DESCRIPTION

Connect to 141.85.224.106:31338. Bad things happen when you don't initialize local variables.

RESOURCES

As part of the challenge I received an executable file called **uninitialized** as an attachment for analysis as well as its **equivalent source code**.

APPROACHES

1. Here, the first approach was to directly look over the source code and some observations had to be made:
    a. We have a main function calling a foo function;
    b. The foo function reads a maximum 6 digit number and then reads maximum 16 bytes from keyboard and places them a **buffer plus the initial read number position**. Considering the fact that the buffer only has 1024 bytes but the input number could be at maximum 6 digits, we can do a write at an out of bounds index so this is where I saw the vulnerability;
    c. The foo function call a **verify** function that needs to return somthing appart from 0 in order to get the flag. This function gets the user input buffer as a parameter;
    d. The **verify** function creates another buffer with the value **"Where is the flag?"** and passes it together with the user input buffer further to **verify_idx** function;
    e. The **verify_idx** function compares the first **strlen(buf + n)** characters of the two strings (the one mentioned above and the one given from the user) were buf is the string **"Where is the flag?"** and if those are equal it returns 1 which goes all the way to the **foo** call to **verify**.
    f. The final observation here is that the **n** in the expression **strlen(buf + n)** is undefined.
2. Having all the observations above, our task actually narrows to having the expression **strncmp(buf, str, strlen(buf + n)) == 0** return 1 with n being undefined.
3. **N** being undefined we have to somehow put values on the stack such that when it gets initialized it will gather from the stack the value that we are interested in. This N has to be a value that when added to buf which is **"Where is the flag?"** it will return a string of length 0 as then any two strings are equal when comparing the first 0 characters of them;
4. That being said, the length of the **buf** variable is **18** so we will have to make **N** pick up **18** from the stack when it gets declared and the we will compare the input string and buf on the first (strlen('\0') = 0) elements which will be **true**;
5. That being said, taking into consideration that the variable **N** is going to be declared after the user input reading it means that it will be on the stack below the buffer. This means that we will have to read a negative number from the keyboard and write **18** at that position.
6. Here my approach was to do a **brute-force** starting from **-2048** (because there will be at least one more string with 2048 characters on the stack, the one initialized with **"Where is the flag?"** and go down until I receive back the flag. Luckily, it was immediatly after, at position **-2147**.
7. I adapted the script after that such that I only send 1 request now when needed and if we run the script we obtain the flag which is:

CNS_CTF{8a8a7d0592b0f2e4022cba39e0079fde}