

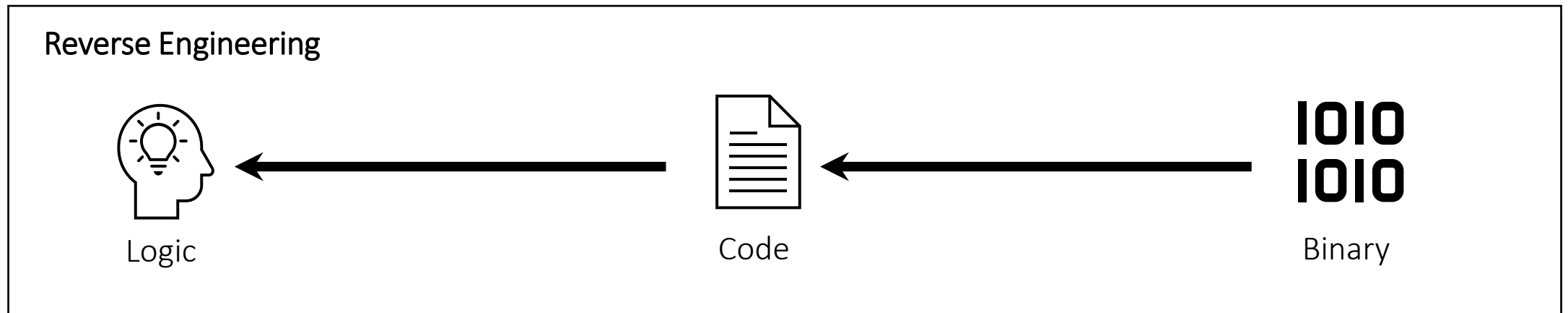
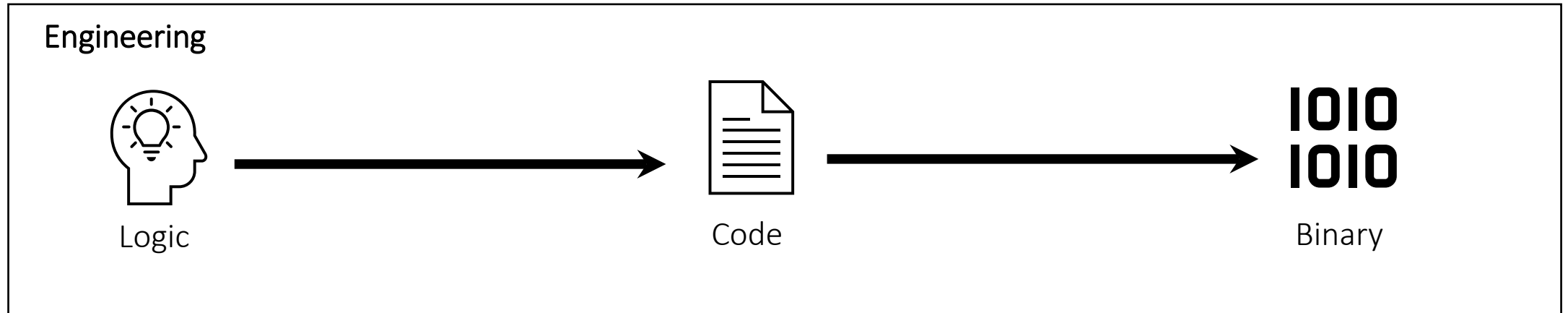
Reverse Engineering

Insu Yun

Today's lecture

- Understand what reverse engineering is
- Understand static analysis and dynamic analysis
- Understand x86 assembly
- Understand how to use decompiler

What is reverse engineering?



Reverse engineering is challenging due to loss of information

```
void decode_string(char* str, int len) {  
    // XOR string with a given length  
    for (int i = 0; i < len; i++) {  
        *str ^= 42;  
        str++;  
    }  
}
```

No variable or
function names

```
void FUN_10000e20(byte *param_1,ulong param_2)  
{  
    int local_1c;  
    byte *local_10;  
  
    local_1c = 0;  
    local_10 = param_1;  
    while ((ulong)(long)local_1c < param_2) {  
        *local_10 = *local_10 ^ 0x2a;  
        local_10 = local_10 + 1;  
        local_1c = local_1c + 1;  
    }  
    return;  
}
```

No comment

Ambiguous
representation

Two types of binary analysis: Static analysis vs Dynamic analysis

- Static analysis: Read and understand binary
 - + Give deep understanding
(e.g., FUN_100000e20 == decode_string?
 - + Can find an input to make "Hello World")
 - Time consuming
 - Error-prone
- Dynamic analysis: Run and infer from results
 - + Understand logic without expensive analysis
(e.g., FUN_100000e20(cryptic, 12) gives us "Hello World",
=> FUN_100000e20 == decode_string?)
 - Shallow understanding

```
void FUN_100000e20(byte *param_1,ulong param_2)
{
    int local_1c;
    byte *local_10;

    local_1c = 0;
    local_10 = param_1;
    while ((ulong)(long)local_1c < param_2) {
        *local_10 = *local_10 ^ 0x2a;
        local_10 = local_10 + 1;
        local_1c = local_1c + 1;
    }
    return;
}
```

x86 assembly

```
int add(int a, int b) {  
    return a + b;  
}  
  
int main() {  
    int a = 3;  
    int b = 7;  
    add(a, b);  
}
```

```
; add  
0x08048426 <+0>:      push    ebp  
0x08048427 <+1>:      mov     ebp,esp  
0x08048429 <+3>:      mov     edx,DWORD PTR [ebp+0x8]  
0x0804842c <+6>:      mov     eax,DWORD PTR [ebp+0xc]  
0x0804842f <+9>:      add     eax,edx  
0x08048431 <+11>:     pop     ebp  
0x08048432 <+12>:     ret  
  
; main  
0x08048403 <+0>:      push    ebp  
0x08048404 <+1>:      mov     ebp,esp  
0x08048406 <+3>:      sub     esp,0x8  
0x08048409 <+6>:      mov     DWORD PTR [ebp-0x8],0x3  
0x08048410 <+13>:     mov     DWORD PTR [ebp-0x4],0x7  
0x08048417 <+20>:     push   DWORD PTR [ebp-0x4]  
0x0804841a <+23>:     push   DWORD PTR [ebp-0x8]  
0x0804841d <+26>:     call   0x80483f6 <add>  
0x08048422 <+31>:     add     esp,0x8  
0x08048425 <+34>:     mov     eax,0x0  
0x0804842a <+39>:     leave  
0x0804842b <+40>:     ret
```

x86 assembly (Registers)

- Small and fast in-CPU memory that can store variables
- General purpose registers
 - 64bit: rax, rbx, rcx, rdx, rdi, rsi, r8-r15, etc. (8byte)
 - 32bit: eax, ebx, ecx, edx, edi, esi (4byte)



x86 assembly (Special registers)

- Program counter
 - 64bit: rip
 - 32bit: eip
- Stack management
 - 64bit: rsp, rbp
 - 32bit: ebp, esp

x86 assembly (Syntax)

- [operator] [operand1], [operand2], etc.
- Two types of x86 assembly syntax styles exist
 - Intel syntax: [operator] [destination], [source]
 - AT&T syntax: [operator] [source], [destination]
- We will use Intel syntax in this course

x86 assembly (Examples)

- `mov eax, 1` – Store the value 1 into the eax register
- `mov eax, ebx` – Move the value of ebx to eax, (i.e., $\text{eax} = \text{ebx}$)
- `add eax, 1` – Add 1 to eax, (i.e., $\text{eax} = \text{eax} + 1$)
- `imul eax, ebx` – Multiply ebx and eax and store the result into eax

- Others: `sub`, `xor`, `or`, `and`, `shl`, `ashr`, `lshr`, ...

x86 assembly (More examples)

- `lea eax, DWORD PTR [ebp - 4]`
 - Load effective address; Store ebp-4 to eax
- `mov eax, DWORD PTR [ebp - 4]`
 - Move the value stored at address ebp-4 to eax
- `mov DWORD PTR [ebp - 4], eax`
 - Store the value of eax to the address of ebp-4

x86 assembly (More examples)

- `mov eax, DWORD PTR [ebp - 4]`
 - Move 4 bytes stored at address `ebp-4` to `eax`
- `mov ax, WORD PTR [ebp - 4]`
 - Move 2 bytes stored at address `ebp-4` to `ax`
- `mov al, BYTE PTR [ebp - 4]`
 - Move 1 byte stored at address `ebp-4` to `al`

x86 assembly (Stack)

- `push src`
== `sub esp, 4`
`mov [esp], src`
- `pop dst`
== `mov $dst, [esp]`
`add esp, 4`
- `leave`
== `mov esp, ebp`
`pop ebp`

x86 assembly (Control flow)

- `cmp DWORD PTR [ebp - 4], 0x1234`
 - Compare the value of the address `ebp-4` to `0x1234` and set EFLAGS
 - i.e., if (`i == 0x1234`)
- Conditional jump
 - `je 0x8048442 <main+94>` ← Equal or zero
 - `jne 0x8048442 <main+94>` ← Not equal or non-zero
 - `jle 0x8048442 <main+94>` ← Less or equal, signed
 - `jbe 0x8048442 <main+94>` ← Below or equal, unsigned
 - `jge 0x8048442 <main+94>` ← Greater or equal, signed
 - `jae 0x8048442 <main+94>` ← Above or equal, unsigned

x86 assembly (Control flow)

- Unconditional jump

- `jmp 0x8048442 <main+94>`

- Function call

- `call 0x8048426 <add>`

- Push a next instruction address (i.e., return address) to stack and jump

- `ret == pop eip`

Endianness

- Byte order to store multi-byte data
 - Big Endian: Most significant byte -> Lowest address
 - Little Endian: Most significant byte -> Highest address
- e.g., `mov DWORD PTR [eax], 0x41424344`

0x08048003	0x44
0x08048002	0x43
0x08048001	0x42
0x08048000	0x41

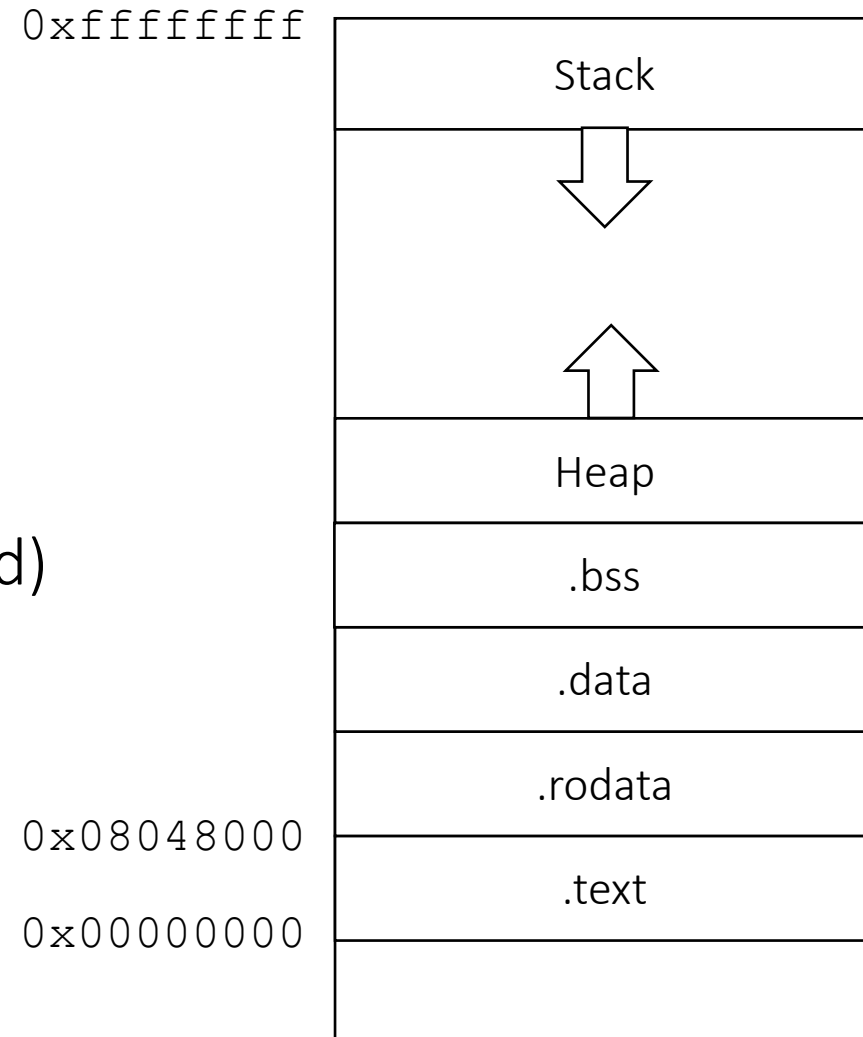
Big Endian

0x08048003	0x41
0x08048002	0x42
0x08048001	0x43
0x08048000	0x44

Little Endian (x86)

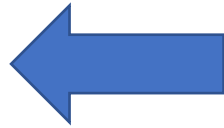
Memory Layout

- Stack
 - Local variables, call contexts, ...
 - Up to 8MB in Linux by default
- Heap
 - Dynamically allocated data
- .bss: R/W global variables (not initialized)
- .data: R/W global variables
- .rodata
 - Read-only data
 - e.g., “Hello World”
- .text: Read-only code



Call stack

```
void baz() {  
    bar();  
    ...  
}  
  
void bar() {  
    foo();  
    ...  
}  
  
void foo() {  
    ...  
}
```



ebp

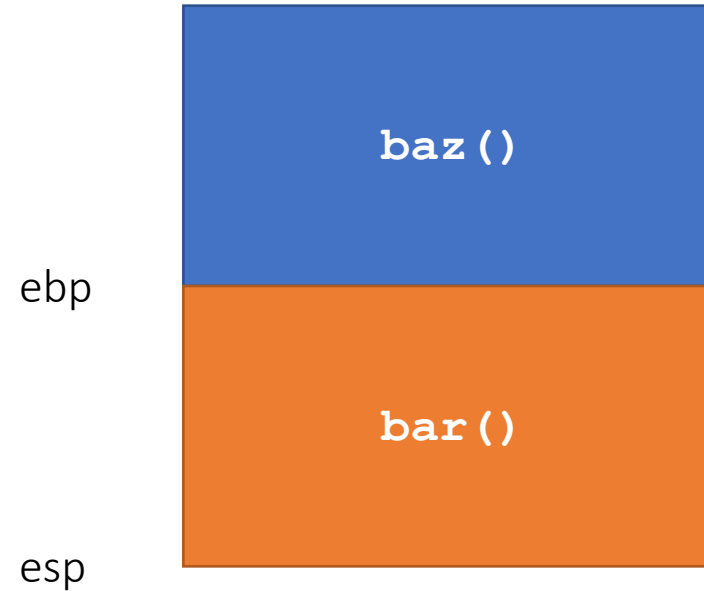
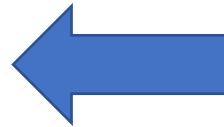


baz()

esp

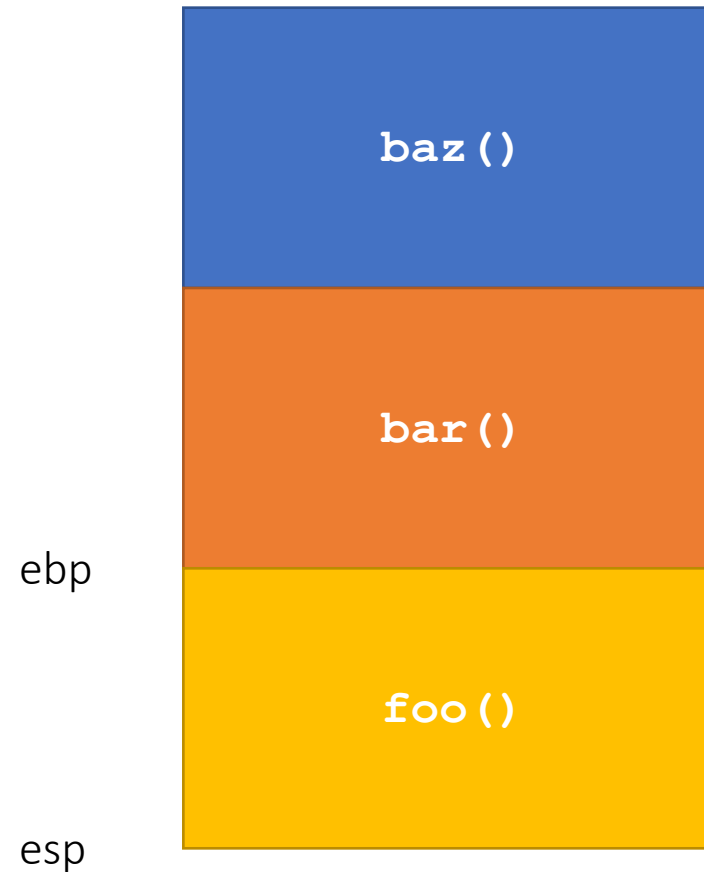
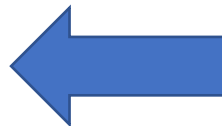
Call stack

```
void baz() {  
    bar();  
    ...  
}  
  
void bar() {  
    foo();  
    ...  
}  
  
void foo() {  
    ...  
}
```



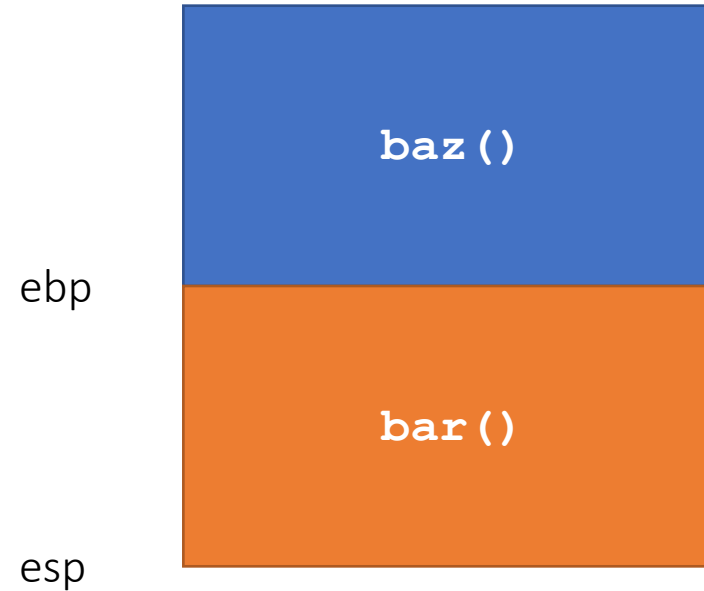
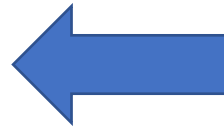
Call stack

```
void baz() {  
    bar();  
    ...  
}  
  
void bar() {  
    foo();  
    ...  
}  
  
void foo() {  
    ...  
}
```



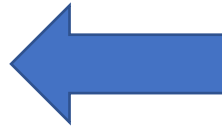
Call stack

```
void baz() {  
    bar();  
    ...  
}  
  
void bar() {  
    foo();  
    ...  
}  
  
void foo() {  
    ...  
}
```



Call stack

```
void baz() {  
    bar();  
    ...  
}  
  
void bar() {  
    foo();  
    ...  
}  
  
void foo() {  
    ...  
}
```

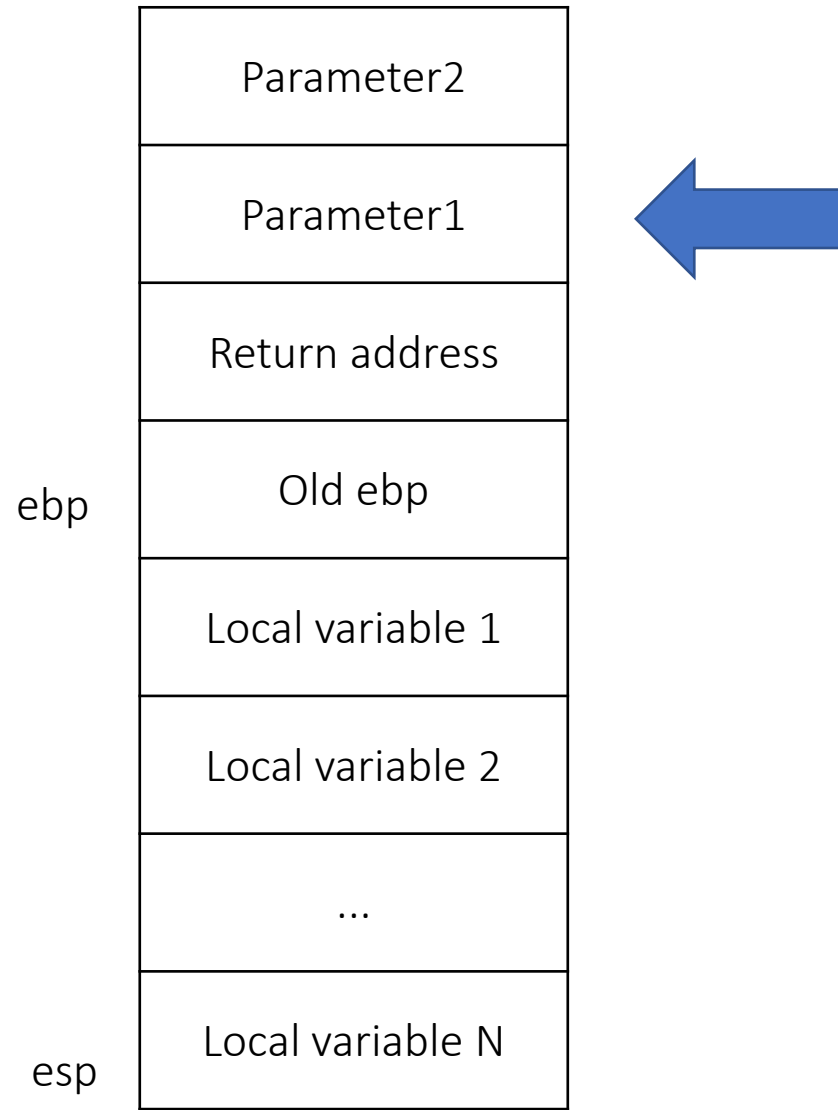


ebp

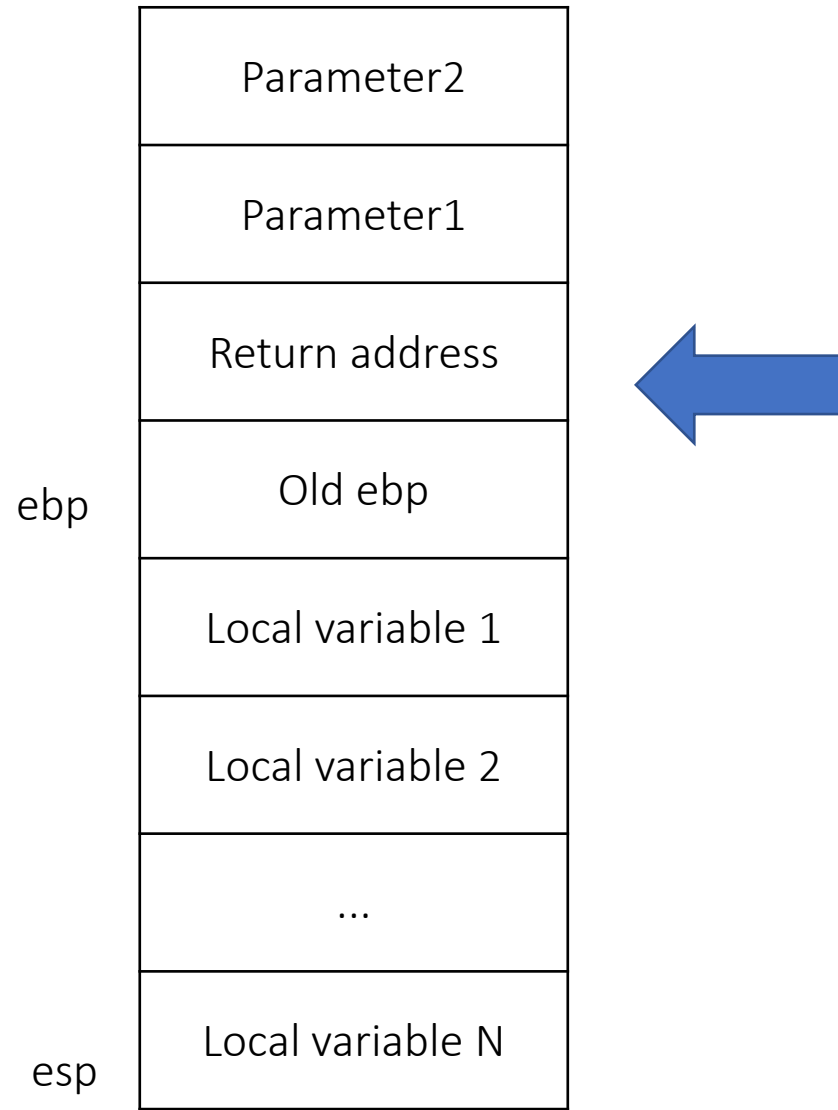


esp

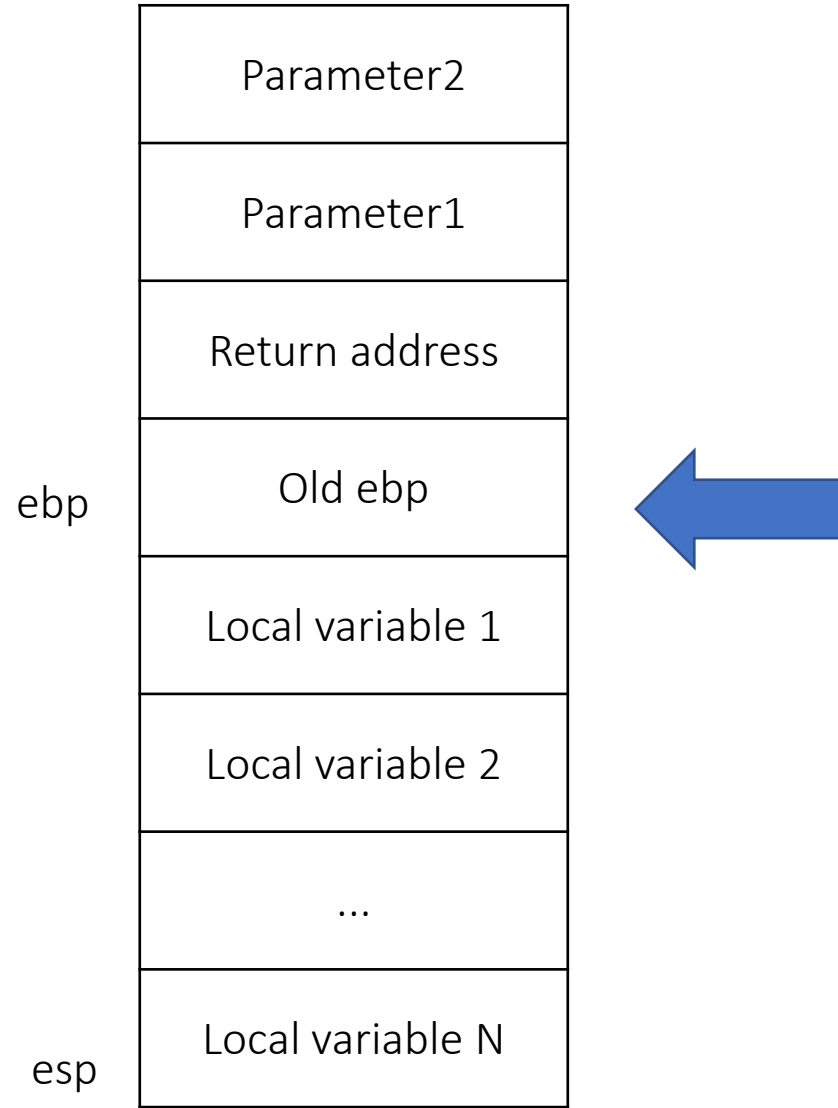
Stack frame



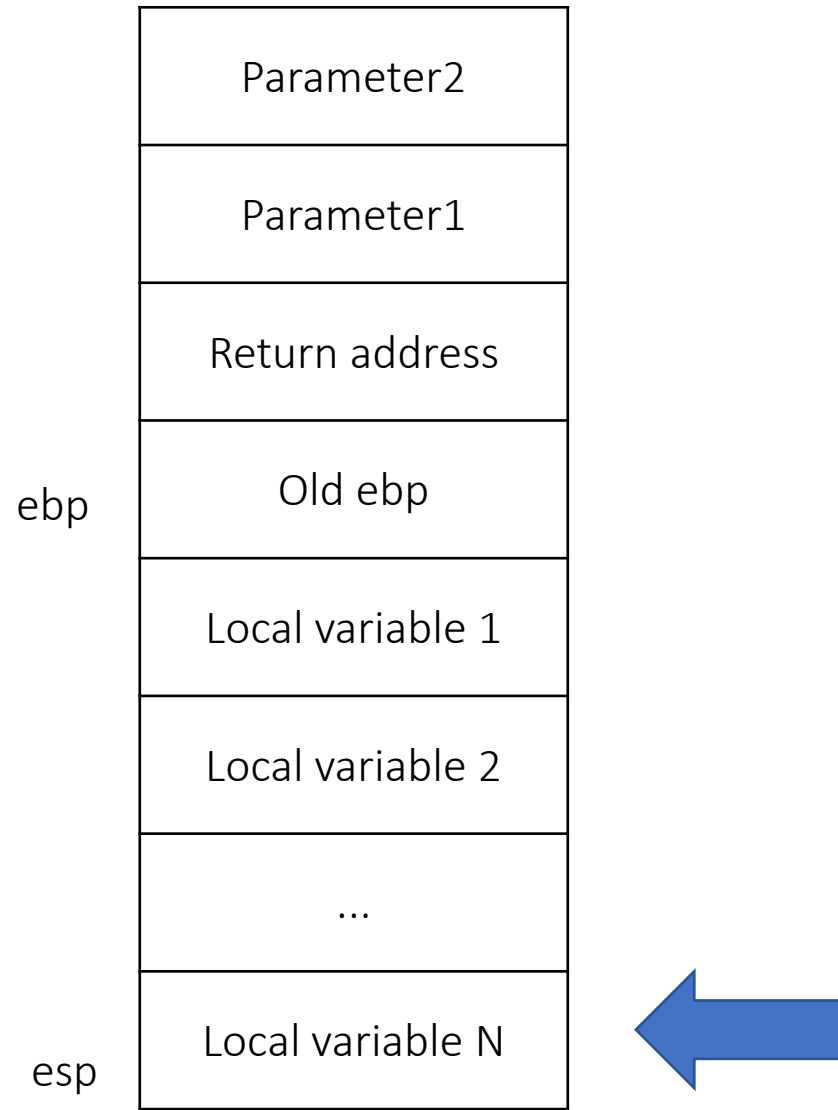
Stack frame



Stack frame



Stack frame



x86 Calling convention (GCC)

- Function arguments
 - Save to stack
- Return value
 - eax
- Register preservation
 - Callee-saved: ebp, edi, esi, ebx
 - Caller-saved: others

x86 assembly

```
int add(int a, int b) {  
    return a + b;  
}  
  
int main() {  
    int a = 3;  
    int b = 7;  
    add(a, b);  
}
```

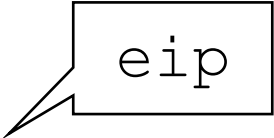
```
; add  
0x08048426 <+0>:      push    ebp  
0x08048427 <+1>:      mov     ebp,esp  
0x08048429 <+3>:      mov     edx,DWORD PTR [ebp+0x8]  
0x0804842c <+6>:      mov     eax,DWORD PTR [ebp+0xc]  
0x0804842f <+9>:      add     eax,edx  
0x08048431 <+11>:     pop     ebp  
0x08048432 <+12>:     ret  
  
; main  
0x08048403 <+0>:      push    ebp  
0x08048404 <+1>:      mov     ebp,esp  
0x08048406 <+3>:      sub     esp,0x8  
0x08048409 <+6>:      mov     DWORD PTR [ebp-0x8],0x3  
0x08048410 <+13>:     mov     DWORD PTR [ebp-0x4],0x7  
0x08048417 <+20>:     push   DWORD PTR [ebp-0x4]  
0x0804841a <+23>:     push   DWORD PTR [ebp-0x8]  
0x0804841d <+26>:     call   0x80483f6 <add>  
0x08048422 <+31>:     add     esp,0x8  
0x08048425 <+34>:     mov     eax,0x0  
0x0804842a <+39>:     leave  
0x0804842b <+40>:     ret
```

esp



```
; add
0x08048426 <+0>:      push    ebp
0x08048427 <+1>:      mov     ebp,esp
0x08048429 <+3>:      mov     edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:      mov     eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:      add     eax,edx
0x08048431 <+11>:     pop     ebp
0x08048432 <+12>:     ret

; main
0x08048403 <+0>:      push    ebp
0x08048404 <+1>:      mov     ebp,esp
0x08048406 <+3>:      sub     esp,0x8
0x08048409 <+6>:      mov     DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:     mov     DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:     push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:     push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:     call   0x80483f6 <add>
0x08048422 <+31>:     add     esp,0x8
0x08048425 <+34>:     mov     eax,0x0
0x0804842a <+39>:     leave
0x0804842b <+40>:     ret
```



esp

...
main's return address
main's old ebp

```
; add
0x08048426 <+0>:      push   ebp
0x08048427 <+1>:      mov    ebp,esp
0x08048429 <+3>:      mov    edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:      mov    eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:      add   eax,edx
0x08048431 <+11>:     pop   ebp
0x08048432 <+12>:     ret

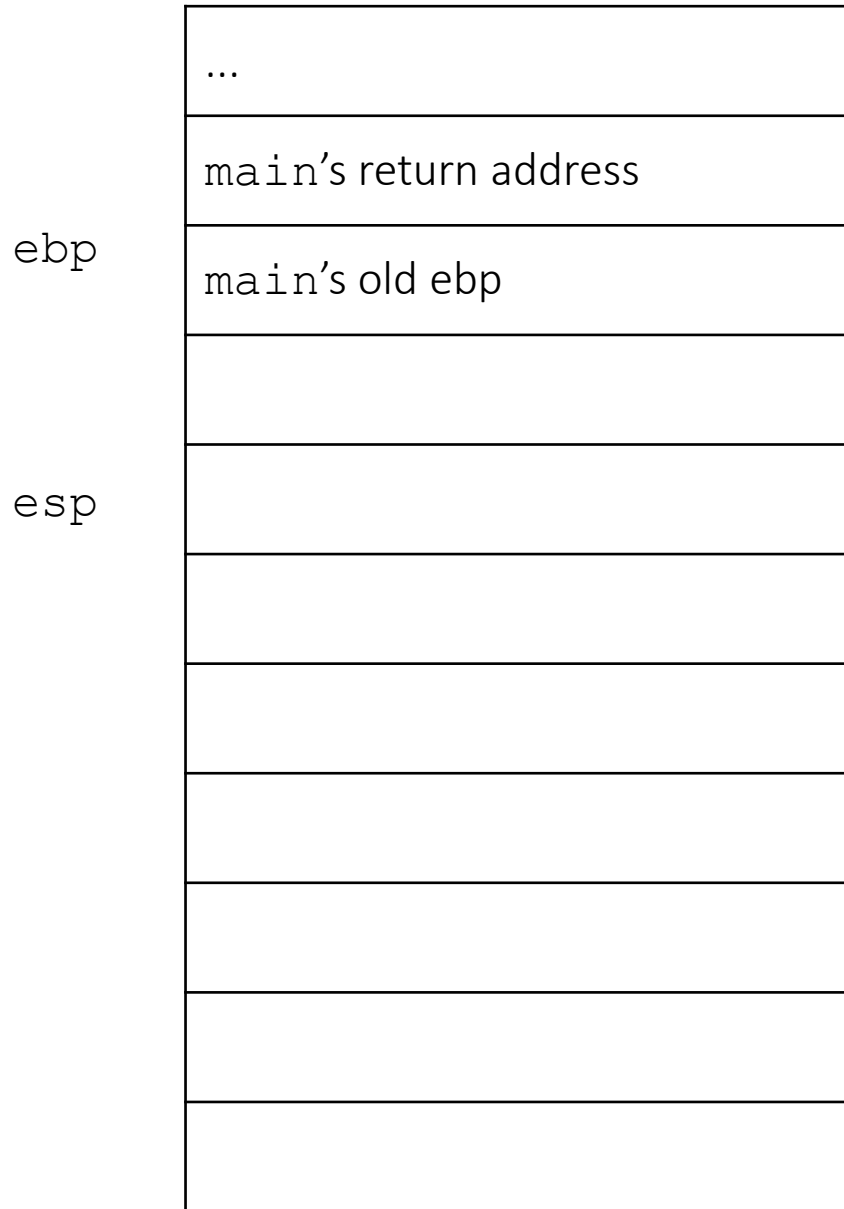
; main
0x08048403 <+0>:      push   ebp
0x08048404 <+1>:      mov    ebp,esp
0x08048406 <+3>:      sub   esp,0x8
0x08048409 <+6>:      mov   DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:     mov   DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:     push  DWORD PTR [ebp-0x4]
0x0804841a <+23>:     push  DWORD PTR [ebp-0x8]
0x0804841d <+26>:     call  0x80483f6 <add>
0x08048422 <+31>:     add   esp,0x8
0x08048425 <+34>:     mov   eax,0x0
0x0804842a <+39>:     leave
0x0804842b <+40>:     ret
```

ebp
esp

...
main's return address
main's old ebp

```
; add
0x08048426 <+0>:      push    ebp
0x08048427 <+1>:      mov     ebp,esp
0x08048429 <+3>:      mov     edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:      mov     eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:      add     eax,edx
0x08048431 <+11>:     pop     ebp
0x08048432 <+12>:     ret

; main
0x08048403 <+0>:      push    ebp
0x08048404 <+1>:      mov     ebp,esp
0x08048406 <+3>:      sub     esp,0x8
0x08048409 <+6>:      mov     DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:     mov     DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:     push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:     push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:     call   0x80483f6 <add>
0x08048422 <+31>:     add     esp,0x8
0x08048425 <+34>:     mov     eax,0x0
0x0804842a <+39>:     leave
0x0804842b <+40>:     ret
```

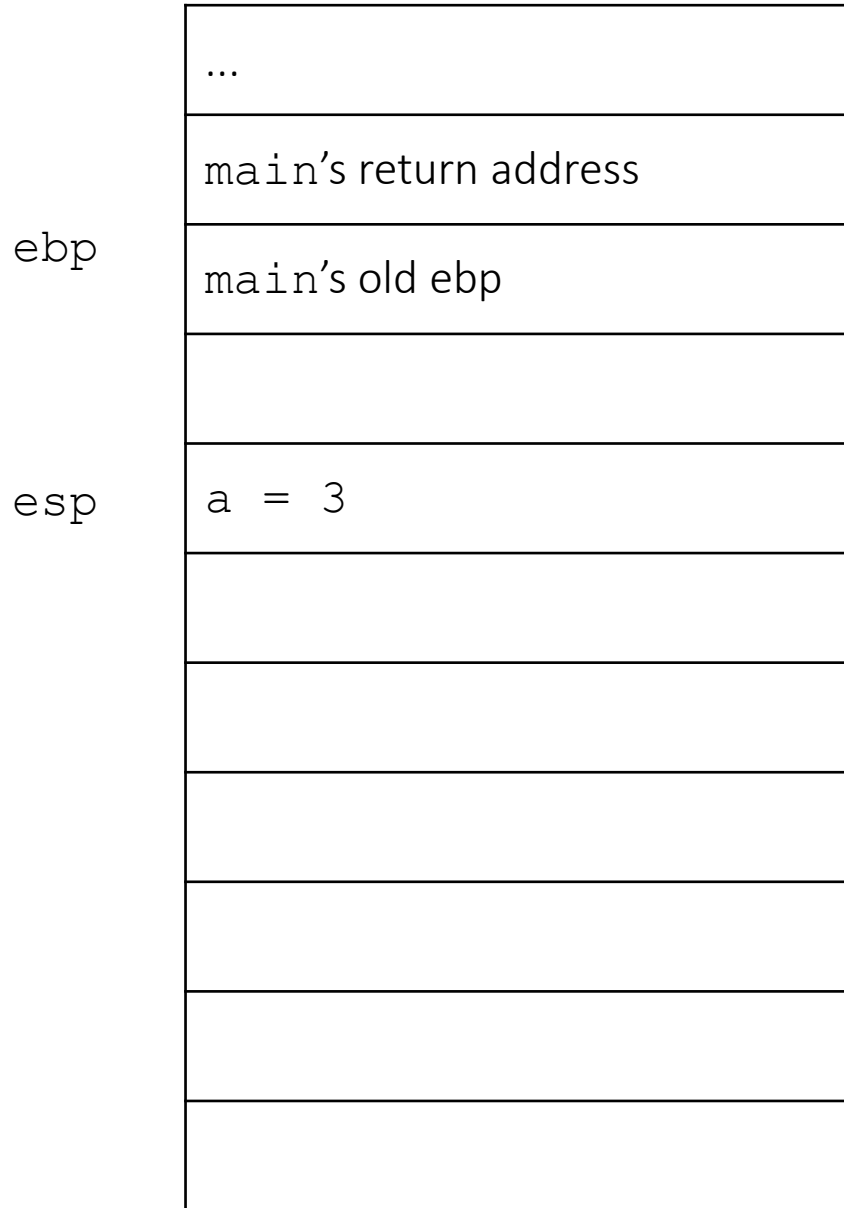


```

; add
0x08048426 <+0>:      push    ebp
0x08048427 <+1>:      mov     ebp,esp
0x08048429 <+3>:      mov     edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:      mov     eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:      add     eax,edx
0x08048431 <+11>:     pop     ebp
0x08048432 <+12>:     ret

; main
0x08048403 <+0>:      push    ebp
0x08048404 <+1>:      mov     ebp,esp
0x08048406 <+3>:      sub     esp,0x8
0x08048409 <+6>:      mov     DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:     mov     DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:     push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:     push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:     call   0x80483f6 <add>
0x08048422 <+31>:     add     esp,0x8
0x08048425 <+34>:     mov     eax,0x0
0x0804842a <+39>:     leave
0x0804842b <+40>:     ret

```

```

; add
0x08048426 <+0>:      push    ebp
0x08048427 <+1>:      mov     ebp,esp
0x08048429 <+3>:      mov     edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:      mov     eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:      add     eax,edx
0x08048431 <+11>:     pop     ebp
0x08048432 <+12>:     ret

; main
0x08048403 <+0>:      push    ebp
0x08048404 <+1>:      mov     ebp,esp
0x08048406 <+3>:      sub     esp,0x8
0x08048409 <+6>:      mov     DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:     mov     DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:     push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:     push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:     call   0x80483f6 <add>
0x08048422 <+31>:     add     esp,0x8
0x08048425 <+34>:     mov     eax,0x0
0x0804842a <+39>:     leave
0x0804842b <+40>:     ret

```

	...
	main's return address
ebp	main's old ebp
	b = 7
esp	a = 3

```

; add
0x08048426 <+0>:      push    ebp
0x08048427 <+1>:      mov     ebp,esp
0x08048429 <+3>:      mov     edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:      mov     eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:      add     eax,edx
0x08048431 <+11>:     pop     ebp
0x08048432 <+12>:     ret

; main
0x08048403 <+0>:      push    ebp
0x08048404 <+1>:      mov     ebp,esp
0x08048406 <+3>:      sub     esp,0x8
0x08048409 <+6>:      mov     DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:     mov     DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:     push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:     push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:     call   0x80483f6 <add>
0x08048422 <+31>:     add     esp,0x8
0x08048425 <+34>:     mov     eax,0x0
0x0804842a <+39>:     leave
0x0804842b <+40>:     ret

```

ebp

...
main's return address
main's old ebp
b = 7
a = 3
7

esp

```
; add
0x08048426 <+0>:      push    ebp
0x08048427 <+1>:      mov     ebp,esp
0x08048429 <+3>:      mov     edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:      mov     eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:      add     eax,edx
0x08048431 <+11>:     pop     ebp
0x08048432 <+12>:     ret

; main
0x08048403 <+0>:      push    ebp
0x08048404 <+1>:      mov     ebp,esp
0x08048406 <+3>:      sub     esp,0x8
0x08048409 <+6>:      mov     DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:     mov     DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:     push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:     push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:     call   0x80483f6 <add>
0x08048422 <+31>:     add     esp,0x8
0x08048425 <+34>:     mov     eax,0x0
0x0804842a <+39>:     leave
0x0804842b <+40>:     ret
```

ebp

...
main's return address
main's old ebp
b = 7
a = 3
7
3

esp

```
; add
0x08048426 <+0>:      push    ebp
0x08048427 <+1>:      mov     ebp,esp
0x08048429 <+3>:      mov     edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:      mov     eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:      add     eax,edx
0x08048431 <+11>:     pop     ebp
0x08048432 <+12>:     ret

; main
0x08048403 <+0>:      push    ebp
0x08048404 <+1>:      mov     ebp,esp
0x08048406 <+3>:      sub     esp,0x8
0x08048409 <+6>:      mov     DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:     mov     DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:     push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:     push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:     call   0x80483f6 <add>
0x08048422 <+31>:     add     esp,0x8
0x08048425 <+34>:     mov     eax,0x0
0x0804842a <+39>:     leave
0x0804842b <+40>:     ret
```

ebp

...
main's return address
main's old ebp
b = 7
a = 3
7
3
add's return address

esp

```
; add
0x08048426 <+0>:      push  ebp
0x08048427 <+1>:      mov   ebp,esp
0x08048429 <+3>:      mov   edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:      mov   eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:      add   eax,edx
0x08048431 <+11>:     pop   ebp
0x08048432 <+12>:     ret

; main
0x08048403 <+0>:      push  ebp
0x08048404 <+1>:      mov   ebp,esp
0x08048406 <+3>:      sub   esp,0x8
0x08048409 <+6>:      mov   DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:     mov   DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:     push  DWORD PTR [ebp-0x4]
0x0804841a <+23>:     push  DWORD PTR [ebp-0x8]
0x0804841d <+26>:     call  0x80483f6 <add>
0x08048422 <+31>:     add   esp,0x8
0x08048425 <+34>:     mov   eax,0x0
0x0804842a <+39>:     leave
0x0804842b <+40>:     ret
```

ebp

...
main's return address
main's old ebp
b = 7
a = 3
7
3
add's return address
add's old ebp

esp

```
; add
0x08048426 <+0>:      push   ebp
0x08048427 <+1>:      mov    ebp,esp
0x08048429 <+3>:      mov    edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:      mov    eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:      add    eax,edx
0x08048431 <+11>:     pop    ebp
0x08048432 <+12>:     ret

; main
0x08048403 <+0>:      push   ebp
0x08048404 <+1>:      mov    ebp,esp
0x08048406 <+3>:      sub    esp,0x8
0x08048409 <+6>:      mov    DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:     mov    DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:     push  DWORD PTR [ebp-0x4]
0x0804841a <+23>:     push  DWORD PTR [ebp-0x8]
0x0804841d <+26>:     call  0x80483f6 <add>
0x08048422 <+31>:     add    esp,0x8
0x08048425 <+34>:     mov    eax,0x0
0x0804842a <+39>:     leave
0x0804842b <+40>:     ret
```

ebp
esp

...
main's return address
main's old ebp
b = 7
a = 3
7
3
add's return address
add's old ebp

```
; add
0x08048426 <+0>:      push    ebp
0x08048427 <+1>:      mov     ebp,esp
0x08048429 <+3>:      mov     edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:      mov     eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:      add     eax,edx
0x08048431 <+11>:     pop     ebp
0x08048432 <+12>:     ret

; main
0x08048403 <+0>:      push    ebp
0x08048404 <+1>:      mov     ebp,esp
0x08048406 <+3>:      sub     esp,0x8
0x08048409 <+6>:      mov     DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:     mov     DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:     push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:     push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:     call   0x80483f6 <add>
0x08048422 <+31>:     add     esp,0x8
0x08048425 <+34>:     mov     eax,0x0
0x0804842a <+39>:     leave
0x0804842b <+40>:     ret
```

ebp
esp

...
main's return address
main's old ebp
b = 7
a = 3
7
3
add's return address
add's old ebp

```
; add
0x08048426 <+0>:      push    ebp
0x08048427 <+1>:      mov     ebp,esp
0x08048429 <+3>:      mov     edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:      mov     eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:      add     eax,edx
0x08048431 <+11>:     pop     ebp
0x08048432 <+12>:     ret

; main
0x08048403 <+0>:      push    ebp
0x08048404 <+1>:      mov     ebp,esp
0x08048406 <+3>:      sub     esp,0x8
0x08048409 <+6>:      mov     DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:     mov     DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:     push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:     push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:     call   0x80483f6 <add>
0x08048422 <+31>:     add     esp,0x8
0x08048425 <+34>:     mov     eax,0x0
0x0804842a <+39>:     leave
0x0804842b <+40>:     ret
```

edx = 3

0x0804842c <+6>: mov eax,DWORD PTR [ebp+0xc]

ebp
esp

...
main's return address
main's old ebp
b = 7
a = 3
7
3
add's return address
add's old ebp

```
; add
0x08048426 <+0>:      push    ebp
0x08048427 <+1>:      mov     ebp,esp
0x08048429 <+3>:      mov     edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:      mov     eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:      add     eax,edx
0x08048431 <+11>:     pop     ebp
0x08048432 <+12>:     ret

; main
0x08048403 <+0>:      push    ebp
0x08048404 <+1>:      mov     ebp,esp
0x08048406 <+3>:      sub     esp,0x8
0x08048409 <+6>:      mov     DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:     mov     DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:     push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:     push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:     call   0x80483f6 <add>
0x08048422 <+31>:     add     esp,0x8
0x08048425 <+34>:     mov     eax,0x0
0x0804842a <+39>:     leave
0x0804842b <+40>:     ret
```

eax = 7



...
main's return address
main's old ebp
b = 7
a = 3
7
3
add's return address
add's old ebp

ebp
esp

```

; add
0x08048426 <+0>:      push    ebp
0x08048427 <+1>:      mov     ebp,esp
0x08048429 <+3>:      mov     edx,DWORD PTR [eax]
0x0804842c <+6>:      mov     eax,DWORD PTR [eax]
0x0804842f <+9>:      add    eax,edx
0x08048431 <+11>:     pop    ebp
0x08048432 <+12>:     ret

; main
0x08048403 <+0>:      push    ebp
0x08048404 <+1>:      mov     ebp,esp
0x08048406 <+3>:      sub    esp,0x8
0x08048409 <+6>:      mov     DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:     mov     DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:     push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:     push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:     call   0x80483f6 <add>
0x08048422 <+31>:     add    esp,0x8
0x08048425 <+34>:     mov     eax,0x0
0x0804842a <+39>:     leave
0x0804842b <+40>:     ret

```

eax
= 7 + 3 = 10

ebp

...
main's return address
main's old ebp
b = 7
a = 3
7
3
add's return address
add's old ebp

esp

```
; add
0x08048426 <+0>:      push    ebp
0x08048427 <+1>:      mov     ebp,esp
0x08048429 <+3>:      mov     edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:      mov     eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:      add     eax,edx
0x08048431 <+11>:     pop     ebp
0x08048432 <+12>:     ret

; main
0x08048403 <+0>:      push    ebp
0x08048404 <+1>:      mov     ebp,esp
0x08048406 <+3>:      sub     esp,0x8
0x08048409 <+6>:      mov     DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:     mov     DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:     push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:     push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:     call   0x80483f6 <add>
0x08048422 <+31>:     add     esp,0x8
0x08048425 <+34>:     mov     eax,0x0
0x0804842a <+39>:     leave
0x0804842b <+40>:     ret
```

	...
	main's return address
ebp	main's old ebp
	b = 7
	a = 3
	7
esp	3
	add's return address
	add's old ebp

```

; add
0x08048426 <+0>:      push    ebp
0x08048427 <+1>:      mov     ebp,esp
0x08048429 <+3>:      mov     edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:      mov     eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:      add     eax,edx
0x08048431 <+11>:     pop     ebp
0x08048432 <+12>:     ret

; main
0x08048403 <+0>:      push    ebp
0x08048404 <+1>:      mov     ebp,esp
0x08048406 <+3>:      sub     esp,0x8
0x08048409 <+6>:      mov     DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:     mov     DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:     push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:     push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:     call   0x80483f6 <add>
0x08048422 <+31>:     add     esp,0x8
0x08048425 <+34>:     mov     eax,0x0
0x0804842a <+39>:     leave
0x0804842b <+40>:     ret

```

	...
	main's return address
ebp	main's old ebp
	b = 7
esp	a = 3
	7
	3
	add's return address
	add's old ebp

```

; add
0x08048426 <+0>:      push    ebp
0x08048427 <+1>:      mov     ebp,esp
0x08048429 <+3>:      mov     edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:      mov     eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:      add     eax,edx
0x08048431 <+11>:     pop     ebp
0x08048432 <+12>:     ret

; main
0x08048403 <+0>:      push    ebp
0x08048404 <+1>:      mov     ebp,esp
0x08048406 <+3>:      sub     esp,0x8
0x08048409 <+6>:      mov     DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:     mov     DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:     push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:     push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:     call   0x80483f6 <add>
0x08048422 <+31>:     add     esp,0x8
0x08048425 <+34>:     mov     eax,0x0
0x0804842a <+39>:     leave
0x0804842b <+40>:     ret

```

	...
	main's return address
ebp	main's old ebp
	b = 7
esp	a = 3
	7
	3
	add's return address
	add's old ebp

```

; add
0x08048426 <+0>:      push    ebp
0x08048427 <+1>:      mov     ebp,esp
0x08048429 <+3>:      mov     edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:      mov     eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:      add    eax,edx
0x08048431 <+11>:     pop    ebp
0x08048432 <+12>:     ret

; main
0x08048403 <+0>:      push    ebp
0x08048404 <+1>:      mov     ebp,esp
0x08048406 <+3>:      sub    esp,0x8
0x08048409 <+6>:      mov     DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:     mov     DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:     push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:     push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:     call   0x80483f6 <add>
0x08048422 <+31>:     add    esp,0x8
0x08048425 <+34>:     mov    eax,0x0
0x0804842a <+39>:     leave
0x0804842b <+40>:     ret

```

esp

...
main's return address
main's old ebp
b = 7
a = 3
7
3
add's return address
add's old ebp

```
; add
0x08048426 <+0>:      push    ebp
0x08048427 <+1>:      mov     ebp,esp
0x08048429 <+3>:      mov     edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:      mov     eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:      add     eax,edx
0x08048431 <+11>:     pop     ebp
0x08048432 <+12>:     ret

; main
0x08048403 <+0>:      push    ebp
0x08048404 <+1>:      mov     ebp,esp
0x08048406 <+3>:      sub     esp,0x8
0x08048409 <+6>:      mov     DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:     mov     DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:     push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:     push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:     call   0x80483f6 <add>
0x08048422 <+31>:     add     esp,0x8
0x08048425 <+34>:     mov     eax,0x0
0x0804842a <+39>:     leave
0x0804842b <+40>:     ret
```

x86 Calling convention (GCC)

- Function arguments
 - Save to stack
- Return value
 - eax
- Register preservation
 - Callee-saved: ebp, edi, esi, ebx
 - Caller-saved: others

x86 (32bit)

- Function arguments
 - rdi, rsi, rdx, rcx, r8, r9
 - Save to stack
- Return value
 - rax
- Register preservation
 - Callee-saved: rbp, rdi, rsi, rbx, r12-r15
 - Caller-saved: others

x86-64

Tool for dynamic analysis: GNU DeBugger (GDB)

- Can read assembly of a program
 - `disassemble main`
- Can read the status of a program such as registers and memory
 - `x/[Length?][Format] [Expression]`
 - `x/wx $eax` (print the value of eax as 32-bit integer)
 - `x/s 0x804858f` (read the string value at the address 0x804858f)
 - `x/wx 0x804858f` (read the integer value at the address 0x804858f)
- Can set a breakpoint
 - `b main` (break if main function is called)
 - `b *main+53` (break before running the instruction at main+53)

GDB cont.

- Execute a program
 - `r`
- Controlling the execution after a break
 - `c` (continue to a next breakpoint)
 - `ni` (run a instruction, do not get into the function)
 - `si` (run an instruction, get into the function)

pwndbg

- A gdb plugin for exploit development
- We highly recommend to use pwndbg (which is install by default) for your further assignment
- Ref: <https://github.com/pwndbg/pwndbg>

```
Breakpoint 1, main (argc=1, argv=0xffffdf04) at bomb.c:22
22      bomb.c: No such file or directory.
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA

EAX 0xf7fc6dbc (environ) -> 0xffffdf0c ← 0x0
EBX 0xffffde70 ← 0x1
ECX 0xffffde70 ← 0x1
EDX 0xffffde94 ← 0x0
EDI 0xf7fc5000 (_GLOBAL_OFFSET_TABLE_) ← 0x1b2db0
ESI 0xf7fc5000 (_GLOBAL_OFFSET_TABLE_) ← 0x1b2db0
EBP 0xffffde58 ← 0x0
ESP 0xffffde50 -> 0xffffde70 ← 0x1
EIP 0x80488ec (main+17) ← cmp    dword ptr [ebx], 1

> 0x80488ec <main+17>    cmp    dword ptr [ebx], 1
↓
0x80488f1 <main+22>    mov    eax, dword ptr [stdin@GLIBC_2.0] <0x804c600>
0x80488f6 <main+27>    mov    dword ptr [infile], eax <0x804c614>
0x80488fb <main+32>    jmp   main+151 <0x8048972>
↓
0x8048972 <main+151>   call  set_apikey <0x80498d7>
0x8048977 <main+156>   call  initialize_bomb <0x8048e04>
0x804897c <main+161>   sub    esp, 0xc
0x804897f <main+164>   push  0x8049bec
0x8048984 <main+169>   call  puts@plt <0x80486e0>
0x8048989 <main+174>   add    esp, 0x10
0x804898c <main+177>   sub    esp, 0xc

00:0000 | esp 0xffffde50 -> 0xffffde70 ← 0x1
01:0004 | 0xffffde54 ← 0x0
... ↓
03:000c | 0xffffde5c -> 0xf7e2a647 (__libc_start_main+247) ← add    esp, 0x10
04:0010 | 0xffffde60 -> 0xf7fc5000 (_GLOBAL_OFFSET_TABLE_) ← 0x1b2db0
... ↓
06:0018 | 0xffffde68 ← 0x0
07:001c | 0xffffde6c -> 0xf7e2a647 (__libc_start_main+247) ← add    esp, 0x10
```

Tool for static analysis: Ghidra

- Ghidra: A decompiler developed by NSA
 - IDA would be better, but too expensive 😞
- You can download <https://ghidra-sre.org/>

How to use ghidra

- Create a new project: File -> New project
- Import a file: File -> Import a file
 - e.g., `scp -P9000 YOUR_ID@teemo.kaist.ac.kr:/ee517/lab01/tut01-crackme/crackme0x00 ./`
 - NOTE: -P is a capital character unlike ssh
 - Or you can use filezilla for ssh copy

How to use ghidra

- Symbol tree -> Functions -> main

The screenshot displays the Ghidra IDE interface with three main panels highlighted by callouts:

- Symbol Tree:** Located on the left, it shows a hierarchical view of the program's symbols. The `main` function is selected and highlighted in blue.
- Assembly:** The central panel shows the assembly code for the `main` function. The assembly is displayed in a light blue background, with the function signature `int __cdecl main(int argc, char * * argv)` visible. A callout box labeled "Assembly" points to this section.
- Decompiler:** The right panel shows the decompiled C code for the `main` function. The code includes variable declarations, a password check, and a loop. A callout box labeled "Decompiler" points to this section.

At the bottom of the window, the console shows the current instruction being executed: `08048688 main PUSH EBP`.

Decompiler

```
int main(int argc, char **argv)
{
    int iVar1;
    char buf [16];

    puts("IOLI Crackme Level 0x00");
    printf("Password: ");
    scanf("%s", buf);
    iVar1 = strcmp(buf, "250381");
    if (iVar1 == 0) {
        puts("Password OK :)");
        print_key("lab01:tutorial");
    }
    else {
        puts("Invalid Password!");
    }
    return 0;
}
```

Note on bomb

Lose 5 points!

```

      __,--~/~      `---.
     _/_,'---(      ,      )
    __ /          <      /      ) \__
-----===;;;'=====-----===;;;'-----
      \/_  ~'~!~!~!~!~!\~!~)~!~/
      (_ ( \ (      >      \)
      \_( _ <          >_>'
          ~ ` -i'  ::>|--'
              I;|.|.|.
              <|i::|i|\`.
              (` ^''`-' ')

```

BOOM!! The bomb has blown up.

Tip: Try to read assembly as much as you can!

- Recommend: Solve *first THREE* challenges in bomblab without decompiler
- Reason:
 - In shellcode lab, you need to write assembly code
 - Writing exploit requires understanding of assembly (e.g., ROP)
 - We have tedious binaries in later lab challenges, which make a decompiler fail to analyze

Anti-reversing

```
LOAD: 0A04B0D0 loc_A04B0D0:
LOAD: 0A04B0D0
LOAD: 0A04B0D6
LOAD: 0A04B0D9
LOAD: 0A04B0DA
LOAD: 0A04B0DB
LOAD: 0A04B0DC
LOAD: 0A04B0DE
LOAD: 0A04B0DF
LOAD: 0A04B0E1
LOAD: 0A04B0E1 loc_A04B0E1:
LOAD: 0A04B0E1
LOAD: 0A04B0E1
LOAD: 0A04B0E6
LOAD: 0A04B0E7
LOAD: 0A04B0E9
LOAD: 0A04B0EB
LOAD: 0A04B0EC
LOAD: 0A04B0EE

sub esp, 4
mov [esp], esi
push ecx
push edi
push eax
jz short near ptr loc_A04B0E1+2
push eax
jnz short near ptr loc_A04B0E1+1

mov eax, 0BE535258h
xlat
sub dl, [edx]
aad 81h
out dx, al
jz short loc A04B14D
jmp near ptr 70CDE25Dh
```

Annotations:

- `loc_A04B0E1+2` and `loc_A04B0E1+1` are circled in black.
- `70CDE25Dh` is highlighted in red.
- Comments include: `; CODE XREF: start+B↑j`, `; CODE XREF: LOAD:0A04B0DF↑j`, and `; LOAD:0A04B0DC↑j`.
- A black arrow points from the circled `loc_A04B0E1+1` to the `mov eax, 0BE535258h` instruction.