Homework 4

COM402 - Information Security and Privacy 2018

- The homework is due **Sunday, April 29, 2018 at 23h55 on Moodle**. Submission instructions are on Moodle. Submissions sent after the deadline **WILL NOT** be graded.
- In the event that you find vulnerabilities, you are welcome to disclose them to us (can even have a bonus !)

Exercise 1: Protect your favorite movies (25p)

This time you play the role of a company that collects movie ratings, e.g., IMDb, and you want to securely disclose them. You have a rating database in clear, where each row has the format `[user, movie, date, rating]` database in clear and you will have to disclose it to the public, e.g., researchers, so that they can retrieve only certain information. This is a scenario you see in real life, albeit simplified in our case, of data sets released so that people can derive knowledge by computing statistics, but without being able to learn more about individuals.

First you need to download the following file from our server:
http://com402.epfl.ch/download/com402_hw4_ex1.tgz
Put all three files in the same directory and write your code in `run_anonymization.py`, as follows.

1a - good anonymization

`anonymize_1(db)`- this function receives the clear-text database (an array of `[user, movie, date, rating]`- arrays) and needs to return an anonymized database in the same format such that:

- no `(user,movie)` tuples can be retrieved, even in the case of an external partial database
- the date is anonymized

Even though you anonymize the database, you still have to be able to retrieve some data from it: `get_movies_with_rating(anon, rating)` will be given your anonymized database and it has to return all movies with the given rating.

1b - lesser anonymization

`anonymize_2(db)` takes the same database as `1a` and has to return an anonymized database in that same format so that:

- no users can be recovered

- for security, you also have to take out the dates

The utility function this time is `get_top_rated(anon, movie)`, which takes the anonymized database and a movie. For every user in the anonymized database that has rated the movie, the function returns the top-rated movie (or movies, if there is more than one with the top-rating).

Anonymization

For **full anonymization** of a field, replace it with a single asterisk: `*` - an empty field or other filling character will not be recognized by the anonymization-function and will not be accepted. However, keep in mind that not all fields have to be fully anonymized!

Obtaining the token

The script `run_anonymization.py` is set up so that you can submit it on the com402 website and obtain either the token, or hints on what you're doing wrong. Keep in mind that our tester runs your script multiple times, with different anonymized databases.
`if __name__ == "__main__"` takes care of all this, so don't change it!

When both parts 1a and 1b are correct, you will get a single token.

Exercise 2: bcrypt (25p)

This is a simple exercise where you need to write a minimalistic server (using Flask). The server should expect a POST request on `127.0.0.1:5000/hw4/ex2` with JSON data containing the `user` and `pass`. The only thing your server has to do then is to hash the password using **bcrypt** and send back the hash with status code 200. Make sure you UTF-8 encode the password **before computing its hash**! You should submit your script, named `hw4_ex2.py`, via the submission page.

Exercise 3: Defense of the data (25p)

You're being asked to build a super minimal website using some data stored in a MySQL database. Of course, you're starting to worry about the security nightmare this project might have.

Well not really, so many libraries do the job for us now.

Run the Docker image:

```
docker run -it -p 80:80 --rm --name ex3 dedis/com402_hw4_ex3 bash
```

You will find inside the skeleton script `site.py` where you have to fill in two endpoints `/messages` and `/users`. All information needed is included in `site.py`.
The goal is to make sure your script is not susceptible to SQLi attacks.

Once you have finished, you can upload your own `site.py` to the submission webpage.
Good luck !

Exercise 4: Towards a secure web server in vacuum (25p)

As being a hacker yourself, you know that sole HTTPS serving on you web server does not solve all the problems. Attackers can still try to downgrade users' connections or even make use of laid-back users who still access all the websites by typing http://…
So you know that you cannot rely on users to do all the security precautions.

The goal of this exercise is to further learn how to configure the security parameters of an nginx server to properly protect connecting users!

To start with, run the Docker image:

```
docker run -it -p 80:80 -p 443:443 --name hw4_ex4 dedis/com402_hw4_ex4
```

Make sure that you bind both ports 80 and 443 of your container to localhost, when you run the container (it's set with the -p flag in the command above).

In the container you will find the same setup as in the exercise 4 of HW2 where you had to build a barebone https server that used a certificate signed by our DEDIS-CA. Yes, you have to use the new container because it has a new ssh key written in which allows us to verify your setup.

You need to start with the same steps as in HW2 by adding HTTPS support to the `/etc/nginx/conf.d/default.conf` file. You can reuse your dedis-signed certificate from HW2 (no self-signed is needed). If you do not have it anymore, generate a new one.
For the sake of verification, make sure to type `localhost` in the field Common Name (CN) when generating a request for a certificate to be issued by our DEDIS-CA which you will then use in your nginx server. If you do not do this, the hostname check of the ssl library will not pass, and the connection to your server will be abrupted.

Then you need implement **HTTP Strict Transport Security (HSTS)** in your nginx server. It should be set for all subdomains and with max-age period of 1 year. The latter stands for how long browsers should store cached information about HSTS for this website.
Your next step to implement **explicit redirect from http from https**. The redirect must be supplied with **"Moved Permanently"** status code.

Still having the after-taste the downgrade attack that you carried out yourself, you decide that this must never happen on your server! So your sacrifice backward compatibility and configure your server to **only** connect using **TLSv1.2**. Only hardcore!

Then you realize that what if someone sends an HTTP DELETE request to your server and erases all your carefully collected COM402 course notes? That is not what you have been working hard for! So you decide to limit the types of **HTTP requests that your server will accept, allowing only GET, HEAD and POST**. If someone sends another type of request, e.g., DELETE, you should return **the status code 405** as your response.

Finally, you need to configure your nginx server to notify browsers that they need **to enable XSS protection for your website**. The protection is usually already enabled by default in browsers but it can be manually disabled by users. If you configure it explicitly, it will have a priority over user's configuration.

To summarise, your tasks are:

1. Running HTTPS server
2. HSTS
3. Redirect
4. TLSv1.2
5. Allow only HTTP GET, HEAD, and POST
6. XSS protection

Before testing, add your email address to the `/www/index.html` file in the form of

```
email = “firstname.lastname@epfl.ch”
```

so we know who you are. Once you think your setup is correct, you can run the verification script **inside the container**:

```
./verify.sh
```

You should see your token if everything is fine. Copy the configuration file `default.conf` as `ex4.conf` and submit it via Moodle.