Software Security Final Project

Sang Mai, Paul Ries, Zyra De Los Reyes

Documentation

Our group used the delmar server provided by umsl and worked with C and C++.

The hashing algorithm we used was SHA256:

*unsigned char \*SHA256(const unsigned char \*d, size\_t n, unsigned char \*md);*

Format for password files are as followed:

Plaintext: *username password*

Hashed: *username hash*

Salt Has: *username salt hash*

**TASK 1:**

In the first task1, we used a pseudo-menu system to take input from user. The program asks for a selection of password file:

(1: Plaintext), (2: Hashed) or (3. Salt Hash)

The program then ask user if they want to:

(1: Create an account) or (2. Log in)

The program then asks for username and password from user.

Creating an account: places the username/pass/salt/hash into the corresponding files (plaintext.txt, hashed.txt, salt.txt) based on formats provided.

Logging in: grabs data from file and compares with data inputted from the user. The password from user is hashed for selection 2 and 3.

**TASK 2:**

Program asks user for length of passwords(3-8), program then asks for number of passwords you want to create.

For plaintext: program creates username and password with random generation, and then puts to plaintext.txt.

For hashed: program creates username and password, hashes the password, and then puts to hashed.txt.

For salt: program creates username, password and salt, hashes password+salt, and then puts to hashed.txt.

**TASK 3:**

Program accepts a password file(hashed.txt or salt.txt) and a max password length. The program then uses brute force permutations(aaa->aab), hashes these permutations and compares them with the hash from the file.

For the case of the salt, we grab the salt from the file and append it to all the permutations before hashing.

If the hash does not match, it continues to the next permutation until a match is found.

**Struggles**: we had problems getting the openssl library to work at firstt, but after many attempts finally got it working. We first used strcat, but switched to strncat to be more secure, which was pretty easy as we knew the amount of items we wanted to copy.

**Conclusions:**

-Salt adds a lot of complexity to cracking the password, we only had 1 byte for the salt, but it still added time to cracking.

-Longer passwords require much more time for brute forcing because it forces attacker to go through much more permutations.

-Cracking passwords are doable, but you want to make your password or hashing method complex enough so attackers have to spend a lot of time attempting to crack it.