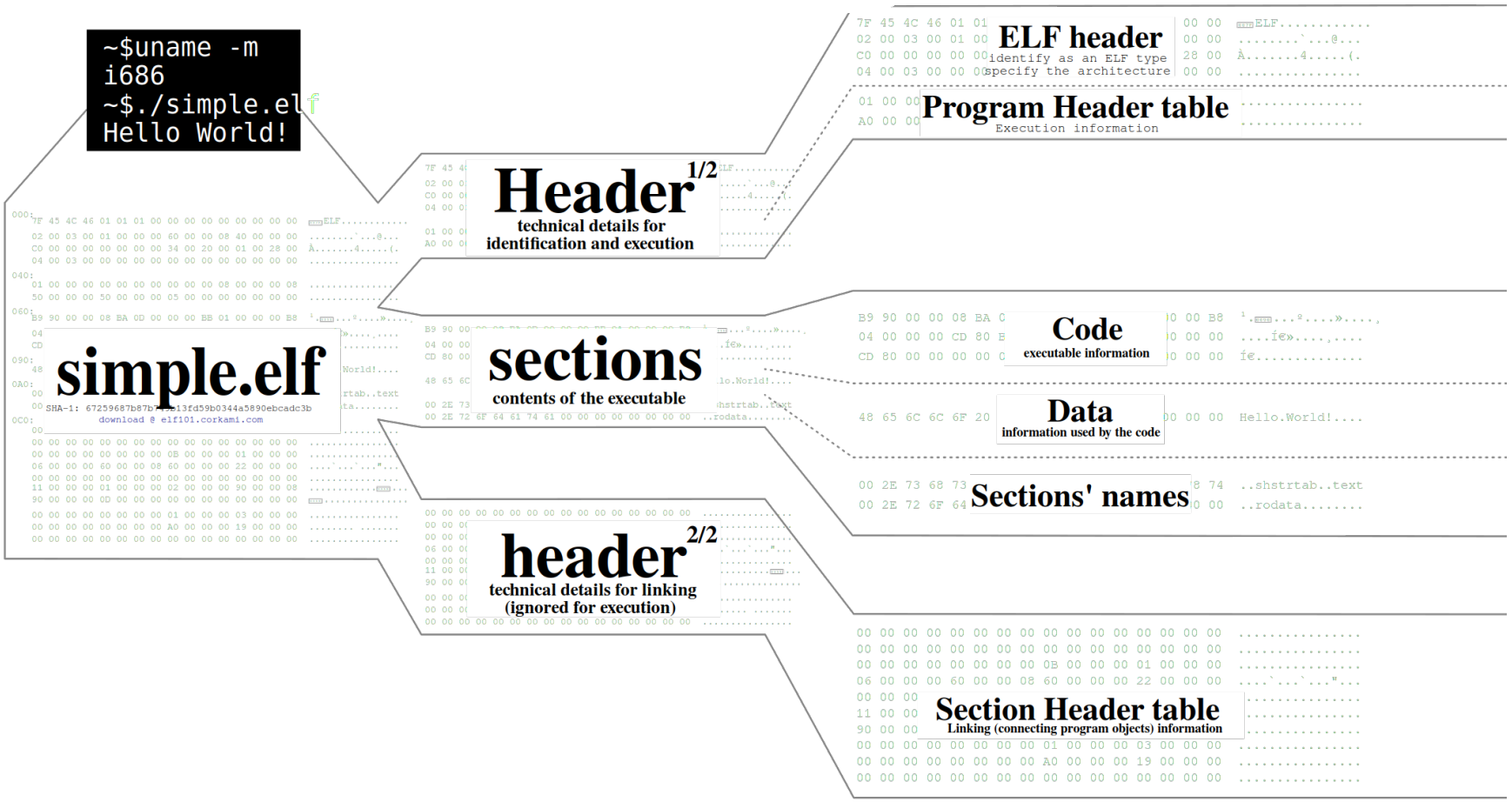
ELF Binary Obfuscation

Index

Overview



Img: https://github.com/corkami/pics/blob/master/binary/elf101/elf101-64.svg

ELF Header

```
typedef struct
{
  unsigned char         e_ident[EI_NIDENT];     /* Magic number and other info */
  Elf64_Half            e_type;                 /* Object file type */
  Elf64_Half            e_machine;              /* Architecture */
  Elf64_Word            e_version;              /* Object file version */
  Elf64_Addr            e_entry;                /* Entry point virtual address */
  Elf64_Off             e_phoff;                /* Program header table file offset */
  Elf64_Off             e_shoff;                /* Section header table file offset */
  Elf64_Word            e_flags;                /* Processor-specific flags */
  Elf64_Half            e_ehsize;               /* ELF header size in bytes */
  Elf64_Half            e_phentsize;            /* Program header table entry size */
  Elf64_Half            e_phnum;                /* Program header table entry count */
  Elf64_Half            e_shentsize;            /* Section header table entry size */
  Elf64_Half            e_shnum;                /* Section header table entry count */
  Elf64_Half            e_shstrndx;             /* Section header string table index */
} Elf64_Ehdr;
```


Code: https://code.woboq.org/userspace/glibc/elf/elf.h.html

*Program header is a linked list

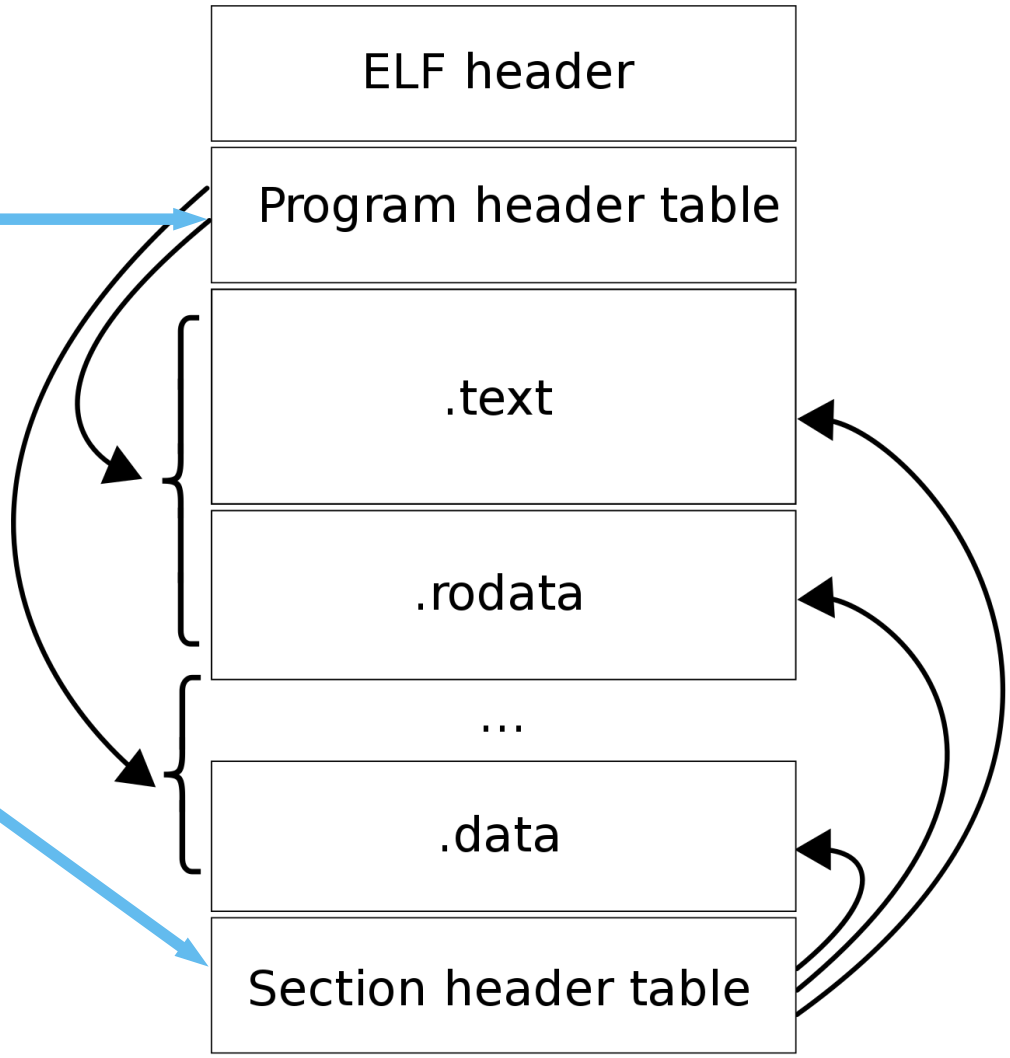
Sections & Segments

Segments

- Equals to one or more memory pages
- Permissions located in program header table

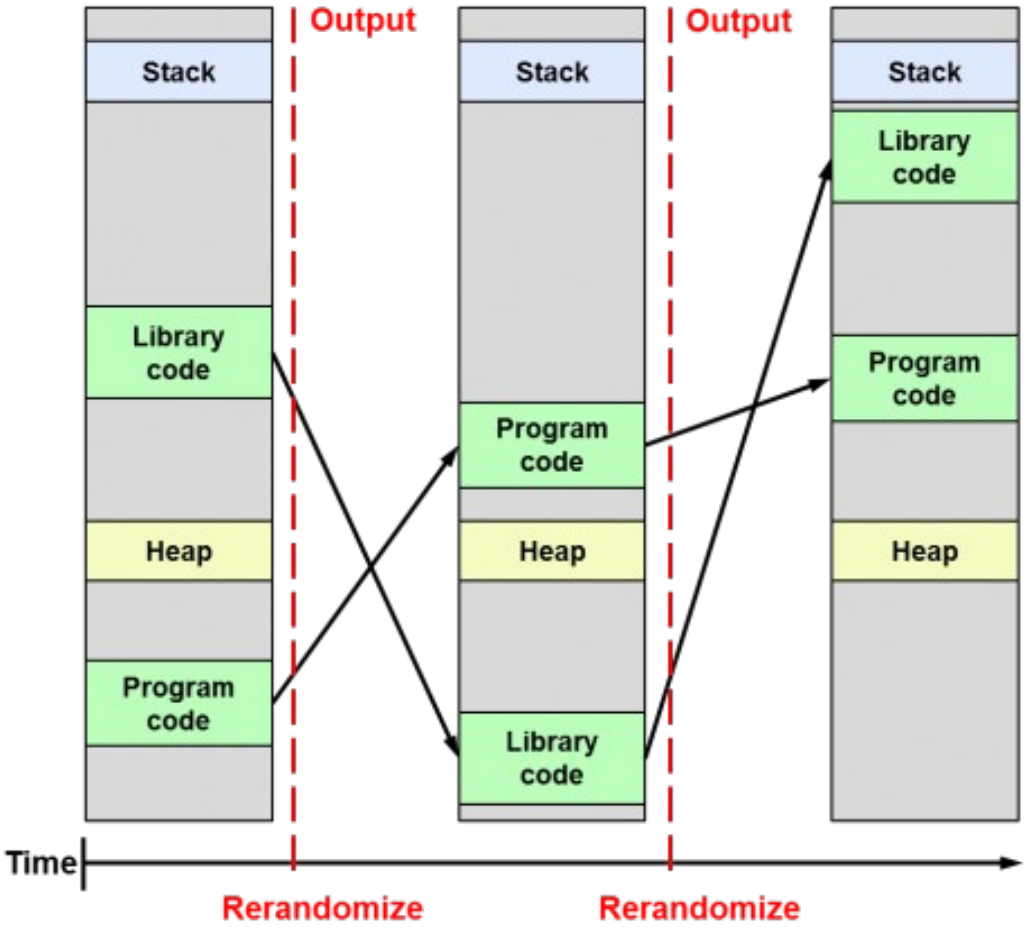
Sections

- Purely optional
- Metadata to divide a segment
- Can be loaded into memory but doesn't have to



Img: https://upload.wikimedia.org

Address Space Layout Randomization



Randomized base addresses of shared libraries and segments of the binary and stack

Img: http://www.daniloaz.com/en/differences-between-aslr-kaslr-and-karl/

Address Space Layout Randomization

Dynamic Relocations

Normal x64 relocation struct

```
4 typedef struct {
5         uint64_t   r_offset;
6         uint64_t   r_info;
7 } Elf64_Rel;
```

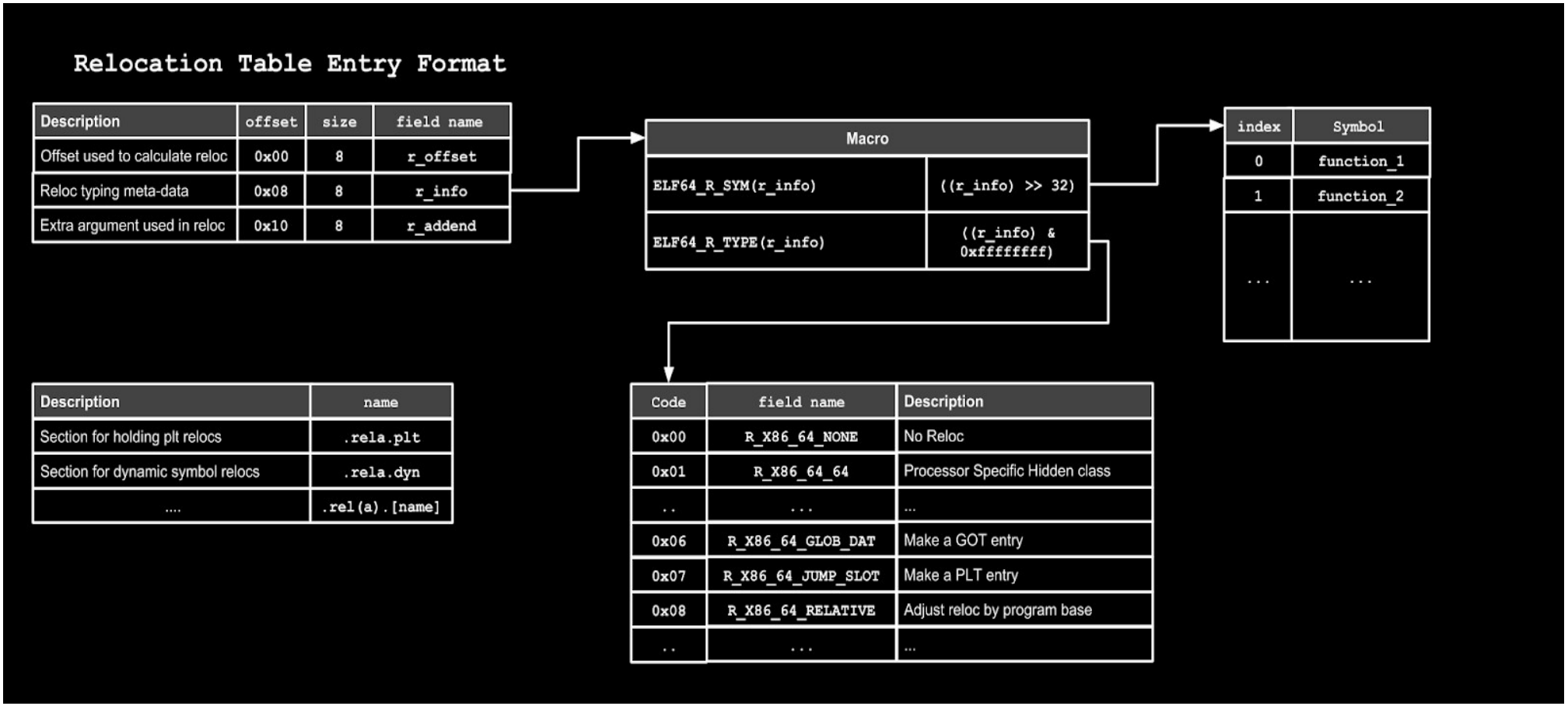
Relocation struct with addend

```
4 typedef struct {
5         uint64_t   r_offset;
6         uint64_t   r_info;
7         int64_t r_addend;
8 } Elf64_Rela;
```

- Relocations patch the binary on load time
- Many different relocation types exists
- Only a few ever get used
- The other ones are mostly unknown

More info: Oracle relocation list

Dynamic Relocations



Note: Upper 4 bytes of r_info are the index, the lower 4 bytes are for the type

Img: Relocation Cheat Sheet

The beginnings

My goal:

Building a simple ELF obfuscator for ASLR binaries...

Which led to:

Oh nooo,
My debugger broke (｡◕‿◕｡)

0Pack in action

So what happened here?

Entrypoint is **0**

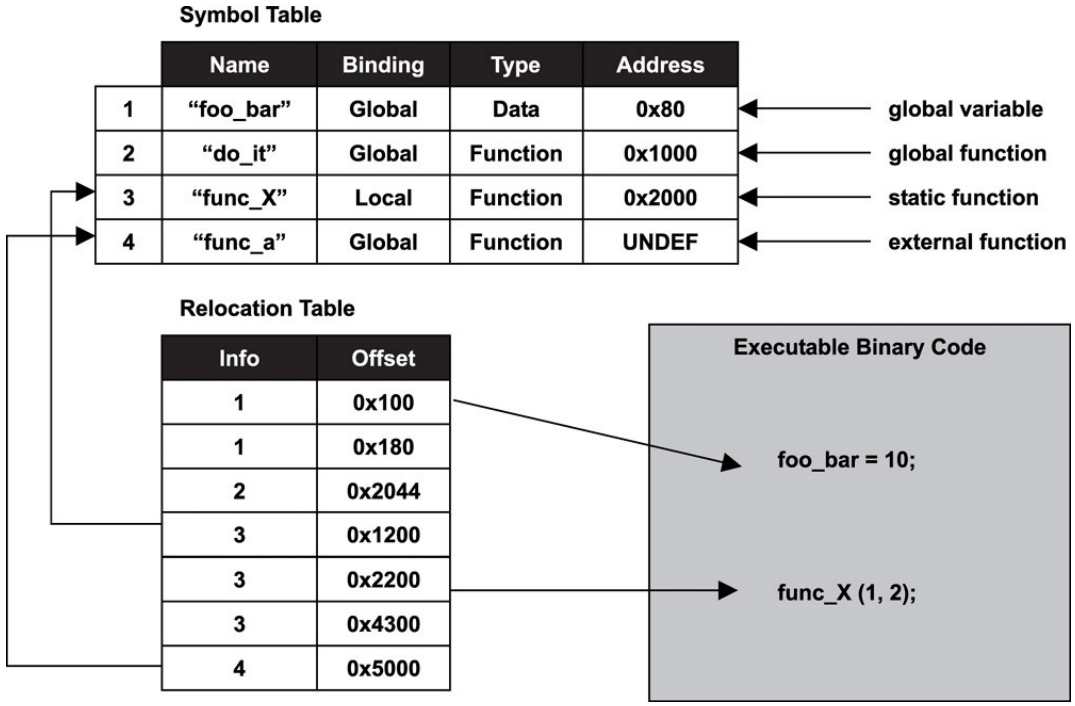
Debugger checks if **entrypoint** is **NULL**

If true

They **parse** the **symbol table** to locate the main function

Sets the entrypoint to the **main function**

The debugger sets the **breakpoint** at the **wrong address**!



Img: http://www.embeddedlinux.org

So how do we get a working executable out of this?

Execution now starts here

```
typedef struct
{
  unsigned char      e_ident[EI_NIDENT];
  Elf64_Half         e_type;
  Elf64_Half         e_machine;
  Elf64_Word         e_version;
  Elf64_Addr         e_entry;
  Elf64_Off          e_phoff;
  Elf64_Off          e_shoff;
  Elf64_Word         e_flags;
  Elf64_Half         e_ehsize;
  Elf64_Half         e_phentsize;
  Elf64_Half         e_phnum;
  Elf64_Half         e_shentsize;
  Elf64_Half         e_shnum;
  Elf64_Half         e_shstrndx;
} Elf64_Ehdr;
```

First idea

ELF Header seen as code

```
,=< 0x00000000      7f45            jg 0x47
|   0x00000002      4c460201        add r8b, byte [rcx]
|   0x00000006      0100            add dword [rax], eax
|   0x00000008      0000            add byte [rax], al
|   0x0000000a      0000            add byte [rax], al
|   0x0000000c      0000            add byte [rax], al
|   0x0000000e      0000            add byte [rax], al
|   0x00000010      0300            add eax, dword [rax]
|   0x00000012      3e0001          add byte ds:[rcx], al
|   0x00000015      0000            add byte [rax], al
|   0x00000017      00e0            add al, ah
|   0x00000019      06              invalid
|   0x0000001a      0000            add byte [rax], al
|   0x0000001c      0000            add byte [rax], al
|   0x0000001e      0000            add byte [rax], al
|   0x00000020      400000          add byte [rax], al
|   0x00000023      0000            add byte [rax], al
|   0x00000025      0000            add byte [rax], al
|   0x00000027      00602f          add byte [rax + 0x2f], ah
|   0x0000002a      0000            add byte [rax], al
|   0x0000002c      0000            add byte [rax], al
|   0x0000002e      0000            add byte [rax], al
|   0x00000030      0000            add byte [rax], al
|   0x00000032      0000            add byte [rax], al
|   0x00000034      400038          add byte [rax], dil
|   0x00000037      000a            add byte [rdx], cl
|   0x00000039      004000          add byte [rax], al
|   0x0000003c      2300            and eax, dword [rax]
|   0x0000003e      2200            and al, byte [rax]
|   ;-- segment.PHDR:
|   0x00000040      06              invalid
|   0x00000041      0000            add byte [rax], al
|   0x00000043      000400          add byte [rax + rax], al
|   0x00000046      0000            add byte [rax], al
    0x00000048      400000          add byte [rax], al
```

0x7f45 => jg 0x47

- ELF magic translates to indirect jump
- Jumps into program header
- Change program header value to another jump?

Solution

Table 4.10: Relocation Types

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	*word64*	S
R_X86_64_JUMP_SLOT	7	*word64*	S

Writes the
Symbol value + **A**ddend
to the target location

The symbol value can be set
to 0

Relocations now
can overwrite any bytes!

Relocation struct with addend

```
4 typedef struct {
5         uint64_t   r_offset;
6         uint64_t   r_info;
7         int64_t r_addend;
8 } Elf64_Rela;
```


ELF ABI: https://refspecs.linuxfoundation.org

0Pack inner workings

Set entrypoint to 0

Set first segment to RWX

(*) Append code that erases shellcode after Execution

Append a relative jmp to original entrypoint

Convert shellcode to R_X86_64_64 Relocations

* Not yet implemented in the poc

Any Questions?