2025.09.12

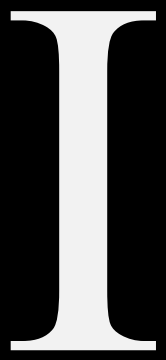
ISA and Tools

가디언 시스템 보안 & 취약점 분석 세미나 1.2

임준서

x86-64

ISA



ISA

Introduction

하드웨어와 소프트웨어 연결

Arithmetic, Logic, Control, Memory

RISC vs CISC

x86은 CISC

x86-64

a.k.a. AMD64, EM64T

64비트 아키텍쳐

기존 x32 ISA와 호환

ref) www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html

x86-64

표기법

Intel vs AT&T

Intex Syntax

```
instr   dest,sourc
emov    eax,[ecx]
```

AT&T Syntax

```
instr   source, dest
movl    (%ecx), %eax
```

AT&T Bell Labs 에서 제작

세미나 및 사용 tool에서는 intel syntax를 따름

x86-64

Registers

General Regs: 16개, 64bit

`rax, rdi, rsi, rdx, rcx, …`

Segment Regs: 6개, 64bit

`CS, DS, SS, ES, FS, GS`

RIP

다음에 실행할 명령어 주소

x86-64

Registers

Flag Regs: 연산 결과 flag

CF, PF, AF, ZF, SF, …

Floating-Point & Vector Regs

소수점 연산 + 최적화에 활용

x86-64

Flags

비트	이름	의미
0	CF (Carry Flag)	덧셈/뺄셈에서 자리올림(캐리) or 빌림 발생
2	PF (Parity Flag)	결과 하위 8비트의 1의 개수가 짝수면 1
4	AF (Auxiliary Carry)	BCD 연산용 (거의 안 씀)
6	ZF (Zero Flag)	결과가 0이면 1
7	SF (Sign Flag)	결과가 음수면 1
8	TF (Trap Flag)	1이면 한 명령어씩 실행(디버깅)
9	IF (Interrupt Enable)	1이면 외부 인터럽트 허용
10	DF (Direction Flag)	문자열 처리 시 방향 (0=증가, 1=감소)
11	OF (Overflow Flag)	부호 있는 연산에서 오버플로우 발생 시 1

x86-64

Data movement

Mnemonic	의미	비고
mov dst, src	값 복사	범용적
movzx/movsxd	크기 확장 (zero/sign)	작은 → 큰 레지스터
lea dst, [addr]	주소 계산	dst = effective address
push src	스택에 저장	rsp 감소
pop dst	스택에서 꺼냄	rsp 증가

x86-64

Data movment

CISC에서는 mov + 연산 가능

```
mov rdi, [rbp-0x20]
```

```
lea rdi, [rbp-0x20]
```

x86-64

Arithmetic / Logic

Mnemonic	의미	비고
add/sub	덧셈/뺄셈	CF/ZF/SF/OF 갱신
imul/idiv	정수 곱/나눗셈	느림, 특수 레지스터 사용
inc/dec	+1 / −1	OF만 바뀜 (CF는 유지)
and/or/xor/not	비트 연산	마스크 작업
shl/shr/sar	시프트	shift arithmetic right
cmp	뺄셈 후 결과 버림	플래그만 갱신

x86-64

Test and Branching

Mnemonic	의미	비고
test a, b	AND 결과만 플래그에 반영	값은 버림
cmp a, b	a-b 결과만 플래그에 반영	
jz/jnz	Zero Flag 기반 분기	je = jz
js/jns	Sign Flag 기반 분기	
jc/jnc	Carry Flag 기반 분기	
jo/jno	Overflow Flag 기반 분기	

x86-64

Control

Mnemonic	의미	비고
jmp label	무조건 점프	
call func	함수 호출, ret으로 복귀	rsp에 return addr 저장
leave	`mov rsp, rbp; pop rbp;`	ret 전 스택 정리
ret	스택에서 복귀 주소 pop	
loop label	RCX-- 하고 0 아니면 점프	잘 안 씀
int n	소프트웨어 인터럽트	옛날 도스/BIOS, 디버깅

x86-64

Syscall

Mnemonic	의미	비고
syscall	유저 → 커널 모드 전환	리눅스 64비트 시스템콜
sysret	커널 → 유저 모드 복귀	
hlt	CPU halt	OS idle 루프에서 사용
cpuid	CPU 정보 질의	보안/최적화 코드에서 자주
int 80	interrupt 80	x86에서 syscall

Calling Convention II

x86-64 ABI

Application Binary Interface

ABI

binary 수준 interface

Calling convention, Regs, Syscall

windows와 linux은 위 내용들이 다르기에 호환이 안 된다,

x86 ABI는 다루지 않는다.

x86은 cdecl, stdcall, fastcall, thiscall 등을 참고

여기서 x86-64에 대한 리눅스의 ABI를 볼 수 있다.

Application Binary Interface

x86-64 ELF

Executable and Linkable Format

Unix 기반 시스템의 표준 실행파일

컴파일 시 산출되는 .o, .obj 파일이 ELF format임

x86-64 ELF

Registers

Register	Usage	function calls
%rax	temporary register; with variable arguments passes information about the number of vector registers used; 1st return register	No
%rbx	callee-saved register; optionally used as base pointer	Yes
%rcx	used to pass 4th integer argument to functions	No
%rdx	used to pass 3rd argument to functions; 2nd return register	No
%rsp	stack pointer	Yes
%rbp	callee-saved register; optionally used as frame pointer	Yes
%rsi	used to pass 2nd argument to functions	No
%rdi	used to pass 1st argument to functions	No
%r8	used to pass 5th argument to functions	No
%r9	used to pass 6th argument to functions	No
%r10	temporary register, used for passing a function's static chain pointer	No
%r11	temporary register	No
%r12–r15	callee-saved registers	Yes
%xmm0–%xmm1	used to pass and return floating point arguments	No
%xmm2–%xmm7	used to pass floating point arguments	No
%xmm8–%xmm15	temporary registers	No
%mmx0–%mmx7	temporary registers	No
%st0,%st1	temporary registers; used to return long double arguments	No
%st2–%st7	temporary registers	No
%fs	Reserved for system (as thread specific data register)	No
mxcsr	SSE2 control and status word	partial
x87 SW	x87 status word	No
x87 CW	x87 control word	Yes

refspecs.linuxbase.org
x86_64-abi-0.99.pdf

Calling Convention

Function call

```
1 int funcion(int a0, int a1, int a2, int a3){
2     return a0
3 }
```

Calling Convention

Function call

```
; a0는 실제론 메모리 주소
mov rdi, [a0]
mov rsi, [a1]
mov rdx, [a2]
mov rcx, [a3]
call function
; rax = function(a0, a1, a2, a3)
```

Calling Convention

Function call

```
1 double function(double a0, double a1, int a2, double a3)
2 {
3     return a0;
4 }
```

Calling Convention

Function call

```
1 ; a0는 실제론 메모리 주소
2 mov XMM0, [a0]
3 mov XMM1, [a1]
4 mov rdi, [a2]
5 mov XMM2, [a3]
6 call function
7 ; XMM0에 반환값이 들어있음
```

Calling Convention

Inside a function

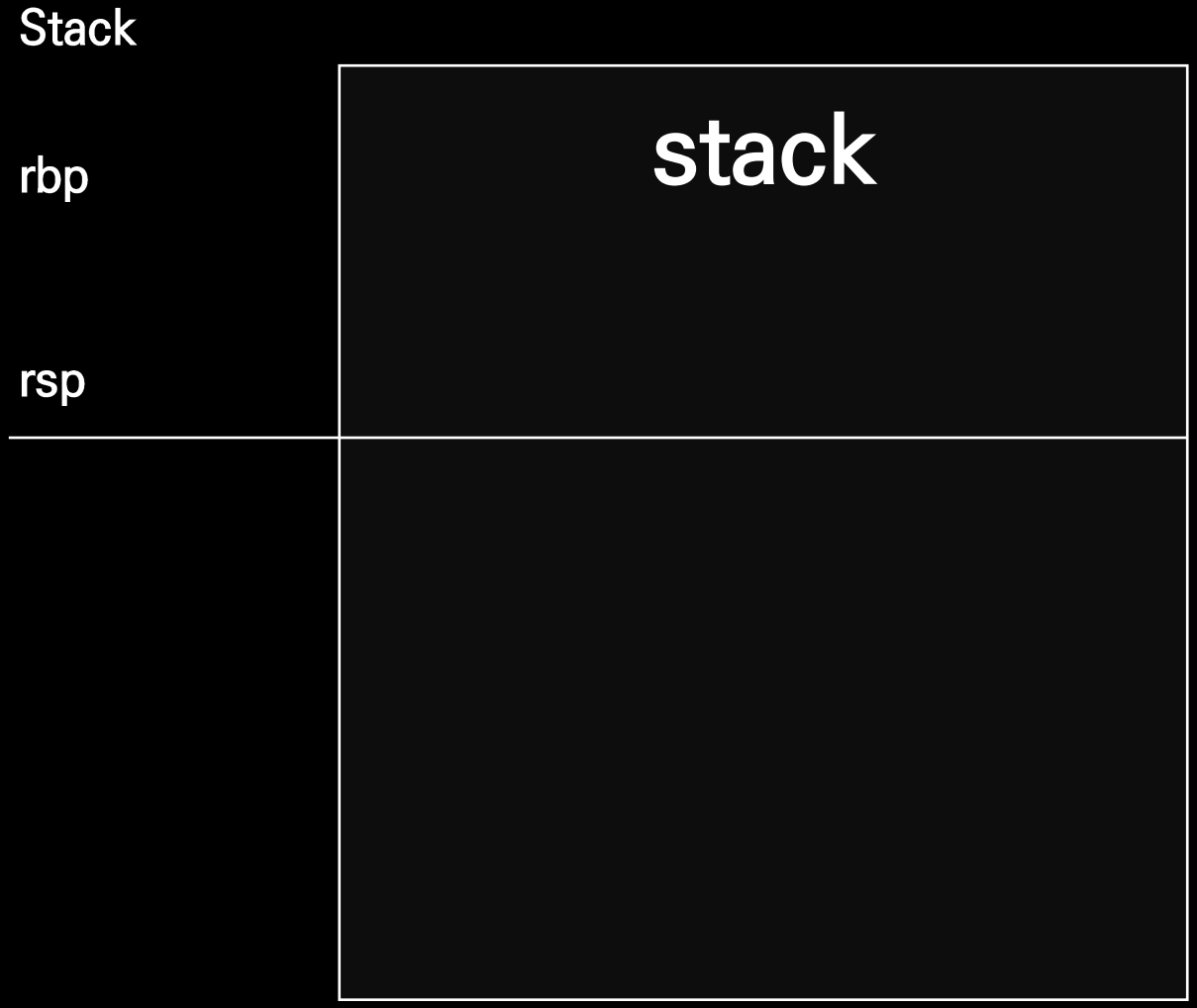
```
1  0x000011c9  f30f1efa  endbr64
2  0x000011cd  55        push rbp
3  0x000011ce  4889e5    mov rbp, rsp
4  0x000011d1  4883c480  add rsp, 0xffffffffffffff80 ; -0x80
5  ...
6  0x000014c0  c9        leave
7  0x000014c1  c3        ret
```

call, leave, ret

Explained

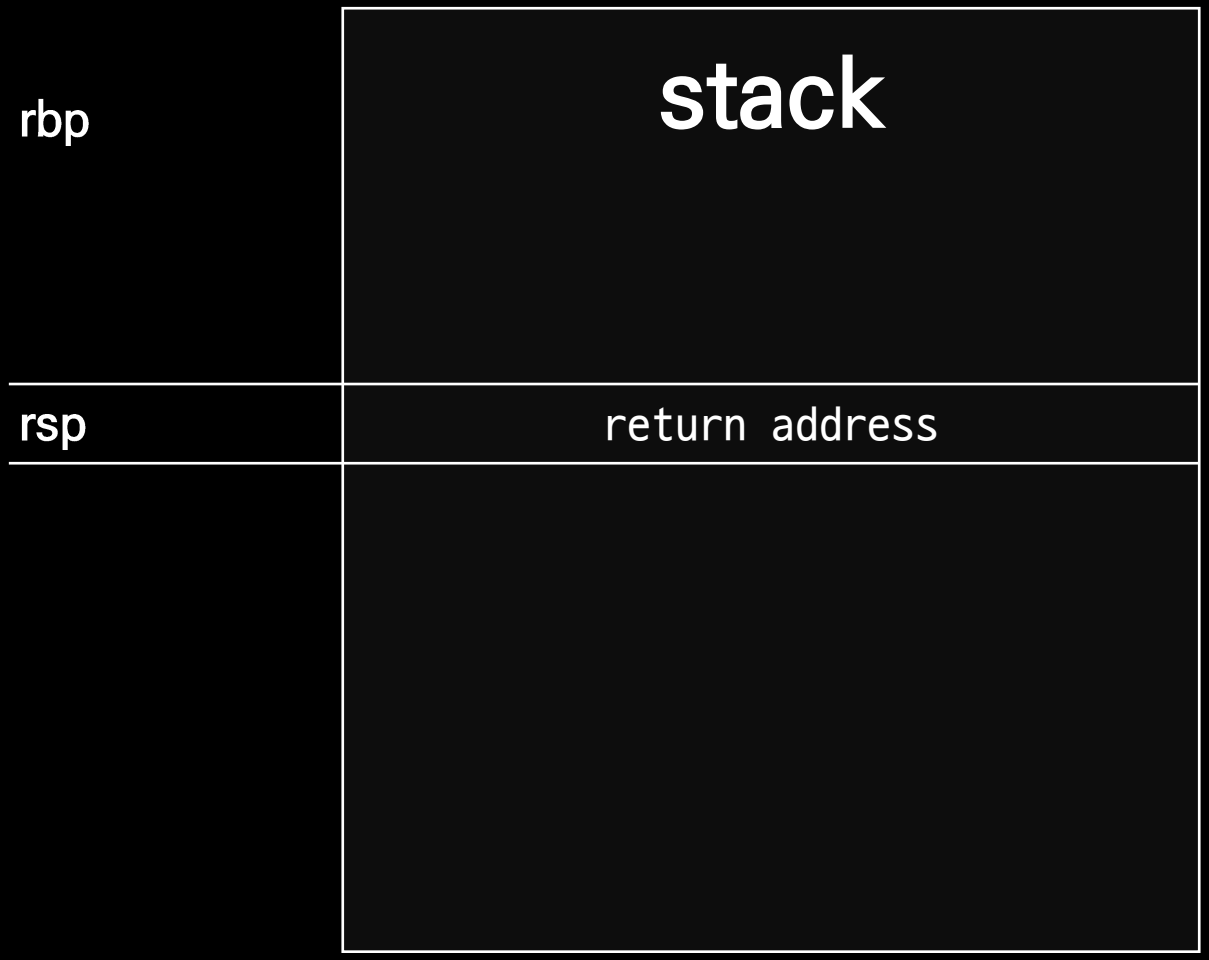
```
call = push rip; jmp target;
leave = mov rsp, rbp; pop rbp;
ret = pop rip;
```

Calling Convention



Calling Convention

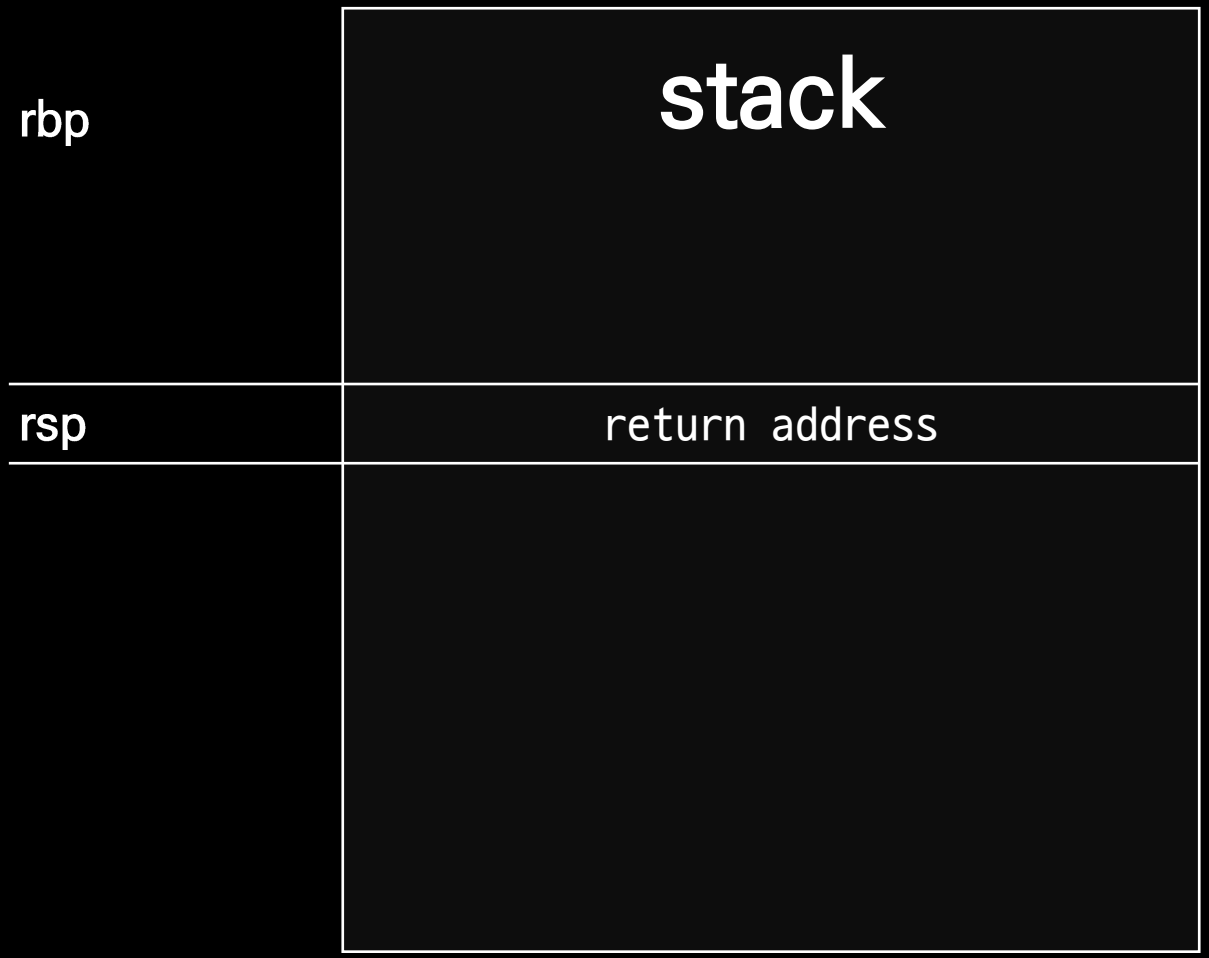
Call func



Calling Convention

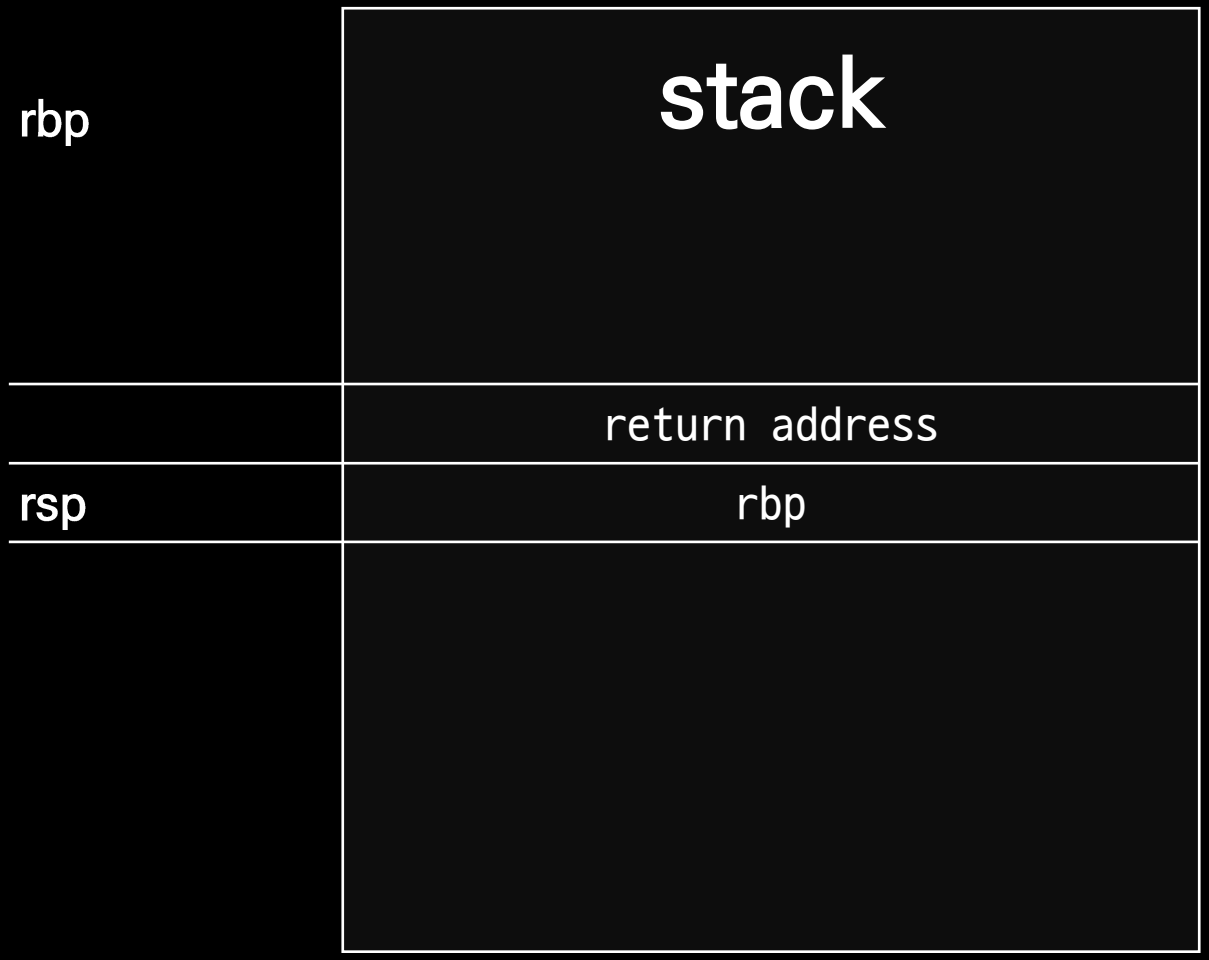
`rip = function address`

Call func



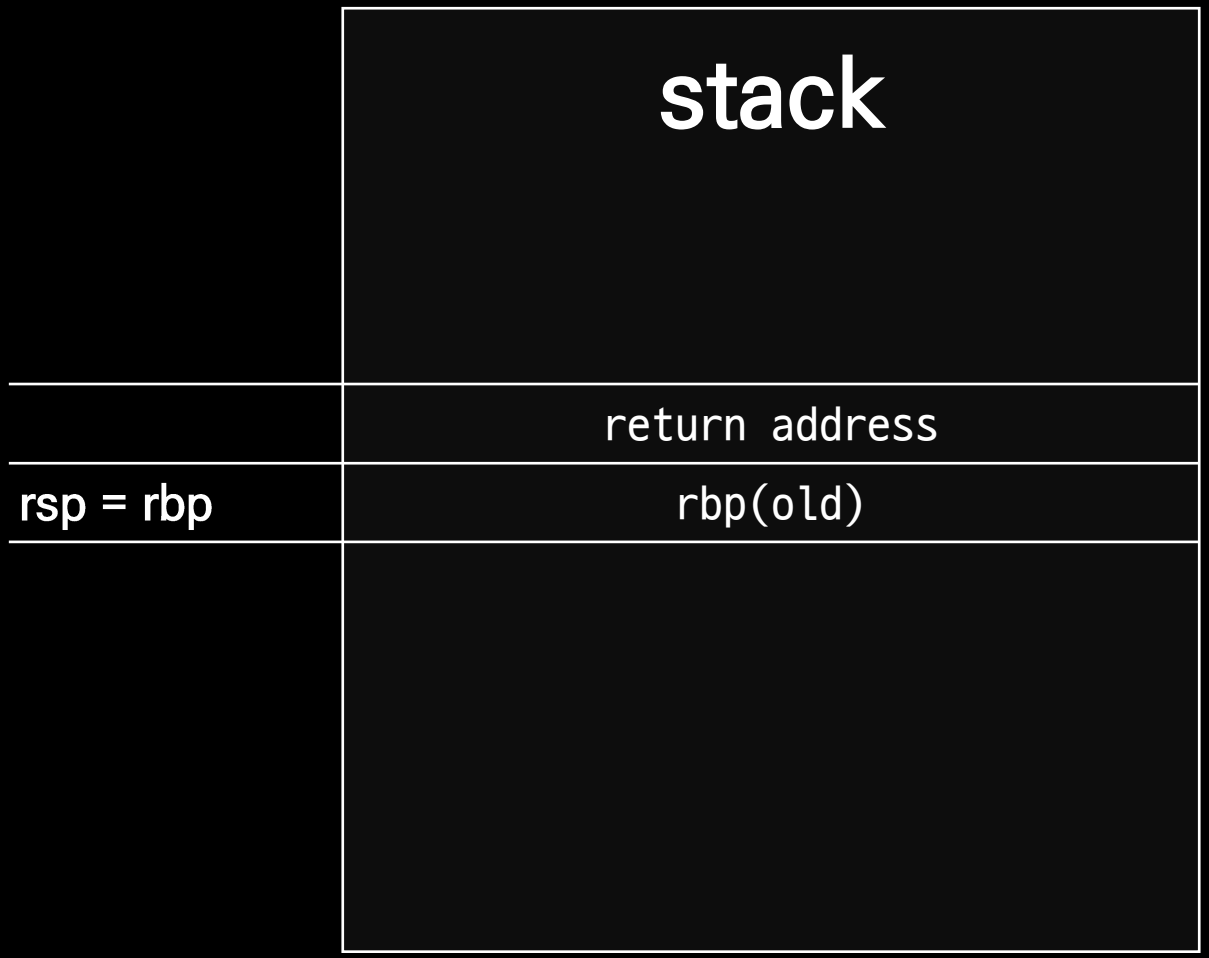
Calling Convention

Prologue



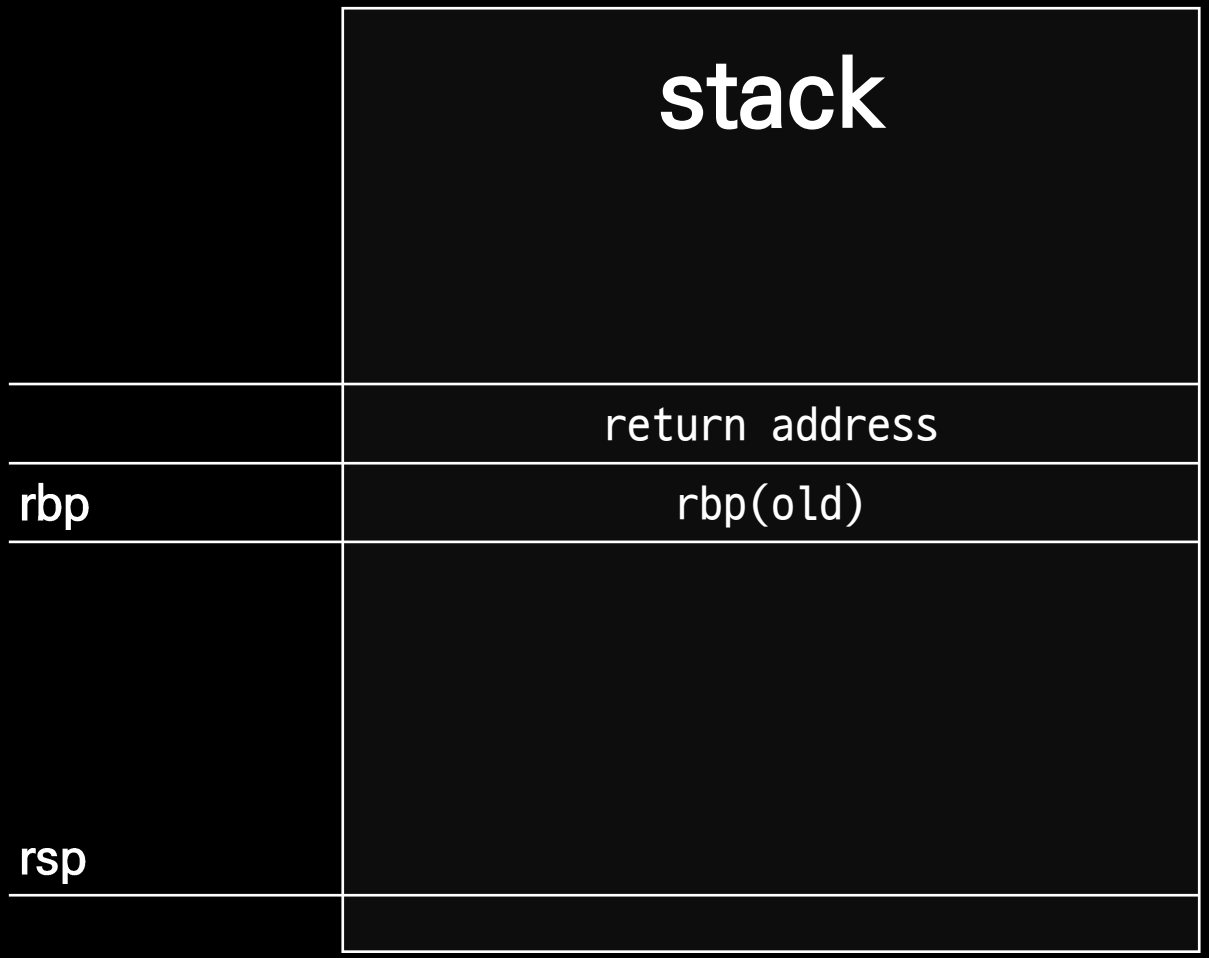
Calling Convention

Prologue



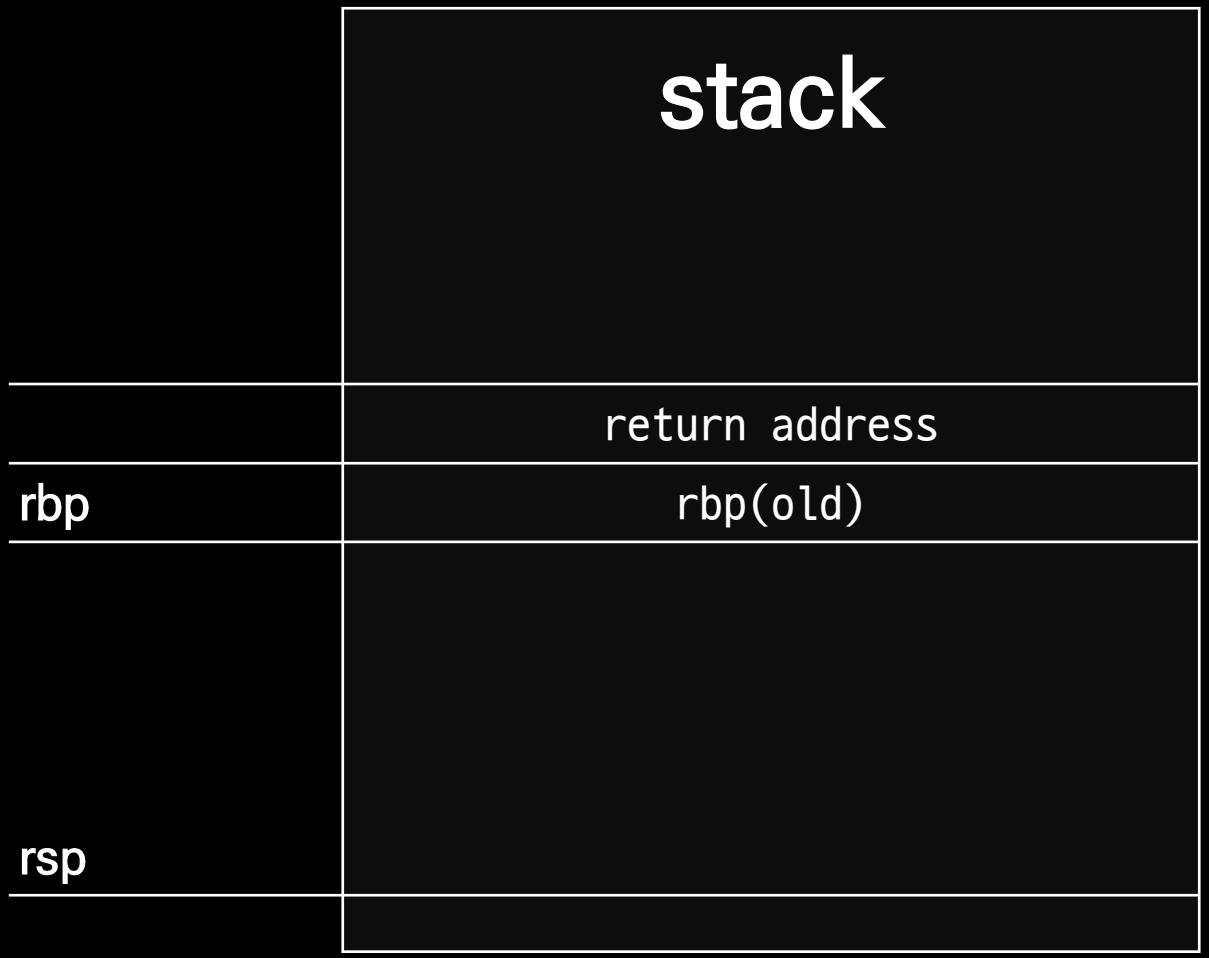
Calling Convention

Prologue



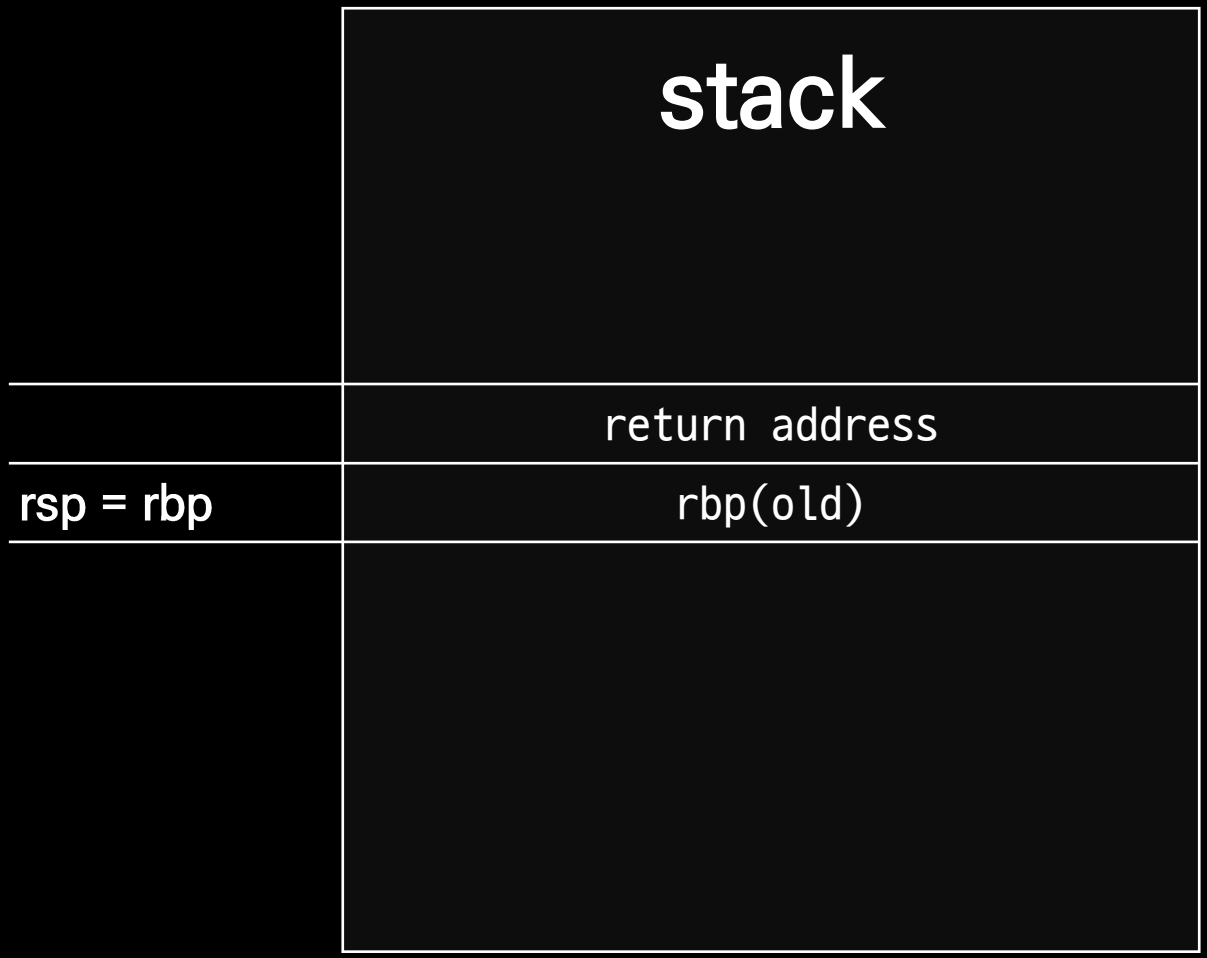
Calling Convention

Epilogue – leave



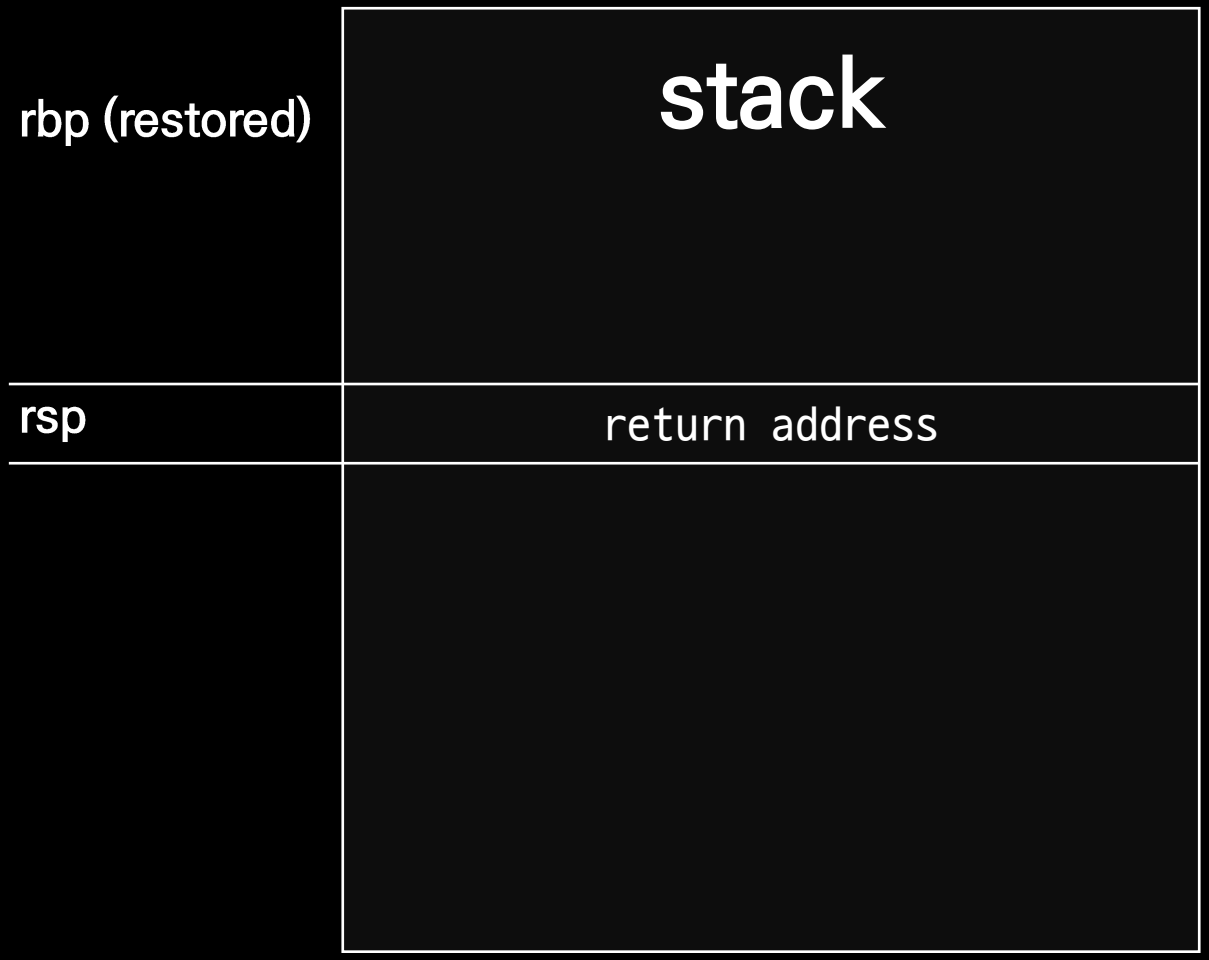
Calling Convention

Epilogue – leave



Calling Convention

Epilogue – leave



Calling Convention

`rip = return address`

Epilogue – ret

GDB / IDA

How to use tools

III

실습 [rev-basic-0]

http://dreamhack.io/wargame/challenges/14

실습 [rev-basic-0]

http://dreamhack.io/wargame/challenges/14

ELF header

.text
.data
.rodata
.bss
.plt
.got
.eh_frame

ELF header

실습

.text 코드 영역

.data 초기화 된 데이터

```
static int data = 10;
```

.bss 초기화 안 된 데이터

```
static int data;
```

Block Stated by Symbol

ELF header

실습

.rodata 읽기 전용 영역

.plt 외부 함수 호출용 (jmp)

.got 외부 함수 호출용 (addr)

Read Only DATA

Procedure Linkable Table

Global Offset Table

ELF header

실습

.eh_frame stack unwinding

```
throw std::runtime_error("error message");
```

Exception Handling FRAME

2025.09.12

Q&A

질문이 있다면 하십시오

임준서

2.5cm-2.5cm 떨어진 제목 36px

제목 하단의 부제목 18px

3.5cm 떨어진 내용 1 32px

좌측으로 0.5cm 떨어진 내용 하단의 설명 18px

3.5cm 떨어진 내용 2 32px

좌측으로 0.5cm 떨어진 내용 하단의 설명 18px

3.5cm 떨어진 내용 3 32px

좌측으로 0.5cm 떨어진 내용 하단의 설명 18px

1cm-1cm 떨어진 주석 12px

1cm-1cm 떨어진 주석 12px

2.5cm-3.5cm 떨어진 제목 36px

제목 하단의 부제목 18px

3.5cm 떨어진 내용 1 32px

```
Git init
Git status
Git add text.txt
Git add .
Git commit
Ctrl+C
Git commit -m “genesis”
Git log
Git log --oneline
```

```
Git add .
Git reset .
Git commit -m “add README”
Git log --oneline -n 3
Git commit –a –m “hello”
```

1cm-1cm 떨어진 주석 12px

1cm-1cm 떨어진 주석 12px