**SQL Injection with PHP**

**Student**: Patrick Walsh

**School**: University of Maryland Global Campus

**Date**: 4/24/2021

**Course**: SDEV 350 7380

**Professor**: Reggie Haseltine

Existing Application Description:

An existing PHP-based application connects to a database providing functionality for retrieving table information for a specific employee. The application also has the ability to update specific user fields. In both cases, a Web-based form is used allowing a user to search for an employee and then either display their employee information or update it. The web-form display functionality asks the user to enter a specific employee ID and then retrieves the following fields:

• Employee\_id,

• firstname,

• lastname,

• salary,

• birthdate,

• SSN,

• phonenumber,

• address,

• email,

• nickname,

• Password

The update form allows the user to enter a specific employee ID and then update most of the fields in the list above. As you look at this list, you should definitely be concerned. Part of this exercise will be to identify those concerning fields and make recommendations for changes. However, you were also given access to the PHP code and flag the following code as being suspect.

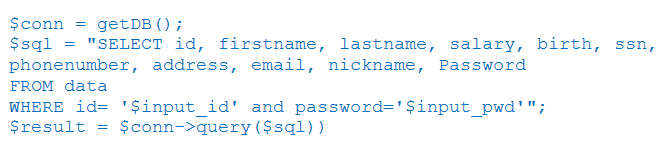
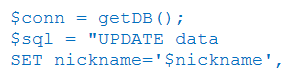


Figure 1. PHP Retrieve Data code



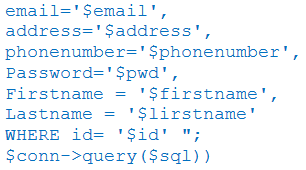


Figure 2. PHP Update Data cod

**Questions:**

1. *As you review the overall functionality of the application. What concerns do you have with the data and display of the data. For example, are there any privacy concerns? What are best practices for protecting private data? What are best practices for changing a password? Note: you can ignore the SQL Injection issues in this discussion as we will be addressing that in a separate requirement.*

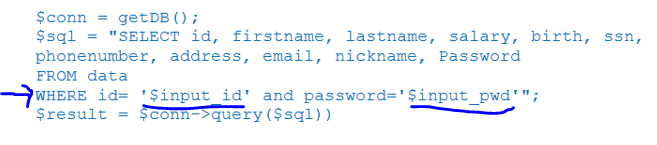
There are some major security issues with this web application and the way it is setup. First of all, the data fields are all stored in a single table rather than dividing the fields into separate, related tables with Primary Key/Foreign Key relations. Using multiple tables in this way follows the Relational Database model, offering numerous advantages, including the ability to arrange tables according to business concepts (IBM Cloud Education, 2019). Another security issue with this database setup is having the Password field displayed along with other sensitive personal information like date of birth, address, and Social Security Number (SSN). Passwords should be encrypted and not viewable by anyone other than the user, not even an administrator (Carnegie Mellon University, 2007). Additionally, this web application allows users to update personal information of employees, which is a major security concern since this violates the Principle of least privilege (Gegick & Barnum, 2005). The ability to update personal information should be given only to users with the express need to do so, such as an accounts administrator. Moreover, this privilege should be granted to roles rather than directly to users (Gegick & Barnum, 2005). For general users, a basic read-only role with access to a limited number of fields may be appropriate, such as the ability to view the employee’s first and last name, phone number, and email. Other, more sensitive fields such as SSN and salary should be restricted since these fields are highly personal in nature and could be misused by malicious actors to commit identify theft and other fraudulent activities. To protect their password, users should use a strong password with a combination of letters, numbers, and special characters, and should not use the same password for multiple accounts (Carnegie Mellon University, 2007). If a user suspects that their account has been compromised, they should immediately change their password and report the incident to an information security officer or administrator (Carnegie Mellon University, 2007).

1. *Now that your concerns about the application have been documented, what specific recommendations do you have to address your concerns? Be specific with your recommendations. You should consider items such as using roles to restrict access, limiting access to the form, encrypting fields, applying security controls and others. Discuss if the recommended changes will impact the functionality of the application and if so, why that is Okay.*

I recommend that the database first be redesigned by breaking the data fields in the single table into multiple, relational tables associated by Primary Key/Foreign Key pairs. This will not only ensure that the database is following a good relational database model, but also provide enhanced security. For example, one table could store the employee ID as the Primary Key, and the first name, last name, email, and phone number as separate fields. Another table could then be tied to the first table through the employee ID as a Foreign Key relation while storing the password as a separate field. The database administrator can then govern access to these separate tables through roles. One role could provide read access to the first table to view general information about employees, while another role could provide read access to an encrypted version of the password, or a hash of the password so that the actual password cannot be leaked. This approach provides access control so that employees’ sensitive information and passwords is not being compromised. Additionally, the web application could restrict access to viewing any employee information without first being authenticated, such as through a separate login form, or a network restricted protocol which prevents users outside the network from viewing the information.

1. *Review the code and determine if SQL Injection is a problem. Describe specifically how you would test to see if SQL injection was a problem. For example, show exactly what you would input into the form fields to determine if SQL injection was a problem. Discuss the code issues pointing to where the SQL injection could happen.*

This application is vulnerable to SQL injection. The easiest way to carry out an SQL injection attack is to insert data into the query form which will always resolve to true so that the attacker can bypass any conditional statements found in the query form. For example, the statement “if 1=1” will always resolve to true, so this kind of syntax can be inserted into a query to launch an SQL injection attack. The code builds an SQL SELECT query that returns all fields in the table (this, in itself is a security issue) as long as the ‘input\_id’ and ‘input\_pwd’ match the user ID and password. See the code snippet below:



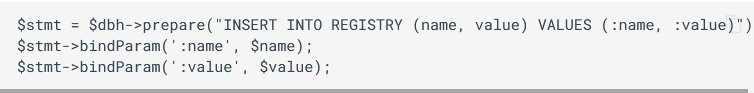
Assuming that a user is entering a username and password in two separate fields of the web application, an attacker could get around the password restriction by adding the following text to the end of the password field:

‘ OR 1=’1

This extra text at the end of the password gets added to the query, essentially tricking the database into reading the query as “if the user ID and password are correct, or if 1 = 1, return the requested information to the user.” This ensures that the requested information is always returned.

1. *Now, that you have identified the code issues, describe what you need to do to fix it. You don’t have to rewrite the code, but you do need to provide specific mitigation strategies, explain why the strategy works, and provide an example code (in PHP) that would resolve the issue.*

To prevent malicious text strings from manipulating database responses and causing SQL injection, the first line of defense is to use Prepared statements to execute parameterized database queries (OWASP, 2021a). Parameterized queries through Prepared statements are defined at the application development level and are used to help validate that the queries being sent to the database contain expected input. The syntax for Prepared statements varies between different programming languages, but PHP uses bindParam() to enforce parameterized queries (OWASP, 2021a). The code snippet below (OWASP, 2021b) shows the syntax for creating a Prepared statement through bindParam() where a name and value are being inserted into a form:



The original PHP code could be rewritten to something like this:

*$sql = “SELECT id, firstname, lastname, salary, birth, ssn, phonenumber, address, email, nickname, Password*

*FROM data*

*WHERE id= (:input\_id) AND password= (:input\_pwd)”;*

*bindParam(‘:input\_id’, $input\_id);*

*bindParam(‘:input\_pwd’, $input\_pwd);*

**References**

Carnegie Mellon University. (2007, December 1). *Guidelines for Password Management - Information Security Office - Computing Services* - Carnegie Mellon University. https://www.cmu.edu/iso/governance/guidelines/password-management.html

Gegick, M., & Barnum, S. (2005, September 14). *Least Privilege | CISA*. Cybersecurity & Infrastructure Security Agency (CISA). https://us-cert.cisa.gov/bsi/articles/knowledge/principles/least-privilege

IBM Cloud Education. (2019, August 6). *Relational Databases*. IBM. https://www.ibm.com/cloud/learn/relational-databases

Open Web Application Security Project (OWASP). (2021a). *SQL Injection Prevention - OWASP Cheat Sheet Series.* OWASP Cheat Sheet Series. https://cheatsheetseries.owasp.org/cheatsheets/SQL\_Injection\_Prevention\_Cheat\_Sheet.html

Open Web Application Security Project (OWASP). (2021b). *Query Parameterization - OWASP Cheat Sheet Series.* OWASP Cheat Sheet Series. https://cheatsheetseries.owasp.org/cheatsheets/Query\_Parameterization\_Cheat\_Sheet.html