二进制漏洞-栈溢出

测试平台

系统：CentOS release 6.10 (Final)、32 位

内核版本：Linux 2.6.32-754.10.1.el6.i686　i686 i386 GNU/Linux

gcc 版本：　4.4.7 20120313 (Red Hat 4.4.7-23) (GCC)

gdb 版本：GNU gdb (GDB) Red Hat Enterprise Linux (7.2-92.el6)

libc 版本：libc-2.12.so

漏洞原理

在对栈缓冲区进行写操作时（如 memcpy），未对缓冲区大小进行判断，导致写入数据长度可能大于缓冲区长度。

通用利用方式

写入数据覆盖返回地址，使返回地址指向恶意代码起始地址。由于我是基于本地测试，也就是 libc 库的版本已知，而基于远程攻击或不同版本的 libc 库可能会存在差异。

漏洞测试程序

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

void wh(const char* w) {
        if(strncmp(w, "say", 3) == 0) {
                printf("you said hello world!\n");
        } else {
                printf("you said fuck you!\n");
        }
}

int main() {
        char buf[16];

        memset(buf, 0, 16);
        printf("please input a word!\n");

        scanf("%s", buf);

        wh(buf);
        return 0;

}
```

很明显代码在执行 scanf 时未对缓冲区大小进行判断，存在栈溢出漏洞。

注意如无特殊说明，本文的 exp 都是基于该源码编译的二进制实现的。

所有测试均在 linux 环境下进行

开启 FORTIFY

1、FORTIFY 其实非常轻微的检查，用于检查是否存在缓冲区溢出的错误。适用情形是程序采用大量的字符串或者内存操作函数，如 memcpy，memset，stpcpy，strcpy，strncpy，strcat，strncat，sprintf，snprintf，vsprintf，vsnprintf，gets 以及宽字符的变体。

2、FORTIFY_SOURCE 机制对格式化字符串有两个限制(1)包含%n 的格式化字符串不能位于程序内存中的可写地址。(2)当使用位置参数时，必须使用范围内的所有参数。所以如果要使用%7$x，你必须同时使用 1,2,3,4,5 和 6。

gcc -D_FORTIFY_SOURCE=1 -O1 仅仅只会在编译时进行检查（特别像某些头文件 #include <string.h>）

gcc -D_FORTIFY_SOURCE=2 -O2 程序执行时也会有检查（如果检查到缓冲区溢出，就终止程序)

```
[sp00f@localhost fortify]$ pwn checksec test_fortify
[!] Pwntools does not support 32-bit Python.  Use a 64-bit release.
[*] '/home/sp00f/vul_test/stack_overflow/fortify/test_fortify'
    Arch:     i386-32-little
    RELRO:    No RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
    FORTIFY:  Enabled
[sp00f@localhost fortify]$
```

先让我们看看添加 FORTIFY 编译选项(第一条影响)前后的汇编区别，未添加 fortify 选项：

```
080484b5 <main>:
 80484b5:       55                      push   %ebp
 80484b6:       89 e5                   mov    %esp,%ebp
 80484b8:       83 e4 f0                and    $0xfffffff0,%esp
 80484bb:       83 ec 20                sub    $0x20,%esp
 80484be:       c7 44 24 08 10 00 00    movl   $0x10,0x8(%esp)
 80484c5:       00
 80484c6:       c7 44 24 04 00 00 00    movl   $0x0,0x4(%esp)
 80484cd:       00
 80484ce:       8d 44 24 10             lea    0x10(%esp),%eax
 80484d2:       89 04 24                mov    %eax,(%esp)
 80484d5:       e8 8e fe ff ff          call   8048368 <memset@plt>
 80484da:       c7 04 24 01 86 04 08    movl   $0x8048601,(%esp)
 80484e1:       e8 b2 fe ff ff          call   8048398 <puts@plt>
 80484e6:       b8 16 86 04 08          mov    $0x8048616,%eax
 80484eb:       8d 54 24 10             lea    0x10(%esp),%edx
 80484ef:       89 54 24 04             mov    %edx,0x4(%esp)
 80484f3:       89 04 24                mov    %eax,(%esp)
 80484f6:       e8 8d fe ff ff          call   8048388 <__isoc99_scanf@plt>
 80484fb:       8d 44 24 10             lea    0x10(%esp),%eax
 80484ff:       89 04 24                mov    %eax,(%esp)
 8048502:       e8 6d ff ff ff          call   8048474 <wh>
 8048507:       b8 00 00 00 00          mov    $0x0,%eax
```

添加编译 forify 编译选项后的反汇编，从两幅图对比可以看到开启 forify 后调用 memset 汇编指令被替换了其他指令。

```
080484a0 <main>:
 80484a0:       55                      push   %ebp
 80484a1:       89 e5                   mov    %esp,%ebp
 80484a3:       83 e4 f0                and    $0xfffffff0,%esp
 80484a6:       53                      push   %ebx
 80484a7:       83 ec 2c                sub    $0x2c,%esp
 80484aa:       8d 5c 24 10             lea    0x10(%esp),%ebx
 80484ae:       c7 44 24 10 00 00 00    movl   $0x0,0x10(%esp)
 80484b5:       00
 80484b6:       c7 44 24 14 00 00 00    movl   $0x0,0x14(%esp)
 80484bd:       00
 80484be:       c7 44 24 18 00 00 00    movl   $0x0,0x18(%esp)
 80484c5:       00
 80484c6:       c7 44 24 1c 00 00 00    movl   $0x0,0x1c(%esp)
 80484cd:       00
 80484ce:       c7 44 24 04 03 86 04    movl   $0x8048603,0x4(%esp)
 80484d5:       08
 80484d6:       c7 04 24 01 00 00 00    movl   $0x1,(%esp)
 80484dd:       e8 66 fe ff ff          call   8048348 <__printf_chk@plt>
 80484e2:       89 5c 24 04             mov    %ebx,0x4(%esp)
 80484e6:       c7 04 24 19 86 04 08    movl   $0x8048619,(%esp)
 80484ed:       e8 76 fe ff ff          call   8048368 <__isoc99_scanf@plt>
 80484f2:       89 1c 24                mov    %ebx,(%esp)
 80484f5:       e8 46 ff ff ff          call   8048440 <wh>
```

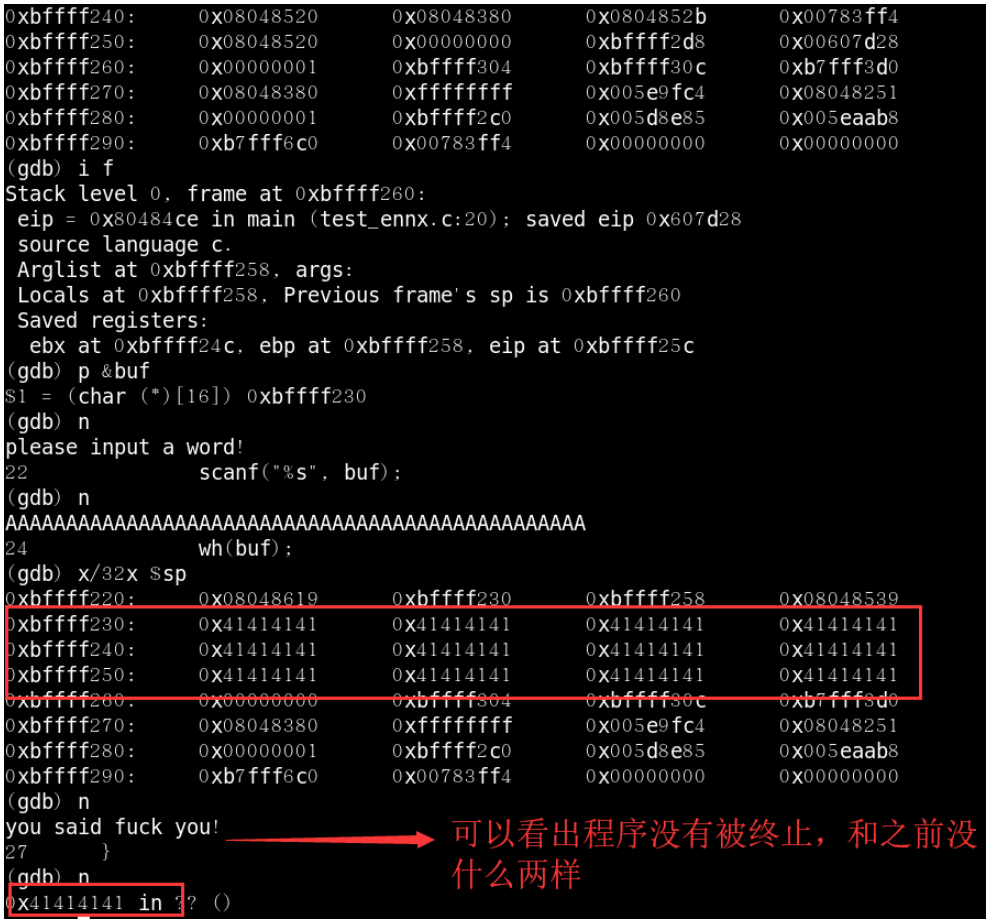
memset没了

同样我们通过 gdb 调试可以看到原来的 memset 被替换成了__builtin__memset_chk

```
Reading symbols from /home/sp00f/vul_test/stack_overflow/fortify/test_fortify...done.
(gdb) b 22
Breakpoint 1 at 0x80484aa: file /usr/include/bits/string3.h, line 22.
(gdb) r
Starting program: /home/sp00f/vul_test/stack_overflow/fortify/test_fortify

Breakpoint 1, main () at /usr/include/bits/string3.h:85
85        return __builtin___memset_chk (__dest, __ch, __len, __bos0 (__dest));
(gdb) n
20              printf("please input a word!\n");
(gdb) x/32x $sp
0xbffff220:     0x00000016      0x0804979c      0xbffff258      0x08048539
0xbffff230:     0x00000000      0x00000000      0x00000000      0x00000000
0xbffff240:     0x08048520      0x08048380      0x0804852b      0x00783ff4
0xbffff250:     0x08048520      0x00000000      0xbffff2d8      0x00607d28
0xbffff260:     0x00000001      0xbffff304      0xbffff30c      0xb7fff3d0
0xbffff270:     0x08048380      0xffffffff      0x005e9fc4      0x08048251
0xbffff280:     0x00000001      0xbffff2c0      0x005d8e85      0x005eaab8
0xbffff290:     0xb7fff6c0      0x00783ff4      0x00000000      0x00000000
```

我们继续覆盖返回地址看程序是否会被终止，从下图可以看出程序没有被终止，和我们之前



测试没什么两样。

我先后在编译时测试，编译选项中不加入-O1、-O2 发现 memset 不会被替换，并且用 checksec 检测也不到开启 fortify，也就是说加了优化选项 memset 就被优化了，而只有加了优化选项后 fortify 才会被开启。

在让我们看看添加了 FORTIFY 编译选项后对第二条关于 printf 格式化的影响，我们重新编写一个明显包含格式化字符串的程序，代码如下：

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main() {
        char buf[16];

        memset(buf, 0, 16);
        printf("please input a word!\n");

        printf("%7$#x\n");

        scanf("%s", buf);

        return 0;

}
```

测试运行如下图，很明显它检测出了格式化字符串异常。

```
[sp00f@localhost fortify]$ ./test_fortify_print
please input a word!
*** invalid %N$ use detected ***
已放弃
```

漏洞分析（略）

实现 exp（略）