Polaris (Launched 1998)

- *Username:* `polaris`
- ▶ *Click to reveal password:*
- *Points:* 5 for checkpoint, 5 for code, 5 for writeup

Relevant lectures: 5 - Mitigating Memory Safety Vulnerabilities

> **STORY**
> The Spica logs seem to be definitive proof of EvanBot's existence, but without further clues, you seem to have hit a dead end. Luckily, some time later, CSA assigns you to deorbit Polaris, a former Gobian spy satellite. As the space race became more competitive, newer Gobian Union satellites like Polaris introduced stack canaries to protect top-secret information from enemy spies. Although stack canaries were considered state-of-the-art defense at the time, we now know that they can be defeated. Hack into Polaris to see what intelligence it contains, and don't forget to deorbit it afterwards.

For this question, **stack canaries are enabled**. You need to make sure the value of the canary isn't changed when the function returns, but you still need to overwrite the RIP. Can you find a way to get around this mitigation?

Note: This Project will use 4 random bytes as the canary, instead of 3 random bytes and 1 NULL byte. This is different from what is taught in lecture! For exams, you should still assume that the canary always has one NULL byte.

The vulnerable `dehexify` program takes an input and converts it so that its hexadecimal escapes are decoded into the corresponding ASCII characters. Any non-hexadecimal escapes are outputted as-is. For example:

```
$ ./dehexify
\x41\x42   # outputs AB
XYZ        # outputs XYZ
```

Note that we are not inputting the byte `\x41` here. Instead, we are inputting a literal backslash and the literal characters `x`, `4`, and `1`. Also note that you can decode multiple inputs within a single execution of a program.

For this question, you will write an `interact` script. Instead of doing simple output, the `interact` script has the ability to send **and receive** from the vulnerable program. This means that the output from the program can be used to affect your next input to the program.

The `interact` API

The `interact` script lets you send multiple inputs and read multiple outputs from a program. In particular, you have access to the following variables and functions:

- `SHELLCODE`: This variable contains the shellcode that you should execute. Rather than opening a shell, it prints the `README` file, which contains the password. Note that this is different from the shellcode in the previous two questions.
- `p.start()`: This function reads starts the vulnerable program.
- `p.send(s)`: This function sends a byte string `s` to the C program. You must include a newline `'\n'` at the end of your input string `s` (this is like pushing "Enter" when manually typing input).
- `p.recv(n)`: This function reads `n` bytes from the C program's output.
- `p.recvline()`: This function reads all bytes until a newline (`'\n'`) from the C program's output. The newline is included at the end of the returned string.

An example of sending and receiving is provided in the starter code on the vulnerable server.

Note that in Python, to send a literal backslash character, you must escape it as `\\`.

Tips

- You might want to save some C program output and input part of it back into the C program. No hex decoding or little-endian reversing is necessary to do this. For example:

```
foo = p.recv(12)  # receive 12 bytes of output
bar = foo[4:8]    # slice the second word of the output
```

```
p.send(bar)       # send the second word back to the C program
```

- You can display bytes in a readable format as follows:

```
foo = p.recv(12)
print([hex(ord(c)) for c in foo])
```

- Keep in mind that the function does not return immediately after the buffer overflow takes place (it might help to look at what codes are executed next and think about what it does to the stack), so you will need to account for any extra behavior so that the stack is set up correctly when the function returns.

Deliverables

- A script `interact`
- A writeup.


CONQUERORS OF THE
FINAL FRONTIER