CSE 410/510 Special Topics: Software Security

Instructor: Dr. Ziming Zhao

Location: Obrian 109
Time: Monday, Wednesday 5:00PM-6:20PM

Last Class

1. Stack-based buffer overflow
    a. Overwrite Saved EBP.

This Class

1. Stack-based buffer overflow
    a. Defense.

crackme4h

```
void printsecret(int i, int j, int k)
{
  if (i == 0xdeadbeef && j == 0xC0DECAFE && k == 0xD0D0FACE)
    print_flag();

  exit(0);}

int main(int argc, char *argv[])
{
  char buf[8];

  if (argc != 2)
    return 0;

  strcpy(buf, argv[1]);
}
```

crackme4

```
000012b7 <main>:
   12b7:f3 0f 1e fb        endbr32
   12bb:55                 push   %ebp
   12bc:89 e5              mov    %esp,%ebp
   12be:83 ec 08           sub    $0x8,%esp
   12c1:83 7d 08 02              cmpl   $0x2,0x8(%ebp)
   12c5:74 07              je     12ce <main+0x17>
   12c7:b8 00 00 00 00           mov    $0x0,%eax
   12cc:eb 1a              jmp    12e8 <main+0x31>
   12ce:8b 45 0c           mov    0xc(%ebp),%eax
   12d1:83 c0 04           add    $0x4,%eax
   12d4:8b 00              mov    (%eax),%eax
   12d6:50                 push   %eax
   12d7:8d 45 f8           lea    -0x8(%ebp),%eax
   12da:50                 push   %eax
   12db:e8 fc ff ff ff     call   12dc <main+0x25>
   12e0:83 c4 08           add    $0x8,%esp
   12e3:b8 00 00 00 00           mov    $0x0,%eax
   12e8:c9                 leave
   12e9:c3                 ret
   12ea:66 90              xchg   %ax,%ax
   12ec:66 90              xchg   %ax,%ax
   12ee:66 90              xchg   %ax,%ax
```

Arg3 = 0xd0doface

Arg2 = 0xcodecafe

Arg1 = 0xdeadbeef

4 bytes

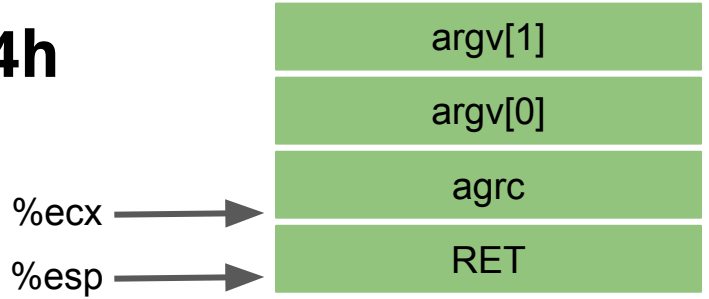
RET = printsecret

crackme4h

```
000012c6 <main>:
    12c6: f3 0f 1e fb       endbr32
    12ca: 8d 4c 24 04           lea    0x4(%esp),%ecx
    12ce: 83 e4 f0          and    $0xfffffff0,%esp
    12d1: ff 71 fc          pushl  -0x4(%ecx)
    12d4: 55                push   %ebp
    12d5: 89 e5             mov    %esp,%ebp
    12d7: 51                push   %ecx
    12d8: 83 ec 14          sub    $0x14,%esp
    12db: 89 c8             mov    %ecx,%eax
    12dd: 83 38 02          cmpl   $0x2,(%eax)
    12e0: 74 07             je     12e9 <main+0x23>
    12e2: b8 00 00 00 00        mov    $0x0,%eax
    12e7: eb 1d             jmp    1306 <main+0x40>
    12e9: 8b 40 04              mov    0x4(%eax),%eax
    12ec: 83 c0 04          add    $0x4,%eax
    12ef: 8b 00             mov    (%eax),%eax
    12f1: 83 ec 08          sub    $0x8,%esp
    12f4: 50                push   %eax
    12f5: 8d 45 f0          lea    -0x10(%ebp),%eax
    12f8: 50                push   %eax
    12f9: e8 fc ff ff ff    call   12fa <main+0x34>
    12fe: 83 c4 10          add    $0x10,%esp
    1301: b8 00 00 00 00        mov    $0x0,%eax
    1306: 8b 4d fc          mov    -0x4(%ebp),%ecx
    1309: c9                leave
    130a: 8d 61 fc          lea    -0x4(%ecx),%esp
    130d: c3                ret
```

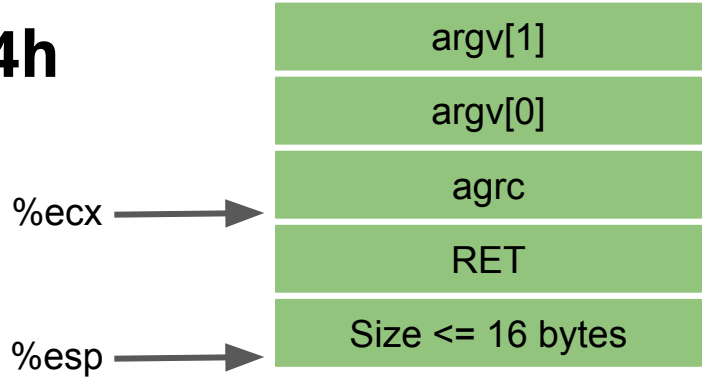
crackme4h

```
000012c6 <main>:
   12c6: f3 0f 1e fb        endbr32
   12ca: 8d 4c 24 04          lea    0x4(%esp),%ecx
   12ce: 83 e4 f0           and    $0xfffffff0,%esp
   12d1: ff 71 fc           pushl  -0x4(%ecx)
   12d4: 55                 push   %ebp
   12d5: 89 e5              mov    %esp,%ebp
   12d7: 51                 push   %ecx
   12d8: 83 ec 14           sub    $0x14,%esp
   12db: 89 c8              mov    %ecx,%eax
   12dd: 83 38 02           cmpl   $0x2,(%eax)
   12e0: 74 07              je     12e9 <main+0x23>
   12e2: b8 00 00 00 00           mov    $0x0,%eax
   12e7: eb 1d              jmp    1306 <main+0x40>
   12e9: 8b 40 04                 mov    0x4(%eax),%eax
   12ec: 83 c0 04           add    $0x4,%eax
   12ef: 8b 00              mov    (%eax),%eax
   12f1: 83 ec 08           sub    $0x8,%esp
   12f4: 50                 push   %eax
   12f5: 8d 45 f0           lea    -0x10(%ebp),%eax
   12f8: 50                 push   %eax
   12f9: e8 fc ff ff ff     call   12fa <main+0x34>
   12fe: 83 c4 10           add    $0x10,%esp
   1301: b8 00 00 00 00           mov    $0x0,%eax
   1306: 8b 4d fc           mov    -0x4(%ebp),%ecx
   1309: c9                 leave
   130a: 8d 61 fc           lea    -0x4(%ecx),%esp
   130d: c3                 ret
```



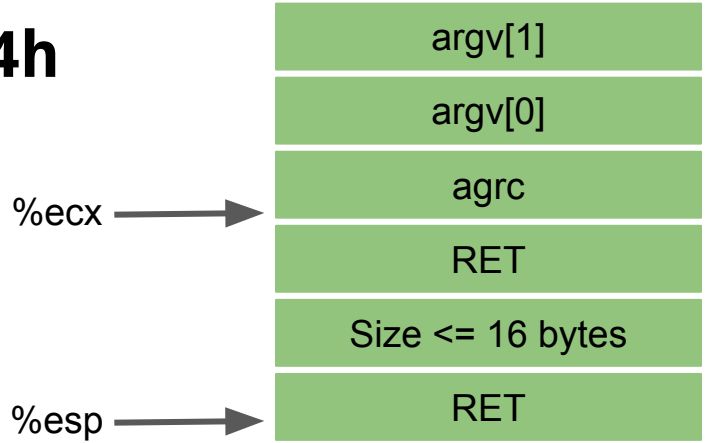
crackme4h

```
000012c6 <main>:
   12c6: f3 0f 1e fb        endbr32
   12ca: 8d 4c 24 04              lea    0x4(%esp),%ecx
   12ce: 83 e4 f0           and    $0xfffffff0,%esp
   12d1: ff 71 fc           pushl  -0x4(%ecx)
   12d4: 55                 push   %ebp
   12d5: 89 e5              mov    %esp,%ebp
   12d7: 51                 push   %ecx
   12d8: 83 ec 14           sub    $0x14,%esp
   12db: 89 c8              mov    %ecx,%eax
   12dd: 83 38 02           cmpl   $0x2,(%eax)
   12e0: 74 07              je     12e9 <main+0x23>
   12e2: b8 00 00 00 00           mov    $0x0,%eax
   12e7: eb 1d              jmp    1306 <main+0x40>
   12e9: 8b 40 04                 mov    0x4(%eax),%eax
   12ec: 83 c0 04           add    $0x4,%eax
   12ef: 8b 00              mov    (%eax),%eax
   12f1: 83 ec 08           sub    $0x8,%esp
   12f4: 50                 push   %eax
   12f5: 8d 45 f0           lea    -0x10(%ebp),%eax
   12f8: 50                 push   %eax
   12f9: e8 fc ff ff ff     call   12fa <main+0x34>
   12fe: 83 c4 10           add    $0x10,%esp
   1301: b8 00 00 00 00           mov    $0x0,%eax
   1306: 8b 4d fc           mov    -0x4(%ebp),%ecx
   1309: c9                 leave
   130a: 8d 61 fc           lea    -0x4(%ecx),%esp
   130d: c3                 ret
```



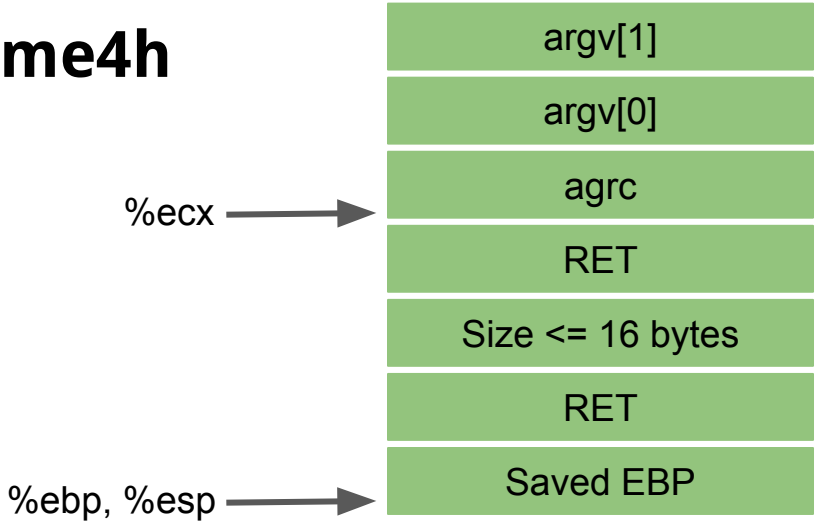
crackme4h

```
000012c6 <main>:
   12c6: f3 0f 1e fb        endbr32
   12ca: 8d 4c 24 04           lea    0x4(%esp),%ecx
   12ce: 83 e4 f0           and    $0xfffffff0,%esp
   12d1: ff 71 fc           pushl  -0x4(%ecx)
   12d4: 55                 push   %ebp
   12d5: 89 e5              mov    %esp,%ebp
   12d7: 51                 push   %ecx
   12d8: 83 ec 14           sub    $0x14,%esp
   12db: 89 c8              mov    %ecx,%eax
   12dd: 83 38 02           cmpl   $0x2,(%eax)
   12e0: 74 07              je     12e9 <main+0x23>
   12e2: b8 00 00 00 00        mov    $0x0,%eax
   12e7: eb 1d              jmp    1306 <main+0x40>
   12e9: 8b 40 04              mov    0x4(%eax),%eax
   12ec: 83 c0 04           add    $0x4,%eax
   12ef: 8b 00              mov    (%eax),%eax
   12f1: 83 ec 08           sub    $0x8,%esp
   12f4: 50                 push   %eax
   12f5: 8d 45 f0           lea    -0x10(%ebp),%eax
   12f8: 50                 push   %eax
   12f9: e8 fc ff ff ff     call   12fa <main+0x34>
   12fe: 83 c4 10           add    $0x10,%esp
   1301: b8 00 00 00 00        mov    $0x0,%eax
   1306: 8b 4d fc           mov    -0x4(%ebp),%ecx
   1309: c9                 leave
   130a: 8d 61 fc           lea    -0x4(%ecx),%esp
   130d: c3                 ret
```



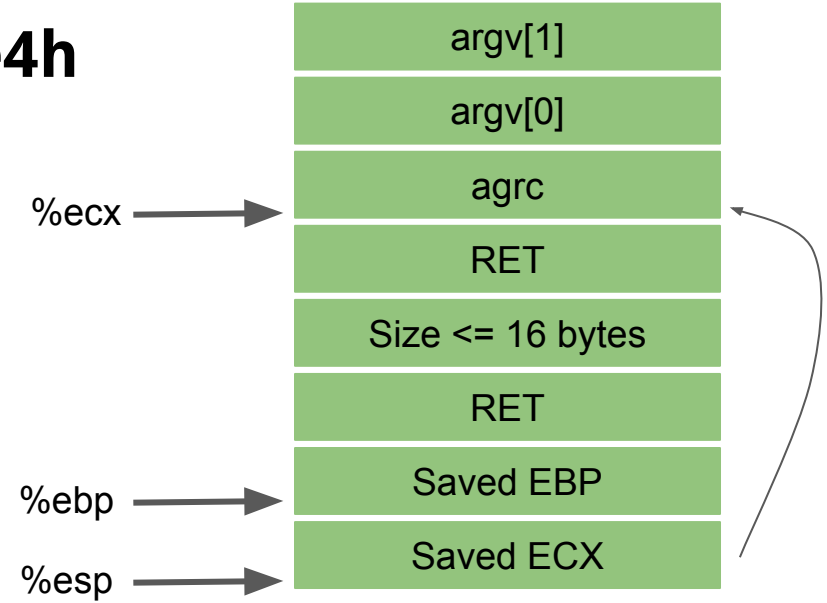
crackme4h

```
000012c6 <main>:
   12c6: f3 0f 1e fb        endbr32
   12ca: 8d 4c 24 04           lea    0x4(%esp),%ecx
   12ce: 83 e4 f0           and    $0xfffffff0,%esp
   12d1: ff 71 fc           pushl  -0x4(%ecx)
   12d4: 55                 push   %ebp
   12d5: 89 e5              mov    %esp,%ebp
   12d7: 51                 push   %ecx
   12d8: 83 ec 14           sub    $0x14,%esp
   12db: 89 c8              mov    %ecx,%eax
   12dd: 83 38 02           cmpl   $0x2,(%eax)
   12e0: 74 07              je     12e9 <main+0x23>
   12e2: b8 00 00 00 00        mov    $0x0,%eax
   12e7: eb 1d              jmp    1306 <main+0x40>
   12e9: 8b 40 04              mov    0x4(%eax),%eax
   12ec: 83 c0 04           add    $0x4,%eax
   12ef: 8b 00              mov    (%eax),%eax
   12f1: 83 ec 08           sub    $0x8,%esp
   12f4: 50                 push   %eax
   12f5: 8d 45 f0           lea    -0x10(%ebp),%eax
   12f8: 50                 push   %eax
   12f9: e8 fc ff ff ff     call   12fa <main+0x34>
   12fe: 83 c4 10           add    $0x10,%esp
   1301: b8 00 00 00 00        mov    $0x0,%eax
   1306: 8b 4d fc           mov    -0x4(%ebp),%ecx
   1309: c9                 leave
   130a: 8d 61 fc           lea    -0x4(%ecx),%esp
   130d: c3                 ret
```



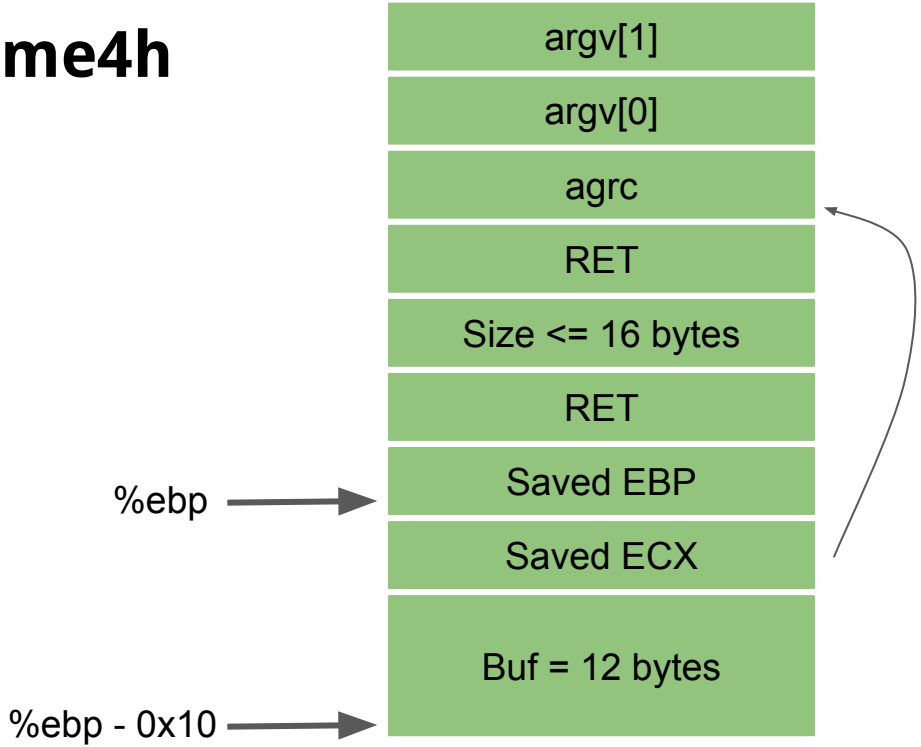
crackme4h

```
000012c6 <main>:
   12c6: f3 0f 1e fb        endbr32
   12ca: 8d 4c 24 04              lea    0x4(%esp),%ecx
   12ce: 83 e4 f0           and    $0xfffffff0,%esp
   12d1: ff 71 fc           pushl  -0x4(%ecx)
   12d4: 55                 push   %ebp
   12d5: 89 e5              mov    %esp,%ebp
   12d7: 51                 push   %ecx
   12d8: 83 ec 14           sub    $0x14,%esp
   12db: 89 c8              mov    %ecx,%eax
   12dd: 83 38 02           cmpl   $0x2,(%eax)
   12e0: 74 07              je     12e9 <main+0x23>
   12e2: b8 00 00 00 00           mov    $0x0,%eax
   12e7: eb 1d              jmp    1306 <main+0x40>
   12e9: 8b 40 04                 mov    0x4(%eax),%eax
   12ec: 83 c0 04           add    $0x4,%eax
   12ef: 8b 00              mov    (%eax),%eax
   12f1: 83 ec 08           sub    $0x8,%esp
   12f4: 50                 push   %eax
   12f5: 8d 45 f0           lea    -0x10(%ebp),%eax
   12f8: 50                 push   %eax
   12f9: e8 fc ff ff ff     call   12fa <main+0x34>
   12fe: 83 c4 10           add    $0x10,%esp
   1301: b8 00 00 00 00           mov    $0x0,%eax
   1306: 8b 4d fc           mov    -0x4(%ebp),%ecx
   1309: c9                 leave
   130a: 8d 61 fc           lea    -0x4(%ecx),%esp
   130d: c3                 ret
```



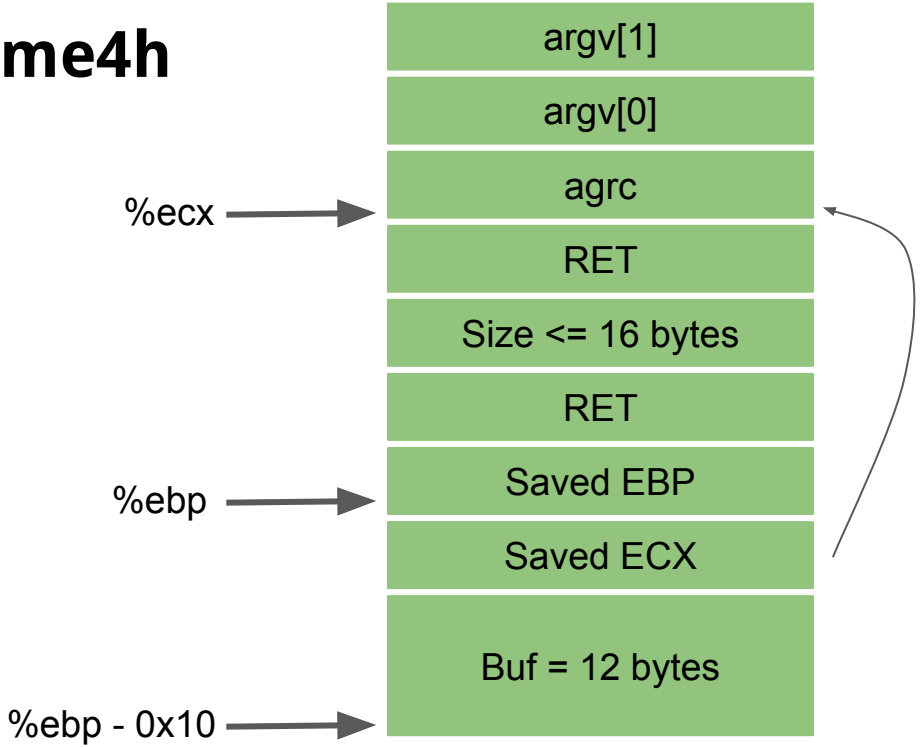
crackme4h

```
000012c6 <main>:
   12c6: f3 0f 1e fb        endbr32
   12ca: 8d 4c 24 04            lea    0x4(%esp),%ecx
   12ce: 83 e4 f0           and    $0xfffffff0,%esp
   12d1: ff 71 fc           pushl  -0x4(%ecx)
   12d4: 55                 push   %ebp
   12d5: 89 e5              mov    %esp,%ebp
   12d7: 51                 push   %ecx
   12d8: 83 ec 14           sub    $0x14,%esp
   12db: 89 c8              mov    %ecx,%eax
   12dd: 83 38 02           cmpl   $0x2,(%eax)
   12e0: 74 07              je     12e9 <main+0x23>
   12e2: b8 00 00 00 00             mov    $0x0,%eax
   12e7: eb 1d              jmp    1306 <main+0x40>
   12e9: 8b 40 04                mov    0x4(%eax),%eax
   12ec: 83 c0 04           add    $0x4,%eax
   12ef: 8b 00              mov    (%eax),%eax
   12f1: 83 ec 08           sub    $0x8,%esp
   12f4: 50                 push   %eax
   12f5: 8d 45 f0           lea    -0x10(%ebp),%eax
   12f8: 50                 push   %eax
   12f9: e8 fc ff ff ff     call   12fa <main+0x34>
   12fe: 83 c4 10           add    $0x10,%esp
   1301: b8 00 00 00 00             mov    $0x0,%eax
   1306: 8b 4d fc           mov    -0x4(%ebp),%ecx
   1309: c9                 leave
   130a: 8d 61 fc           lea    -0x4(%ecx),%esp
   130d: c3                 ret
```



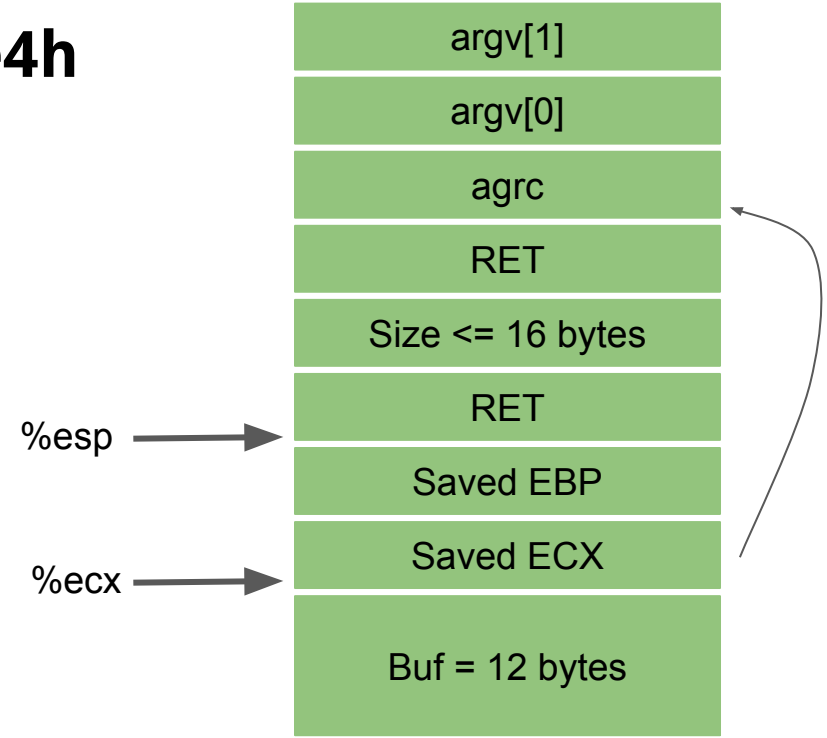
crackme4h

```
000012c6 <main>:
    12c6: f3 0f 1e fb       endbr32
    12ca: 8d 4c 24 04           lea    0x4(%esp),%ecx
    12ce: 83 e4 f0          and    $0xfffffff0,%esp
    12d1: ff 71 fc          pushl  -0x4(%ecx)
    12d4: 55                push   %ebp
    12d5: 89 e5             mov    %esp,%ebp
    12d7: 51                push   %ecx
    12d8: 83 ec 14          sub    $0x14,%esp
    12db: 89 c8             mov    %ecx,%eax
    12dd: 83 38 02          cmpl   $0x2,(%eax)
    12e0: 74 07             je     12e9 <main+0x23>
    12e2: b8 00 00 00 00            mov    $0x0,%eax
    12e7: eb 1d             jmp    1306 <main+0x40>
    12e9: 8b 40 04              mov    0x4(%eax),%eax
    12ec: 83 c0 04          add    $0x4,%eax
    12ef: 8b 00             mov    (%eax),%eax
    12f1: 83 ec 08          sub    $0x8,%esp
    12f4: 50                push   %eax
    12f5: 8d 45 f0          lea    -0x10(%ebp),%eax
    12f8: 50                push   %eax
    12f9: e8 fc ff ff ff    call   12fa <main+0x34>
    12fe: 83 c4 10          add    $0x10,%esp
    1301: b8 00 00 00 00            mov    $0x0,%eax
    1306: 8b 4d fc          mov    -0x4(%ebp),%ecx
    1309: c9                leave
    130a: 8d 61 fc          lea    -0x4(%ecx),%esp
    130d: c3                ret
```



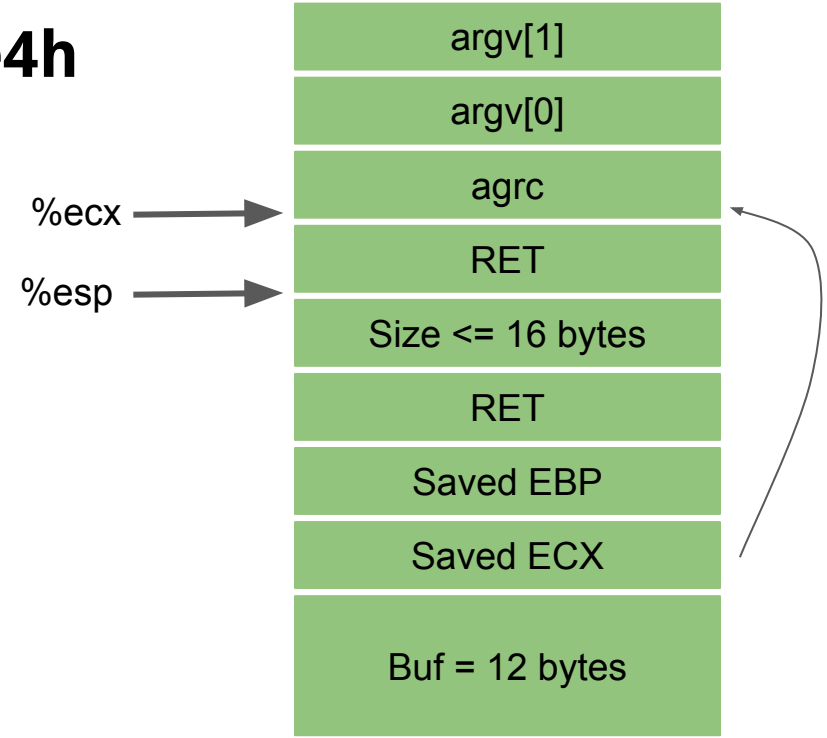
crackme4h

```
000012c6 <main>:
   12c6: f3 0f 1e fb        endbr32
   12ca: 8d 4c 24 04              lea    0x4(%esp),%ecx
   12ce: 83 e4 f0           and    $0xfffffff0,%esp
   12d1: ff 71 fc           pushl  -0x4(%ecx)
   12d4: 55                 push   %ebp
   12d5: 89 e5              mov    %esp,%ebp
   12d7: 51                 push   %ecx
   12d8: 83 ec 14           sub    $0x14,%esp
   12db: 89 c8              mov    %ecx,%eax
   12dd: 83 38 02           cmpl   $0x2,(%eax)
   12e0: 74 07              je     12e9 <main+0x23>
   12e2: b8 00 00 00 00           mov    $0x0,%eax
   12e7: eb 1d              jmp    1306 <main+0x40>
   12e9: 8b 40 04                 mov    0x4(%eax),%eax
   12ec: 83 c0 04           add    $0x4,%eax
   12ef: 8b 00              mov    (%eax),%eax
   12f1: 83 ec 08           sub    $0x8,%esp
   12f4: 50                 push   %eax
   12f5: 8d 45 f0           lea    -0x10(%ebp),%eax
   12f8: 50                 push   %eax
   12f9: e8 fc ff ff ff     call   12fa <main+0x34>
   12fe: 83 c4 10           add    $0x10,%esp
   1301: b8 00 00 00 00           mov    $0x0,%eax
   1306: 8b 4d fc           mov    -0x4(%ebp),%ecx
   1309: c9                 leave
   130a: 8d 61 fc           lea    -0x4(%ecx),%esp
   130d: c3                 ret
```

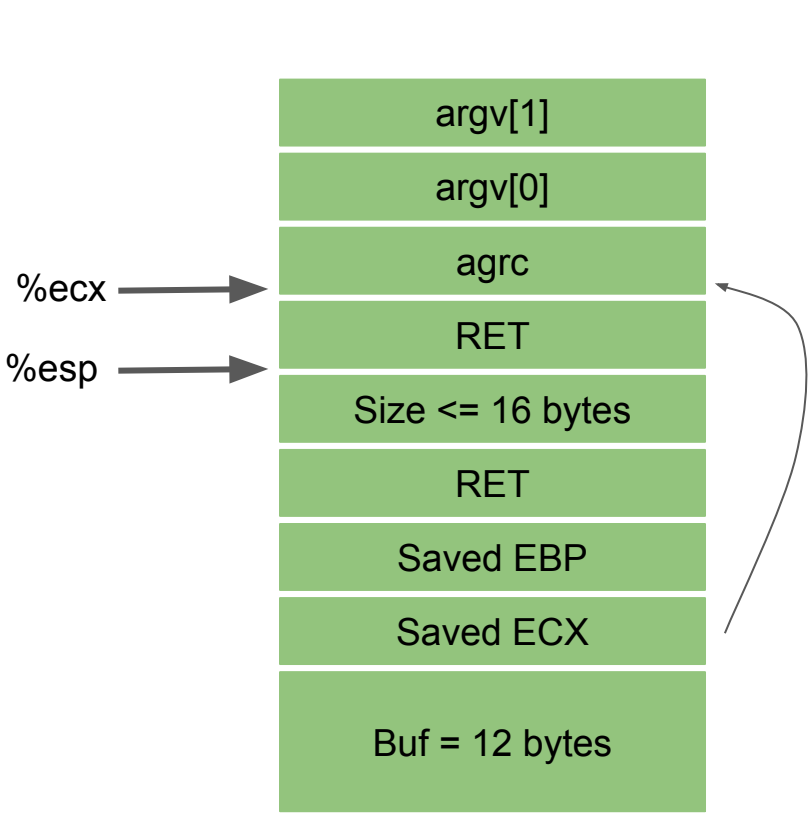


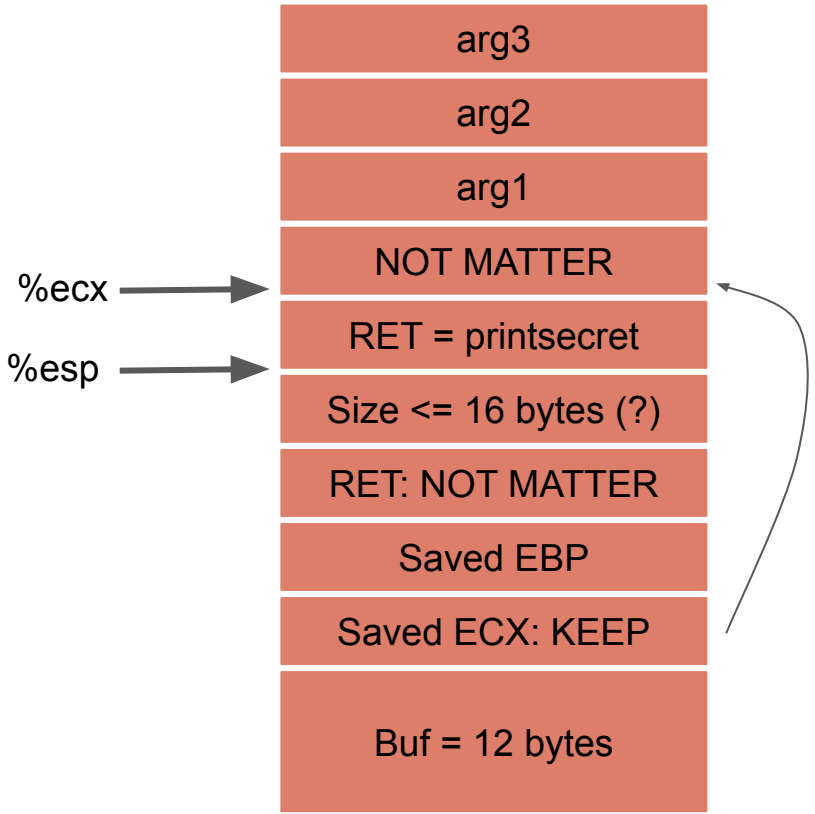
crackme4h

```
000012c6 <main>:
   12c6: f3 0f 1e fb        endbr32
   12ca: 8d 4c 24 04           lea    0x4(%esp),%ecx
   12ce: 83 e4 f0           and    $0xfffffff0,%esp
   12d1: ff 71 fc           pushl  -0x4(%ecx)
   12d4: 55                 push   %ebp
   12d5: 89 e5              mov    %esp,%ebp
   12d7: 51                 push   %ecx
   12d8: 83 ec 14           sub    $0x14,%esp
   12db: 89 c8              mov    %ecx,%eax
   12dd: 83 38 02           cmpl   $0x2,(%eax)
   12e0: 74 07              je     12e9 <main+0x23>
   12e2: b8 00 00 00 00        mov    $0x0,%eax
   12e7: eb 1d              jmp    1306 <main+0x40>
   12e9: 8b 40 04              mov    0x4(%eax),%eax
   12ec: 83 c0 04           add    $0x4,%eax
   12ef: 8b 00              mov    (%eax),%eax
   12f1: 83 ec 08           sub    $0x8,%esp
   12f4: 50                 push   %eax
   12f5: 8d 45 f0           lea    -0x10(%ebp),%eax
   12f8: 50                 push   %eax
   12f9: e8 fc ff ff ff     call   12fa <main+0x34>
   12fe: 83 c4 10           add    $0x10,%esp
   1301: b8 00 00 00 00        mov    $0x0,%eax
   1306: 8b 4d fc           mov    -0x4(%ebp),%ecx
   1309: c9                 leave
   130a: 8d 61 fc           lea    -0x4(%ecx),%esp
   130d: c3                 ret
```



Crackme4h
Craft the exploit





crackme464

```
0000000000001200 <printsecret>:
    1200: f3 0f 1e fa          endbr64
    1204: 55                   push   %rbp
    1205: 48 89 e5             mov    %rsp,%rbp
    1208: 48 83 ec 10          sub    $0x10,%rsp
    120c: 89 7d fc             mov    %edi,-0x4(%rbp)
    120f: 89 75 f8             mov    %esi,-0x8(%rbp)
    1212: 89 55 f4             mov    %edx,-0xc(%rbp)
    1215: 81 7d fc ef be ad de cmpl   $0xdeadbeef,-0x4(%rbp)
    121c: 75 32                jne    1250 <printsecret+0x50>
    121e: 81 7d f8 fe ca de c0 cmpl   $0xc0decafe,-0x8(%rbp)
    1225: 75 29                jne    1250 <printsecret+0x50>
    1227: 81 7d f4 ce fa d0 d0 cmpl   $0xd0d0face,-0xc(%rbp)
    122e: 75 20                jne    1250 <printsecret+0x50>
    1230: 48 8d 3d d9 2d 00 00 lea    0x2dd9(%rip),%rdi        # 4010 <s>
    1237: e8 6d ff ff ff       callq  11a9 <decrypt>
    123c: 48 89 c6             mov    %rax,%rsi
    123f: 48 8d 3d c2 0d 00 00 lea    0xdc2(%rip),%rdi        # 2008 <_IO_stdin_used+0x8>
    1246: b8 00 00 00 00       mov    $0x0,%eax
    124b: e8 50 fe ff ff       callq  10a0 <printf@plt>
    1250: bf 00 00 00 00       mov    $0x0,%edi
    1255: e8 56 fe ff ff       callq  10b0 <exit@plt>
```

Return to here!!

Conditions we depend on to pull off the attack of *returning to shellcode on stack*

1. The ability to put the shellcode onto stack (env, command line)
2. The stack is executable
3. The ability to overwrite RET addr on stack before instruction **ret** is executed or to overwrite Saved EBP
4. Know the address of the destination function

Conditions we depend on to pull off the attack of *returning to shellcode on stack*

1. The ability to put the shellcode onto stack (env, command line)
2. ~~The stack is executable~~
3. The ability to overwrite RET addr on stack before instruction **ret** is executed or to overwrite Saved EBP
4. Know the address of the destination function

Defense 1:
Data Execution Prevention
(DEP, W⊕X, NX)

Harvard vs. Von-Neumann Architecture

Harvard Architecture

The Harvard architecture stores machine instructions and data in separate memory units that are connected by different busses. In this case, there are at least two memory address spaces to work with, so there is a memory register for machine instructions and another memory register for data. Computers designed with the Harvard architecture are able to run a program and access data independently, and therefore simultaneously. Harvard architecture has a strict separation between data and code. Thus, Harvard architecture is more complicated but separate pipelines remove the bottleneck that Von Neumann creates.

Von-Neumann architecture

In a Von-Neumann architecture, the same memory and bus are used to store both data and instructions that run the program. Since you cannot access program memory and data memory simultaneously, the Von Neumann architecture is susceptible to bottlenecks and system performance is affected.

Older CPUs

Older CPUs: Read permission on a page implies execution. So all readable memory was executable.

AMD64 – introduced NX bit (No-eXecute in 2003)

Windows Supporting DEP from Windows XP SP2 (in 2004)

Linux Supporting NX since 2.6.8 (in 2004)

Modern CPUs

Modern architectures support memory permissions:

- **PROT_READ** allows the process to read memory
- **PROT_WRITE** allows the process to write memory
- **PROT_EXEC** allows the process to execute memory

gcc parameter ***-z execstack*** to disable this protection

```
ziming@ziming-XPS-13-9300:~/Dropbox/myTeaching/System Security - Attack and Defense for Binaries UB 2020/code/overflow6$ readelf -l of6

Elf file type is DYN (Shared object file)
Entry point 0x1090
There are 12 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x00000034 0x00000034 0x00180 0x00180 R   0x4
  INTERP         0x0001b4 0x000001b4 0x000001b4 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x00000000 0x00000000 0x003f8 0x003f8 R   0x1000
  LOAD           0x001000 0x00001000 0x00001000 0x002d4 0x002d4 R E 0x1000
  LOAD           0x002000 0x00002000 0x00002000 0x001ac 0x001ac R   0x1000
  LOAD           0x002ed8 0x00003ed8 0x00003ed8 0x00130 0x00134 RW  0x1000
  DYNAMIC        0x002ee0 0x00003ee0 0x00003ee0 0x000f8 0x000f8 RW  0x4
  NOTE           0x0001c8 0x000001c8 0x000001c8 0x00060 0x00060 R   0x4
  GNU_PROPERTY   0x0001ec 0x000001ec 0x000001ec 0x0001c 0x0001c R   0x4
  GNU_EH_FRAME   0x002008 0x00002008 0x00002008 0x0005c 0x0005c R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RWE 0x10
  GNU_RELRO      0x002ed8 0x00003ed8 0x00003ed8 0x00128 0x00128 R   0x1
```

```
ziming@ziming-XPS-13-9300:~/Dropbox/myTeaching/System Security - Attack and Defense for Binaries UB 2020/code/overflow6$ readelf -l of6nx

Elf file type is DYN (Shared object file)
Entry point 0x1090
There are 12 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x00000034 0x00000034 0x00180 0x00180 R   0x4
  INTERP         0x0001b4 0x000001b4 0x000001b4 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x00000000 0x00000000 0x003f8 0x003f8 R   0x1000
  LOAD           0x001000 0x00001000 0x00001000 0x002d4 0x002d4 R E 0x1000
  LOAD           0x002000 0x00002000 0x00002000 0x001ac 0x001ac R   0x1000
  LOAD           0x002ed8 0x00003ed8 0x00003ed8 0x00130 0x00134 RW  0x1000
  DYNAMIC        0x002ee0 0x00003ee0 0x00003ee0 0x000f8 0x000f8 RW  0x4
  NOTE           0x0001c8 0x000001c8 0x000001c8 0x00060 0x00060 R   0x4
  GNU_PROPERTY   0x0001ec 0x000001ec 0x000001ec 0x0001c 0x0001c R   0x4
  GNU_EH_FRAME   0x002008 0x00002008 0x00002008 0x0005c 0x0005c R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x10
  GNU_RELRO      0x002ed8 0x00003ed8 0x00003ed8 0x00128 0x00128 R   0x1
```

What DEP cannot prevent

Can still corrupt stack or function pointers or critical data on the heap

As long as RET (saved EIP) points into legit code section, W⊕X protection will not block control transfer

Ret2libc 32bit
Bypassing NX

Discovered by Solar Designer, 1997

Ret2libc

Now programs built with non-executable stack.

Then, how to run a shell? Ret to C library ***system("/bin/sh")*** like how we called printsecret() in overflowret

Description

The C library function **int system(const char *command)** passes the command name or program name specified by **command** to the host environment to be executed by the command processor and returns after the command has been completed.

Declaration

Following is the declaration for system() function.

```
int system(const char *command)
```

Parameters

- **command** − This is the C string containing the name of the requested variable.

Return Value

The value returned is -1 on error, and the return status of the command otherwise.

Buffer Overflow Example: code/overflowret4 32-bit (overflowret4_no_excNry_32)

```
int vulfoo()
{
  char buf[30];

  gets(buf);
  return 0;
}

int main(int argc, char *argv[])
{
  vulfoo();
  printf("I pity the fool!\n");
}
```

Conditions we depend on to pull off the attack of *ret2libc*

1. ~~The ability to put the shellcode onto stack (env, command line)~~
2. ~~The stack is executable~~
3. The ability to overwrite RET addr on stack before instruction **ret** is executed or to overwrite Saved EBP
4. Know the address of the destination function and arguments

Control Hijacking Attacks

Control flow

- Order in which individual statements, instructions or function calls of a program are executed or evaluated

Control Hijacking Attacks (Runtime exploit)

- A control hijacking attack exploits a program error, particularly a memory corruption vulnerability, at application runtime to subvert the intended control-flow of a program.
- Alter a code pointer (i.e., value that influences program counter) or, Gain control of the instruction pointer %eip
- Change memory region that should not be accessed

Code Injection Attacks

Code-injection Attacks

- a subclass of control hijacking attacks that subverts the intended control-flow of a program to previously injected malicious code

Shellcode

- code supplied by attacker − often saved in buffer being overflowed − traditionally transferred control to a shell (user command-line interpreter)
- machine code − specific to processor and OS − traditionally needed good assembly language skills to create − more recently have automated sites/tools

Code-Reuse Attack

Code-Reuse Attack: a subclass of control-flow attacks that subverts the intended control-flow of a program to invoke an unintended execution path inside the original program code.

Return-to-Libc Attacks (Ret2Libc)
Return-Oriented Programming (ROP)
Jump-Oriented Programming (JOP)

Exercise: Overthewire /maze/maze2

Overthewire

http://overthewire.org/wargames/

1. Open a terminal
2. Type: ssh -p 2225 maze2@maze.labs.overthewire.org
3. Input password: **fooghihahr**
4. cd /maze; this is where the binary are
5. Your goal is to get the password of maze3