Welcome

You're one step closer to become part of Wildlife's application security team 💪. These challenges will help us understand your skillset, the way you organize your thoughts and work. Please read the instructions below and follow the steps required to complete each challenge.

You've got **two days** to complete the challenge, take your time and **focus on code quality**, documentation and unittesting. The time you spend on the challenge will depend on your skillset.

Please submit your answers before the two-day limit. Upload all your code to a GitHub or GitLab repository. It doesn't matter if there are incomplete sections or the code doesn't work.

Good luck!

Programming challenge

The challenge is divided into **5 iterations**, solve as many as you can in the given order, one by one, and **focus on code quality** before starting the next iteration. External libraries are allowed, but they should provide you with primitives and not solve one of the core challenges. Use your judgment.
Feel free to use any programming language.

Include a README.md with the documentation on how to use what you developed, and known limitations (if any). For example, if you were able to code Iteration 1 and 2, but not the rest, this should be documented in the README.md.

The result for this challenge needs to be uploaded in a GitHub or GitLab repository and the link to it should be sent in a reply to the email where you got this document.

Iteration 1

Create a port scanner that receives in a command-line argument the hostname to scan and report all the TCP open ports from 1 to 1024.

```
$ port-scanner my-hostname.com
21 open
22 open
23 open
```

Iteration 2

Add concurrency to your port scanner to support checking multiple ports in a faster manner. Include an optional **[-t <threads>]** argument to let the user select the number of threads **(default is 5)**.

```
$ port-scanner -t 4 my-hostname.com
23 open
21 open
23 open
```

Iteration 3

Your port scanner is now easily identifiable by our defensive firewall that detects two patterns:

Attackers opening and closing 3 or more consecutive ports

> For example, someone connecting to port 20, then 21, and finally 22 will trigger the alarm.

Attackers opening and closing at least 4 ports in a predictable timely manner.

> For example, someone connecting to port 10, 1 second later to port 11, 1 second later to port 12, and when finally connecting 1 second later to port 13 will trigger the alarm.

Include a **[-f] optional argument** to disable this feature. The feature should be enabled by default. Passing the -f flag *would* trigger the firewall.

```
$ port-scanner -t 4 -f my-hostname.com
23 open
21 open
23 open
```

Iteration 4

Add a feature to detect if the open port is an HTTP server, avoid connecting twice to each port or our firewall will detect you!.

```
$ port-scanner -t 4 -f my-hostname.com
23 open
21 open
23 open
80 open (http)
```

Iteration 5

Create a docker image to compile (if necessary) and run your port-scanner from a docker container.

```
$ docker run port-scanner:latest -t 10 www.victim.com
```