DESCRIPTION

Jim solved the security issue by removing the call to system from the program. We don't think that's enough.

Connect to 141.85.224.106:31343 and get the flag.

RESOURCES

As part of the challenge I received an executable file called **canary-2** as an attachment for analysis as well as a **libc** version used for compilation.

APPROACHES

1. The first approach here was to run the program and after running it I could see that it is requiring a input string which after it is provided it is returned back; However the possible patterns based on the input string are pretty spread out so I decided to decompile the program using Ghidra such that I get an ideea of what is happening.
2. Thus, the decompilation looks like this:

```
puts("Hello, ");
puts("Welcome to CNS CTF");
while( true ) {
  puts("Do you want to continue? [y/n]");
  gets(local_28);
  pcVar1 = strchr(local_28,0x6e);
  if (pcVar1 != (char *)0x0) break;
  pcVar1 = strchr(local_28,0x79);
  if (pcVar1 == (char *)0x0) {
    puts("Hmmm, not a valid option. Let\'s try again.");
  }
  else {
    printf("You chose ");
    printf(local_28);
    puts("\nI don\'t think that is the correct choice. Try again.");
  }
}
puts("Okay then, goodbye!");
if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
                    /* WARNING: Subroutine does not return */
  __stack_chk_fail();
}
```

3. Here we can see that we are reading a string with **gets** function from **stdin.** If that string contains the character **0x6E (n)** it directly leaves the while loop. However, if the string contains the character **0x79 (y)** we receive backwards with a **printf** the string that we sent initially.
4. Additionally, we can also see that at the end, the function **__stack_chk_fail();** is called which means that there is a **canary** value present on the stack.
5. Searching through the executable, we can see that there is no more a function called **flaggy** which if we call we are going to receive the flag. This means that using the same **gets** vulnerability we can open a shell ourselves

6. Being in a **while loop** the idea is the firstly send a string that contains an **y** and expose the **canary** value. And then use another string containing an **y** such that we override the return address of **run** function to call **puts@plt(puts@got)** such that we leak the address of **puts** in our version of **libc** and then having the **offset** of **puts** we can determine the start of **libc** and any function inside it. We are interested of **system(„/bin/sh”)**.

7. That being said, we need to somehow expose the **canary value from the first call of printf**. Any function receives any number of arguments. The first 6 arguments are in registers. The **canary** value is placed on the stack right below the **base pointer.** The buffer sent to **printf** by default is at address **rbp – 0x20**, the canary value is just above the **buffer**, but below the **rbp**. This means, that the canary value is at **rbp – 8.** This means, that in order to arrive and print the canary value we need to go to (rbp – 0x8) from (rbp – 0x20) which means a 0x18 = 24 bytes gap. So, having 6 parameters in registers and then going 24 bytes gap, which is (3 x 8 bytes) so another 3 parametrs, we need to pass 6 + 3 = 9 parameters and print the 10th one.
8. In order to send to printf a specific string formatter such that it prints a specific parameter we can send to printf the following format: **%<position>$<specifier>**. So in our case it will be something like **%9$x**.
9. Then we can just save this value as the canary for the further overflow.

10. The final overflow will have a payload that will override the **canary** value with the value saved above and the return address of **run function** to a **pop_rdi_ret** gadget that will put the first parameter as **puts@got** and then will call **puts@plt** with it. Then, after leaking the **puts** address we will jump back to beginning of **run** function where we will override the return address of **run function** to a **pop_rdi_ret** gadget that will put the first parameter as the **„/bin/sh”** string and the will call **system** with it.
11. If you run the script (**python3 script.py**), you will get a shell on the server and if we run **cat /home/ctf/flag** we will get the flag which is:

CNS_CTF{9bf632118c90f4c6e0f50d918493a56d}