

БИНАРНЫЙ ТРАНСЛЯТОР

Баринов Денис

МФТИ, гр.Б05-931

ЦЕЛИ

- ▶ Исследовать ELF файл. Понять устроен минимальный ELF.
- ▶ Научиться переводить бинарный файл, полученный с помощью собственного языка программирования, в исполняемый файл для x86-64.
- ▶ Перевести решение квадратного уравнения.

Исследование формата занимает бОльшую часть времени работы, вот некоторые из самых полезных ссылок:

-<https://cirosantilli.com/elf-hello-world#program-header-table> (Описание структуры ELF)

-<http://www.sunshine2k.de/coding/javascript/onlineelfviewer/onlineelfviewer.html> (Вывод ELF файла)

-https://ru.wikipedia.org/wiki/Executable_and_Linkable_Format (Много о структуре заголовках)

-<http://timelessname.com/elfbin/> (Минимальный Hello World)

Все ссылки будут в конце на отдельном слайде

После тщательного и долгого изучения, смотрим на ELF header написанного на asm. В моём случае получилось так:

```
denis@ubuntu:~$ readelf -h asm.out
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:                                EXEC (Executable file)
  Machine:                             Advanced Micro Devices X86-64
  Version:                             0x1
  Entry point address:                  0x401000
  Start of program headers:              64 (bytes into file)
  Start of section headers:             8848 (bytes into file)
  Flags:                                0x0
  Size of this header:                   64 (bytes)
  Size of program headers:               56 (bytes)
  Number of program headers:              3
  Size of section headers:               64 (bytes)
  Number of section headers:              6
  Section header string table index:      5
```

С помощью readelf -h FILENAME

В синих рамочках – то, что надо изменить для нашего ELF

Так как мы внимательно читали все сайты, то нам известно, что всё что нам нужно для минимального ELF это:

-ELF header

-Program header

-code

Поэтому мы изменяем ELF header так:

Первые 8 байт:

```
elf_header->e_ident[0] = '\0'; // 0x7f
elf_header->e_ident[1] = 'E';
elf_header->e_ident[2] = 'L';
elf_header->e_ident[3] = 'F';

elf_header->e_ident[4] = 0x02;
elf_header->e_ident[5] = 0x01;
elf_header->e_ident[6] = 0x01;
elf_header->e_ident[7] = 0x00;
elf_header->e_ident[8] = 0x00;
```

```
const elf::Elf64_Half C_my_e_type      = 0x0002;
const elf::Elf64_Half C_my_e_machine   = 0x003E;
const elf::Elf64_Word C_my_e_version   = 0x00000001;
const elf::Elf64_Addr C_my_e_entry     = 0x000000000000400078;
const elf::Elf64_Off  C_my_e_phoff     = 0x000000000000000040;
const elf::Elf64_Off  C_my_e_shoff     = 0x000000000;
const elf::Elf64_Word C_my_e_flags     = 0x00000000;
const elf::Elf64_Half C_my_e_ehsize    = 0x0040; // 64 bit
const elf::Elf64_Half C_my_e_phentsize = 0x0038; // 64 bit
const elf::Elf64_Half C_my_e_phnum     = 0x0001; // default
const elf::Elf64_Half C_my_e_shentsize = 0x0040; // 64 bit
const elf::Elf64_Half C_my_e_shnum     = 0x0000; // because without any sections
const elf::Elf64_Half C_my_e_shstrndx  = 0x0000; // because without any sections
```

PROGRAM HEADER

```
const elf::Elf64_Word C_my_p_type    = 0x00000001;  
const elf::Elf64_Word C_my_p_flags   = 0x00000005;  
const elf::Elf64_Off  C_my_p_offset  = 0x0000000000000000;  
const elf::Elf64_Addr C_my_p_vaddr   = 0x0000000000040000;  
const elf::Elf64_Addr C_my_p_paddr   = 0x0000000000040000;  
const elf::Elf64_Xword C_my_p_filesz = 0x0000000000000000; // TBA  
const elf::Elf64_Xword C_my_p_memsz  = 0x0000000000000000; // TBA  
const elf::Elf64_Xword C_my_p_align  = 0x0000000000001000;
```

filesz и memsz определим в самом конце трансляции
(так как пока что непонятно, сколько будет занимать новый файл)

Чтобы заполнить эту структуры я взял шаблон, сделал namespace и отдельный файл:

Взято из:

<https://code.woboq.org/linux/include/elf.h.html#65>

```
13 #include <stdint.h>
14
15 namespace elf
16 {
17
18     typedef uint16_t Elf64_Half;
19     typedef uint32_t Elf64_Word;
20     typedef uint64_t Elf64_Addr;
21     typedef uint64_t Elf64_Off;
22     typedef uint64_t Elf64_Xword;
23
24
25     /**
26      *====
27      *;;
28      *;;
29      *;;
30      *====
31      */
32     struct Elf64_Ehdr
33     {
34         unsigned char e_ident[16]; /* Magic number and other info */
35         Elf64_Half e_type; /* Object file type */
36         Elf64_Half e_machine; /* Architecture */
37         Elf64_Word e_version; /* Object file version */
38         Elf64_Addr e_entry; /* Entry point virtual address */
39         Elf64_Off e_phoff; /* Program header table file offset */
40         Elf64_Off e_shoff; /* Section header table file offset */
41         Elf64_Word e_flags; /* Processor-specific flags */
42         Elf64_Half e_ehsize; /* ELF header size in bytes */
43         Elf64_Half e_phentsize; /* Program header table entry size */
44         Elf64_Half e_phnum; /* Program header table entry count */
45         Elf64_Half e_shentsize; /* Section header table entry size */
46         Elf64_Half e_shnum; /* Section header table entry count */
47         Elf64_Half e_shstrndx; /* Section header string table index */
48     };
49
50     /**
51     *====
52     *;;
53     *;;
54     *;;
55     *====
56     */
57     struct Elf64_Phdr
58     {
59         Elf64_Word p_type; /* Segment type */
60         Elf64_Word p_flags; /* Segment flags */
61         Elf64_Off p_offset; /* Segment file offset */
62         Elf64_Addr p_vaddr; /* Segment virtual address */
63         Elf64_Addr p_paddr; /* Segment physical address */
64         Elf64_Xword p_filesz; /* Segment size in file */
65         Elf64_Xword p_memsz; /* Segment size in memory */
66         Elf64_Xword p_align; /* Segment alignment */
67     };
68 }
```

Узнаем коды команд. В этом нам поможет radare2.

(Установка: `sudo apt install radare2`)

!!!Это очень полезная программа, которая нам ещё очень понадобится, поэтому очень рекомендую!!!

Создадим asm файл с нужными командами и посмотрим на их коды:

r2 FILENAME

pd CNT (CNT – количество строчек, которые вы хотите вывести)

0x00401005	6650	push ax
0x00401007	6658	pop ax
0x00401009	6689c6	mov si, ax
0x0040100c	6689df	mov di, bx
0x0040100f	6689f0	mov ax, si
0x00401012	6689fb	mov bx, di
0x00401015	6601fe	add si, di
0x00401018	6629fe	sub si, di
0x0040101b	66f7f3	div bx



q (выход из radare2)

Заведём файл со всеми кодами (следующий сайт)

Полученный файл выглядит примерно следующим образом:

Это только часть файла

Но по ней можно выделить 2 вида команд: из 1 байта и из нескольких. Для второго вида сделаем соответствующие функции, чтобы не писать это каждый раз вручную;

```
//-----  
inline void Move_Di_Bx (unsigned char *res, int *counter)  
{  
    res[(*counter)++] = C_mov_di_bx[0];    ///  
    res[(*counter)++] = C_mov_di_bx[1];    ///  
    res[(*counter)++] = C_mov_di_bx[2];    ///  
}  
//-----  
inline void Pop_Reg (unsigned char *res, int *counter, const int offset)  
{  
    res[(*counter)++] = C_pop_reg;    ///  
    res[(*counter)++] = C_pop_start_reg + offset;    ///  
}  
//-----
```

```
1  
2  
3  //=====================================================================  
4  //=====================================================================  LINUX OPCODES FOR BINARY TRANSLATOR  //=====================================================================  
5  //=====================================================================  
6  // Linux_opcodes.h  
7  
8  const unsigned char C_push_reg      = 0x66;  
9  const unsigned char C_push_start_reg = 0x50;  
10  
11  const unsigned char C_pop_reg       = 0x66;  
12  const unsigned char C_pop_start_reg = 0x58;  
13  
14  const unsigned char C_end[12] = {0xb8, 0x3c, 0, 0, 0,  
15                                  0xbf, 0, 0, 0, 0,  
16                                  0x0f, 0x05};  
17  
18  const unsigned char C_add_ax_bx[3] = {0x66, 0x01, 0xd8};  
19  const unsigned char C_sub_ax_bx[3] = {0x66, 0x29, 0xd8};  
20  
21  const unsigned char C_mov_si_ax[3] = {0x66, 0x89, 0xc6};  
22  const unsigned char C_mov_di_bx[3] = {0x66, 0x89, 0xdf};  
23  const unsigned char C_mov_ax_si[3] = {0x66, 0x89, 0xf0};  
24  const unsigned char C_mov_bx_di[3] = {0x66, 0x89, 0xfb};  
25  
26  const unsigned char C_xor_ah_ah[2] = {0x30, 0xe4};  
27  
28  const unsigned char C_div_bx[3]     = {0x66, 0xf7, 0xf3};  
29  const unsigned char C_mul_bx[3]     = {0x66, 0xf7, 0xe3};  
30  
31  const unsigned char C_mov_offset_al[3] = {0x88, 0x04, 0x25};  
32  
33  const unsigned char C_mov_rdi_not_reg = 0xbf;  
34  
35  const unsigned char C_mov_rdx_not_reg = 0xba;  
36  
37  const unsigned char C_mov_rax_not_reg = 0xb8;  
38  
39  const unsigned char C_mov_rsi_offset[2] = {0x48, 0xbe};
```

Далее используя файл Commands.h и дефайны (так же, как мы делали собственный язык программирования) пишем код для каждой команды:

```
DEF_CMD(SUB, 4,
{
    offsets_arr[pos] = counter;

    REALLOC_RES

    Pop_Reg (res, &counter, E_si);
    Pop_Reg (res, &counter, E_di);

    Sub_Si_Di (res, &counter);

    Push_Reg (res, &counter, E_si);

    REALLOC_RES

    pos++;

    break;
}, 0)
```



665e	pop si
665f	pop di
6629fe	sub si, di
6656	push si

Практически все команды переводятся легко. (Если уже есть коды и удобные функции)

Особо интересной и сложной была команда PRT (print). У меня получилось так:

Действительно, если сравнивать с той же командой SUB, то выглядит очень громоздко и непонятно. Это из-за того, что чтобы что-то вывести надо:

- поместить после исполняемого кода то, что хотим вывести
- в коде правильно указать смещение (во фрагменте кода - TBA)
- вызвать `syscall` с правильными аргументами

Первый пункт легко реализуем – просто записываем в отдельный буфер, который потом запишем в файл после основного. (Это у меня `strings_arr`)

Второе обсудим подробнее, так как этот метод применяется и для `jmp`, `ja`, ...

Сложность третьего исчезает после того, как разобрались с первым пунктом.

```
DEF_CMD(PRT, 72,
{
    offsets_arr[pos] = counter;

    REALLOC_RES
    pos++;

    char* helper = (buf + pos);
    int len = strlen(helper);

    pos += (len);

    memmove(strings_arr + strings_arr_pos, helper, len);
    strings_arr_pos += 20 - strings_arr_pos % 20;

    res[counter++] = C_mov_rax_not_reg;
    * (int *) (res + counter) = 1;

    counter += sizeof(int);

    res[counter++] = C_mov_rdi_not_reg;
    * (int *) (res + counter) = 1;

    counter += sizeof(int);

    REALLOC_RES
    Move_Rsi_Offset(res, &counter);

    for (int i = 0; i < 8; i++)
        res[counter++] = 0;    // TBA

    REALLOC_RES

    res[counter++] = C_mov_rdx_not_reg;

    for (int i = 0; i < 4; i++)
        res[counter++] = 0;    // TBA

    Syscall(res, &counter);

    break;
}, 1)
```

Как искать смещение в новом файле?

Обсудим, как находить нужные смещения. Для этого заведём `offsets_arr`, который так раз и будет за это отвечать. И всё что нужно – обновлять его перед каждой командой. (Если пролистать назад, то его можно заметить в самом начале SUB и PRT)

Тогда если `buf` – исходный буфер, то если мы хотим смещение на `buf[pos]` в нашем новом файле, то достаточно посмотреть на `offsets_arr[pos]`. Но не стоит забывать, что правильные смещения появятся только после прохода по всему файлу, поэтому аргументы для `jmp` и т.д. мы заполняем не сразу, следующим проходом.

Заполнив заголовки, массивы и указав нужные смещения, запишем через fwrite в файл и не забудем указать filesz и memsz.

Открой полученный файл с помощью уже упомянутого radare2 и увидим следующее:

И так будет более 300 строчек!
Что лично меня очень впечатлило!

Проверим и увидим долгожданные надписи:

no_root

inf_cnt_of_root

0_1_root

```
0x00000078 e900000000 jmp 0x7d
0x0000007d 66be0000 mov si, 0
0x00000081 6656 push si
0x00000083 6658 pop ax
0x00000085 66be0000 mov si, 0
0x00000089 6656 push si
0x0000008b 665b pop bx
0x0000008d 66be0100 mov si, 1
0x00000091 6656 push si
0x00000093 6659 pop cx
0x00000095 6650 push ax
0x00000097 66be0000 mov si, 0
0x0000009b 6656 push si
0x0000009d 665e pop si
0x0000009f 665f pop di
0x000000a1 6639fe cmp si, di
0x000000a4 7405 je 0xab
0x000000a6 e934010000 jmp 0x1df
0x000000ab 6653 push bx
0x000000ad 66be0000 mov si, 0
0x000000b1 6656 push si
0x000000b3 665e pop si
0x000000b5 665f pop di
0x000000b7 6639fe cmp si, di
0x000000ba 7405 je 0xc1
0x000000bc e964000000 jmp 0x125
0x000000c1 6651 push cx
0x000000c3 66be0000 mov si, 0
0x000000c7 6656 push si
0x000000c9 665e pop si
0x000000cb 665f pop di
0x000000cd 6639fe cmp si, di
0x000000d0 7405 je 0xd7
0x000000d2 e927000000 jmp 0xfe
0x000000d7 b801000000 mov eax, 1
0x000000dc bf01000000 mov edi, 1
0x000000e1 48beec034000 movabs rsi, 0x4003ec
0x000000eb ba0f000000 mov edx, 0xf
0x000000f0 0f05 syscall
0x000000f2 b83c000000 mov eax, 0x3c
0x000000f7 bf00000000 mov edi, 0
0x000000fc 0f05 syscall
0x000000fe b801000000 mov eax, 1
0x00000103 bf01000000 mov edi, 1
0x00000108 48be00044000 movabs rsi, 0x400400
0x00000112 ba07000000 mov edx, 7
0x00000117 0f05 syscall
0x00000119 b83c000000 mov eax, 0x3c
```


ИТОГИ

- Мы полностью разобрались в том, как устроен ELF файл.
- Теперь числа не %x кажутся очень странными и неестественными.
- Смотреть исходник теперь не кажется занятием для хакеров, а совершенно повседневной деятельностью.
- Сделан бинарный транслятор с одного процессора на другой, что несомненно очень большой полезный и главное – интересный опыт.

Хотелось бы тут отметить самые важные детали работы: (на мой взгляд)

- Разобраться с ELF форматом (да, это очень долго, но поверьте, оно того стоит!)
- Экспериментировать и постоянно тестировать. Это очень ускорит понимание того, что и как надо писать. (radare2 в помощь)
- Отдельно тестировать каждую команду, а не сразу многие, хоть и кажется, что они простые и всё работает.
- Не забывать, что исполняемый код начинается не с 0x40000, а с 0x40078. (В нашем случае, так как обычно выравнивание стоит 0x1000, а не как 0 у нас)
- Если что-то не работает ещё раз почитать, потому что если всё понял и осознал, то код уже пишется быстро и без проблем

А тут я оставлю ссылки и инструменты, которыми я пользовался.
(Так как я совсем немного рассказал про структуру файла, то совету, их почитать (а то совсем неинтересно было бы:D))

ELF FILE:

<https://cirosantilli.com/elf-hello-world#program-header-table>
<http://www.sunshine2k.de/coding/javascript/onlineelfviewer/onlineelfviewer.html>
https://ru.wikipedia.org/wiki/Executable_and_Linkable_Format
<http://timelessname.com/elfbin/>
<https://code.woboq.org/linux/include/elf.h.html#65>

Linux:

`sudo apt install radare2`
`r2 -n FILENAME` (-n программа с 0 байта (а не 0x40000))
`pd CNT` (дизасемблирует CNT строк)
`q` (выход из radare2)

`hd FILENAME` (Дамп в %x файла)
`readelf -a FILENAME` (выводит всю служебную информацию – заголовки, сегменты и т.д.)
`readelf -h FILENAME` (выводит только ELF header файла)
`readelf -I FILENAME` (выводит информацию о секциях и сегментах)

`strip FILENAME` (я не рассказал об этой команде, хотя она очень полезная на моменте осознания минимального файла. Так что стоит ознакомиться с ней. Например, можно тут: <https://www.linuxlib.ru/manpages/STRIP.1.shtml>)