Le format ELF(Executable and Linking Format)

le format ELF(base)

Format ELF

ELF(Executable and Linking Format) est un format de fichier standard dans les os de type unix-like(linux en général) pour les fichiers exécutables, les bibliothèques partagées et les fichiers Objets.

Structure d'un Fichier ELF (à connaitre pour ce cours)

un fichier **ELF** contient plusieurs sections et segments qui servent à gérer les différentes parties d'un programme en mémoire

1. **En-tête ELF(ELF header)** :

situé au début du fichier, il contient des informations sur le type du fichier, l'architecture, les adresses d'entrée, et les offsets pour les autres sections du fichier.

Pour afficher l'en-tête d'un ELF on peut utiliser la commande < readelf -h nom_du_binaire>

exemples:

ici nous avons les l'en-tête de la **libc** dont le type est DYN pour dynamique est un **fichier objet partagé**

et notre deuxième exécutable est un programme c qui est de type **fichier exécutable**

Sur ce screenshot, nous pouvons voir différentes informations sur ce binaire, comme son type, le nombre d'en-têtes, le point d'entrée, etc.

Program Header Table : Il contient les informations nécessaires au chargeur du système pour mapper le fichier ELF en mémoire, en indiquant quelles parties du fichier correspondent aux segments de mémoire (stack, heap, section,etc.).

1. **Sections** : ce sont des parties d'un ELF qui ont un rôle bien défini :

.text : Contient le code exécutable (instructions du programme).

.data : Contient les variables(locales et globales) initialisées du programme .

.bss : Contient les variables(locales et globales) non initialisées.

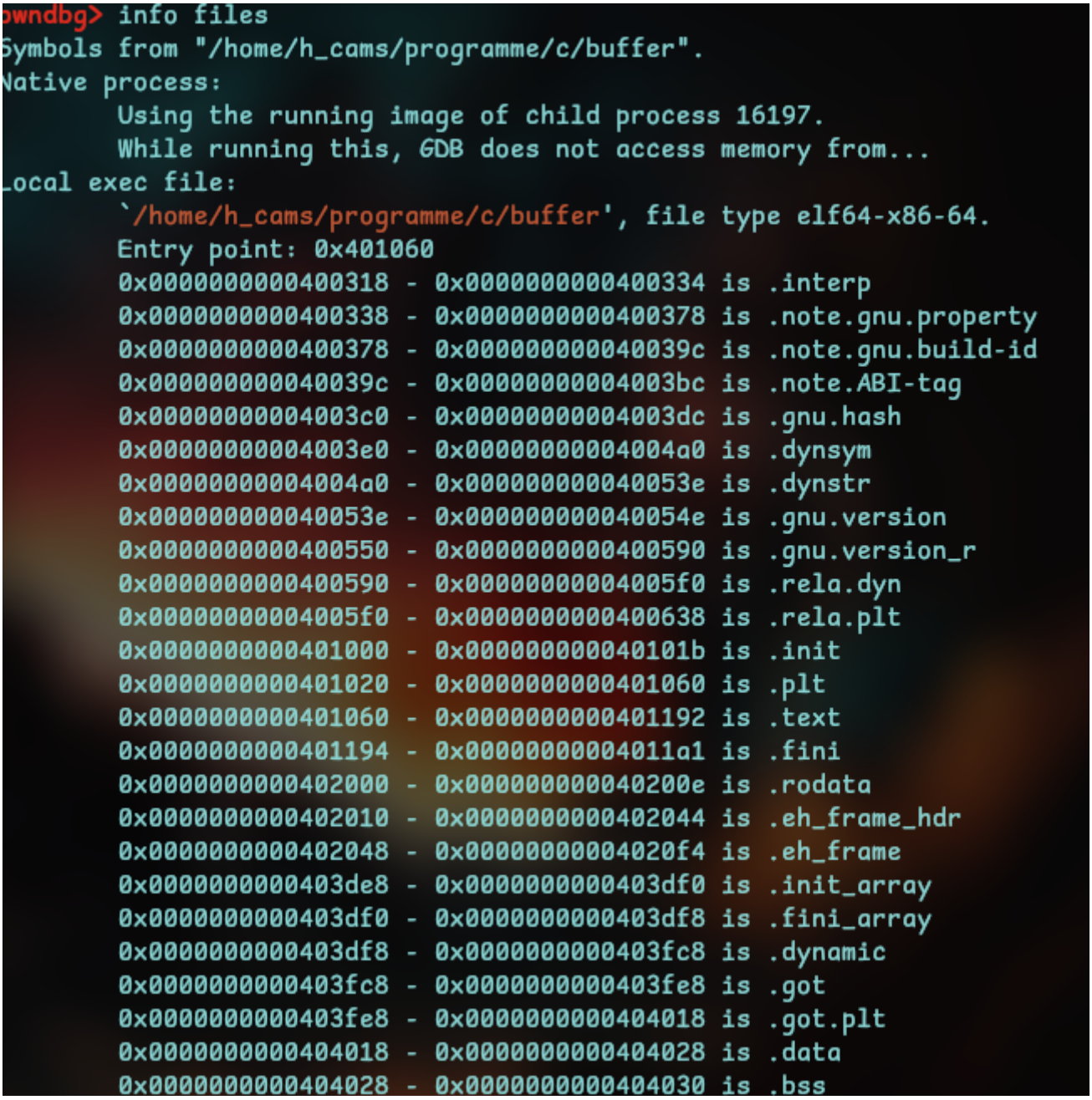
.rodata : Contient les données en lecture seule, comme les chaînes de caractères constantes. ex: les chaînes que vous écrivez par exemple dans vos fonctions d'affichage de messages sur la sortie standard

.symtab et **.dynsym** : Tables de symboles utilisées pour identifier les fonctions et les variables dans le programme.

Un symbole est une chaîne de caractères dans un fichier binaire, associée à une adresse mémoire, une variable, ou une fonction.

.rel et .rela : Sections utilisées pour le relogement (relocation) des adresses lors du chargement du programme ou de l'utilisation de bibliothèques partagées.

Vous pouvez le voir dans cet exemple, certaines sections du binaire C, avec la sortie de GDB ci-dessous :



Les types de fichiers ELF (liste non exhaustive)

- **Executable** : Un fichier ELF pouvant être directement exécuté par le système d’exploitation.
- **Shared Object** : Une bibliothèque partagée, utilisée à la fois au moment de la liaison dynamique et de l’exécution pour accéder à certaines fonctions qui ne se trouvent pas dans notre programme, comme "*printf*" en C par exemple. Ce sont généralement des fichiers ayant une extension .so, comme la l*ibc*

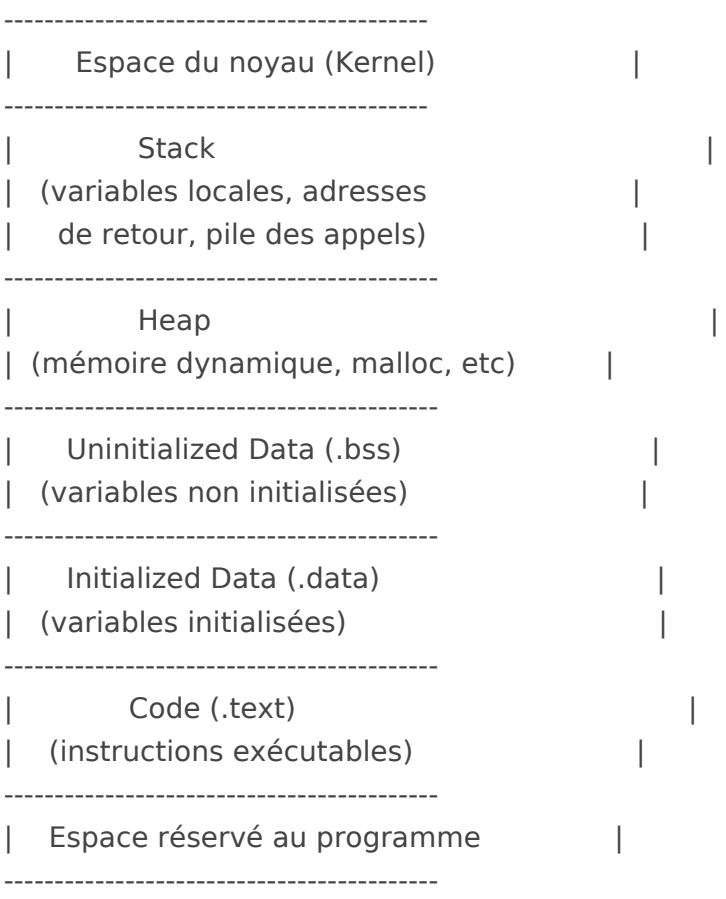
Chargement et exécution d’un ELF

Lorsqu’un fichier ELF est exécuté :

1. Le **chargeur** du système lit le **program header** pour charger les segments appropriés en mémoire.
2. Il alloue de la mémoire pour la pile (stack) et le tas (heap).
3. Il place les segments de code (.text) et de données (.data, .bss) dans les sections de la mémoire correspondantes.
4. Enfin, il commence l’exécution en sautant à l’adresse spécifiée dans le **point d'entrée** du fichier ELF (généralement défini dans le **ELF Heade**r).

Mappage d'un **ELF** en mémoire :

Représentation de la mémoire pour un programme en cours d'exécution :



Espace kernel : gère les ressources, assure la gestion des erreurs, exécute les appels système, etc.

Stack : est une structure de données qui contient les variables locales et les adresses de retour des fonctions. Elle croît vers le bas.

Heap: est une Zone de mémoire utilisée pour les allocations dynamiques (malloc, calloc). Elle croît vers le haut, en partant du bas

> ce n'est pas vraiment la structure d'un elf vu qu'il contient d'autres choses donc juste le minimum !

Les Registres 86_64

Les registres sont des espaces mémoire situés dans le processeur, dont le rôle est de permettre un accès rapide aux données. Dans un ordinateur, ils constituent la mémoire la plus petite, mais aussi la plus rapide.

Lorsque notre ordinateur effectue des calculs, les données sont stockées dans les registres du processeur. Par exemple, lorsqu'on effectue une addition, le processeur utilise des registres pour stocker temporairement les valeurs des nombres à additionner.

```
ma_fonction addition(a:entier , b:entier){
  retourner a+b
}
```

dans l'exemple de la fonction ci-dessus, le processeur stocke la valeur de a dans un registre supposons Z et pour faire l'addition il ajoute b à Z qui contient déjà la valeur de a donc Z = Z+b

Types de registres :

Ils existe plusieurs types de registres en x64. Nous avons :

Les registres généraux : globalement se sont des registres qui sont utiliser pour contenir des données (variables, adresses mémoire, ...) :

.RAX (accumulateur) : Principalement utilisé pour effectuer des calculs arithmétiques et logiques. Il est également utilisé pour stocker la valeur de retour d’une fonction.

.RBX (base register) : N'a pas de rôle spécifique, mais peut être utilisé pour stocker des valeurs temporaires ou manipuler des indices de tableaux.

.RCX (compteur) : Généralement utilisé comme compteur de boucle ou pour certaines instructions répétitives (rep, loop).

.RDX (data register) : Utilisé pour stocker la partie haute du résultat lors d’opérations nécessitant plus de 64 bits (par exemple, une multiplication). Il est souvent combiné avec RAX dans ce cas.

.R8 à R15 : Registres généralistes comme RBX, utilisés pour stocker des valeurs temporaires et pour passer des arguments aux fonctions en x86_64 (R8 et R9 étant les 5^{e} et 6^{e} arguments).

.RSI (Source Index) & RDI (Destination Index) : Utilisés pour le passage des arguments aux fonctions (RSI pour le deuxième argument, RDI pour le premier argument). Peuvent être utiliser assi comme RBX

.RFLAGS : est un registre qui contient des valeurs indiquant létat du processeur lors de l'exécution d'un programme.

> Pour le Pwn voici les registres qui sont importants :

> Les registres de Pile:
> RBP (base pointer)
> Le registre RBP est utilisé pour sauvegarder l'adresse du cadre de pile(stack frame) dans une fonction. Le cadre de pile représente l'état de la pile au moment de l'appel d'une fonction.
> Grâce à RBP, il est possible de localiser les variables locales et d'y accéder facilement, car il définit la base de la pile pour cette fonction particulière. Il se situe juste avant les variables local d'une fonction et après le RIP (instruction pointeur)
> RSP (stack pointer)
> Le registre RSP pointe vers le sommet de la pile, c'est-à-dire l'endroit où la prochaine valeur sera empilée ou dépilée. Lors de l'exécution d'une fonction, il change dynamiquement, car chaque nouvelle valeur ajoutée à la pile fait que RSP diminue (puisque la pile croît vers les adresses basses).

> Le registre RIP contient l'adresse de la prochaine instruction à exécuter. Il pointe vers l'emplacement mémoire où le processeur doit se rendre pour récupérer la prochaine instruction à exécuter dans le programme.
> C'est un registre clé dans le contrôle du flux d'exécution, car il permet au processeur de savoir où il en est dans l'exécution du programme et quelle sera l'instruction suivante à exécuter.

RIP et attaques :

Dans le contexte des attaques comme le buffer overflow, le RIP joue un rôle crucial. Lorsqu'un payload malveillant est écrit dans la pile, il peut modifier le RIP pour rediriger l'exécution du programme vers un code malveillant. Cela se fait souvent en écrivant l'adresse qui contient le debut de notre payload dans le RIP, amenant le processeur à exécuter ce code malveillant.
Le but de ces attaques est souvent de faire en sorte que le RIP pointe vers une adresse sur laquelle on a le contrôle, afin d'exécuter une série d'instructions bien précises (selon notre objectif et les

morceaux de code disponibles dans le binaire) par le processeur.

Analyse d'un Binaire

Outils pour analyse du binaire

Fonctions vulnérables de la libc

Ici nous verrons quelques fonctions vulnérables de la libc aux attaques du type buffer overflow.

Exploiter un Buffer Overflow

Exploitatoin buffer overflow