Lec 10: Shellcoding

CS492E: Introduction to Software Security

Sang Kil Cha

Recap: Executing *Shellcode*

- Small piece of code that is used as the payload

- Shellcode can run any arbitrary logic
  - Download /etc/passwd
  - Install malicious software (malware)
  - …

- But typically executing `/bin/sh` is enough
  - This is the most powerful attack: we can run arbitrary commands
  - You can also achieve this with relatively ***small amount of code***
  - This is the reason why we call it as shellcode (code that typically runs shell)

Shellcoding

How to write code that executes `/bin/sh`?

execve() Function in libc

```
EXECVE(2)                  Linux Programmer's Manual                  EXECVE(2)

NAME
       execve - execute program

SYNOPSIS
       #include <unistd.h>

       int execve(const char *filename, char *const argv[],
                  char *const envp[]);
```

Executable path

Command line arguments

Environment variables

```
/*
int execve(const char *filename, char *const argv[],
           char *const envp[]);
*/

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

System Calls allow a program to interface with OS

```
0805c3e0 <__execve>:
 805c3e0:       53                      push   ebx
 805c3e1:       8b 54 24 10             mov    edx,DWORD PTR [esp+0x10]
 805c3e5:       8b 4c 24 0c             mov    ecx,DWORD PTR [esp+0xc]
 805c3e9:       8b 5c 24 08             mov    ebx,DWORD PTR [esp+0x8]
 805c3ed:       b8 0b 00 00 00          mov    eax,0xb
 805c3f2:       cd 80                   int    0x80
 ...
```

Register	Meaning
eax	System call number
ebx	1st argument
ecx	2nd argument
edx	3rd argument

Register	Meaning
esi	4th argument
edi	5th argument
ebp	6th argument
eax	Return value

List of System Calls for x86

See: /usr/include/x86_64-linux-gnu/asm/unistd_32.h

```
#define __NR_restart_syscall 0
#define __NR_exit 1
#define __NR_fork 2
#define __NR_read 3
#define __NR_write 4
#define __NR_open 5
#define __NR_close 6
#define __NR_waitpid 7
#define __NR_creat 8
#define __NR_link 9
#define __NR_unlink 10
#define __NR_execve 11
#define __NR_chdir 12
…
```

0xb

How about x86-64?

- Always consider using ‘R’ as a prefix
  - RAX, RBX, RSP, RIP, etc.
- 64-bit addresses
- Totally different syscall calling convention compared to x86

x86-64 Syscall Calling Convention

Register	Meaning
`rax`	System call number
`rdi`	1st argument
`rsi`	2nd argument
`rdx`	3rd argument

Register	Meaning
`r10`	4th argument
`r8`	5th argument
`r9`	6th argument
`rax`	Return value

* Use `syscall` instruction instead of `int 80` to generate a software interrupt

SYSCALL (SYSENTER) vs. INT 80?

SYSCALL is optimized to provide the maximum performance for system calls from user code running at privilege level 3 to operating system or executive procedures running at privilege level 0.

INT 80 = x86 (invalid on x86-64)
SYSCALL = x86-64 (invalid on x86)
SYSENTER = both x86 and x86-64

CALL gs:0x10?

- The instruction dereferences (gs_base + 0x10), where gs_base means an address of the TCB of the current process.

- In x86, TCB + 0x10 stores a stub code that essentially uses SYSENTER inside.

Exercise: Writing Hello World

```
.intel_syntax noprefix
.global _start
_start:
mov eax, 4
mov ebx, 1
lea ecx, msg
lea edx, len
int 0x80
mov eax, 1
mov ebx, 0
int 0x80
msg:
  .ascii "Hello world!\n"
  len = . - msg
```

gcc –m32 –o hello hello.s -nostdlib

Exercise: Writing Hello World

```
.intel_syntax noprefix
.global _start
 start:
```

Is this a shellcode?

```
int 0x80
mov eax, 1
mov ebx, 0
int 0x80
msg:
  .ascii "Hello world!\n"
  len = . - msg
```

Key Property of Shellcode

There should be no direct reference to code/data!

Shellcode Version of Hello World?

- Use `push` instructions to push the string to the stack
  - Don't forget to push a zero (null) at the end!

- Get the address of the string from `esp`

Shellcoding Practice

- Write your own execve shellcode!

- Test it with shelleval: https://github.com/sangkilc/shelleval

Questions?