System Security - Attack and Defense for Binaries

CS 4390/5390, Spring *2026*

Instructor: MD Armanuzzaman (***Arman***)

Buffer Overflow

- Stack-based buffer overflow (Sequential buffer overflow)
  - Brief history of buffer overflow
  - Information C function needs to run
  - C calling conventions (x86, x86-64)
  - Overflow local variables
  - Overflow RET address to execute a function
  - 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 worm 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 (misc/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 (misc/globallocalv 32bit)

```
ctf@misc_globallocalv_32:/$ ./misc_globallocalv_32
g_i is at 0x60a97020
g_u is at 0x60a9704c
l_i in main() is at 0xffc4cc8c
l_u in main() is at 0xffc4cc88
l_i in func() is at 0xffc4cc5c
l_u in func() is at 0xffc4cc58
p in func() is at 0xffc4cc70
```

Global and Local Variables (misc/globallocalv 32bit)

```
ctf@misc_globallocalv_64:/$ ./misc_globallocalv_64
g_i is at 0x5a7203332020
g_u is at 0x5a7203332050
l_i in main() is at 0x7fff0829c8fc
l_u in main() is at 0x7fff0829c8f8
l_i in func() is at 0x7fff0829c8cc
l_u in func() is at 0x7fff0829c8c8
p in func() is at 0x7fff0829c8bc
```

C/C++ Function in x86

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

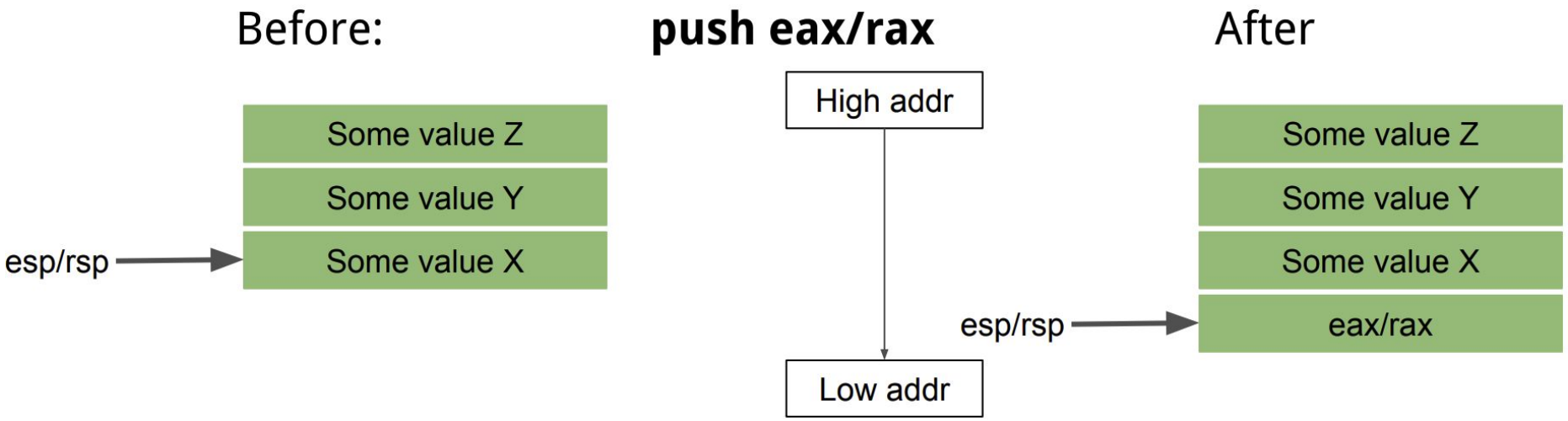
The assembly language supports this on x86

- esp/rsp holds the address of the top of the stack
- push eax/rax **1)** *decrements* the stack pointer (esp/rsp) 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

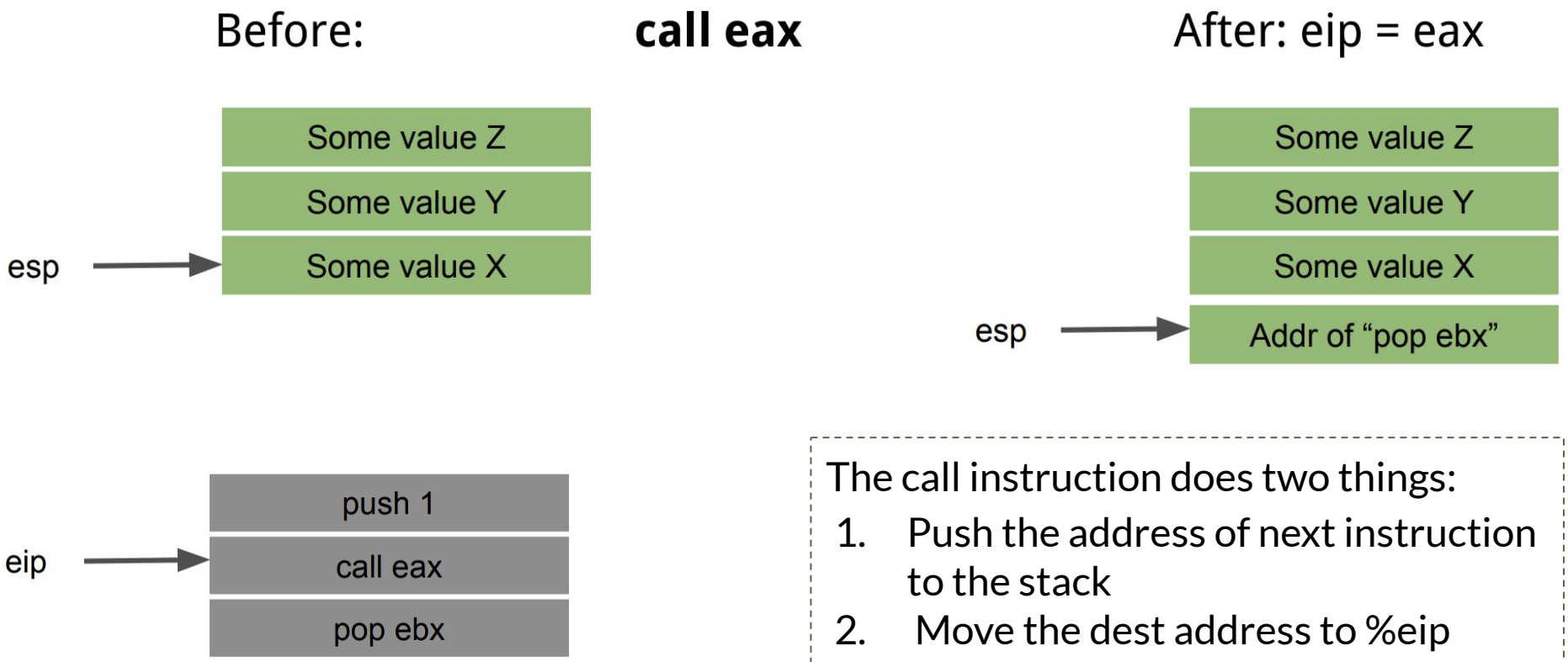
push, pop, call, ret, enter, leave

x86/64 Instructions that affect Stack



x86/64 Instructions that affect Stack

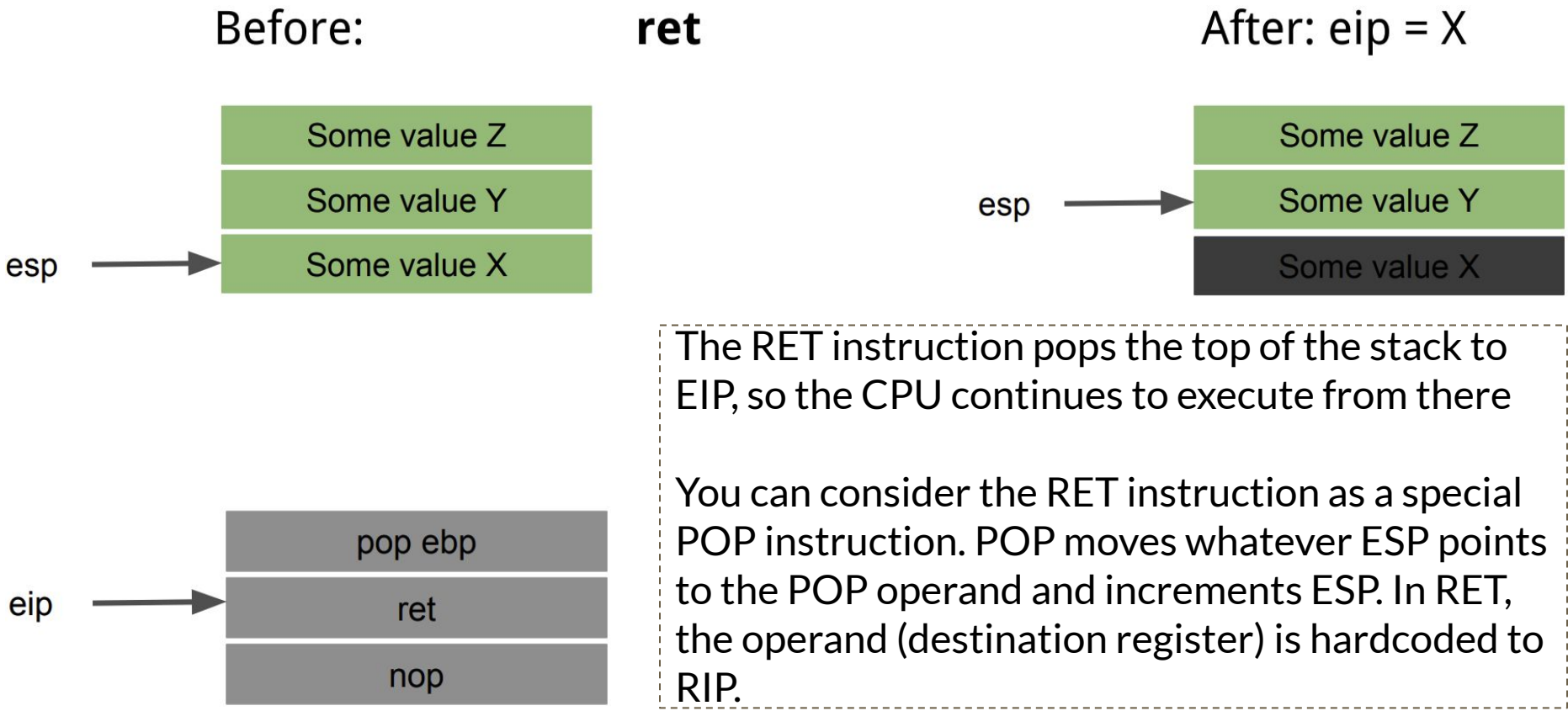
x86/64 Instructions that affect Stack



The call instruction does two things:

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

x86/64 Instructions that affect Stack



The RET instruction pops the top of the stack to EIP, so the CPU continues to execute from there

You can consider the RET instruction as a special POP instruction. POP moves whatever ESP points to the POP operand and increments ESP. In RET, the operand (destination register) is hardcoded to RIP.

x86/64 Instructions that affect Stack

```
push ebp
mov ebp, esp
sub esp, #imm
```

enter

x86/64 Instructions that affect Stack

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

leave

Function Frame

Functions would like to use the stack to allocate space for their local variables. Can we use the stack pointer (*esp/rsp*) 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 x86/64, frame pointer is called the *base pointer*, and is stored in ebp/rbp

Function Frame

A function's Stack Frame

- Starts with where ebp/rbp points to
- Ends with where esp/rsp 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

Calling Convention

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 ebp, esp)
- Creates space on stack for local variables (sub esp, #imm)
- 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 ebp, esp; (change the base pointer to the stack)

sub esp, 10; (allocating a local stack space)

Callee Deallocate a stack (Function epilogue)

mov esp, ebp

pop ebp

ret

Global and Local Variables (misc/globallocalv)

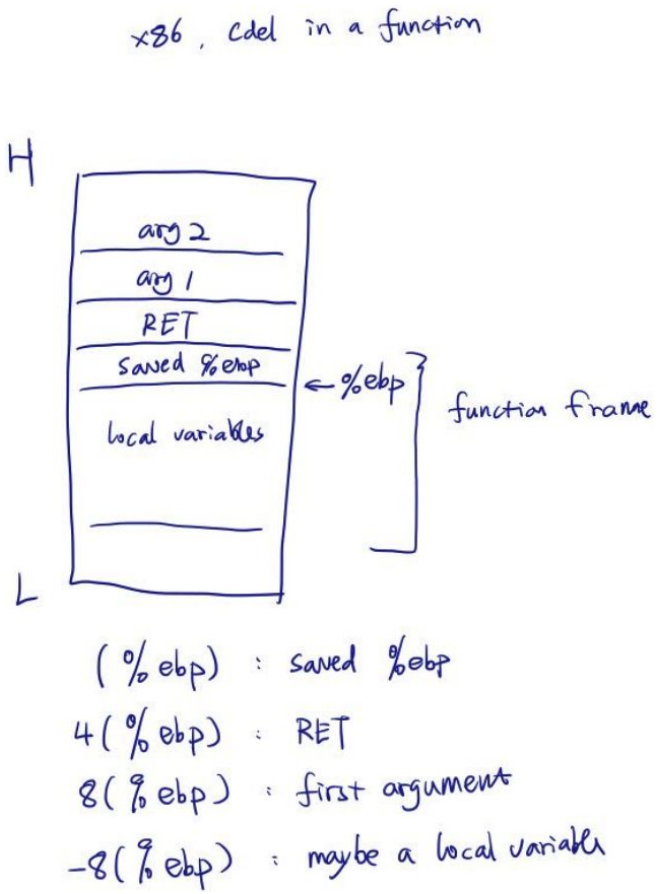
```
int func(int p)
{
        int l_i = 10;
        int l_u;

        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;
}
```

main(): function call

```
128c:    83 c4 10          add   esp,0x10
128f:    83 ec 0c          sub   esp,0xc
1292:    6a 0a             push  0xa
1294:    e8 34 ff ff ff    call  11cd <func>
```

```
000011cd <func>:
   11cd:    f3 0f 1e fb         endbr32
   11d1:    55                  push  ebp
   11d2:    89 e5               mov   ebp,esp
   11d4:    83 ec 18            sub   esp,0x18
   11d7:    c7 45 f4 0a 00 00 00   mov   DWORD PTR [ebp-0xc],0xa
   11de:    83 ec 08            sub   esp,0x8
   11e1:    8d 45 f4            lea   eax,[ebp-0xc]
   11e4:    50                  push  eax
   11e5:    68 08 20 00 00      push  0x2008
   11ea:    e8 fc ff ff ff      call  11eb <func+0x1e>
   11ef:    83 c4 10            add   esp,0x10
   11f2:    83 ec 08            sub   esp,0x8
   11f5:    8d 45 f0            lea   eax,[ebp-0x10]
   11f8:    50                  push  eax
   11f9:    68 20 20 00 00      push  0x2020
   11fe:    e8 fc ff ff ff      call  11ff <func+0x32>
   1203:    83 c4 10            add   esp,0x10
   1206:    83 ec 08            sub   esp,0x8
   1209:    8d 45 08            lea   eax,[ebp+0x8]
   120c:    50                  push  eax
   120d:    68 38 20 00 00      push  0x2038
   1212:    e8 fc ff ff ff      call  1213 <func+0x46>
   1217:    83 c4 10            add   esp,0x10
   121a:    b8 00 00 00 00      mov   eax,0x0
   121f:    c9                  leave
   1220:    c3                  ret
```

Draw the stack (x86 cdecl)



x86 Stack Usage (32bit)

- Negative indexing over ebp

  mov eax, [ebp - 0x8]

  lea eax, [ebp - 24]

- Positive indexing over ebp

  mov eax, [ebp + 8]

  mov eax, [ebp + 0xc]

- Positive indexing over esp

x86 Stack Usage (32bit)

- Accesses local variables (negative indexing over ebp)

  mov eax, [ebp - 0x8] - value at ebp - 0x8

  lea eax, [ebp - 24] - address as ebp - 0x24

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

  mov eax, [ebp + 8] - 1st arg

  mov eax, [ebp + 0xc] - 2nd arg

- Positive indexing over esp

  Function arguments to callee

Stack example: misc/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: misc/fiveParameters_32

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

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

X86 disassembly

globallocalv_fast_32

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.

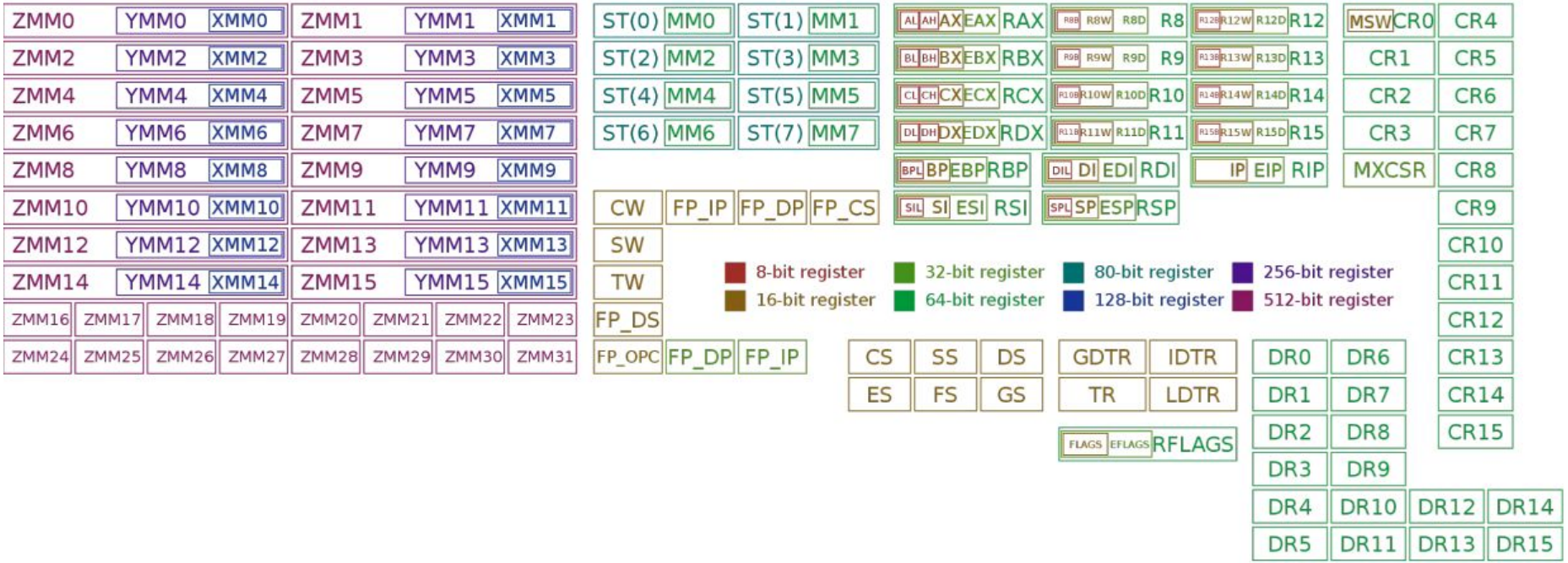
```
int __attribute__ ((fastcall)) func(int p)
```

x86-64 (64 bit) 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: misc/fiveParameters_64

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

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

X86 disassembly

x86 Stack Usage

- Access local variables (negative indexing over rbp)

  mov rax, [rbp-8]

  lea rax, [rbp-0x24]

- Access function arguments from caller

  mov rax, rdi

- Setup parameters for callee

  mov rdi, rax

Overwrite Local Variables

Data-only Attack

Buffer Overflow Example: overflowlocal1

```
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]);
}
```

```
000012c4 <vulfoo>:
   12c4:    55              push  ebp
   12c5:    89 e5             mov   ebp,esp
   12c7:    83 ec 18          sub   esp,0x18
   12ca:    8b 45 08          mov   eax,DWORD PTR [ebp+0x8]
   12cd:    89 45 f4          mov   DWORD PTR [ebp-0xc],eax
   12d0:    83 ec 08          sub   esp,0x8
   12d3:    ff 75 0c          push  DWORD PTR [ebp+0xc]
   12d6:    8d 45 ee          lea   eax,[ebp-0x12]
   12d9:    50              push  eax
   12da:    e8 fc ff ff ff       call  12db <vulfoo+0x17>
   12df:    83 c4 10          add   esp,0x10
   12e2:    83 7d f4 00          cmp   DWORD PTR [ebp-0xc],0x0
   12e6:    74 07             je    12ef <vulfoo+0x2b>
   12e8:    e8 10 ff ff ff       call  11fd <print_flag>
   12ed:    eb 10             jmp   12ff <vulfoo+0x3b>
   12ef:    83 ec 0c          sub   esp,0xc
   12f2:    68 45 20 00 00       push  0x2045
   12f7:    e8 fc ff ff ff       call  12f8 <vulfoo+0x34>
   12fc:    83 c4 10          add   esp,0x10
   12ff:    b8 00 00 00 00       mov   eax,0x0
   1304:    c9              leave
   1305:    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;
}
```

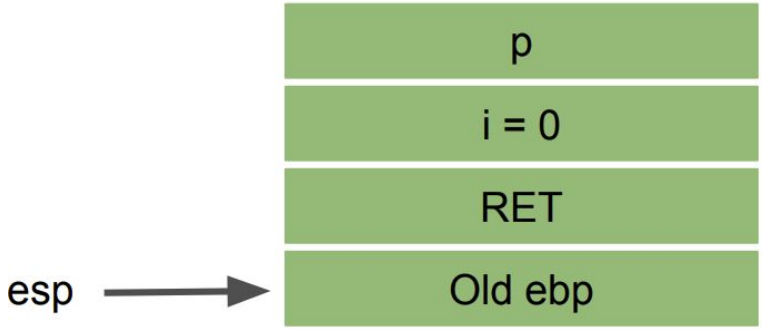
```
char *strcpy(char *dest, const char *src)
{
        char *save = dest;
        while(*dest++ = *src++);
        return save;
}
```

Buffer Overflow Example: overflowlocal1_32

```
000012c4 <vulfoo>:
   12c4:    55              push  ebp
   12c5:    89 e5             mov   ebp,esp
   12c7:    83 ec 18          sub   esp,0x18
   12ca:    8b 45 08          mov   eax,DWORD PTR [ebp+0x8]
   12cd:    89 45 f4          mov   DWORD PTR [ebp-0xc],eax
   12d0:    83 ec 08          sub   esp,0x8
   12d3:    ff 75 0c          push  DWORD PTR [ebp+0xc]
   12d6:    8d 45 ee          lea   eax,[ebp-0x12]
   12d9:    50              push  eax
   12da:    e8 fc ff ff ff      call  12db <vulfoo+0x17>
   12df:    83 c4 10          add   esp,0x10
   12e2:    83 7d f4 00         cmp   DWORD PTR [ebp-0xc],0x0
   12e6:    74 07             je    12ef <vulfoo+0x2b>
   12e8:    e8 10 ff ff ff      call  11fd <print_flag>
   12ed:    eb 10             jmp   12ff <vulfoo+0x3b>
   12ef:    83 ec 0c          sub   esp,0xc
   12f2:    68 45 20 00 00      push  0x2045
   12f7:    e8 fc ff ff ff      call  12f8 <vulfoo+0x34>
   12fc:    83 c4 10          add   esp,0x10
   12ff:    b8 00 00 00 00      mov   eax,0x0
   1304:    c9              leave
   1305:    c3              ret
```

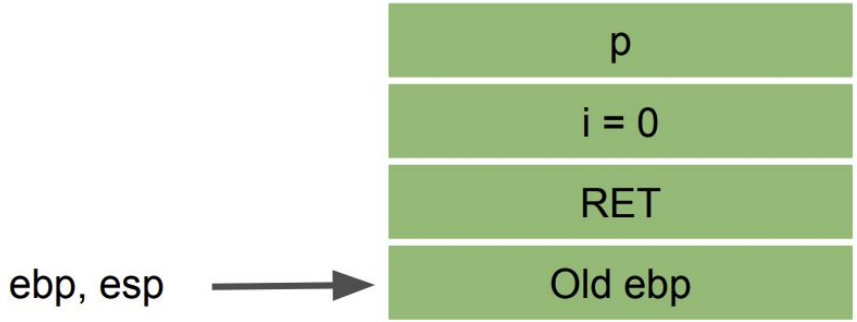
Buffer Overflow Example: overflowlocal1_32

```
000012c4 <vulfoo>:
   12c4:    55              push   ebp
   12c5:    89 e5           mov    ebp,esp
   12c7:    83 ec 18        sub    esp,0x18
   12ca:    8b 45 08        mov    eax,DWORD PTR [ebp+0x8]
   12cd:    89 45 f4        mov    DWORD PTR [ebp-0xc],eax
   12d0:    83 ec 08        sub    esp,0x8
   12d3:    ff 75 0c        push   DWORD PTR [ebp+0xc]
   12d6:    8d 45 ee        lea    eax,[ebp-0x12]
   12d9:    50              push   eax
   12da:    e8 fc ff ff ff  call   12db <vulfoo+0x17>
   12df:    83 c4 10        add    esp,0x10
   12e2:    83 7d f4 00     cmp    DWORD PTR [ebp-0xc],0x0
   12e6:    74 07           je     12ef <vulfoo+0x2b>
   12e8:    e8 10 ff ff ff  call   11fd <print_flag>
   12ed:    eb 10           jmp    12ff <vulfoo+0x3b>
   12ef:    83 ec 0c        sub    esp,0xc
   12f2:    68 45 20 00 00  push   0x2045
   12f7:    e8 fc ff ff ff  call   12f8 <vulfoo+0x34>
   12fc:    83 c4 10        add    esp,0x10
   12ff:    b8 00 00 00 00  mov    eax,0x0
   1304:    c9              leave
   1305:    c3              ret
```



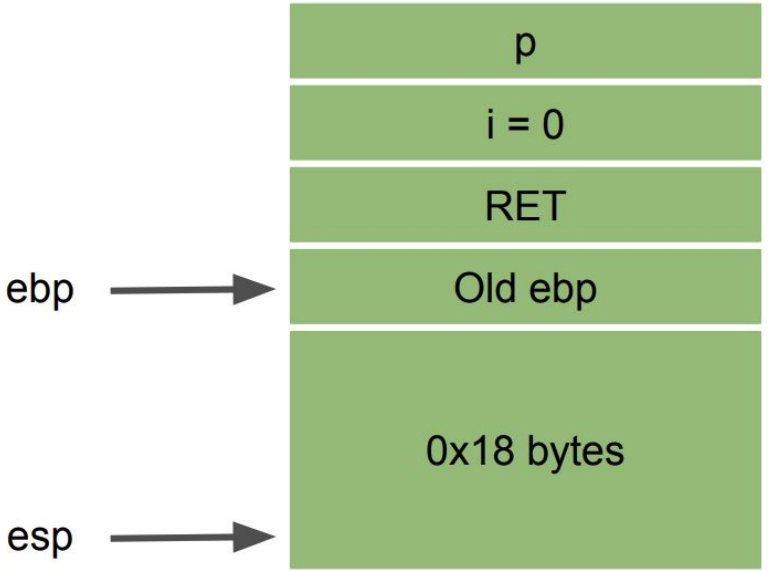
Buffer Overflow Example: overflowlocal1_32

```
000012c4 <vulfoo>:
   12c4:    55              push   ebp
   12c5:    89 e5           mov    ebp,esp
   12c7:    83 ec 18        sub    esp,0x18
   12ca:    8b 45 08        mov    eax,DWORD PTR [ebp+0x8]
   12cd:    89 45 f4        mov    DWORD PTR [ebp-0xc],eax
   12d0:    83 ec 08        sub    esp,0x8
   12d3:    ff 75 0c        push   DWORD PTR [ebp+0xc]
   12d6:    8d 45 ee        lea    eax,[ebp-0x12]
   12d9:    50              push   eax
   12da:    e8 fc ff ff ff  call   12db <vulfoo+0x17>
   12df:    83 c4 10        add    esp,0x10
   12e2:    83 7d f4 00     cmp    DWORD PTR [ebp-0xc],0x0
   12e6:    74 07           je     12ef <vulfoo+0x2b>
   12e8:    e8 10 ff ff ff  call   11fd <print_flag>
   12ed:    eb 10           jmp    12ff <vulfoo+0x3b>
   12ef:    83 ec 0c        sub    esp,0xc
   12f2:    68 45 20 00 00  push   0x2045
   12f7:    e8 fc ff ff ff  call   12f8 <vulfoo+0x34>
   12fc:    83 c4 10        add    esp,0x10
   12ff:    b8 00 00 00 00  mov    eax,0x0
   1304:    c9              leave
   1305:    c3              ret
```



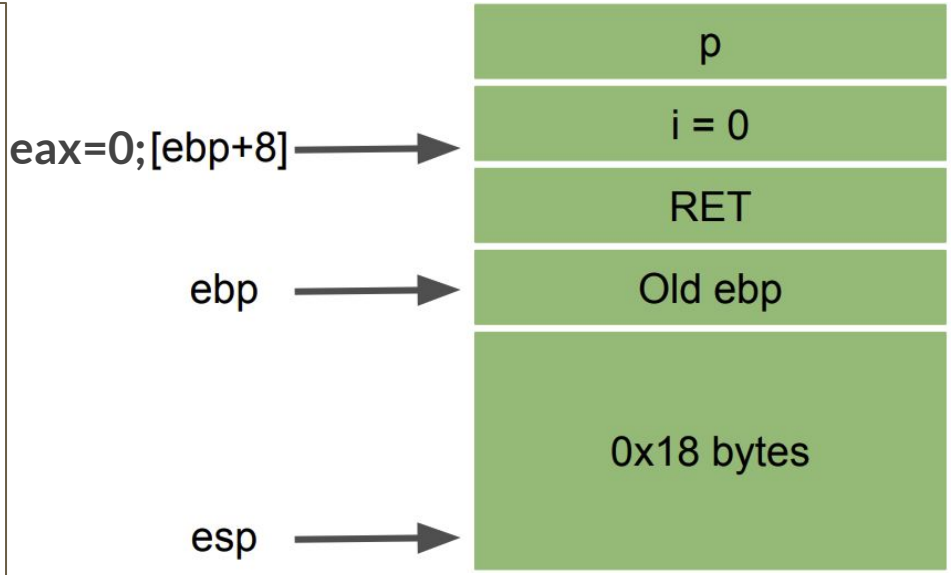
Buffer Overflow Example: overflowlocal1_32

```
000012c4 <vulfoo>:
   12c4:    55              push  ebp
   12c5:    89 e5             mov   ebp,esp
   12c7:    83 ec 18          sub   esp,0x18
   12ca:    8b 45 08          mov   eax,DWORD PTR [ebp+0x8]
   12cd:    89 45 f4          mov   DWORD PTR [ebp-0xc],eax
   12d0:    83 ec 08          sub   esp,0x8
   12d3:    ff 75 0c          push  DWORD PTR [ebp+0xc]
   12d6:    8d 45 ee          lea   eax,[ebp-0x12]
   12d9:    50              push  eax
   12da:    e8 fc ff ff ff      call  12db <vulfoo+0x17>
   12df:    83 c4 10          add   esp,0x10
   12e2:    83 7d f4 00         cmp   DWORD PTR [ebp-0xc],0x0
   12e6:    74 07             je    12ef <vulfoo+0x2b>
   12e8:    e8 10 ff ff ff      call  11fd <print_flag>
   12ed:    eb 10             jmp   12ff <vulfoo+0x3b>
   12ef:    83 ec 0c          sub   esp,0xc
   12f2:    68 45 20 00 00      push  0x2045
   12f7:    e8 fc ff ff ff      call  12f8 <vulfoo+0x34>
   12fc:    83 c4 10          add   esp,0x10
   12ff:    b8 00 00 00 00      mov   eax,0x0
   1304:    c9              leave
   1305:    c3              ret
```



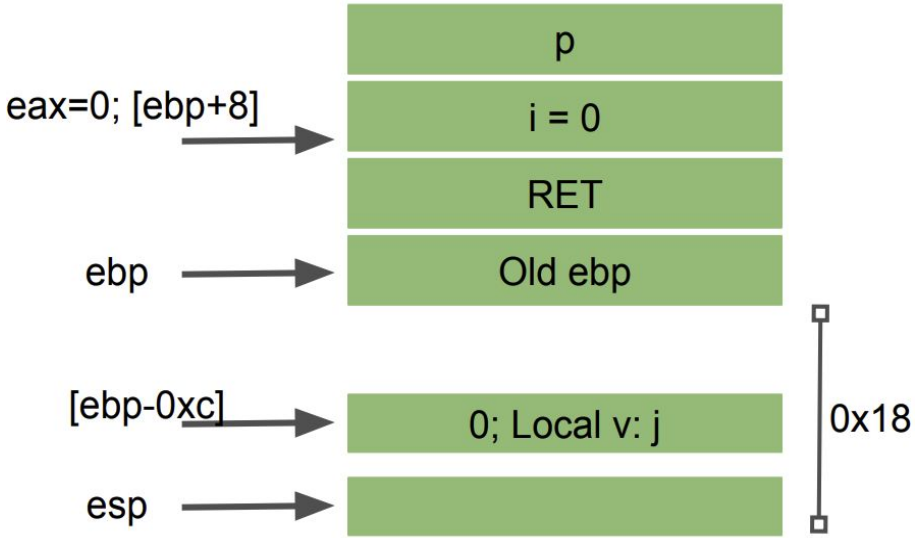
Buffer Overflow Example: overflowlocal1_32

```
000012c4 <vulfoo>:
   12c4:    55              push  ebp
   12c5:    89 e5             mov   ebp,esp
   12c7:    83 ec 18          sub   esp,0x18
   12ca:    8b 45 08          mov   eax,DWORD PTR [ebp+0x8]
   12cd:    89 45 f4          mov   DWORD PTR [ebp-0xc],eax
   12d0:    83 ec 08          sub   esp,0x8
   12d3:    ff 75 0c          push  DWORD PTR [ebp+0xc]
   12d6:    8d 45 ee          lea   eax,[ebp-0x12]
   12d9:    50              push  eax
   12da:    e8 fc ff ff ff       call  12db <vulfoo+0x17>
   12df:    83 c4 10          add   esp,0x10
   12e2:    83 7d f4 00          cmp   DWORD PTR [ebp-0xc],0x0
   12e6:    74 07             je    12ef <vulfoo+0x2b>
   12e8:    e8 10 ff ff ff       call  11fd <print_flag>
   12ed:    eb 10             jmp   12ff <vulfoo+0x3b>
   12ef:    83 ec 0c          sub   esp,0xc
   12f2:    68 45 20 00 00       push  0x2045
   12f7:    e8 fc ff ff ff       call  12f8 <vulfoo+0x34>
   12fc:    83 c4 10          add   esp,0x10
   12ff:    b8 00 00 00 00       mov   eax,0x0
   1304:    c9              leave
   1305:    c3              ret
```



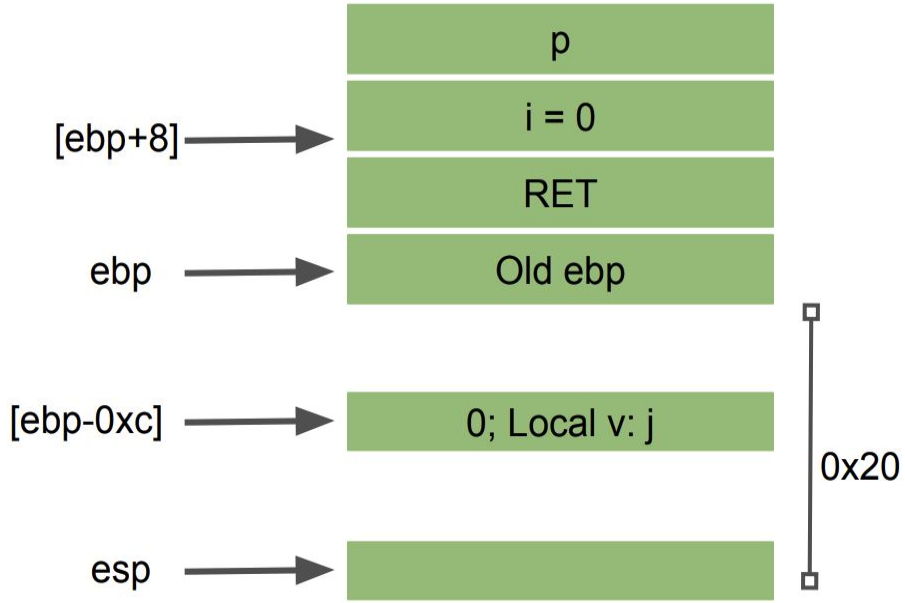
Buffer Overflow Example: overflowlocal1_32

```
000012c4 <vulfoo>:
   12c4:    55              push  ebp
   12c5:    89 e5             mov   ebp,esp
   12c7:    83 ec 18          sub   esp,0x18
   12ca:    8b 45 08          mov   eax,DWORD PTR [ebp+0x8]
   12cd:    89 45 f4          mov   DWORD PTR [ebp-0xc],eax
   12d0:    83 ec 08          sub   esp,0x8
   12d3:    ff 75 0c          push  DWORD PTR [ebp+0xc]
   12d6:    8d 45 ee          lea   eax,[ebp-0x12]
   12d9:    50              push  eax
   12da:    e8 fc ff ff ff       call  12db <vulfoo+0x17>
   12df:    83 c4 10          add   esp,0x10
   12e2:    83 7d f4 00          cmp   DWORD PTR [ebp-0xc],0x0
   12e6:    74 07           je    12ef <vulfoo+0x2b>
   12e8:    e8 10 ff ff ff       call  11fd <print_flag>
   12ed:    eb 10           jmp   12ff <vulfoo+0x3b>
   12ef:    83 ec 0c          sub   esp,0xc
   12f2:    68 45 20 00 00       push  0x2045
   12f7:    e8 fc ff ff ff       call  12f8 <vulfoo+0x34>
   12fc:    83 c4 10          add   esp,0x10
   12ff:    b8 00 00 00 00       mov   eax,0x0
   1304:    c9              leave
   1305:    c3              ret
```



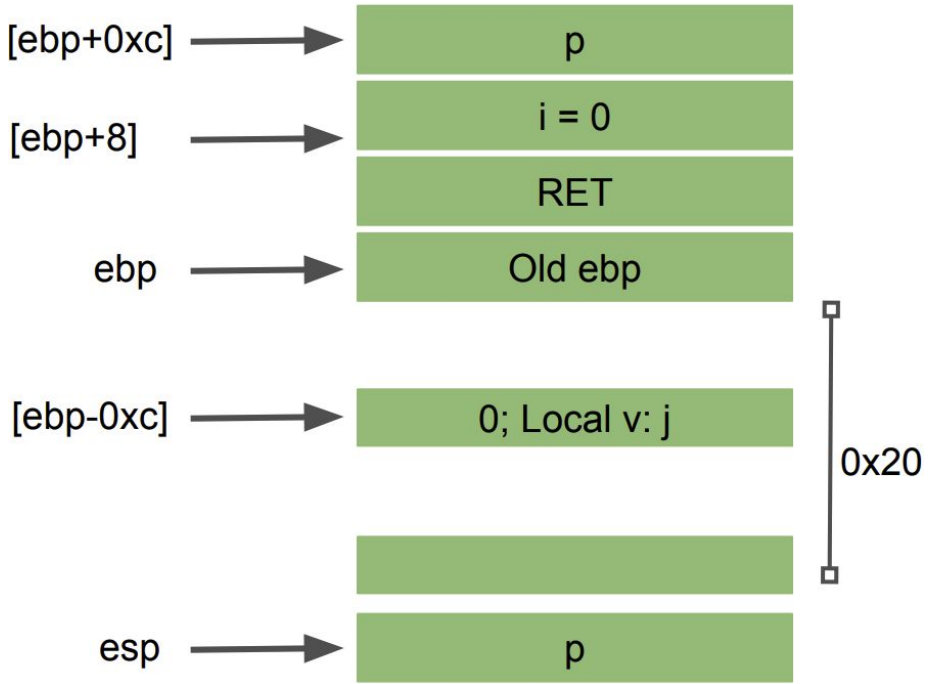
Buffer Overflow Example: overflowlocal1_32

```
000012c4 <vulfoo>:
   12c4:    55              push  ebp
   12c5:    89 e5             mov   ebp,esp
   12c7:    83 ec 18          sub   esp,0x18
   12ca:    8b 45 08          mov   eax,DWORD PTR [ebp+0x8]
   12cd:    89 45 f4          mov   DWORD PTR [ebp-0xc],eax
   12d0:    83 ec 08          sub   esp,0x8
   12d3:    ff 75 0c          push  DWORD PTR [ebp+0xc]
   12d6:    8d 45 ee          lea   eax,[ebp-0x12]
   12d9:    50              push  eax
   12da:    e8 fc ff ff ff      call  12db <vulfoo+0x17>
   12df:    83 c4 10          add   esp,0x10
   12e2:    83 7d f4 00         cmp   DWORD PTR [ebp-0xc],0x0
   12e6:    74 07           je    12ef <vulfoo+0x2b>
   12e8:    e8 10 ff ff ff      call  11fd <print_flag>
   12ed:    eb 10           jmp   12ff <vulfoo+0x3b>
   12ef:    83 ec 0c          sub   esp,0xc
   12f2:    68 45 20 00 00      push  0x2045
   12f7:    e8 fc ff ff ff      call  12f8 <vulfoo+0x34>
   12fc:    83 c4 10          add   esp,0x10
   12ff:    b8 00 00 00 00      mov   eax,0x0
   1304:    c9              leave
   1305:    c3              ret
```



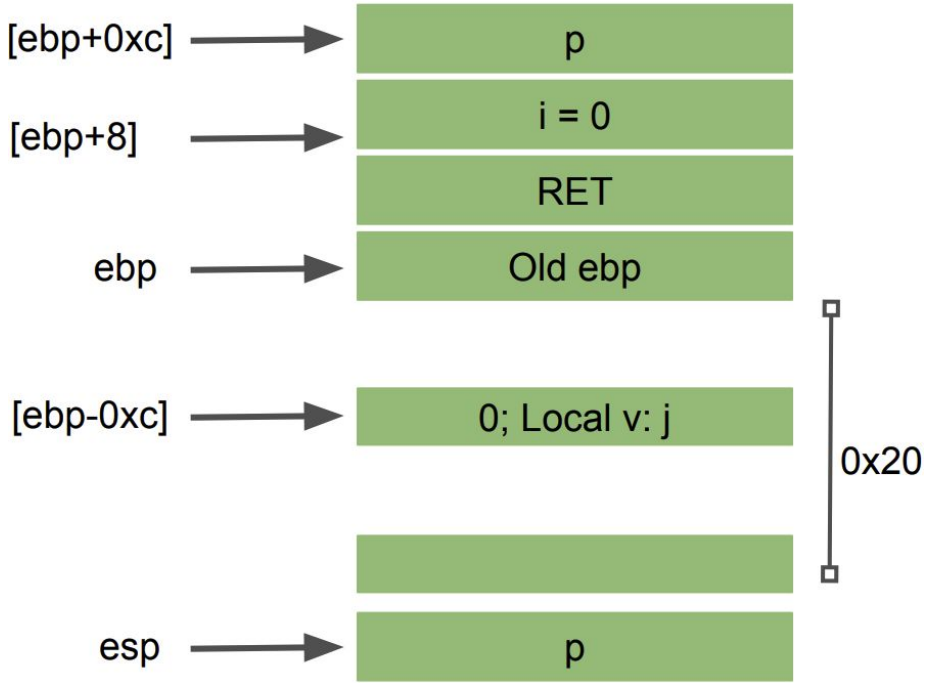
Buffer Overflow Example: overflowlocal1_32

```
000012c4 <vulfoo>:
   12c4:    55              push  ebp
   12c5:    89 e5             mov   ebp,esp
   12c7:    83 ec 18          sub   esp,0x18
   12ca:    8b 45 08          mov   eax,DWORD PTR [ebp+0x8]
   12cd:    89 45 f4          mov   DWORD PTR [ebp-0xc],eax
   12d0:    83 ec 08          sub   esp,0x8
   12d3:    ff 75 0c          push  DWORD PTR [ebp+0xc]
   12d6:    8d 45 ee          lea   eax,[ebp-0x12]
   12d9:    50              push  eax
   12da:    e8 fc ff ff ff      call  12db <vulfoo+0x17>
   12df:    83 c4 10          add   esp,0x10
   12e2:    83 7d f4 00         cmp   DWORD PTR [ebp-0xc],0x0
   12e6:    74 07             je    12ef <vulfoo+0x2b>
   12e8:    e8 10 ff ff ff      call  11fd <print_flag>
   12ed:    eb 10             jmp   12ff <vulfoo+0x3b>
   12ef:    83 ec 0c          sub   esp,0xc
   12f2:    68 45 20 00 00      push  0x2045
   12f7:    e8 fc ff ff ff      call  12f8 <vulfoo+0x34>
   12fc:    83 c4 10          add   esp,0x10
   12ff:    b8 00 00 00 00      mov   eax,0x0
   1304:    c9              leave
   1305:    c3              ret
```



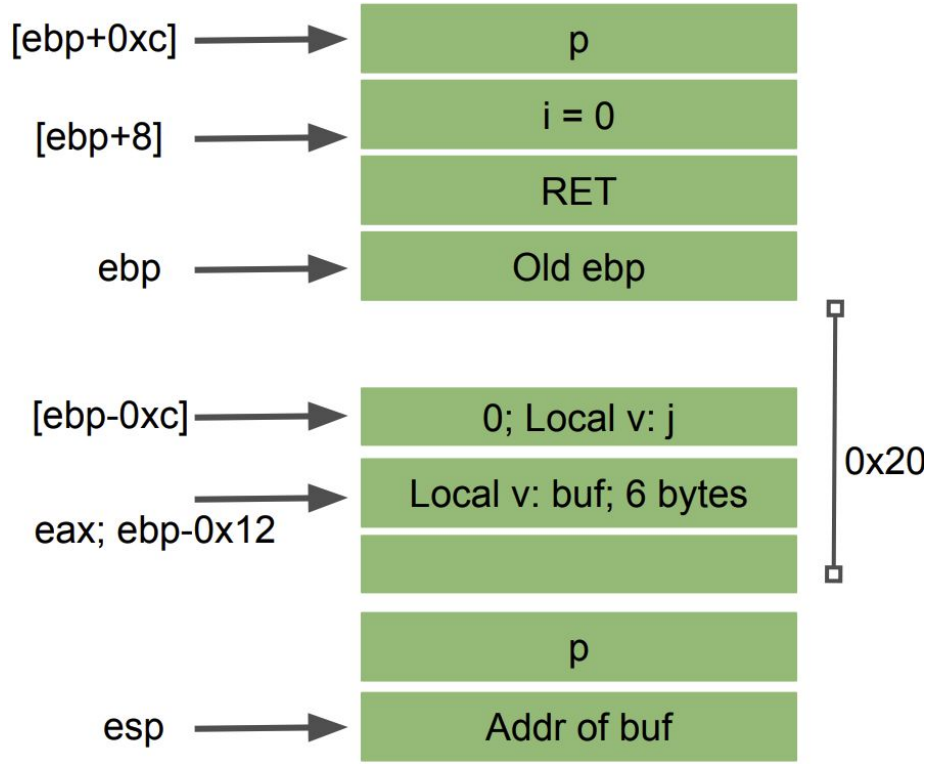
Buffer Overflow Example: overflowlocal1_32

```
000012c4 <vulfoo>:
   12c4:    55              push  ebp
   12c5:    89 e5             mov   ebp,esp
   12c7:    83 ec 18          sub   esp,0x18
   12ca:    8b 45 08          mov   eax,DWORD PTR [ebp+0x8]
   12cd:    89 45 f4          mov   DWORD PTR [ebp-0xc],eax
   12d0:    83 ec 08          sub   esp,0x8
   12d3:    ff 75 0c          push  DWORD PTR [ebp+0xc]
   12d6:    8d 45 ee          lea   eax,[ebp-0x12]
   12d9:    50              push  eax
   12da:    e8 fc ff ff ff      call  12db <vulfoo+0x17>
   12df:    83 c4 10          add   esp,0x10
   12e2:    83 7d f4 00         cmp   DWORD PTR [ebp-0xc],0x0
   12e6:    74 07           je    12ef <vulfoo+0x2b>
   12e8:    e8 10 ff ff ff      call  11fd <print_flag>
   12ed:    eb 10           jmp   12ff <vulfoo+0x3b>
   12ef:    83 ec 0c          sub   esp,0xc
   12f2:    68 45 20 00 00      push  0x2045
   12f7:    e8 fc ff ff ff      call  12f8 <vulfoo+0x34>
   12fc:    83 c4 10          add   esp,0x10
   12ff:    b8 00 00 00 00      mov   eax,0x0
   1304:    c9              leave
   1305:    c3              ret
```



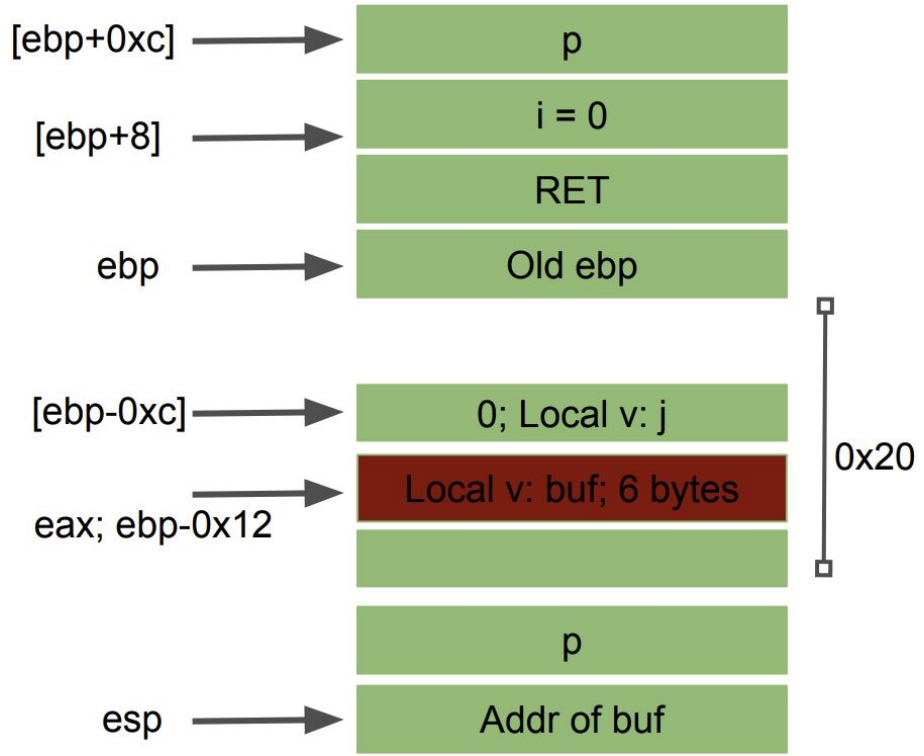
Buffer Overflow Example: overflowlocal1_32

```
000012c4 <vulfoo>:
   12c4:    55              push  ebp
   12c5:    89 e5             mov   ebp,esp
   12c7:    83 ec 18          sub   esp,0x18
   12ca:    8b 45 08          mov   eax,DWORD PTR [ebp+0x8]
   12cd:    89 45 f4          mov   DWORD PTR [ebp-0xc],eax
   12d0:    83 ec 08          sub   esp,0x8
   12d3:    ff 75 0c          push  DWORD PTR [ebp+0xc]
   12d6:    8d 45 ee          lea   eax,[ebp-0x12]
   12d9:    50              push  eax
   12da:    e8 fc ff ff ff       call  12db <vulfoo+0x17>
   12df:    83 c4 10          add   esp,0x10
   12e2:    83 7d f4 00          cmp   DWORD PTR [ebp-0xc],0x0
   12e6:    74 07           je    12ef <vulfoo+0x2b>
   12e8:    e8 10 ff ff ff       call  11fd <print_flag>
   12ed:    eb 10           jmp   12ff <vulfoo+0x3b>
   12ef:    83 ec 0c          sub   esp,0xc
   12f2:    68 45 20 00 00       push  0x2045
   12f7:    e8 fc ff ff ff       call  12f8 <vulfoo+0x34>
   12fc:    83 c4 10          add   esp,0x10
   12ff:    b8 00 00 00 00       mov   eax,0x0
   1304:    c9              leave
   1305:    c3              ret
```



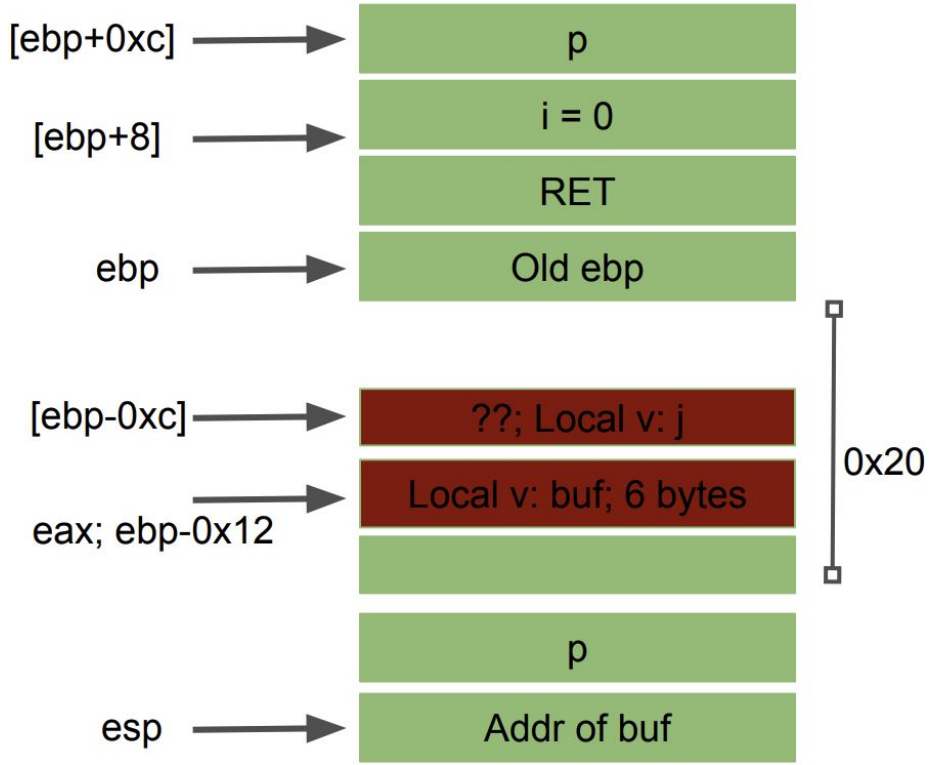
Buffer Overflow Example: overflowlocal1_32

```
000012c4 <vulfoo>:
   12c4:    55              push  ebp
   12c5:    89 e5             mov   ebp,esp
   12c7:    83 ec 18          sub   esp,0x18
   12ca:    8b 45 08          mov   eax,DWORD PTR [ebp+0x8]
   12cd:    89 45 f4          mov   DWORD PTR [ebp-0xc],eax
   12d0:    83 ec 08          sub   esp,0x8
   12d3:    ff 75 0c          push  DWORD PTR [ebp+0xc]
   12d6:    8d 45 ee          lea   eax,[ebp-0x12]
   12d9:    50              push  eax
   12da:    e8 fc ff ff ff       call  12db <vulfoo+0x17>
   12df:    83 c4 10          add   esp,0x10
   12e2:    83 7d f4 00          cmp   DWORD PTR [ebp-0xc],0x0
   12e6:    74 07           je    12ef <vulfoo+0x2b>
   12e8:    e8 10 ff ff ff       call  11fd <print_flag>
   12ed:    eb 10           jmp   12ff <vulfoo+0x3b>
   12ef:    83 ec 0c          sub   esp,0xc
   12f2:    68 45 20 00 00       push  0x2045
   12f7:    e8 fc ff ff ff       call  12f8 <vulfoo+0x34>
   12fc:    83 c4 10          add   esp,0x10
   12ff:    b8 00 00 00 00       mov   eax,0x0
   1304:    c9              leave
   1305:    c3              ret
```



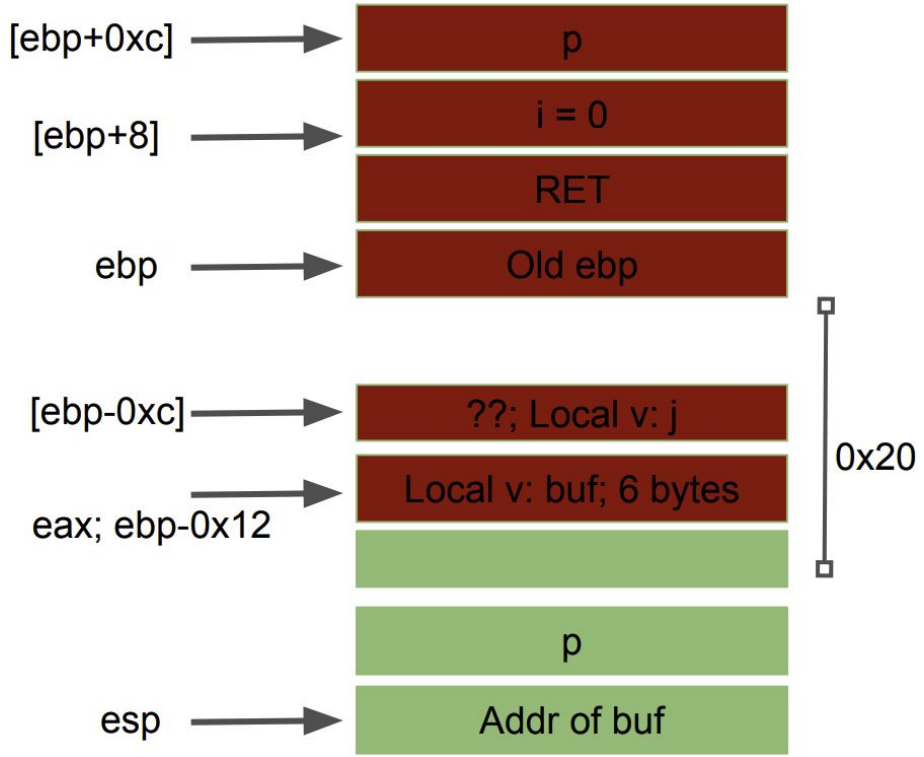
Buffer Overflow Example: overflowlocal1_32

```
000012c4 <vulfoo>:
   12c4:    55              push  ebp
   12c5:    89 e5             mov   ebp,esp
   12c7:    83 ec 18          sub   esp,0x18
   12ca:    8b 45 08          mov   eax,DWORD PTR [ebp+0x8]
   12cd:    89 45 f4          mov   DWORD PTR [ebp-0xc],eax
   12d0:    83 ec 08          sub   esp,0x8
   12d3:    ff 75 0c          push  DWORD PTR [ebp+0xc]
   12d6:    8d 45 ee          lea   eax,[ebp-0x12]
   12d9:    50              push  eax
   12da:    e8 fc ff ff ff       call  12db <vulfoo+0x17>
   12df:    83 c4 10          add   esp,0x10
   12e2:    83 7d f4 00          cmp   DWORD PTR [ebp-0xc],0x0
   12e6:    74 07           je    12ef <vulfoo+0x2b>
   12e8:    e8 10 ff ff ff       call  11fd <print_flag>
   12ed:    eb 10           jmp   12ff <vulfoo+0x3b>
   12ef:    83 ec 0c          sub   esp,0xc
   12f2:    68 45 20 00 00       push  0x2045
   12f7:    e8 fc ff ff ff       call  12f8 <vulfoo+0x34>
   12fc:    83 c4 10          add   esp,0x10
   12ff:    b8 00 00 00 00       mov   eax,0x0
   1304:    c9              leave
   1305:    c3              ret
```



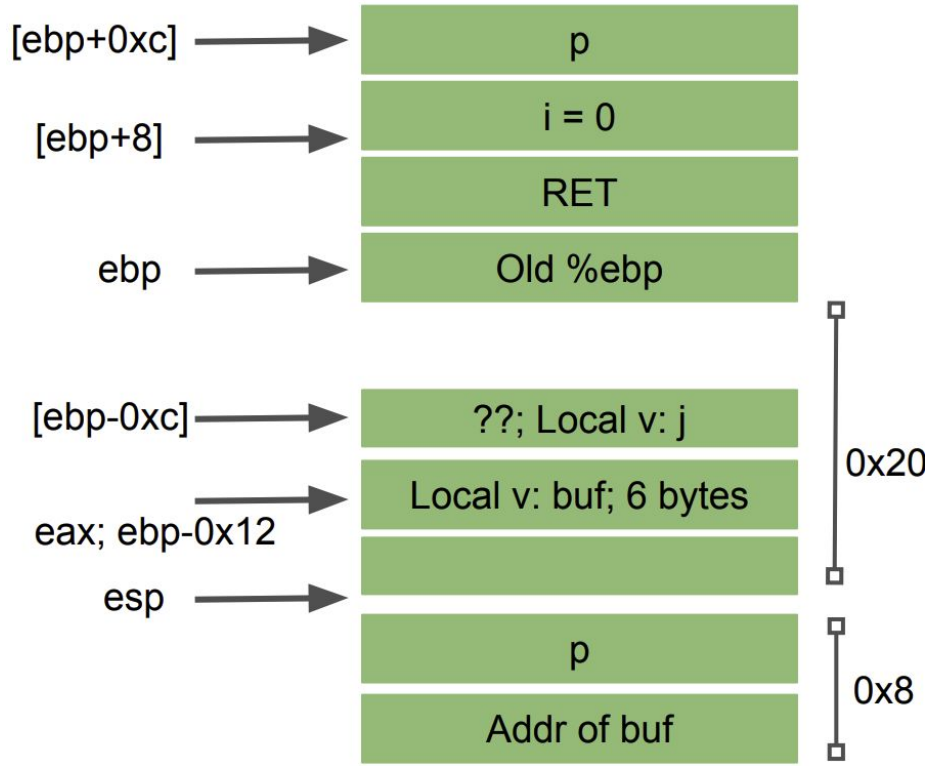
Buffer Overflow Example: overflowlocal1_32

```
000012c4 <vulfoo>:
   12c4:    55              push  ebp
   12c5:    89 e5             mov   ebp,esp
   12c7:    83 ec 18          sub   esp,0x18
   12ca:    8b 45 08          mov   eax,DWORD PTR [ebp+0x8]
   12cd:    89 45 f4          mov   DWORD PTR [ebp-0xc],eax
   12d0:    83 ec 08          sub   esp,0x8
   12d3:    ff 75 0c          push  DWORD PTR [ebp+0xc]
   12d6:    8d 45 ee          lea   eax,[ebp-0x12]
   12d9:    50              push  eax
   12da:    e8 fc ff ff ff       call  12db <vulfoo+0x17>
   12df:    83 c4 10          add   esp,0x10
   12e2:    83 7d f4 00          cmp   DWORD PTR [ebp-0xc],0x0
   12e6:    74 07           je    12ef <vulfoo+0x2b>
   12e8:    e8 10 ff ff ff       call  11fd <print_flag>
   12ed:    eb 10           jmp   12ff <vulfoo+0x3b>
   12ef:    83 ec 0c          sub   esp,0xc
   12f2:    68 45 20 00 00       push  0x2045
   12f7:    e8 fc ff ff ff       call  12f8 <vulfoo+0x34>
   12fc:    83 c4 10          add   esp,0x10
   12ff:    b8 00 00 00 00       mov   eax,0x0
   1304:    c9              leave
   1305:    c3              ret
```



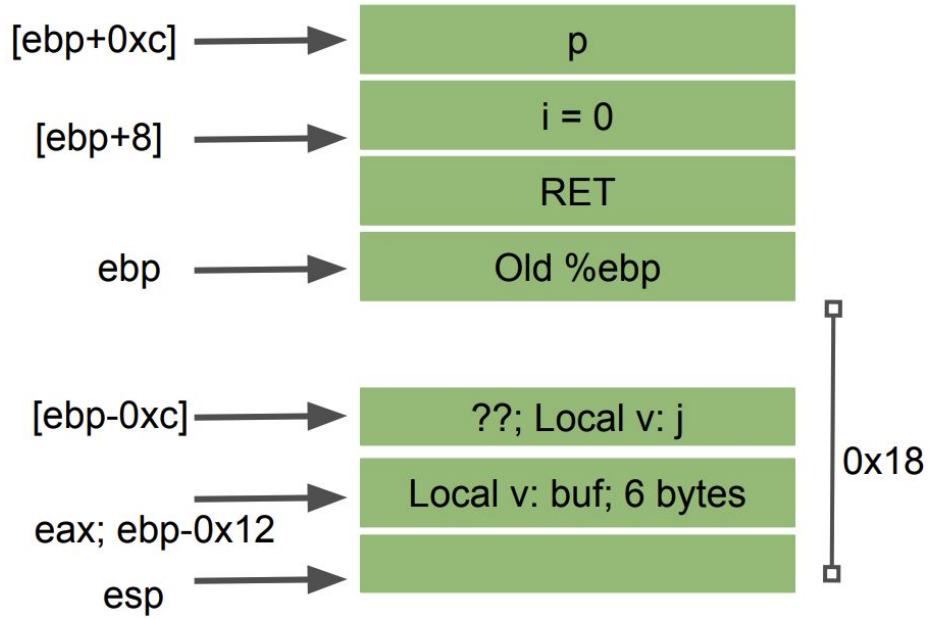
Buffer Overflow Example: overflowlocal1_32

```
000012c4 <vulfoo>:
   12c4:    55              push  ebp
   12c5:    89 e5             mov   ebp,esp
   12c7:    83 ec 18          sub   esp,0x18
   12ca:    8b 45 08          mov   eax,DWORD PTR [ebp+0x8]
   12cd:    89 45 f4          mov   DWORD PTR [ebp-0xc],eax
   12d0:    83 ec 08          sub   esp,0x8
   12d3:    ff 75 0c          push  DWORD PTR [ebp+0xc]
   12d6:    8d 45 ee          lea   eax,[ebp-0x12]
   12d9:    50              push  eax
   12da:    e8 fc ff ff ff       call  12db <vulfoo+0x17>
   12df:    83 c4 10          add   esp,0x10
   12e2:    83 7d f4 00          cmp   DWORD PTR [ebp-0xc],0x0
   12e6:    74 07           je    12ef <vulfoo+0x2b>
   12e8:    e8 10 ff ff ff       call  11fd <print_flag>
   12ed:    eb 10           jmp   12ff <vulfoo+0x3b>
   12ef:    83 ec 0c          sub   esp,0xc
   12f2:    68 45 20 00 00       push  0x2045
   12f7:    e8 fc ff ff ff       call  12f8 <vulfoo+0x34>
   12fc:    83 c4 10          add   esp,0x10
   12ff:    b8 00 00 00 00       mov   eax,0x0
   1304:    c9              leave
   1305:    c3              ret
```



Buffer Overflow Example: overflowlocal1_32

```
000012c4 <vulfoo>:
   12c4:    55              push  ebp
   12c5:    89 e5             mov   ebp,esp
   12c7:    83 ec 18          sub   esp,0x18
   12ca:    8b 45 08          mov   eax,DWORD PTR [ebp+0x8]
   12cd:    89 45 f4          mov   DWORD PTR [ebp-0xc],eax
   12d0:    83 ec 08          sub   esp,0x8
   12d3:    ff 75 0c          push  DWORD PTR [ebp+0xc]
   12d6:    8d 45 ee          lea   eax,[ebp-0x12]
   12d9:    50              push  eax
   12da:    e8 fc ff ff ff       call  12db <vulfoo+0x17>
   12df:    83 c4 10          add   esp,0x10
   12e2:    83 7d f4 00          cmp   DWORD PTR [ebp-0xc],0x0
   12e6:    74 07           je    12ef <vulfoo+0x2b>
   12e8:    e8 10 ff ff ff       call  11fd <print_flag>
   12ed:    eb 10           jmp   12ff <vulfoo+0x3b>
   12ef:    83 ec 0c          sub   esp,0xc
   12f2:    68 45 20 00 00       push  0x2045
   12f7:    e8 fc ff ff ff       call  12f8 <vulfoo+0x34>
   12fc:    83 c4 10          add   esp,0x10
   12ff:    b8 00 00 00 00       mov   eax,0x0
   1304:    c9              leave
   1305:    c3              ret
```



Buffer Overflow Example: overflowlocal1_64

```
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]);
}
```

```
000000000000125e <vulfoo>:
   125e:    55              push   rbp
   125f:    48 89 e5        mov    rbp,rsp
   1262:    48 83 ec 20     sub    rsp,0x20
   1266:    89 7d ec        mov    DWORD PTR [rbp-0x14],edi
   1269:    48 89 75 e0     mov    QWORD PTR [rbp-0x20],rsi
   126d:    8b 45 ec        mov    eax,DWORD PTR [rbp-0x14]
   1270:    89 45 fc        mov    DWORD PTR [rbp-0x4],eax
   1273:    48 8b 55 e0     mov    rdx,QWORD PTR [rbp-0x20]
   1277:    48 8d 45 f6     lea    rax,[rbp-0xa]
   127b:    48 89 d6        mov    rsi,rdx
   127e:    48 89 c7        mov    rdi,rax
   1281:    e8 aa fd ff ff  call   1030 <strcpy@plt>
   1286:    83 7d fc 00     cmp    DWORD PTR [rbp-0x4],0x0
   128a:    74 0c           je     1298 <vulfoo+0x3a>
   128c:    b8 00 00 00 00  mov    eax,0x0
   1291:    e8 f3 fe ff ff  call   1189 <print_flag>
   1296:    eb 0c           jmp    12a4 <vulfoo+0x46>
   1298:    48 8d 3d a6 0d 00 00  lea    rdi,[rip+0xda6]     # 2045
<_IO_stdin_used+0x45>
   129f:    e8 9c fd ff ff  call   1040 <puts@plt>
   12a4:    b8 00 00 00 00  mov    eax,0x0
   12a9:    c9              leave
   12aa:    c3              ret
```

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

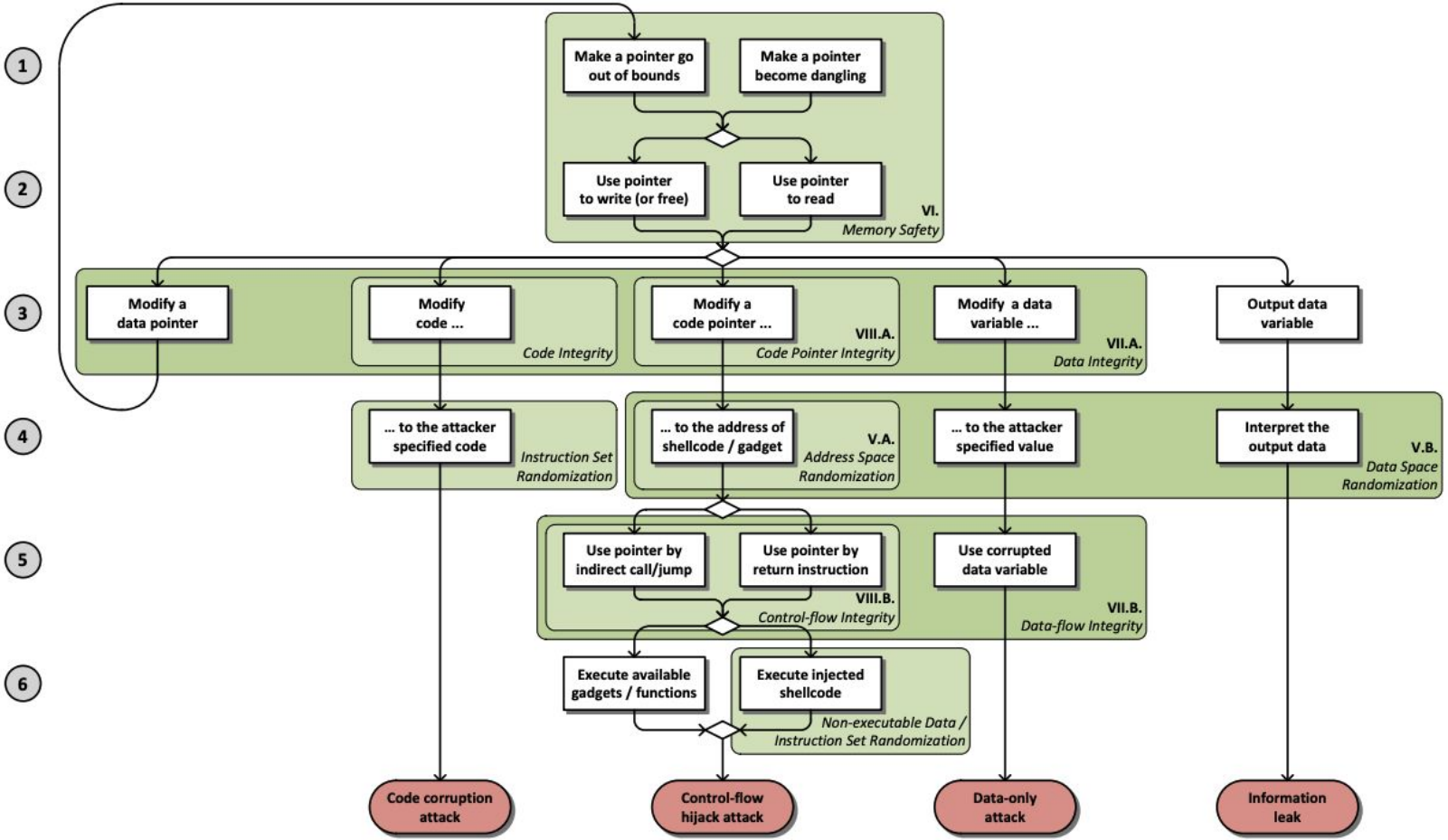
Run a program and use another program’s output as a parameter

```
./program $(python2 -c "print '\x12\x34'*5")
```

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
```



SoK: Eternal War in Memory. IEEE S&P 2013