Saaruhan Sellappah
ING 3 CS B

Rapport Projet Shell Code

Sujet : Infecteur ELF de type (ptnote - > ptload)

CY-Tech ing 3 Cyber-Sécurité 2024/2025

Sommaire :

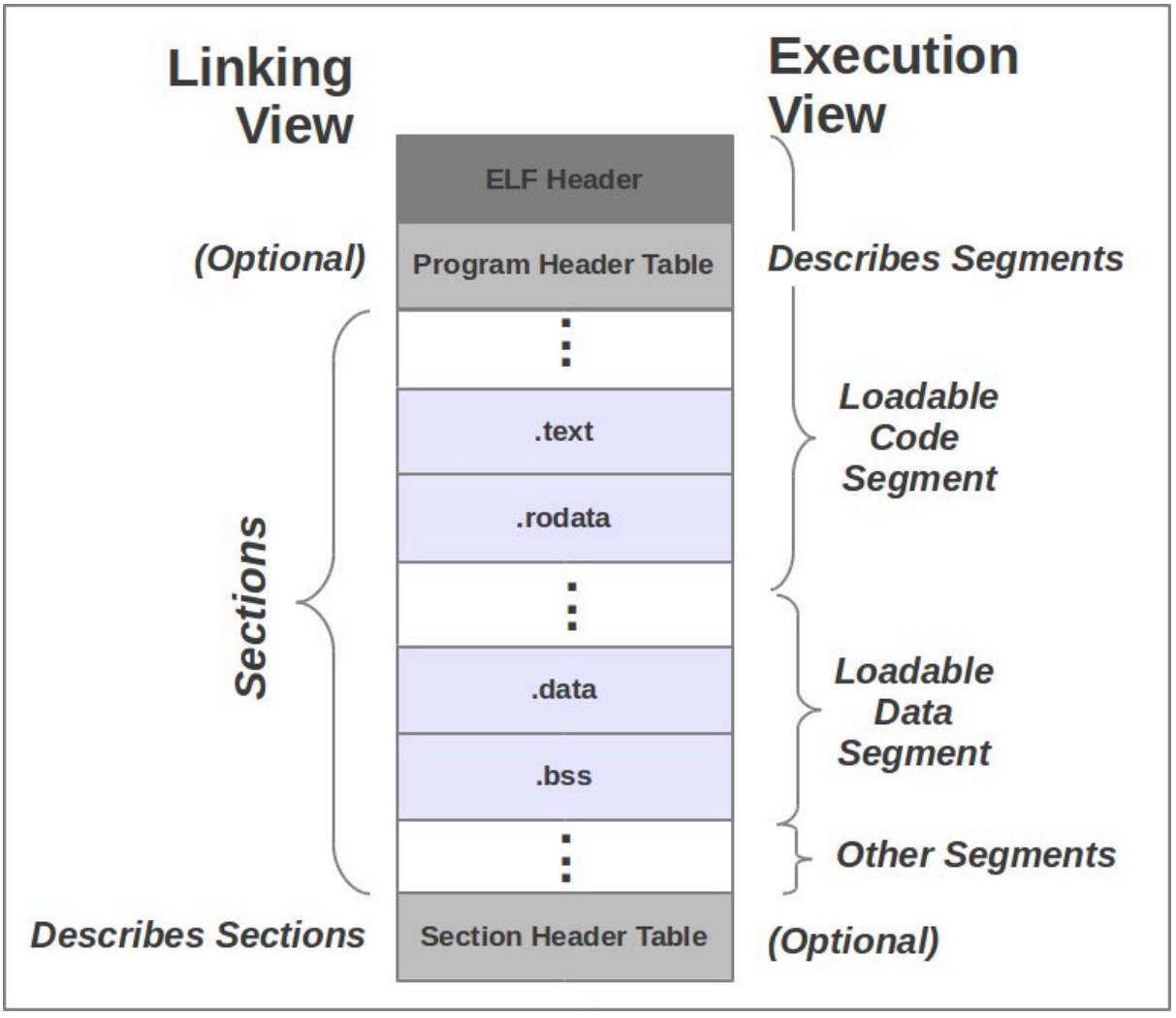
Introduction :

L’objectif du projet est de réaliser un infecteur ELF de type PT_NOTE -> PT_LOAD. Plus précisément ce projet vise à modifier le premier segment PT_NOTE d’un ELF en un PT_LOAD exécutable qui pointe vers la fin du fichier contenant le shellcode. Cela nécessite de comprendre en profondeur le fonctionnement d’un fichier ELF ainsi que sa structure. Ainsi, l’objectif va être d’ajouter du code à la fin du fichier qui sera exécuté lors de l'exécution du binaire. Cette modification devra être réalisée en respectant les alignements et les contraintes imposées par l’architecture ELF.

Présentation de notions clés :

Un fichier ELF (Excutable LInked Format) est un format standard utilisé pour les fichiers exécutables, les bibliothèques partagées, et les fichiers de vidage mémoire (core dumps) sur les systèmes d'exploitation de type Unix, comme Linux. Développé pour succéder à d'anciens formats tels que a.out, ELF est aujourd'hui l'un des formats les plus répandus pour les binaires dans ces environnements.

L'une des caractéristiques principales d'un ELF est qu'il est composé de sections et segments. Les sections contiennent des données qui sont nécessaires afin de générer ou exécuter les fichiers exécutables. Les segments sont des unités logiques utilisées par le système d'exploitation afin de charger en mémoire les parties importantes du fichier.



Les composants principaux d'un ELF sont les suivants :

- Un entête ELF qui contient des informations globales sur le fichier, comme son entry point, son architectre cible ainsi que son type.
- La table de sections qui regroupe des groupes logiques du fichier.
- La table des segments : Cette partie permet de mapper des parties spécifiques du fichier ELF en mémoire.

La table des segments contient les segments qui nous intéressent, à savoir le segment PT_NOTE et le segment PT_LOAD.

Le segment PT_NOTE est un segment qui contient des auxilliaires ou des métadonnées pour le fichier ELF. Il est utilisé pour des annotations, des identifiants ou des informations destinées à des outils de débogage ou des processus spécifiques. A contrario le segment PT_LOAD définit les portions du fichier ELF qui doivent être chargés en mémoire afin de permettre l'exécution du programme.

Les différents segments possèdent des champs qui permettent de les définir :

```
struct Elf64_Phdr {
    Elf64_Word p_type;
    Elf64_Word p_flags;
    Elf64_Off p_offset;
    Elf64_Addr p_vaddr;
    Elf64_Addr p_paddr;
    Elf64_Xword p_filesz;
    Elf64_Xword p_memsz;
    Elf64_Xword p_align;
};
```

Chaque champs est définis par un octet ou plusieurs octets dont voici les tailles pour un elf 64 bits :

Nom du champ	Taille (64 bits)
p_type	4 octets
p_flags	8 octets
p_offset	8 octets
p_vaddr	8 octets
p_filesz	8 octets
p_memsz	8 octets

Ces champs sont importants car il s'agit des champs principaux que j'ai modifés dans mon projet nécessaire ensuite à l'infection ptnote -> ptload.

Points réussis :

Tout d'abord, dans le projet je travaille avec un fichier binaire résultant de la compilation d'un fichier helloworld.c. Voici les informations le concernant (utilisation de la commande readelf -l nomdufichierbinaire)

```
  -v --version          Afficher le numéro de version de readelf
saaru@saaru-Latitude-5520:~/Bureau/ProjetShellCode$ readelf -l hello

Type de fichier ELF est DYN (fichier exécutable indépendant de la position)
Point d'entrée 0x1060
Il y a 13 en-têtes de programme, débutant à l'adresse de décalage 64

En-têtes de programme :
  Type           Décalage           Adr.virt           Adr.phys.
                 Taille fichier     Taille mémoire      Fanion Alignement
  PHDR           0x0000000000000040 0x0000000000000040 0x0000000000000040
                 0x00000000000002d8 0x00000000000002d8  R      0x8
  INTERP         0x0000000000000318 0x0000000000000318 0x0000000000000318
                 0x000000000000001c 0x000000000000001c  R      0x1
      [Réquisition de l'interpréteur de programme: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000628 0x0000000000000628  R      0x1000
  LOAD           0x0000000000001000 0x0000000000001000 0x0000000000001000
                 0x0000000000000189 0x0000000000000189  R E    0x1000
  LOAD           0x0000000000002000 0x0000000000002000 0x0000000000002000
                 0x000000000000011c 0x000000000000011c  R      0x1000
  LOAD           0x0000000000002db8 0x0000000000003db8 0x0000000000003db8
                 0x0000000000000258 0x0000000000000260  RW     0x1000
  DYNAMIC        0x0000000000002dc8 0x0000000000003dc8 0x0000000000003dc8
                 0x00000000000001f0 0x00000000000001f0  RW     0x8
  NOTE           0x0000000000000338 0x0000000000000338 0x0000000000000338
                 0x0000000000000030 0x0000000000000030  R      0x8
  NOTE           0x0000000000000368 0x0000000000000368 0x0000000000000368
                 0x0000000000000044 0x0000000000000044  R      0x4
  GNU_PROPERTY   0x0000000000000338 0x0000000000000338 0x0000000000000338
                 0x0000000000000030 0x0000000000000030  R      0x8
  GNU_EH_FRAME   0x0000000000002014 0x0000000000002014 0x0000000000002014
                 0x000000000000003c 0x000000000000003c  R      0x4
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     0x10
  GNU_RELRO      0x0000000000002db8 0x0000000000003db8 0x0000000000003db8
                 0x0000000000000248 0x0000000000000248  R      0x1

 Correspondance section/segment :
  Sections de segment...
```

Les champs importants dans le fichier elf sont les suivants : le point d'entrée qui vaut ici 0x1060, il s'agit de l'adresse virtuelle du point d'entrée du programme. Ensuite les champs type correspondent aux types des segments présents dans la table de segments qui se trouve à l'adresse de décalage 64.

L’objectif premier était de déterminer si le fichier binaire hello était bien un ELF afin que lors de la recherche du premier PT_NOTE, il n’y ait pas de litige.

Pour cela, j’ai comparé les 4 premiers octet du fichier (en utilisant l’appel système read) avec ceux d’un ELF.

```
+ elfMagic db 0x7F, 'E', 'L', 'F' ; il s'agit de la signature elf
+
+
```

S’il ne s’agit pas d’un fichier ELF alors un message indiquant qu’il ne s’agit pas d’un fichier ELF est affiché.

Ensuite, j’ai tenté d’extraire l’octet du premier p_type du premier PT_NOTE. Afin d’effectuer cette opération , j’ai tenté de lire le header et récupérer l’entête de programme, puis à partir de cela récupérer l’offset du p_type mais je suis resté bloqué dessus plusieurs semaines (avant mon premier push, voir le fichier header.asm que j’avais écris et modifié plus tard afin de tenter de refaire fonctionner)

N’ayant pas réussi cette étape l’objectif était de trouver l’octet contenant le p_type du premier segment afin de le modifier à tout prix. J’ai donc changé manuellement afin de pouvoir changer le PT_NOTE en PT_LOAD en utilisant patchelf. Cette étape ayant réussi, j’ai à l’aide de la commande “ cmp binaire1 binaire2” tenté de voir quel octet était modifié, me donnant la valeur 456, avec binaire1 le fichier originel et binaire2 une copie du fichier elf.

Voici la capture d’écran après les modifications effectuées.

```
saaru@saaru-Latitude-5520:~/Bureau/ProjetShellCode$ readelf -l hello

Type de fichier ELF est DYN (fichier exécutable indépendant de la position)
Point d'entrée 0x338
Il y a 13 en-têtes de programme, débutant à l'adresse de décalage 64

En-têtes de programme :
  Type           Décalage           Adr.virt           Adr.phys.
                 Taille fichier     Taille mémoire      Fanion Alignement
  PHDR           0x0000000000000040 0x0000000000000040 0x0000000000000040
                 0x00000000000002d8 0x00000000000002d8  R      0x8
  INTERP         0x0000000000000318 0x0000000000000318 0x0000000000000318
                 0x000000000000001c 0x000000000000001c  R      0x1
      [Réquisition de l'interpréteur de programme: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000628 0x0000000000000628  R      0x1000
  LOAD           0x0000000000001000 0x0000000000001000 0x0000000000001000
                 0x0000000000000189 0x0000000000000189  R E    0x1000
  LOAD           0x0000000000002000 0x0000000000002000 0x0000000000002000
                 0x000000000000011c 0x000000000000011c  R      0x1000
  LOAD           0x0000000000002db8 0x0000000000003db8 0x0000000000003db8
                 0x0000000000000258 0x0000000000000260  RW     0x1000
  DYNAMIC        0x0000000000002dc8 0x0000000000003dc8 0x0000000000003dc8
                 0x00000000000001f0 0x00000000000001f0  RW     0x8
  LOAD           0x0000000000000338 0x0000000000000338 0x0000000000000338
                 0x0000000000000030 0x0000000000000030    E    0x8
  NOTE           0x0000000000000368 0x0000000000000368 0x0000000000000368
                 0x0000000000000044 0x0000000000000044  R      0x4
  GNU_PROPERTY   0x0000000000000338 0x0000000000000338 0x0000000000000338
                 0x0000000000000030 0x0000000000000030  R      0x8
  GNU_EH_FRAME   0x0000000000002014 0x0000000000002014 0x0000000000002014
                 0x000000000000003c 0x000000000000003c  R      0x4
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     0x10
  GNU_RELRO      0x0000000000002db8 0x0000000000003db8 0x0000000000003db8
                 0x0000000000000248 0x0000000000000248  R      0x1
```

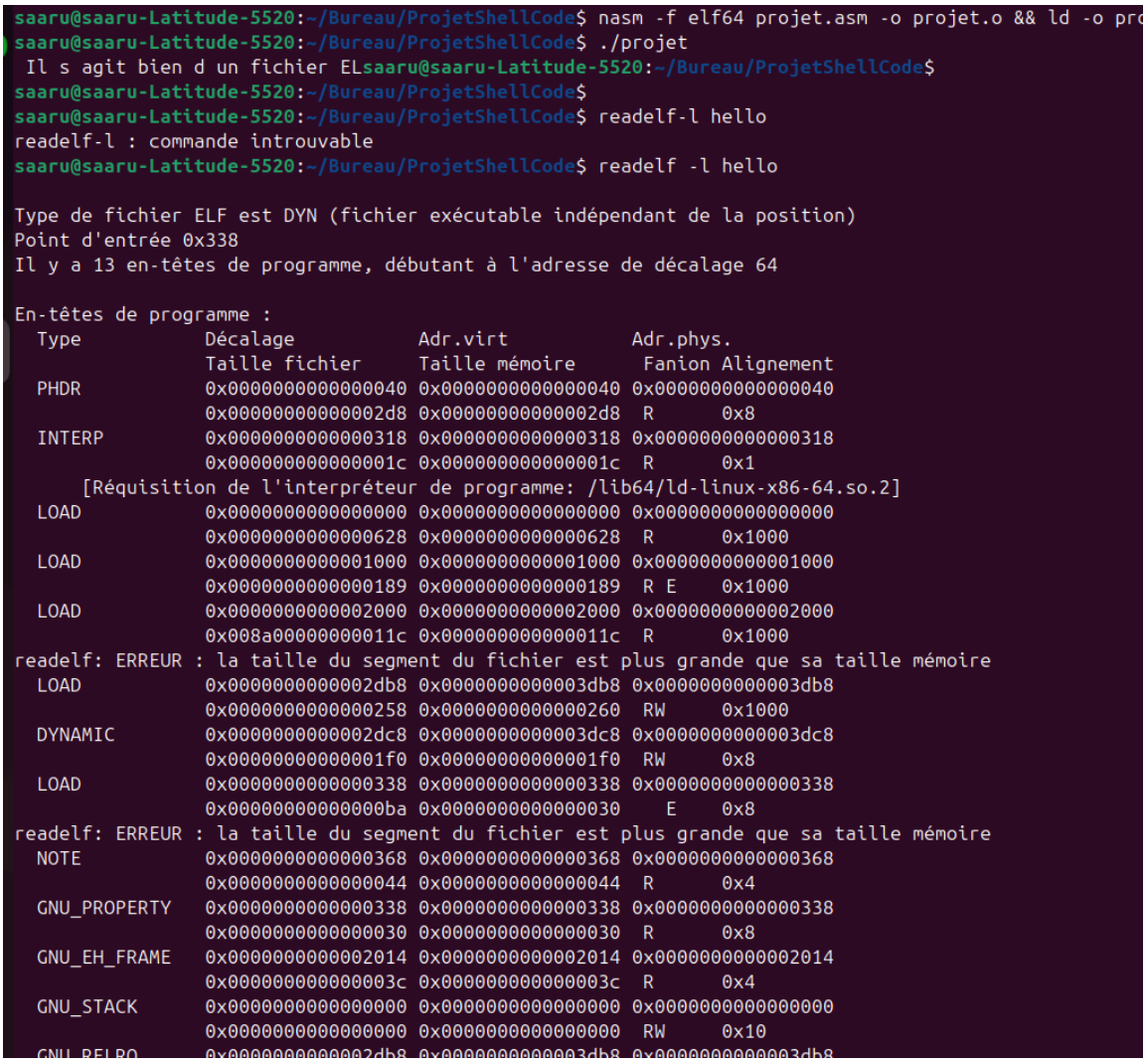
On voit que le point d'entrée est 0x338 qui correspond à l'adresse virtuelle du pt note modifié. Une autre modification effectuée est l'ajout de la taille du shellcode dans p_memsz et p_filesz (voir fichier octetmodif.asm) afin d'acueuilir le shellcode

Ensuite j'ai réussi à injecter le shellcode à la fin du fichier (voir programme append.asm qui prend un fichier binaire hello et ajoute un shellcode à la fin du fichier)

Points de blocages rencontrés.

Lors de la réalisation du projet, j'ai rencontré 2 points de blocages principales. Le premier, le plus chronophage, a été de parcourir les segments de la table de segments et de déterminer p_offset pour le premier segment. Cela m'a pris environ 3 semaines avant le premier push. N'ayant pas pu régler le problème j'ai donc opté pour une recherche "manuelle" du segment PT_NOTE afin de déterminer l'octet à modifier. A partir de cela, j'ai pu modifier les champs du segment PT_NOTE modifié en effectuat un décalage.

Le 2ème problème a été l'exécution du shellcode après ajout de la taille du shellcode dans p_memsz et pfilesz. En effet lors de l'exécution du projet une erreur est survenue dans la memoire du segment PT_NOTE.



Conclusion :

Ce projet avait pour objectif de transformer le segment PT_NOTE en PT_LOAD afin d'infecter un fichier ELF et d'exécuter un shellcode ajouté en fin de fichier. Malgré certaines difficultés techniques, telles que la récupération automatique des offsets des segments ou l'identification précise du premier segment PT_NOTE, des avancées ont été réalisées. En modifiant manuellement les champs p_type, p_memsz, et p_filesz, j'ai réussi à transformer le PT_NOTE en un segment PT_LOAD exécutable afin de trouver l'octet à modifier tout en accueillant le shellcode avec le fichier append.asm.. Cependant, des erreurs de contraintes de mémoire ont mis en lumière des points à approfondir, notamment dans la gestion des alignements et des sections ELF. Ce projet m'a permis d'acquérir une compréhension approfondie de la structure ELF, du fonctionnement des segments, ainsi que des défis liés à l'infection de fichiers binaires. Les résultats obtenus montrent une progression significative dans l'atteinte des objectifs initiaux, malgré quelques blocages qui nécessiteront des améliorations futures.

Les points de blocage auraient pu être réglé plus aisément