BENDJELIDA Rayane

Cybersecurity A

(1)https://linux-audit.com/elf-binaries-on-linux-understanding-and-analysis/

(2)https://tmpout.sh/1/2.html

(3)https://www.symbolcrash.com/2019/03/27/pt_note-to-pt_load-injection-in-elf/

(4)https://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/

(5)Learning Linux Binary Analysis by Ryan "elfmaster" O'Neill

Consigne :

- Exploit the vulnerability
- Infect a file to achieve persistence through the previously done infector
- Continue the pwned processus without crashing
- Verify the infected binary can be launched
- Search for technique that are reliable across hosts

Bonus :

- Mitigated binary (ask me for it)
- Assembly stager
- Packer
- Anti detection (except packer), like multiple behaviors

Bonus for maldev :

- Have a working infected binary (e.g you infect ls, when launches ls it launches your parasite AND the original code)
- Have a working infected binary that keeps the argument (ls -la) and does not segfault on exit
- Research and implement some concepts regarding defensive solutions escape and anti forensics methods
- Make a second version that is less noisy regarding logs (auditd/filebeats and so on
- Be able to search recusively for files to infect if there are none in the local directory

If you make a stager, you can show some of your working code by doing a video showcasing how it works and proof that it worked

First we create a new kali machine install github, pwdbg and cutter to make sure the test don't destroy anything

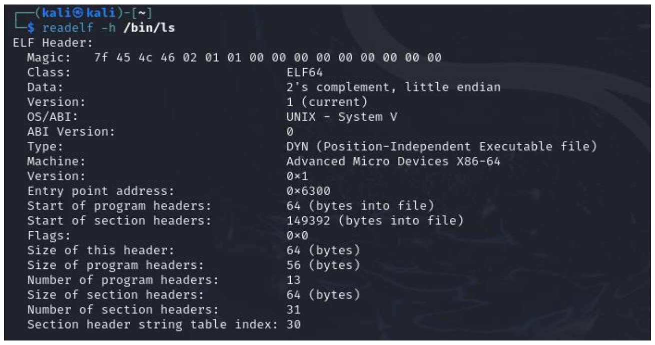
to add changes, stage them, and commit:

```
git add .        # Stage all changes
git commit -m "Your commit message"
```

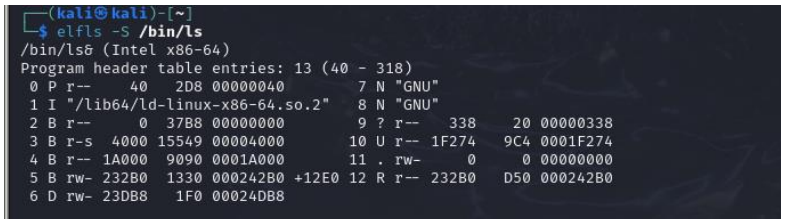
to push your changes back to GitHub:

```
git push origin main
```

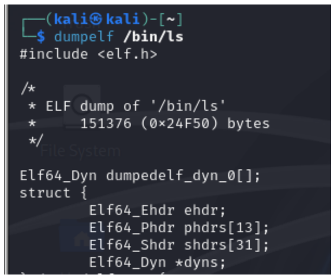
Some command for my understanding (please ignore the following screenshot)



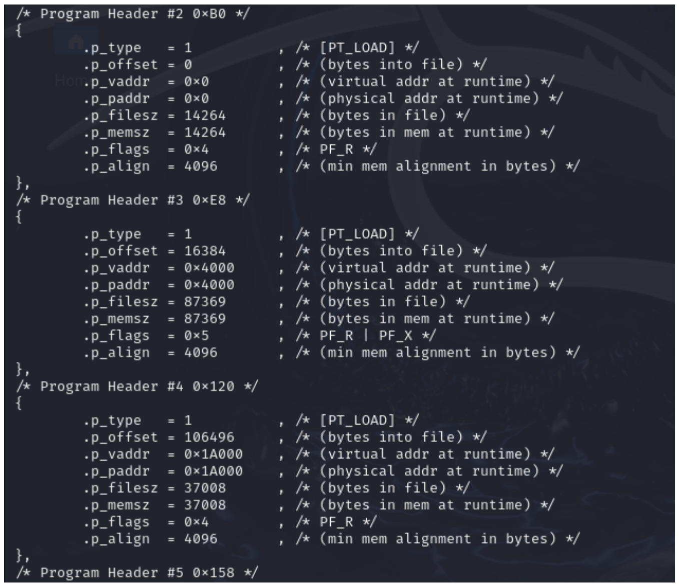
```
┌──(kali㉿kali)-[~]
└─$ readelf -h /bin/ls
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Position-Independent Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0×1
  Entry point address:               0×6300
  Start of program headers:          64 (bytes into file)
  Start of section headers:          149392 (bytes into file)
  Flags:                             0×0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         13
  Size of section headers:           64 (bytes)
  Number of section headers:         31
  Section header string table index: 30
```



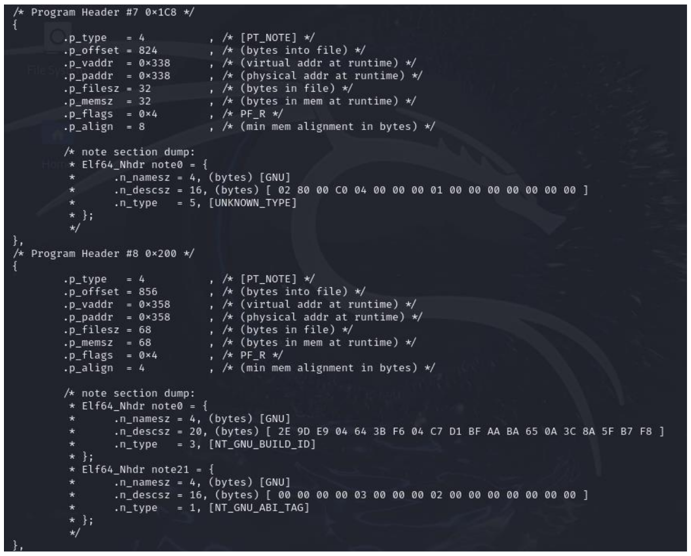
```
┌──(kali㉿kali)-[~]
└─$ elfls -S /bin/ls
/bin/ls& (Intel x86-64)
Program header table entries: 13 (40 - 318)
 0 P r--    40   2D8 00000040        7 N "GNU"
 1 I "/lib64/ld-linux-x86-64.so.2"   8 N "GNU"
 2 B r--     0  37B8 00000000        9 ? r--   338    20 00000338
 3 B r-s  4000 15549 00004000       10 U r-- 1F274   9C4 0001F274
 4 B r-- 1A000  9090 0001A000       11 . rw-     0     0 00000000
 5 B rw- 232B0  1330 000242B0 +12E0 12 R r-- 232B0   D50 000242B0
 6 D rw- 23DB8   1F0 00024DB8
```



```
┌──(kali㉿kali)-[~]
└─$ dumpelf /bin/ls
#include <elf.h>

/*
 * ELF dump of '/bin/ls'
 *     151376 (0×24F50) bytes
 */

Elf64_Dyn dumpedelf_dyn_0[];
struct {
        Elf64_Ehdr ehdr;
        Elf64_Phdr phdrs[13];
        Elf64_Shdr shdrs[31];
        Elf64_Dyn *dyns;
```

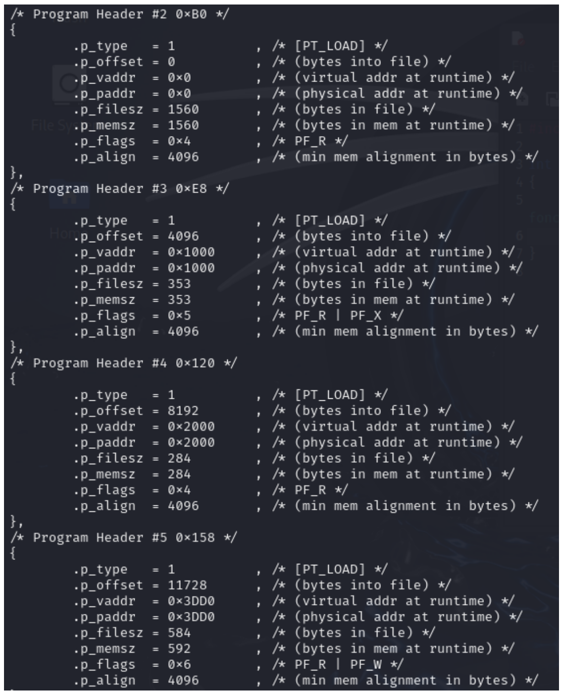


```
/* Program Header #2 0×B0 */
{
        .p_type   = 1          , /* [PT_LOAD] */
        .p_offset = 0          , /* (bytes into file) */
        .p_vaddr  = 0×0        , /* (virtual addr at runtime) */
        .p_paddr  = 0×0        , /* (physical addr at runtime) */
        .p_filesz = 14264      , /* (bytes in file) */
        .p_memsz  = 14264      , /* (bytes in mem at runtime) */
        .p_flags  = 0×4        , /* PF_R */
        .p_align  = 4096       , /* (min mem alignment in bytes) */
},
/* Program Header #3 0×E8 */
{
        .p_type   = 1          , /* [PT_LOAD] */
        .p_offset = 16384      , /* (bytes into file) */
        .p_vaddr  = 0×4000     , /* (virtual addr at runtime) */
        .p_paddr  = 0×4000     , /* (physical addr at runtime) */
        .p_filesz = 87369      , /* (bytes in file) */
        .p_memsz  = 87369      , /* (bytes in mem at runtime) */
        .p_flags  = 0×5        , /* PF_R | PF_X */
        .p_align  = 4096       , /* (min mem alignment in bytes) */
},
/* Program Header #4 0×120 */
{
        .p_type   = 1          , /* [PT_LOAD] */
        .p_offset = 106496     , /* (bytes into file) */
        .p_vaddr  = 0×1A000    , /* (virtual addr at runtime) */
        .p_paddr  = 0×1A000    , /* (physical addr at runtime) */
        .p_filesz = 37008      , /* (bytes in file) */
        .p_memsz  = 37008      , /* (bytes in mem at runtime) */
        .p_flags  = 0×4        , /* PF_R */
        .p_align  = 4096       , /* (min mem alignment in bytes) */
},
/* Program Header #5 0×158 */
```

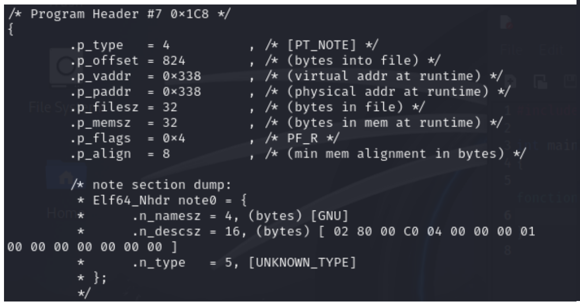


```
/* Program Header #7 0×1C8 */
{
        .p_type   = 4          , /* [PT_NOTE] */
        .p_offset = 824        , /* (bytes into file) */
        .p_vaddr  = 0×338      , /* (virtual addr at runtime) */
        .p_paddr  = 0×338      , /* (physical addr at runtime) */
        .p_filesz = 32         , /* (bytes in file) */
        .p_memsz  = 32         , /* (bytes in mem at runtime) */
        .p_flags  = 0×4        , /* PF_R */
        .p_align  = 8          , /* (min mem alignment in bytes) */

        /* note section dump:
         * Elf64_Nhdr note0 = {
         *      .n_namesz = 4, (bytes) [GNU]
         *      .n_descsz = 16, (bytes) [ 02 80 00 C0 04 00 00 00 01 00 00 00 00 00 00 00 ]
         *      .n_type   = 5, [UNKNOWN_TYPE]
         * };
         */
},
/* Program Header #8 0×200 */
{
        .p_type   = 4          , /* [PT_NOTE] */
        .p_offset = 856        , /* (bytes into file) */
        .p_vaddr  = 0×358      , /* (virtual addr at runtime) */
        .p_paddr  = 0×358      , /* (physical addr at runtime) */
        .p_filesz = 68         , /* (bytes in file) */
        .p_memsz  = 68         , /* (bytes in mem at runtime) */
        .p_flags  = 0×4        , /* PF_R */
        .p_align  = 4          , /* (min mem alignment in bytes) */

        /* note section dump:
         * Elf64_Nhdr note0 = {
         *      .n_namesz = 4, (bytes) [GNU]
         *      .n_descsz = 20, (bytes) [ 2E 9D E9 04 64 3B F6 04 C7 D1 BF AA BA 65 0A 3C 8A 5F B7 F8 ]
         *      .n_type   = 3, [NT_GNU_BUILD_ID]
         * };
         * Elf64_Nhdr note21 = {
         *      .n_namesz = 4, (bytes) [GNU]
         *      .n_descsz = 16, (bytes) [ 00 00 00 00 03 00 00 00 02 00 00 00 00 00 00 00 ]
         *      .n_type   = 1, [NT_GNU_ABI_TAG]
         * };
         */
},
```

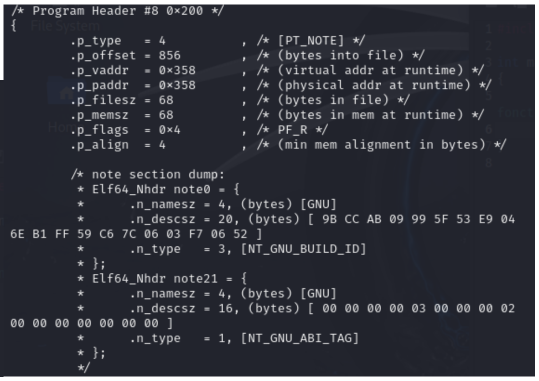
Our « target » elf that is simply a compiled c code that prints text shows the following dumpelf



```
/* Program Header #2 0×B0 */
{
        .p_type   = 1          , /* [PT_LOAD] */
        .p_offset = 0          , /* (bytes into file) */
        .p_vaddr  = 0×0        , /* (virtual addr at runtime) */
        .p_paddr  = 0×0        , /* (physical addr at runtime) */
        .p_filesz = 1560       , /* (bytes in file) */
        .p_memsz  = 1560       , /* (bytes in mem at runtime) */
        .p_flags  = 0×4        , /* PF_R */
        .p_align  = 4096       , /* (min mem alignment in bytes) */
},
/* Program Header #3 0×E8 */
{
        .p_type   = 1          , /* [PT_LOAD] */
        .p_offset = 4096       , /* (bytes into file) */
        .p_vaddr  = 0×1000     , /* (virtual addr at runtime) */
        .p_paddr  = 0×1000     , /* (physical addr at runtime) */
        .p_filesz = 353        , /* (bytes in file) */
        .p_memsz  = 353        , /* (bytes in mem at runtime) */
        .p_flags  = 0×5        , /* PF_R | PF_X */
        .p_align  = 4096       , /* (min mem alignment in bytes) */
},
/* Program Header #4 0×120 */
{
        .p_type   = 1          , /* [PT_LOAD] */
        .p_offset = 8192       , /* (bytes into file) */
        .p_vaddr  = 0×2000     , /* (virtual addr at runtime) */
        .p_paddr  = 0×2000     , /* (physical addr at runtime) */
        .p_filesz = 284        , /* (bytes in file) */
        .p_memsz  = 284        , /* (bytes in mem at runtime) */
        .p_flags  = 0×4        , /* PF_R */
        .p_align  = 4096       , /* (min mem alignment in bytes) */
},
/* Program Header #5 0×158 */
{
        .p_type   = 1          , /* [PT_LOAD] */
        .p_offset = 11728      , /* (bytes into file) */
        .p_vaddr  = 0×3DD0     , /* (virtual addr at runtime) */
        .p_paddr  = 0×3DD0     , /* (physical addr at runtime) */
        .p_filesz = 584        , /* (bytes in file) */
        .p_memsz  = 592        , /* (bytes in mem at runtime) */
        .p_flags  = 0×6        , /* PF_R | PF_W */
        .p_align  = 4096       , /* (min mem alignment in bytes) */
```



```
/* Program Header #7 0×1C8 */
{
        .p_type   = 4          , /* [PT_NOTE] */
        .p_offset = 824        , /* (bytes into file) */
        .p_vaddr  = 0×338      , /* (virtual addr at runtime) */
        .p_paddr  = 0×338      , /* (physical addr at runtime) */
        .p_filesz = 32         , /* (bytes in file) */
        .p_memsz  = 32         , /* (bytes in mem at runtime) */
        .p_flags  = 0×4        , /* PF_R */
        .p_align  = 8          , /* (min mem alignment in bytes) */

        /* note section dump:
         * Elf64_Nhdr note0 = {
         *      .n_namesz = 4, (bytes) [GNU]
         *      .n_descsz = 16, (bytes) [ 02 80 00 C0 04 00 00 00 01
00 00 00 00 00 00 00 ]
         *      .n_type   = 5, [UNKNOWN_TYPE]
         * };
         */
```



```
/* Program Header #8 0×200 */
{
        .p_type   = 4          , /* [PT_NOTE] */
        .p_offset = 856        , /* (bytes into file) */
        .p_vaddr  = 0×358      , /* (virtual addr at runtime) */
        .p_paddr  = 0×358      , /* (physical addr at runtime) */
        .p_filesz = 68         , /* (bytes in file) */
        .p_memsz  = 68         , /* (bytes in mem at runtime) */
        .p_flags  = 0×4        , /* PF_R */
        .p_align  = 4          , /* (min mem alignment in bytes) */

        /* note section dump:
         * Elf64_Nhdr note0 = {
         *      .n_namesz = 4, (bytes) [GNU]
         *      .n_descsz = 20, (bytes) [ 9B CC AB 09 99 5F 53 E9 04
6E B1 FF 59 C6 7C 06 03 F7 06 52 ]
         *      .n_type   = 3, [NT_GNU_BUILD_ID]
         * };
         * Elf64_Nhdr note21 = {
         *      .n_namesz = 4, (bytes) [GNU]
         *      .n_descsz = 16, (bytes) [ 00 00 00 00 03 00 00 00 02
00 00 00 00 00 00 00 ]
         *      .n_type   = 1, [NT_GNU_ABI_TAG]
         * };
         */
```

Now we start the infection thanks to the step to follow by (2)

I have a segmentation fault in the loop that looks for the pt_note ; fix Incrementing the offset by the right entry size

I have another problem : the program never ends it seems (gdb) like the pt_note was found but it wasn't converted as a pt_load :

0x40107f <pt_note_found>   mov   dword ptr [rax + rbx], 1   <Cannot dereference [0x28d8]>

This was fixed because the stack buffer was not pointing to the right thing (or because i did not save the right position of the stack buffer)

Another segmentation fault after changing the value of pt_note into a pt_load. When i try to calculate and Set Virtual Address. It looks like r15 is not pointing correctly wich is weird because at that point r15 is supposed to be the base address of the loaded ELF file buffer…

To fix this i retrieve `stat.st_size` with a fstat beforehand in [r15 + 48] this fix the seg fault but the pt_note hasn't been converted it's weird because i clearly pass the command in gdb and after looking at registeries i see that the problem is that it works but i don't open the file in write mode and to modify the buffer back to the file

So i just write changes back to fileread into the first arg with a read() call and modified the READ in the begining into a READ_AND_WRITE

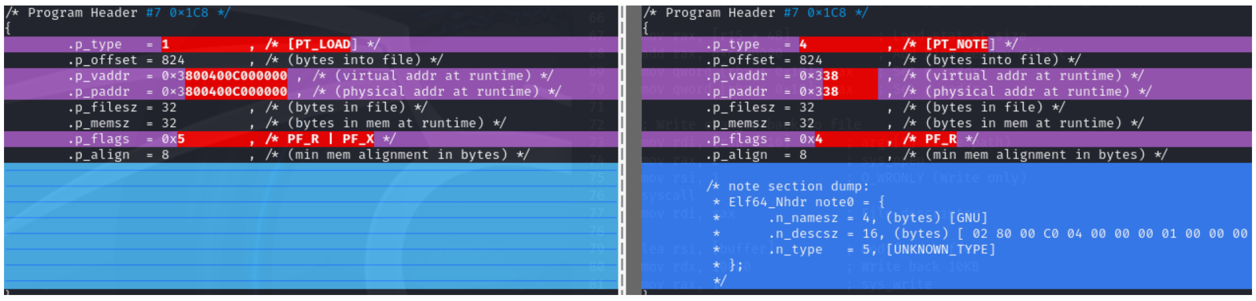
So now the pt_note is finally converted into a pt_load

Thanks to a classmate (ibrahim) i have the following command that compares two file, one before the infection and the other after

dumpelf target > a

dumpelf target > b

vimdiff a b



```
/* Program Header #7 0x1C8 */
{
        .p_type   = 1               , /* [PT_LOAD] */
        .p_offset = 824             , /* (bytes into file) */
        .p_vaddr  = 0x3800400C000000 , /* (virtual addr at runtime) */
        .p_paddr  = 0x3800400C000000 , /* (physical addr at runtime) */
        .p_filesz = 32              , /* (bytes in file) */
        .p_memsz  = 32              , /* (bytes in mem at runtime) */
        .p_flags  = 0x5             , /* PF_R | PF_X */
        .p_align  = 8               , /* (min mem alignment in bytes) */
}
```

```
/* Program Header #7 0x1C8 */
{
        .p_type   = 4               , /* [PT_NOTE] */
        .p_offset = 824             , /* (bytes into file) */
        .p_vaddr  = 0x338           , /* (virtual addr at runtime) */
        .p_paddr  = 0x338           , /* (physical addr at runtime) */
        .p_filesz = 32              , /* (bytes in file) */
        .p_memsz  = 32              , /* (bytes in mem at runtime) */
        .p_flags  = 0x4             , /* PF_R */
        .p_align  = 8               , /* (min mem alignment in bytes) */

        /* note section dump:
         * Elf64_Nhdr note0 = {
         *       .n_namesz = 4, (bytes) [GNU]
         *       .n_descsz = 16, (bytes) [ 02 80 00 C0 04 00 00 00 01 00 00 00
         *       .n_type   = 5, [UNKNOWN_TYPE]
         * };
         */
}
```

Now i just put the « virus » that is supposed to print some text, but it's not working, whenever i call delta function i get a segmentation fault.

Fix is instead of calling the delta function we lea rsi, [rel v_start] to directly compute the address of the injected code

Also the p_vaddr an p_paddr were not aligned correctly not the virus does execute

Now the problem is that when the virus execute it goes into a seg fault. Maybe we did not jump to the original entry point correctly but after looking at gdb i see that the jmp ra xis at cause of seg fault and i moved r14(he contains the entry point) into rax just before so I am indeed right : r14 did not contain the right entry point or was corrupted somehow

Maybe i'm supposed to also subtract the size of the virus into r14 (my entry point address). Or most probably i got the wrong offset… Or i can try reading the file by myseft and then hardcode the entry point