Mark Kardash

26 September 2023

SDEV350 6380

Professor Carl Eyler

1. Explaining Vulnerabilities:

After reviewing the information presented to me about the supposed application, I can see that there are indeed some concerning security issues with it. The list of employee data given after entering just the ID seems a bit excessive, and could compromise user privacy. Not all of this data is needed by companies. It is highly doubtful that a company will ask for an employee’s nickname or salary (except, maybe, for a salary review). It is also safer to only show the employees’ SSN to authorized individuals in the company, such as the CEO, rather than just anyone who enters an ID into a field. Therefore, this application needs to have a few privileges restricted.

However, too many privileges are far from the only issue present in this app. Perhaps the greatest danger is the fact that it can be modified from the inside. The logic behind this can be understood when looking at the code that is flagged as suspect. Let us review the code:

$conn = getDB(); $sql = "SELECT id, firstname, lastname, salary, birth, ssn, phonenumber, address, email, nickname, Password

FROM data WHERE id= '$input\_id' and password='$input\_pwd'"; $result = $conn->query($sql))

As can be seen above, everything is written in a single, long SQL statement, and then executed as a query. This sort of syntax makes it very easy for a hacker to mess with the code. For example, a cybercriminal can get unauthorized access to data by injecting a code string of their choice. For instance:

$conn = getDB(); $sql = "SELECT id, firstname, lastname, salary, birth, ssn, phonenumber, address, email, nickname, Password

FROM data WHERE id= '$input\_id' or ‘1’=’1’ and password='$input\_pwd' or ‘1’=’1’"; $result = $conn->query($sql))

Above, the hacker makes it so that the system will EITHER display the data if it has the desired username and password, OR display data if the condition injected by the hacker is true. Since the condition ‘1’=’1’ is always true, instead of displaying data for an employee with a specific username and password, the system now displays data for every single employee in the database, essentially leaking private information, and harming the company. The conditions are NOT supposed to be included in the same query. There are no prepared statements, which leaves the data completely unprotected from attacks.

Now that we are done with the main code itself, we move on to the “Retrieve Code” portion. First and foremost, only specific individuals should ever be able to retrieve code from an application. This is because the source code belongs to the developers who created the application, and can be viewed only by those who have administrative privileges granted to them. In the retrieval query itself, we witness the exact same mistake seen in the previous code:

$conn = getDB();

$sql = "SELECT id, firstname, lastname, salary, birth, ssn,

phonenumber, address, email, nickname, Password

FROM data WHERE id= '$input\_id' and password='$input\_pwd'";

$result = $conn->query($sql))

What we see is that the id and password conditions are again hardcoded into one large “SELECT” statement. Hardcoding our data is practically never a good idea, as it becomes an open invitation for hackers to inject malicious code. This can be demonstrated with an example similar to the previous one:

$conn = getDB();

$sql = "SELECT id, firstname, lastname, salary, birth, ssn,

phonenumber, address, email, nickname, Password

FROM data WHERE id= '$input\_id' or ‘a’=’a’ and password='$input\_pwd' or ‘a’=’a’";

$result = $conn->query($sql))

Above, the hacker has injected their own condition, ‘a’=’a’, which is, as we know, always true. Thus, just like in the previous sample of code, the program immediately gives access to data for all employees, rather than for the particular one requested.

The exact same thing can be seen in the “Update Code” sample. The names, emails, and other information are to be written SEPARATELY from the main query. I should also point out the spelling error in the “Retrieve Data Code” section, where the last name is spelled as “$lirstname”, although that may have been the user’s choice for a sample last name.

When it comes to protecting user data, best practices for this include, first and foremost, a set of clear cybersecurity policies within the company, and a well-thought-out strategy for managing the data. Without clear rules about these things, other data protection strategies will be of no use. The next best practice is the restriction of unnecessary privileges. Administrative privileges should only be given to administrators, and any other persons authorized by them. This will reduce the possibility of data leaks, injection, and other problems. Data and login credentials must be well-protected by techniques like encryption and firewalls, and employees must receive well-designed training on cybersecurity. Also, relating to the privilege restrictions, the company must only provide access to necessary data, so as not to reveal valuable private information to the wrong users.

When it comes to changing a password, it is recommended to:

1. Come up with something completely new, rather than reusing a password that has been used in the past (ex. One of the last 4-5 passwords).
2. Authenticate the account with multiple factors. Multi-factor authentication helps ensure that it is not easy for a hacker to take over an account and assume someone else’s identity.
3. As always, do not make any password too easy. Use plenty of special characters and numbers, and keep a minimum length of 8-12 characters. This helps protect the account from password-guessing attacks, such as brute force attacks.
4. Set up a password expiration time. Make it so that your password could expire after a set period of time (ex. 3-4 months). The practice ensures that hackers do not have a lot of time to crack the same password, since it is replaced with a new one each time.
5. Using password managers will help you keep your passwords organized and taken care of. This could also protect from attacks.
6. Specific Recommendations for Problems Detected:

Practically all the things I will recommend have to do with making the queries safe, and ensuring employee data protection. The recommendations are as follows:

1. Restrict user privileges. This is perhaps the first thing to be done. As said before, administrators are supposed to be the only ones with admin privileges. In addition to this, only authorized personnel (such as developers and testers) should be able to retrieve code from the application and modify it. This will prevent random persons from accessing the code and performing cyberattacks. As for regular users, it is perhaps wise to keep some of the more sensitive information (Ex. SSN, Salary) out of reach for them, and only available to individuals like the CEO or the CFO of the company. As of right now, it looks like every single person who enters an ID into a field of the form can gain access to this data. Its removal can be performed by simply removing the specific information from the “SELECT” query in the code.
2. Use prepared statements. A prepared statement puts parameters on the queries that are to be executed. As such, the ID and Password requested would be written separately from the “SELECT” query, in a string which would then be included in a Prepared Statement, and executed through it. Since the data requested will now be enclosed in the prepared statement, hackers will not be able to perform SQL injection with it, and the possibility of a leak will be significantly reduced.
3. Use strong and protected passwords. As mentioned above, the best passwords contain both numbers and special characters, besides letters. Letters included must be both capital and lowercase, and the password must not be anyone’s private information (ex. Birthdate). The best password length to use is around 8-12 characters. A recommendation I would give is that the password expires every 4 months, so that the user needs to update it again. Also, as said before, a different password must be used than the last 4-5 passwords. These measures will reduce the chance of brute-force attacks, and gives less time for the attacker to guess the password. Another technique that will protect from brute-force attacks is multi-factor authentication. The ability to authenticate with more than one method (Ex. By phone AND by email) will make sure, to the extent possible, that the account is being accessed by the correct user.
4. Encrypting form fields. In addition to restricting the privileges given to users, we can also limit unwanted access in another way. By encrypting the fields in the user form (That is, turning them into a ciphered text of “random” characters), we can ensure that only the individuals we choose have access to certain pieces of information. The decryption key can be given to them if needed.
5. Implement firewalls. A firewalls across the entire system would monitor data coming in and out, ensuring that no unnecessary/malicious code is present. It is essentially a digital wall of protection for the company.

The changes all of these measures will do to the application will not cause it any damage. More code will simply be added, and despite the source code becoming more complicated, it is acceptable, because it is done to protect the software from attacks.

1. As we saw earlier, SQL Injection is one of the main problems in this supposed application. It was demonstrated previously that injection could occur here if a condition that is always true (Ex. ‘1’=’1’ or ‘a’=’a’) is inserted into the “FROM data WHERE id” portion of the code. This will give the user access to the entire database, instead of just one employee’s information. To achieve SQL injection through the fields of the form, we can place the words “username’ or ’a’=’a” or “admin’ or ’a’=’a” into both the ID and Username fields. In combination with the altered query inside the code, and the fact that the condition ’a’=’a’ is always true, the attacker will successfully infiltrate the system by logging in if the code is not protected.
2. The best way to fix the dangerously weak code is to add prepared statements. In doing that, we must perform the following actions:
3. The ID and Password values should be REMOVED from the main query, and placed into a separate “request.getParameter()” string (Ex. String “Credentials”). Since there are two values, they may be separated into two strings (Ex. String “Emp\_ID”, String “Emp\_Pass’). Their values in the main query should be replaced with “?”.
4. The main query should then be placed into a prepared statement (Ex. “PreparedStatement prpdst”).
5. The values for the credentials must then be set using a “prpdst.setString()” command, like this:

Prpdst.setString(1, Emp\_ID)

Prpdst.setString(1, Emp\_Pass)

1. Finally, the prepared statement must be executed using the “executeQuery” command.

By creating prepared statements, we make the queries and credentials protected separately from each other. Thus, a cybercriminal is no longer able to inject their own conditions (Such as “’1’=’1’” or “’a’=’a’”) into the code, making it safe and proving prepared statements to be an effective strategy against attacks.