Ryan Solomon
Kevin Kosta
Erin Wilhjelm
Dorian Meade

Mini-Project 1 --- Salt/Pepper Hash and Web Exploit Prevention

GitHub: https://github.com/erinwilhjelm/Security
https://github.com/erinwilhjelm/Security2
Website: https://software-security-271021.appspot.com/

Initiating exploits can be extremely easy or somewhat challenging, depending on the security of the system being penetrated. Some specific types of exploits are: Cross Site Scripting, in which a user could add client-side scripts to initiate certain commands within the application, Indirect Object Referencing, where an attacker exploits an internal object within the application, and even User Enumeration, in which an attacker is able to brute force to try and figure out users that are a part of the system.

In our case, the application is url exploitable, meaning an attacker is able to modify the url and still receive sensitive data that is not supposed to be on the page. For example, after a user logs out, an attacker could still add "/profile" to the url and receive all data about that person's profile. Some other exploits that could be done are the ones mentioned above, like XSS, IDOR, and User enumeration.

There are many differences between password hashing and password encryption. Password hashing is a one-way operation, while encrypting is a two-way function. For hashing, once you apply the certain hash function to the password, there is no way to get the original string back. The one way someone could get the original string back is to create what is known as a "collision," which is essentially trying to find a password string that provides the same hash as the original. Encrypting is different from hashing in the sense that it is two-way, meaning you can decrypt the encryption to receive the original string if you have the given key.

Encrypting a password is a lot more dangerous to the overall user because an attacker could potentially get access to the stored key in the application and use it to access the passwords in whatever database is being used. This sort of attack is impossible with a hash because it is much more difficult to reverse engineer a hashed password.

Salting and peppering passwords enable very specific things within the application. A salt is essentially a randomly generated value, usually an integer, that is stored alongside the password hash to make it extremely challenging to use a hash table to crack the password. Since each password contains its own salt, each individual password must be brute forced in order to crack. The one underlying issue is that the salt and the passwords are stored in the database, so a compromise on the database’s side means losing both the salt and the stored passwords.

A pepper is similar to a salt in the sense that it is a stored value within the application, but the main difference to the salt is that it is not stored within the database. It is usually placed somewhere in the application’s source. Peppers are created this way in order to ensure that a compromise of the application’s database does not mean the entirety of the stored passwords can be brute forced and stolen.