

**实验报告**



**题目： 缓冲区溢出攻击实验**

**班 级： 2020211306**

**学 号： 2020211376**

**姓 名： 马天成**

**学 院： 计算机学院**

**2021年 12 月 4 日**

目录

[一、实验目的 3](#_Toc89767188)

[二、实验环境 3](#_Toc89767189)

[三、实验内容 3](#_Toc89767190)

[四、实验步骤及实验分析 4](#_Toc89767191)

[准备工作 4](#_Toc89767192)

[攻击位置 - getbuf() 4](#_Toc89767193)

[代码逐行分析 5](#_Toc89767194)

[攻击方式选择 5](#_Toc89767195)

[文件命名解释 6](#_Toc89767196)

[Ⅰ. touch1 6](#_Toc89767197)

[初步理解 7](#_Toc89767198)

[攻击功能与过程 7](#_Toc89767199)

[攻击文件 8](#_Toc89767200)

[攻击结果 8](#_Toc89767201)

[Ⅱ. touch2 8](#_Toc89767202)

[初步理解 9](#_Toc89767203)

[攻击功能与过程 9](#_Toc89767204)

[攻击文件 11](#_Toc89767205)

[攻击结果 11](#_Toc89767206)

[Ⅲ. touch3 11](#_Toc89767207)

[初步理解 12](#_Toc89767208)

[攻击功能与过程 13](#_Toc89767209)

[攻击文件 15](#_Toc89767210)

[攻击结果 15](#_Toc89767211)

[Ⅳ. touch4 15](#_Toc89767212)

[初步理解 15](#_Toc89767213)

[攻击功能与过程 16](#_Toc89767214)

[攻击文件 17](#_Toc89767215)

[攻击结果 17](#_Toc89767216)

[Ⅴ. touch5 17](#_Toc89767217)

[初步理解 17](#_Toc89767218)

[攻击功能与过程 18](#_Toc89767219)

[攻击文件 20](#_Toc89767220)

[攻击结果 21](#_Toc89767221)

[EX：另类方法 21](#_Toc89767222)

[五、总结体会 22](#_Toc89767223)

[六、诚信声明 22](#_Toc89767224)

# **一、实验目的**

1. C语言程序的机器级表示。  
2. 掌握GDB调试器的用法。  
3. C编译器生成的x86-64机器代码，理解不同控制结构生成的基本指令模式，过程的实现。

4. 掌握两种缓冲区攻击方法，进一步理解软件漏洞的危害。

# 二、实验环境

SecureCRT（10.120.11.12）

Linux

Objdump命令反汇编

GDB调试工具

积分榜（http://10.120.11.13:19320/scoreboard）

**报告邮寄（最迟时间：2020年12月8日晚23：59）：**

**大二班（5-8班）：**yangyyj98@bupt.edu.cn

# 三、实验内容

登录bupt1服务器，在home目录下可以找到一个targetn.tar文件，解压后得到如下文件：

README.txt；

ctarget；

rtarget；

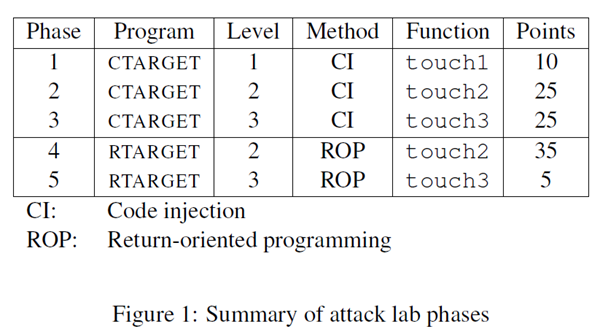
cookie.txt；

farm.c；

hex2raw。

ctarget和rtarget运行时从标准输入读入字符串，这两个程序都存在缓冲区溢出漏洞。通过代码注入的方法实现对ctarget程序的攻击，共有3关，输入一个特定字符串，可成功调用touch1，或touch2，或touch3就通关，并向计分服务器提交得分信息；通过ROP方法实现对rtarget程序的攻击，共有2关，在指定区域找到所需要的小工具，进行拼接完成指定功能，再输入一个特定字符串，实现成功调用touch2或touch3就通关，并向计分服务器提交得分信息；否则失败，但不扣分。因此，本实验需要通过反汇编和逆向工程对ctraget和rtarget执行文件进行分析，找到保存返回地址在堆栈中的位置以及所需要的小工具机器码。实验的具体内容见实验说明，尤其需要认真阅读各阶段的Some Advice提示。

本实验包含了5个阶段（或关卡），难度逐级递增。各阶段分数如下所示：



# 四、实验步骤及实验分析

1. Attack Lab的内容针对的是CS-APP中第三章中关于程序安全性描述中的栈溢出攻击。

2. 在这个Lab中，我们需要针对不同的目的编写攻击字符串来填充一个有漏洞的程序的栈来达到执行攻击代码的目的。

3. 攻击方式分为**代码注入攻击**与**返回导向编程攻击**。

## 准备工作

主目录中有一个 **target201.tar**。

通过**tar -xvf targetk.tar > lab3**，提取文件到文件夹 **lab3**。

文件lab3中的文件包括：

**README.txt**：描述目录内容的文件

**ctarget**：易受代码注入攻击的可执行程序；

**rtarget**：易受面向返回编程攻击的可执行程序；

**cookie.txt**：一个 8 位十六进制代码，等于是身份认证；

**farm.c**：目标攻击串的源代码，可以截取适当指令来生成面向返回的编程攻击；

**hex2raw**：生成攻击字符串；

## 攻击位置 - getbuf()

**CTARGET** 和 **RTARGET** 都从标准输入读取字符串。

他们使用下面定义的函数 **getbuf()** 来做到这一点：

1 unsigned getbuf()

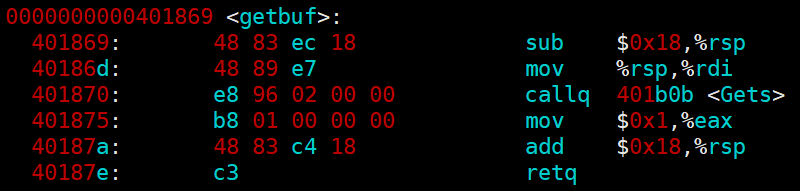
2 {

3 char buf[BUFFER\_SIZE];

4 Gets(buf);

5 return 1;

6 }



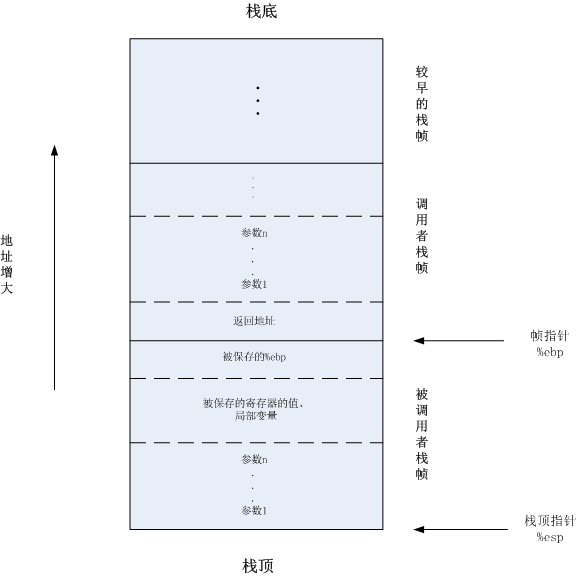
**Picture**-getbuf() 汇编代码

### 代码逐行分析

1. **rsp** = **rsp** - **0x18** ：申请了一块24字节的空间；不同于正常函数的栈-8，它额外申请了16个字节空间，将其作为缓冲区进行存储buf数组；
2. **rdi** = **rsp** ：将当前栈顶作为参数传入Gets()函数，进行信息的读取；
3. **callq Gets()** ：调用Gets函数进行数据读入；
4. **eax** = **1** ：说明函数完成了它的“功能”，将返回值设置为1；
5. **rsp** = **rsp** + **0x18** ：进行返回前的操作，rsp返回原值；
6. **ret** ：返回，“释放”函数栈。

### 攻击方式选择

我们用图来说明栈的结构：



我们可以确定**BUFFER\_SIZE**的大小为**0x18**。这个**BUFFER\_SIZE**是由服务器生成的。

在**0x18**字节的栈被**Gets**函数写满之后，多出来的字符会被写入**getbuf()** 函数的栈外。

**target**给了**0x18**的缓冲区大小。我可以在这里：

* 写指令进行代码注入攻击方式；
* 将缓冲区填满后进行返回导向编 程攻击，将栈上的覆盖进行攻击。

**touch1**：熟悉攻击方式，熟悉栈；

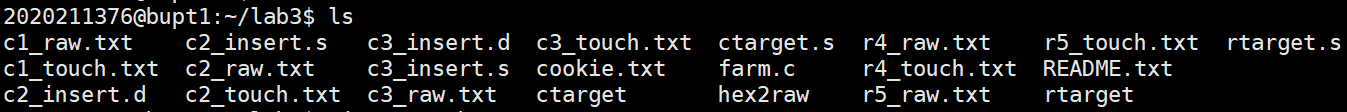
**touch2**：进行简单的代码注入攻击；

**touch3**：进行较难的代码注入攻击；

**touch4**：进行简单的返回导向编程攻击；

**touch5**：进行较难的返回导向编程攻击。

## 文件命名解释



**Picture**\_lab3\_files

只解释自己创建的Files：

**①**. **ctarget**中：

* **ctarget.s** ：反汇编的 .s文件；
* **ck\_touch.txt** ( **k** = 1 , 2 , 3 ) ：注入的字节以数字表示形式txt；
* **ck\_raw.txt** ( **k** = 1 , 2 , 3 ) ：ck\_touch.txt用hex2raw转出来的字节txt；
* **ck\_insert.s** ( **k** = 2 , 3 ) : 需要注入的汇编指令txt；
* **ck\_insert.d** ( **k** = 2 , 3 ) : 用objdump -d指令创建的带字节的汇编指令txt；

**②**. **rtarget**中：

* **rtarget.s** ：反汇编的 .s文件；
* **rk\_touch.txt** ( **k** = 1 , 2 ) ：注入的字节以数字表示形式txt；
* **rk\_raw.txt** ( **k** = 1 , 2 ) ：ck\_touch.txt用hex2raw转出来的字节txt；

## Ⅰ. touch1

1 void test()

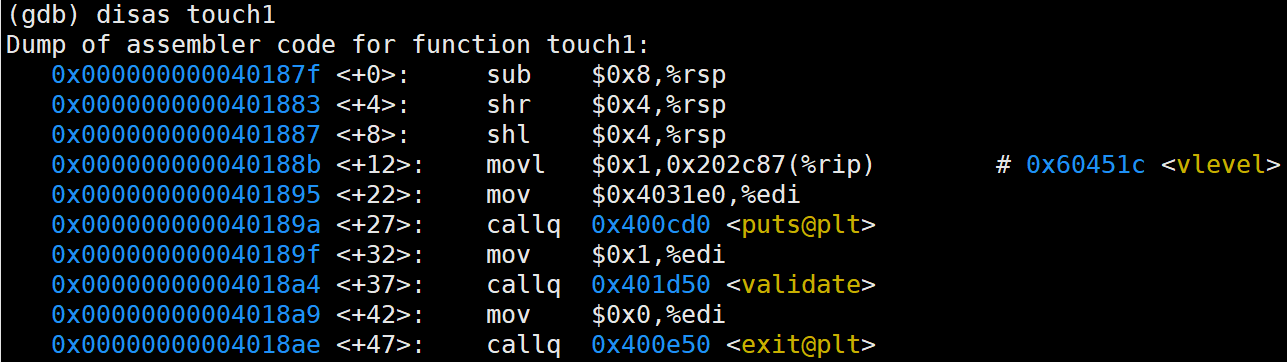
2 {

3 int val;

4 val = getbuf();

5 printf("No exploit. Getbuf returned 0x%x\n", val);

6 }

 **Picture**\_touch1\_code

### 初步理解

**①**. 函数 **test()**

--------------------------------------------------------------

void test()

{

int val;

val = getbuf();

printf("NO explit. Getbuf returned 0x%x\n", val);

}

---------------------------------------------------------

调用了函数**getbuf() :**执行返回语句时，程序会继续执行test() 函数中的语句。

而我们要改变这个行为，使 getbuf返回的时候，执行 touch1（）而不是返回test（）。

**②**. 函数 **touch1()**

---------------------------------------------------------------

void touch1()

{

vlevel = 1; // **vlevel - 检验子，=1 则认定是touch1()**

printf("Touch!: You called touch1()\n");

validate(1);

exit(0);

}

----------------------------------------------------------

函数内部并没有说明操作，只需要能成功进入函数就算攻击成功。

### 攻击功能与过程

**功能**：将返回地址改成touch1的首地址。

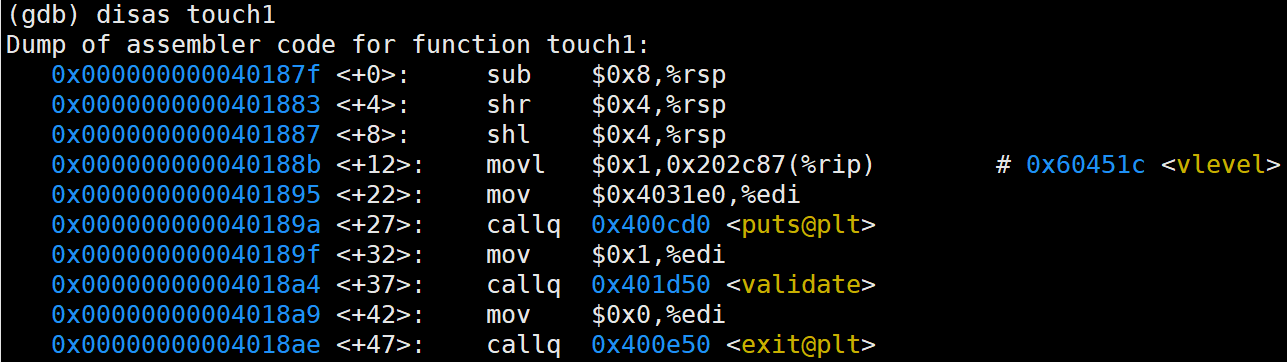
**过程**：

**①**. **将缓冲区填满**

这个只需要将申请的0x18字节的空间全部填满即可。

我选择全部填成00字节。

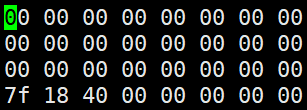
**②**. **将getbuf()的返回地址改为touch1()的首地址**



可知，我们需要将返回地址覆盖为8字节是：**7f 18 40 00 00 00 00 00**

### 攻击文件

**①. c1\_touch.txt**

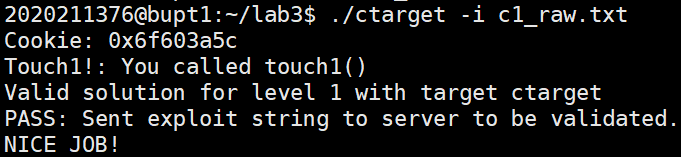
****

前三行将缓冲区填满，第四行将返回地址改为touch1()的首地址**0x40187f**，以**小端方式**存储。

**②. c1\_raw.txt**



### 攻击结果



**Picture**\_c1\_attack\_success

## Ⅱ. touch2

1 void touch2(unsigned val)

2 {

3 svlevel = 2; /\* Part of validation protocol \*/

4 if (val == cookie) {

5 printf("Touch2!: You called touch2(0x%.8x)\n", val);

6 validate(2);

7 } else {

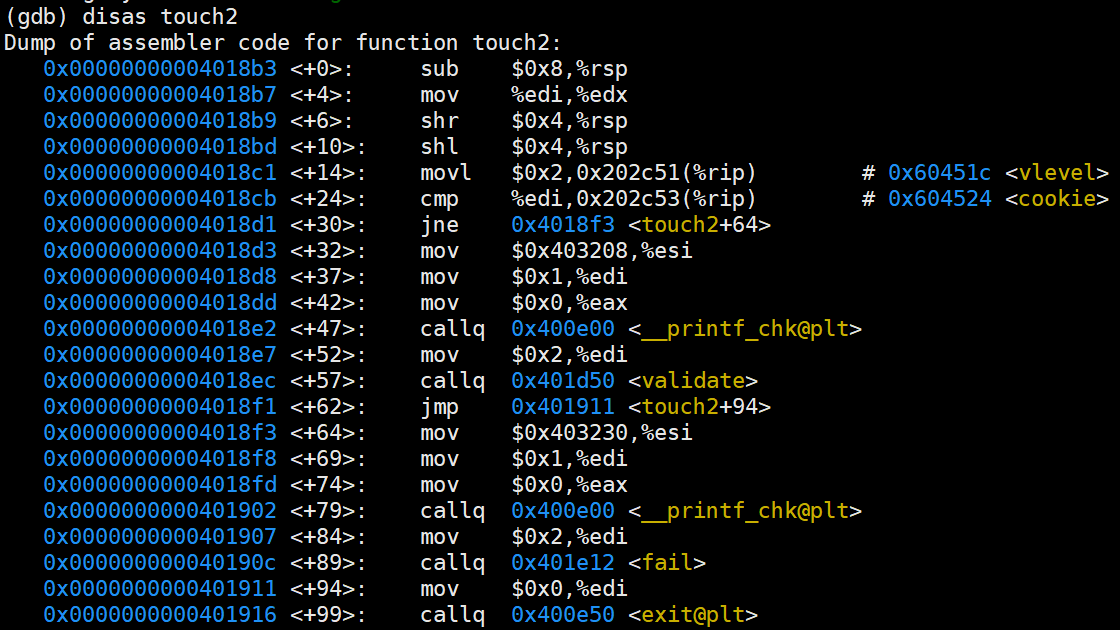
8 printf("Misfire: You called touch2(0x%.8x)\n", val);

9 fail(2);

10 }

11 exit(0);

12 }



**Picture**\_touch2\_code

### 初步理解

代码的功能是**进行cookie数字匹配**。touch2的参数存放在寄存器rdi中，我们就是要将其设为cookie。

* touch2也是代码注入攻击。我们需要将**合适的代码以字节形式注入到缓存区里**进行攻击指令。
* 在touch1的基础上，我们需要实现将**cookie以16进制数字形式注入第一个寄存器**并且返回touch2进行touch2的数字匹配。

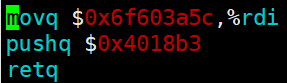
### 攻击功能与过程

**功能**：

* 将**cookie**的十六进制数传入第一个参数寄存器；
* 进行字节指令的注入和返回地址的修改。

**过程**：

**①. 写汇编代码**



**Picture**\_insert\_s

这里实现的是将**cookie**传入**rdi**寄存器（第一个参数寄存器）；

并且将返回地址修改成**touch2()** **起始地址**；

ex：（由开头汇编代码知起始地址为**0x4018b3**）

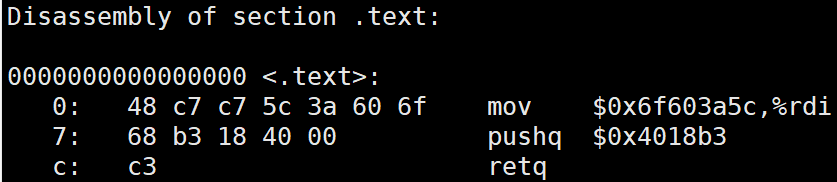
**②. 用汇编代码生成 .d 文件**



**Picture**\_生成\_d

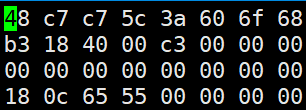
用上述指令进行 c2\_insert.d 文件的生成。

生成文件如下：



**Picture**\_insert\_d

可以通过这个知道我们指令的字节形式。

 **③. 编写touch.txt**

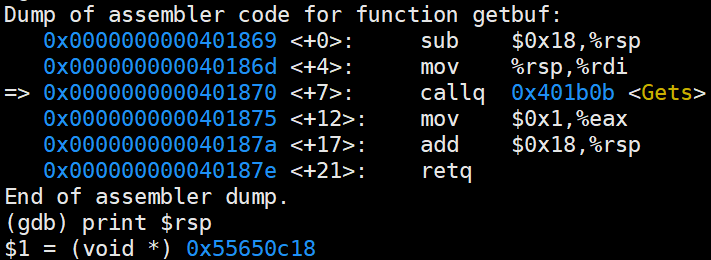
// 将指令以字节形式存起来

// 栈顶地址，便于getbuf() 执行完毕后进行touch3() **Picture**\_touch\_ans

前三行是缓存区，进行汇编代码字节形式的存储；

第四行是我的栈顶；要将**返回地址设在栈顶**才能在getbuf返回后执行我的攻击指令。

**rsp**：



**Picture**\_rsp

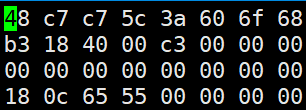
知道我要存的地址是**0x55650c18**，以**小端方式**存储。

**④. 生成raw.txt**

****

### 攻击文件

**①. c2\_touch.txt**



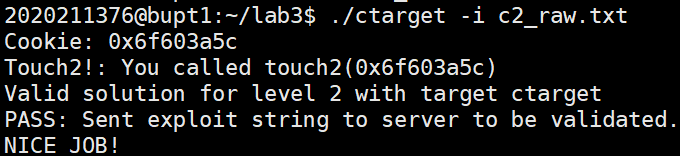
**Picture**\_touch

**②. c2\_raw.txt**

****

**Picture**\_raw

### 攻击结果



**Picture**\_c2\_attack\_success

## Ⅲ. touch3

1 /\* Compare string to hex represention of unsigned value \*/

2 int hexmatch(unsigned val, char \*sval)

3 {

4 char cbuf[110];

5 /\* Make position of check string unpredictable \*/

6 char \*s = cbuf + random() % 100;

7 sprintf(s, "%.8x", val);

8 return strncmp(sval, s, 9) == 0;

9 }

11 void touch3(char \*sval)

12 {

13 vlevel = 3; /\* Part of validation protocol \*/

14 if (hexmatch(cookie, sval)) {

15 printf("Touch3!: You called touch3(\"%s\")\n", sval);

16 validate(3);

17 } else {

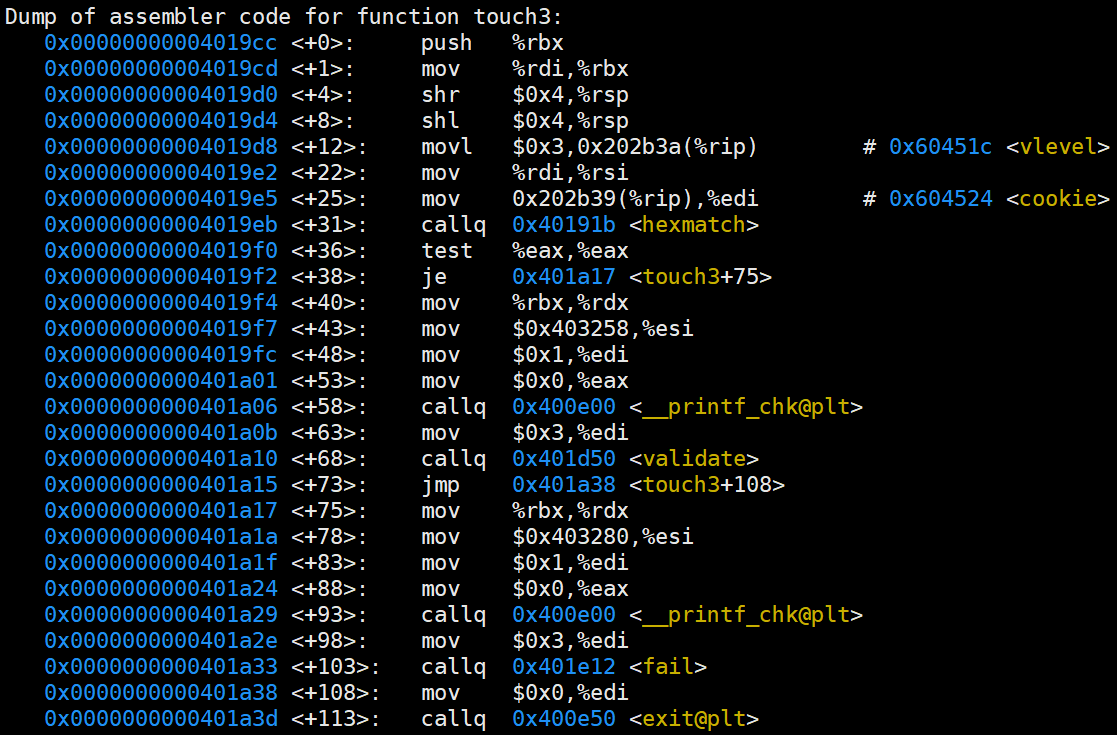
18 printf("Misfire: You called touch3(\"%s\")\n", sval);

19 fail(3);

20 }

21 exit(0);

22 }



**Picture**\_touch\_code

### 初步理解

在前两题的基础上，我们默认能进入touch3；

touch3() ：

需要在touch3() 调用hexmatchh() 并成功运行返回1。

hexmatch() ：

需要将**cookie以字节形式**存在一个空间里并且将其首地址以参数形式传给**rdi**寄存器。

无需担心匹配问题，他只是随机生成一个空间匹配。

所以需要在**getbuf() 运行完毕**后进行cookie字节形式的匹配。

### 攻击功能与过程

**功能**：

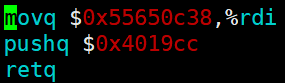
* 将cookie以字节形式存在一个空间里；
* 进入函数touch3() ；
* 把cookie首地址传给rdi。

**过程**：

设计**栈指令**： 地址增大

|  |
| --- |
| Cookie字节空间 |
| 返回地址 -> 指令 |
| 00 |
| 填充指令 |
| 填充指令 |

**①. 写汇编代码**



**Picture**\_insert\_s

这里实现的是将**cookie地址**传入**rdi**寄存器（第一个参数寄存器）；

并且将返回地址修改成**touch3()** **起始地址**；

数据：

* 由开头汇编代码知起始地址为**0x4019cc**
* 我将cookie字节存在了缓冲区上边的第二行空间，所以地址是**0x55650c38**。

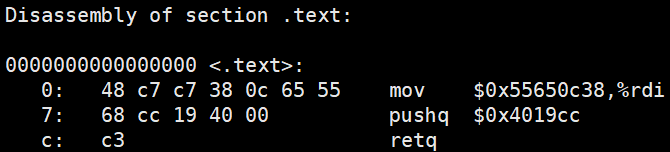
**②. 用汇编代码生成 .d 文件**



**Picture**\_生成\_d

用上述指令进行 **c3\_insert.d** 文件的生成。

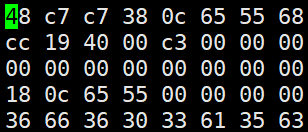
生成文件如下：



**Picture**\_insert\_d

可以通过这个知道我们指令的字节形式。

**③. 编写touch.txt**



// 将指令以字节形式存起来

// 栈顶地址 getbuf() 执行完毕后进行touch3() // 字节形式的cookie

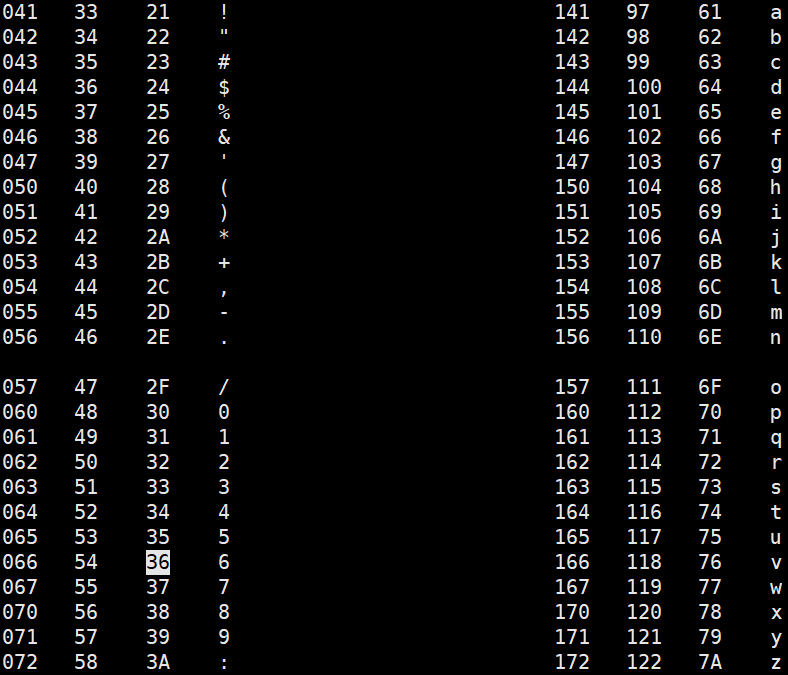
**Picture**\_touch\_ans

前三行是缓存区，进行汇编代码字节形式的存储；

第四行是我的栈顶；要将**返回地址设在栈顶**才能在getbuf返回后执行我的攻击指令；

第五行是我的字节cookie。

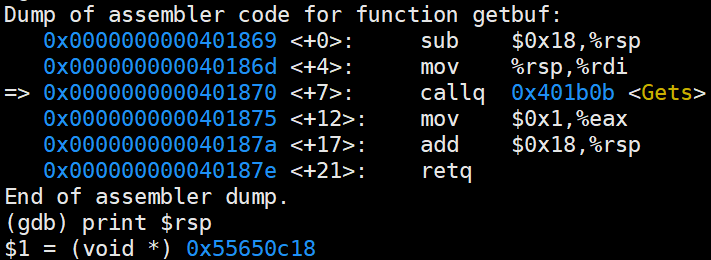
**cookie字节**：



**Picture**\_man\_ascii

查表知我的**cookie字节**是 **36 66 36 30 33 61 35 63**

**rsp**：



**Picture**\_rsp

知道我要存的地址是**0x55650c18**，以**小端方式**存储。

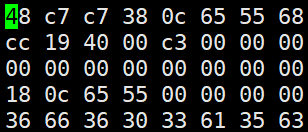
**④. 生成raw.txt**



**Picture**\_raw

### 攻击文件

**①. c2\_touch.txt**



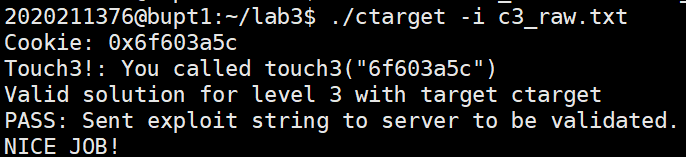
**Picture**\_touch

**②. c2\_raw.txt**



**Picture**\_raw

### 攻击结果



**Picture**\_c3\_attack\_success

## Ⅳ. touch4

由于本题和level2很相似，就是重复level2的任务，所以这里就不展开赘述了。

### 初步理解

因为栈是**随机化**的，我不能将代码注入执行，也不能找到固定的rsp值。这使得代码注入不可行。

所以这里要用到**截取原有指令**进行指令拼凑，达到想要的目的。

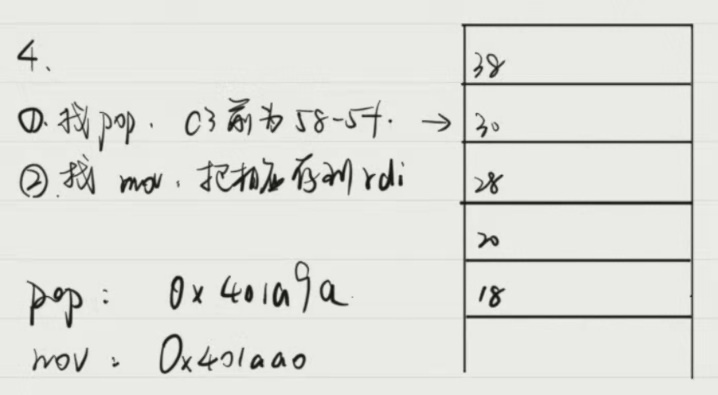
程序提供了**farm**，我们可以在这里找到指令（也只有在这里）。

### 攻击功能与过程

**功能**：与level2相同，这里我们主要实现代码指令的选择：

避免将**cookie数字**直接写入指令，我们使用截取**pop**指令将立即数传入**rdi**寄存器。

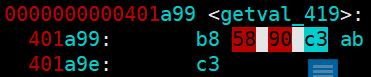
但是我们只有pop %rax 指令，所以我们只有**“曲线救国”**：

* **pop %rax**
* **movq %rax %rdi**

**过程**：（在level2的基础上）

**①. 找指令（查表）**

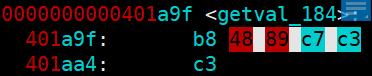
**pop Picture**\_手稿



**Picture**\_pop

加上位移量为 **0x401a9a** ;

**move**

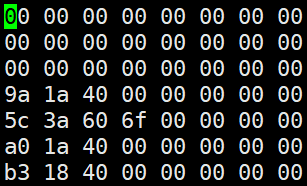
****

**Picture**\_move

加上位移量为**0x401aa0**。

**②. 写攻击文件**

如上图栈所示，我需要将**缓冲区填满**后进行指令的截取和使用。

****

// 填满缓冲区

// **pop**下一行的立即数给 %rax

// **cookie**十六进制数字

// **mov** %rax %rdi

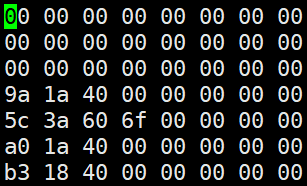
// 返回 touch4 ( touch4 首地址为**0x4018b3** )

**Picture**\_touch

这样就能实现我们第四题的功能啦！

### 攻击文件

**①. r4\_touch.txt**



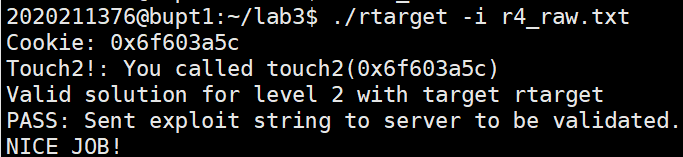
**Picture**\_touch

**②. 生成 raw.txt**

****

**Picture**\_raw

### 攻击结果



**Picture**\_r4\_attack\_success

## Ⅴ. touch5

### 初步理解

此题与level3类似，这里也不太多赘述。

我们要实现的就是和level3一样的功能并且不能使用代码注入，只能返回攻击。

所以我们需要的就是：

找到合适的截取指令，**将其地址塞到我们缓冲区之上合适位置**。

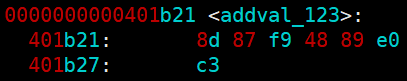
### 攻击功能与过程

**功能**：与level3类似，但难点在于我们不能直接写出我们cookie字节的地址。

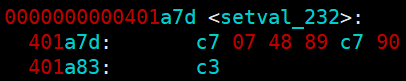
因为有add\_xy函数我们可以直接用，所以我们可以用**rsp加位移量**解决。

**①. 找到能用指令**

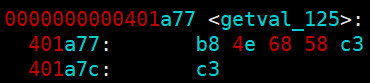
官方提示我们可以使用8个指令解决。这里只放出该8个指令，其实总共有19个指令可选。



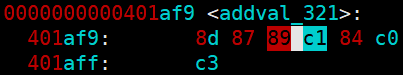
**Picture**\_rax=rsp



**Picture**\_rdi=rax



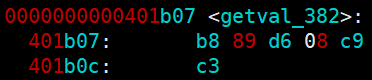
**Picture**\_pop rax



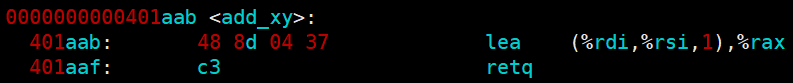
**Picture**\_ecx=eax&test al al



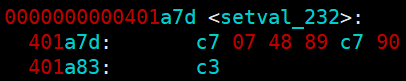
**picture**\_edx=ecx



**Picture**\_esi=edx&or cl cl

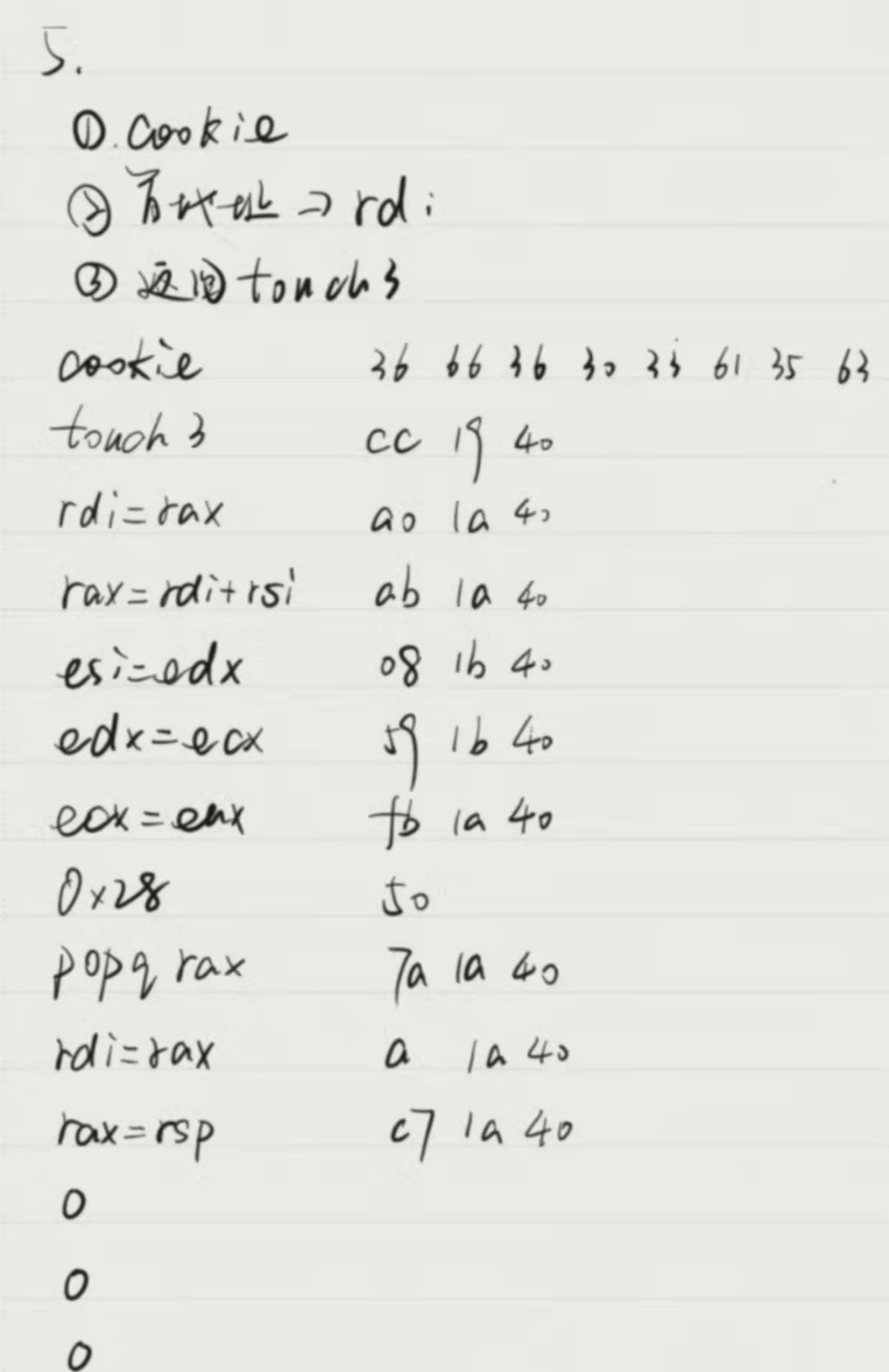


**Picture**\_add\_xy



**Picture**\_rdi=rax

**②. 拼凑指令**

****

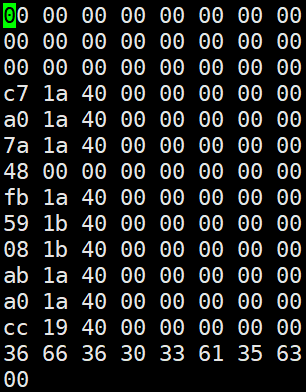
**Picture**\_手稿

通过这种方式就能实现：

* **rdi** = **rsp**+**位移量** ( 我设计的为**0x48** )

达到将cookie字节地址输入rdi的效果。

**③. 编写r5\_touch.txt**

****

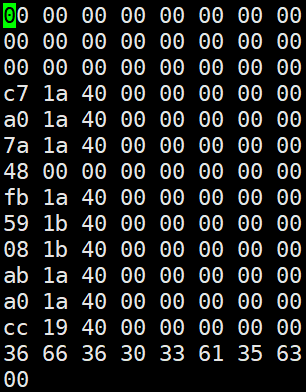
**Picture**\_touch

**④. 生成r5\_raw.txt**



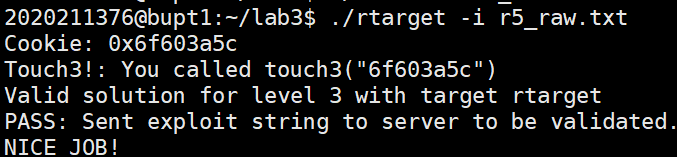
**Picture**\_生成\_s

### 攻击文件



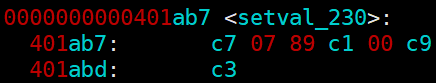
**Picture**\_touch

### 攻击结果



**Picture**\_r5\_attack\_success

### EX：另类方法

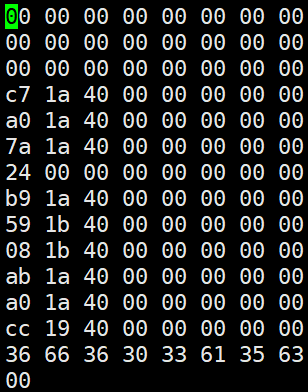


**Picture**\_eca=eax & ecx\*=2

这个指令的功能是：

* **ecx = eax**
* **ecx = ecx\*2**

所以**把偏移量修改为原来的1/2并且使用该指令**也可以过！（为**0x24**）



**Picture**\_ex\_touch

（其实我最开始是指令选错**debug**出来的这个方法）

# 五、总结体会

**问题**：

* **小端方式**输入字节；
* 不清楚**缓冲区**的工作方式；
* 不清楚**栈随机化**的特征；
* 不会编写指令（针对rtarget5）。

**实验投入的时间和精力**：

* ctarget1：2h
* ctarget2：2h
* ctarget3：0.5h
* rtarget4：1h
* rtarget5：6h
* 实验报告：6h

**收获**：

* 了解当程序不能很好地保护自己免受**缓冲区溢出**时，攻击者可以利用安全漏洞的不同方式。
* 更好地了解如何编写更安全的程序，以及编译器和操作系统提供的一些功能，使程序不易受到攻击。
* 对**x86-64机器码**的**堆栈和参数传递机制**有更深入的了解。
* 更深入地了解 **x86-64 指令的编码方式**。
* 获得更多使用 **GDB** 和 **OBJDUMP** 等调试工具的经验。

# 六、诚信声明

需要填写如下声明，并在底部给出手写签名的电子版。

在完成本次实验过程中，我曾分别与以下各位同学就以下方面做过交流：

在我提交的程序中，还在对应的位置以注释形式记录了具体的参考内容。

我独立完成了本次实验除以上方面之外的所有工作，包括分析、设计、编码、调试与测试。

我清楚地知道，从以上方面获得的信息在一定程度上降低了实验的难度，可能影响起评分。

我从未使用他人代码，不管是原封不动地复制，还是经过某些等价转换。

我未曾也不会向同一课程（包括此后各届）的同学复制或公开我这份程序的代码，我有义务妥善保管好它们。

我编写这个程序无意于破坏或妨碍任何计算机系统的正常运行。

 我清楚地知道，以上情况均为本课程纪律所禁止，若违反，对应的实验成绩将按照0分计。

（签名）