression de logiciels malveillants y compris sur des périphériques USB. Fonctionnant à l'aide d'une base de données de signatures de virus constamment mise à jour, contenant des schémas uniques correspondant à des logiciels malveillants connus. Lorsqu'une clé USB est insérée dans un système, l'outil Clamscan, intégré à ClamAV, va permettre d'analyser le contenu de la clé à la recherche de ces signatures.

Lors de la phase d'incorporation de cet outils dans notre station, nous travaillions encore avec cette règle udev :



```
1 ACTION=="add", KERNEL == "sd[b-z]", RUN+="/chemin/vers/trigger.sh"
```

Cela signifie que lorsque le système avait créé le répertoire /dev/sdb correspondant à notre clé, le fichier *trigger.sh* se lançait. Lorsque le script se lançait, la clé n'était pas encore montée et donc impossible à analyser. Il fallait alors monter la clé puis l'analyser. Cependant notre station est équipée de deux ports USB cela signifie que deux clés peuvent être insérées en même temps ce qui pose le problème de l'endroit où nous allons monter la clé.



Pour pallier ce problème nous nous sommes intéressés à l'identifiant que possède une clé USB. Ce dernier est unique, nous pouvons l'utiliser comme point de montage en créant un répertoire avec cet identifiant.

Mais comment récupérer cet identifiant dans le script bash ?

Nous avons essayé de faire différentes commandes comme `lsusb` puis de sélectionner seulement la partie qui nous intéresse mais la commande ne rentrait pas ce qu'il fallait, comme si le script était exécuté trop tôt et que la clé n'avait pas encore cet identifiant. En s'intéressant un peu plus en détail sur les règles udev, nous nous sommes aperçus qu'elles pouvaient fournir au script des variables donc une variable qui s'appelle `{ID_SERIAL_SHORT}` [23]. Cette variable nous fournira exactement l'identifiant dont nous avons besoin. Ainsi le script permettait de récupérer cette variable, créer un dossier qui servira de point de montage, monter la clé, faire le scan puis démonter et supprimer le dossier pour laisser le système la monter automatiquement. Ainsi nous avions un fichier bash qui ressemblait à cela :

```
● ● ●
1 #Recupere l'identifiant de la cle USB
2 usb_id="${ID_SERIAL_SHORT}"
3
4 #Monte la cle dans un repertoire cree avec comme nom l'id
5 sudo mkdir /media/user/$usb_id
6 sudo mount /dev/sdb /media/user/$usb_id
7
8 #Scan le repertoire de la cle
9 sudo clamscan --recursive --cross-fs=yes --infected /media/user > /var/log/temp_log.txt
10
11 #Permet de copier vers le fichier GateKeepr.log et l'afficher a l'ecran
12 PYTHONPATH="/var/GateKeepr/Interface"
13 python -c 'exec(open("/var/GateKeepr/Interface/affichage.py").read());
14 copy_file_content("/var/log/temp_log.txt","/var/log/GateKeepr.log")'
15
16 #Demonte et supprime le repertoire de la cle
17 sudo umount /media/user/$usb_id
18 sudo rmdir /media/user/$usb_id
19
```



Sur l'utilitaire clamscan, l'option `-recursive` permet de scanner un répertoire et tous les sous-répertoires, c'est ce qui permet de s'assurer que tous les fichiers et répertoires de la clé soient analysés. L'option `-cross-fs=yes` permet d'autoriser clamscan à traverser ces points de montage et à analyser les fichiers présents dans d'autres systèmes de fichiers montés. Cela permet de scanner l'entièreté d'un périphérique et cela même s'il a plusieurs partitions comme par exemple les disques durs. Enfin, l'option `-infected` permet de renvoyer seulement les fichiers qui sont détectés comme infectés, cela permet de simplifier la gestion des résultats d'analyse en se concentrant uniquement sur les fichiers potentiellement dangereux.

Cette solution fonctionne très bien lorsqu'on branche une clé USB mais la règle udev étant trop spécifique, aucun script se lançait si un autre périphérique USB était inséré. Cela s'explique par le fait que la règle udev se lançait uniquement s'il y avait un nouveau répertoire comme `/dev/sdb` or ce répertoire peut être créé que par des périphériques de stockage de masse. Nous avons alors changé la règle udev pour qu'elle soit plus générale et qu'elle soit lancée à l'insertion de n'importe quel périphérique USB. On a également eu d'autres problèmes sur le temps que met le scan. Effectivement, le scan mettant un temps non négligeable pour parcourir toute la clé, cela rallonge le temps du processus. Cependant, les règles udev ne sont faites que pour exécuter des processus courts et au bout d'un certain temps, si les processus liés ne sont pas terminés, cette règle va les tuer. Une des possibilités pour pallier ce problème est d'utiliser un service qui va redémarrer à chaque fois que l'on insère une clé USB et qui exécutera le programme `display.sh`. Pour cela il faudra un fichier `.service` de configuration qui ressemble à cela :

```
1 [Unit]
2 Description=Service pour exécuter le script display.sh
3
4 [Service]
5 ExecStart=/var/GateKeepr/Securite/display.sh
6
7 [Install]
8 WantedBy=default.target
```



On a une partie Service ou on définit *ExecStart* qui va permettre de lancer un script au lancement du démon. Puis une partie *Install* où l'on précise que c'est uniquement lorsque la cible est atteinte qu'il doit lancer le script c'est à dire lorsqu'il est lancé. Cela permet de s'assurer que le programme se lancera au bon moment.

Puis nous modifions la règle udev pour qu'elle relance ce service :

```
● ● ●  
1 ACTION=="add",SUBSYSTEM=="usb",ENV{DEVTYPE}=="usb_device",RUN+="/bin/systemctl restart display.service"
```

Maintenant que nous avons modifié la règle udev pour qu'elle se lance à chaque fois qu'un périphérique est branché et non dès qu'un répertoire est créé, nous allons devoir modifier notre script *usb_id.sh*. Le temps que tous les scripts soient lancés, nous nous sommes aperçus que c'était quasiment identique au temps que mettait le système pour monter la clé. Nous n'allons donc plus monter manuellement la clé mais attendre que le système l'a fait.

Pour cela nous allons faire une boucle en bash qui nous permettra d'attendre qu'un fichier soit créé dans le dossier */media/user* qui est le point de montage des clés. Une fois que ce fichier sera créé cela voudra dire que nous pouvons faire le scan. Pour ce faire, nous allons utiliser l'outil *inotifywait* [24] qui va surveiller les modifications qui sont apportées à un répertoire telles que la création de fichier, les modifications, les suppressions, les déplacements, etc.

Nous utiliserons toujours la fonction python qui va copier les informations du scan qui sont initialement mises dans *temp_log.txt* puis va les coller dans *GateKeepr.log* ce qui va permettre de les afficher à l'écran.

Ainsi on met un listener sur la création de fichier/répertoire dans le chemin */media/user* et si il y'a création d'un fichier ou répertoire alors on sortira de la boucle *while*, pour exécuter le scan et la fonction python.



```
1 #Attend que la cle soit montee
2 while true; do
3     directory=$(inotifywait -e create --format '%w' /media/user)
4     if [ -n "$directory" ]; then
5         echo "Cle bien montee dans /media/user : $directory"
6         break
7     fi
8     sleep 1
9 done
10
11 #Scan le repertoire de la cle
12 sudo clamscan --recursive --cross-fs=yes --infected /media/user > /var/log/temp_log.txt
13
14 #Permet de copier vers le fichier GateKeepr.log et l'afficher a l'ecran
15 PYTHONPATH="/var/GateKeepr/Interface"
16 python -c 'exec(open("/var/GateKeepr/Interface/affichage.py").read());
17 copy_file_content("/var/log/temp_log.txt","/var/log/GateKeepr.log")'
```



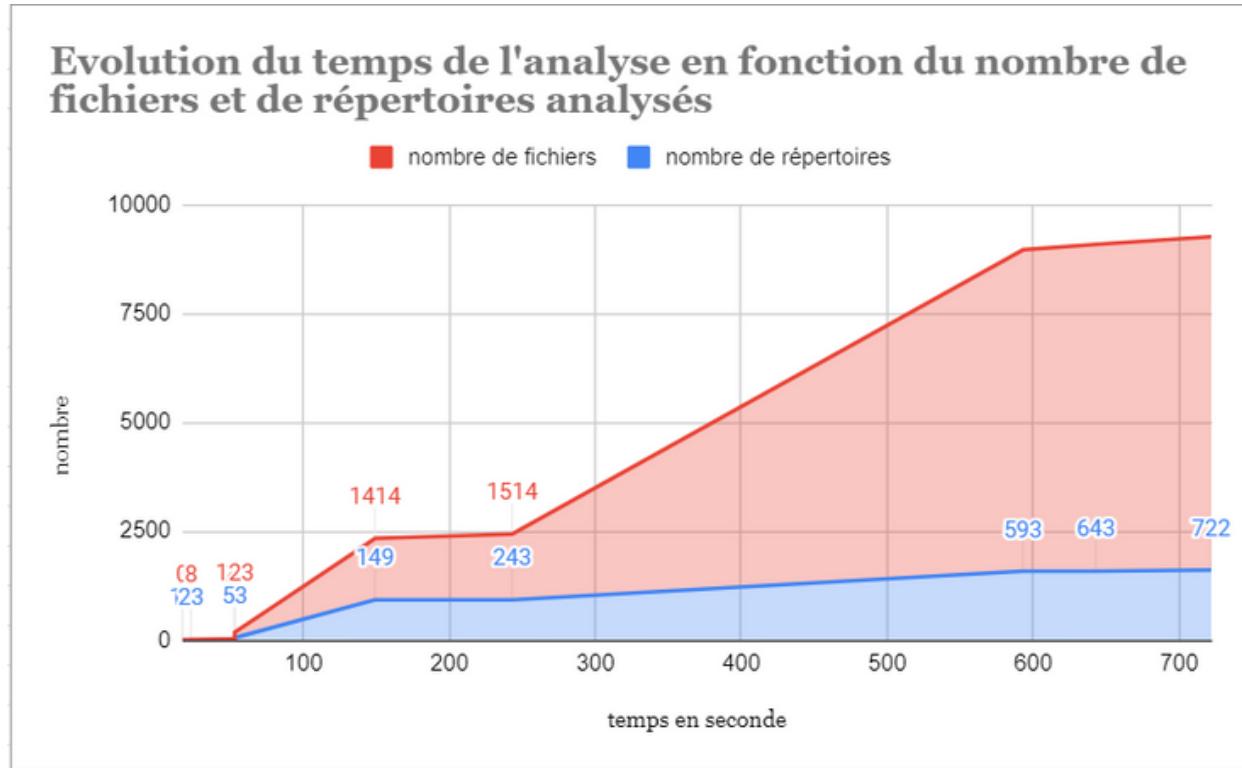
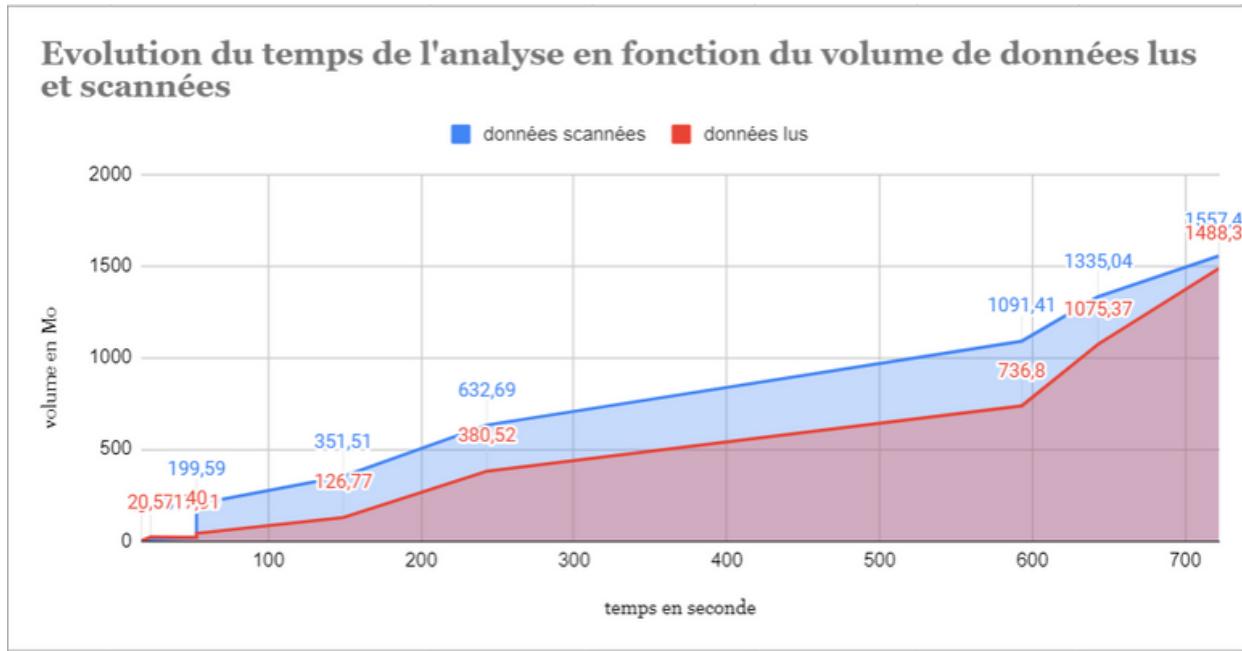
Avec ces modifications, cela nous permet de scanner le périphérique USB à la recherche de fichier malveillant et d'assurer à l'utilisateur d'utiliser sa clé comme il le souhaite. Cependant cet outil met un certain temps à scanner l'entièreté de la clé.

Effectivement, peu après avoir commencé à utiliser l'outil open source ClamAV, nous avons remarqué un temps de scan assez long. Cela même pour des petites clé USB avec très peu de données (moins de 100 MB). Nous avons décidé de faire une analyse du temps de scan par rapport à la taille de la clé qui était scannée. Nous voulions savoir s'il était possible de réduire ce temps en utilisant le démon (clamdscan). Lors de la réalisation de ce comparatif, le démon ne fonctionnait pas sur la machine virtuelle ce qui ne permettait pas l'objectif initial. Finalement, nous avons décidé de ne pas utiliser le démon pour d'autres raisons. Principalement parce qu'il ne donne pas assez de précision sur les résumés des scans. Cependant, il est toujours intéressant de regarder ce que nous avons appris quant au comportement du clamscan classique.

L'analyse s'est faite sous une machine virtuelle avec les mêmes caractéristiques que notre Raspberry Pi et le même système d'exploitation afin d'avoir des résultats cohérents. Aucun autre processus n'était en cours sur la machine virtuelle alors que lors des scans sur le boîtier, nous aurons l'interface en python qui tournera. C'est la seule différence notable, on peut donc normalement attendre quelques secondes supplémentaires sur la Raspberry Pi.



Les résultats peuvent être présentés sous la forme de graphiques que voici :



On remarque sans grande surprise que plus la clé contient de données et plus le temps d'analyse est long. Il est intéressant de noter que même une clé vide, sans aucun fichier nécessite un temps de scan d'une vingtaine de secondes. Il y aura donc forcément un temps minimum d'une vingtaine de secondes pour chaque scan.



Lors du test, le temps maximal était de 722 secondes (soit 12 min) pour une taille de 1560 MB. Si nous nous intéressons aux allures des graphiques, nous constatons que l'évolution du temps est quasiment linéaire. Il y a quelques disparités qui proviennent d'autres facteurs. En effet, en regardant le seconde graphique, on remarque que plus il y a de répertoires et de fichiers, plus l'analyse sera longue. A taille égale, un périphérique de stockage avec davantage de répertoires et de fichiers, mettra plus de temps à être analysé qu'un autre avec moins de répertoires et fichiers. Cela induit que si nous avons un gros fichier équivalent en taille à de multiple fichiers plus petits, ce premier sera analysé plus rapidement.

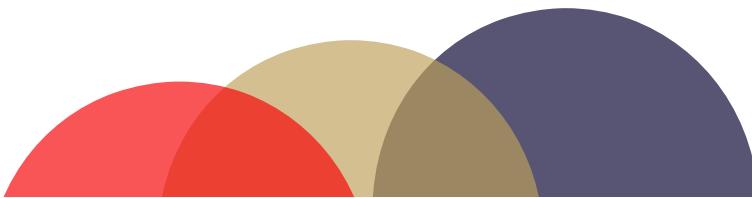
Lors des tests, nous avons observé que plus nous lancions des analyses sur les mêmes fichiers, plus ils étaient traités rapidement. Autrement dit, si nous lançons plusieurs fois un scan sur un périphérique avec exactement les mêmes données, le scan nécessite de moins en moins de temps à quelques secondes près. De la même manière, il y aura toujours plus de données scannées que de données lus, nous ne savons pas comment expliquer ce phénomène. Notre hypothèse serait que pour chaque sous répertoire scanné, le processus revienne au répertoire parent avant d'aller dans un autre répertoire. Cela pourrait expliquer l'influence du nombre de répertoires sur le temps d'analyse.

Nous avons effectué des recherches pour réduire ces temps d'analyse. Sur certains forums, nous avons pu y lire des astuces pour accélérer le processus en utilisant le multithreading et en réduisant la base de données des fichiers analysés [25]. Pour ce faire, il a fallu modifier la ligne de commande qui lance le scan en suivant la syntaxe suivante :

```
find <directory> -type f | xargs -P $(nproc) -n 1 clamscan [options]
```

Cette commande est spécifique pour le multithreading avec la variable nproc qui correspond au nombre de processeurs du système. Afin de limiter la base de données et d'exclure certains type de fichier par le biais de leur extension nous avons utilisé la commande :

```
clamscan --exclude="*.jpg,*jpeg,*.png,*.gif,* bmp" --no-mail --no-ole2 --  
no-html [options] <path>
```





En mixant les deux on obtient notre nouvelle commande pour les scans ClamAV en multithread avec une base de données réduite :

```
find <directory> -type f | xargs -P $(nproc) clamscan --  
exclude="*.jpg,*.jpeg,*.png,*.gif,*.bmp" --no-mail --no-ole2 --no-html
```

Lors des tests, cela n'a pas fonctionné correctement. Effectivement, le scan ne s'est pas lancé et en plus cela, il a démonté le périphérique USB. Nous avions des erreurs concernant l'option `-exclude` qui serait propre à une version antérieure. Un message de prévention apparaissait en disant 'Ignoring deprecated option'. Si nous utilisions le multithreading seul, le temps de scan c'était beaucoup amélioré en passant de trois à une minute. Néanmoins, cela ne fonctionne que sur la machine virtuelle. Sur la carte Raspberry Pi, l'option ne fait presque aucune différence. Nous avons alors cherché à comprendre cette différence. Il est possible que la carte ne supporte pas le multithreading car nous avons trouvé sur certains forums, des personnes qui indiquaient que le processeur n'était pas compatible avec la technologie SMT (Simultaneous multithreading) qui permet une exécution en parallèle de plusieurs threads sur un même cœur [26]. De plus, sur certain site, nous sommes tombés sur des articles spécifiant que l'outil ClamAV n'était pas prévu pour le multithreading [27] sauf pour le démon lorsque l'on utilise l'option `-multiscan` donc il semble assez logique que nos modifications n'aient pas marché.

Alors, pourquoi cela aurait-il fonctionné sur la machine virtuelle ?

Possiblement parce que nos machines Windows ou nos processeurs Intel auraient pris en charge le multithreading et auraient automatiquement réparti le processus entre les coeurs du processeur de l'ordinateur sur lequel la machine virtuelle tournait.

Ainsi, au final, nous n'avons pas trouvé de moyen de réduire le temps de l'analyse bien qu'il existe certaines libertés dans les options pour ce faire. Elles dépendent grandement de la programmation de l'antivirus et des caractéristiques hardware de notre carte.



KEYLOGGER

Étant donné que notre système peut être utilisé par des personnes voulant contourner la fonction première de notre boîtier en branchant directement des claviers pour y avoir accès ou même des clés USB qui injectent du code, il est important de se prémunir de ce genre d'attaque. Pour pallier cette problématique, nous avions d'abord pensé au projet UKIP (USB Keystroke Injection Protection) qui est un projet open source de Google permettant de bloquer les injections de commandes en fonction de la vitesse de frappe [28]. On pouvait notamment configurer le nombre maximum de touches que le périphérique ou l'utilisateur avait le droit de faire avant d'être détecté ou encore la vitesse de frappe. Malheureusement les touches et les commandes étaient exécutées jusqu'à ce que le blocage intervienne, ce qui nous a donc fait changer d'avis car cela rendait notre système trop vulnérable. Finalement nous avons opté pour un système de keylogger, c'est-à-dire un programme qui va intercepter les touches claviers et les analyser afin de détecter toute activité potentiellement malveillante. En faisant ce programme nous ne voulons pas intercepter les touches dans le but de collecter des informations diverses et variées comme pourrait le faire les keylogger traditionnels. Nous voulons bel et bien comparer ces touches et les différentes commandes à une base de données.

En utilisant une base de données, cela nous permet de nous adapter aux différents contextes d'utilisation de notre boîtier, il peut s'avérer que certaines commandes soient interdites dans une entreprise mais tout à fait inoffensives chez un particulier. Cette base de données nous apporte le côté flexible notamment par le fait que cette dernière est facile à mettre à jour et peut ainsi répondre plus rapidement aux nouvelles menaces. De plus, cela permet de se protéger un peu plus des attaques d'ingénierie sociale, c'est-à-dire des attaques ou des cybercriminels qui incitent des utilisateurs à effectuer des actions malveillantes.



Ce programme a été réalisé en python et sera appelé par le script *display.py* après avoir identifié que le périphérique qui vient d'être branché est un clavier ou se fait passer pour tel.

Dans la version implémentée dans la carte, les commandes sont analysées après avoir appuyé sur la touche entrer, si la commande est considérée comme malveillante alors le programme s'arrête et on affiche un message sur le boîtier. Si la commande ne comporte pas d'éléments dangereux nous ne faisons rien. Cependant si nous utilisons notre boîtier comme filtre sur un ordinateur, il est possible de mettre en place un buffer dans lequel nous allons stocker les commandes et, une fois que ce buffer est plein, les commandes s'exécutent.

Nous commençons par les différents import dont nous avons besoin dans ce programme ainsi que le shebang qui permet de préciser au système d'exploitation que le fichier qui va suivre est un script en langage Python. Pour les imports, nous avons besoin du module *keyboard* qui va nous permettre d'intercepter les touches clavier, du module *os* qui va permettre de lancer d'autres programmes et du module *signal* qui permettra de stopper le programme.

```
● ● ●  
1  #!bin/python3  
2  from pynput import keyboard  
3  from Xlib import X, display  
4  
5  import subprocess  
6  import os  
7  import signal
```

Nous avons ensuite la première fonction ainsi que les variables utilisées dans ce script.



```
1 # Liste de commandes considérées comme néfastes
2 commandes_nefastes = []
3
4 # Chaîne de caractères pour stocker la commande en cours de saisie
5 commande_en_cours = ""
6
7 # Fonction pour charger la blacklist depuis un fichier
8 def load_blacklist(file_path):
9     try:
10         with open(file_path, 'r') as file:
11             for line in file:
12                 command = line.strip() # Supprimer les espaces et les sauts de ligne
13                 commandes_nefastes.append(command)
14
15         write_file("/var/log/log_file.txt","Blacklist chargée avec succès.")
16     except FileNotFoundError:
17         write_file("/var/log/log_file.txt","Fichier introuvable :"+str(file_path))
18     except Exception as e:
19         write_file("/var/log/log_file.txt","Une erreur s'est produite lors du chargement de la blacklist :"+str(e))
```

La liste `commandes_nefastes` sera remplie par la fonction `load_blacklist` qui va venir ouvrir le fichier contenant les mots et commandes interdites et les ajouter à la liste. On a ensuite une variable `commande_en_cours` dans laquelle nous allons stocker chaque touche de frappe et les analyser une fois que la touche “entrer” sera appuyée. Nous avons décidé de gérer différentes erreurs qui peuvent arriver notamment si le chemin d'accès au chemin n'est pas le bon et qu'il n'arrive donc pas à le trouver ou alors d'autres erreurs moins usuelles. Cependant, le fichier restera à un emplacement fixe et ces gestions d'erreurs ne seront donc pas réellement utiles une fois le programme mis en place. Pour remplir notre fichier blacklist, nous avons regardé les commandes les plus dangereuses sur internet ainsi que certains mots [29]. On y retrouve des commandes puissantes comme la fork bomb qui permet de créer un grand nombre de processus jusqu'à ce que le système d'exploitation ait atteint sa limite de ressource, ou alors toute redirection vers des fichiers tels que `/dev/sda` qui pourrait écraser des contenus importants dont le système d'exploitation. Mais on y retrouve également des commandes plus basiques comme le sudo, qui permet d'exécuter des commandes en ayant des droits privilégiés, ou même `rm` et `mv` qui permet respectivement de supprimer et de déplacer des fichiers.

Vous retrouverez la liste complète dans le GitHub et plus précisément dans le fichier `blacklist.txt`.



```
1 # Charger la blacklist depuis un fichier
2 load_blacklist("/var/GateKeepr/Securite/blacklist.txt")
```

Dans le programme nous faisons appel à la fonction *load_blacklist* décrite plus haut pour pouvoir charger les commandes interdites.

```
1 #Fonction pour ecrire dans le fichier log
2 def write_file(path_file,text):
3     with open(path_file,'w') as file:
4         file.write(text)
5     return
6
7 #Permet d'arreter le programme a l'aide d'un signal
8 def simulate_end():
9     # Obtenir l'identifiant de processus (PID) du programme en cours
10    current_pid = os.getpid()
11
12    # Envoyer un signal de terminaison au processus en cours
13    os.kill(current_pid, signal.SIGINT)
14
15    #Permet de regarder si un element de la commande est nefaste
16    def check_command(commande):
17        words=commande.split(" ")
18        for elmt in words:
19            if elmt in commandes_nefastes:
20                return True
21        return False
```

Etant donné que nous allons tout écrire dans un fichier log pour ensuite gérer l'affichage à l'écran, il est important d'avoir une fonction qui va permettre cette action. Pour cela nous utilisons la fonction *write_file* qui va ouvrir le fichier en écriture ce qui signifie qu'elle va écraser le contenu existant s'il y en a un. Elle va ensuite écrire le contenu de la variable fournie en paramètre. Sur le côté fonctionnel, au début nous pensions que si une commande malveillante était tapée, le clavier ne devrait plus pouvoir être utilisé et le programme devait s'arrêter. La méthode *exit* ou *quit* a posé problème notamment s'il y a la présence d'un buffer. En effet, lorsqu'une commande interdite est saisie, le programme s'arrête mais exécute toutes les commandes tapées précédemment.



C'est pour cela que nous avons fait une nouvelle fonction qui va directement arrêter le processus qui fait tourner le programme en envoyant un signal au PID préalablement récupéré. Cependant, le fait de "démonter" le clavier pour qu'il ne puisse plus écrire puis de quitter le programme paraissait trop complexe, nous avons donc décidé de changer de stratégie. Cette nouvelle stratégie consiste à laisser le programme intercepter toutes les touches claviers et ce même s'il y a des touches ou des commandes interdites. La seule chose qui changera sera alors l'affichage en fonction de ce qui est tapé.

Pour la vérification des commandes, nous utilisons la fonction *check_command* qui va séparer la chaîne de caractère donnée en paramètre pour n'avoir accès que aux mots, puis nous faisons une vérification afin de savoir si un mot n'est pas dans notre liste de mots interdits.

Maintenant que nous avons toutes les fonctions dont nous avons besoin pour pouvoir gérer les commandes qui seront tapées et les impacts que cela aura, nous allons nous intéresser au listener, c'est-à-dire le mécanisme servant à détecter et gérer les événements du système. Nous allons le mettre en place sur le clavier et voir la fonction qui sera appelée à chaque fois qu'une touche est appuyée.

```
● ● ●  
1 # Ajout du gestionnaire d'événements pour intercepter les touches pressées  
2 listener = keyboard.Listener(on_press=on_keypress)  
3 listener.start()  
4  
5 # Boucle principale pour maintenir le programme en cours d'exécution  
6 try:  
7     listener.join()  
8 except KeyboardInterrupt:  
9     pass
```

On peut voir la création d'un listener puis une boucle *try / except* qui va permettre de faire tourner le programme en continu.



Pour la phase de développement nous laissons la partie de l'exception pour pouvoir interrompre le programme proprement. Cependant, une fois le produit fini, l'utilisateur ne devra pas pouvoir faire des raccourcis clavier, il faudra donc les bloquer. Nous allons maintenant voir la fonction que nous avons mis à notre listener quand une touche est appuyée : `on_press()`.

Pour commencer, nous allons regarder si la touche qui vient d'être appuyée possède un attribut de type `char`. Cette première condition va nous permettre de récupérer directement la touche si c'est un caractère imprimable, ou récupérer le nom de la touche dans le cas contraire. Par exemple si on appuie sur le '`a`' on obtiendra '`a`' par contre si on appuie sur la touche '`espace`' on obtiendra '`key.space`'.

```
1 # Fonction de traitement des touches pressées
2 def on_keypress(key):
3     global commande_en_cours
4
5     # Vérifier si la touche est un caractère imprimable
6     if hasattr(key, 'char'):
7         touche = key.char
8     else:
9         # Utiliser le nom de la touche pour les touches spéciales
10        touche = key.name
```

Cela nous emmène sur les conditions suivantes qui vont permettre de remplacer ces touches non imprimables par ce qu'elles font, le tout dans la `commande_en_cours`.



```
1 #Permet de faire un espace dans la commande en cours
2 if key==keyboard.Key.space:
3     commande_en_cours+=" "
4
5 #Permet de supprimer un element de la commande en cours
6 elif key == keyboard.Key.backspace:
7     commande_en_cours = commande_en_cours[:-1]
8
9 elif key==keyboard.Key.shift:
10    pass
11 # Vérifier si le raccourci Ctrl est pressé
12 elif key == keyboard.Key.ctrl:
13     write_file("/var/log/GateKeepr.log","Attention vous avez appuye sur la touche Controle\nCela peut etre malveillant ! ")
14
15 # Vérifier si le raccourci alt est pressé
16 elif key == keyboard.Key.alt:
17     write_file("/var/log/GateKeepr.log","Attention vous avez appuye sur la touche Alt\nCela peut etre malveillant ! ")
18
19 elif key == keyboard.Key.cmd:
20     write_file("/var/log/GateKeepr.log","Attention vous avez appuye sur la touche Windows\nCela peut etre malveillant ! ")
21
```

Comme expliqué, les touches espace et retour en arrière nous permettent de mettre un espace et de supprimer le dernier élément de la commande en cours. Pour ce qui est de la commande shift qui peut être utilisée en combinaison avec d'autres touches, il ne faut pas qu'elle impacte la commande en cours c'est pour cela qu'on lui donne l'instruction pass. Parmi les attaques qui utilisent un clavier, beaucoup utilisent des raccourcis clavier notamment par l'appui sur une touche ctrl, alt ou windows. C'est pour cela que nous avons décidé, en plus de bloquer la touche, d'afficher à l'écran un message particulier pour chaque touche. Nous allons maintenant voir la partie la plus importante de cette fonction: l'appui sur la touche entrer.

Effectivement, s'il y a un appui sur la touche entrer, cela signifie que la commande a fini d'être tapée. C'est à ce moment-là que nous devons vérifier ce qui vient d'être saisi pour être sûr qu'il n'y a rien de malveillant, nous faisons donc appel à notre fonction check_command(). Ainsi si nous détectons une commande malveillante alors une phrase s'affiche pour avertir l'utilisateur puis la commande en cours est réinitialisée. Et si on appuie sur une touche différente de celles citées précédemment on ajoute tout simplement la touche à la commande en cours.



```

1 elif key == keyboard.Key.enter:
2
3     if commande_en_cours:
4         # Analyse de la commande en cours
5         print(commande_en_cours)
6         #if any(nefaste in commande_en_cours for nefaste in commandes_nefastes):
7         #if commande_en_cours=="exit":
8             #simulate_end()
9         if check_command(commande_en_cours):
10             write_file("/var/log/GateKeepr.log","\\nLa commande en cours est détectée comme potentiellement néfaste. L'exécution est bloquée.")
11             #simulate_end()
12
13     # Réinitialisation de la commande en cours
14     commande_en_cours = ""
15
16 else:
17     # Ajout de la touche à la commande en cours
18     commande_en_cours += touche

```

Cette dernière fonction qui s'appelle `executer_commandes_buffer()` sert si jamais nous souhaitons utiliser notre station comme filtre sur un ordinateur, dans ce cas là nous devons exécuter les commandes d'un buffer, qu'il soit de taille 1 ou plus, après avoir vérifié si ces commandes sont bonnes :

```

1 # Exécute toutes les commandes du buffer
2 def executer_commandes_buffer():
3     print("Exécution des commandes :")
4     for commande in commandes_buffer:
5         subprocess.call(commande, shell=True)
6         print(f"- {commande}")
7     print("Fin de l'exécution des commandes.")
8
9     # Réinitialisation du buffer
10    commandes_buffer.clear()

```

Ainsi avec ce programme et ces fonctionnalités nous sommes en mesure, une fois qu'un périphérique de type clavier est branché, d'intercepter les touches et les commandes pour pouvoir les analyser et les bloquer. Cependant, lors des tests avec la rubber ducky, nous nous sommes aperçu que les commandes tapées par la clef USB malveillante étaient beaucoup trop rapides. Effectivement, dû à l'USB 3.0 qui est dans notre boîtier, le taux de transfert des données est plus rapide que le temps de lancement du programme. Une des solutions possible serait de recommencer le programme mais dans un langage plus bas niveau comme le C. Malgré l'efficacité du programme pour certaines attaques, il existe toujours des façons de le contourner et de pouvoir exécuter du code. Un vecteur d'attaque possible serait de taper lettre par lettre une commande dans un fichier puis l'exécuter. Ainsi à chaque lettre la touche entrer sera appuyé mais ne sera pas reconnu comme malveillant.

INTERFACE GRAPHIQUE (PYQT)



Dès l'amorce de notre projet, notre aspiration était de donner à l'utilisateur final la possibilité d'influer sur le fonctionnement de notre boîtier. En d'autres termes, nous souhaitions lui offrir une vision en temps réel de ce qui se déroule lorsqu'il branche un périphérique USB, tout en lui permettant de faire des choix et d'entreprendre certaines actions. Pour répondre à cette exigence, l'intégration d'un affichage était incontournable, avec pour rôle d'agir comme l'intermédiaire entre nos scripts et l'utilisateur. Sans cet affichage, notre projet perdrat une grande part de son utilité et l'expérience utilisateur s'en trouverait d'autant plus altérée. Notre objectif est de rendre GateKeepr accessible au plus grand nombre, quel que soit le niveau de compétences en matière de sécurité informatique. C'est pourquoi nous avons décidé d'intégrer une interface graphique dynamique, programmée en langage Python et exploitant la puissante bibliothèque PyQt5. Cette bibliothèque, que nous avions pour certains déjà utilisée, offre une navigation intuitive et conviviale, facilitant ainsi la prise en main du boîtier. Grâce à ces choix techniques, nous souhaitons offrir une expérience utilisateur fluide et agréable, sans pour autant compromettre les fonctionnalités avancées de GateKeepr.

Nous savions principalement ce que nous voulions avoir sur l'écran. Nous avions conscience que la plupart du temps aucun périphérique ne serait inséré. Le Raspberry pi doit être alimenté pour fonctionner et l'utilisateur n'a aucun moyen de l'éteindre sauf en débranchant le boîtier. Ainsi, seulement une infime partie du temps un utilisateur viendra utiliser notre boîtier et des actions seront effectuées. Il fallait avoir un écran d'attente qui sera l'affichage en continu lorsque rien ne se produit, lorsque l'utilisateur n'a pas inséré de périphérique.



INTERFACE GRAPHIQUE (PYQT)

La première version de l'interface utilisateur ne comportait que deux pages différentes. L'une lorsque rien ne se produit, et l'autre pour afficher les résultats des actions de notre station. Techniquement parlant, coder cela avec PyQt n'était pas le plus compliqué. Une fois que nous connaissons les bases de ce module avec ses multiples labels et fonctions ainsi que la mécanique de création des pages, tout s'enchaîne sans trop de difficultés. Le plus complexe était de rendre tout cela automatique. En effet, on ne veut pas que l'utilisateur ait besoin de faire quelque action pour lancer le scan ou afficher les résultats. Bien que cela aurait été plus simple à implémenter avec un simple bouton qui entraîne divers événements. Il fallait donc un moyen de rendre notre interface autonome. Sachant que l'interface serait un programme Python lancé en continu et que tout est réalisé en temps réel, les problèmes ont vite commencé à venir. La stratégie que nous avons employée pour changer de page automatiquement est d'utiliser un listener PyQt sur un fichier nommé *log_file.log*. Ce listener prend la forme suivante :

```
● ● ●  
1 # Start monitoring the file for changes  
2 file_path = "/var/log/log_file.txt"  
3  
4 global file_watcher  
5 file_watcher = QFileSystemWatcher([file_path])
```

L'idée est de regarder en continu le fichier *log_file.log* dans lequel est écrit les résultats des actions entrepris par GateKeopr, que ce soit le type de périphérique (avec le programme *display.py*), le résultat du scan ClamAV (programme *usb_id.sh*) ou encore le *keylogger.py*.

La ligne de code suivante nous permet d'effectuer une action dès que le listener rencontre une modification du fichier *log_file.log*.

```
● ● ●  
1 file_watcher.fileChanged.connect(lambda: switchPage(0, 1))
```



L'action que nous faisons ici est de lancer la fonction switchPage() depuis la page à index 1 vers la page à index 2. A ce moment il n'y avait encore que deux pages.

La fonction switchPage() ressemble à ceci :

```
● ○ ●
1 def switchPage(current_index, new_index, file_path=None, clear_file_flag=False):
2     if current_index == 0:
3         stacked_widget.setCurrentIndex(new_index)
4     else:
5         if clear_file_flag:
6             file_watcher.removePath(file_path) # Remove the file from the watcher temporarily
7             clear_file(file_path) # Clear the file
8             file_watcher.addPath(file_path) # Add the file back to the watcher
9         stacked_widget.setCurrentIndex(new_index)
```

Cette fonction a été modifiée à de nombreuses reprises. Son fonctionnement actuel est de changer la page courante en modifiant l'index de la variable stacked_widget. Cette modification est différente si on est sur la page d'accueil (index 0) ou sur une autre page. Effectivement, lors des changements de page, on supprime le contenu du fichier log_file.log après avoir déconnecté le listener pour ne pas changer une fois de plus de page. Si on passe à la page suivante, c'est qu'une action a été effectuée et qu'il faut supprimer le contenu du fichier pour le remplacer par le nouveau. On aurait pu faire différemment et à chaque fois qu'on écrit dans le fichier on s'assure qu'il soit vide. Cette approche nous permet de supprimer le contenu du fichier lorsque l'utilisateur est sur la page avec les résultats et revient sur la page de garde.

Dans cette fonction nous faisons appelle à la fonction clear_file() qui ne fait rien d'autre que de supprimer tout le contenu du fichier donc l'emplacement est passé en paramètre. Nous avons laissé les print en anglais pour le debug. Ils ne seront pas utilisés pour l'utilisateur.

```
● ○ ●
1 #Function to clear the log file
2 def clear_file(file_path):
3     try:
4         with open(file_path, 'w') as file:
5             file.truncate(0) # Truncate the file size to 0
6             print(f"Contents of {file_path} cleared.")
7     except IOError:
8         print(f"An error occurred while clearing {file_path}.")
```



Par la suite, notre objectif était d'afficher le texte dans l'emplacement réservé sur la seconde page. Pour ce faire, notre idée originelle était de détecter le changement de page et de lancer une fonction qui lirait le contenu du fichier log pour en formater l'affichage avant de l'écrire dans sur la page des résultats.

Nous avions alors comme code les deux lignes suivantes au niveau de la fonction main du fichier Python *GateKeepr.py*:

```
1 if stacked_widget.currentIndex()==0:  
2     stacked_widget.currentChanged.connect(lambda : affichage.parse_scan_result(file_path,text_label))
```

Comme nous n'avions que deux pages, cette technique fonctionnait parfaitement en corrélation avec la fonction *parse_scan_result()* du fichier *affichage.py*.

Voici la fonction en question :

```
1 def parse_scan_result(file_path, text_box= None):  
2     with open(file_path, 'r') as file:  
3         content = file.read()  
4  
5     # Extract potential viruses  
6     virus_lines = content.split('\n')  
7     potential_viruses = []  
8     for line in virus_lines:  
9         if line.startswith('/'): # If it's a header  
10            parts = line.split(':')  
11            virus_location = parts[0].strip()  
12            virus_type = parts[1].strip()  
13            potential_viruses.append((virus_location, virus_type))  
14  
15     # Extract number of potential viruses  
16     num_viruses = len(potential_viruses)  
17  
18     # Extract scan time  
19     scan_time_line = next((line for line in virus_lines[::-1] if line.startswith('Time:')), None)  
20     if scan_time_line is not None:  
21         scan_time_parts = scan_time_line.split(':')  
22         scan_time_seconds = scan_time_parts[1].split()[0].strip() if scan_time_parts else ""  
23     else:  
24         scan_time_seconds = 0  
25  
26     # Build the result string  
27     result = "Résultats:\n\n"  
28     result += "Nombre de virus potentiels: "+str(num_viruses)+"\n\n"  
29     result += "Emplacement des virus:\n\n"  
30     for virus_location, virus_type in potential_viruses:  
31         result += str(virus_location)+"\n "+str(virus_type)+"\n"  
32     result += "\nTemps de scan: "+str(scan_time_seconds)  
33  
34     # Set the result text in the PyQt label  
35     if text_box is not None:  
36         text_box.setText(result)  
37  
38     return potential_viruses
```



INTERFACE GRAPHIQUE (PYQT)

Cette fonction est adaptée au retour du scan ClamAV dont la forme varie selon les résultats des scan. Si des virus sont présents, le scan va indiquer l'emplacement des fichiers infectés dès leur découverte avant même le résumé du scan.

```
/media/admin/USB_DISK/testvirus/EICAR.COM:Eicar-SignatureFOUND  
/media/admin/USB_DISK/EICAR.COM: Eicar-Signature FOUND
```

----- SCAN SUMMARY -----

Known viruses: 8666781

Engine version: 0.103.8

Scanned directories: 6

Scanned files: 25

Infected files: 2

Data scanned: 28.15 MB

Data read: 15.84 MB (ratio 1.78:1)

Time: 16.291 sec (0 m 16 s)

Start Date: 2023:05:28 17:00:39

End Date: 2023:05:28 17:00:55

Ce fonctionnement à changé notre façon de raisonner. Nous avions un listener sur le fichier log qui induirait un changement de page ainsi que la lecture du fichier pour afficher les résultats. Or comme le scan écrit en plusieurs fois et non pas en une seule (sauf si aucun virus n'est détecté), il a fallu écrire le résumé du scan dans un fichier temporaire puis transférer ce contenu vers le fichier log_file.log pour entraîner le changement de page et l'affichage correct.

La fonction de transfert est basique, c'est de la lecture, puis de l'écriture dans deux fichiers différents.



INTERFACE GRAPHIQUE (PYQT)

```
1 def copy_file_content(source_file, destination_file):
2     try:
3         # Open the source file in read mode
4         with open(source_file, 'r') as source:
5             # Read the contents of the source file
6             content = source.read()
7
8         # Open the destination file in write mode
9         with open(destination_file, 'w') as destination:
10            # Write the contents to the destination file
11            destination.write(content)
12
13            print("File content copied successfully!")
14    except IOError:
15        print("An error occurred while copying the file.")
```

Maintenant que nous avons un changement de page automatique en fonction du résultat des scans ClamAV ainsi qu'un affichage personnalisé des résultats, nous avons pensé à ajouter des fonctionnalités pour l'utilisateur. En l'occurrence, deux boutons l'un pour supprimer les menaces trouvées et l'autre pour les mettre en quarantaine.

```
1 def suppress_files(label):
2     file_path = "/var/log/log_file.txt" # Replace with the path to the log file
3     virus_locations = parse_scan_result(file_path)
4
5     if not virus_locations:
6         label.setText("Pas de virus à supprimer")
7         return
8
9     result = "Conclusion : \n\n"
10
11    for location, _ in virus_locations:
12        try:
13            os.remove(location)
14            result += "Fichier supprimé: \n "+str(location)+"\n"
15        except FileNotFoundError:
16            result += "Fichier non trouvé: \n "+str(location)+"\n"
17        except PermissionError:
18            result += "Permission refusée: \n "+str(location)+"\n"
19    label.setText(result)
```



INTERFACE GRAPHIQUE (PYQT)

```
● ● ●
```

```
1 def quarantine_files(label, quarantine_directory):
2     #create the directory if doesn't exist
3     if not os.path.exists(quarantine_directory):
4         os.makedirs(quarantine_directory)
5
6
7     file_path = "/var/log/log_file.txt" # Replace with the path to the log file
8     virus_locations = parse_scan_result(file_path)
9
10    if not virus_locations:
11        label.setText("Pas de virus à mettre en quarantaine")
12        return
13
14    result = "Conclusion : \n\n"
15
16
17    #Join the location of the potential virus
18    for location, _ in virus_locations:
19        file_name = os.path.basename(str(location))
20        destination = os.path.join(quarantine_directory, file_name)
21
22        try:
23            shutil.move(str(location), str(destination))
24            result += "Fichier en quarantaine: \n\n "+str(location)+" \n -----> \n "+str(destination)+"\n"
25        except FileNotFoundError:
26            result += "Fichier non trouvé: \n "+str(location)+"\n"
27        except PermissionError:
28            result += "Permission refusé: \n "+str(location)+"\n"
29
30    label.setText(result)
```

Ces deux fonctions gèrent également l'affichage et le retour utilisateur par rapport aux actions effectuées. Elles intègrent les potentiels problèmes rencontrés lors de la manipulation notamment les erreurs de droit et les fichiers introuvables.

Une partie du code intéressant à comprendre est la classe `MainWindow()`. Cette classe crée les instances de toutes les pages et la variable `stacked_widget` qui sauvegarde la page actuelle et les autres (anciennement sous forme de pile car on n'avait que deux page d'où le nom de la variable). De plus, cette classe est la classe principale, appelée dans la fonction `main` pour servir d'objet à la création de toutes les autres pages et éléments graphiques.



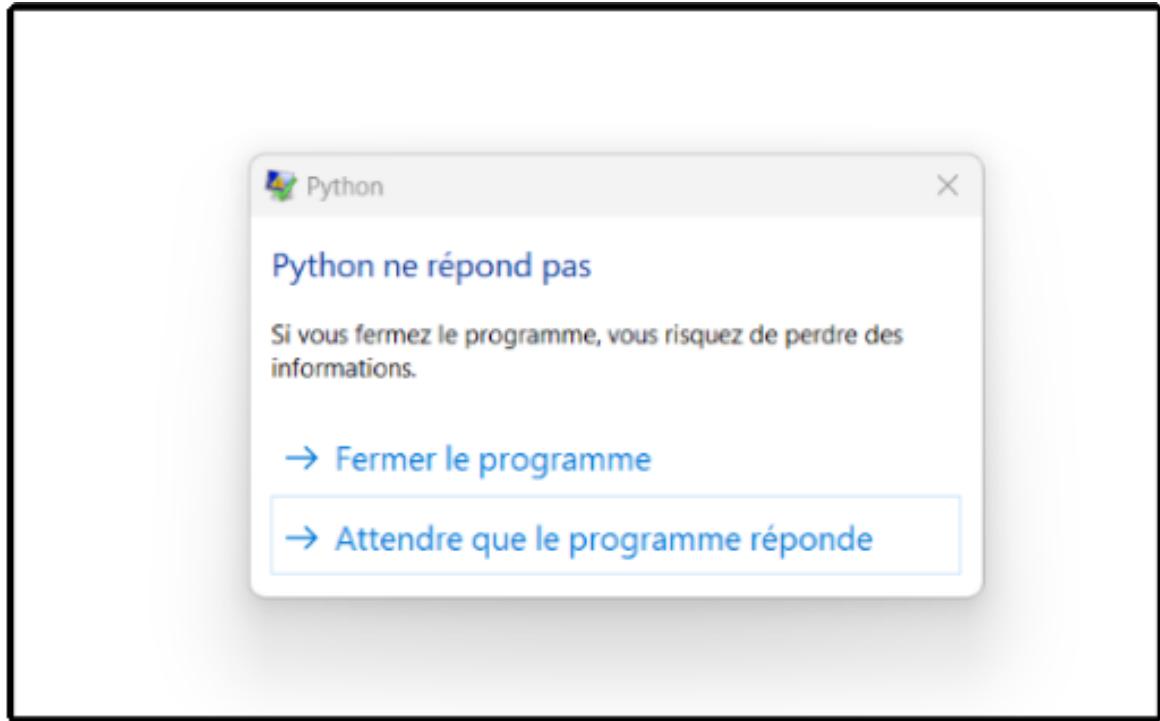
INTERFACE GRAPHIQUE (PYQT)

```
1 if __name__ == "__main__":
2
3     app = QApplication(sys.argv)
4     global window
5     window = MainWindow()
6     window.show()
7     #To hide the cursor
8     window.setCursor(QCursor(Qt.BlankCursor))
9
10    # Start monitoring the file for changes
11    file_path = "/var/log/GateKeepr.log"
12    eject_path = "/var/log/eject.log"
13
14    global file_watcher,eject_watcher
15    file_watcher = QFileSystemWatcher([file_path])
16    eject_watcher = QFileSystemWatcher([eject_path])
17
18    WritePID()
19
20
21
22    # if stacked_widget.currentIndex()==0:
23    #     stacked_widget.currentChanged.connect(lambda : affichage.parse_scan_result(file_path,text_label))
24
25    sys.exit(app.exec_())
```

C'est au final dans la classe `MainWindow()` que nous gérons les changements de page. Après plusieurs échecs et des recherches, nous avons compris qu'il était plus intéressant de s'occuper du changement des pages dans cette classe plutôt que dans la classe main. En effet, depuis la classe main, le changement de page ne fonctionnait qu'une seule fois. C'est logique car elle est appelée seulement au lancement du code. Au contraire, la classe `MainWindow()` est en perpétuelle exécution tant que l'application tourne. Nous pouvons donc utiliser le module `QTimer` de la bibliothèque `PyQt5.Core` pour exécuter une partie du code en boucle selon un temps bien précis. Nous avions à plusieurs reprises essayé de faire des boucles `while` ou `for` dans le code, cependant, cela n'est pas compatible avec `PyQt`. Sachant que l'application est en temps réel, si nous faisons une boucle, le reste du code est mis en attente ce qui a pour conséquence de soit faire planter l'ensemble du programme, soit d'avoir une fenêtre vide blanche.



INTERFACE GRAPHIQUE (PYQT)



Les dernières modifications que nous avons apportées sur l'interface sont la gestion de pages en fonction de l'éjection du périphérique. Le comportement actuel ne prend pas en compte cette possibilité, si l'utilisateur retire le périphérique avant les résultats, rien ne se produit. Avec les modifications, si l'utilisateur retire son périphérique avant les résultats, nous retournons sur la page principale et nous arrêtons tous les processus en cours d'exécution. Cela nous a posé problème étant donné que si nous ne faisons que le changement de page, les processus tels que le programme *keylogger.py* ou le scan ClamAV ont déjà été lancés. Si nous ne les stoppons pas, leur exécution continue et change les pages alors qu'aucun périphérique n'est branché et il n'y a donc aucune utilité à cela. C'est même pire, cela entraîne des problèmes de confusion lors de l'écriture dans le fichier *log_file.log*. Pour pallier ce problème, nous avons, dans un premier temps, ajouté une nouvelle règle udev pour effectuer une action lorsque l'utilisateur retire le périphérique. Cette règle lance un programme bash qui à pour rôle d'écrire dans un fichier *eject.log*. Nous avons intégré un nouvel listener dans le programme *GateKeepr.py* spécialement pour cette mécanique.



INTERFACE GRAPHIQUE (PYQT)

Cela nous permet d'éviter les conflits avec le fichier *log_file.log* et de changer les pages aussitôt que le périphérique est débranché. La seconde étape consistait en l'ajout d'une fonction python qui récupère le pid du programme lancé par *GateKeepr.py* pour le kill et stopper son exécution vu que l'utilisateur à débrancher précipitamment le périphérique avant la fin.



```
1 def pid_liste():
2     processes=psutil.process_iter()
3     name,pid=[], []
4     for process in processes:
5         try :
6             pid.append(process.pid)
7             name.append(process.name())
8             #print("PID : "+str(process.pid)+", name : "+str(process.name()))
9         except (psutil.NoSuchProcess,psutil.AccessDenied,psutil.ZombieProcess):
10             pass
11     if "keylogger.py" in name:
12         index = name.index("keylogger.py")
13         #print("L'index du processus 'keylogger.py' est :" +str(index))
14         #print("Le PID associe est : "+str(pid[index]))
15         # Obtenir l'identifiant de processus (PID) du programme en cours
16         current_pid = pid[index]
17
18         # Envoyer un signal de terminaison au processus en cours
19         #os.kill(current_pid, signal.SIGINT)
20         os.system("sudo killall -e keylogger.py")
21         switchPage(2,0,"/var/log/GateKeepr.log",True)
22     if "clamscan" in name:
23         #index = name.index("clamscan")
24         #print("L'index du processus 'clamscan' est :" +str(index))
25         #print("Le PID associe est : "+str(pid[index]))
26         os.system("sudo killall -e clamscan")
27         switchPage(1,0,"/var/log/GateKeepr.log",True)
28     else :
29         switchPage(3,0)
30     return
```



INTERFACE GRAPHIQUE (PYQT)

Au final, la classe MainWindow() contient les parties de code les plus importantes pour l'automatisation de l'interface. L'une des fonctions principales qui s'occupe de l'automatisation de la gestion des pages est dans cette fonction.

```
20      self.timer = QTimer(self)
21      self.timer.timeout.connect(lambda : self.updateCounter())
22      self.timer.start(1000) # Trigger every 1 second (1000 milliseconds)
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59      def updateCounter(self):
60          #match the correct next page depending on the next step GateKeepr will do
61          current_index = self.stacked_widget.currentIndex()
62          if current_index == 0:
63              file_watcher.fileChanged.connect(lambda : self.switchPage(1))
64              hide_show_gif/loading_label, False)
65              self.stacked_widget.currentChanged.connect(lambda: text_label_2.setText(affichage.process_log_file()))
66              current_index = self.stacked_widget.currentIndex()
67          elif current_index == 1:
68              USB_type = affichage.read_last_line()
69              if USB_type == "2":
70                  if not is_file_connected(eject_watcher,"/var/log/eject.log"):
71                      eject_watcher.addPath("/var/log/eject.log")
72                      hide_show_gif/loading_label, True)
73                      eject_watcher.fileChanged.connect(lambda: pid_liste())
74                      file_watcher.fileChanged.connect(lambda : switchPage(2))
75                      self.stacked_widget.currentChanged.connect(lambda: affichage.parse_scan_result(file_path, text_label_3))
76                      current_index = self.stacked_widget.currentIndex()
77          elif USB_type=="3":
78              if not is_file_connected(eject_watcher,"/var/log/eject.log"):
79                  eject_watcher.addPath("/var/log/eject.log")
80                  hide_show_gif/loading_label, False)
81                  eject_watcher.fileChanged.connect(lambda: pid_liste())
82                  file_watcher.fileChanged.connect(lambda : switchPage(3))
83                  self.stacked_widget.currentChanged.connect(lambda : text_label_4.setText(affichage.process_log_file()))
84                  current_index=self.stacked_widget.currentIndex()
85          else:
86              #eject_watcher.fileChanged.connect(lambda: pid_liste())
87              eject_watcher.removePath("/var/log/eject.log")
88              file_watcher.fileChanged.connect(lambda: text_label_2.setText(affichage.process_log_file()))
89              hide_show_gif/loading_label, False)
90              #self.stacked_widget.currentChanged.connect(lambda: text_label_4.setText(affichage.process_log_file()))
91              current_index = self.stacked_widget.currentIndex()
92          elif current_index == 2:
93              if not is_file_connected(eject_watcher,"/var/log/eject.log"):
94                  eject_watcher.addPath("/var/log/eject.log")
95                  hide_show_gif/loading_label, False)
96                  eject_watcher.fileChanged.connect(lambda:self.switchPage(0))
97                  file_watcher.fileChanged.connect(lambda:self.switchPage(0))
98                  current_index = self.stacked_widget.currentIndex()
99          elif current_index == 3:
100             keylogger=True
101             if not is_file_connected(eject_watcher,"/var/log/eject.log"):
102                 eject_watcher.addPath("/var/log/eject.log")
103                 eject_watcher.fileChanged.connect(lambda:self.switchPage(0))
104                 if keylogger==False:
105                     file_watcher.fileChanged.connect(lambda:self.switchPage(0))
106                     hide_show_gif/loading_label, False)
107                     current_index = self.stacked_widget.currentIndex()
```



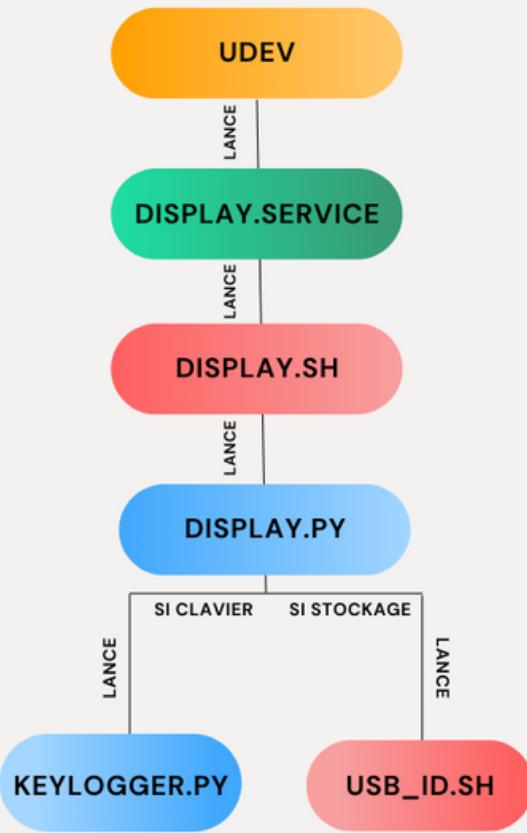
INTERFACE GRAPHIQUE (PYQT)

Afin de résumer l'intégralité de la gestion des pages et des différents fichiers exécutés, nous avons fait deux schémas.

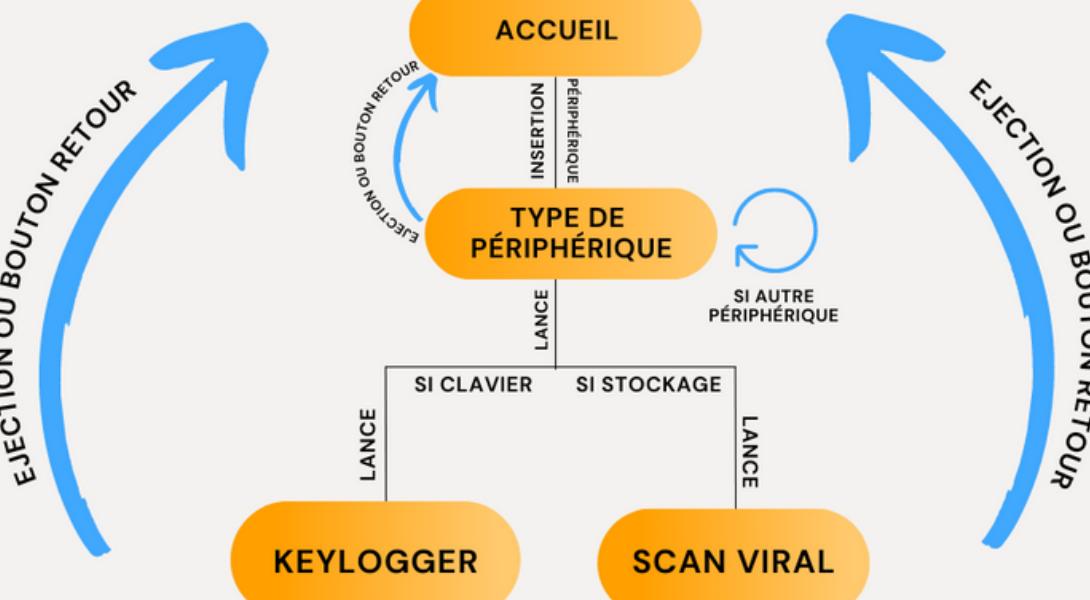
GESTION DES FICHIERS

LÉGENDE :

- FICHIER BASH
- FICHIER PYTHON
- PROCESSUS DÉMON



GESTION DES PAGES DE L'INTERFACE





DIFFÉRENTS TESTS

Afin de pouvoir bloquer un maximum d'attaques USB, il était essentiel pour notre équipe de comprendre le fonctionnement de différents périphériques souvent utilisés à des fins malveillantes. Nous avons eu l'opportunité d'emprunter certains de ces périphériques à notre tuteur, notamment le Digispark, le Rubber Ducky et le Bash Bunny, qui sont largement connus dans la communauté de la sécurité informatique:

- Le Digispark est un microcontrôleur basé sur l'ATtiny85 d'Atmel. Il peut être programmé pour imiter un périphérique USB, ce qui permet d'exécuter des actions automatiques sur le système cible, telles que l'injection de code malveillant. Nous avons étudié sa programmation et son mode de fonctionnement afin de mieux comprendre ses capacités et les types d'attaques qu'il peut effectuer, celui-ci étant d'une accessibilité déconcertante, tant au niveau du prix que de sa programmation en arduino.

```
● ● ●  
1 #define kbd_fr_fr  
2 #include "DigiKeyboard.h"  
3 void setup() {  
4     //empty  
5 }  
6 void loop() {  
7     DigiKeyboard.delay(2000);  
8     DigiKeyboard.sendKeyStroke(0);  
9     DigiKeyboard.sendKeyStroke(KEY_R, MOD_GUI_LEFT);  
10    DigiKeyboard.delay(600);  
11    DigiKeyboard.print("https://youtu.be/dAz$z(ZgXcAMt=$#s");  
12    DigiKeyboard.sendKeyStroke(KEY_ENTER);
```

- Le Rubber Ducky est un périphérique USB développé par la société Hak5, capable d'émuler un clavier. Il est programmé à l'aide d'un langage de script spécifique qui lui permet d'envoyer des commandes et des instructions au système cible. Nous avons examiné les scripts disponibles ainsi que l'utilisation de sites spécifiques permettant la génération de scripts sur mesure, et étudié la manière dont le Rubber Ducky pouvait être utilisé pour exécuter des attaques basées sur des frappes au clavier, telles que l'injection de commandes malveillantes. Le plus gros problème que nous avons rencontré sur celui-ci étant sa vitesse d'exécution nettement supérieure aux autres.



- Le Bash Bunny est un autre périphérique USB multifonctionnel, lui aussi développé par Hak5. Il est conçu pour exécuter différentes attaques sur un système cible, telles que le vol de données, l'exploitation de vulnérabilités et la capture de mots de passe. Son fonctionnement est assez simple, nous avons juste à placer les payloads dans les répertoires dédiés sur la clé, et un switch situé sur le côté nous permet de choisir entre les deux payloads chargés ou le mode de configuration. Sur le GitHub officiel [30] sont fournis les dossiers à mettre sur la clé ainsi qu'une bibliothèque d'exemples et d'outils. Nous avons donc pu explorer ses fonctionnalités et ses modes d'opération pour mieux comprendre ses techniques d'attaque, et sa capacité à émuler n'importe quelle interface a été précieuse dans nos divers essais.

Exemples de codes simples en shell appelés par le payload :

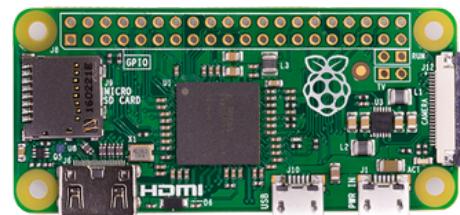
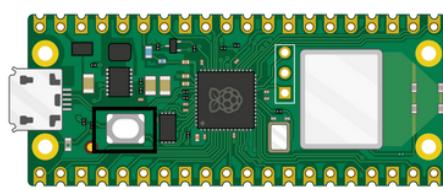
```
● ● ●  
1 # Sets up speech module  
2  
3 $s=New-Object -ComObject SAPI.SpVoice  
4 $s.Rate = -2  
5 $s.Speak("Votez Gate Keeper")  
6  
7 # Opens the browser with a video  
8  
9 Start-Process "https://www.youtube.com/watch?v=dQw4w9WgXcQ"
```

```
● ● ●  
1 #!/bin/bash  
2 #  
3 # Title: Mass Storage, Keyboard and Ethernet controller Test  
4 # Description: Bash Bunny payload to test mass storage, keyboard and ethernet controller attack mode functionality and check ou display.py script  
5 # Version: 1.0  
6 # Category: Test  
7  
8 # Init  
9 LED SETUP  
10  
11 # La bash bunny est en configuration stockage de masse  
12 ATTACKMODE STORAGE  
13 Q DELAY 15000  
14  
15 # Ensuite elle passe en mode clavier  
16 ATTACKMODE HID  
17 LED ATTACK  
18 QUACK STRING echo Hello World  
19 Q ENTER  
20 LED STAGES  
21 Q DELAY 15000  
22  
23 # Enfin elle se configure en tant que contrôleur ethernet  
24 ATTACKMODE HID ECM_ETHERNET  
25  
26 LED FINISH
```



En plus de ces périphériques empruntés, nous avons également utilisé des Raspberry Pi Pco et Zéro W qui appartenaient à l'un des membres de notre groupe :

- La Raspberry Pico est la plus petite carte de la gamme avec un prix de seulement 5 €, disposant d'un processeur dual-core RP2040 elle a un profil très intéressant pour faire du pentest physique. Un projet Github appelé pico-duky [31] a notamment été créé pour transformer cette carte en une Rubber Ducky avec certaines améliorations comme la prise en charge des fichiers en ducky script non compilés, la possibilité de mettre plusieurs payload et la plus grande variété de langue de clavier disponible. De plus, l'utilisation d'un jumper entre les pins permet un fonctionnement similaire à la Bash Bunny et son switch.



- Quant à la Raspberry Pi Zero W, c'est la grande sœur de la Pico, avec un coût de 18 € elle se positionne entre la Pico et la carte classique à la fois en termes de performance et en termes de taille. Elle possède des caractéristiques intéressantes comme son module WIFI ou encore son emplacement de carte microSD qui lui permet d'accueillir un véritable système d'exploitation et de stocker des fichiers. L'utilisation d'un projet GitHub appelé P4wnP1 A.L.O.A [32] permet de transformer la carte en un outil de pentest physique sans fil et comparable à la Bash Bunny grâce à une version customisée et épurée de la célèbre distribution de pentest Kali Linux. Son fonctionnement est simple, il faut brancher la carte à un ordinateur cible via le port data et non le port power. Avec un deuxième périphérique (attaquant, pouvant être un pc ou un smartphone), servant à la configuration et l'envoi de payload à distance, nous avons juste à nous connecter au réseau WIFI généré par la carte et à l'adresse : <http://172.24.0.1:8000> pour avoir accès aux différents outils de configuration et d'exécution. Presque tout est configurable ce qui fait d'elle un outil redoutable, bien plus que la Bash Bunny, même si cela fait plusieurs années qu'il n'y a pas eu de mise à jour donc la configuration est un peu hasardeuse.

DIFFÉRENTS TESTS



- Quant aux payloads, ils sont codés en JavaScript et permettent essentiellement lancer des attaques d'injections de commandes mais en réalité, il est possible de faire tout et n'importe quoi. De plus, un GitHub propose des payloads déjà écrits [33].

```

1 # lance la video de notre projet sous linux
2 layout('fr');
3 delay(500);
4 typingSpeed(500,200);
5 type("wget https://youtu.be/ST22Cu-a1sA");
6 press("ENTER");
7
8 type("uuuuuuu\n")
9 type("uu$$$$$$$$$$$$$uu\n")
10 type("uu$$$$$$$$$$$$$uu\n")
11 type("u$$$$$$$$$$$$$uu\n")
12 type("u$$$$$$$$$$$$$uu\n")
13 type("u$$$$$$$$$$$$$uu\n")
14 type("u$$$$$$$$$$$$$uu\n")
15 type("u$$$$$$$$$$$$$uu\n")
16 type("u$$$$$$: :$$$$: :$$$$$u\n")
17 type(":$ $$: u$u $ $$:\n")
18 type(" $$u u$u u$$\n")
19 type(" $$u u$$u u$$\n")
20 type(" :$$$$$uu$$$ $$$uu$$$:\n")
21 type(" :$$$$$$: :$$$$$$:\n")
22 type(" u$$$$$$$u$$$$$u\n")
23 type(" u$:$:$:$:$:u\n")
24 type(" uu$ $ $ $ $uu\n")
25 type(" u$$$ $$$$u$u$$$ u$$$ \n")
26 type(" $$$$uu :$$$$$$: uu$$$$$\n")
27 type(" u$$$$$$$uu :::: uuuu$$$$$\n")
28 type(" $$$: :$$$$$$$uu uu$$$$$$: :$$:\n")
29 type(" :: :$$$$$$$uu ::::\n")
30 type(" uuuu :$$$$$$$uu\n")
31 type(" u$$$uuu$$$$$$$uu :$$$$$$$$$uuu$$\n")
32 type(" $$$$$$::: :$$$$$$$:\n")
33 type(" :$$$$: :$$$$::\n")
34 type(" $$$: $$$:\n")
35 press("ENTER");
36 press("ENTER");
37 type(" Votez pour GateKeepr !! \n")

```

P4wnP1 A.L.O.A.

USB SETTINGS WIFI SETTINGS BLUETOOTH NETWORK SETTINGS TRIGGER ACTIONS HIDSCRIPT EVENT LOG GENERIC SETTINGS

USB Gadget Settings

Enabled
Enable/Disable USB gadget (if enabled, at least one function has to be turned on)

Vendor ID
Example: 0x1d6b
0x1d6b

Product ID
Example: 0x1337
0x1347

Manufacturer Name
MaMe82

Product Name
P4wnP1 by MaMe82

Serial Number
deadbeef1337

CDC ECM
Ethernet over USB for Linux, Unix and OSX

MAC addresses for CDC ECM

RNDIS
Ethernet over USB for Windows (and some Linux kernels)

MAC addresses for RNDIS

Keyboard
HID Keyboard functionality (needed for HID Script)

Mouse
HID Mouse functionality (needed for HID Script)

Custom HID device
Raw HID device function, used for covert channel

Serial Interface
Provides a serial port over USB

Mass Storage
Emulates USB flash drive or CD-ROM



SÉCURITÉ DU CODE

Un enjeu majeur de notre système a été de le rendre le plus sécurisé possible, en particulier au niveau du système d'exploitation et du code. En effet, en tant que produit de sécurité il est important de faire en sorte que le système ne soit pas corrompu et qu'il puisse rester opérationnel le plus longtemps possible. Il existe un risque d'attaque physique dans laquelle un individu pourrait débrancher le système, ouvrir le boîtier pour détruire la carte ou encore changer la carte micro SD contenant le système d'exploitation mais nous avons estimé que ce type d'attaque est peu probable et nous avons conçu notre boîtier en conséquence. Par contre, les attaques logicielles sont beaucoup plus probables et plus compliquées à déceler. Notre système a été conçu de sorte à analyser les frappes dans un buffer séparé du reste du système pour en cas de danger les bloquer et nous avons aussi utilisé un antivirus pour scanner les périphériques de stockage. Malgré toutes ces solutions de sécurité, des failles peuvent subsister comme l'exploitation de failles zero day, une combinaison de frappes non blacklistées ou encore un virus qui passerait au travers de l'antivirus. Afin de limiter l'impact de ces attaques sur le logiciel, on a étudié nos programmes principalement codés en python à la recherche de failles susceptibles d'être utilisées pour compromettre le système. Pour cela on a commencé par regarder des guides de bonnes pratiques sur la programmation d'applications sécurisées [34][35] mais nous n'avons pas pu vraiment en tenir compte car notre système n'exige pas d'authentification ni de chiffrement de données. Nous avons également essayé de réduire au maximum les droits de lecture, d'écriture et d'exécution de certains fichiers avec la commande `chmod` pour limiter la modification des codes et le vol d'informations. Nous avons ensuite décidé d'utiliser des analyseurs de code statique qui sont des programmes utilisant divers outils pour obtenir des informations sur le fonctionnement des codes sans jamais les exécuter [36]. Le premier analyseur appelé SafetyCLI [37] et développé par la société PyUp Cybersecurity permet de vérifier les dépendances python via un terminal et de produire un rapport des vulnérabilités trouvées. Son installation et son utilisation sont très simples :



```
theo@raspberry:~ $ safety check
=====
          /$$$$$$
         /$$$$$/ | $$ \  $$|/$$$$$$| /$$$$$/ | $$ /$$/
        | $$ $$ | | $$ | $$|/$$$$$$/| | $$ | $$|/$$ |
       \ \_ $$ | | $$ | $$|/$$$$$$/| | $$ | $$|/$$ |
        /$$$$$/| | $$ | $$|/$$$$$$/| | $$ | $$|/$$ |
        | $$ | $$|/$$ |$$|/$$ |$$|/$$ |$$|/$$ |$$
by pyup.io
=====

REPORT

Safety is using PyUp's free open-source vulnerability database. This data is 30 days old and limited.
For real-time enhanced vulnerability data, fix recommendations, severity reporting, cybersecurity support,
https://pyup.io or email sales@pyup.io

Safety v2.3.5 is scanning for Vulnerabilities...
Scanning dependencies in your environment:

-> /usr/local/lib/python3.9/dist-packages
-> /home/theo/.local/lib/python3.9/site-packages
-> /usr/lib/python3/dist-packages

Using non-commercial database
Found and scanned 141 packages
Timestamp 2023-06-20 14:25:58
36 vulnerabilities found
0 vulnerabilities ignored

=====
VULNERABILITIES FOUND
=====
```

Une fois l'analyse terminée, le rapport nous a indiqué avoir trouvé 36 vulnérabilités dans les différentes librairies python installées sur la carte qui pour la plupart étaient d'anciennes versions. Nous avons donc mis à jour manuellement les librairies utilisées par nos programmes pour avoir les derniers correctifs de sécurité. Ensuite nous avons utilisé le programme open-source Bandit [38] développé par PyCQA (Python Code Quality Authority) dont le but est de détecter les failles de sécurité basiques dans les programmes python. Pour l'installer et l'utiliser, il suffit de quelques commandes :

```
● ○ ●
1 $ virtualenv bandit-env
2 $ python3 -m venv bandit-env
3 $ source bandit-env/bin/activate
4 $ pip install bandit
5 $ bandit -r -v /path to the code/
```

Nous avons analysé nos quatre fichiers python un par un et nous avons obtenu plusieurs erreurs. La plupart des erreurs concernent l'utilisation des librairies os et subprocess dans les fichiers *display.py*, *keylogger.py* et *gateKeepr.py* car l'analyseur estime que démarrer un processus ou exécuter des commandes dans un shell depuis un script python présente des risques importants, notamment d'injection de commandes.



Bandit



```
(bandit-env) theo@raspberry:~ $ bandit -r /home/theo/Documents/code/keylogger.py
[main] INFO profile include tests: None
[main] INFO profile exclude tests: None
[main] INFO cli include tests: None
[main] INFO cli exclude tests: None
[main] INFO running on Python 3.9.2
Run started:2023-06-20 20:44:35.572463

Test results:
>> Issue: [B404:blacklist] Consider possible security implications associated with the subprocess module.
Severity: Low Confidence: High
CWE: CWE-78 (https://cwe.mitre.org/data/definitions/78.html)
More Info: https://bandit.readthedocs.io/en/1.7.5/blacklists/blacklist\_imports.html#b404-import-subprocess
Location: /home/theo/Documents/code/keylogger.py:5:0
4
5     import subprocess
6     import os

----->> Issue: [B602:subprocess_popen_with_shell_equals_true] subprocess call with shell=True identified, security issue.
Severity: High Confidence: High
CWE: CWE-78 (https://cwe.mitre.org/data/definitions/78.html)
More Info: https://bandit.readthedocs.io/en/1.7.5/plugins/b602\_subprocess\_popen\_with\_shell\_equals\_true.html
Location: /home/theo/Documents/code/keylogger.py:115:8
114     for commande in commandes_buffer:
115         subprocess.call(commande, shell=True)
116         print(f"- {commande}")

-----
Code scanned:
    Total lines of code: 79
    Total lines skipped (#nosec): 0

Run metrics:
    Total issues (by severity):
        Undefined: 0
        Low: 1
        Medium: 0
        High: 1
    Total issues (by confidence):
        Undefined: 0
        Low: 0
        Medium: 0
        High: 2
Files skipped (0):
```

Cependant les vulnérabilités qui sont triées à la fois en fonction de leur严重性 mais aussi en fonction de leur probabilité d'occurrence ne sont pas toutes dans la zone rouge. De plus, nous avions déjà connaissance de ces problèmes mais nous n'avons malheureusement pas d'autre choix que de faire ainsi pour assurer le bon fonctionnement du système. Enfin, nous avons utilisé une extension dans le logiciel Visual Studio Code appelée Snyk [39] qui scanne et fournit une analyse de code, y compris des dépendances open-source et des configurations d'infrastructure en tant que code. Après avoir téléchargé l'extension et l'avoir configuré en suivant les instructions de la documentation, nous avons lancé un scan sur l'ensemble de nos scripts en python. L'analyse n'a pas révélé de failles de sécurité mais seulement deux recommandations pour l'amélioration du code concernant l'écriture dans les fichiers qui n'ont pas d'impact direct sur la fiabilité de notre système.



CODE SECURITY
Snyk found no vulnerabilities! ✓
Analysis finished at 11:37 PM, 06/20/23

CONFIGURATION
Snyk found no issues! ✓
Analysis finished at 11:37 PM, 06/20/23

CODE QUALITY
Snyk found 2 issues
Analysis finished at 11:37 PM, 06/20/23

- ✓ H display.py code - 1 issue
 - H Use binary mode in open (current ...)
- ✓ H keylogger.py code - 1 issue
 - H Use binary mode in open (current ...)

H Use binary mode in `open` [:19] (current mode is `a` [:19]) to avoid encoding-related issues for written file, on Windows or with Python 3.
High This issue happens on line 19

This issue was fixed by 371 projects. Here are 3 example fixes.

sebastien/tahchee ◀ Example 1/3 ▶

```
directory = os.getcwd()  
# The write function only creates a file if it does not exist  
def write( path, data ):  
    if os.path.exists(path): return  
    fd = open(path, "w")  
    fd = open(path, "wb")  
    fd.write(data)  
    fd.close()
```

**snyk**

Nous pouvons aussi noter que le langage python n'est pas conseillé pour développer des applications sécurisées pour différentes raisons dont les principales sont le nombre de vulnérabilités présentes dans les packages et le manque de contrôle [40]. L'ANSSI recommande le langage C pour sécuriser les applications sensibles [41]. Nous avons également pensé à ce qui pourrait être changé pour améliorer la sécurité et la fiabilité de notre système comme l'utilisation d'un antivirus payant tel qu'Avast ou Kaspersky mais aussi l'ajout de nouvelles fonctionnalités comme la mise à jour sécurisée du système.



CONCLUSION

CONCLUSION

Pour clore ce rapport et ces deux mois de projet, nous pouvons affirmer que c'était une expérience très enrichissante pour chacun d'entre nous. En effet, nous avons à la fois acquis de nouvelles compétences techniques, notamment dans le domaine de la cybersécurité. Nous étions étrangers quant aux normes USB et leur gestion par les systèmes d'exploitation. Ces compétences fraîchement acquises nous seront grandement utiles pour les deux prochaines années en filière cybersécurité. Nous avons pris plaisir à travailler sur GateKeepr. C'est initialement un projet que nous avons proposé. Il nous tenait à cœur de travailler sur un projet avec du sens, qui répond à une problématique d'actualité délaissé par les grandes entreprises. Tout au long de ces deux mois de recherches, d'essais, de bug et de réussites, nous avons créé des liens entre les membres de l'équipe. Nous n'avons pas seulement développé des connaissances, des compétences techniques, mais aussi des atouts managériaux que l'on pourrait qualifier de 'soft skills'. Il nous a fallu nous organiser pour mener à bien ce projet, rendre des comptes toutes les semaines, avancer chacun de notre côté sur certaines tâches ce qui nécessite une communication et une organisation pointilleuse. Finalement, nous sommes désormais à la fois, plus familier avec les menaces ainsi que le fonctionnement des périphériques USB et à la fois mieux organisés et prêts à travailler en groupe de nouveau. Nous continuerons tous les quatre en filière cybersécurité l'année prochaine à l'ESIEE et dans le cadre de nos études, nous aimerions continuer ce projet afin de porter GateKeepr encore plus loin. Nous sommes déjà conscients de certains points d'amélioration. Plus particulièrement, le fait de revoir le choix du langage de programmation et d'utiliser du C qui est plus sécurisé et bas niveau que le python que nous avons actuellement. Nous souhaitons de même, pouvoir détecter et prévenir les attaques des USB Killer, ces clés USB qui envoient une décharge de courant sur un port pour créer une surtension et complètement détruire les composants d'un ordinateur.



CONCLUSION

Une autre voie pourrait être de retravailler la partie Keylogger, l'analyse des touches de clavier, afin d'avoir notre boîtier constamment branché à un ordinateur et faire office de filtre. Concernant, l'interface graphique, il est vrai que notre choix s'est rapidement porté vers la librairie PyQt. Néanmoins, si nous changeons de langage et voulons quelque chose de plus simple, il nous faudra nous renseigner sur d'autres possibilités comme GTK, wxWidgets et CEGUI.

Nous sommes fiers de ce que nous avons accompli, du produit final que nous avons pu présenter lors de la journée des projets. Nous avons pu sensibiliser le public concernant les menaces USB et leur offrir une solution. Pour les personnes avec plus de connaissances techniques, nous avons pu expliciter le fonctionnement et la technique qui se cachaient derrière notre station de détection des menaces USB. Un grand nombre des visiteurs de notre stand nous ont demandé un prix d'achat de GateKeepr. Nous l'avons compris, il nous reste encore beaucoup à améliorer avant d'avoir un projet totalement complet et pourquoi pas envisager une commercialisation.



REMERCIEMENTS



REMERCIEMENTS

Nous souhaitons exprimer notre sincère gratitude à toutes les personnes qui ont contribué à la réussite de notre projet GateKeepr. Tout d'abord, nous aimerais remercier chaleureusement Monsieur Perrotton, notre tuteur dévoué, qui nous a accompagnés et guidés tout au long de ce parcours passionnant lors de nos réunions hebdomadaires. Votre expertise et vos conseils précieux ont été d'une aide inestimable pour l'avancé du projet. Nous tenons également à exprimer notre reconnaissance envers Monsieur Bues et Monsieur Llorens pour nous avoir prêté leur matériel essentiel, ce qui a rendu ce projet possible et nous a permis de mener à bien nos expérimentations. Nos remerciements vont également à Monsieur Pagazani, qui a imprimé notre boîtier en 3D avec précision et professionnalisme. Enfin, nous souhaitons adresser nos remerciements les plus sincères à l'école ESIEE Paris et à toute l'équipe de la journée des projets, qui nous ont offert l'opportunité de vivre ces moments inoubliables. Ces souvenirs resteront gravés dans notre mémoire tout au long de nos études et au delà.

Valentin, Théo, Camille et Mathis

BIBLIOGRAPHIE

Lien vers l'intégralité des codes: <https://github.com/Skorthis/GateKeepr>

- [1] <https://www.honeywellforge.ai/us/en/campaigns/industrial-cybersecurity-threat-report-2022>
- [2] <https://www.numerama.com/tech/162737-et-vous-vous-feriez-quoi-dune-cle-usb-trouvee-dans-la-rue.html>
- [3] <https://arsen.co/blog/attaque-usb-definition>
- [4] <https://www.honeywellforge.ai/us/en/whitepaper/usb-hardware-attack-platforms-cybersecurity-report>
- [5] <https://prod-edam.honeywell.com/content/dam/honeywell-edam/pmt/hps/products/software/cyber-security/smx/White-Paper-USB-Security-Myths-vs-Reality.pdf?download=false>
- [6] <https://sospc.name/nombre-ordinateurs-par-systeme-exploitation-monde/>
- [7] <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/>
- [8] <https://docs.clamav.net/Introduction.html>
- [9] <https://raspberrytips.fr/temperature-raspberry-pi/>
- [10] <https://www.raspberrypi.com/software/raspberry-pi-desktop/>
- [11] <https://pimylifeup.com/raspberry-pi-virtualbox/>
- [12] <https://github.com/usbwall/usbwall>
- [13] <https://usbguard.github.io/>
- [14] <https://usbguard.github.io/documentation/compilation.html>
- [15] <https://www.usb.org/defined-class-codes>
- [16] <http://www.xat.nl/riscos/sw/usb/class.htm>
- [17] https://access.redhat.com/documentation/fr-fr/red_hat_enterprise_linux/7/html/security_guide/sec-using-usbguard
- [18] <https://doc.ubuntu-fr.org/udev>
- [19] https://doc.ubuntu-fr.org/tutoriel/script_shell
- [20] <https://fr.wikipedia.org/wiki/Shebang>
- [21] <https://docs.clamav.net/>
- [22] <https://doc.ubuntu-fr.org/clamav>
- [23] [https://wiki.archlinux.org/title/Udev_\(Fran%C3%A7ais\)](https://wiki.archlinux.org/title/Udev_(Fran%C3%A7ais))
- [24] <https://linux.die.net/man/1/inotifywait>
- [25] [https://wiki.archlinux.org/title/ClamAV_\(Fran%C3%A7ais\)](https://wiki.archlinux.org/title/ClamAV_(Fran%C3%A7ais))

BIBLIOGRAPHIE

- [26] <https://raspberrypi.stackexchange.com/questions/117303/can-one-core-of-the-pi-4-run-multiple-hardware-threads-simultaneously>
- [27] <https://clamav-users.clamav.narkive.com/u3lbbs1Z/using-clamscan-with-multiple-cores>
- [28] <https://github.com/google/ukip>
- [29] <https://fr.techtribune.net/linux/commandes-les-plus-dangereuses-vous-ne-devriez-jamais-les-executer-sous-linux/437501/>
- [30] <https://github.com/hak5/bashbunny-payloads>
- [31] <https://github.com/dbisu/pico-ducky>
- [32] https://github.com/RoganDawes/P4wnP1_aloa
- [33] <https://github.com/NightRang3r/P4wnP1-A.L.O.A.-Payloads>
- [34] <https://korben.info/developpement-securise-apprendre-a-maitriser-risque.htm>
- [35] <https://leclerc-web.fr/comment-securiser-proteger-son-code-python/>
- [36] <https://geekflare.com/fr/find-python-security-vulnerabilities/>
- [37] <https://pyup.io/safety/>
- [38] <https://github.com/PyCQA/bandit>
- [39] <https://docs.snyk.io/integrations/ide-tools/visual-studio-code-extension>
- [40] <https://itrgames.com/articles/193242/est-ce-que-python-est-securise.html>
- [41] <https://www.ssi.gouv.fr/guide/regles-de-programmation-pour-le-developpement-securise-de-logiciels-en-langage-c/>