03/17/2022
Jacob Glik
Writeup

Security Vulnerabilities in Saving Passwords to a Database

User oriented programs used to save user passwords to a database so that they could compare the passwords on file with the one the user enters to validate the user, and let them into their account. This method was found to be unsafe because if the database had a breach or was leaked: suddenly all those users lost all security. This method was also unpopular amongst the privacy oriented: claiming that they were uncomfortable with anyone, even the company that secures their account, knowing their password. Corporations had to adapt.

Hashing was invented in the late 1940s but didn't become commonplace in the corporate world until the early 1970s. Systems engineers realized that they did not need to know the password to validate it. If they could convert the password into some other form that could not be converted back, but only that password could be converted to that string of numbers and letters, then they could authenticate the user without knowing the correct password. Even Though this system was genius, converting passwords into password-hashes and not being able to reverse the process, black hat hackers were able to find a way around the system yet again. If one was dedicated: they could use the same hashing algorithm that the corporation used, and calculate hashes of random passwords until they generate a table of all the possible hashes, called a rainbow-table. At this point, finding a password that maps to a hash becomes lookup time, which is not time at all.

Corporations decided: it was time to start salting their hashes. Salting a hash is when you calculate a random value and add it to the result of a hash each iteration of the hashing process. You must also save this random value along with the hash for future use if you ever want to get to the fine hash again. Salting tries to negate the devastating effect of a rainbow table by changing the password each step of the hashing process, thus changing the hashing process itself. Therefore if a rainbow table is to be used, it must be recalculated with the obtained hash in order to stay effective. This became standard in the unix system in the late 1980s but is still the way we store our passwords today.

I propose we take another step to security: the final step. The weakest link is always what is attacked during a security breach. In a security system, the weakest link is the file or database that stores passwords, password-hashes, salted password-hashes, or any other combination of these general practices (pepperd salted hashes, pickled salted hashes, peppered pickled salted hashes, etc…). The issue is the file itself. It is time to stop storing any information about the user's password entirely. In the new system, the users information will be written to a file, and the file will be encrypted with their peppered pickled salted hashed password, the salt, the pepper, and the pickle jar, will all be stored, but the final product (peppered pickled salted hashed password) will not. In order to authenticate a user they enter two pieces of information: a password and some type of username (email, name, etc…), the password is then converted to the final form using the stored salt, pepper, and one of the pickles. The system then tries to decrypt the user file and if successful will attempt to validate the username (email, name, etc…) to confirm that the file was decrypted to its original form successfully. Once this is complete, we have proven that the password - username combination is valid for this user. Without storing the final product, without anything to leak, rainbow tables become truly useless.