Reverse Engineering

Tools

Mainack Mondal

CS60112
Spring 2024

Today’s Class

- What even is Rev Eng?
- Bare hands: Objdump
- The hammer: GDB
- The sledgehammer: Radare2
- The jackhammer: Angr

What even is Rev Eng?

What even is Rev Eng?

Reading compiled source code!

(plus or minus the reading)

The Four Horsemen of Rev Eng

```
00000000000088a9 <main>:
    88a9:       55                      push   %rbp
    88aa:       48 89 e5                mov    %rsp,%rbp
    88ad:       53                      push   %rbx
    88ae:       48 83 ec 58             sub    $0x58,%rsp
    88b2:       89 7d ac                mov    %edi,-0x54(%rbp)
    88b5:       48 89 75 a8             mov    %rsi,-0x58(%rbp)
    88b9:       64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
    88c0:       00 00
    88c2:       48 89 45 e8             mov    %rax,-0x18(%rbp)
    88c6:       31 c0                   xor    %eax,%eax
    88c8:       48 8d 45 c8             lea    -0x40(%rbp),%rax
    88cc:       48 8d 15 75 07 01 00    lea    0x10775(%rip),%rdx        # 19048 <Pango::SCALE_XX_LARGE+0x8>
    88d3:       48 89 d6                mov    %rdx,%rsi
    88d6:       48 89 c7                mov    %rax,%rdi
    88d9:       e8 f2 fa ff ff          call   83d0 <Glib::ustring::ustring(char const*)@plt>
    88de:       48 8d 45 b0             lea    -0x50(%rbp),%rax
    88e2:       48 8d 4d c8             lea    -0x40(%rbp),%rcx
    88e6:       ba 00 00 00 00          mov    $0x0,%edx
    88eb:       48 89 ce                mov    %rcx,%rsi
    88ee:       48 89 c7                mov    %rax,%rdi
    88f1:       e8 1a fa ff ff          call   8310 <Gtk::Application::create(Glib::ustring const&, Gio::Application::Flags)@plt>
    88f6:       48 8d 45 c8             lea    -0x40(%rbp),%rax
    88fa:       48 89 c7                mov    %rax,%rdi
    88fd:       e8 9e fd ff ff          call   86a0 <Glib::ustring::~ustring()@plt>
    8902:       48 8d 45 b0             lea    -0x50(%rbp),%rax
    8906:       48 89 c7                mov    %rax,%rdi
    8909:       e8 3a 03 00 00          call   8c48 <std::__shared_ptr_access<Gtk::Application, (__gnu_cxx::__lock_policy)2, false, false>::operator->() const>
    890e:       48 89 c1                mov    %rax,%rcx
    8911:       48 8b 55 a8             mov    -0x58(%rbp),%rdx
    8915:       8b 45 ac                mov    -0x54(%rbp),%eax
    8918:       89 c6                   mov    %eax,%esi
    891a:       48 89 cf                mov    %rcx,%rdi
    891d:       e8 c8 04 00 00          call   8dea <int Gtk::Application::make_window_and_run<pinscope::Pinscope>(int, char**)>
    8922:       89 c3                   mov    %eax,%ebx
```

Read

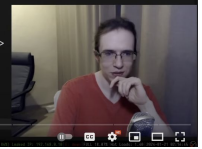
```
dknite in @ Collosus in pinscope/build on |/ main
> gdb pinscope
GNU gdb (GDB) 13.2
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from pinscope...

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://sourceware.org/elfutils-artifacts>
Enable debuginfod for this session? (y or [n]) n
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.
(No debugging symbols found in pinscope)
(gdb) b main
Breakpoint 1 at 0x9260
(gdb) r
Starting program: /home/dknite/work/cpp/pinscope/build/pinscope
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/usr/lib/libthread_db.so.1".

Breakpoint 1, 0x00005555555d260 in main ()
(gdb) disass main,main+20
Dump of assembler code from 0x55555555d260 to 0x55555555d274:
=> 0x00005555555d260 <main+0>: push   %r12
   0x00005555555d262 <main+2>: mov    %rsi,%r12
   0x00005555555d265 <main+5>: lea    0x7dd4(%rip),%rsi        # 0x555555565010
   0x00005555555d26c <main+12>:        push   %rbp
   0x00005555555d26d <main+13>:        mov    %edi,%ebp
   0x00005555555d26f <main+15>:        push   %rbx
   0x00005555555d270 <main+16>:        sub    $0x40,%rsp
End of assembler dump.
(gdb)
```

Execute

```
dknite in @ Collosus in ~/work/stuff via C v13.2.1-gcc
> r2 -e bin.cache=true -w stupid
WARN: Cannot resolve symbol address __libc_start_main
WARN: Cannot resolve symbol address __ITM_deregisterTMCloneTable
WARN: Cannot resolve symbol address __gmon_start__
WARN: Cannot resolve symbol address __ITM_registerTMCloneTable
WARN: Cannot resolve symbol address __cxa_finalize
[0x00001060]> aa
INFO: Analyze all flags starting with sym. and entry0 (aa)
INFO: Analyze all functions arguments/locals (afvnsbbb*)
[0x00001060]> pdf @main
            ;-- section..text:
            ; DATA XREF from entry0 @ 0x1060(r)
  59: int main (int64_t argc);
            ; arg int64_t argc @ rsi
            0x00001060      53              push rbx                ; [14] -r-x section size 313 named .text
            0x00001061      48b7e           mov rbx, qword [rsi]    ; arg2
            0x00001064      489fef          mov rdi, rbx
            0x00001067      e804:ffffff     call sym.imp.strlen     ; size_t strlen(const char *s)
            0x0000106c      488d3918f8f8.   lea rsi, str..stupid    ; 0x2988a ; ".stupid"
            0x00001073      489fef          mov rdi, rbx
            0x00001076      4889c2          mov rdx, rax
            0x00001079      e842:ffffff     call sym.imp.memcmp     ; int memcmp(const void *s1, const void *s2, size_t n)
            0x0000107e      89c2            mov ecx, eax
            0x00001080      b801000000      mov eax, 1
            0x00001085      85d2            test edx, edx
           ,=< 0x00001087      75b2            jne 0x108b
           |   0x00001089      5b              pop rbx
           |   0x0000108a      c3              ret
           |-> 0x0000108b      488d3d7d0f00.   lea rdi, str.Secret_Message ; 0x2080d ; "Secret Message"
           |   0x00001092      e899:ffffff     call sym.imp.puts       ; int puts(const char *s)
           |   0x00001097      31c0            xor eax, eax
           |   0x00001099      5b              pop rbx
           |   0x0000109a      c3              ret
[0x00001060]> s 0x1087
[0x00001087]> wa jmp 0x108b
INFO: Written 2 byte(s) (jmp 0x108b) = wx eb02 @ 0x00001087
[0x00001087]> q -?
[0x00001089]> pd 5
           |   0x00001089      5b              pop rbx
           |   0x0000108a      c3              ret
           |   0x00001080      b801000000      mov eax, 1
           |   0x00001085      85d2            test edx, edx
           ,=< 0x00001087      eb02            jmp 0x108b
           |   0x00001089      5b              pop rbx
           |   0x0000108a      c3              ret
[0x00001089]> quit
```

Patch

```
00000000000097a0 <main>:
    97a0:   48 83 ec 08             sub    rsp,0x8
    97a4:   e8 99 da 02 00          call   37242 <GetFrameTime>
    97a9:   f3 0f 59 05 5f 38 0f    mulss  xmm0,DWORD PTR [rip+0xf385f]        # fd010 <_IO_stdin_used+0x10>
    97b0:   00
    97b1:   b8 02 00 00 00          mov    eax,0x2
    97b6:   48 8d 3d 47 38 0f 00    lea    rdi,[rip+0xf3847]        # fd004 <_IO_stdin_used+0x4>
    97bd:   0f 28 c8                movaps xmm1,xmm0
    97c0:   66 0f ef c0             pxor   xmm0,xmm0
    97c4:   f3 0f 58 c8             addss  xmm1,xmm0
    97c8:   f3 0f 5a c9             cvtss2sd xmm1,xmm1
    97cc:   66 0f 28 c1             movapd xmm0,xmm1
    97d0:   e8 6b f8 ff ff          call   9040 <printf@plt>
    97d5:   31 c0                   xor    eax,eax
    97d7:   48 83 c4 08             add    rsp,0x8
    97db:   c3                      ret
```



Be Tsoding

Let’s read some assembly!

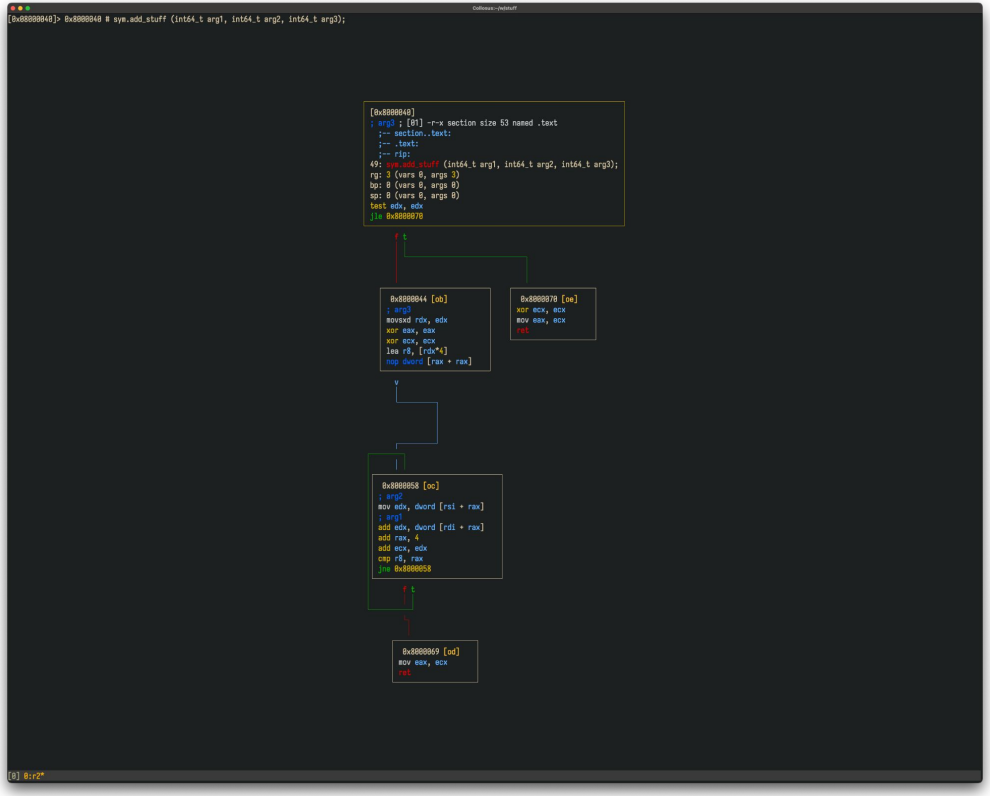
```
 0:   85 d2                   test   edx,edx
 2:   7e 2c                   jle    30 <add_stuff+0x30>
 4:   48 63 d2                movsxd rdx,edx
 7:   31 c0                   xor    eax,eax
 9:   31 c9                   xor    ecx,ecx
 b:   4c 8d 04 95 00 00 00    lea    r8,[rdx*4+0x0]
12:   00
13:   0f 1f 44 00 00          nop    DWORD PTR [rax+rax*1+0x0]
18:   8b 14 06                mov    edx,DWORD PTR [rsi+rax*1]
1b:   03 14 07                add    edx,DWORD PTR [rdi+rax*1]
1e:   48 83 c0 04             add    rax,0x4
22:   01 d1                   add    ecx,edx
24:   49 39 c0                cmp    r8,rax
27:   75 ef                   jne    18 <add_stuff+0x18>
29:   89 c8                   mov    eax,ecx
2b:   c3                      ret
2c:   0f 1f 40 00             nop    DWORD PTR [rax+0x0]
30:   31 c9                   xor    ecx,ecx
32:   89 c8                   mov    eax,ecx
34:   c3                      ret
```

Let’s read some assembly!

```
 0:   85 d2                    test   edx,edx
 2:   7e 2c                    jle    30 <add_stuff+0x30>
 4:   48 63 d2                 movsxd rdx,edx
 7:   31 c0                    xor    eax,eax
 9:   31 c9                    xor    ecx,ecx
 b:   4c 8d 04 95 00 00 00     lea    r8,[rdx*4+0x0]
12:   00
13:   0f 1f 44 00 00           nop    DWORD PTR [rax+rax*1+0x0]
18:   8b 14 06                 mov    edx,DWORD PTR [rsi+rax*1]
1b:   03 14 07                 add    edx,DWORD PTR [rdi+rax*1]
1e:   48 83 c0 04              add    rax,0x4
22:   01 d1                    add    ecx,edx
24:   49 39 c0                 cmp    r8,rax
27:   75 ef                    jne    18 <add_stuff+0x18>
29:   89 c8                    mov    eax,ecx
2b:   c3                       ret
2c:   0f 1f 40 00              nop    DWORD PTR [rax+0x0]
30:   31 c9                    xor    ecx,ecx
32:   89 c8                    mov    eax,ecx
34:   c3                       ret
```

```
int add_stuff(int *a, int *b, int cnt) {
  int sum = 0;
  for (int i = 0; i < cnt; i++) {
    sum += a[i] + b[i];
  }
  return sum;
}
```

Let’s read some assembly!



Structure?

Is reading enough?

```
work/cpp/infosec_demos via C v12.2.0-gcc
❯ ./run.sh ./echo
Enter the string
Hello World!
Here you go: roW!dlleH ol

work/cpp/infosec_demos via C v12.2.0-gcc took 9s
❯ ./run.sh ./echo
Enter the string
Riddle me this soldier.
Here you go: so islr.edihdldRie te m
```

Is reading enough?

```
00000000000010a0 <main>:
    10a0:       53                      push   rbx
    10a1:       48 8d 15 5c 0f 00 00    lea    rdx,[rip+0xf5c]        # 2004 <_IO_stdin_used+0x4>
    10a8:       48 89 d7                mov    rdi,rdx
    10ab:       48 89 d0                mov    rax,rdx
    10ae:       e8 9d ff ff ff          call   1050 <printf@plt>
    10b3:       31 c0                   xor    eax,eax
    10b5:       e8 26 01 00 00          call   11e0 <read_str>
    10ba:       48 8d 15 55 0f 00 00    lea    rdx,[rip+0xf55]        # 2016 <_IO_stdin_used+0x16>
    10c1:       48 89 c3                mov    rbx,rax
    10c4:       48 89 d7                mov    rdi,rdx
    10c7:       48 89 d0                mov    rax,rdx
    10ca:       e8 81 ff ff ff          call   1050 <printf@plt>
    10cf:       48 89 df                mov    rdi,rbx
    10d2:       e8 69 ff ff ff          call   1040 <puts@plt>
    10d7:       48 89 df                mov    rdi,rbx
    10da:       e8 51 ff ff ff          call   1030 <free@plt>
    10df:       31 c0                   xor    eax,eax
    10e1:       5b                      pop    rbx
    10e2:       c3                      ret
    10e3:       66 2e 0f 1f 84 00 00    cs nop WORD PTR [rax+rax*1+0x0]
    10ea:       00 00 00
    10ed:       0f 1f 00                nop    DWORD PTR [rax]
```

GDB = Runtime Analysis

When does reading assembly become insufficient?

GDB = Runtime Analysis

- Dynamically loaded code may not be what you expect it to be
- Loaders can do sneaky things to your code

- ELF relocations can be used to perform computations (more on this later)
- Code is data and binaries can patch themselves

GDB for Rev Eng

- Use stepi (si) and nexti (ni) instead of step and next
- Use disass [START,[END]] to view current asm
- Watchpoints are good for observing memory location changes
- You can script GDB with Python (although r2 is more powerful)

sprint (Google CTF 2020, Quals)

https://ctftime.org/writeup/23032

Stripped Binaries

How do you identify functions?

Angr - Concolic Analysis

- Does everything that r2 can do, but as a library
- Symbolic execution and constraint solver
- Decompiler (similar to Ghidra, but less refined)
- Loader as a library

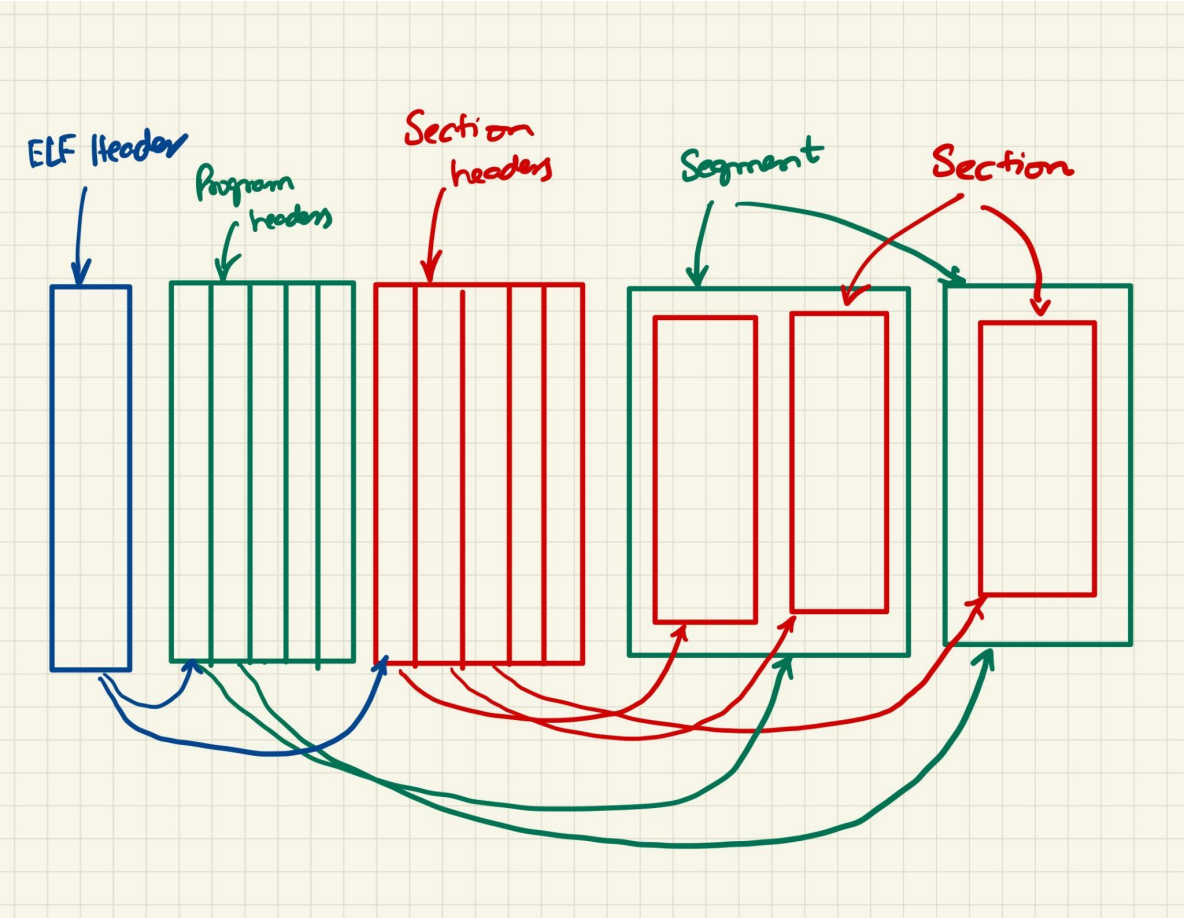
You can edit binaries!

```
            ;-- section..text:
            ; DATA XREF from entry0 @ 0x1164(r)
/ 98: int main (int argc, char **argv, char **envp);
|           ; var int64_t var_8h @ rsp+0x8
|           0x000010e0      53             push rbx                    ; [15] -r-x section size 729 named .text
|           0x000010e1      488d3d220f..   lea rdi, str.Enter_the_password: ; 0x200a ; "Enter the password:" ; const char *s
|           0x000010e8      4883ec10       sub rsp, 0x10
|           0x000010ec      e84fffffff     call sym.imp.puts           ; int puts(const char *s)
|           0x000010f1      488d7c2408     lea rdi, [var_8h]           ; int64_t arg1
|           0x000010f6      e8c5010000     call fcn.000012c0
|           0x000010fb      488b742408     mov rsi, qword [var_8h]
|           0x00001100      4889c7         mov rdi, rax
|           0x00001103      4889c3         mov rbx, rax
|           0x00001106      e845010000     call fcn.00001250
|           0x0000110b      84c0           test al, al
|       ,=< 0x0000110d      7425           je 0x1134
|       |   0x0000110f      488d3d080f..   lea rdi, str.Here_you_go:   ; 0x201e ; "Here you go: " ; const char *format
|       |   0x00001116      31c0           xor eax, eax
|       |   0x00001118      e813ffffff     call sym.imp.printf         ; int printf(const char *format)
|       |   0x0000111d      31c0           xor eax, eax
|       |   0x0000111f      e82c020000     call fcn.00001350
|       |   ; CODE XREF from main @ 0x1140(x)
|      .--> 0x00001124      4889df         mov rdi, rbx                ; void *ptr
|      :|   0x00001127      e844ffffff     call sym.imp.free           ; void free(void *ptr)
|      :|   0x0000112c      4883c410       add rsp, 0x10
|      :|   0x00001130      31c0           xor eax, eax
|      :|   0x00001132      5b             pop rbx
|      :|   0x00001133      c3             ret
|      :|   ; CODE XREF from main @ 0x110d(x)
|      :`-> 0x00001134      488d3df10e..   lea rdi, str.Wrong_password__try_again ; 0x202c ; "Wrong password, try again" ; const char *s
|      :    0x0000113b      e800ffffff     call sym.imp.puts           ; int puts(const char *s)
\      `==< 0x00001140      ebe2           jmp 0x1124
```

beginner (Google CTF 2020, Quals)

https://github.com/luker983/google-ctf-2020/tree/master/reversing/beginner

Anatomy of ELF



ELF Relocations - Static

```
        global _start
        section .text

_start:
        mov rdi, 1
        mov rsi, msg
        mov rdx, 9
        mov rax, 1
        syscall

        xor rdi, rdi
        mov rax, 60
        syscall

        section .data
msg:    db "hi there", 10
```

```
0000000000001000 <_start>:
    1000:       bf 01 00 00 00          mov    edi,0x1
    1005:       48 be 00 30 00 00 00    movabs rsi,0x3000
    100c:       00 00 00
    100f:       ba 09 00 00 00          mov    edx,0x9
    1014:       b8 01 00 00 00          mov    eax,0x1
    1019:       0f 05                   syscall
    101b:       48 31 ff                xor    rdi,rdi
    101e:       b8 3c 00 00 00          mov    eax,0x3c
    1023:       0f 05                   syscall
```

```
[0x00001000]> s 0x3000
[0x00003000]> px 9
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x00003000  6869 2074 6865 7265 0a                  hi there.
[0x00003000]> q
```

ELF Relocations - Runtime

```
Breakpoint 1.1, 0x0000555555555000 in _start ()
(gdb) disass
Dump of assembler code for function _start:
=> 0x0000555555555000 <+0>:     mov    $0x1,%edi
   0x0000555555555005 <+5>:     movabs $0x555555557000,%rsi
   0x000055555555500f <+15>:    mov    $0x9,%edx
   0x0000555555555014 <+20>:    mov    $0x1,%eax
   0x0000555555555019 <+25>:    syscall
   0x000055555555501b <+27>:    xor    %rdi,%rdi
   0x000055555555501e <+30>:    mov    $0x3c,%eax
   0x0000555555555023 <+35>:    syscall
```

ELF Relocations - Dynamic Segment

```
Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000000040 0x0000000000000040
                 0x00000000000001c0 0x00000000000001c0  R      0x8
  INTERP         0x0000000000000200 0x0000000000000200 0x0000000000000200
                 0x000000000000001c 0x000000000000001c  R      0x1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000288 0x0000000000000288  R      0x1000
  LOAD           0x0000000000001000 0x0000000000001000 0x0000000000001000
                 0x0000000000000025 0x0000000000000025  R E    0x1000
  LOAD           0x0000000000002000 0x0000000000002000 0x0000000000002000
                 0x0000000000000000 0x0000000000000000  R      0x1000
  LOAD           0x0000000000002ed0 0x0000000000002ed0 0x0000000000002ed0
                 0x0000000000000139 0x0000000000000139  RW     0x1000
  DYNAMIC        0x0000000000002ed0 0x0000000000002ed0 0x0000000000002ed0
                 0x0000000000000130 0x0000000000000130  RW     0x8
  GNU_RELRO      0x0000000000002ed0 0x0000000000002ed0 0x0000000000002ed0
                 0x0000000000000130 0x0000000000000130  R      0x1
```

```
Dynamic section at offset 0x2ed0 contains 15 entries:
  Tag        Type                         Name/Value
 0x0000000000000004 (HASH)               0x220
 0x000000006ffffef5 (GNU_HASH)           0x230
 0x0000000000000005 (STRTAB)             0x268
 0x0000000000000006 (SYMTAB)             0x250
 0x000000000000000a (STRSZ)              1 (bytes)
 0x000000000000000b (SYMENT)             24 (bytes)
 0x0000000000000015 (DEBUG)              0x0
 0x0000000000000007 (RELA)               0x270
 0x0000000000000008 (RELASZ)             24 (bytes)
 0x0000000000000009 (RELAENT)            24 (bytes)
 0x0000000000000016 (TEXTREL)            0x0
 0x000000000000001e (FLAGS)              TEXTREL
 0x000000006ffffffb (FLAGS_1)            Flags: PIE
 0x000000006ffffff9 (RELACOUNT)          1
 0x0000000000000000 (NULL)               0x0
```

ELF Relocations - Relocation Table

```c
typedef struct __attribute__((packed)) {
   uintptr_t offset;
   uint32_t info;
   uint32_t type;
   int64_t addend;
} Elf64_Rela;
```

```
Relocation section '.rela.dyn' at offset 0x270 contains 1 entry:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000001007  000000000008 R_X86_64_RELATIVE                    3000
No processor specific unwind information to decode
```

ELF Relocations - Applying Relocations

Name	Value	Field	Calculation
R_X86_64_NONE	0	none	none
R_X86_64_64	1	*word64*	S + A
R_X86_64_PC32	2	*word32*	S + A − P
R_X86_64_GOT32	3	*word32*	G + A
R_X86_64_PLT32	4	*word32*	L + A − P
R_X86_64_COPY	5	none	none
R_X86_64_GLOB_DAT	6	*wordclass*	S
R_X86_64_JUMP_SLOT	7	*wordclass*	S
R_X86_64_RELATIVE	8	*wordclass*	B + A
R_X86_64_GOTPCREL	9	*word32*	G + GOT + A − P

ELF Symbol Table

```c
typedef struct __attribute__((packed)) {
    uint32_t name;
    uint8_t info;
    uint8_t _other;
    uint16_t sec_idx;
    uintptr_t value;
    uint64_t size;
} Elf64_Sym;
```

```
Symbol table '.symtab' contains 9 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS rawhello.asm
     2: 0000000000003000     0 NOTYPE  LOCAL  DEFAULT   10 msg
     3: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS
     4: 0000000000002ed0     0 OBJECT  LOCAL  DEFAULT    9 _DYNAMIC
     5: 0000000000001000     0 NOTYPE  GLOBAL DEFAULT    7 _start
     6: 0000000000003009     0 NOTYPE  GLOBAL DEFAULT   10 __bss_start
     7: 0000000000003009     0 NOTYPE  GLOBAL DEFAULT   10 _edata
     8: 0000000000003010     0 NOTYPE  GLOBAL DEFAULT   10 _end
```

Making ELF Turing Complete

Three instructions make a Turing complete machine:

- add
- mov
- jnz

Turing Complete Relocations

Type	Mnemonic	Operation
R_X86_64_COPY	COPY	memcpy(r.offset, s.value, s.size)
R_X86_64_64	SYM	*(base + r.offset) = s.value + r.addend + base
R_X86_64_RELATIVE	RELATIVE	*(base + r.offset) = r.addend + base

Using Relocations - mov

```
mov [0xdeadbeef], 10
    r = {type = RELATIVE, offset = 0xdeadbeef, symbol = 0
         , addend = 10}
```

Using Relocations - mov

```
mov [0xdeadbeef], [%foo]
    r = {type = COPY, offset = 0xdeadbeef, symbol = foo
         , addend = 0}
    s = {name = foo, value = 0xb0000000, type=FUNC,
         , sec_idx = 1, size = 8}
```

Using Relocations - add

```
add [0xdeadbeef], %foo, 0x02
    r = {type = SYM, offset = 0xdeadbeef, symbol = foo
         , addend = 2}
    s = {name = foo, value = 1, type=FUNC,
         , sec_idx = 1, size = 8}
```

Using Relocations - jmp

```
while (lm ≠ NULL) {
   r = lm→dyn[DT_RELA];
   end = r + lm→dyn[DT_RELASZ];
   while (r < end) {
       relocate(lm, r, &lm→dyn[DT_SYM]);
       r += lm→dyn[DT_RELAENT];
   }
}
```

1. Set the value of `lm->prev` so that the same relocation table is processed on the next `while` loop iteration.
   - The original `lm->prev` value needs to be restored later to allow the executable to eventually run
2. Set the value of `lm->dyn[DT_RELA]` to point to the the jump's destination (relocation entry)
3. Update the size of `lm->dyn[DT_RELASZ]` to reflect the "new" relocation table size
4. Clobber the value of `end` so that the RTLD does not process the next relocation entry

eldar (Google CTF 2022, Quals)

https://joshl.ca/posts/googlectf2022/