Lorenzo Cavada Mat. 220502

Second exercise of OffTech about SQL Injection. More information about the exercise can be found at this link. The code used in this exercise can be found here.

The flaw.

The main security flaw was represented by the lack of proper input validation and sanitization in the application. This carelessness made possible for a malicious user to act as an authenticated user to perform action which look like if they were performed by an employee.
In particular the problem was related to the way used to create the query. Basically it was a simple string concatenation. The user input were read, added into the query throw string concatenation, and then the result was executed using the `query()` function which does not perform any check on the actually query that is going to execute. The flaw is spread all around the application, starting from the login form till all the other option that include any user input giving to the attacker a large attack surface.

The fix.

The solution to this flaw is pretty straightforward. To avoid any future exploit I proceeded to add some sanitization whenever a new user input was read. To sanitize them I used the `prepare statemend` function of PHP which include some sanitazation on the imput avoiding the risk of injection. In this way the string passed as parameters get satized adding an escape value in front of some sensible character such as <'>, <> and so on. A version of the fixed code with the atch is available with this memo.

Thoughts on the breach.

The breach was for sure serious especially because no visible traces were left by the malicious user because and each command looked like if it was performed by an authorized user. Is so impossible to distinguish a valid transaction from a malicious one. In some database is possible to see a register of all the commands performed on the database, if this should be the case may be possible to recreate the history of the change into the database and reverse them to recreate the original state. If not present then we should rely on backups in order to restore a trusted state of the database. In this analysis I focused on the SLQ Injection problem but should be notice that the application comes with many other vulnerabilities and flaws:

- Big numbers are not handled in a proper way, PHP treats big integers as float causing some problems in operation with large numbers (more info here and a possible solution here).
- The service used to send money between employees does not handle cases of homonymy, the list should include some other unique identifier such as the ID or the email.
- The service used to send money between employees will crash if any employee is chosen from the list, proper input validation should always take place.
- The database store the employee password making it possible for an honest-but-curios administrator to see them, hashed password should be saved in the database.

- User id and password are shown in clear in a hidden field in the user page making it possible for a malicious user to see the password if an employee forgets to log out before leaving the workstation.