Introduction à la sécurité TP 3.5 - Injections SQL

Maciej Korczynski & Simon Fernandez

Novembre 2022

Objectives

- Deploy a simple PHP webpage
- Deploy a simple MySQL database
- Query the database with PHP
- Discover the basic principles of an SQL Injection attack and simple remediation methods

Setting up the target website

Installation

On your target VM, install MySQL and the packages needed for Apache2 to handle PHP:

```
$ apt install mysql-server php libapache2-mod-php php-mysql
```

- Enable PHP for Apache2 with the `a2enmod` command.

MySQL setup

Configuration

MySQL is a simple yet powerful database system. It is available on almost all OS. By default, the databases are only served on `localhost`, but this can be configured in `/etc/mysql/mysql.conf.d/mysqld.cnf`.

To interact with a MySQL database, we need to connect as a specific user. The system `root` user can connect to MySQL without any password and access all the databases, but for security reasons, we will create a new MySQL user, with a password.

- Using the `CREATE USER` command, create a new user called `site`, with a strong password.

By default, this user will only have the permissions to read data. Give it write permissions using the `GRANT ALL PRIVILEGES ON * . * TO 'site'@'localhost';` command. You can now log out of `mysql`, and log again as `site`.

Architecture

MySQL stores data in objects called databases. Each database can contain multiple tables. Each table is made of entries.

- Create a `website` database. It will contain all the tables used by our simple website.
- Inside this database, create a `users` table that will store usernames and passwords (in clear-text). Then manually fill it with a few entries.

PHP

PHP is a programming language used to have server-side dynamic pages, meaning that the webpages are not static, and are dynamicaly generated on the server when a client requests them. This allows HTML pages to contain dynamic data, like the content of a database for example.

If a webpage ending in `.php` is queried, Apache will automaticaly check if it contains PHP blocks to execute them and build the HTML page before serving it to the client. For example, if the `hello.php` file contains :

```
<html>
<body>

<h1>My first PHP page</h1>

<?php
    echo "Hello World!";
?>

</body>
</html>
```

when a user queries this page, Apache will detect the `<?php ... ?>` block, execute it like any programming language, and all outputs are inserted in the page. The resulting page that will be served to the client becomes :

```
<html>
<body>

<h1>My first PHP page</h1>

Hello World!

</body>
</html>
```

Forms

HTML pages can contain fields, sliders, checkboxes, to build forms, allowing the user to send data to the server with a POST request.

- What is the difference between a GET and a POST request ?

The content of the POST request can then be processed by PHP to build a webpage depending on the content of the fields.

- Using the `<form>` block, build a small webpage called `index.html` allowing users to send a user and a password to a `login.php` page (this page will be built later on).

MySQL queries from PHP

The PHP language can query MySQL databases to get data. This is not the case by default but the `php-mysql` package that we installed in the beginning will give us all the needed tools to do so.

The following PHP code will build a simple query to a local MySQL database, to check if it contains a given username and password :

```php
// The parameters to connect to the database
$servername = "localhost";
$username = "site";
$password = "mypassword";
$dbname = "website";

// Create connection to the database
$conn = new mysqli($servername, $username, $password, $dbname);

// Check if the connection worked
if ($conn->connect_error) {
   die("Connection failed: " . $conn->connect_error);
```

```
}

// Build a SQL request from the parameters of the POST request
// SELECT * FROM users WHERE username="bob" AND password="azerty"
$sql = "SELECT * FROM users
        WHERE username=\"" . $_POST["username"] .
        "\" AND password=\"" . $_POST["password"] . "\"";
// Send the query to the database and get the result
$result = $conn -> query($sql);


// Check if some entries matched
if ($result->num_rows > 0) {
    // If at least one entry matches inside the table
    // Print all the entries that match
    while ($user = $result->fetch_assoc()){
        // Print the name of the user
        echo "Valid user : " . $user["username"] .
             " - " . $user["password"] . "<br>";
    }
} else {
    // If no entries were returned, this combination of username
    // and password do not exist, so the login is denied.
    echo "Invalid username or password";
}

// Close the connection to the database
$conn->close();
```

- Write a `login.php` page containing this code and the HTML needed to display the login result to the user
- Deploy your simple website so that your client VM can query the webpage, fill forms and get the results (go back to TP1 if you forgot how to do it)
- Check if the usernames and passwords that you put in your `users` table work well.

SQL Injection

SQL is a powerful tool to manage databases, but it can have flaws if it is not deployed with care.

SQL injections are a type of attack that use the fact that PHP (or any programming language) must craft a SQL query as a generic string, and then send this string to the server. So if someone can modify the query string and inject malicious code, the effects of the query can be unexpected.

- What would append if the user set the `username` field in the form to be `"` ? Why ? What query would be sent to MySQL ? (don't hesitate to check apache logs or use PHP to print the query before it is sent)
- What could an attacker write in the `username` and `password` fields to get all the users and passwords in the table ?
- What else could an attacker do ?
- How could we fix our code to avoid those security threats ?