ICC de bluz

demon1
lena cakep banget - ztz
User Name - Pablu

Reverse Engineering

xor_madness

100

Bombombini Gusini adalah seorang mahasiswa tahun pertama jurusan Teknologi Informasi yang tengah mendalami cryptography dan malware analysis di mata kuliah Peretasan Beretika. Suatu hari, dosen memberikan tugas berupa sebuah binary file bernama xor_madness.bin. Katanya jika ia berhasil mendapatkan "sesuatu" dari binary file tersebut, maka ia akan langsung mendapatkan nilai A. Bantulah ia untuk bisa mendapatkan "sesuatu" tersebut.

author: mojitodev

A file named `xor_madness.bin` was provided. Upon opening it, it appeared to contain only random characters. Given the name “xor_madness,” I suspected XOR encryption and decided to attempt brute forcing using CyberChef: https://gchq.github.io/CyberChef/#recipe=XOR_Brute_Force(1,100,0,'Standard',false,true,false,'')&input=VXp9d1pHUEdVaHpqJ0xxIH0gYUwifSJMdX8ndEx9aidMcSd9dG4.

Sure enough, the flag was revealed using key 13.

Flag: `FindITCTF{iy4_b3n3r_1n1_fl4g_ny4_b4ng}`

ezBird

995

Beat my score and I will give you the flag. Don't cheat!

Hint: My score = 9999999

author: hilmo

We are given a game called `ezBird`. This game is similar to `Flappy Bird` but much easier. The objective is to achieve a score of `9999999` in order to obtain the flag. The game is written using `Unity` and uses `il2cpp` to compile the C# code into `C++`.

We can use `il2cppdumper` to decompile the `ezBird` file. After decompiling, we get several files, one of which is `Assembly-CSharp.dll`. We can open this file using `ilSpy`. Once opened, we can see several classes inside this file.

Since I was too lazy to go through all the steps, the gist of it is that when we open `Assembly-CSharp.dll`, there is a class that handles the 2D collision detection. Every time we collide in the game, our score increases by 1. However, there is also a variable called the anti-cheat score, which decreases by 0.3 every time we hit a collision. Additionally, there are two other variables that are set to one when our score reaches `9999999`, or when the anti-cheat score reaches `-3000000`.

Because I didn't want to go through all the patching work, I simply used `Cheat Engine` to find the address of these functions. After locating it, I directly nopped the instructions that handle the anti-cheat score check. After bypassing this check, I was able to pass through a pipe and the flag appeared on the screen.

Flag: `FindITCTF{EZZZZREVERSemPvSS}`

Miscellaneous

cek-cek

100

Hei, aku baru belajar python. Semoga aku tidak melupakan sesuatu.

author: hilmo

nc ctf.find-it.id 7001

We are given a Python script that opens a file we input, and if we choose not to open a file, it reveals the flag (but hashed using `blake2b`).

```
if __name__ == "__main__":
    with open("/flag.txt", "w") as f:
        f.write(FLAG)
    flag_file = os.open("/flag.txt", os.O_RDONLY)
    flag_data = os.read(flag_file, 1024)
    if FLAG.encode() != flag_data:
        print("flag file is corrupted")
        exit(1)

    while True:
        print("Do you want check my file?")
        print("1. yes")
        print("2. no")

        choice = input(">>> ")
        if choice == "1":
            file_name = input("file name: ")
            print(open_file(file_name))
        elif choice == "2":
            print("ok, here the flag:")
            print(flag)
        else:
            print("invalid choice")
```

Since the file `/flag.txt` is opened and read but never closed, we can exploit this to retrieve the flag. By using `os.open`, we can access the file through `/proc/self/fd`, which is a symbolic link to the file descriptors currently opened by the process. Then, using `os.read`, we can read the contents of the open file descriptor.

By inputting `/proc/self/fd/5` as the filename, we can successfully read and obtain the flag.

Flag: `FindITCTF{cl0s3_y0ur_f1l3s_1mmed14t3ly_0r_w0w0_w1ll_f1nd_y0u}`

your-journey-2

100

perjalananmu berlanjut, tahun lalu masih adem sekarang tidak. kalau berhenti sekarang ntar malah ga sampe sampe, mending gas aja terus

author: hilmo

nc ctf.find-it.id 7101

We are given two Python files, `main.py` and `word.py`. The `word.py` file contains a blacklist of words that must not appear in the input. The `main.py` file contains the code that will execute our input. If we enter the correct input, we will get the flag.

```
import re

from hidden import *
from word import *

while True:
    ans = (
        input(
            f'{lagu}\nHmm something seems wrong with the song, isn't it supposed to
be "Ayo Ayo Ganyang si b.e.b.a.n 🌸"\n$'
        )
        .strip()
        .lower()
    )

    if any(char in ans for char in block):
        print(
                f'\nUnfortunately, you used forbidden words. Now the "official" has
been promoted\n'
        )
        break
    if not re.match("^[\x20-\x7E]*$", ans):
        print("\nOh no, you're trying to use non-alphabet characters :>\n")
        break
    try:
        eval(ans + "()")
        print("Is this the right end?\n")
    except Exception as e:
        print(e)
           print(f'\n{ascii2}\nOh no, you were attacked by "The colleagues of the
official"\n')
        break
```

Since the blacklist is not too extensive, we can try each word one by one. First, I am curious about the content of `hidden.py`, so I immediately use the `help` function to check the contents of `hidden.py`. It turns out that the module contains a function to read folders.

```
help> hidden
Help on module hidden:

NAME
    hidden

FUNCTIONS
    viewfolder(path: str)

DATA
    FLAG = 'FIndITCTF{y0u_f0und_1t!_or_d1d_y0u?}'

FILE
    /challenge/hidden.py
```

We can attempt to read the folder in the current directory by calling the `viewfolder` function.

```
$viewfolder('.')
endingsatu
flag.txt
main.py
hidden.py
word.py
endingtiga
endingdua
```

We see that there are many folders inside, as well as several fake flags. So, I decided to try each one and eventually found the real flag in the `endingdua` folder by executing the following code:

```
print(open('endingdua/flag.txt').read())
```

Flag: `FindITCTF{k0n0h4_m4ju_m4sy4r4k4t_m4kmur}`

distorted

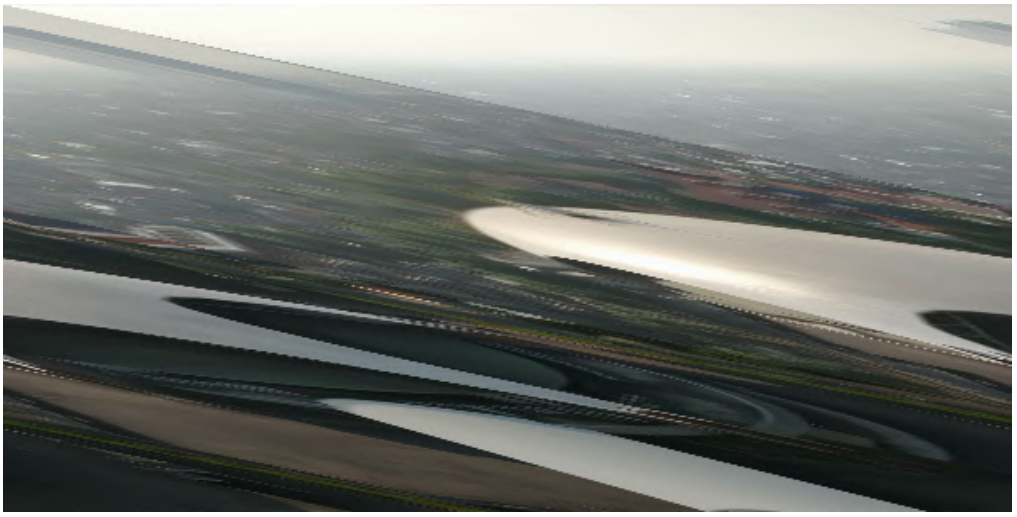
100

GAMBARNYA MLEYOTT. Setiap row bergeser 5 pixels lebih dari row sebelumnya. Gimana nih biar gambarnya kelihatan dan lokasinya bisa dicari?

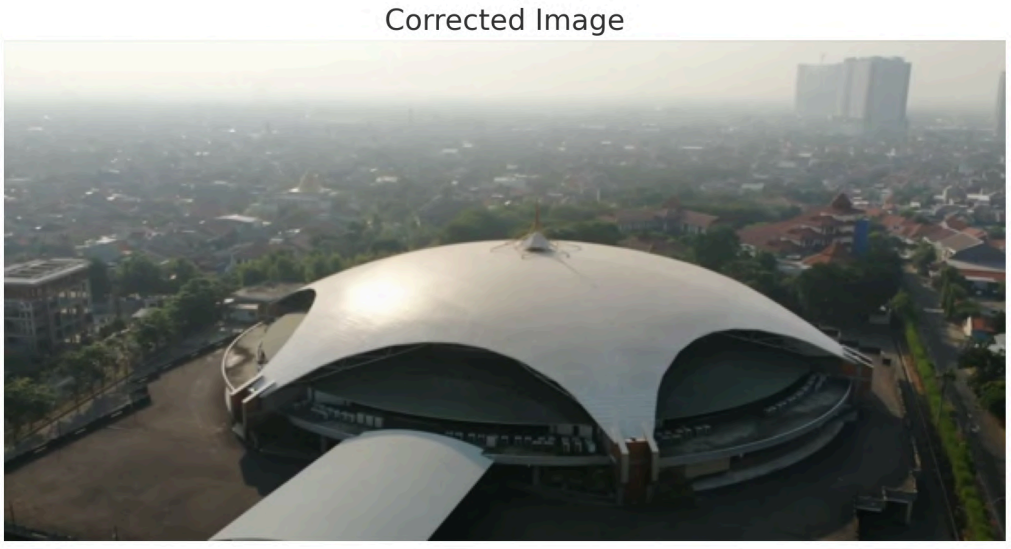
- Format Flag: FindITCTF{Lintang_Bujur_Nama_Tempat}
- case insensitive

author: Azmi
attachment: location.png

We are given location.png file.



I tried to ask chatgpt for help to restore the row swipe as the challenge description said. and it worked.

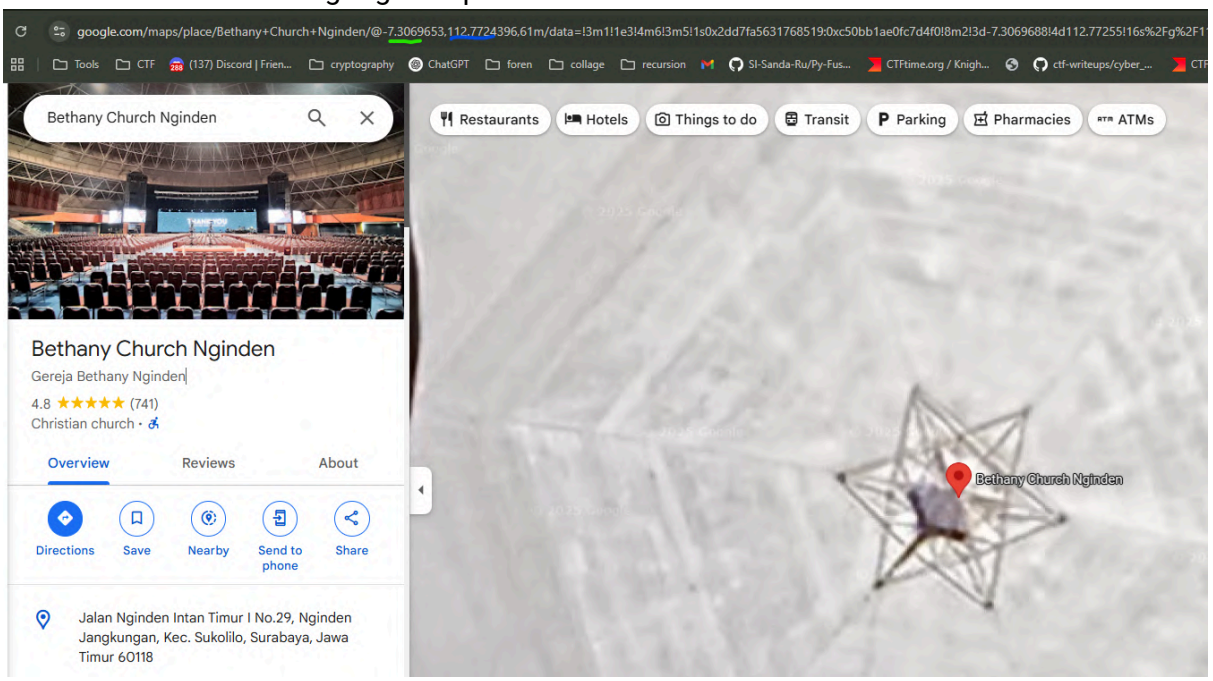


Use Google Lens, and we will know this is “Gereja Bethany Nginden”

▼ View Hint

(4 angka di belakang desimal / .231245 = .2312) (Nama Lokasi Ikutin Format Google Maps)

based on the hint, we take the coordinates up to 4 numbers behind the comma and the location name follows in google maps.



Flag: FindITCTF{-7.3069_112.7725_Gereja_Bethany_Nginden}

Forensics

Oversharing

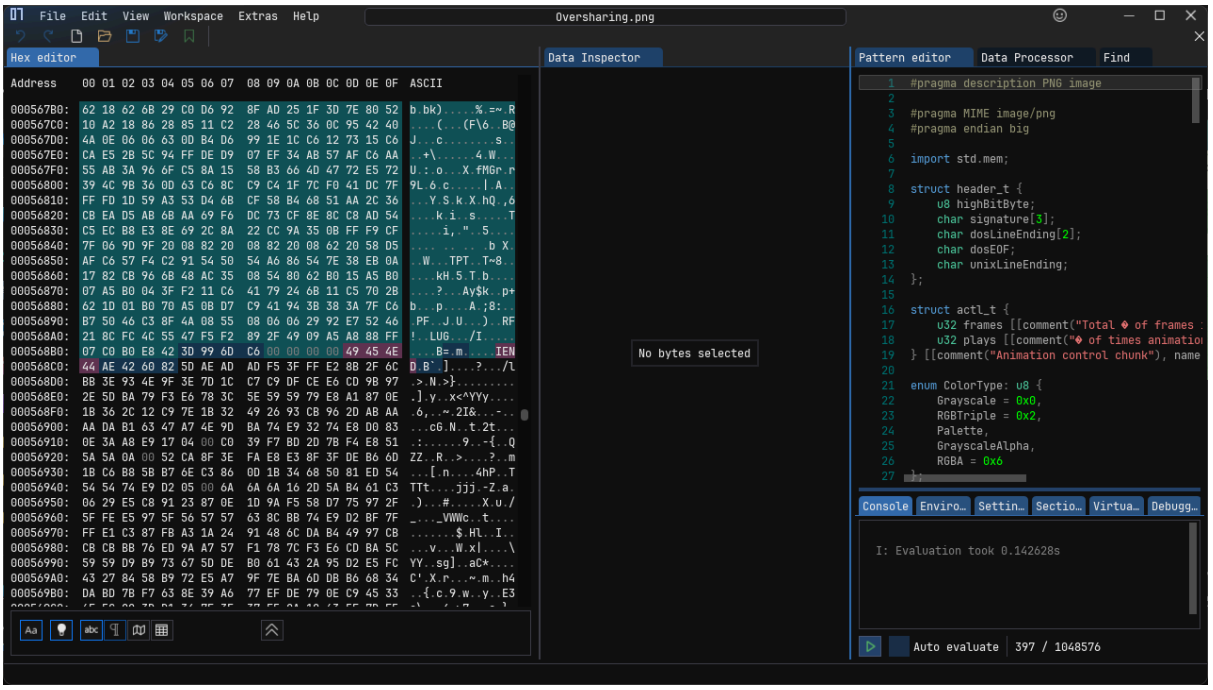
775

Yo man wassup,

I am so excited, finally passed my probation. Just got assigned to this new high impact project. The IT guy just gave me my account for the project, check out the chat! Can't wait to login and show my worth :D

author: BerlianGabriel

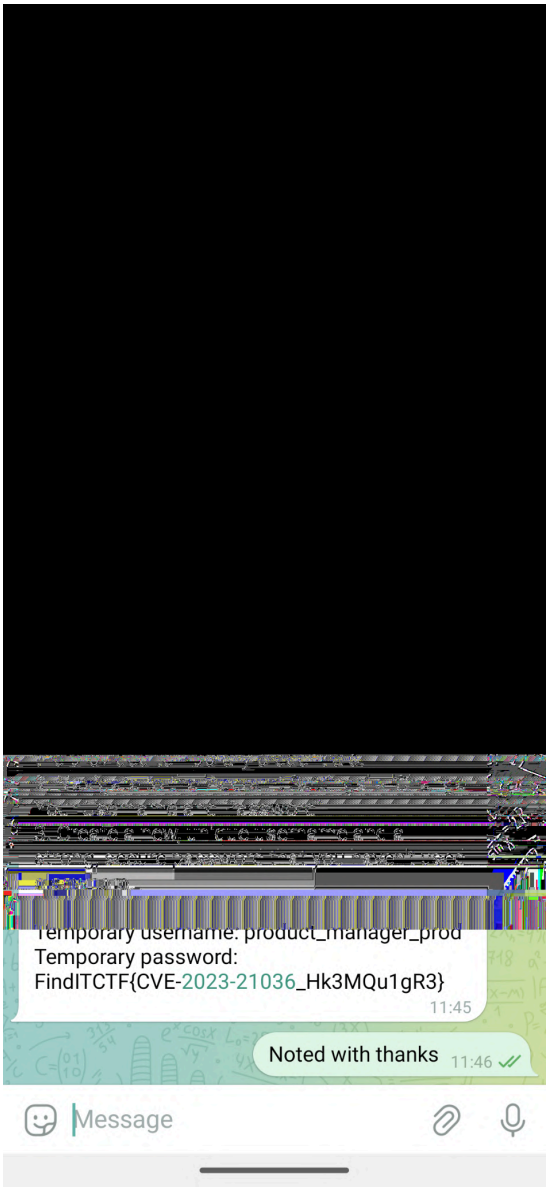
We are given a file called `Oversharing.png`, which contains a chat image between the sender and the receiver. When I examined the bytes of the image, I found something interesting: there are two `IEND` markers within the byte sequence of the image.



After further investigation, I discovered that this PNG image had been modified. I tried removing the second `IEND` and saved it as a new PNG file, but nothing happened.

Because the image appeared to be cropped, I decided to search Google using the keyword `Image cropped writeup`. I found a writeup that explained how to restore a cropped PNG image, which was caused by a bug known as `aCropalypse`. This bug existed in a screenshot application on Android. The bug allowed users to edit images and save them, but the saved image did not fully erase the original parts of the image. In other words, the edited image still contained portions of the original image that were supposed to be removed.

So, we can use the website [acropalypse] to restore the cropped image. Select `Pixel 5` and upload the image to the website. Once processed, we will get the full image that had been cropped.



Flag: `FindITCTF{CVE-2023-21036_Hk3MQu1gR3}`

new-waifu

957

pecinta waifu ternyata kurang puas sama stealer dia sebelumnya, sekarang dia bikin info stealer baru

jangan lupa FindITCTF{}

author: hilmo

We were given a file `new-waifu.exe`, and it turns out that this is an information stealer. When executed, the file sends some information to `waifu.find-it.id`. But since we can reverse engineer it, let's just go ahead and reverse engineer it (I'm not an expert in forensics anyway).

```
v4 = main_decrypt(
       (unsigned int)"HRcWVuQC/n7mnDoEu5+sl/G9E31MdHGhO+gCONit0qksf9KKwOEW8wI1g6fWxhVG6AM6a5SYmODiGLYTkNZnL7zxR4Ql5INkw"
                     "TD6sQdFeEsp+pZi0RaR1r57skYupRNr0oJ2+GNcfcseMCKDZmmDcizFc6iCpVIiXdEmLLH+74rINJTux/elD76eL4347zSBWX"
                     "xDI1HmMRThFh4FHag598ojf4aurIIiyv5nLooji7xxqI0hq6hcf5fkzuChg0r3E23x/ZWMlvEGp8W4SDahq5Hj3egzH/XGlbb"
                     "JFnNcv6/yXlCGc74JblPrXux8xA==",
       320,
       (unsigned int)"ShimpaziniBananiwaifu.find-it.id0123456789abcdefTerminateProcessexec: no command2384185791015625012345678 9ABCDEFinvalid exchangeno route
       16,
       v2,
       v6,
       v7,
       v8);
```

This image shows the parameters used to call the `main_decrypt` function.

```
61  v22 = crypto_aes_NewCipher(v54, v49, v50, 0, a5, v17, v18, v19, v20);
62  if ( v27 )
63    return 0LL;
64  v28 = crypto_cipher_NewGCM(v22, v21, 0, 0, a5, v23, v24, v25, v26);
65  if ( v29 )
66    return 0LL;
```

In the `main_decrypt` function, we can see that **AES GCM** is used for decrypting the data. We also see that there is a `key` sourced from `key_source` and `ciphertext` from `base64_ciphertext`.

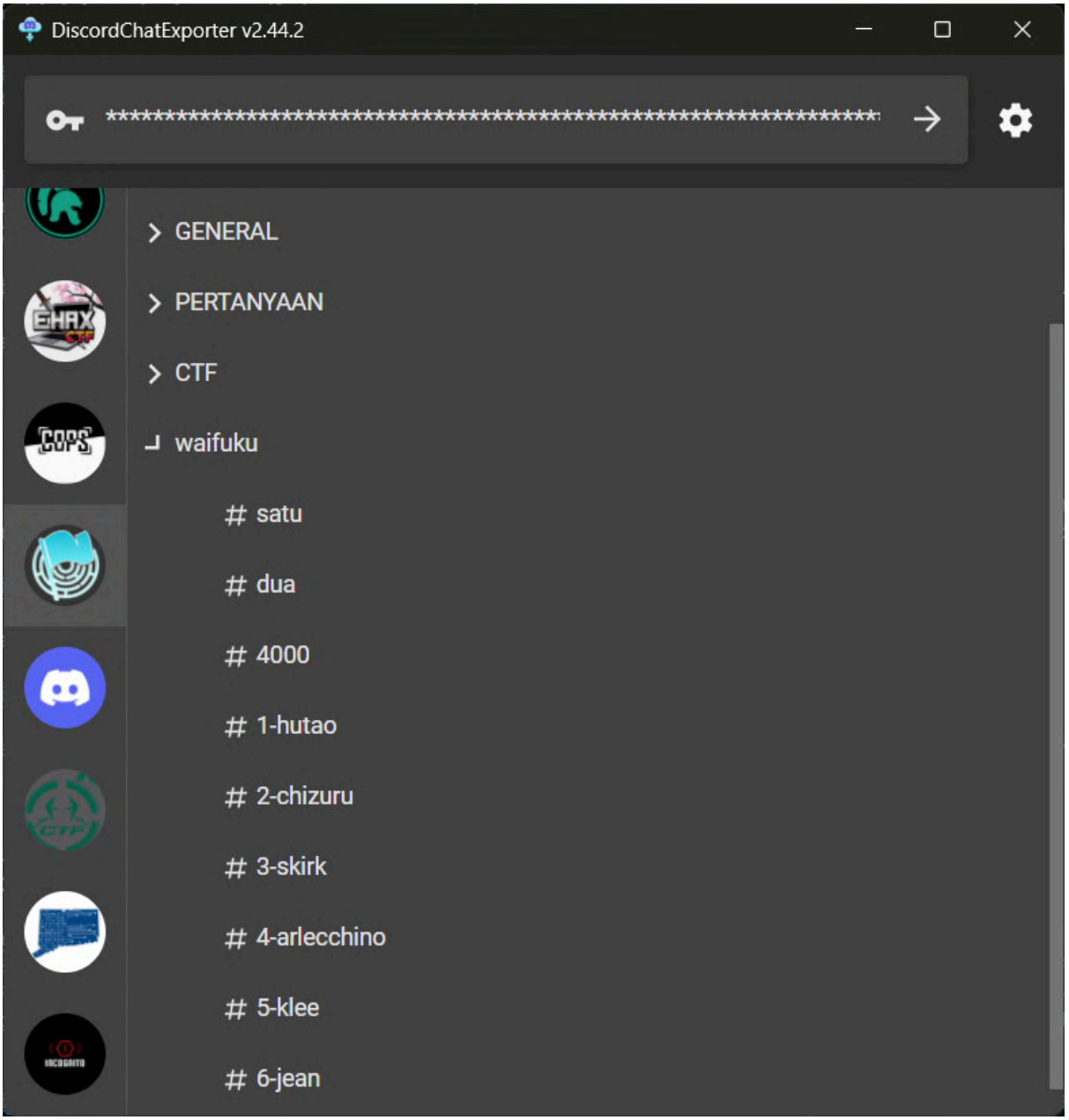
```
import base64
from Crypto.Cipher import AES

base64_ciphertext = (

"6kGrsTkEwA0bU526wlqbjD98CkEP7jqAw3oEm2oTHS9wnXWer25jeDmNnCq088+mKXXN36QcQkFGL7xp8h
PqvV66JZkRinvWFW/pZ"

"Tybuo5NR9MsKLl5LKQ+APRU5e10h41D2y607Z90x0yzHkuCeuG4vzBYiTwzNv0YvXAEkn8VUEklZ3ngsg9
T6UZm4HvJdJUiPkzg8p"

"VBJNaCmgSdkYCdzuhBPDhnbGo+xqQAvsl97hjlNFNKTqazjdLJgJLwl+juzLQwzBUiW9Fi55aGmIbrO2SD
FTZSSlJNhNXqcTlA"
```

```
)
key_source = (
        "ShimpaziniBananiwaifu.find-it.id0123456789abcdefTerminateProcessexec: no
command23841857910156250123456789ABCDEFinvalid exchangeno route to hostinvalid
argumentmessage too longobject is remoteremote I/O
errorSetFilePointerExOpenProcessTokenRegQueryInfoKeyWRegQueryValueExWDnsNameCompare
_WCreateDirectoryWFlushFileBuffersGetComputerNameWGetFullPathNameWGetLongPathNameWR
emoveDirectoryWNetApiBufferFreeDuplicateTokenExGetCurrentThreadGetModuleHandleWRtlV
irtualUnwindinteger overflowgcshrinkstackofftracefpunwindoffGC scavenge waitGC
worker (idle)page trace flush/gc/gogc:percent, not a functiongc: unswept span KiB
work (bg), mheap.sweepgen=runtime: nelems=workbuf is
emptymSpanList.removemSpanList.insertbad special kindbad summary dataruntime: addr
= runtime: base = runtime: head = timeBeginPeriod"
)

key = key_source[:16].encode("utf-8")
data = base64.b64decode(base64_ciphertext)

nonce_size = 12
nonce = data[:nonce_size]
ciphertext_and_tag = data[nonce_size:]

try:
    cipher = AES.new(key, AES.MODE_GCM, nonce=nonce)
            plaintext = cipher.decrypt_and_verify(ciphertext_and_tag[:-16],
ciphertext_and_tag[-16:])
    print("Decrypted plaintext:", plaintext.decode("utf-8"))
except Exception as e:
    print("Decryption failed:", str(e))
```

Decrypting the `base64_ciphertext` using the `key_source` we get the following plaintext:

All waifu names except the ones that are minors in Discord CTF FindIt 2025, in reverse order, spaces replaced with underscores. Who even thought of using Golang with native Windows APIs, haha, idea came at midnight :D

Okay, now we just need to use `DiscordChatExporter` to get all the waifu names from the Discord channel.



From the exported chat, we get the following waifu names:

```
1. hutao
2. chizuru
3. skirk
4. arlecchino
5. klee
6. jean
```

Since `klee` is considered a minor, we exclude it and the remaining names in reverse order are `jean_arlecchino_skirk_chizuru_hutao`.

Flag: `FindITCTF{jean_arlecchino_skirk_chizuru_hutao}`

waifuku

964

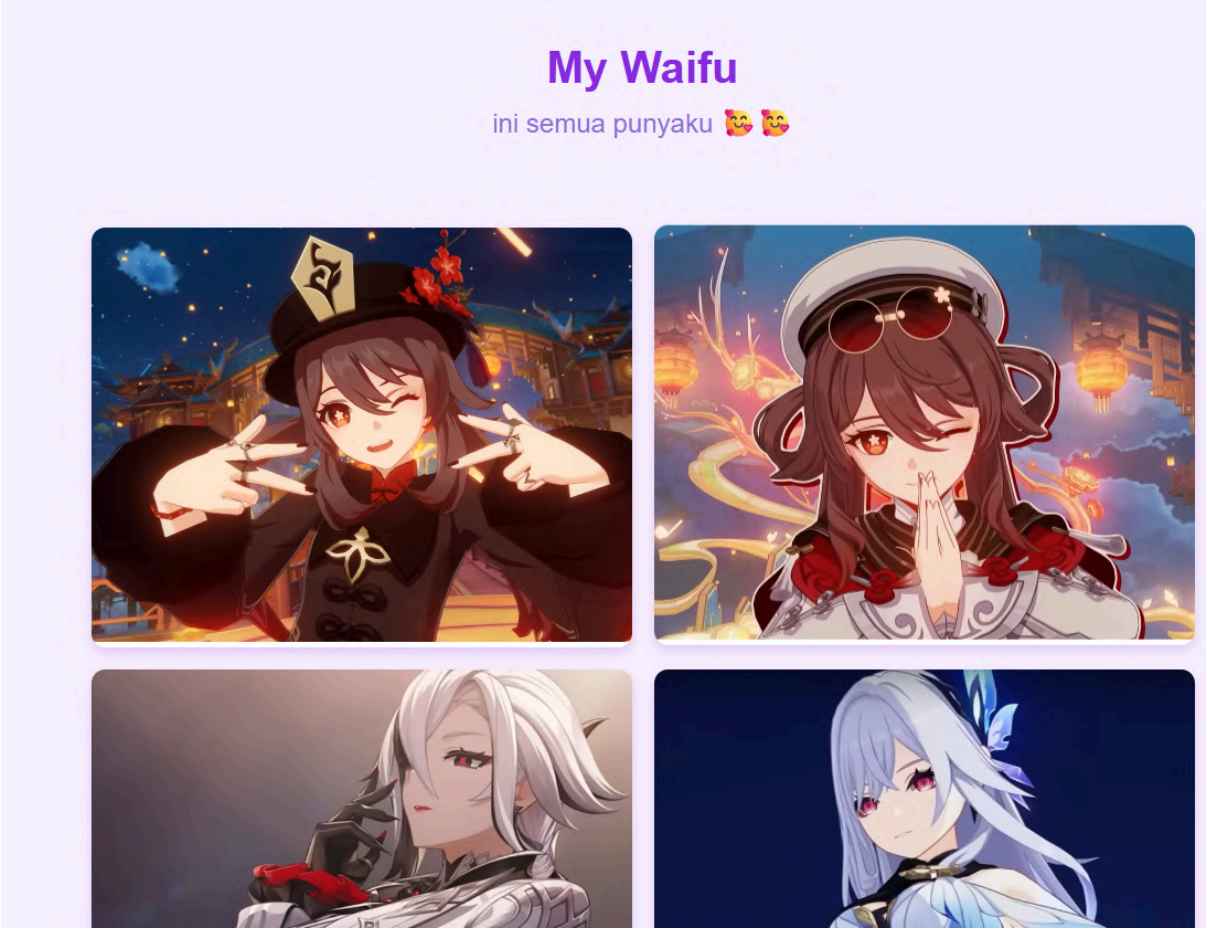
pecinta waifu ternyata seorang info stealer? hahh 🥶🥶 bongkar semua kedoknya dia

format flag: FindITCTF

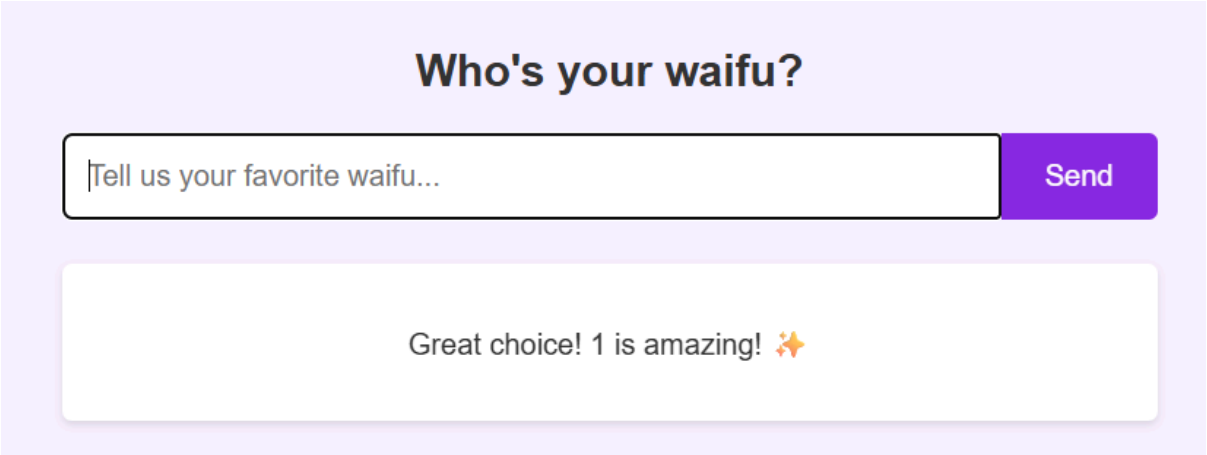
author: hilmo

http://ctf.find-it.id:7201

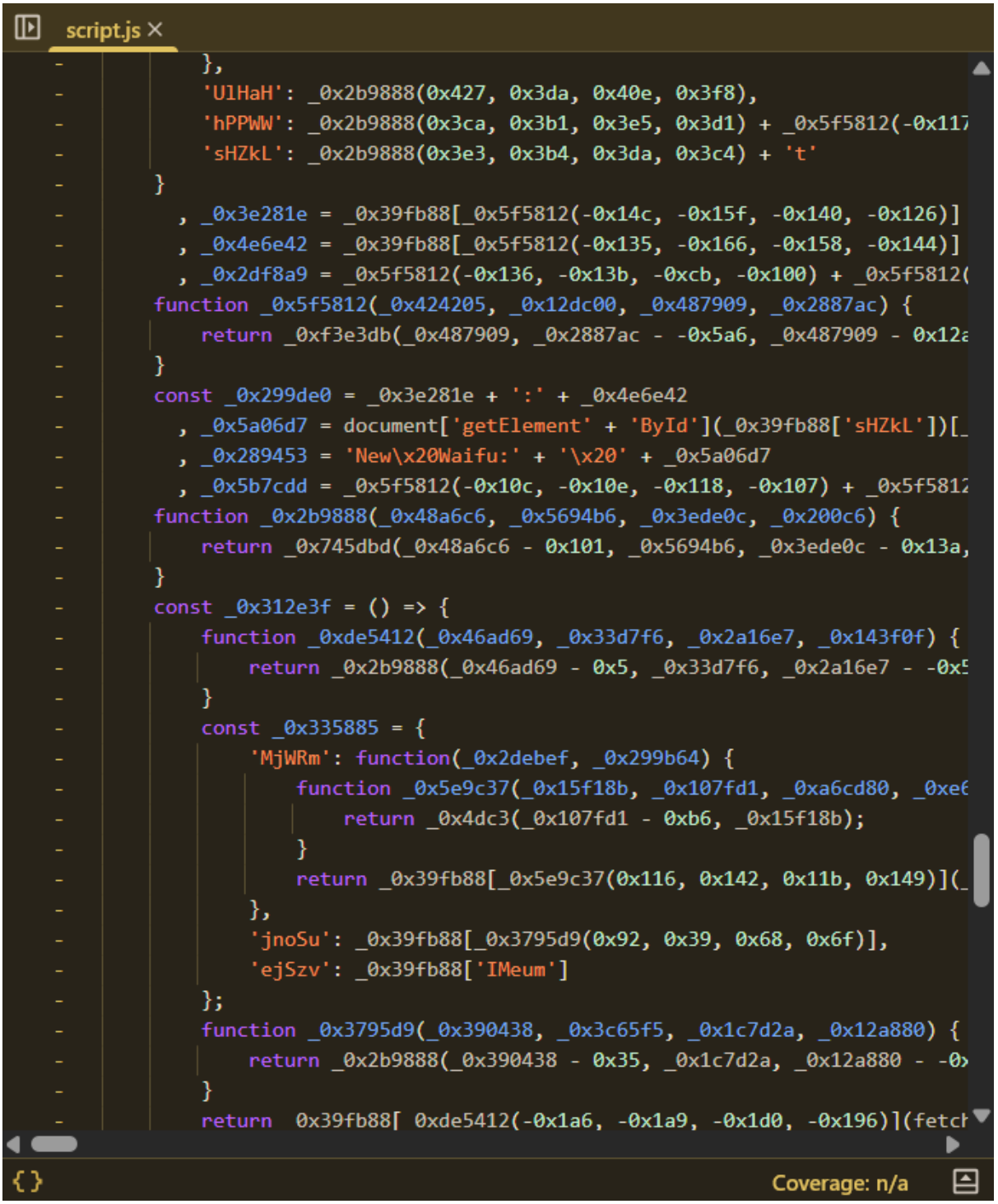
Forensics but we got a website???



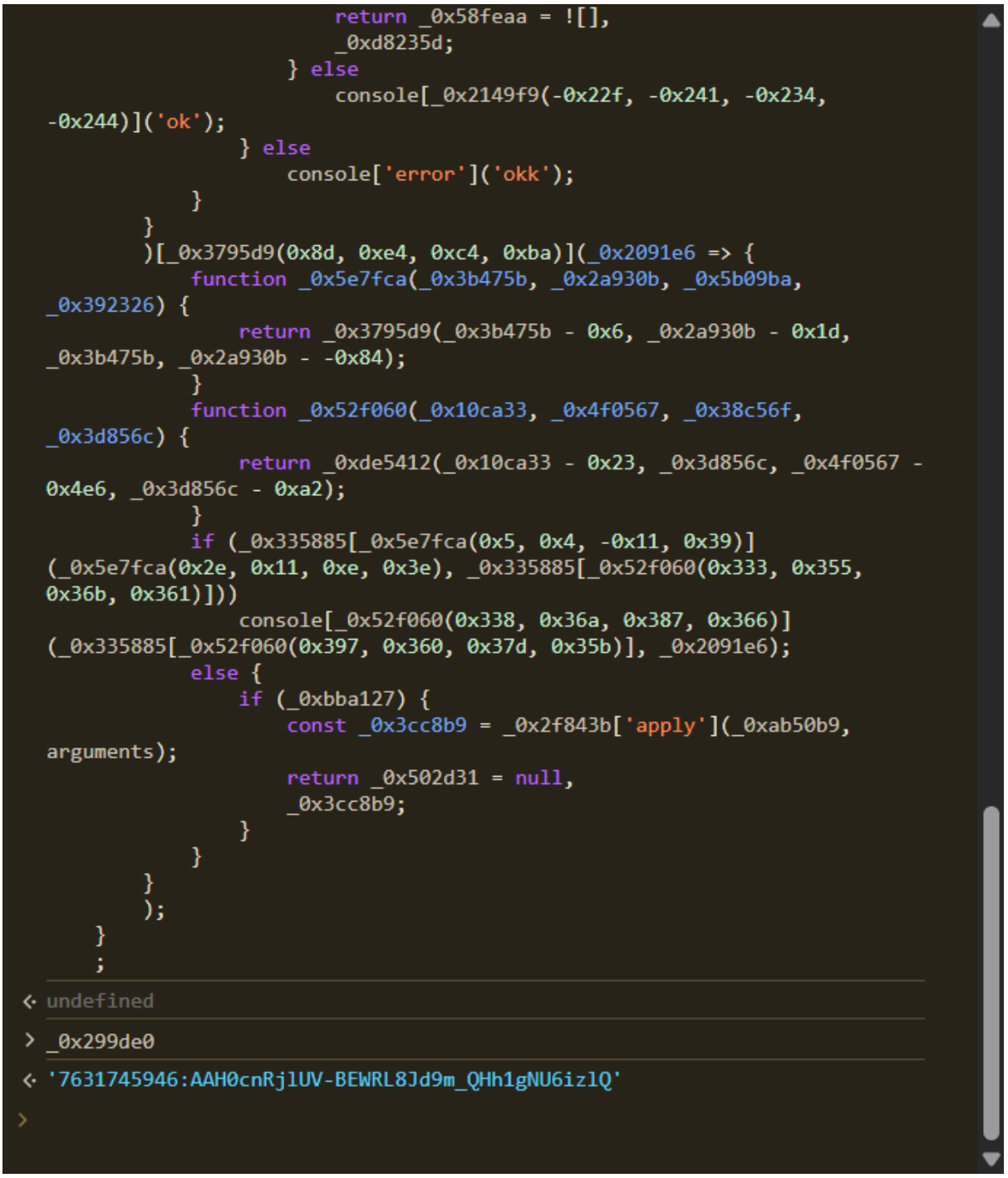
There is a form to submit our waifu. Let's try submitting one. After submission, a message pops up.



So, this likely involves some JavaScript code that runs on the page and possibly makes a request to the backend server. However, the message looks like regular JavaScript execution, so we take a closer look at the source code.



Since the JavaScript code is obfuscated, I decided to inspect the function and variable contents directly from the browser console.



```
                        return _0x58feaa = ![],
                        _0xd8235d;
                    } else
                        console[_0x2149f9(-0x22f, -0x241, -0x234,
-0x244)]('ok');
                } else
                    console['error']('okk');
            }
        }
        )[_0x3795d9(0x8d, 0xe4, 0xc4, 0xba)](_0x2091e6 => {
            function _0x5e7fca(_0x3b475b, _0x2a930b, _0x5b09ba,
_0x392326) {
                return _0x3795d9(_0x3b475b - 0x6, _0x2a930b - 0x1d,
_0x3b475b, _0x2a930b - -0x84);
            }
            function _0x52f060(_0x10ca33, _0x4f0567, _0x38c56f,
_0x3d856c) {
                return _0xde5412(_0x10ca33 - 0x23, _0x3d856c, _0x4f0567 -
0x4e6, _0x3d856c - 0xa2);
            }
            if (_0x335885[_0x5e7fca(0x5, 0x4, -0x11, 0x39)]
(_0x5e7fca(0x2e, 0x11, 0xe, 0x3e), _0x335885[_0x52f060(0x333, 0x355,
0x36b, 0x361)]))
                console[_0x52f060(0x338, 0x36a, 0x387, 0x366)]
(_0x335885[_0x52f060(0x397, 0x360, 0x37d, 0x35b)], _0x2091e6);
            else {
                if (_0xbba127) {
                    const _0x3cc8b9 = _0x2f843b['apply'](_0xab50b9,
arguments);
                    return _0x502d31 = null,
                    _0x3cc8b9;
                }
            }
        }
        );
    }
    ;
undefined
> _0x299de0
'7631745946:AAH0cnRjlUV-BEWRL8Jd9m_QHh1gNU6izlQ'
>
```

By copying the content of the `simpasini()` function and inspecting the `_0x299de0` variable, I discovered what looks like a Telegram Bot token: `7631745946:AAH0cnRjlUV-BEWRL8Jd9m_QHh1gNU6izlQ`.

This appears to be a valid bot token. Let's try checking the bot's update history using Telegram's API:

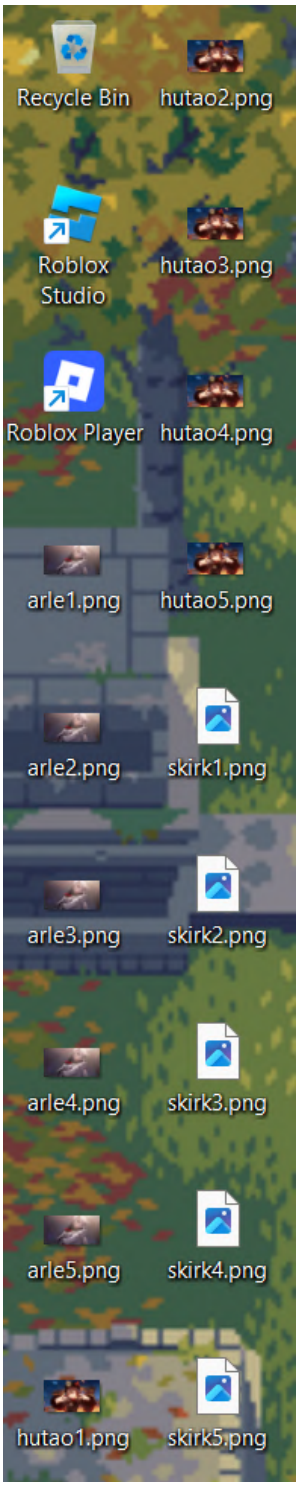
```
curl -X GET "https://api.telegram.org/bot7631745946:AAH0cnRjlUV-BEWRL8Jd9m_QHh1gNU6izlQ/getUpdates"
```

At the time of writing this, unfortunately, the updates are already gone—perhaps this method was unintended?. However, earlier during the event, I was able to view some Telegram updates. The messages contained files uploaded to the bot and a few strings in Unicode format, which can be decoded into readable text.

After trying to download some of the uploaded files, I found one file that contained the flag.

```
curl https://api.telegram.org/file/bot7631745946:AAH0cnRjlUV-BEWRL8Jd9m_QHh1gNU6izlQ/documents/file_1.exe -o file_1.exe
```

Before I tried to use `strings` I first ran the binary file, but what happened was that it was trolled by the problem setter.



Running the `strings` command on `file_1.exe` gives us some interesting strings. One of them is `FindITCTF`, which is exactly what we are looking for.

```
strings file_1.exe | grep "FindIT"
```

```
coded with illegal parameterschacha20poly1305: bad nonce length passed to Sealchacha20poly1305: bad nonce length passed to Opencrypto/rsa: public exponent too small or negativeFindITCTF{s
3m4ngHt_PuG1_____h1dup_N4Rut0000}go package net: dynamic selection of DNS resolver
```

Flag: `FindITCTF<s3m4ng4t_P4G1__________h1dup_N4Rut00000>`

Web Exploitation

PixelPlaza

655

I'm a consultant, but my client is using a new technology I'm not familiar with. Can I outsource this whitebox pentest project to you?

author: BerlianGabriel

http://ctf.find-it.id:6001

We are given a web application built using Go, and we also have access to its source code. Upon examining the source code, we can see that the application uses `embed` to store static files within the binary. The code reveals that several files are stored in the `public` folder, including an `index.html` file, which is displayed when we access the root path `/`.

There are also some interesting handlers, such as `/` and `/static/`. When we access `/static/`, we receive static files from the `public` folder, while accessing `/` serves the `index.html` file from the same folder.

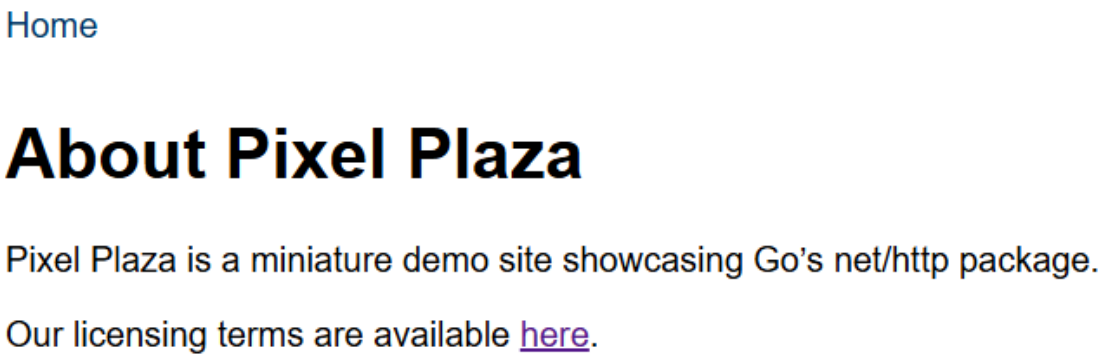
```
fileServer := http.FileServer(http.FS(webFS))
mux.Handle("/static/", http.StripPrefix("/static/", fileServer))
mux.HandleFunc("/", staticHandler)
```

What is particularly interesting is that the handler for the root path `/` is `staticHandler`, which serves the `index.html` file if we access the root path `/`, and serves static files from the directory inside the binary if we access other paths.

```
func staticHandler(w http.ResponseWriter, r *http.Request) {
    if r.URL.Path == "/" {
        data, _ := webFS.ReadFile("public/index.html")
        w.Write(data)
        return
    }
    p := "." + r.URL.Path
    if _, err := os.Stat(p); err != nil {
        io.WriteString(w, "Resource not found.")
        return
    }
    f, err := os.Open(p)
    if err != nil {
        http.NotFound(w, r)
        return
    }
    defer f.Close()
    fi, err := f.Stat()
    if err != nil {
        http.NotFound(w, r)
```

```
        return
    }
    http.ServeContent(w, r, filepath.Base(p), fi.ModTime(), f)
}
```

However, we can also access files outside the binary folder using path traversal. For example, we can try to access the `banner.png` file located in the `docs` folder by using the path `/..%2fdocs/banner.png`. But when I attempted to find `flag.txt` in several typical paths, I couldn't locate it.



After revisiting the website, I noticed there was a link in the "About" section for viewing the license. However, this link leads to `/docs/text` and shows a message indicating that the resource was not found.

We can try path traversal again to access `/..%2fdocs/text` by navigating to `http://ctf.find-it.id:6001/..%2f/docs/text`, and this reveals the contents of the `text` file, which contains the flag we are looking for.

Flag: `FindITCTF{g0L4nG_4lL0wS_p4th_Tr4V3rs4L???}`

Simple Heist

100

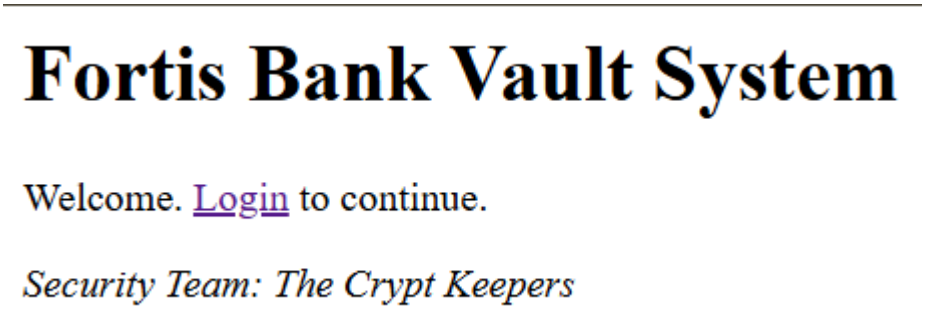
gampang sekali, tinggal cari kunci dari brankasnya

cuma internal yang boleh tau banyak hal

author: hilmios

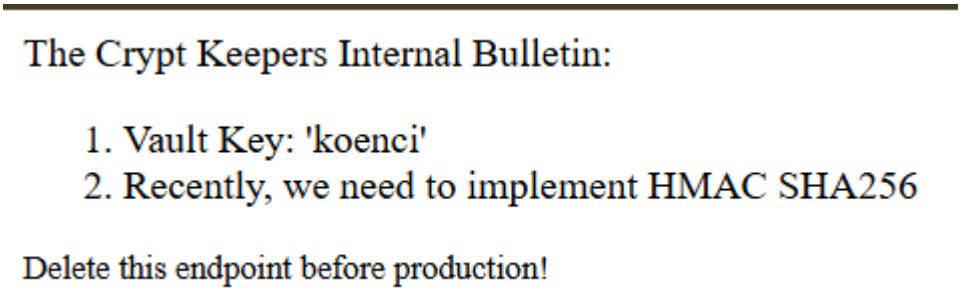
http://ctf.find-it.id:10001

When we visit the homepage, we will see a screen like this:

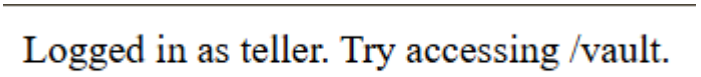


Here, we can log in, but we will immediately be logged in as the user `teller` from `Fortis Bank`.

Since the description includes the keyword `internal`, we try to access `/internal`, and we get a screen like this:



When we check the cookies, we find two cookies: `auth` and `sig`. The `auth` cookie contains `user:teller|bank:Fortis Bank`, and the `sig` cookie contains the **HMAC SHA256** of the `auth` value.



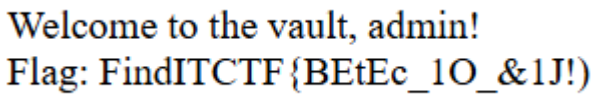
On the `/login` page, we are instructed to access `/vault`, but we cannot access it because we are not `admin` and the bank we are using is `Fortis Bank`.

We try to modify the `auth` cookie to `user:admin|bank:Fortis Bank` and then update the `sig` cookie to be the **HMAC SHA256** of the new `auth` value.

Here's how we can create **HMAC SHA256** using CyberChef.

```
auth: "user:admin|bank:Fortis Bank"
sig: 7f5976dcdc018b18b360aad2d4c5b3efe099db2bbba363bad5c1932b137f41ba
```

After obtaining the new `auth` and `sig` cookies, we try to access `/vault` and will see a screen like this:



Flag: `FindITCTF{BEtEc_1O_&1J!)`

Crypto

Weak

930

Simple login. By the way, I think using a common secret is a bad idea 🗿

author: hilmo

nc ctf.find-it.id 7301

Given a connection and the source code of the file `source.py`. Let's look at the source code.

```python
def register(name):
    token = pce(name)

    data = {
        "name": name,
        "user_id": random.randint(1, 100),
        "token": token,
    }

    cookie = jwt.encode(data, secret, algorithm="HS256")
    print("Store this cookie for login:", cookie)
```

This function allows a user to register. It creates a user token by calling the `pce()` function, and generates a JWT (JSON Web Token) using a shared secret. The token is then returned to the user as a "cookie" used for future authentication.

```python
def pce(str):
    iv = get_random_bytes(16)

    spce = (
                                    f"name={str}_{prefix     *     random.randint(1,
100)};uid={random.randint(1,10000000)}"
    )
    bpce = bytes(spce, "utf-8")
    p = pad(bpce, 16)
    c = AES.new(secret2, AES.MODE_CBC, iv)
    e = c.encrypt(p)

    return f"{e.hex()}+{iv.hex()}+{rand.hex()}"


def pce_decrypt(enc):
    e = bytes.fromhex(enc[0])
    iv = bytes.fromhex(enc[1])
```

```
    c = AES.new(secret2, AES.MODE_CBC, iv)
    d = c.decrypt(e)

    return unpad(d, AES.block_size).decode("utf-8")
```

The `pce()` function is responsible for creating the token's encrypted part using AES in CBC mode. The structure of the plaintext being encrypted includes the username and a UID. It returns the ciphertext, IV, and a static random value as a token string. The `pce_decrypt()` function does the reverse—it decrypts this token back into plaintext using the same key and IV.

```
def login(name, cookie):
    decoded = jwt.decode(cookie, secret, algorithms=["HS256"])

    if decoded["name"] != name:
        print("Whoops! This cookie is not for you.")
        return

    if decoded["name"] == "admin":
        print(pce_decrypt(decoded["token"].split("+")))
        if (
            decoded["name"]
            == pce_decrypt(decoded["token"].split("+")).split(";")[0].split("=")[1]
            and rand.hex() == decoded["token"].split("+")[2]
        ):
            print("GG, here your flag: ", FLAG)
            return
        else:
            print("Whoops! This cookie is not for you.")

    print("Welcome back, " + decoded["name"] + "!")
```

In the login function, the server first decodes the JWT and checks that the username in the cookie matches the submitted name. If the username is `admin`, it decrypts the token and performs two checks:

- The decrypted name inside the token must also be `"admin"`.
- The static random value at the end of the token must match the global `rand.hex()` value.

If both pass, the flag is printed. Otherwise, the access is denied.

To exploit this vulnerability, we start by registering a new user, such as `adm1n`, and capture the JWT token generated. We then extract the AES IV and ciphertext from the token.

Due to the use of AES in CBC mode, we can apply a **bit-flipping attack** to the IV in order to manipulate the first block of decrypted plaintext. Our goal is to modify the decrypted string

so that the `"name=adm1n_"` part becomes `"name=admin;"`. We don't need to decrypt the ciphertext—just control what it decrypts to by flipping bits in the IV.

After adjusting the IV, we recombine the ciphertext, modified IV, and original random value, and construct a new JWT. We also change the `name` field in the JWT payload to `"admin"`. This tricks the server into thinking the token belongs to `admin` and passes both verification checks.

Since the JWT is signed using a common secret key (`secret`) and the algorithm is `HS256`, it's susceptible to **brute-force attacks** if the secret is weak or guessable. Tools like `jwt_tool.py` or `John the Ripper` can try to brute-force the JWT signature using wordlists such as `rockyou.txt`.

```
import json

import jwt
from pwn import *


def bit_flipping(ciphertext, position, xor_value):
    """
       Modifies the encrypted text by XORing a specified value at a specified
position.

    :param ciphertext: (bytes) The input ciphertext.
      :param position: (int) The position in the ciphertext where XOR should be
applied.
    :param xor_value: (int) The value to XOR at the specified position.
    :return: (bytes) The modified ciphertext.
     :raises ValueError: If the position or xor_value is not of type 'int' or if the
position is invalid.
    :raises Exception: If an error occurs during bit flipping.
    """
    try:
        ciphertext = bytes.fromhex(ciphertext)
        byte_list = bytearray(ciphertext)
        if not isinstance(position, int) or not isinstance(xor_value, int):
            raise ValueError('Position and xor_value must be integers')

        # print(byte_list[16:])
        # if position < 0 or position >= len(byte_list[16:]):
        #     raise ValueError("Invalid position")
        byte_list[position] ^= xor_value
        return bytes(byte_list)
    except Exception as e:
        print(f'Bit flipping error: {e}')
        raise


# for i in range(128):
#     c = AES.new(b"1234567890123456", AES.MODE_CBC, bit_flipping(
```

```
#         '01d1b8ca22ef2b83e9def93a8d83c3a9',
#         8,
#         i))
#                                                                 d             =
c.decrypt(bytes.fromhex('b8916ad7004d32356f34639d85861830ab1d16dbd4a74e6f25c372dc60
21f4ef'))
#     # if d[10] == 'g':
#     print(f'{i}: {d}')

# context.log_level = 'debug'

r = remote('ctf.find-it.id', 7301)

r.sendlineafter(b'Enter your choice (1/2/3):', b'1')
r.sendlineafter(b'Enter your name:', b'adm1n')

r.recvuntil(b'Store this cookie for login:')
cookie = r.recvline().strip()
log.info(f'Cookie: {cookie}')

payload = base64.b64decode(cookie.split(b'.')[1] + b'==').decode()
payload = json.loads(payload)

user_id = payload['user_id']
token = payload['token']
iv = token.split('+')[1]

bit_flip = bit_flipping(iv, 8, 88).hex()
bit_flip = bit_flipping(bit_flip, 10, 100).hex()

token = (
    payload['token'].split('+')[0]
    + '+'
    + bit_flip
    + '+'
    + payload['token'].split('+')[2]
)

payload = {'user_id': user_id, 'name': 'admin', 'token': token}

cookie = jwt.encode(payload, b'internet', algorithm='HS256')

r.sendlineafter(b'Enter your choice (1/2/3):', b'2')
r.sendlineafter(b'Enter your name:', b'admin')
r.sendlineafter(b'Enter your cookie:', cookie.encode())

r.interactive()
```

Flag: `FindITCTF{W1_w0k_d3_t0k_n0t_0nl1_t0k_d3_t0k}`

spacemonkey

971

Our space monkey ceaser has gone missing in space, before he went missing he sent us two message.

author: Timothy Tanuwidjaya

release.7z

After unzipping `release.7z`, we get two files: `code.txt` and `cipher.txt`. At first, I assumed `code.txt` contained Morse code. After decrypting it, I got a readable message.

Based on that message, I suspected the text in `cipher.txt` was encrypted using the Playfair cipher, with the keyword `FOREVER`. So I tried decrypting it, but the output wasn't the flag, and I got stuck there.

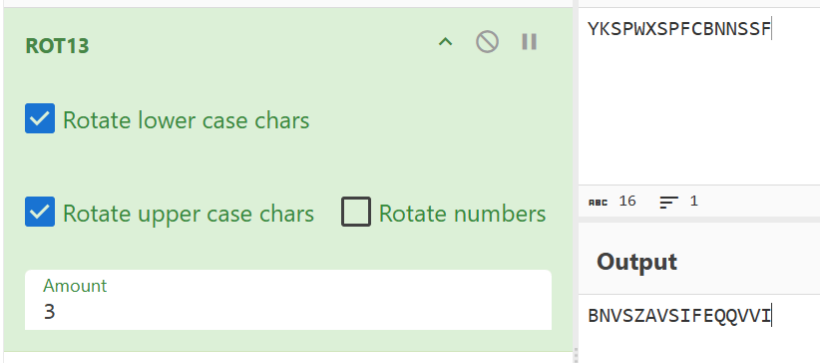
Later, the author released a hint:

▼ View Hint
YKSPWXSPFCBNNSSF

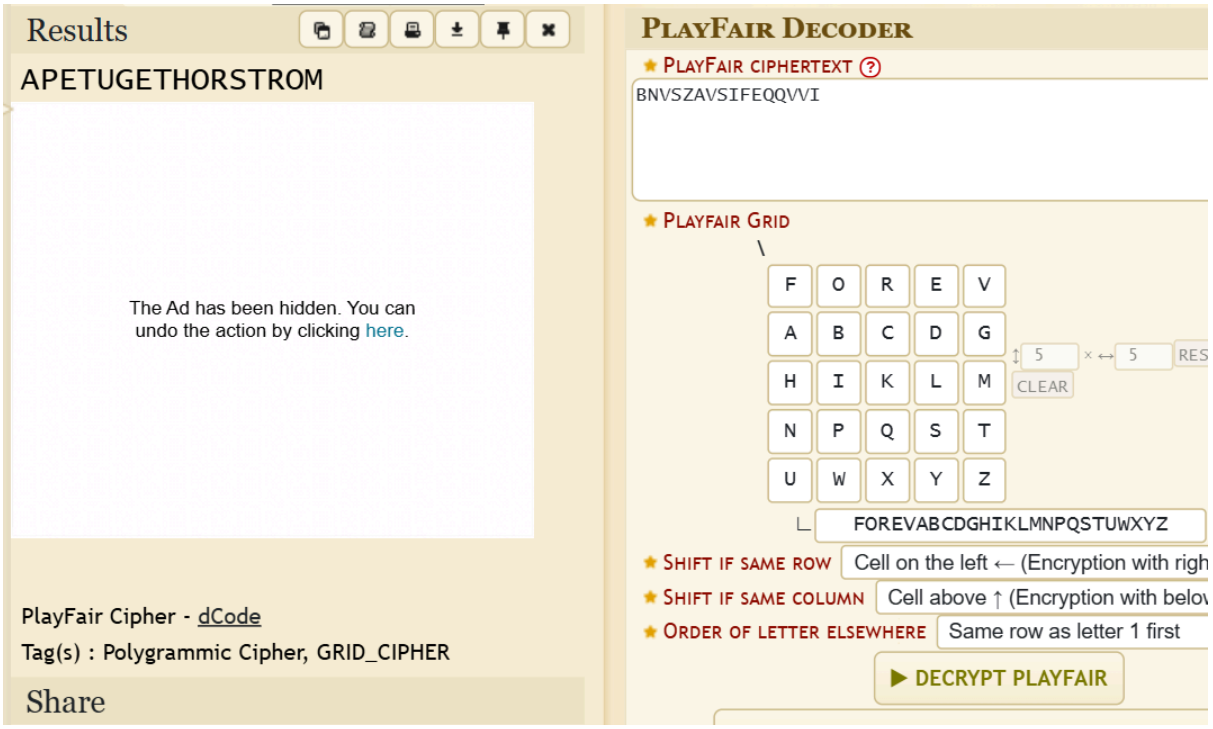
▼ View Hint
Hint on working with cipher.txt: Shift 3

▼ View Hint
5×5, append, the rest is abcd

we given new cipher(?) and hint said the cipher working if we shift/rotate it by 3.



then, decrypt the output with playfair cipher. and use "FOREVER" in playfair grid, the rest is abcd....



Flag: `FindITCTF{APETUGETHORSTROM}`

Kwisatz ZKPerach

896

Could you help young Paul Atreides pass the test from the menacing Reverend Mother?

PS: You don't need to have watched Dune to solve this, but it's a very good movie nonetheless.

author: BerlianGabriel

nc ctf.find-it.id 6101
chall.py

We were given the `chall.py` file, which is the source code for the server. So let's take a look and analyze it.

```
from math import gcd
import random
from flag import flag

class Verifier:
    def __init__(self, y, n):
        self.y = y
        self.n = n
        self.previous_ss = set()
        self.previous_zs = set()

    def spin_roulette(self) -> int:
                                                        return       random.randint(0,
115792089237316195423570985008687907853269984665640564039457584007913129639934)

    def verify(self, s, z, b) -> bool:
        if s in self.previous_ss or z in self.previous_zs:
            print("Bad: repeated s or z")
            return False

        self.previous_ss.add(s)
        self.previous_zs.add(z)

        n = self.n
        y = self.y
        if s == 0:
            print("Bad: s = 0")
            return False
        if gcd(s, n) != 1:
            print("Bad: gcd(s, n) != 1")
            return False
        return pow(z, 2, n) == (s * pow(y, 1 - b, n)) % n
```

```
def main():
    print("Welcome to the mind reading test! Let's see if you're worthy of becoming
the Kwisatz Haderach")
    no = 0
    passed = 0
    n_rounds = 256

    while no < 100:
        if passed >= 100:
                print("Paul Atreides, you've convinced me that you are worthy to be
part of the Bene Gesserit")
            print(flag)
            return

                                                                        n         =
1020531697072943163948579766455988687349070148742004146110200458073575158575174293 8
892976099986403177553363193830393487376567969420541261258134979327616363126253347 14
861054404980720422628493090750342040516620916854112863268863744587072628738305639 03
773773821076228615047462121311793214684571036869046349789852622250839238997290781 73
292553918759616384301941301278845655112236714906572052945789912210749004588396399 36
789079334776958500031487797059636528036936295861130163307443416011583371445983593 38
601977716906142937631000209274422092691356806581113699230299088400015329341575567 01
107140402652365541506235916261071723
        print(f"n = {n}")

        x = random.randrange(1, n)
        y = pow(x, 2, n)
        print(f"y = {y}")

          print("\nCan you read my mind? I'm playing the secret message in my head
now.")
        print("1) yes\n2) no, I can't read your mind at the moment")
        choice1 = input("Your choice [1/2]: ").strip()
        if choice1 == "2":
            no += 1
            continue

           print("Now, there are many spies in this public place. Show me that you
know the secret message without showing me the secret message")
        verifier = Verifier(y, n)

        for i in range(n_rounds):
            s = int(input("Give me an s: ")) % n

            print("Let's spin the gigantic roulette to determine your fate:")
            b = verifier.spin_roulette()
            print(b)
```

```
            print("Are you ready to show me?")
              print("1) yes\n2) no, I am not ready, I need to calm my mind\n3) no, I
forgot the mind reading from earlier")
            choice2 = input("Your choice [1/2/3]: ").strip()
            if choice2 == "2":
                no += 1
                if no >= 100:
                    return
                continue
            elif choice2 == "3":
                no += 1
                if no >= 50:
                    return
                passed=0
                break

            z = int(input("Give me a z: ")) % n
            if verifier.verify(s, z, b % 2):
                      print(f"Good, you are telling the truth, but I am still not
convinced")
                passed += 1
            else:
                 print("I am the Reverend Mother, I can see through your lies. Don't
you dare lie to me!")
                return

     print("You're just wasting my time, Atreides. You will never be the Kwisatz
Haderach")

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

We need to input the correct values of `s` and `z` at least 100 times. To do that, we first have to predict the value of `b` before we can input `s`. In other words, we need to predict `b`.

To solve this challenge, we can use `randcrack`. In order to predict `b`, `randcrack` needs at least 19,968 bits of random output. We can get these bits from the random values generated by the roulette spins. Since each roulette spin gives us 256 bits, we only need 78 spins to gather enough bits — leaving us with 22 spare attempts we can skip if needed.

After predicting `b`, we can generate a random value for `z` (making sure `s` and `z` are not the same). Then, we calculate `s` using the following logic:

- If `y` is even, keep `y` as is
- If `y` is odd, set `y = 1`

- Then compute:
  $z^2 = s * y \bmod n$
  Which means:
  $s = (z^2 * y^{-1}) \bmod n$

this solver.py:

```
import random
from pwn import *
from randcrack import RandCrack

context.log_level = 'debug'
HOST = 'ctf.find-it.id'
PORT = 6101
io = remote(HOST, PORT)
n                                                                              =
1020531697072943163948579766455988687349070148742004146110200458073575158575l
7429388929760999864031775533631938303934873765679694205412612581349793276163 6
3126253347148610544049807204226284930907503420405166209168541128632688637445 8
7072628738305639037737738210762286150474621213117932146845710368690463497898 5
2622250839238997290781732925539187596163843019413012788456551122367149065720 5
2945789912210749004588396399367890793347769585000314877970596365280369362958 6
1130163307443416011583371445983593386019777169061429376310002092744220926913 5
6806581113699230299088400015329341575567011071404026523655415062359162610717 2
3

io.recvuntil(b'y = ')
y = int(io.recvline().strip())
inv_y = pow(y, -1, n)
io.sendlineafter(b'Your choice [1/2]:', b'1')

rc = RandCrack()
for i in range(78):
    io.sendlineafter(b'Give me an s: ', b'3')
       io.recvuntil(b"Let's  spin  the  gigantic  roulette  to  determine  your
fate\n")

    b = int(io.recvline().decode().split('\n')[0])
    log.info(f'Get b = {b}')

    while b > 0:
        rc.submit(b % (1 << 32))
        b >>= 32

    io.sendlineafter(b'Your choice [1/2/3]:', b'2')
```

```
for i in range(256 - 78):
    b = rc.predict_randint(
        0,

115792089237316195423570985008687907853269984665640564039457584007913129639934,
    )
    z = random.randint(0, n - 1)
    s = (pow(z, 2, n) * pow(inv_y, 1 - (b % 2), n)) % n
    io.sendlineafter(b'Give me an s: ', str(s).encode())
    io.recvuntil(b"Let's spin the gigantic roulette to determine your fate")
    io.recvline()
    server_b = int(io.recvline().strip())
    assert b == server_b, f'Predicted {b}, got {server_b}'
    io.sendlineafter(b'Your choice [1/2/3]: ', b'1')
    io.sendlineafter(b'Give me a z: ', str(z).encode())
    res = io.recvline().strip()
    if b'Good' not in res:
        print(res)
        log.error(f'Failed on round {i}')
        exit()
    log.info(f'Round {i} passed')

io.interactive()
```

Flag: `FindITCTF{1f_ZKP_3xiSt_1n_Dune_Truthsayer_w1Ll_g0_3xt1ncT}`

caesar cipher

100

Pada suatu malam, Tung Tung Tung Tung Sahur ingin mendatangi seorang pemuda yang tidak bangun sahur setelah dipanggil sahur sebanyak 3 kali, tetapi tidak nyaut. Masalahnya adalah pintu kamar pemuda tersebut terkunci dengan password tertentu, tetapi terdapat file cipher.txt yang tersimpan dalam flashdisk di dekatnya yang bisa digunakan untuk menemukan passwordnya. Bantulah Tung Tung Tung Tung sahur untuk menemukan passwordnya!

author: mojitodev

Diberikan sebuah file yang berisi:

Ymnx nx f xjhwjy ymj vzny htzw fyyjw. Qnkj ymj bnqq gj f xjhtsi bj bnqq gjfyyj, jshwduynts ymj tahu ts ymj xtrj tk ymj ufxxfrrnsl gjktwj. Tzlm rjxxflj, ymj htsyfsy tk ymj xtrj qnkj f hfjxfw ns yjcy. Qjilmynts ymj jshwduy rjxxflj kwtr f wjfi ymj rjxxflj yt ymj fxyjw. Rjxxflj xynsl ymnx KnsiNYHYK{Mrrrrr_1_W89qqd_i5sy_pstb_Ym8_U5xxbtwi}

Seperti judulnya ini adalah caesar cipher jadi tinggal pake https://www.dcode.fr/caesar-cipher buat dapetin plaintextnya, maka plaintext yang kita dapatkan adalah:

Ini adalah rahasia quit cour atter. Hidup akan menjadi yang kedua kita akan mengalahkan, enkripsi yang pertama pada beberapa passamming sebelumnya. Ough pesan, isi dari beberapa kehidupan caesar dalam teks. Ledghtion mengenkripsi pesan dari baca pesan ke aster. Pesan menyengat ini FindITCTF{Hmmmm_1_R89lly_d5nt_know_Th8_P5ssword}

Flag: `FindITCTF{Hmmmm_1_R89lly_d5nt_know_Th8_P5ssword}`

Osint

destroyer

100

Kau tahu? ada suatu kaum yang dikurung dari zaman dahulu hingga sekarang. Mereka bakal bisa naik pesawat gak ya wkwkwkkwkw.

Format FLAG: FindITCTF{coordinateX_coordinateY}

author: hilmo
street_view.png

given street view screenshot.

First, I tried using Google Lens on the building in the file, and that helped us identify the location. So I checked the street view in Mestia Airport using Google Earth, and we can see a similar view there.

customize the street view until it looks exactly like the screenshot provided

Flag: FindITCTF{43.056574_42.7503479}

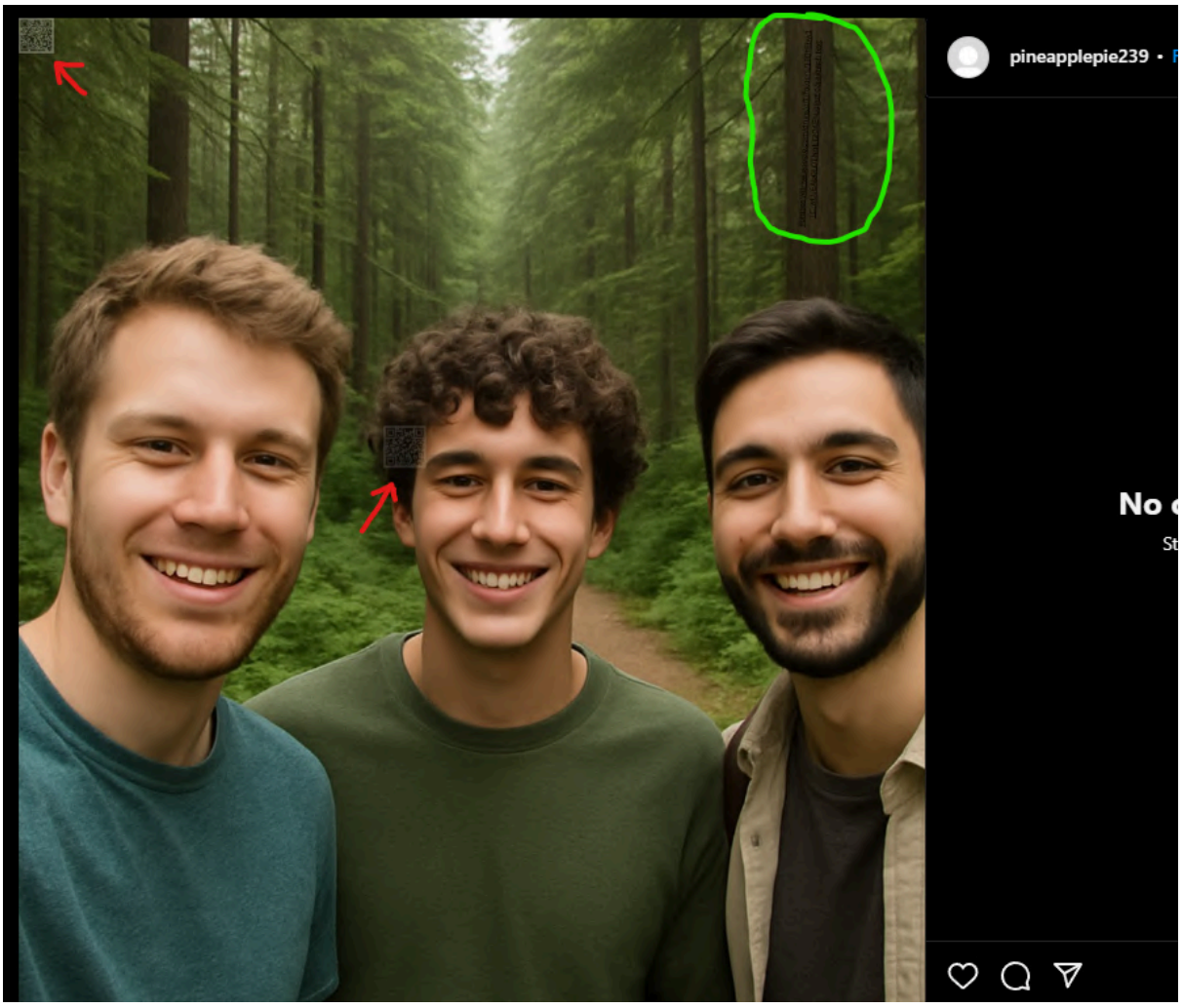
bff

949

kamu adalah seorang detektif, dan di sewa seseorang untuk mencari sebuah doksli, penelusuranmu mempertemukanmu dengan tiga sekawan. setelah mematai-matai mereka, mereka menyimpannya pada file rahasia, karena mereka pelupa, mereka menaruh banyak jejak di sosial media mereka untuk mengakses file itu....

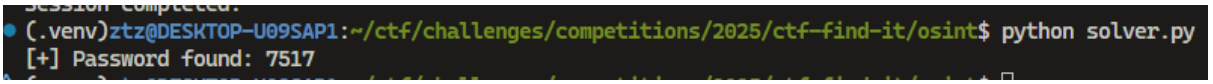
author: mojitodev
start.txt

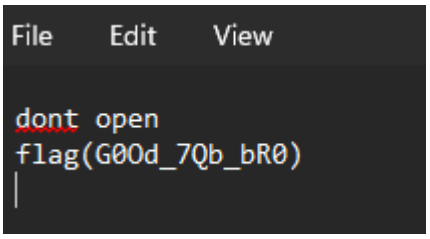
We were given an Instagram profile URL in `start.txt`: https://www.instagram.com/pineapplepie239/. One of the posts looked really suspicious — I noticed a QR code near the person in the center. But the author was just trolling, *wkwkwk*. Then I spotted another QR code in the upper-left corner… and yep, another troll. I thought that was it, but then my eyes caught something else — a hidden Google Drive link in the tree trunk.



https://drive.google.com/drive/u/0/folders/1FOtEbw1TC-wUJEAb0LQTNqET20AEhLqg

On the Google Drive link, there are two files: `clue.txt` and `dont-open.zip`. To unzip the `.zip` file, a password is required. I tried brute-forcing it — and it worked.





Flag: FindITCTF(G0Od_7Qb_bR0)