Homework 3

COM402 - Information Security and Privacy 2018

- The homework is due **Sunday, 15th of April, 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 !)
- Our server has now ports 80 and 443 open to the outside world so you do not have to to use VPN anymore when you are outside of campus.

Exercise 1: Find out about your favorite movies

We talked in the course about the Netflix-de-anonymization. In this exercise you have to de-anonymize the dedis-database that holds your secret movie-ratings. The database is given to you as a csv-file with the following format:

```
sha256(salt | email), sha256(salt | movie), date, rating
```

The salt is the same for the whole database. There are 189 emails that correspond to the students and teachers of COM-402. Your goal is to find out what movies you rated in the dedis-database.
To de-anonymize the list, you get a second csv-file from IMDb in the format below. This second list is smaller than the first list.

```
email, movie, date, rating
```

For all sub-exercises, the entries in the IMDb are a strict subset of the entries in the dedis-database.
The exercise has **3 sets of csv-files** (dedis-db and IMDb-db), with increasing difficulty to recover the movies you rated:

1. **(5p)** Dates are giving it away - each user rated the movie at the same date in both the dedis-db and IMDb-db.
   Hint: some dates might have more than one rating, so you need to make sure you remove these doubles.
2. **(10p)** More realistic: the dates are random, reflecting the fact that you won't rate a movie the same day with Netflix and the IMDb.
   However, a simple frequency-attack on the movies is enough to map the movies to the hashes, then you'll have to fit the IMDb with the dedis-database.
   Hint: Once you mapped the hashes of the movies to the plain names of the movies, search for any user who rated all films you find in its public ratings.

3. **(15p)** More realistic database: dates of ratings in dedis-database and IMDb are not the same, but similar. Dates are within 14 days, with a triangular distribution, using python's *random.choices* and a weight of `[1, 2, 3, …, 14, 13, 12, …, 1]`.
   Hint: First search for user-name hash/plaintext overlap and fit those to find the hash of your email. Then you can search for the closest overlap of the public ratings and the anonymous ratings of your email.

To get the exercise-files, you have to enter your email-address on http://com402.epfl.ch/hw3 and download the ***hw3_ex1_email@epfl.ch.tgz*** file. Inside you'll find 6 files.
For exercise 1 you get dedis-1.csv, imdb-1.csv
For exercise 2 it's dedis-2.csv, imdb-2.csv
For exercise 3 it's dedis-3.csv, imdb-3.csv

You can solve the exercise in any language you like, even paper and pencil, if you're not afraid of searching for overlaps in 156 names. Of course you should not solve the exercise for somebody else. For every exercise you have to **submit the list of your favorite movies** to the com402-website and retrieve the token. **Make sure you remove the quotation marks from the movie names**!

Exercise 2: L33t hax0r5

In this exercise you will need to crack some passwords! The exercise consists of 3 parts. To get the exercise file you have to enter your email address on http://com402.epfl.ch/hw3 and download your 'hw3_ex2.txt' file. In this file you will find 10 password hashes for each part of the exercise. Your goal is to crack those hashes and reveal the passwords.

2a (6p) Brute force

In this part you should implement a brute-force attack. Passwords are randomly generated from the set of lowercase letters and digits ('abcd...xyz0123...9') and have length 4, 5, or 6 characters. Generated passwords are then hashed with SHA256 and corresponding hexdigests are sent to you in the file.

2b (7p) Dictionary attack with rules

In the previous part of the exercise you implemented the brute force attack and hopefully noticed that it takes a lot of time to crack a random password. Unfortunately, people very rarely use random passwords. Instead they use some common words and sometimes modify them a bit. This is a fortunate fact for password crackers, because they can use 'dictionary attacks' to crack the passwords more efficiently than with brute-force. In this part you should implement one such dictionary attack. We generate a password by selecting a word from a large dictionary and then randomly applying some of the common user modifications:

- capitalize the first letter and every letter which comes after a digit (for example: 'com402dedis' becomes 'Com402Dedis'). If you are using Python, this is easily achieved by '*title()*' function from string module (*'com402dedis'.title()* will give you *'Com402Dedis'*)

- change ‘e’ to ‘3’
- change ‘o’ to ‘0’ (that’s small letter ‘o’ to zero)
- change ‘i’ to ‘1’

(For example, ‘window’ can become ‘W1ndow’, or ‘w1nd0w’, …)
In the file you received you will find the SHA256 hexdigests of passwords generated in this way. Your task is to crack them using a dictionary attack. Dictionaries can be found online (e.g. https://wiki.skullsecurity.org/Passwords).
Note1: the words we used to generate passwords don’t contain any special characters, in other words they contain only uppercase and lowercase letters and digits.
Note2: Not all dictionaries are the same, be aware that if you implement the attack correctly but you can’t crack the passwords, then you might be using a dictionary which doesn’t contain all the words as the dictionary we used.

2c (7p) Dictionary attack with salt

In the previous part of the exercise you implemented a dictionary attack. You should notice that once you have a dictionary you can compute the hashes of all those words in it, and create a lookup table. This way, each next password you want to crack is nothing more than a query in the lookup table. Because of this, passwords are usually ‘salted’ before hashing. In this part of the exercise you should implement another attack using a dictionary. We generate a password by simply selecting a random word from a dictionary and appending a random salt to it. The password is then hashed with SHA256 and hexdigest and salt are sent to you in the file. Your task is to crack the passwords using a dictionary.
Note: Salt is exactly two characters long and it contains only hexadecimal characters.

Submission

Once you discover the passwords, you can submit your solution to the submission web page: http://com402.epfl.ch/hw3. For each part of the exercise you should submit one file which contains all 10 cracked passwords separated by comma (‘,’). The file should contain only the passwords, not the hashes, not the salts, not anything else. For each part of the exercise you will receive one token. You will receive a token only if your solution contains all 10 cracked passwords. Also, you have to upload your source code via moodle. Put everything in one file named *hw3_ex2* with appropriate extension. You can use a programming language of your choice.

Note1: The attacks you implement might (and probably will) take some time when you run them. Depending on your hardware and your implementation, the brute force attack may run more than 30 minutes, and 2b and 2c attacks more than 10 minutes.

Note2: Don’t forget to upload your source code via moodle, otherwise your solution will not be graded!

Exercise 3: P0wn it

You can't imagine the number of bad developers out there. We just found one and we'd like you to hack his website!
For the two sub-exercises, we provide some hints at the end of the description. You should try to solve them without reading the hints first, you'll gain much more experience this way ;)

Setup

You must first download the docker image containing the website alongside with the database. If you remember the lecture, you should know that having the website and database on the same server is already a sign of trouble…

You can run the web server using the following command:

```
docker run --rm -it -p 80:80 --name ex3 dedis/com402_hw3_ex3
<email>
```

To find out on which IP address the webserver is running, type:

```
docker inspect ex3
```

You should now be able to see the website at the indicated address.
Since the website is super rudimentary, here are the three URLs that you can try:

```
http://<ip>/
http://<ip>/personalities
http://<ip>/messages
```

Fortunately enough, you can access the container:

```
docker exec -it ex3 /bin/bash
```

You should look into the `/root` folder, see what you can find… :)

----------------------- **HINT #1** ----------------------------------

- V2VsbCwgdGhhdCBzaXRlIHNtZWxscyBsaWtlIHNvbWUgU1FMIGluamVjdGlvbnMgYXJlIHBvc3NpYmxlIExvb2sgYXQgYC9yb290L3N0YXJ0dXAuc2hgIHRvIHNlZSBob3cgdGhlIFNRTCBzY2hlbWEgbG9va3MgbGlrZSE=

3a (10p) Relatively easy

In this first part, we ask you to find the secret message written by a super secret agent with the email address “james@bond.mi5”.

You must write a **python3 script** that connects to the IP address of the docker in your machine and **outputs the secret message in stdout.** Your script should use the libraries `requests` (as in the previous homework) and `BeautifulSoup`. The latter is a HTML parsing library enabling you to quickly find the right information in a HTML file. To install it: `pip3 install beautifulsoup4`. You can refer to the documentation here.

Once you think your script is correct, you need to submit it to the com402 website. But, **before you submit your script**, modify it such that **your script tries to connect to `127.0.0.1` and not the IP address of the docker in your machine.** On our side, 127.0.0.1 will run in the same container as the docker web server. One quick way to not forget is to do something like:

```
addr = “<docker_ip> if len(sys.argv) > 1 else “127.0.01”
```

and run your script locally by giving any argument. Our verification script won’t give any argument to your script and it should connect to the right address.

You can then submit it to `com402.epfl.ch/hw3/` as exercise 3a.

---------------------- **HINT #2** ----------------------------------

- 089 111 117 032 100 111 110 226 128 153 116 032 097 108 119 097 121 115 032 110 101 101 100 032 097 032 102 111 114 109 032 102 111 114 032 083 081 076 032 105 110 106 101 099 116 105 111 110 115 032 059 041

3b (15p) Now that you’re warmed up...

This time, you have to find the password of your sworn enemy, namely, the one and only, **inspector Derrick**. He is registered in the database under the name “`inspector_derrick`” but we need access to his password.
Your script should connect to `127.0.0.1` and output the password

---------------------- **HINT #3** ----------------------------------

----

Exercise 4: How does it feel to be downgraded?

(25 p)

In this exercise, you need to get back to the cyber-cafe of HW1. The cyber-cafe still smells of freshly brewed coffee but the newbies have solved HW2 and now make sure to only browse HTTPS websites. You still have access to the router in the cafe, but the traffic you intercept is just TLS-Handshakes and then random bytes, no more free goodies for you.

However, you have been looking online and found out that TLS can also be attackable ;)
Time to set the new plan into practice!

Your luck comes from the common and unfortunate reality of sacrificing security for the sake of usability. You have discovered online that even though researchers work hard to produce more secure and sound communication protocols, vendors try to keep to capability of using older protocols in their browsers, so that users can access “forgotten-to-be-upgraded” web servers. The negotiation between a server and a client browser, in order to agree which version of the TLS protocol to use, is sometimes referred as a “downgrade dance”. But as some say, the older the version of TLS used, the easier the life of a hacker is!

Your goal in this exercise is to trick a client and a server into using TLSv1 for their communication, even though they are both able to speak it up to TLS1.2 and will surely try to do so. Your path to success is to convince the client that the server does not understand versions of TLS newer than v1, and your task is find out how do this.

Setup

1. You have the same setup as in Ex.3-4 HW1 with the *client* docker container trying to connect to our com402 server and the *attacker* container where you can implement a Man-in-the-middle attack to downgrade the connection between the client and the server.
2. To start with, download `run_dockers.sh` file from Moodle. You need to run it with your EPFL (or UNIL ;)) email address as an argument, the script will create the two containers and a subnetwork connecting them. The client container with the IP address `172.16.0.2` will try every 10 seconds to connect to `com402.epfl.ch` **using SSL** and will have the attacker container (`172.16.0.3`) set as its default gateway.
3. As in HW1, you need to use iptables to set up packet forwarding on the attacker and NFQUEUE rules to be able to accept or drop packets. This time `run-dockers.sh` doesn’t come with the iptables rules. Therefore, you will have to install the rules **on the attacker container** by yourselves. You can refer to the the HW1 handout for information on the iptables rules, however, you need to slightly change them to fit this exercise’s scenario. One way to understand if you are in the right track is to use

Wireshark in order to observe outgoing traffic for `com402` (IP address `128.178.156.165`). If not, your rules are most likely incorrect. Yet, observing traffic for `com402` does not mean your rules are complete. **Make sure that you understand what these rules do as they are integral to solving this exercise.**

4. To run your code in the attacker, follow the same steps as in HW1 Ex3-4.
5. As soon as you accomplish your tasks and our server realizes that it has been connected using TLSv1, it will issue a gratitude token in your name. You will be able to find the token in `myprecious.txt` file that will appear on your computer in the folder shared with the Docker containers.

Your daily lifesaver

Wireshark will come in very handy to observe the exchange of packets (or lack thereof) between your computer and the server. One of the many great things about Wireshark is that it allows you to filter the captured packet traces based on user-defined rules. Our `com402` server's IP address is `128.178.156.165` so you might find it useful to filter based on this IP address.

Have fun!

Food for thought (and perhaps for the exam): what other attacks to downgrade HTTPS were performed in the past? What were the patches adopted by the community? Some keywords: SSL stripping, HSTS, Poodle, etc