Chapitre 3
Les concepts fondamentaux de GNU/Linux

1. Introduction

Ce chapitre est dédié aux concepts fondamentaux propres à l'environnement GNU/Linux. Les notions abordées seront illustrées au travers de la famille Debian, mais pourront s'appliquer également aux autres distributions. Nous considérons ces notions nécessaires à la compréhension des études de cas qui seront détaillées dans le chapitre GNU/Linux : études de cas.

Les concepts présentés ne sont qu'une partie de l'étendue des connaissances existantes pour ce système, mais apportent, selon nous, les bases indispensables pour utiliser au mieux les outils et exploiter correctement les résultats obtenus. Nous ferons référence à de nombreux ouvrages pour compléter ou préciser certaines notions.

2. Historique

Le système d'exploitation GNU (*GNU's Not Unix*) a débuté en 1983 lorsque **Richard Stallman** décida de développer un système d'exploitation compatible UNIX. Sept années plus tard, le projet **GNU/Hurd** proposait un ensemble de programmes et de bibliothèques et s'appuyait sur le micro-noyau GNU/Mach. Bien que ce micro-noyau semblait prometteur, c'est finalement vers le noyau **Linux**, développé par **Linus Torvald**, que Richard Stallman s'orienta. GNU/Linux était né et avec lui des myriades de déclinaisons à venir.

Notons que le noyau Hurd a fait son chemin, certes plus modeste, avec la Debian GNU/Hurd. Le système d'exploitation macOS s'appuie sur Mach pour son noyau monolithique Xnu.

Nous avons limité nos concepts aux distributions **GNU/Linux**, ce qui exclut par exemple Chrome OS ou Android. Chrome OS s'appuie sur Linux complété d'un environnement d'exécution basé sur le navigateur Chrome de Google. Android, quant à lui, s'appuie également sur Linux et emprunte quelques outils à GNU, mais cela reste à la marge. Les développeurs d'Android ont développé leurs propres outils (par exemple Bionic remplace la GNU C Library).

Ci-dessous les principales familles GNU/Linux :

– Debian, probablement l'une des branches les plus connues par les utilisateurs, notamment avec la démocratisation d'Ubuntu. Officiellement fondée le 16 août 1993 par Ian Murdock, cette distribution a été pensée comme ouverte, laissant libre chaque développeur d'y contribuer. Debian est la concaténation du nom de son épouse Debra et de lui-même Ian. De nombreuses distributions s'appuient sur Debian (https://upload.wikimedia.org/wikipedia/commons/9/96/Liste_des_distributions_Linux.svg) dont les plus connues sont Ubuntu, Linux Mint, Kali Linux, etc.

– Red Hat, une société et une distribution GNU/Linux créées en 1994. Connue dans le monde professionnel pour héberger de nombreux serveurs (*Red Hat Entreprise Linux* ou RHEL), elle propose également des produits gratuits comme Fedora ou bien CentOS. À noter que RedHat a annoncé en 2020 la fin de CentOS en tant que reconstruction de RHEL 8. Il reste donc CentOS Stream qui servira de branche de développement pour RHEL et des alternatives telles que Rocky Linux ou bien AlmaLinux.

– Suse ou *Software und System Entwicklung* (SUSE) est une compagnie allemande qui sort en 1994 une distribution GNU/Linux. Elle se découpe en versions payantes SUSE Linux Enterprise Server/SUSE Linux Enterprise Desktop et en version gratuite openSUSE.

– Slackware, disponible pour la première fois le 16 juillet 1993, est une distribution créée par Patrick Volkerding. Elle se veut proche de la philosophie Unix en mettant en avant la stabilité de son écosystème. L'idée est de fournir une solution stable (les paquets ne sont pas remplacés s'ils ne sont pas testés et vérifiés). Le développement est contrôlé, il n'y a ni contributeur ni dépôt.

– Arch Linux, distribution initiée par Judd Vinet dès 2002, elle sera proposée en 2006 pour la première fois. Elle s'appuie le principe des *rolling release*, ce qui signifie qu'il n'y a pas de version majeure ; il faut donc continuellement être vigilant.

Cette liste va naturellement exclure des distributions intéressantes qui, bien qu'elles soient moins répandues, demeurent d'intérêt au regard de certaines particularités. **NixOS** en est un bon exemple avec son gestionnaire de paquets et son système de configuration déclaratif unique.

Les **études de cas** et autres exemples présentés dans cet ouvrage s'appuient sur la distribution **Ubuntu de la branche Debian**. Ce choix est **subjectif** et il est bien entendu tout à fait possible d'utiliser aussi bien une distribution Slackware qu'une distribution Fedora pour répondre à ce besoin. Notez que certaines commandes sont propres à la distribution retenue, par exemple le gestionnaire de paquets.

Les distributions GNU/Linux trouvent leur place dans l'investigation numérique en tant que plates-formes dédiées à l'analyse et comme environnement sur lequel un incident de sécurité peut se dérouler. Il existe des distributions **dédiées** ou tout du moins possédant déjà de nombreux outils destinés à l'analyse, tels que :

– SANS SIFT Workstation ;
– Parrot OS ;
– CAINE ;
– PALADIN EDGE ;
– TSURUGI ;
– CSI Linux.

Un point intéressant est la possibilité de concevoir sa propre distribution. Le chapitre Boîte à outils présentera quelques technologies telles que **Packer**, **Vagrant** ou bien **Terraform** permettant à tout à chacun de créer sa distribution, son infrastructure et de procéder à son déploiement.

3. Présentation de GNU/Linux

3.1 Noyau

Linux est un noyau dit **monolithique** c'est-à-dire qu'il fonctionne comme une seule entité et est donc présent sous la forme d'un seul exécutable. Principal composant du système d'exploitation, il sert d'interface entre le matériel et les processus utilisateur. Ses fonctions sont les suivantes :

– gestion de la mémoire ;
– gestion des processus ;
– pilotes de périphériques ;
– système de fichiers ;
– réseau ;
– gestion des interruptions et ordonnancement ;
– mécanisme de sécurité.

Ci-dessous les fichiers correspondant au noyau présents dans le répertoire /boot :

– **initrd-xxx.img** (*initial RAM disk*) : il s'agit d'une image disque temporaire correspondant à un système de fichiers minimal utilisée lors du processus de démarrage.
– **System.map-xxx** : contient une correspondance entre les adresses mémoire et les symboles du noyau. Les symboles sont utilisés pour représenter des fonctions et variables. Il s'agit du fichier le plus intéressant dans le cadre de l'investigation numérique notamment, car les profils Volatility s'appuient dessus.
– **Vmlinuz-xxx** : version compressée du noyau. Il est chargé par le chargeur d'amorçage (GRUB) dans la mémoire et est accompagné par le fichier **initramfs** qui contient les outils et pilotes nécessaires pour monter le système de fichiers racine.

3.2 Processus

Cette partie étant **commune à Microsoft Windows**, pour éviter toute redondance, nous ne détaillerons que certaines particularités et invitons le lecteur à compléter avec le chapitre Les concepts fondamentaux de Microsoft Windows.

Les systèmes Linux s'appuient sur le format ELF (*Executable and Linkable Format*) (figure 1) pour représenter les **exécutables**, les fichiers objets, les **librairies partagées** et les ***core dumps***. Les indications **Elf64_*** font référence aux différentes structures composant ce format et sont consultables ici : http://ftp.rpm.org/api/4.4.2.2/structElf64__Ehdr.html.

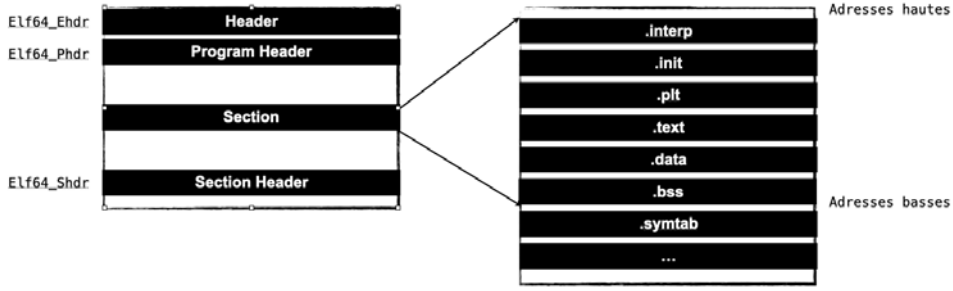

Figure 1 : structure d'un fichier ELF

L'exécutable, une fois lancé, devient un processus avec les valeurs du compteur ordinal (ou pointeur d'instruction), des registres et des variables et se trouve mappé en mémoire (figure 2). Tous les éléments définissant l'environnement d'exécution d'un processus sont nommés **contexte**. Cela regroupe l'identifiant du processus (PID), l'identifiant du processus père (PPID), l'identifiant du groupe associé au processus (PGID), le répertoire de travail courant, son espace d'adressage virtuel et la communication inter-processus.

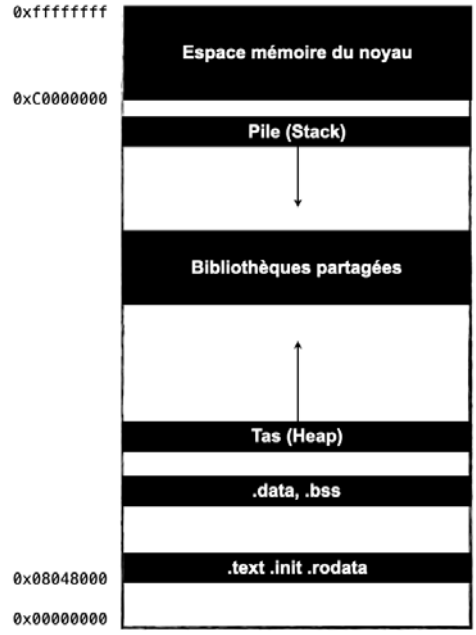

Figure 2 : représentation d'un processus ELF mappé en mémoire

L'espace d'adresses virtuelles se compose de segments de données (figure 2), d'une pile d'exécution et de segments de code. L'espace occupé par ces différents segments pour un programme donné est accessible via la commande `size` :

– .text : contient le code compilé du programme exécuté. Cela correspond au segment de code.

– .data : contient les variables globales et statiques initialisées. Cela correspond au segment de données.

– .bss : contient les variables non initialisées. Cela correspond au segment de données.

– pile/stack : contient de nombreuses variables (adresse de retour, variables locales, données liées aux fonctions). Tout comme Windows, la pile est de type LIFO (*Last In First Out*), ce qui indique que la dernière donnée est la première à être retirée. Cela correspond à la pile d'exécution.

Le noyau gère **une liste des processus en cours d'exécution** ou ***task list***. **Chaque processus possède son bloc de contrôle de processus** (PCB), il contient des données utilisées par le système d'exploitation pour gérer le processus. Cela inclut le PID, l'état du processus, la priorité du processus, etc. Cette structure joue un rôle crucial lors des changements de **contexte**, c'est-à-dire lorsque le système d'exploitation fait passer un process d'un état à un autre.

Chaque processus s'appuie sur la structure `task_structs` définie dans le fichier sched.h :
https://github.com/torvalds/linux/blob/master/include/linux/sched.h

Cette structure contient de nombreux éléments relatifs au processus tels que son état, ses registres et son espace d'adressage. Pour faire le parallèle, il s'agit de la structure équivalente à celle utilisée pour le format PE à savoir _EPROCESS.

La ligne suivante issue du fichier sched.h précise que tasks est de type `list_head`. La structure `list_head` présente dans `linux/types.h` permet de définir une **liste chaînée**. Cela permet donc au processus d'avoir accès à tous les processus actifs juste en parcourant cette liste.

```
struct list_head tasks;
```

La structure `list-head` est composée d'un pointeur vers l'élément suivant (`next`) et d'un autre vers l'élément précédent (`prev`). Cette liste doublement chaînée est au cœur de la capacité du système à parcourir la liste des processus.

```
struct list_head {
   struct list_head *next, *prev;
};
```

Le parcours des processus est facilité par le pointeur `init_task` qui pointe le **début de cette liste**. L'équivalent pour la liste chaînée contenant les _EPROCESS est `PsActiveProcessHead`.

Il est intéressant de se demander à quoi cela peut servir d'avoir accès à cette liste chaînée.

Tout comme la structure EPROCESS, Volatility s'appuie sur la structure `task_struck` dans le but d'afficher tous les processus ainsi que les éléments disponibles comme le nom, le PID ou bien le chemin. C'est le principe utilisé par le module Volatility `linux_pstree`.

À quoi ressemble un processus et quelles sont les informations à notre disposition ?

De nombreuses commandes permettant de lister les processus (`top`, `htop`, `ps`...). Nous utiliserons principalement `ps` qui, grâce a ses nombreux arguments, est un outil tout adapté pour obtenir les informations souhaitées.

Commençons par lister les processus avec `ps -ax` (`a` inclut tous les processus des autres utilisateurs, `x` affiche les processus n'ayant pas de terminal de contrôle).

```
PID TTY STAT TIME COMMAND
2193 tty1 Sl+ 0:00 /usr/lib/gnome-settings-daemon/gsd-xsettings
2196 tty1 Sl+ 0:00 /usr/lib/gnome-settings-daemon/gsd-a11y-settings
2197 tty1 Sl+ 0:00 /usr/lib/gnome-settings-daemon/gsd-clipboard
2199 tty1 Sl+ 0:04 /usr/lib/gnome-settings-daemon/gsd-color
```

La première colonne PID correspond au numéro d'identification du processus. De la même manière, le PPID est le PID du processus Parent, il correspond à celui qui a lancé le processus PID.

La commande `ps -ajx` ajoute le `-j` qui sert à afficher les PGID (*Process Group Identifier*), SID (*Session ID*) et PPID (*Parent Process ID*). À noter qu'il y a un processus particulier qui aura toujours le PID à 1, il s'agit de l'« init process » qui correspond au premier processus lancé lors du démarrage du système (pour les distributions utilisant le gestionnaire d'initialisation SysVInit). Dans le cas présent, la distribution GNU/Linux utilise le gestionnaire d'initialisation systemd, donc le premier processus est systemd.

```
PPID PID PGID SID TTY TPGID STAT UID TIME COMMAND
0 1 1 1 ? -1 Ss 0 0:20 /lib/systemd/systemd --system --deserial
0 2 0 0 ? -1 S 0 0:00 [kthreadd]
2 3 0 0 ? -1 I< 0 0:00 [rcu_gp]
2 4 0 0 ? -1 I< 0 0:00 [rcu_par_gp]
```