Challenge 2

Deadline: Check Piazza

Shell command language interpretation is tricky! There is so much that depends on the execution environment and other parameters controlled by the user, not to mention the quirks of individual shell implementations.

Combine that with classic Linux attacks, and things can quickly escalate.

Here are your learning goals:

- Experience common anti-patterns and insecure code wreak havoc firsthand, before we discuss them more formally later.
- Bear witness to the horrors of system().
- Infer application behavior from blackbox observations, and write your own exploits.

Your Task

Your job is to exploit 5 programs installed under `/usr/local/bin/`

1. **get_time:** print the current time and date
2. **get_ip:** print the IP address of a given network interface
3. ~~**get_primes:** compute and print prime numbers~~ *(canceled!)*
   moon: let's spice it up with something more exciting; try and see what this does
4. **fantasy_pub_generator:** generate a random fantasy-world pub name
5. **prog5do:** like sudo, but for prog5; run a target command with prog5's privileges

These programs have their **setgid** bit enabled. That is, they run with the effective privileges of the groups that own them—namely, the groups **prog1** through **prog5**. (Revisit the *Linux Security* lecture if you forget what that means!)

Your goal? Execute `/usr/local/bin/win` with the effective group IDs of the programs above. For example, you will successfully have exploited `get_time` when you run `win` with the effective group ID of **prog1**. Each time you do that, you'll get a **solution token**. Collect all 5 tokens to complete the challenge.

That's all Folks!

The Approach

There's no source code. What to do?

You are logged into the same machine the programs run on, so you can observe them in action. So, go ahead, do some lightweight blackbox dynamic analysis. We call this *reverse engineering*.

Have you heard of `ltrace` and `strace`? **ltrace** alone is perfectly sufficient to complete this challenge. It gets the job done efficiently, and it is the approach I recommend. As always, remember to RTFM.

If you want to dig deeper, full-scale debugging with `gdb` is always an option available to you. You can even dabble in disassembly and leverage static analysis to complement your dynamic techniques, but none of that is necessary for this challenge.

Tips and Hints

- Here's another way to think about this challenge: Trick the vulnerable programs into executing `win` for you. Or, the equivalent: You make the vulnerable program drop you in a shell running with elevated privileges, and then you execute `win` manually.
- Here are some specific tips:
  `get_time`: Tutorial. You should solve this in ~a minute. If you have no clue what you are supposed to do, drop everything, and revisit the lecture recording.
  `get_ip`: Filtering for bad input is an anti-pattern, it is error prone. Developers often end up making mistakes and miss dangerous input.
  `moon`: A considerably complex program. Remember the classic "code coverage problem" in dynamic program testing. Executing a program with one specific input doesn't execute all the code compiled into a binary, obviously. What if there's a vulnerability in an if/else branch you didn't enter? If you want to see interesting behavior, you'd better execute all possible code branches and maximize your coverage. You do that by exercising the program in every way possible. Inputs aren't limited to your command line arguments, there are other data sources that influence execution.
  `fantasy_pub_generator`: Keep your eyes peeled. It's not always about system().
  `prog5do`: Wait wut? This already does exactly what you need without exploiting anything; it executes the target command with the privileges of **prog5**. Where's the challenge in that? Oh… there seems to be a custom layer of protection there. You should analyze how that works exactly before you plan your attack.
- **IMPORTANT:** Don't get distracted by the mysterious functionality of `moon`. Your goal is the same as always: Find a vulnerability that lets you control what gets executed. *However… Once you complete the challenge, if you couldn't get enough, and you want to see something cool, keep digging.*

Submission

1. Create the challenge directory tree `~/submissions/challenge2/`
2. Create `submission.txt` in the challenge directory. Copy all 5 tokens into this file, each on a single line. Make sure to enter them in order, the token for **prog1** goes on line 1, **prog2** goes on line 2…
3. Run `submit challenge2`

Wait a few moments, and check the results. If you got it right, get some ice cream. Don't forget to check out the Hall of Fame: https://www.khoury.northeastern.edu/home/kaan/rankings.html

Good luck, and happy exploitation!