Intel x64 Assembly Cheatsheet

DEFCON Toronto Introduction to Linux 64-bit Binary Exploitation

By superkojiman

Sections

`.text`	Program code
`.bss`	Variables
`.data`	Initialized data or constants

Registers

General purpose	RAX, RCX, RDX, RBX, RSI, RDI, R8 to R15
Base/frame pointer	RBP
Stack pointer	RSP
Instruction pointer	RIP
RFLAGS	Status register

64-bit registers have sub registers
RAX (64-bits), EAX (32-bits), AX (16-bits), AH (8-bits), AL (8-bits)
R8 (64-bits), R8D (32-bits), R8B (16-bits)

Function call convention

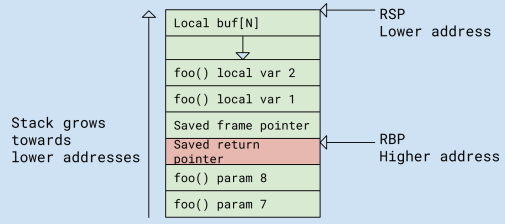
First six function parameters passed into registers; additional pushed on stack.
RIP + 8 pushed on the stack as saved return pointer.
Execution jumps into the function.
Function prologue sets up stack frame.
Save any return value to RAX.
Function epilogue releases stack frame and returns to calling function.

Stack frames

Function prologue	`push rbp`
	`mov rbp,rsp`
	`sub rsp,N`

Function epilogue	`leave` `ret`
Local variables	`$rbp - offset`
Saved return pointer	`$rbp + 8`
Function parameters on stack	`$rbp + offset`

Stack frame layout for foo()



Endianess

Intel processors store words in little-endian.
Least-significant byte is stored at the smallest address.
That means if you want to store 0x4005a1 you store it as 0xa10540

MOV (Move)

Copy RAX into RCX	`mov rcx, rax`
Copy 123 into RCX	`mov rcx, 123`
Copy 123 into 0x600000	`mov 0x600000, 123`
Copy dereferenced RBP-0x50 into RCX	`mov rcx, [rbp - 0x50]`

Copies operand 2 into operand 1

LEA (Load Effective Address)

Copy result of RBP-0x50 into RAX	`lea rax, [rbp - 0x50]`

Copies address specified in operand 2 into operand 1

ADD (Addition)

Add RAX and RCX, and store result in RAX	`add rax, rcx`
Add 12 to RAX	`add rax, 12`

SUB (Subtraction)

Subtract RCX from RAX, and store result in RAX	`sub, rax, rcx`
Subtract 12 from RAX	`sub rax, 12`

PUSH

Push data onto the stack	`push 123`

POP

Remove data from the stack into a register	`pop rbx`

LEAVE

Clear the stack frame. Equivalent of	`mov rsp,rbp; pop rbp`

RET (Return)

Return from function call. Equivalent of	`pop rip`

SYSCALL (System Call)

Execute system call	`syscall`

Set RAX to the system call ID to execute:
`read: rax=0x0, write: rax=0x1, execve: rax=0x3b, exit: rax=0x3c`
Parameters are passed in RDI, RSI, RDX, R10, R8, and R9.
Return value of syscall is saved in RAX.
System calls: https://w3challs.com/syscalls/?arch=x86_64

CALL

Execute function foo()	`call foo`

The first six parameters are passed in RDI, RSI, RDX, RCX, R8, and R9.
Additional parameters are pushed on the stack.
Return value of function is saved in RAX.

XOR (Exclusive OR)

Can be used as a shortcut to zero a `xor rax, rax`
Useful when writing shellcode because it's only 1 byte in size and does not result in NULL bytes.

CMP (Compare)

Compare RAX, RBX by subtracting RBX from RAX and setting the FLAGS register accordingly `cmp rax, rbx`

Jump

Jump to location	`jmp 0x40062d`
Jump to register	`jmp rsp`

Jump from CMP

Jump if zero	`jz 0x40062d`
Jump if equal	`je 0x40062d`
Jump if greater	`jg 0x40062d`
Jump if less	`jl 0x40062d`
Jump if greater or equal	`jge 0x40062d`
Jump if less or equal	`jle 0x40062d`

References

Introduction to x64 Assembly: https://software.intel.com/en-us/articles/introduction-to-x64-assembly/
System Calls: https://w3challs.com/syscalls/?arch=x86_64