Project 2

Verification based SQL Injection Defence

The following is the necessary information required to know about a website to defend it from a SQL Injection

1. Be Familiar with the threats to the security.

   The user input are the real threats to the security of the system. In case of an SQL Injection attack, the attacker can enter the queries in the form of special characters thereby making it a Boolean statement, which will give the attacker, access to the database. After getting access to the database, the attacker can tamper the data i.e. delete, update or insert false information. The input is given from the HTML or the client side of the application.

2. Vulnerabilities related to the development of the website system.

   In the code of login.php consider the following line:

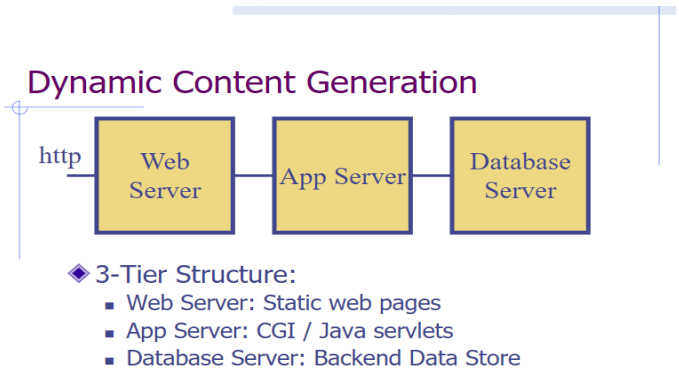
```
16  $query=mysqli_query($connection,"SELECT * FROM userpass WHERE user='$user'AND pass='$pass'");
```

Here, the user inputs are given by the client and are cross checked from the database, if the inputs match the database columns of user and pass, the user or the client is given access to the database. So, for this, if the user and pass entries are not sanitised or verified properly, the attacker can take advantage of this and enter the special query which can lead to the attacker getting access of the database.

Structure of the Web Application:

Whenever we access the internet, we require a web browser. The web browser has a program running in it for accessing the internet. This program or application is called a web application.

Consider the following figure which shows the proper working and the different layers in a web application. A web application is made of 3 tiers:



1. Web Server.

   This tier gets input from the user and the processing is shown. Here in this layer, HTML and JavaScript are used as the languages to help interact with the user. This layer is processed by the web browser which also analyses it. This layer is also known as a GUI or a Graphical User Interface. The main aim of this layer is to get input data from the user and send it to the App server.

2. App Server.

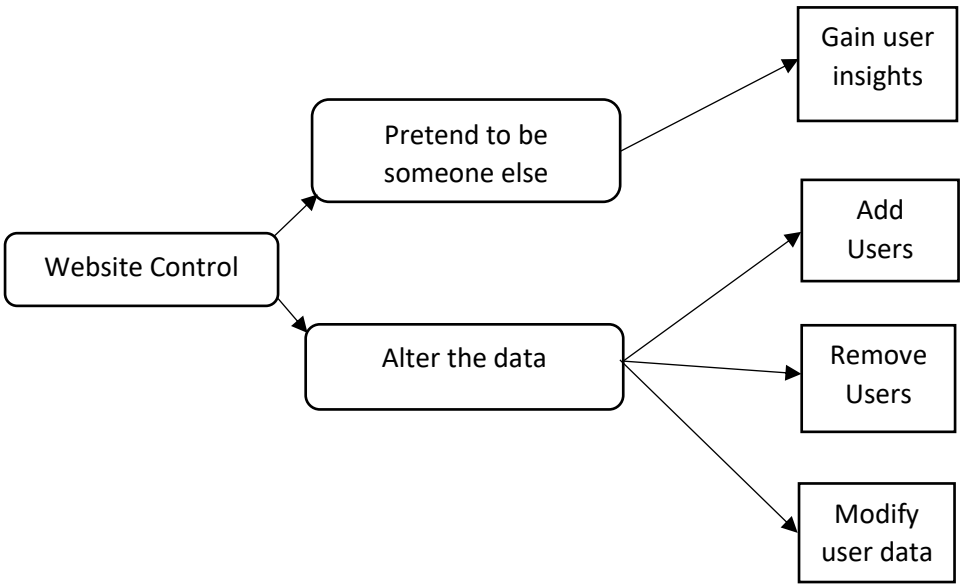
   This tier acts as a hallway or a gateway between the web server and the database server. The user input is processed here in this tier with the help of a validation code. The main purpose of this is to get the data from the user input via the web browser and process it by checking the credentials with the database server. If the credentials are authorized then the user is given access to his data. The programming language used here in our project is PHP.

3. Database Server.

   This tier only stores the data that has been generated by the users when they create their accounts. The only role of this database server is to keep a record of the number of users and their sensitive data. The app server takes care of the validation and takes the desired action based on the data present in this database server. This server has no security checks; hence it is always at stake. The attacker can alter the database if he gets access to this server.

Attack Tree:

Attack trees are used to describe the different possibilities in which a system can be attacked. Designing an attack tree can help us get insights of the necessary counter steps that are required to prevent the attacks to the system in future. The following attack tree diagram shows the ways in which an attacker can gain access to the website and control the data involved.



Here, the attack tree shows the website control as the main primary goal. The attacker gets control of the website and then the “pretending to be someone else” and “alter the data” are the next sub goals. The attacker, after gaining access to the website can either pretend to be some specific user for which he has hacked the system or can alter the database by accessing the database server. There are two ways to do this. The first method being giving user input in the form a Boolean statement which is true always and thereby giving access to the attacker. The string which is given as input i.e. username and password: 1’ or ‘1’=’1. Using this the attacker gains access to the first user. The second method is gaining access to a specific user i.e. 1' or '1' = '1’ and user<>'user1.

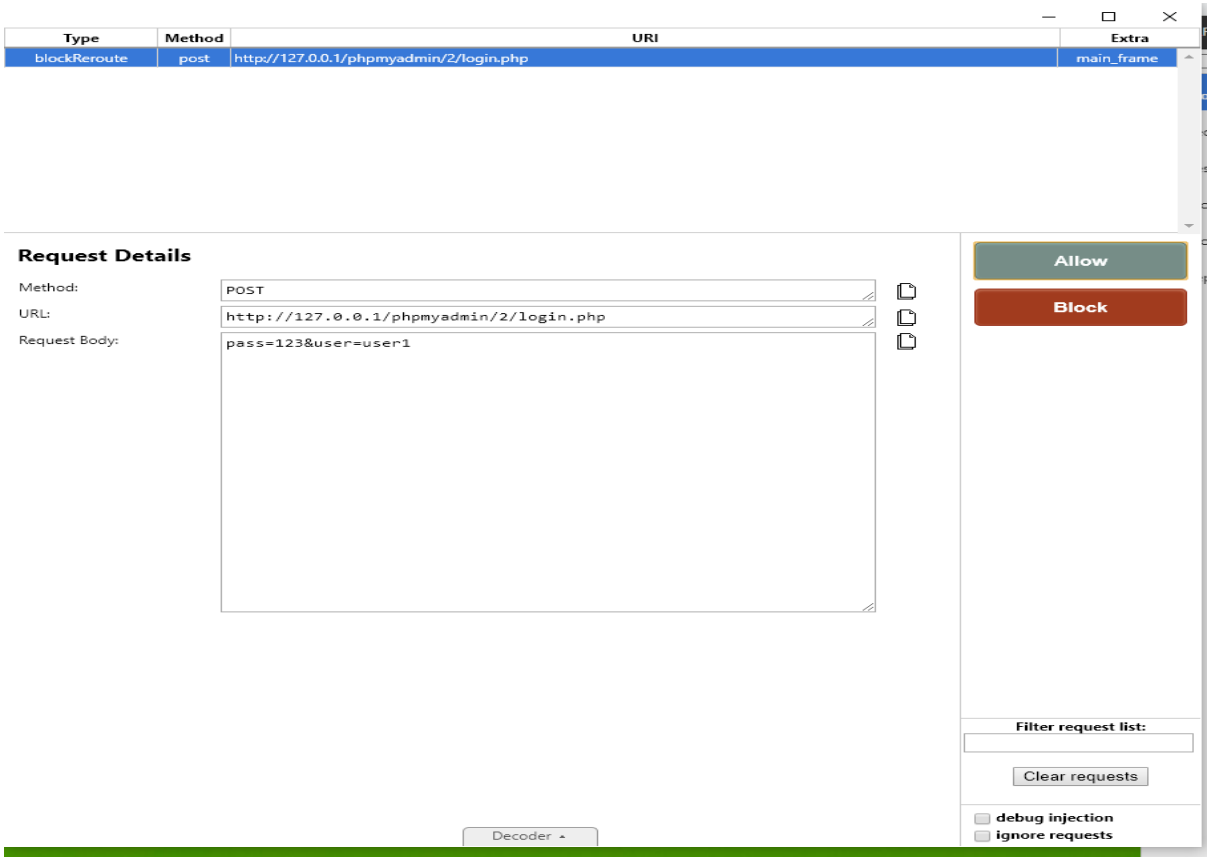
The attacker can delete an entire database after gaining access or can modify it completely as per his liking. Deleting a database can be done by ‘; DROP TABLE and inserting fake users or data can be done by: INSERT INTO and ‘; UPDATE. Here in all the inputs the attacker needs to add the special characters such as ; : ‘ “ . Here the role of verifying the data comes into play. We can set a certain condition on the login such that the user can enter only alphabets and numbers, and only a specific set of special characters are to be allowed. For the validation we make use of JavaScript. The input string can be checked to match the regular expression.

```
<script>
function validate()
    {
        var bool=/^[-\w\.\$@\*\!]{1,10}$/.test(document.getElementById("user").value);
         var next=/^[-\w\.\$@\*\!]{1,15}$/.test(document.getElementById("pass").value);
        if(bool==false || next==false)
        {
            document.getElementById("form").action="invalid.php";
        }
      }
</script>
```

The validate function makes a check for both, user and password. This is used for matching the inputs with the regular expression. Even if one of both, user or pass is false the validation will fail and the attacker will not be able to access the database. We are making use of bool for the username input and next for the password input. There are some special characters specified are the only ones which are allowed. If the attacker enters something other than the ones specified, an error will be thrown and validation will fail thereby redirecting the user to the error page. In the similar way, the password field can contain up to 15 characters.

There are two ways in which the malicious strings can be passed to the SQL database. The one being client side which has been taken care of. The other being browser side.

Tamper Chrome (Browser Side):



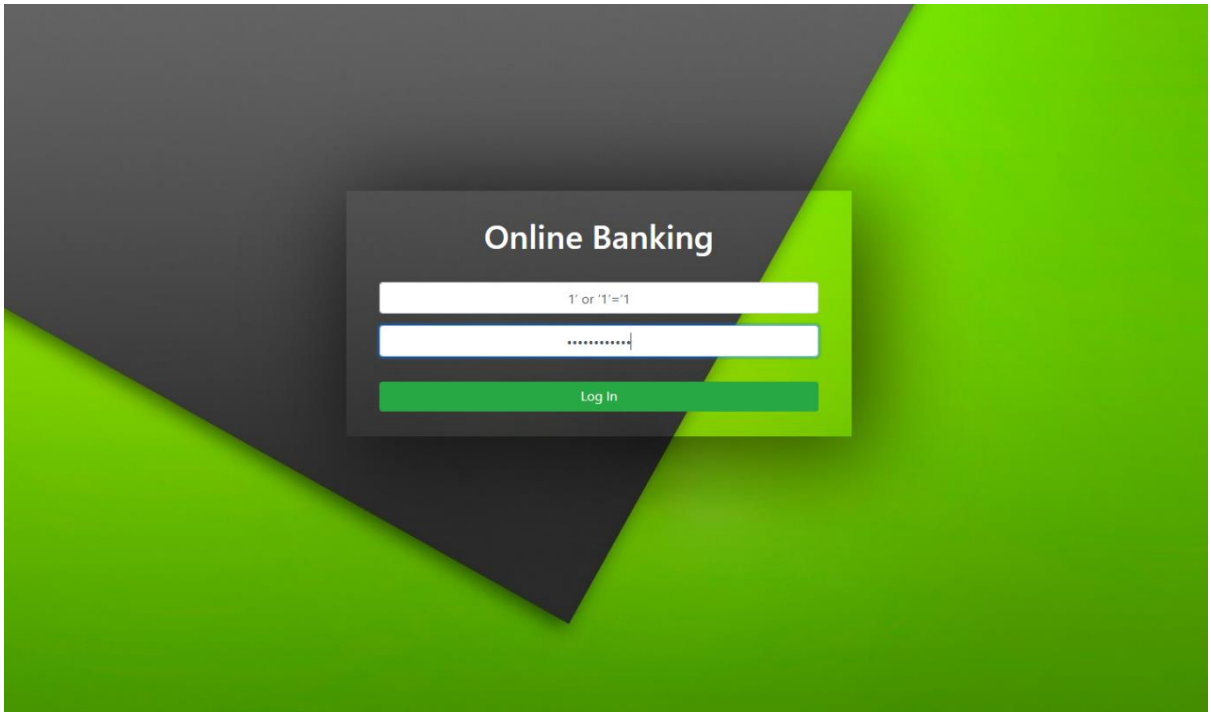
In Google Chrome, there is an extension known as Tamper Chrome (fig above). In request body, one can change the pass and user to the string which will bypass the security check by always being true and giving the attacker access to the data.

To prevent this from happening, we can add an extra condition in the login.php where the data will be revalidated to prevent the tampering after it has been passed from the index.php to login.php. For this we make use of the preg_match which again makes sure that the input matches the regular expression.

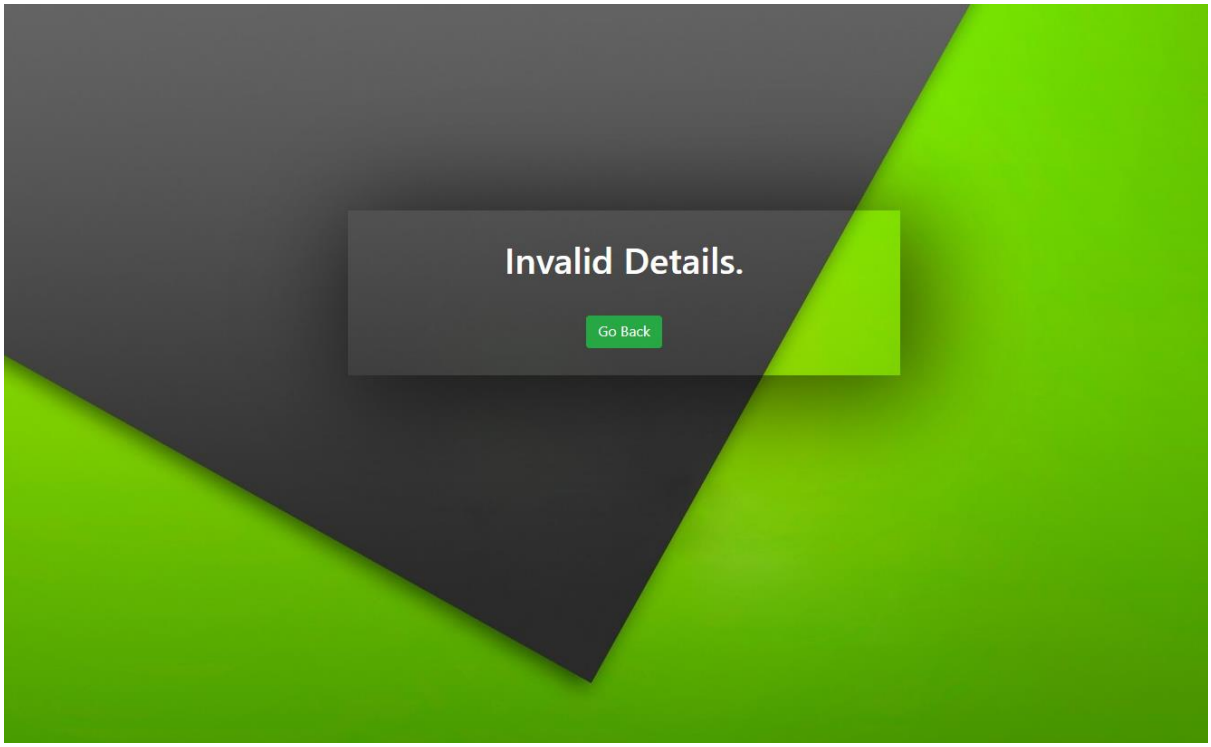
```
11  if((preg_match("/^[-\w\.\$@\*\!]{1,10}$/",$user)) && (preg_match("/^[-\w\.\$@\*\!]{1,10}$/",$pass)) )
```

Once the inputs are verified in both sides, the system is now safe and the attacker cannot gain access to the database.

Consider the working as below:



When the string for bypassing the security check is entered, it should give access to the attacker. However, this does not happen after the validation and the following page is displayed.



It shows an error as the string no longer holds true, due to the special characters which are not allowed and the attacker is redirected to the error page.

This is the working of Verification based SQL Injection Defence Mechanism.

References:

1. https://www.google.com/
2. https://www.w3schools.com/sql/sql_injection.asp
3. https://en.wikipedia.org/wiki/SQL_injection
4. https://stackoverflow.com/questions/2622856/is-preg-match-safe-enaught-in-input-satinization
5. https://stackoverflow.com/questions/4641119/mysql-query-based-on-string-within-a-preg-match-or-preg-replace
6. https://slideplayer.com/slide/5231514/
7. https://docs.citrix.com/en-us/netscaler/12/application-firewall/top-level-protections/html-sql-injection-check.html