Exercise 8

Logical Vulnerabilities

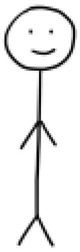
Log into your VM (`user` / `1234`), open a terminal and type in `infosec pull 8`.

- When prompted, enter your username and password
- Once the command completes, your exercise should be ready at `/home/user/8/`

When you finish solving the assignment, submit your exercise with `infosec push 8`.

Background

Meet Bob.



(Bob is from Wait But Why. It's a blog you should read.)

Question 1 (20 pt)

In `q1/`, Bob wrote the `run.py` script, which accepts a path to a file containing a Python dictionary with a username, a password and a command, and executes that command **if these credentials are correct**.

You can check it out by running `python q1.py --example foo`, which creates a valid file called `foo`. You can then use this file by running `python run.py foo`, which validates it and executes its command, echoing `cool`. Cool.

Inside `q1.py`, implement the `generate_exploit` function, so it returns a string that causes `run.py` to print `hacked`. The catch - you should do this without using a valid username and password combination! Describe your solution in `q1.txt`.

- **Hint**: Look for a logical vulnerability in `run.py`'s implementation
- **Note**: It's OK for `run.py` to crash and/or exit with an exception after it prints `hacked`.

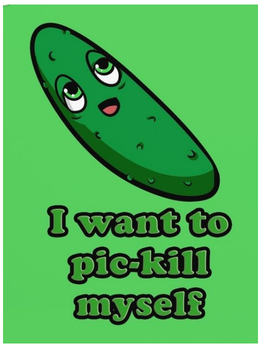
Question 2 (15 pt)

In `q2/`, Bob learned from his mistake, googled more standard ways to store Python data and thus stumbled upon the [Pickle module](). Unfortunately, Bob is not very attentive, so he missed the huge red warning at the beginning of the documentation.

Again, you can check it out by running `python q2.py --example foo` to create a valid file for example, and `python run.py foo` to see it in action. Note that Pickle is a weird binary format, so to view it you'd have to use `hexdump foo` or something similar.

Inside `q2.py`, implement the `generate_exploit` function, so it returns pickled data that causes `run.py` to print `hacked`. Again, do this without using a valid username and password combination, and describe your solution in `q2.txt`.

- **Note**: It's OK for `run.py` to crash and/or exit with an exception after it prints `hacked`.
- **Hint**: Use the internet and search for Pickle exploits. Look what I [found]():



Question 3 (15 pt)

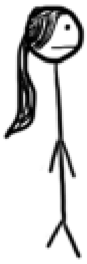
In `q3/`, Bob has had enough with all this dynamic evaluation, and decided to use his own format: `username:password:command`. Unfortunately, Bob is not a very good programmer.

As usual, you can run `python q3.py --example foo` and `python run.py foo`. Inside `q3.py`, implement the `generate_exploit` function, so it returns a string that it causes `run.py` to print `hacked`. Again, do this without using a valid username and password combination, and describe your solution in `q3.txt`.

- **Hint**: Look for a logical vulnerability in `run.py`'s implementation.
- **Note**: In this question, `run.py` **must NOT crash!**

Question 4 (30 pt)

Bob was a little upset by all this, so he called his friend. Meet Alice.



Alice is a better programmer, and she even knows about cryptography (because people seem to keep dragging her into it all the time). In `q4/` they divide the work so she implements the validation, and he implements the execution.

Alice decides to use JSON, and instead of usernames and password, uses cryptographic signatures. Bob isn't familiar with JSON.

You know the drill - `python q4.py --example foo` and `python run.py foo`. Inside `q4.py`, implement the `generate_exploit` function, so it returns a string that causes `run.py` to print `hacked`. Again, do this without using a valid username and password combination, and describe your solution in `q4.txt`.

- **Hint 1**: Look for a logical vulnerability in `run.py`'s implementation
- **Hint 2**: This is not a cryptography exercise - you are not expected to break RSA, but bonus points will be given if you do
- **Hint 3**: This one is a bit tricky. Go back over the recitation for inspiration.
- **Note**: In this question, `run.py` **must NOT crash!**

Question 5 (30 pt)

In `q5/` Alice taught Bob about JSON, so now we're OK. She also upgraded the cryptographic signature algorithm to rehash a gazillion times, making brute-forcing near impossible. It does make the validation pretty slow, but so what.

For this exercise, we don't provide you with any template. Inside `q5.py` (which is currently an empty file), implement code so that `python q5.py` will run `run.py` and make `run.py` print `hacked`. You'll have to call `run.py` from `q5.py` by yourself. **Give a detailed description of your solution in `q5.txt`.**

Since you're implementing `q5.py` from scratch, we've added `example.json` so you can see `python run.py example.json` in action.

- **Hint 1**: Look for a logical vulnerability in `run.py`'s implementation
- **Hint 2**: This one is a bit tricky. Go back over the recitation and lesson 4 for inspiration.
- **Note**: In this question, **nothing should crash!**

Final notes:

- Yes, the points sum up to 110; this time it's intentional, it's not a typo
    - You still need just a 100
- Document your code
- **Really document question 5!** Explain in depth what you're doing and how - you are writing the entire code from scratch, and it's hard to check automatically as is, don't make it harder than necessary :/
- Don't use absolute paths to any of the files! You can assume all the code will be ran from within the exercise directory, so use relative paths with `./`
- Don't use any additional third party libraries that aren't already installed on your machine (i.e. don't install anything).
- The answers for this exercise are really really short. Like, for realz. If it takes 100 lines for all the questions together, that's way too much and there's probably a simpler solution