ENIGMA UNVEILED

Solution:

Downloading the Challenge Binary

- The first step was to download the challenge binary from the provided link or platform.

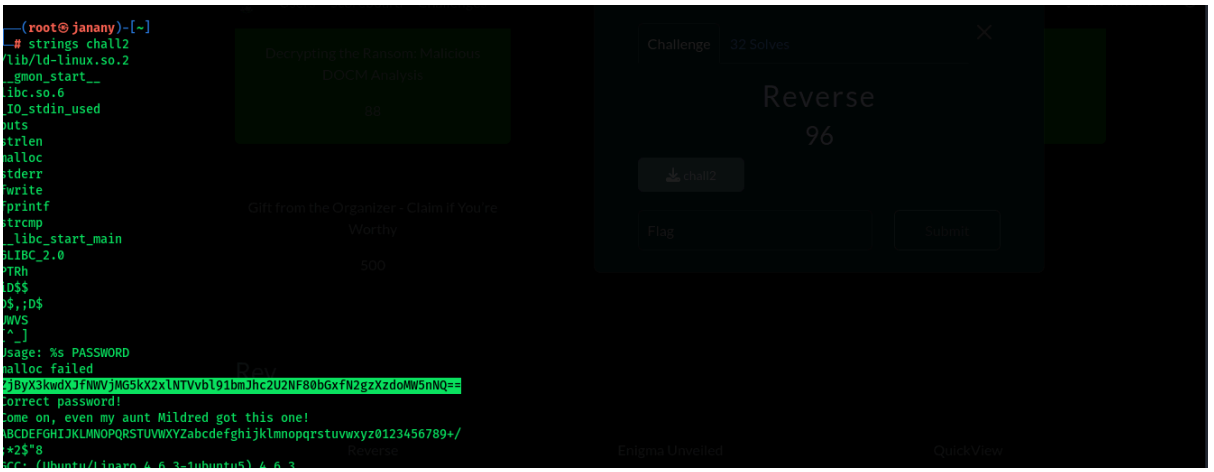
Analyzing the File Format

- Used the `file` command to determine the type of binary:
- 

  The output revealed that the binary is in **ELF format**, meaning it's a Linux-based executable file.

Inspecting the Binary for Strings

- As this was the first challenge in reversing, instead of diving into advanced tools like IDA , I opted for a simpler approach using the `strings` command:

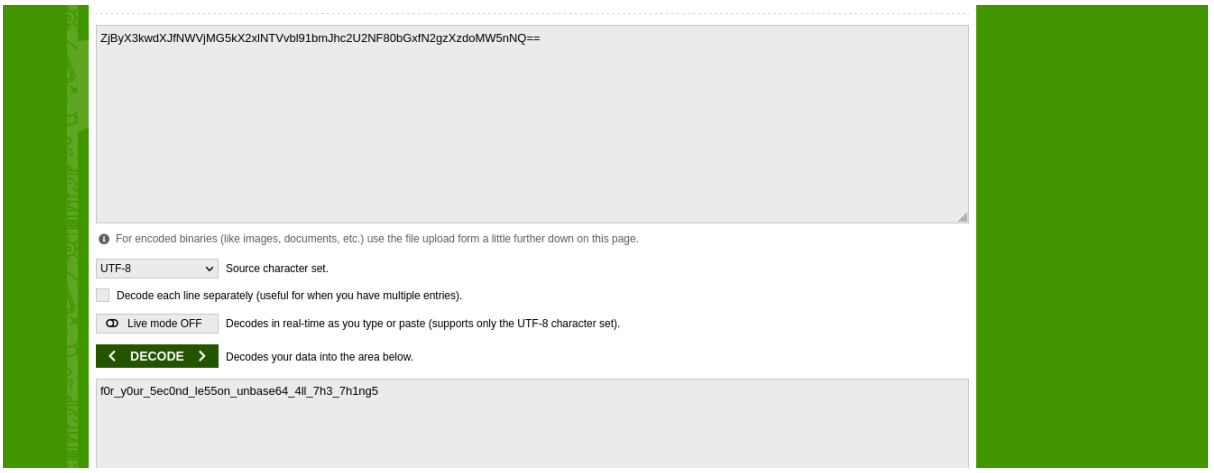


- The command displayed all human-readable strings within the binary.

Identifying the Flag

- While scanning the output of `strings`, I spotted a **base64-encoded string** that looked like the potential flag.

Decoding the Flag

- Copied the base64-encoded string and decoded it using an online decoder (or via a command-line tool like `base64`):
- 

- The decoded output revealed the flag!

Reverse

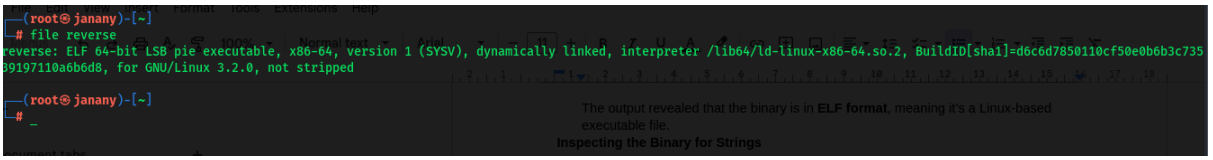
Solution:

Downloading the Challenge Binary

- The first step was to download the challenge binary from the provided link or platform.

Analyzing the File Format

- Used the `file` command to determine the type of binary:



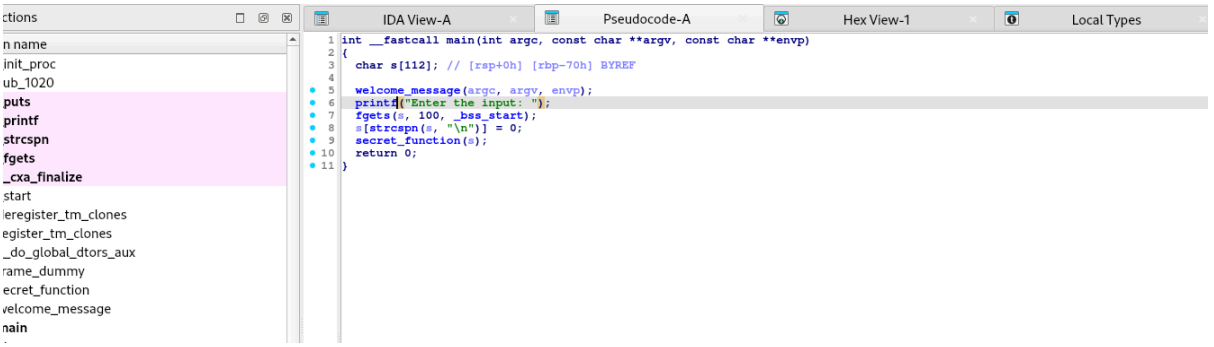
Inspecting the Binary for Strings

- I used the `strings` command and unfortunately no interesting information I could fine

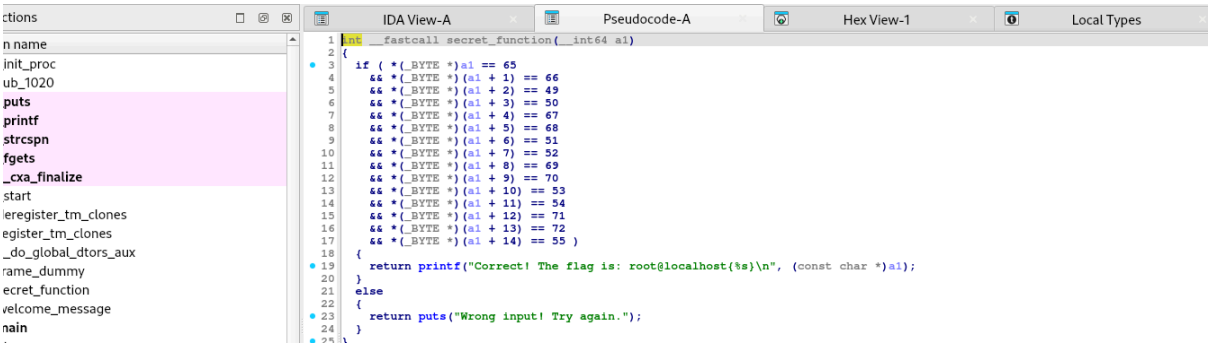
Opening the Binary in IDA

- Loaded the binary into **IDA Pro** for deeper analysis.

Navigating to the `main` Function

- Using IDA's decompiler, I went straight to the `main` function, which often contains essential logic for program execution.
- While analyzing the `main` function, I noticed a call to an **interesting secret function**. This function caught my attention as it wasn't immediately apparent from `main`.
- 

Analyzing the Secret Function

- Decompiled the secret function using IDA's pseudo-code view to understand its logic.
- Within the function, I found a **sequence of random hexadecimal values** stored in an array. This was the critical clue indicating the flag might be constructed from these values.
- 

Converting Hex Values to ASCII

```
int __fastcall secret_function(__int64 a1)
{
  if ( *(_BYTE *)a1 == 'A'
    && *(_BYTE *)(a1 + '\x01') == 'B'
    && *(_BYTE *)(a1 + 2) == '1'
    && *(_BYTE *)(a1 + 3) == '2'
    && *(_BYTE *)(a1 + 4) == 'C'
    && *(_BYTE *)(a1 + 5) == 'D'
    && *(_BYTE *)(a1 + 6) == '3'
    && *(_BYTE *)(a1 + 7) == '4'
    && *(_BYTE *)(a1 + 8) == 'E'
    && *(_BYTE *)(a1 + 9) == 'F'
    && *(_BYTE *)(a1 + 10) == '5'
    && *(_BYTE *)(a1 + 11) == '6'
    && *(_BYTE *)(a1 + 12) == 'G'
    && *(_BYTE *)(a1 + 13) == 'H'
    && *(_BYTE *)(a1 + 14) == '7' )
  {
    return printf("Correct! The flag is: root@loca
  }
  else
  {
    return puts("Wrong input! Try again.");
  }
}
```

Constructing the Flag

- Placed the decoded string within the given flag format `root@localhost{}`.

Submitting the Flag

- Submitted the constructed flag and successfully completed the challenge.