Assignment 2

IDATT2503 - Security in programming and cryptography
Fall 2023

Francesca Grimaldi

1 PyCalc challenge

1.1 Summary

After connecting to `10.9.8.1:5006`, the calculator program welcomes the user suggesting to type `help` to display all the available commands. The possible ones are the following:

- The first, `echo`, prints out the argument inserted by the user to standard output. For instance, typing `echo hello` would print `hello` back to the user.
- The second one, `op`, evaluates a simple function. The result of typing `op 3+1` would therefore be seeing `4` printed out.

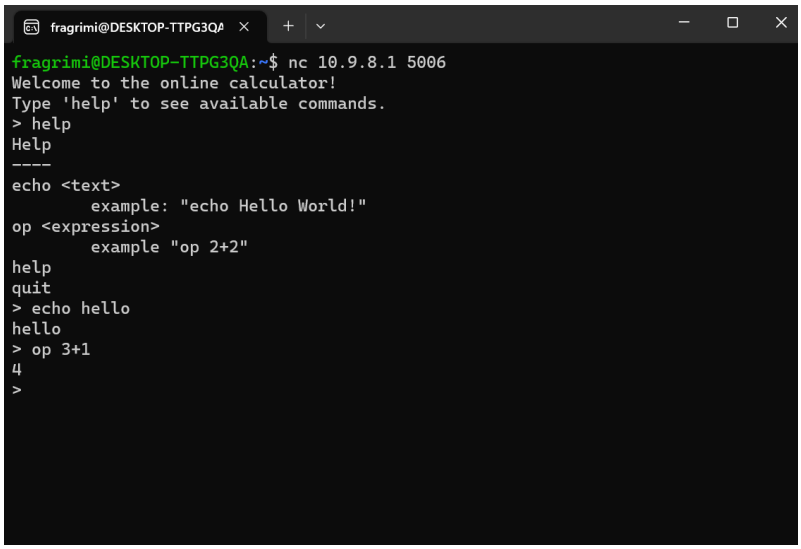

Figure 1: PyCalc operations

The `op` command calculates the sum and other equations using a vulnerable Python function, called `eval()`. In general, it evaluates and executes any specified expression, if it is a legal Python statement. The problem is that accepting untrusted input to `eval()` can allow for arbitrary code execution by malicious users.
Writing

```
op __import__('subprocess').getoutput('ls')
```

reveals the list of all the directories and files present in the directory in which the calculator program is running.

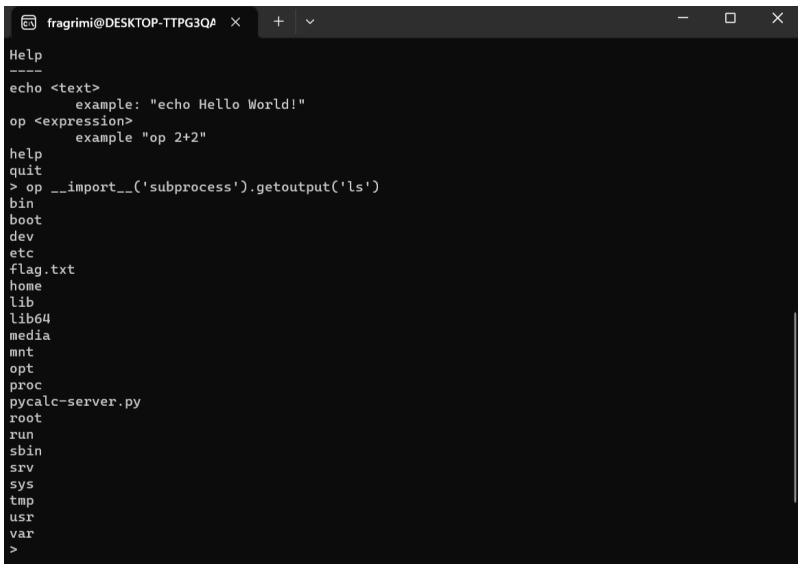

Figure 2: List of files and directories

It is possible to notice that there is also a text file called `flag.txt`. Analogously, in order to show its content it is sufficient to write

```
op __import__('subprocess').getoutput('cat flag.txt')
```

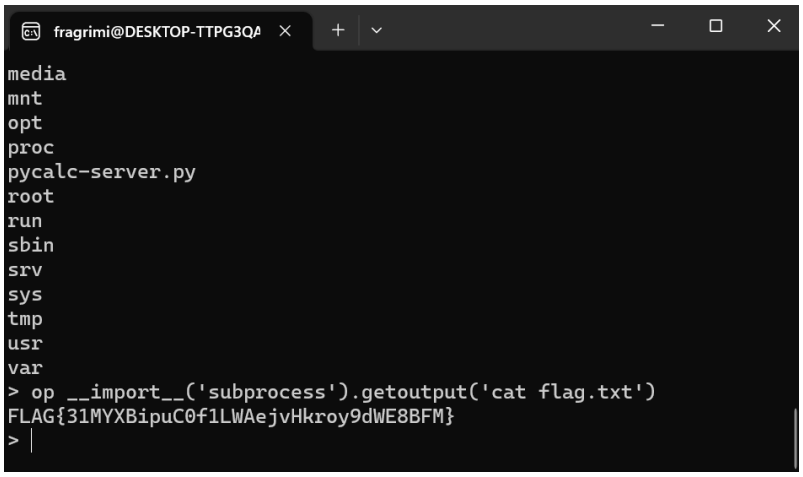

Figure 3: Flag

1.2 Steps to reproduce

The steps to reproduce are the following:

1. Connect to `10.9.8.1:5006`
2. Display the content of the current directory writing

   ```
   op __import__('subprocess').getoutput('ls')
   ```

3. Get the flag printing the content in the `flag.txt` file

   ```
   op __import__('subprocess').getoutput('cat flag.txt')
   ```

1.3 Impact

Using the `eval()` function to evaluate the arithmetic expressions of a calculator unnecessarily exposes the program to significant risks. An attacker could use this function's vulnerabilities to view any file in the directory, modify or delete them, move among the directories, etc. Generally speaking, they can have their own arbitrary code running under the current user privileges.

1.4 Timing

It took me some hours to identify the weakness and how to exploit it writing the Python code in a compact way, but it appears that it can be fixed quickly – possibly within 60 days.

2 Ghidra binary analysis

After opening the file with Ghidra and decompiling it, I located the `main` function.

It is possible to see that the function acquires user input through `fgets` and stores it in `local_28` variable. It then displays it back to the user with a `printf`.

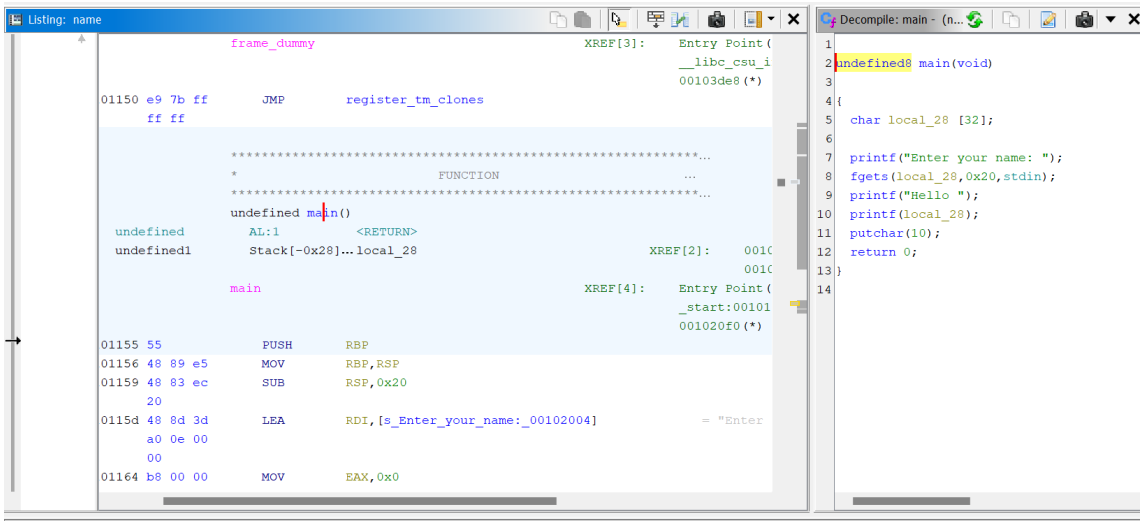

Figure 4: Decompiled main() function with Ghidra

The vulnerability is in the fact that `printf` is not a safe function but is exposed to Format String attacks. An attacker could insert malicious code in the input string, for instance use special characters like `%x` to print contents of the stack.

It could be avoided using other functions like `puts`.