Vega (Launched 1999)

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

Relevant lectures: 4 - Memory Safety Vulnerabilities II

STORY
Vega was a spacecraft developed in a joint mission between Caltopia and the Gobian Union. However, since Caltopia used all uppercase in its software, and the Gobian Union used all lowercase, a utility was needed to convert between uppercase and lowercase. Hack into Vega to learn the truth about EvanBot.

This question has a flaw more subtle than the previous questions. Can you find it? Can you find a way to exploit this seemingly minor vulnerability?

The `exploit` script in this question is slightly different. The output of `egg` is used as an *environment variable*, which means its value is placed at the top of the stack. The output of `arg` is used as the input to the program, passed as an argument on the command line (in the `argv` array to `main`).

Tips

- It might help to read Section 10 of "ASLR Smack & Laugh Reference" by Tilo Müller. (ASLR is disabled for this question, but the idea of the exploit is similar.)
- It might also help to read Section 3.5 (off-by-one vulnerabilities) of the memory safety textbook page.
- Environment variables are stored at the special pointer variable `environ`. To see the addresses and values of environment variables in gdb, you can set a breakpoint anywhere in the program, `run` the program, and run these commands to print out each environment variable as a string:

```
(gdb) p environ[0]
(gdb) p environ[1]
(gdb) p environ[2]
...
```

If you want to view each environment variable as a sequence of bytes (in hex) instead, you can run these commands:

```
(gdb) x/16bx environ[0]
(gdb) x/16bx environ[1]
(gdb) x/16bx environ[2]
...
```

- It may take some trial-and-error to find the output of `egg` among the environment variables. One way to confirm you have the right address is to run `x/2wx [your address]` and check that gdb displays what you put in `egg`.
- Do not attempt to manually bitwise XOR anything longer than 4 bytes (in particular, shellcode). It is too easy to mess up. Instead, consider putting shellcode somewhere else, or use this Python snippet (replace `ZZ` with a constant):

```
output = ''.join((chr(0xZZ^ord(c)) for c in input))
```

- There is a slight chance (1 in 256) that your VM customization causes the value of the SFP to end in `\x00`, which makes this question much harder to solve. You can resolve this by printing out extra garbage bytes in your `egg` script (after whatever you were printing before), which pushes the rest of the stack to different addresses.

Deliverables

- Two scripts, `egg` and `arg`
- A writeup.