# Lab-2-2: Passwords and Hashing

This lab continues the topic of correctly handling of user credentials.

**Q1.** Last class we discussed and performed a variety of attacks against hashed passwords. Briefly explain one main problem of using hashing algorithms such as MD5 and SHA-1 to store passwords.

MD5 have no salt and can easily be cracked using a dictionary or brute attack.

**Q2.** Since last class, updates to the SimpleApp.py program include: 1) Use of the SHA-256 algorithm, and 2) Added support for salted hashes. Have a look at the source code and password storage file (passwords.txt) again. Briefly describe the salting method used and what it attempts to solve.

The password is first salted using a hard-coded salt and then hashed using sha256 and then updates the password. This makes it harder for someone to crack the password

**Q3.** Generating a random salt value can be beneficial. How can we program the generation of a random salt in our application? Provide an example of random salt generation using any programming language.

Random rand = new Random();

byte[] salt = new byte[32];

rand.nextBytes(salt);

**Q4.** We now have a better hash algorithm, as well as salted hashes! Has this made any difference to the security of our stored passwords? Why? Alternatively, why not? To help you answer this question, try to crack the new salted hashes using hashcat. Also, document the hashcat command you used.

It has increased security by creating a hashed and salted algorithm.

COMMAND: hashcat64.exe -m 100 <HASH> rockyou.txt

**Q5.** Do some research and find another password-cracking tool. This could be a tool to attack the actual hash, or a tool to attack the password implementation. Provide some information about the tool, include a reference (website) and give an example of how you used the tool to perform some type of attack.

John the Ripper

It is a free and Open Source password cracker similar to Hashcat.

Example of command to crack sha1 password:

John –-format=raw-sha1 rockyou.txt passwords.txt

**BONUS Q7.** It seems like it would be an excellent idea to implement a better hashing algorithm. Using the knowledge and skills you have gained, update SimpleApp.py to use a more suitable algorithm. For example, you could use bcrypt to hash passwords (see: <https://github.com/erlichmen/py-bcrypt>). Document some code and/or your approach.

I changed *import* *hashlib* to *import bcrypt*.

I then changed *m = hashlib.sha-256()* to *m = bcrypt()*