



Machine Language

A paradox



grader.c

```
enum {BUFSIZE = 48};
char grade = 'D';
char name[BUFSIZE];

/* Read a string into s */
void readString(char *s) {
    char buf[BUFSIZE];
    int i = 0; int c;

    /* Read string into buf[] */
    for (;;) {
        c = fgetc(stdin);
        if (c == EOF || c == '\n')
            break;
        buf[i] = c;
        i++;
    }
    /* Copy buf[] to s[] */
    buf[i] = '\0';
    for (i = 0; i < BUFSIZE; i++)
        s[i] = buf[i];
}
```

```
int main(void) {
    printf("What is your name?\n");
    readString(name);
    if (strcmp(name, "Andrew") == 0)
        grade = 'B';
    printf("%c is your grade, %s.\n",
           grade, name);
    return 0;
}
```

What is your name?
Bob
D is your grade, Bob.

What is your name?
Andrew
B is your grade, Andrew.

What is your name?
[fill in something here]
A is your grade, Susan.

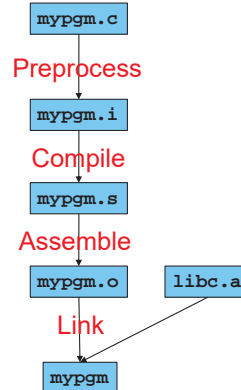
Machine language



This lecture is about

- machine language (in general)
- x86-64 machine language (in particular)
- The assembly and linking processes
- Amusing and important applications to computer security (and therefore, Programming Assignment 5, Buffer Overrun)

The Build Process



Covered in COS 320:
Compiling Techniques

Covered here

Instruction Set Architecture (ISA)



There are many kinds of computer chips out there:

- Intel x86 series
- IBM PowerPC
- ARM
- RISC-V
- MIPS

Each of these different “machine architectures” understands a different machine language

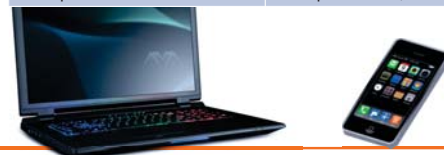
(and, in the old days, dozens more)

CISC and RISC styles of machine language



CISC	RISC
Complex, powerful instructions	Simple do-only-one-thing instructions
Many memory addressing modes (direct, indirect, base+displacement, indexed, scaled indexed)	Few memory addressing modes (typically only base+displacement)
Hardware interpretation is complex	Hardware interpretation is simple
Need relatively few instructions to accomplish a given job	Need more instructions to accomplish a given job
Example: x86-64	Examples: ARM, PowerPC

Energy efficient, battery lasts longer!



Agenda



x86-64 Machine Language

Buffer overrun vulnerabilities

x86-64 Machine Language after Assembly

x86-64 Machine Language after Linking

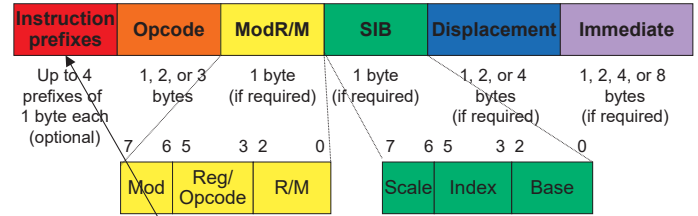
Assembly Language: `addq %rax, %rbx`

Machine Language: `01001000 00000001 11000011`

x86-64 Instruction Format



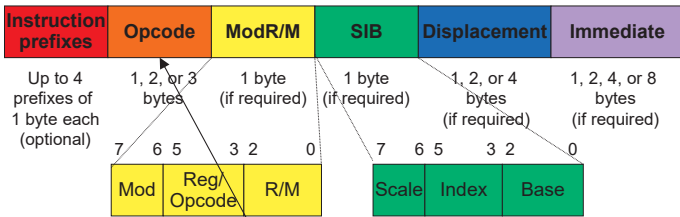
Difficult to generalize about x86-64 instruction format; many instructions use this format



Instruction prefix

- Sometimes a repeat count
- Rarely used; don't be concerned

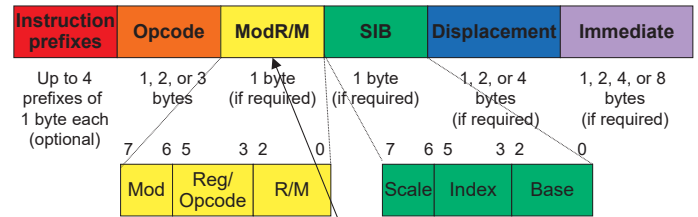
x86-64 Instruction Format (cont.)



Opcode

- Specifies which operation should be performed
 - Add, move, call, etc.
- Sometimes specifies additional (or less) information

x86-64 Instruction Format (cont.)



ModR/M (register mode, register/opcode, register/memory)

- Specifies types of operands (immediate, register, memory)
- Specifies sizes of operands (byte, word, long)
- Sometimes contains an extension of the opcode

x86-64 Instruction Format (cont.)



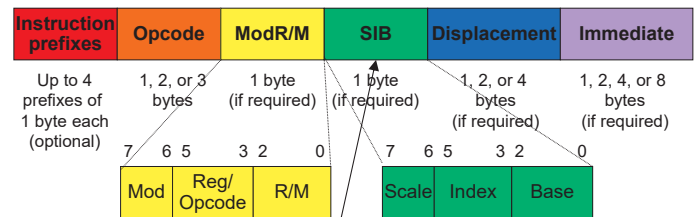
Sometimes 3 bits in ModR/M byte, along with extra bit in another field, specify a register

- For 8-byte registers:

Extra	ModR/M	Register
0	000	RAX
0	001	RCX
0	010	RDY
0	011	RBX
0	100	RSP
0	101	RBP
0	110	RSI
0	111	RDI
1	000	R8
1	001	R9
1	010	R10
1	011	R11
1	100	R12
1	101	R13
1	110	R14
1	111	R15

Similar mappings exist for 4-byte, 2-byte and 1-byte registers

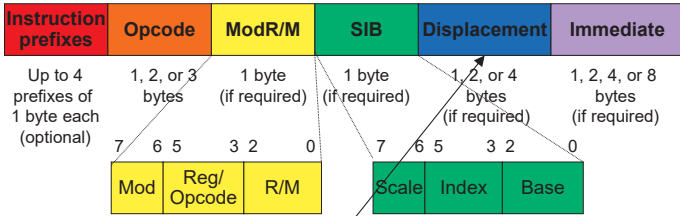
x86-64 Instruction Format (cont.)



SIB (scale, index, base)

- Used when one of the operands is a memory operand that uses a **s**cale, an **i**ndex register, and/or a **b**ase register

x86-64 Instruction Format (cont.)



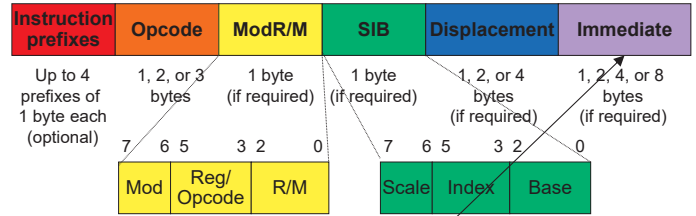
Displacement

- Part of memory operand, or...
- In jump and call instructions, indicates the displacement between the destination instruction and the jump/call instruction
 - More precisely, indicates:

$$[\text{addr of destination instr}] - [\text{addr of instr following the jump/call}]$$
- Uses little-endian byte order

15

x86-64 Instruction Format (cont.)



Immediate

- Specifies an immediate operand
- Uses little-endian byte order

14

Example 1



Assembly lang: `addq %rax, %rbx`
 Machine lang: `4801c3`
 Explanation:

`01001000 00000001 11000011`

Opcode: This is an add instruction whose src operand is an 8-byte register or memory operand and whose dest operand is a 8-byte register

ModR/M: The M field of the ModR/M byte designates a register
 ModR/M: The src register is RAX
 ModR/M: The dest register is RBX

Extra	ModR/M	Register
0	000	RAX/EAX
0	001	RCX/ECX
0	010	RDX/EDX
0	011	RBX/EBX
0	100	RSP/ESP
0	101	RBP/EBP
0	110	RSI/ESI
0	111	RDI/EDI

Observation: Sometimes opcode specifies operation (e.g. add) and format(s) of operand(s)

15

Example 2



Assembly lang: `movl $1, %ebx`
 Machine lang: `bb01000000`
 Explanation:

`10111011 00000001 00000000 00000000 00000000`

Opcode: This is a mov instruction whose src operand is a 4-byte immediate

Opcode: the destination operand is the EBX register
 Immediate: The immediate operand is 1

Observation: Sometimes opcode specifies operation and operand(s)
 Observation: Immediate operands are in little-endian byte order

16

Examples 3, 4



Assembly lang: `pushq %rax`
 Machine lang: `50`
 Explanation:

`01010000`
 Opcode: This is a pushq %rax instruction

Assembly lang: `pushq %rcx`
 Machine lang: `51`
 Explanation:

`01010001`
 Opcode: This is a pushq %rcx instruction

Observation: Sometimes opcode specifies operation and operand(s)
 Observation: pushq is used often, so is optimized into 1 byte

17

Example 5



Assembly lang: `movl -8(%eax,%ebx,4), %edx`
 Machine lang: `678b5498f8`
 Explanation:

`10100111 10001011 01010100 10011000 11111000`

Opcode: This is a mov instruction whose src operand is a 4-byte register or memory operand and whose dest operand is a 4-byte register

ModR/M: The src operand is a register, the dest operand is of the form disp(base,index,scale), the base and index registers are 4-byte registers, and the disp is one-byte
 ModR/M: The destination register is EDX

SIB: The scale is 4
 SIB: The index register is EBX

SIB: The base reg is EAX
 Displacement: The disp is -8

Observation: Two's complement notation
 Observation: Complicated!!!

18

Agenda



x86-64 Machine Language

Buffer overrun vulnerabilities

x86-64 Machine Language after Assembly

x86-64 Machine Language after Linking

A program



```
% a.out
What is your name?
John Smith
Thank you, John Smith.
%
```

```
#include <stdio.h>
int main(int argc, char **argv) {
    char name[12]; int i;
    printf("What is your name?\n");
    for (i=0; ; i++) {
        int c = getchar();
        if (c=='\n' || c ==EOF) break;
        name[i] = c;
    }
    name[i]='\0';
    printf("Thank you, %s.\n", name);
    return 0;
}
```

Why did this program crash?



```
% a.out
What is your name?
adsli57asdkhj5jklids;ahj5;klsaduj5klysdukl5aujksd5ukals;5uj;akukla
Segmentation fault
%
```

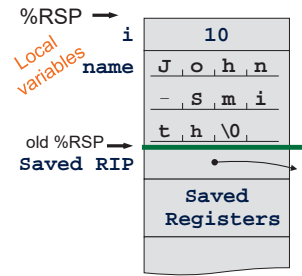
```
#include <stdio.h>
int main(int argc, char **argv) {
    char name[12]; int i;
    printf("What is your name?\n");
    for (i=0; ; i++) {
        int c = getchar();
        if (c=='\n' || c ==EOF) break;
        name[i] = c;
    }
    name[i]='\0';
    printf("Thank you, %s.\n", name);
    return 0;
}
```

Stack frame layout



```
% a.out
What is your name?
John Smith
Thank you, John Smith.
%
```

```
#include <stdio.h>
int main(int argc, char **argv) {
    char name[12]; int i;
    printf("What is your name?\n");
    for (i=0; ; i++) {
        int c = getchar();
        if (c=='\n' || c ==EOF) break;
        name[i] = c;
    }
    name[i]='\0';
    printf("Thank you, %s.\n", name);
    return 0;
}
```

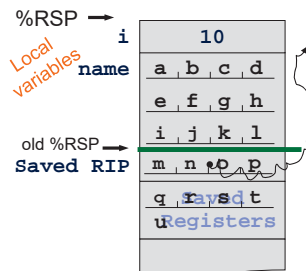


Buffer overrun



```
% a.out
What is your name?
abcdefghijklmnopqrstu
Segmentation fault
%
```

```
#include <stdio.h>
int main(int argc, char **argv) {
    char name[12]; int i;
    printf("What is your name?\n");
    for (i=0; ; i++) {
        int c = getchar();
        if (c=='\n' || c ==EOF) break;
        name[i] = c;
    }
    name[i]='\0';
    printf("Thank you, %s.\n", name);
    return 0;
}
```

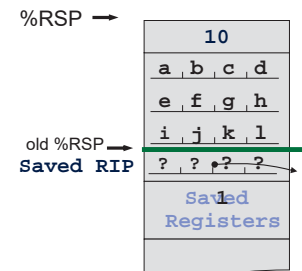


Innocuous? buffer overrun



```
% a.out
What is your name?
abcdefghijklmnopkl????^A^V^V^V
%
```

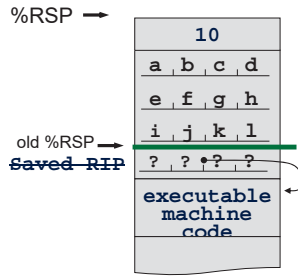
```
#include <stdio.h>
int main(int argc, char **argv) {
    char name[12]; int i;
    printf("What is your name?\n");
    for (i=0; ; i++) {
        int c = getchar();
        if (c=='\n' || c ==EOF) break;
        name[i] = c;
    }
    name[i]='\0';
    printf("Thank you, %s.\n", name);
    return 0;
}
```



Cleverly malicious? Buffer overrun Maliciously clever?



```
% a.out
What is your name?
abcdefghijkl????executable-machine-code...
How may I serve you, master?
%
#include <stdio.h>
int main(int argc, char **argv) {
    char name[12]; int i;
    printf("What is your name?\n");
    for (i=0; ; i++) {
        int c = getchar();
        if (c=='\n' || c == EOF) break;
        name[i] = c;
    }
    name[i]='\0';
    printf("Thank you, %s.\n", name);
    return 0;
}
```



Attacking a web server

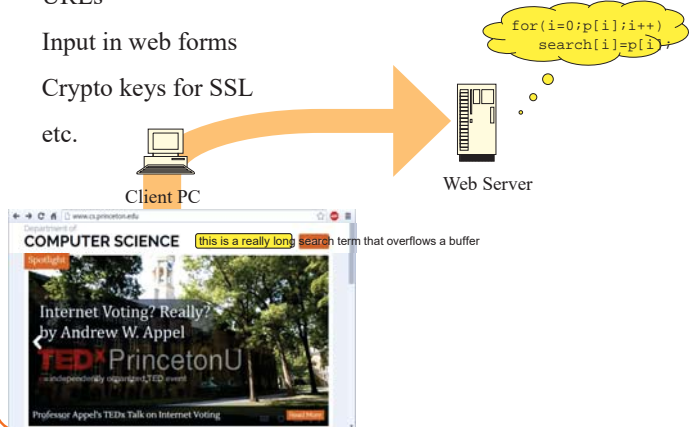


URLs

Input in web forms

Crypto keys for SSL

etc.



Attacking a web browser



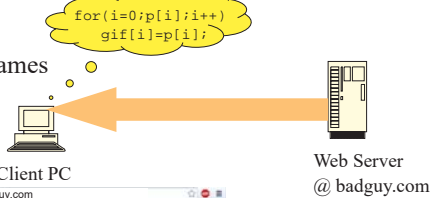
HTML keywords

Images

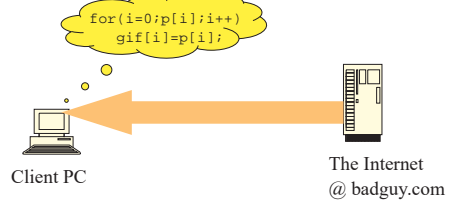
Image names

URLs

etc.



Attacking everything in sight



E-mail client

PDF viewer

Operating-system kernel

TCP/IP stack

Any application that ever sees input directly from the outside

Defenses against this attack



Best: program in languages that make array-out-of-bounds impossible (Java, C#, ML, python,)

If you must program in C: use discipline *and software analysis tools* in C programming always to check bounds of array subscripts

Otherwise, stopgap security patches:

- Operating system randomizes initial stack pointer
- "No-execute" memory permission
- "Canaries" at end of stack frames

Not a single one of these would have prevented the "Heartbleed" attack



Your programming assignment: Attack the "grader" program



```
enum {BUFSIZE = 48};
char grade = 'D';
char name[BUFSIZE];

/* Read a string into s */
void readString(char *s) {
    char buf[BUFSIZE];
    int i = 0; int c;

    /* Read string into buf[] */
    for (;;) {
        c = fgetc(stdin);
        if (c == EOF || c == '\n')
            break;
        buf[i] = c;
        i++;
    }
    /* Copy buf[] to s[] */
    buf[i] = '\0';
    for (i = 0; i < BUFSIZE; i++)
        s[i] = buf[i];
}
```

```
int main(void) {
    printf("What is your name?\n");
    readString(name);
    if (strcmp(name, "Andrew") == 0)
        grade = 'B';
    printf("%c is your grade, %s.\n",
           grade, name);
    return 0;
}
```

What is your name?
Bob
D is your grade, Bob.

What is your name?
Andrew
B is your grade, Andrew.

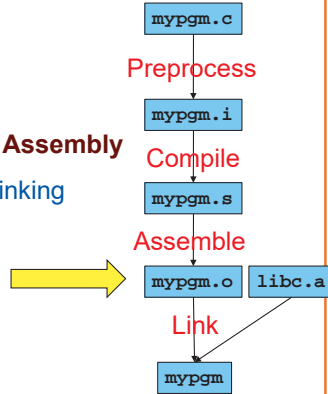
What is your name?
Susan0?!*!????*???!*!%?!?!(*%(*^*?
A is your grade, Susan.

Agenda



x86-64 Machine Language
Buffer overrun vulnerabilities

x86-64 Machine Language after Assembly
x86-64 Machine Language after Linking



An Example Program



A simple (nonsensical) program:

```

#include <stdio.h>
int main(void)
{
    printf("Type a char: ");
    if (getchar() == 'A')
        printf("Hi\n");
    return 0;
}
  
```

```

.section ".rodata"
msg1: .string "Type a char"
msg2: .string "Hi\n"
.section ".text"
.globl main

main:
    movl    $0, %eax
    movq   $msg1, %rdi
    call   printf
    call   getchar
    cmpl   $'A', %eax
    jne   skip
    movl   $0, %eax
    movq   $msg2, %rdi
    call   printf

skip:
    movl   $0, %eax
    ret
  
```

Let's consider the machine lang equivalent after assembly...

Examining Machine Lang: RODATA



Assemble program; run objdump

```

$ gcc217 -c detecta.s
$ objdump --full-contents --section .rodata detecta.o

detecta.o:      file format elf64-x86-64

Contents of section .rodata:
0000 54797065 20612063 6861723a 20004869  Type a char: .Hi
0010 0a00
  
```

Offsets Contents

- Assembler does not know **addresses**
- Assembler knows only **offsets**
- **"Type a char"** starts at offset 0
- **"Hi\n"** starts at offset 0e

Examining Machine Lang: TEXT



Assemble program; run objdump

```

$ gcc217 -c detecta.s
$ objdump --disassemble --reloc detecta.o
detecta.o:      file format elf64-x86-64

Disassembly of section .text:
0000000000000000 <main>:
0:  b8 00 00 00 00      mov     $0x0,%eax
5:  48 c7 c7 00 00 00 00  mov     $0x0,%rdi
c:  e8 00 00 00 00      callq  11 <main+0x11>
11: e8 00 00 00 00      callq  16 <main+0x16>
16: 83 f8 41             cmp     $0x41,%eax
19: 75 11                jne    2c <skip>
1b: b8 00 00 00 00      mov     $0x0,%eax
20: 48 c7 c7 00 00 00 00  mov     $0x0,%rdi
27:  e8 00 00 00 00      callq  2c <skip>
28: R_X86_64_PC32      callq  2c <skip>
2c:  b8 00 00 00 00      mov     $0x0,%eax
31:  c3                  retq

000000000000002c <skip>:
2c:  b8 00 00 00 00      mov     $0x0,%eax
31:  c3                  retq
  
```

Offsets
Machine language
Relocation records
Assembly language

Let's examine one line at a time...

movl \$0, %eax



```

$ gcc217 -c detecta.s
$ objdump --disassemble --reloc detecta.o
detecta.o:      file format elf64-x86-64

Disassembly of section .text:
0000000000000000 <main>:
0:  b8 00 00 00 00      mov     $0x0,%eax
5:  48 c7 c7 00 00 00 00  mov     $0x0,%rdi
c:  e8 00 00 00 00      callq  11 <main+0x11>
11: e8 00 00 00 00      callq  16 <main+0x16>
16: 83 f8 41             cmp     $0x41,%eax
19: 75 11                jne    2c <skip>
1b: b8 00 00 00 00      mov     $0x0,%eax
20: 48 c7 c7 00 00 00 00  mov     $0x0,%rdi
27:  e8 00 00 00 00      callq  2c <skip>
28: R_X86_64_PC32      callq  2c <skip>
2c:  b8 00 00 00 00      mov     $0x0,%eax
31:  c3                  retq

000000000000002c <skip>:
2c:  b8 00 00 00 00      mov     $0x0,%eax
31:  c3                  retq
  
```

movl \$0, %eax



Assembly lang: movl \$0, %eax
Machine lang: b800000000
Explanation:

10111000 00000000 00000000 00000000 00000000

Opcode: This is a mov instruction whose src operand is a 4-byte immediate

Opcode: the destination operand is the EAX register
Immediate: The immediate operand is 0

movq \$msg1, %rdi



```
$ gcc217 -c detecta.s
$ objdump --disassemble --reloc detecta.o
detecta.o:      file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
0:   b8 00 00 00 00          mov     $0x0,%eax
5:   48 c7 c7 00 00 00 00    mov     $0x0,%rdi
                                8: R_X86_64_32S      .rodata
c:   e8 00 00 00 00          callq  11 <main+0x11>
                                d: R_X86_64_PC32    printf-0x4
11:  e8 00 00 00 00          callq  16 <main+0x16>
                                12: R_X86_64_PC32    getchar-0x4
16:  83 f8 41                cmp     $0x41,%eax
19:  75 11                    jne    2c <skip>
1b:  b8 00 00 00 00          mov     $0x0,%eax
20:  48 c7 c7 00 00 00 00    mov     $0x0,%rdi
                                23: R_X86_64_32S      .rodata+0xe
27:  e8 00 00 00 00          callq  2c <skip>
                                28: R_X86_64_PC32    printf-0x4

000000000000002c <skip>:
2c:  b8 00 00 00 00          mov     $0x0,%eax
31:  c3                      retq


```

37

movq \$msg1, %rdi



Assembly lang: movq \$msg1, %rdi
Machine lang: 48 C7 C7 00 00 00 00
Explanation:

```
01001000 11000111 110010111 00000000 00000000 00000000 00000000
Opcode: This is a movq instruction with a 4-byte immediate
source operand and a 8 byte register destination operand
Opcode: The destination register is RDI
Opcode: The destination register is
RDI (cont.)
Disp: The immediate(memory address)
is 0
```

- movq must contain an **address**
- Assembler knew **offset** marked by msg1
 - msg1 marks offset 0 relative to beginning of RODATA section
- But assembler did not know address of RODATA section!
- So assembler didn't know **address** marked by msg1
- So assembler couldn't generate this instruction completely

38

Relocation Record 1



```
$ gcc217 -c detecta.s
$ objdump --disassemble --reloc detecta.o
detecta.o:      file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
0:   b8 00 00 00 00          mov     $0x0,%eax
5:   48 c7 c7 00 00 00 00    mov     $0x0,%rdi
                                8: R_X86_64_32S      .rodata
c:   e8 00 00 00 00          callq  11 <main+0x11>
                                d: R_X86_64_PC32    printf-0x4
11:  e8 00 00 00 00          callq  16 <main+0x16>
                                12: R_X86_64_PC32    getchar-0x4
16:  83 f8 41                cmp     $0x41,%eax
19:  75 11                    jne    2c <skip>
1b:  b8 00 00 00 00          mov     $0x0,%eax
20:  48 c7 c7 00 00 00 00    mov     $0x0,%rdi
                                23: R_X86_64_32S      .rodata+0xe
27:  e8 00 00 00 00          callq  2c <skip>
                                28: R_X86_64_PC32    printf-0x4

000000000000002c <skip>:
2c:  b8 00 00 00 00          mov     $0x0,%eax
31:  c3                      retq


```

39

Relocation Record 1



8: R_X86_64_32S .rodata

This part is always the same,
it's the name of the machine architecture!

Dear Linker,

Please patch the TEXT section at offset
08_H. Patch in a **32-bit, Signed value**. When
you determine the addr of the RODATA
section, place that address in the
TEXT section at the prescribed place.

Sincerely,
Assembler

40

call printf



```
$ gcc217 -c detecta.s
$ objdump --disassemble --reloc detecta.o
detecta.o:      file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
0:   b8 00 00 00 00          mov     $0x0,%eax
5:   48 c7 c7 00 00 00 00    mov     $0x0,%rdi
                                8: R_X86_64_32S      .rodata
c:   e8 00 00 00 00          callq  11 <main+0x11>
                                d: R_X86_64_PC32    printf-0x4
11:  e8 00 00 00 00          callq  16 <main+0x16>
                                12: R_X86_64_PC32    getchar-0x4
16:  83 f8 41                cmp     $0x41,%eax
19:  75 11                    jne    2c <skip>
1b:  b8 00 00 00 00          mov     $0x0,%eax
20:  48 c7 c7 00 00 00 00    mov     $0x0,%rdi
                                23: R_X86_64_32S      .rodata+0xe
27:  e8 00 00 00 00          callq  2c <skip>
                                28: R_X86_64_PC32    printf-0x4

000000000000002c <skip>:
2c:  b8 00 00 00 00          mov     $0x0,%eax
31:  c3                      retq


```

41

call printf



Assembly lang: call printf
Machine lang: e8 00 00 00 00
Explanation:

```
11101000 00000000 00000000 00000000 00000000
Opcode: This is a call instruction with a 4-byte
displacement
Disp: The displacement is 00000000H (0)
```

- call must contain a **displacement**
- Assembler had to generate the displacement:
[addr of printf] – [addr after call instr]
- But assembler didn't know addr of printf
 - printf isn't even present yet!
- So assembler couldn't generate this instruction completely

42



Relocation Record 2

```

$ gcc217 -c detecta.s
$ objdump --disassemble --reloc detecta.o
detecta.o:      file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
 0:  b8 00 00 00 00      mov     $0x0,%eax
 5:  48 c7 c7 00 00 00 00  mov     $0x0,%rdi
                        8:  R_X86_64_32S      .rodata
   c:  e8 00 00 00 00      callq  11 <main+0x11>
                        d:  R_X86_64_PC32     printf-0x4
 11: e8 00 00 00 00      callq  16 <main+0x16>
                        12: R_X86_64_PC32     getchar-0x4
 16: 83 f8 41           cmp     $0x41,%eax
 19: 75 11             jne    2c <skip>
 1b: b8 00 00 00 00      mov     $0x0,%eax
 20: 48 c7 c7 00 00 00 00  mov     $0x0,%rdi
                        23: R_X86_64_32S      .rodata+0xe
 27: e8 00 00 00 00      callq  2c <skip>
                        28: R_X86_64_PC32     printf-0x4

000000000000002c <skip>:
 2c:  b8 00 00 00 00      mov     $0x0,%eax
 31:  c3                retq


```



Relocation Record 2

d: R_X86_64_PC32 printf-0x4

This part is always the same, it's the name of the machine architecture!

Dear Linker,

Please patch the TEXT section at offset 0d_H. Patch in a 32-bit "PC-relative" value. When you determine the addr of printf, compute [addr of printf] - [addr after call] and place the result at the prescribed place.

Sincerely,
Assembler



call getchar

```

$ gcc217 -c detecta.s
$ objdump --disassemble --reloc detecta.o
detecta.o:      file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
 0:  b8 00 00 00 00      mov     $0x0,%eax
 5:  48 c7 c7 00 00 00 00  mov     $0x0,%rdi
                        8:  R_X86_64_32S      .rodata
   c:  e8 00 00 00 00      callq  11 <main+0x11>
                        d:  R_X86_64_PC32     printf-0x4
 11: e8 00 00 00 00      callq  16 <main+0x16>
                        12: R_X86_64_PC32     getchar-0x4
 16: 83 f8 41           cmp     $0x41,%eax
 19: 75 11             jne    2c <skip>
 1b: b8 00 00 00 00      mov     $0x0,%eax
 20: 48 c7 c7 00 00 00 00  mov     $0x0,%rdi
                        23: R_X86_64_32S      .rodata+0xe
 27: e8 00 00 00 00      callq  2c <skip>
                        28: R_X86_64_PC32     printf-0x4

000000000000002c <skip>:
 2c:  b8 00 00 00 00      mov     $0x0,%eax
 31:  c3                retq


```



call getchar

Assembly lang: call getchar
Machine lang: e8 00 00 00 00
Explanation:

11101000 00000000 00000000 00000000 00000000
Opcode: This is a call instruction with a 4-byte displacement
Disp: The displacement is 00000000_H (0)

- call must contain a displacement
- Assembler had to generate the displacement: [addr of getchar] - [addr after call instr]
- But assembler didn't know addr of getchar
 - getchar isn't even present yet!
- So assembler couldn't generate this instruction completely



Relocation Record 3

```

$ gcc217 -c detecta.s
$ objdump --disassemble --reloc detecta.o
detecta.o:      file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
 0:  b8 00 00 00 00      mov     $0x0,%eax
 5:  48 c7 c7 00 00 00 00  mov     $0x0,%rdi
                        8:  R_X86_64_32S      .rodata
   c:  e8 00 00 00 00      callq  11 <main+0x11>
                        d:  R_X86_64_PC32     printf-0x4
 11: e8 00 00 00 00      callq  16 <main+0x16>
                        12: R_X86_64_PC32     getchar-0x4
 16: 83 f8 41           cmp     $0x41,%eax
 19: 75 11             jne    2c <skip>
 1b: b8 00 00 00 00      mov     $0x0,%eax
 20: 48 c7 c7 00 00 00 00  mov     $0x0,%rdi
                        23: R_X86_64_32S      .rodata+0xe
 27: e8 00 00 00 00      callq  2c <skip>
                        28: R_X86_64_PC32     printf-0x4

000000000000002c <skip>:
 2c:  b8 00 00 00 00      mov     $0x0,%eax
 31:  c3                retq


```



Relocation Record 3

12: R_X86_64_PC32 getchar-0x4

Dear Linker,

Please patch the TEXT section at offsets 12_H. Do a 32-bit PC-relative patch. When you determine the addr of getchar, compute [offset of getchar] - [addr after call] and place the result at the prescribed place.

Sincerely,
Assembler

cmpl '\$A', %eax



```
$ gcc217 -c detecta.s
$ objdump --disassemble --reloc detecta.o
detecta.o:      file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
 0:  b8 00 00 00 00      mov     $0x0,%eax
 5:  48 c7 c7 00 00 00 00  mov     $0x0,%rdi
                        8:  R_X86_64_32S      .rodata
 c:  e8 00 00 00 00      callq  11 <main+0x11>
                        d:  R_X86_64_PC32     printf-0x4
11:  e8 00 00 00 00      callq  16 <main+0x16>
                        12: R_X86_64_PC32     getchar-0x4
16:  83 f8 41             cmpl   $0x41,%eax
19:  75 11             jne    2c <skip>
1b:  b8 00 00 00 00      mov     $0x0,%eax
20:  48 c7 c7 00 00 00 00  mov     $0x0,%rdi
                        23: R_X86_64_32S      .rodata+0xe
27:  e8 00 00 00 00      callq  2c <skip>
                        28: R_X86_64_PC32     printf-0x4

000000000000002c <skip>:
2c:  b8 00 00 00 00      mov     $0x0,%eax
31:  c3                 retq


```

49

cmpl '\$A', %eax



Assembly lang: cmpl '\$A', %eax
Machine lang: 83 f8 41
Explanation:

10000011 11111000 01000001

Opcode: This is an instruction whose source operand is a one-byte immediate and whose destination operand is a register or memory

ModR/M: This is a cmpl instruction, and the last three bytes of the ModR/M field specify the destination register

ModR/M: The dest register is EAX

The immediate operand is 41_H ('A')

50

jne skip



```
$ gcc217 -c detecta.s
$ objdump --disassemble --reloc detecta.o
detecta.o:      file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
 0:  b8 00 00 00 00      mov     $0x0,%eax
 5:  48 c7 c7 00 00 00 00  mov     $0x0,%rdi
                        8:  R_X86_64_32S      .rodata
 c:  e8 00 00 00 00      callq  11 <main+0x11>
                        d:  R_X86_64_PC32     printf-0x4
11:  e8 00 00 00 00      callq  16 <main+0x16>
                        12: R_X86_64_PC32     getchar-0x4
16:  83 f8 41             cmpl   $0x41,%eax
19:  75 11             jne    2c <skip>
1b:  b8 00 00 00 00      mov     $0x0,%eax
20:  48 c7 c7 00 00 00 00  mov     $0x0,%rdi
                        23: R_X86_64_32S      .rodata+0xe
27:  e8 00 00 00 00      callq  2c <skip>
                        28: R_X86_64_PC32     printf-0x4

000000000000002c <skip>:
2c:  b8 00 00 00 00      mov     $0x0,%eax
31:  c3                 retq


```

51

jne skip



Assembly lang: jne skip
Machine lang: 75 11
Explanation:

01110101 00010001

Opcode: This is a jne instruction with a one-byte displacement

Disp: The displacement is 11_H (17_D)

- jne must contain a **displacement**
- Assembler had to generate the displacement:
[addr of skip] - [addr after jne instr]
Assembler **did** know addr of skip
- So assembler **could** generate this instruction completely
 $2c_H - 1b_H = 11_H = 17_D$

52

jne skip



Is it clear why jump and call instructions contain displacements instead of addresses?

53

movl \$0, %eax



```
$ gcc217 -c detecta.s
$ objdump --disassemble --reloc detecta.o
detecta.o:      file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
 0:  b8 00 00 00 00      mov     $0x0,%eax
 5:  48 c7 c7 00 00 00 00  mov     $0x0,%rdi
                        8:  R_X86_64_32S      .rodata
 c:  e8 00 00 00 00      callq  11 <main+0x11>
                        d:  R_X86_64_PC32     printf-0x4
11:  e8 00 00 00 00      callq  16 <main+0x16>
                        12: R_X86_64_PC32     getchar-0x4
16:  83 f8 41             cmpl   $0x41,%eax
19:  75 11             jne    2c <skip>
1b:  b8 00 00 00 00      mov     $0x0,%eax
20:  48 c7 c7 00 00 00 00  mov     $0x0,%rdi
                        23: R_X86_64_32S      .rodata+0xe
27:  e8 00 00 00 00      callq  2c <skip>
                        28: R_X86_64_PC32     printf-0x4

000000000000002c <skip>:
2c:  b8 00 00 00 00      mov     $0x0,%eax
31:  c3                 retq


```

54

movl \$0, %eax



Assembly lang: movl \$0, %eax
Machine lang: b800000000
Explanation:

```
10111000 00000001 00000000 00000000 00000000
```

Opcode: This is a mov instruction whose src operand is a 4-byte immediate

Opcode: the destination operand is the EAX register
Immediate: The immediate operand is 0

55

movq \$msg2, %rdi



```
$ gcc217 -c detecta.s
$ objdump --disassemble --reloc detecta.o
detecta.o: file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
 0: b8 00 00 00 00      mov     $0x0,%eax
 5: 48 c7 c7 00 00 00 00      mov     $0x0,%rdi
                               8: R_X86_64_32S      .rodata
   c: e8 00 00 00 00      callq  11 <main+0x11>
                               d: R_X86_64_PC32     printf-0x4
 11: e8 00 00 00 00      callq  16 <main+0x16>
                               12: R_X86_64_PC32     getchar-0x4
 16: 83 f8 41           cmp     $0x41,%eax
 19: 75 11           jne    2c <skip>
 1b: b8 00 00 00 00      mov     $0x0,%eax
 20: 48 c7 c7 00 00 00 00      mov     $0x0,%rdi
                               23: R_X86_64_32S      .rodata+0xe
 27: e8 00 00 00 00      callq  2c <skip>
                               28: R_X86_64_PC32     printf-0x4

000000000000002c <skip>:
 2c: b8 00 00 00 00      mov     $0x0,%eax
 31: c3                retq


```

56

movq \$msg2, %rdi



Assembly lang: movq \$msg2, %rdi
Machine lang: 48 C7 C7 00 00 00 00
Explanation:

```
01001000 11000111 110010111 00000000 00000000 00000000 00000000
```

Opcode: This is a movq instruction with a 4-byte immediate source operand and a 8 byte register destination operand

Opcode: The destination register is RDI

Opcode: The destination register is RDI (cont.)

Disp: The immediate(memory address) is 0

- movq must contain an address
- Assembler knew **offset** marked by msg2
 - msg2 marks offset 0e_H relative to beginning of RODATA section
- But assembler did not know address of RODATA section!
- So assembler didn't know **address** marked by msg2
- So assembler couldn't generate this instruction completely

57

Relocation Record 4



```
$ gcc217 -c detecta.s
$ objdump --disassemble --reloc detecta.o
detecta.o: file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
 0: b8 00 00 00 00      mov     $0x0,%eax
 5: 48 c7 c7 00 00 00 00      mov     $0x0,%rdi
                               8: R_X86_64_32S      .rodata
   c: e8 00 00 00 00      callq  11 <main+0x11>
                               d: R_X86_64_PC32     printf-0x4
 11: e8 00 00 00 00      callq  16 <main+0x16>
                               12: R_X86_64_PC32     getchar-0x4
 16: 83 f8 41           cmp     $0x41,%eax
 19: 75 11           jne    2c <skip>
 1b: b8 00 00 00 00      mov     $0x0,%eax
 20: 48 c7 c7 00 00 00 00      mov     $0x0,%rdi
                               23: R_X86_64_32S      .rodata+0xe
 27: e8 00 00 00 00      callq  2c <skip>
                               28: R_X86_64_PC32     printf-0x4

000000000000002c <skip>:
 2c: b8 00 00 00 00      mov     $0x0,%eax
 31: c3                retq


```

58

Relocation Record 4



```
23: R_X86_64_32S .rodata+0xe
```

Dear Linker,

Please patch the TEXT section at offset 23_H. Patch in a 32-bit Signed value. When you determine the addr of the RODATA section, add 0e_H to that address, and place the result in the TEXT section at the prescribed place.

Sincerely,
Assembler

59

call printf



```
$ gcc217 -c detecta.s
$ objdump --disassemble --reloc detecta.o
detecta.o: file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
 0: b8 00 00 00 00      mov     $0x0,%eax
 5: 48 c7 c7 00 00 00 00      mov     $0x0,%rdi
                               8: R_X86_64_32S      .rodata
   c: e8 00 00 00 00      callq  11 <main+0x11>
                               d: R_X86_64_PC32     printf-0x4
 11: e8 00 00 00 00      callq  16 <main+0x16>
                               12: R_X86_64_PC32     getchar-0x4
 16: 83 f8 41           cmp     $0x41,%eax
 19: 75 11           jne    2c <skip>
 1b: b8 00 00 00 00      mov     $0x0,%eax
 20: 48 c7 c7 00 00 00 00      mov     $0x0,%rdi
                               23: R_X86_64_32S      .rodata+0xe
 27: e8 00 00 00 00      callq  2c <skip>
                               28: R_X86_64_PC32     printf-0x4

000000000000002c <skip>:
 2c: b8 00 00 00 00      mov     $0x0,%eax
 31: c3                retq


```

60

call printf



Assembly lang: call printf
Machine lang: e8 00 00 00 00
Explanation:

```
11101000 00000000 00000000 00000000 00000000
Opcode: This is a call instruction with a 4-byte displacement
Disp: The displacement is 00000000H (0)
```

- call must contain a displacement
- Assembler must generate the displacement: [addr of printf] – [addr after call instr]
- But assembler didn't know addr of printf
 - printf isn't even present yet!
- So assembler couldn't generate this instruction completely

61

Relocation Record 5



```
$ gcc217 -c detecta.s
$ objdump --disassemble --reloc detecta.o
detecta.o: file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
 0: b8 00 00 00 00 mov $0x0,%eax
 5: 48 c7 c7 00 00 00 mov $0x0,%rdi
 8: R_X86_64_32S .rodata
 c: e8 00 00 00 00 callq 11 <main+0x11>
 d: R_X86_64_PC32 printf-0x4
11: e8 00 00 00 00 callq 16 <main+0x16>
12: R_X86_64_PC32 getchar-0x4
16: 83 f8 41 cmp $0x41,%eax
19: 75 11 jne 2c <skip>
1b: b8 00 00 00 00 mov $0x0,%eax
20: 48 c7 c7 00 00 00 mov $0x0,%rdi
23: R_X86_64_32S .rodata+0xe
27: e8 00 00 00 00 callq 2c <skip>
28: R_X86_64_PC32 printf-0x4

000000000000002c <skip>:
2c: b8 00 00 00 00 mov $0x0,%eax
31: c3 retq
```

62

Relocation Record 5



28: R_X86_64_PC32 printf-0x4

Dear Linker,

Please patch the TEXT section at offset 28_H. Patch in a 32-bit PC-relative address. When you determine the addr of printf, compute [addr of printf] – [addr after call] and place the result at the prescribed place.

Sincerely,
Assembler

63

movl \$0, %eax



```
$ gcc217 -c detecta.s
$ objdump --disassemble --reloc detecta.o
detecta.o: file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
 0: b8 00 00 00 00 mov $0x0,%eax
 5: 48 c7 c7 00 00 00 mov $0x0,%rdi
 8: R_X86_64_32S .rodata
 c: e8 00 00 00 00 callq 11 <main+0x11>
 d: R_X86_64_PC32 printf-0x4
11: e8 00 00 00 00 callq 16 <main+0x16>
12: R_X86_64_PC32 getchar-0x4
16: 83 f8 41 cmp $0x41,%eax
19: 75 11 jne 2c <skip>
1b: b8 00 00 00 00 mov $0x0,%eax
20: 48 c7 c7 00 00 00 mov $0x0,%rdi
23: R_X86_64_32S .rodata+0xe
27: e8 00 00 00 00 callq 2c <skip>
28: R_X86_64_PC32 printf-0x4

000000000000002c <skip>:
2c: b8 00 00 00 00 mov $0x0,%eax
31: c3 retq
```

64

movl \$0, %eax



Assembly lang: movl \$0, %eax
Machine lang: b8 00 00 00 00
Explanation:

```
10111000 00000000 00000000 00000000 00000000
Opcode: This is a mov instruction whose source operand is a four-byte immediate and whose destination is EAX
The immediate operand is 0
```

65

ret



```
$ gcc217 -c detecta.s
$ objdump --disassemble --reloc detecta.o
detecta.o: file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
 0: b8 00 00 00 00 mov $0x0,%eax
 5: 48 c7 c7 00 00 00 mov $0x0,%rdi
 8: R_X86_64_32S .rodata
 c: e8 00 00 00 00 callq 11 <main+0x11>
 d: R_X86_64_PC32 printf-0x4
11: e8 00 00 00 00 callq 16 <main+0x16>
12: R_X86_64_PC32 getchar-0x4
16: 83 f8 41 cmp $0x41,%eax
19: 75 11 jne 2c <skip>
1b: b8 00 00 00 00 mov $0x0,%eax
20: 48 c7 c7 00 00 00 mov $0x0,%rdi
23: R_X86_64_32S .rodata+0xe
27: e8 00 00 00 00 callq 2c <skip>
28: R_X86_64_PC32 printf-0x4

000000000000002c <skip>:
2c: b8 00 00 00 00 mov $0x0,%eax
31: c3 retq
```

66

ret



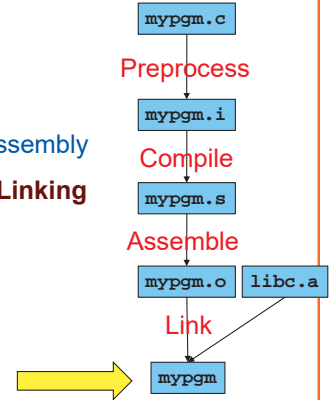
Assembly lang: `ret`
 Machine lang: `c3`
 Explanation:

```
11000011
Opcode: This is a ret (alias retq) instruction
```

Agenda



x86-64 Machine Language
 Buffer overrun vulnerabilities
 x86-64 Machine Language after Assembly
x86-64 Machine Language after Linking



From Assembler to Linker



Assembler writes its data structures to `.o` file

Linker:

- Reads `.o` file
- Writes executable binary file
- Works in two phases: **resolution** and **relocation**

Linker Resolution



Resolution

- Linker resolves references

For this program, linker:

- Notes that labels `getchar` and `printf` are unresolved
- Fetches machine language code defining `getchar` and `printf` from `libc.a`
- Adds that code to TEXT section
- Adds more code (e.g. definition of `_start`) to TEXT section too
- Adds code to other sections too

Linker Relocation



Relocation

- Linker patches ("relocates") code
- Linker traverses relocation records, patching code as specified

Examining Machine Lang: RODATA



Link program; run objdump

```

$ gcc217 detecta.o -o detecta
$ objdump --full-contents --section .rodata detecta

detecta:      file format elf64-x86-64

Contents of section .rodata:
400638 01000200 00000000 00000000 00000000 .....
400648 54797065 20612063 6861723a 20004869 Type a char: .Hi
400658 0a00                                ..
```

(Partial) addresses,
not offsets

RODATA is at ...00400638_H
 Starts with some header info
 Real start of RODATA is at ...00400648_H
 "Type a char: " starts at ...00400648_H
 "Hi\n" starts at ...00400656_H

Examining Machine Lang: TEXT



```
$ gcc217 detecta.o -o detecta
$ objdump --disassemble --reloc detecta
detecta: file format elf64-x86-64
...
Disassembly of section .text:
...
0000000000400514 <main>:
400514: b8 00 00 00 00    mov    $0x0,%eax
400519: 48 c7 c7 48 06 40 00    mov    $0x400648,%rdi
400520: e8 d3 fe ff ff    callq 4003f8 <printf@plt>
400525: e8 ee fe ff ff    callq 400418 <getchar@plt>
40052a: 83 f8 41          cmp    $0x41,%eax
40052d: 75 11            jne    400540 <skip>
40052f: b8 00 00 00 00    mov    $0x0,%eax
400534: 48 c7 c7 56 06 40 00    mov    $0x400656,%rdi
40053b: e8 b8 fe ff ff    callq 4003f8 <printf@plt>

0000000000400540 <skip>:
400540: b8 00 00 00 00    mov    $0x0,%eax
400545: c3              retq

...

```

Link program; run objdump

No relocation records!

Addresses, not offsets

Let's examine one line at a time...

Additional Code



```
$ gcc217 detecta.o -o detecta
$ objdump --disassemble --reloc detecta
detecta: file format elf64-x86-64
...
Disassembly of section .text:
...
0000000000400514 <main>:
400514: b8 00 00 00 00    mov    $0x0,%eax
400519: 48 c7 c7 48 06 40 00    mov    $0x400648,%rdi
400520: e8 d3 fe ff ff    callq 4003f8 <printf@plt>
400525: e8 ee fe ff ff    callq 400418 <getchar@plt>
40052a: 83 f8 41          cmp    $0x41,%eax
40052d: 75 11            jne    400540 <skip>
40052f: b8 00 00 00 00    mov    $0x0,%eax
400534: 48 c7 c7 56 06 40 00    mov    $0x400656,%rdi
40053b: e8 b8 fe ff ff    callq 4003f8 <printf@plt>

0000000000400540 <skip>:
400540: b8 00 00 00 00    mov    $0x0,%eax
400545: c3              retq

...

```

Additional code

movq \$msg1, %rdi



```
$ gcc217 detecta.o -o detecta
$ objdump --disassemble --reloc detecta
detecta: file format elf64-x86-64
...
Disassembly of section .text:
...
0000000000400514 <main>:
400514: b8 00 00 00 00    mov    $0x0,%eax
400519: 48 c7 c7 48 06 40 00    mov    $0x400648,%rdi
400520: e8 d3 fe ff ff    callq 4003f8 <printf@plt>
400525: e8 ee fe ff ff    callq 400418 <getchar@plt>
40052a: 83 f8 41          cmp    $0x41,%eax
40052d: 75 11            jne    400540 <skip>
40052f: b8 00 00 00 00    mov    $0x0,%eax
400534: 48 c7 c7 56 06 40 00    mov    $0x400656,%rdi
40053b: e8 b8 fe ff ff    callq 4003f8 <printf@plt>

0000000000400540 <skip>:
400540: b8 00 00 00 00    mov    $0x0,%eax
400545: c3              retq

...

```

Recall: Real addr of RODATA = ...00400648_H

Linker replaced 00000000_H with real addr of RODATA + 0 = ...00400648_H = addr denoted by msg1

call printf



```
$ gcc217 detecta.o -o detecta
$ objdump --disassemble --reloc detecta
detecta: file format elf64-x86-64
...
Disassembly of section .text:
...
0000000000400514 <main>:
400514: b8 00 00 00 00    mov    $0x0,%eax
400519: 48 c7 c7 48 06 40 00    mov    $0x400648,%rdi
400520: e8 d3 fe ff ff    callq 4003f8 <printf@plt>
400525: e8 ee fe ff ff    callq 400418 <getchar@plt>
40052a: 83 f8 41          cmp    $0x41,%eax
40052d: 75 11            jne    400540 <skip>
40052f: b8 00 00 00 00    mov    $0x0,%eax
400534: 48 c7 c7 56 06 40 00    mov    $0x400656,%rdi
40053b: e8 b8 fe ff ff    callq 4003f8 <printf@plt>

0000000000400540 <skip>:
400540: b8 00 00 00 00    mov    $0x0,%eax
400545: c3              retq

...

```

Addr of printf = ...004003f8_H

Linker replaced 00000000_H with [addr of printf] - [addr after call] = ...004003f8_H - ...00400525_H = ...fffffed3_H = -301_D

call getchar



```
$ gcc217 detecta.o -o detecta
$ objdump --disassemble --reloc detecta
detecta: file format elf64-x86-64
...
Disassembly of section .text:
...
0000000000400514 <main>:
400514: b8 00 00 00 00    mov    $0x0,%eax
400519: 48 c7 c7 48 06 40 00    mov    $0x400648,%rdi
400520: e8 d3 fe ff ff    callq 4003f8 <printf@plt>
400525: e8 ee fe ff ff    callq 400418 <getchar@plt>
40052a: 83 f8 41          cmp    $0x41,%eax
40052d: 75 11            jne    400540 <skip>
40052f: b8 00 00 00 00    mov    $0x0,%eax
400534: 48 c7 c7 56 06 40 00    mov    $0x400656,%rdi
40053b: e8 b8 fe ff ff    callq 4003f8 <printf@plt>

0000000000400540 <skip>:
400540: b8 00 00 00 00    mov    $0x0,%eax
400545: c3              retq

...

```

Addr of getchar = ...00400418_H

Linker replaced 00000000_H with [addr of getchar] - [addr after call] = ...00400418_H - ...0040052a_H = ...fffffee_H = -274_D

movq \$msg2, %rdi



```
$ gcc217 detecta.o -o detecta
$ objdump --disassemble --reloc detecta
detecta: file format elf64-x86-64
...
Disassembly of section .text:
...
0000000000400514 <main>:
400514: b8 00 00 00 00    mov    $0x0,%eax
400519: 48 c7 c7 48 06 40 00    mov    $0x400648,%rdi
400520: e8 d3 fe ff ff    callq 4003f8 <printf@plt>
400525: e8 ee fe ff ff    callq 400418 <getchar@plt>
40052a: 83 f8 41          cmp    $0x41,%eax
40052d: 75 11            jne    400540 <skip>
40052f: b8 00 00 00 00    mov    $0x0,%eax
400534: 48 c7 c7 56 06 40 00    mov    $0x400656,%rdi
40053b: e8 b8 fe ff ff    callq 4003f8 <printf@plt>

0000000000400540 <skip>:
400540: b8 00 00 00 00    mov    $0x0,%eax
400545: c3              retq

...

```

Recall: Real addr of RODATA = ...00400648_H

Linker replaced 00000000_H with real addr of RODATA + e_H = ...00400648_H + e_H = ...00400656_H = addr denoted by msg2

call printf



```
$ gcc217 detecta.o -o detecta
$ objdump --disassemble --reloc detecta
detecta: file format elf64-x86-64
...
Disassembly of section .text:
...
0000000000400514 <main>:
400514: b8 00 00 00 00      mov    $0x0,%eax
400519: 48 c7 c7 48 06 40 00  mov    $0x400648,%rdi
400520: e8 d3 fe ff ff      callq 4003f8 <printf@plt>
400525: e8 ee fe ff ff      callq 400418 <getchar@plt>
40052a: 83 f8 41            cmp    $0x41,%eax
40052d: 75 11              jne    400540 <skip>
40052f: b8 00 00 00 00      mov    $0x0,%eax
400534: 48 c7 c7 56 06 40 00  mov    $0x400656,%rdi
40053b: e8 b8 fe ff ff      callq 4003f8 <printf@plt>

0000000000400540 <skip>:
400540: b8 00 00 00 00
400545: c3
...
```

Addr of printf
= ...004003f8_H

Linker replaced 00000000_H with
[addr of printf] - [addr after call]
= ...004003f8_H - ...00400540_H
= ...fffffeb8_H
= -328_D

79

Summary



x86-64 Machine Language

- CISC: many instructions, complex format
- Fields: prefix, opcode, modR/M, SIB, displacement, immediate

Assembler

- Reads assembly language file
- Generates TEXT, RODATA, DATA, BSS sections
 - Containing machine language code
- Generates **relocation records**
- Writes object (.o) file

Linker

- Reads object (.o) file(s)
- Does **resolution**: resolves references to make code complete
- Does **relocation**: traverses relocation records to patch code
- Writes executable binary file

80