Rapport Projet Parser Fat32/mbr
en python

Julien Climent
Jerry Yong-busson
Robin Geoffroy
Théo Dessolin
Rodolphe Picot

Sommaire

1. Contexte……3
2. Approche (fichier « main.py »)……3
3. Architecture générale (fichier « section.py »)……3
4. Parsage du mbr (fichier « mbr.py »)……4
5. Parsage du secteur de boot (fichier « FAT32SectorBoot.py »)……5
6. Parsage du secteur d'info (fichier « FAT32InfoSector.py »)……5
7. Parsage d'un répertoire (fichier « FAT32Directory.py »)……5
8. Manuel d'utilisation……5
9. Résultats……6
10. Répartition des tâches……6

1. Contexte :

Nous avons en notre possession des dumps de RAM où sont enregistrées des informations sur le système de fichiers de la machine hôte. Nous savons que le système de fichier est le système de fichier FAT32. Par ailleurs, nous savons aussi que le mbr de la machine compromise est présent dans la RAM. Le but du programme sera donc de retrouver ces structures.

2. Approche (fichier « main.py »):

Comme nous avons en notre possession le dump de la RAM et non du disque dur de la machine, nous pouvons en déduire que les structures de données du système de fichiers sont en elles-mêmes intactes mais placées de façon aléatoire dans la RAM. Un exemple, le mbr ne se trouve pas au début de la RAM alors qu'il se trouve sur le premier secteur sur le disque dur.

C'est pour cela qu'il nous est nécessaire de scanner tout le dump. Lors de ce scan, nous essayons de parser les structures que nous avons implémenté (le mbr, le FAT32sectorBoot, le FAT32InfoSector et les répertoires).

Pour gagner du temps, nous ne testons pas à partir de tous les octets de la RAM mais nous nous permettons d'avoir un pas de 0x10 car les structures sont alignées en mémoire.

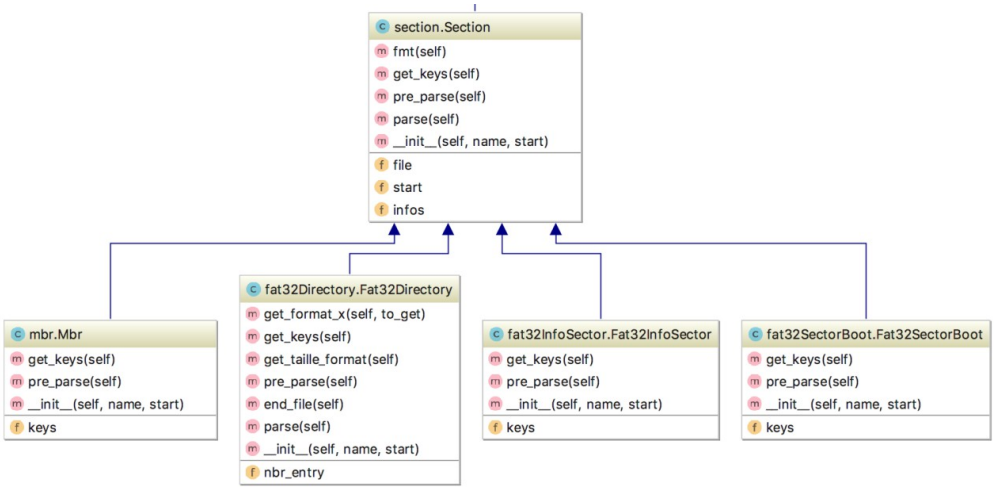
En ce qui concerne l'ordre des structures testées, nous testons le mbr en dernier car sa fonction de pré-parsage (voir la partie sur le fichier « section.py » pour l'utilité du pré-parsage) est plus faible que les autres structures.

3. Architecture générale (fichier « section.py ») :

Etant donné que ce qui change principalement entre chaque structure sont leurs champs et la taille de ces derniers.

C'est dans une optique d'une factorisation de code qu'il a été décidé de faire une classe mère qui centralisera les fonctions que devront implémenter les classes filles qui représenterons les structures voulues. On peut donc imaginer reprendre cette classe *section* pour faire un parser autre qu'un parser FAT32/mbr.

Voici un diagramme de classe représentant la relation entre *section* et les autres classes :



La classe mère *section* fait essentiellement une chose, elle parse selon un format qui sera défini par la classe fille et enregistre dans l'attribut *infos*.

Voici les méthodes qu'une classe fille devra implémenter, tout d'abord, la méthode *pre_parse()*. Dans cette méthode, une classe fille essaiera de parser des champs qui sont spécifiques à la structure qu'elle est censée représenter, typiquement elle pourra essayer de parser un magic number, si elle n'arrive pas à trouver ces éléments spécifiques, elle lève une exception. Ce mécanisme nous est utile pour perdre le moins de temps possible lorsqu'on parse mal une structure, ce qui arrivera souvent puisqu'on procède à un scan de la RAM.

Pour pouvoir donner le format de ce qu'on veut parser à la classe mère *section* une classe fille devra redéfinir la méthode *get_keys()* qui doit renvoyer une liste composé de tuple à deux éléments, le premier élément d'un tuple sera le nom du champ à parser et le deuxième élément la taille de ce champ (en ce qui concerne le format utiliser pour représenter la taille du champ, voir la documentation de la librairie Struct).

4. Parsage du mbr (fichier « mbr.py ») :

Comme la classe *mbr* est une classe fille de la classe *section*, elle doit implémenter les méthodes *pre_parse()* et *get_keys()*.

Concernant la méthode de pré-parsage, le mbr possède un magic number qui est *0xaa55* qui se trouve toujours au même offset par rapport au début du mbr. Il suffit dont de tester si on trouve ce magic number à cet offset pour savoir si on est bien tombé sur un mbr.

Pour la méthode *get_keys()*, elle renvoie juste un attribut de classe représentant le format du mbr. Ce choix d'attribut de classe est dû au fait qu'un mbr à un format qui ne bouge jamais, inutile que toutes les instances aient leur propre format. D'ailleurs, le fait de passer par une méthode que doit redéfinir la classe fille permet à cette dernière d'avoir un format qui permet d'avoir un format dynamique (il faudra alors surcharger la méthode *parse()*, voir l'exemple de « FAT32directory.py »).

5. Parsage du secteur de boot (fichier « FAT32SectorBoot.py ») :

A l'instar du mbr, la classe F*at32SectorBoot* doit aussi implémenter les méthodes *pre_parse()* et *get_keys()*.

Cette classe présente elle aussi un magic number qui est d'ailleurs le même que le mbr, c'est pour cela que sa fonction de pré-parsage doit regarder un deuxième élément pour l'identifier ce qui explique le parsage de la chaîne de caratère « FAT32 » (avec 3 espaces).

Pour la méthode *get_keys()*, la seule différence avec le mbr sont les champs et leur format dans l'attribut de classe que renvoie la méthode.

6. Parsage du secteur d'info (fichier « FAT32InfoSector.py ») :

La mécanique des deux méthodes à implémenter se retrouve encore avec cette classe qui présente cette fois deux magics number, *0x41615252* et *0x61417272*.

7. Parsage d'un répertoire (fichier « FAT32Directory.py ») :

Cette classe permet de parser un répertoire excepté le répertoire racine. En effet chaque répertoire commence par les mêmes lignes, une ligne reprentant le fichier « . » et une ligne représentant le fichier « .. » (voir documentation officielle).

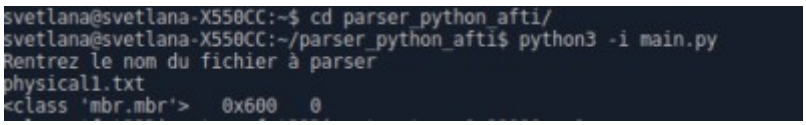
Cette classe a un fonctionnement particulier, en effet elle surcharge la méthode *parse()* et ne fait pas que renvoyer un attribut de classe dans sa méthode *get_keys()*.

En effet, un répertoire est composé d'entrées qui ont un certain format mais un répertoire a un nombre d'entrées utilisées que nous ne connaissons pas à l'avance. C'est pour cela qu'il nous faut redéfinir la méthode *parse()* de la façon suivant :

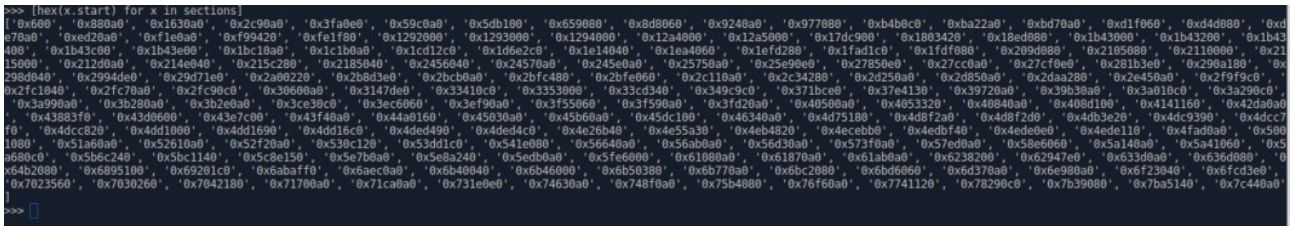
On appelle la méthode *parse()* de *section* à chaque fois qu'on voit qu'il y a encore une entrée utilisée. On définit ainsi *get_keys()* de telle façon à ce que *parse()* ne va pas écraser des informations quand elle sera utilisée plusieurs fois.

8. Manuel d'utilisation :

La manière d'utiliser le programme est la façon suivant, il faut exécuter la ligne de commande « python3 -i main.py » puis rentrer le nom du fichier à parser.



Ensuite, l'utilisateur se retrouvera à l'intérieur de l'interpréteur python3 grâce à l'option -i et pourra effectuer des traitements avec les données ainsi récupérées.



9. Résultats :

Nous avons récupérés les structures importantes du FAT32/mbr comme le mbr, le secteur de boot, le secteur d'info et un certain nombres de répertoires (voir fichier « resultats.txt »).

10. Répartition des tâches :

	Rodolphe	Jerry	Theo	Julien	Robin
Architecture/ conception					
Recherche Documentaire/ tests sur le dump					
section.py					
mbr.py					
Fat32dir					
Fat32section					
Fat32Boot					
relecture/norme					
Rapport					
Main					