Assembly Programming 2

SHELLCODE

@sleepunderflow

Call Frame

Argument 2 (RBP+0x18)

Argument 1 (RBP+0x10)

Return Pointer

RBP <-RBP

Local_variable1 (RBP-0x08)

Local_variable2 (RBP-0x10) <-RSP

ENTER / LEAVE

```
PUSH RBP
MOV RBP, RSP
SUB RSP, 0x48
.....
MOV RSP, RBP
POP RBP
RET
```

```
ENTER
SUB RSP, 0x48
.....
LEAVE
RET
```

Call Frame

Argument 2 (RBP+0x18)

Argument 1 (RBP+0x10)

Return Pointer

RBP <-RBP

Local_variable1 (RBP-0x08)

Local_variable2 (RBP-0x10) <-RSP

Call Frame

Argument 2 (RBP+0x18)

Argument 1 (RBP+0x10)

Return Pointer

RBP <-RBP <-RSP

Local_variable1 (RBP-0x08)

Local_variable2 (RBP-0x10)

Call Frame

Argument 2 (RBP+0x18) <-RBP

Argument 1 (RBP+0x10)

Return Pointer

RBP <-RSP

Local_variable1 (RBP-0x08)

Local_variable2 (RBP-0x10)

Shellcode

- Injected into a program and jumped to
- Usually limited space
- Can be mangled by a program
- Can be anywhere in the memory
- Often terminated by a NULL byte

Minimizing size

MOV RAX, 0:

```
48 b8 00 00 00 00 00 00 00 00 – MOV RAX, 0
48 C7 C0 00 00 00 00 – MOV RAX, 0
48 81 e0 00 00 00 00 – AND RAX, 0
48 25 00 00 00 00 – AND RAX, 0
48 83 E0 00 – AND RAX, 0
48 C1 E8 40 – SHR RAX, 64
48 31 C0 – XOR RAX, RAX
48 33 Co – XOR RAX, RAX
```

NOP:

```
90
66 90
0F 1F 00
0F 1F 40 00
0F 1F 44 00 00
66 0F 1F 44 00 00
0F 1F 80 00 00 00 00
0F 1F 84 00 00 00 00 00
66 0F 1F 84 00 00 00 00 00
XCHG EAX, EAX - 87 C0
```

Avoiding NULL bytes

- Replace MOV reg, 0 with XOR reg, reg
- Use RIP relative forms
- Use smaller registers
- Use shift register operations
- Use negative numbers
- Use different instruction/instruction encoding
- Add that NULL byte in the runtime

Getting address of currently executed instruction

- CALL RIP + 50
- .....
- POP RAX
- JMP RIP - 47

Avoiding shellcode corruption

- Stack moves!
  - If possible place the shellcode after return pointer
- If you've overflown return pointer, you have overflown RBP
  - Avoid LEAVE instructions
- You want always know where exactly the shellcode is
  - NOP sled
- Remember that NULL and \n might terminate the shellcode
  - 64-bit addresses always have NULL byte!

Pointers and Arrays

- Pointer – go to this address and take a value
- Array – go to this address (pointer) and keep taking values until you get to a NULL value
- Array of Arrays – go to this address and keep getting pointers to arrays until you hit NULL

Prefixes

- Operand-Size prefix – 66h
- Address-Size prefix – 67h
- REX Prefix – 40h – 4Fh

Table 3-4. Effective Operand- and Address-Size Attributes in 64-Bit Mode

L Flag in Code Segment Descriptor	1	1	1	1	1	1	1	1
REX.W Prefix	0	0	0	0	1	1	1	1
Operand-Size Prefix 66H	N	N	Y	Y	N	N	Y	Y
Address-Size Prefix 67H	N	Y	N	Y	N	Y	N	Y
Effective Operand Size	32	32	16	16	64	64	64	64
Effective Address Size	64	32	64	32	64	32	64	32

NOTES:

Y: Yes - this instruction prefix is present.

N: No - this instruction prefix is not present.

https://software.intel.com/sites/default/files/managed/39/c5/325462-sdm-vol-1-2abcd-3abcd.pdf

REX prefix

Table 2-4. REX Prefix Fields [BITS: 0100WRXB]

Field Name	Bit Position	Definition
-	7:4	0100
W	3	0 = Operand size determined by CS.D
		1 = 64 Bit Operand Size
R	2	Extension of the ModR/M reg field
X	1	Extension of the SIB index field
B	0	Extension of the ModR/M r/m field, SIB base field, or Opcode reg field

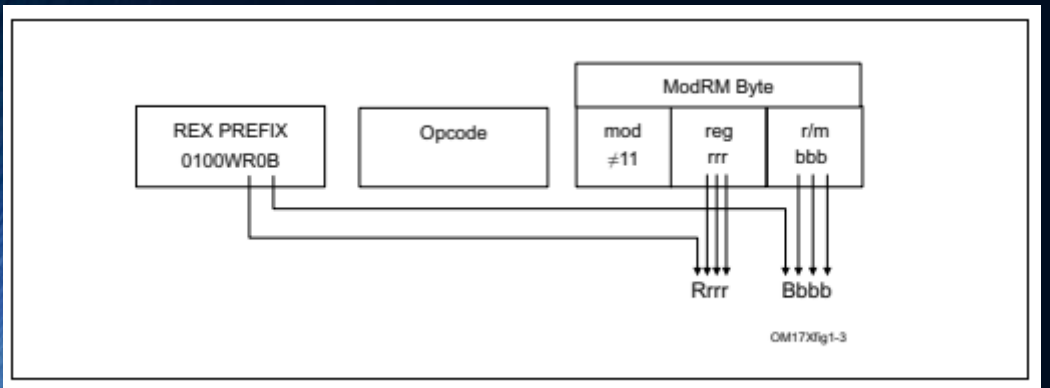


Figure 2-4. Memory Addressing Without an SIB Byte; REX.X Not Used

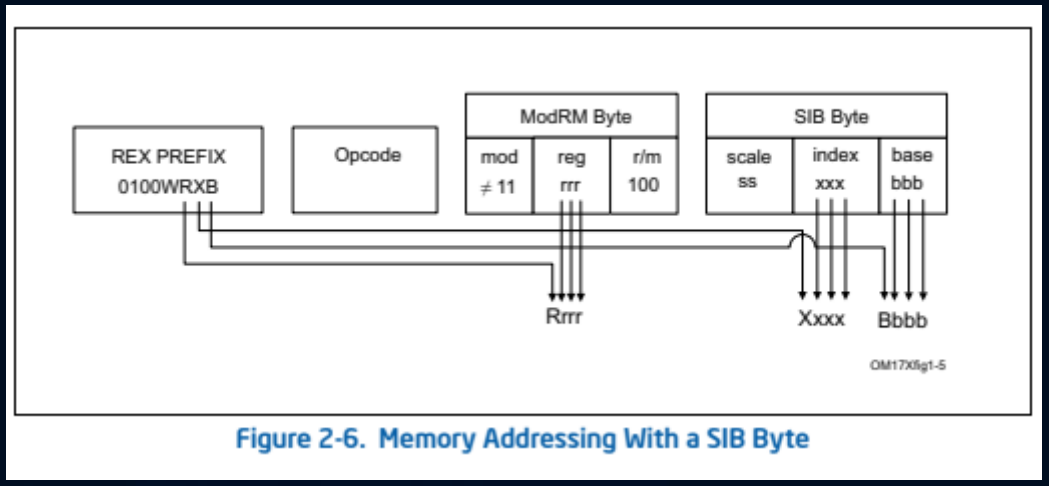
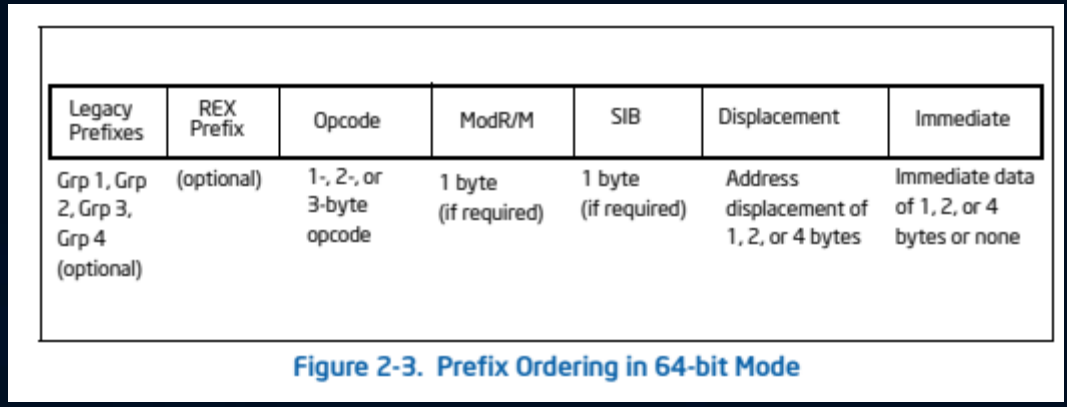


Figure 2-6. Memory Addressing With a SIB Byte

https://software.intel.com/sites/default/files/managed/39/c5/325462-sdm-vol-1-2abcd-3abcd.pdf

Instruction size limit

- Instruction size is limited to 15 bytes



Legacy Prefixes	REX Prefix	Opcode	ModR/M	SIB	Displacement	Immediate
Grp 1, Grp 2, Grp 3, Grp 4 (optional)	(optional)	1-, 2-, or 3-byte opcode	1 byte (if required)	1 byte (if required)	Address displacement of 1, 2, or 4 bytes	Immediate data of 1, 2, or 4 bytes or none

Figure 2-3. Prefix Ordering in 64-bit Mode

https://software.intel.com/sites/default/files/managed/39/c5/325462-sdm-vol-1-2abcd-3abcd.pdf

Network programming

- Different sets of syscalls for x86 and x86_64
- Low level stuff handled by OS
- Uses sockets
- Can use raw sockets but requires root
- Network byte order

Memory protection

- Can be RWX
- Modified using mprotect
- Page based

To Do:

- Reverse shell
- Password protected bind shell
- No NULL byte reverse shell
- Position independent reverse shell

Homework:

- Simple HTTP server