Detection and Prevention of SQL Injection Attack

Ciprian-Mihai Ceausescu

ciprian-mihai.ceausescu@my.fmi.unibuc.ro

University of Bucharest — December 14, 2018

Introduction

SQL (Structure Query Language) injection is one of threats to the applications, which are Web-based application, Mobile application and also Desktop application, which are connected to the database in order to make work with necessary data. By implementing SQL injection, the attacker can gain full access to the application or database. After this step, the attacker can remove or change significant data irresponsibly.

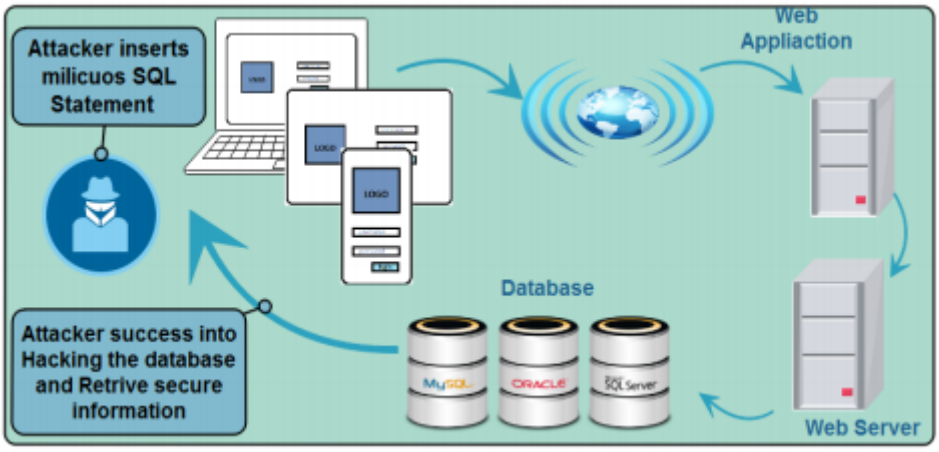
Applications that do not properly validate the user's input make them vulnerable against SQL injection. SQL Injection Attacks (SQLIA) occurs when an attacker is able to insert a series of malicious SQL statements into a query through manipulating user input data for execution by the back-end database. Using this type of threats, applications could be hacked easily and steal the confidential data by the attacker.

Nowadays, the Internet is becoming a wide-spread information infrastructure. The Utilization and fast progress of the Internet infrastructure has motivated the increasing amount of data stored in a database lately. Increasing the number of users and the heavy dependence on digital information is led to the importance of securing data or information spatially if this information about commercial, corporate businesses, institutions, organizations, numerous financial transactions, health information, personal information and other internet based services. Via Internet all web applications can be accessed using any web browsers that run on any operating systems. Web applications have become the interface that is widely used to retrieve or insert these data or information. The web application with a database that stores important information is one of the targets of the SQLIA, since the databases are absolutely accessible by attacker by injecting SQL queries that are retrieved by web application. As user information is frequently kept in these databases, important information is lost and the security violate.

SQL Injection attack

SQL Injection is one of the most common threats to a database system in which the attacker adds SQL statement to an application form input box, to gain access the resources or make changes to data stored into database. Lack of input validation in applications causes attacker to be successful. In an SQL

Injection attack, the attacker injects a string input through the application, which changes or manipulates the SQL statement to the attacker's advantage. An SQL Injection attack can harm the database in various ways, such as unauthorized manipulation of the database, or retrieval of sensitive data. It can also be used to execute system level commands that may cause the system to deny service to the application. This issue is very risky because it can cause data loss or misuse of data by parties who are not authorized and as result the functionality and confidentiality are destroyed. Basically, the process of SQLIA is illustrated in the following figure:



Risks

SQL injection is harmful and the risks associated with it provide motivation for attackers to attack the database. The main consequences of these vulnerabilities are attacks on the following characteristics:

1. **Authorization**: Critical data that are stored in a vulnerable SQL database may be altered by a successful SQLIA.
2. **Authentication**: If there is no any proper control on input fields inside the authentication page, it may be possible to login into a system as a normal user without knowing the authenticated user.
3. **Confidentially**: Usually databases are consisting of sensitive data such as personal information, credit card numbers and / or social numbers. Therefore, loss of confidentially is a terrible problem with SQL Injection vulnerability.
4. **Integrity**: By a successful SQLIA not only an attacker reads sensitive information, but also, it is possible to change or delete this private information.
5. **Database Fingerprinting**: The attacker can determine the type of database being used in backend so that he can use database-specific attacks that correspond to weakness in a particular database management system.

Classical Types SQLIA

1. **Tautologies** - Tautology-based attacks work through injecting code by one or more conditional SQL statement queries in order to make the SQL command evaluate as a true condition such as (1=1) or (- -).

```
Select * from employees where
employee_ID='1'or '1=1'--'AND
employee_password='1234';
```

2. **Piggy-backed Query** - Piggy-backed queries is a type of attack that compromises a database using a query delimiter, such as ";", to inject additional query statements to the original query.

```
SELECT pass FROM userTable WHERE
user_Id='user1' AND Password = 0; drop
userTable
```

3. **Logically Incorrect** - Logically Incorrect attack takes advantage of the error messages that are returned by the database for an incorrect query.

```
SELECT * FROM userTable WHERE
user_Id='1111' AND password='1234'
AND CONVERT (char, no)
```

4. **Union query** - Union query injection is called as statement injection attack. In this attack, attacker insert additional statement into the original SQL statement.

```
SELECT * FROM userTable WHERE
user_Id='1111' UNION SELECT * FROM
memberTable WHERE member_Id='admin' --'
AND password='1234';
```

5. **Stored Procedure** - In this technique, attacker focuses on the stored procedures which are present in the database system. Stored procedures run directly by the database engine. it is a piece of code which is exploitable. Stored procedure gives true or false values for the authorized or unauthorized clients.

```
SELECT Username FROM UserTable
WHERE user_name= 'user1' AND pass=' ';
SHUTDOWN;
```

6. **Inference** - Using Inference attack enable the attacker changing the behavior of a database or application. This type of attack can be classified into two well-known techniques, which are: Blind injection and timing attack.

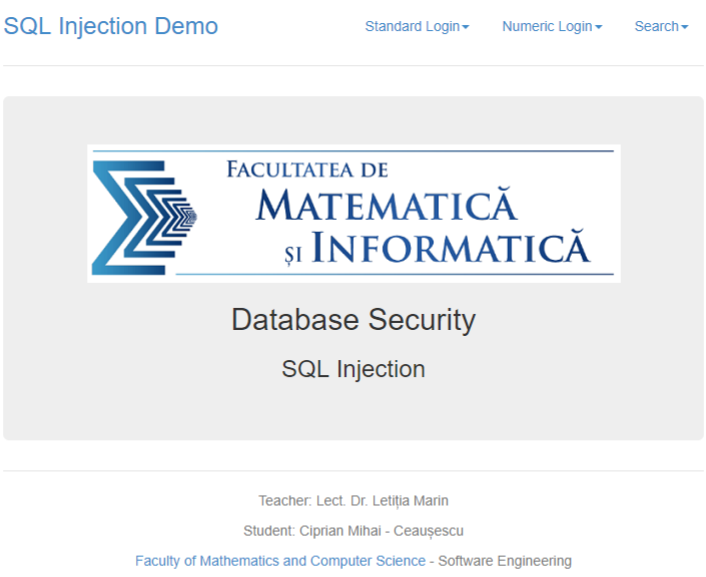
```
SELECT pass FROM userTable WHERE username=
'user' and 1 =0 -- AND pass = AND pin= 0
SELECT info FROM userTable WHERE username=
'user' and = 1 -- AND pass = AND pass= 0
```

```
declare @ varchar (8000) select @s =
db_name () if (ascii (substring (@s, 1, 1)) &
(power (2, 0))) > 0 waitfor delay '0:0:5'
```

7. **Alternate Encodings** - This type of attack occurs when attacker modify the injection query via using alternate encoding, such as hexadecimal, ASCII, and Unicode.

```
SELECT accounts FROM userTable WHERE
login='' AND pin=0; exec
(char(0x73687574646f776e))
```

Project



SQL-Injection Demo

Standard Login | Numeric Login | Search

Vulnerable Standard Login

Username: Username

Password: Password

Sign in

PHP Code:

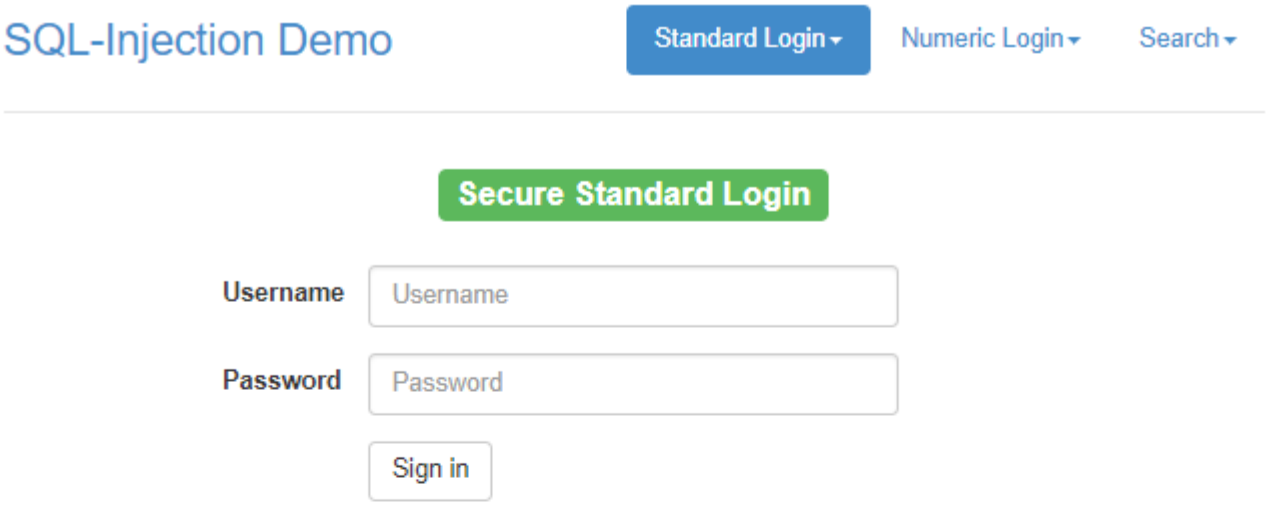
```
$username = $_POST['username'];
$password = $_POST['password'];

$query = sprintf("SELECT * FROM users WHERE username = '%s' AND password = '%s';",
                 $username,
                 $password);

$result = mysqli_query($connection, $query);

if ($result->num_rows > 0)
{
    echo "Authenticated as " . $username;

    // ...
    // $_SESSION['logged_user'] = $username;
    // ...
}
else
{
    echo "Wrong username/password combination.";
}
```



PHP Code:

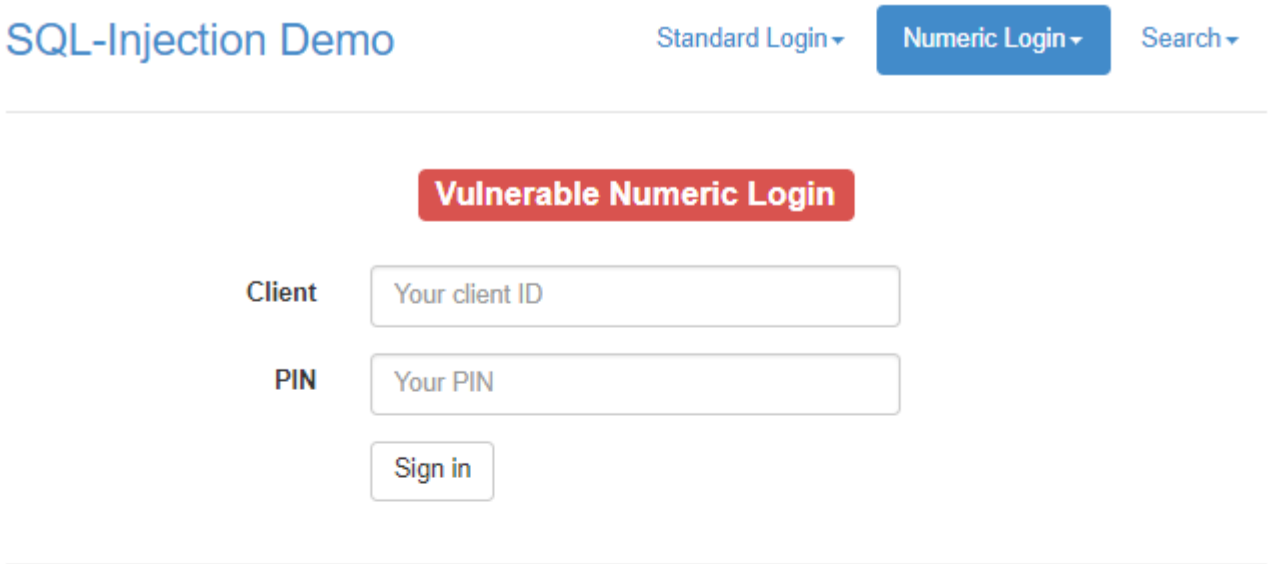
```
$username = $_POST['username'];
$password = $_POST['password'];

$query = sprintf("SELECT * FROM users WHERE username = '%s' AND password = '%s';",
                 mysqli_real_escape_string($connection, $username),
                 mysqli_real_escape_string($connection, $password));

$result = mysqli_query($connection, $query);

if ($result->num_rows > 0)
{
    echo "Authenticated as " . $username;

    // ...
    // $_SESSION['logged_user'] = $username;
    // ...
}
else
{
    echo "Wrong username/password combination.";
}
```



PHP Code:

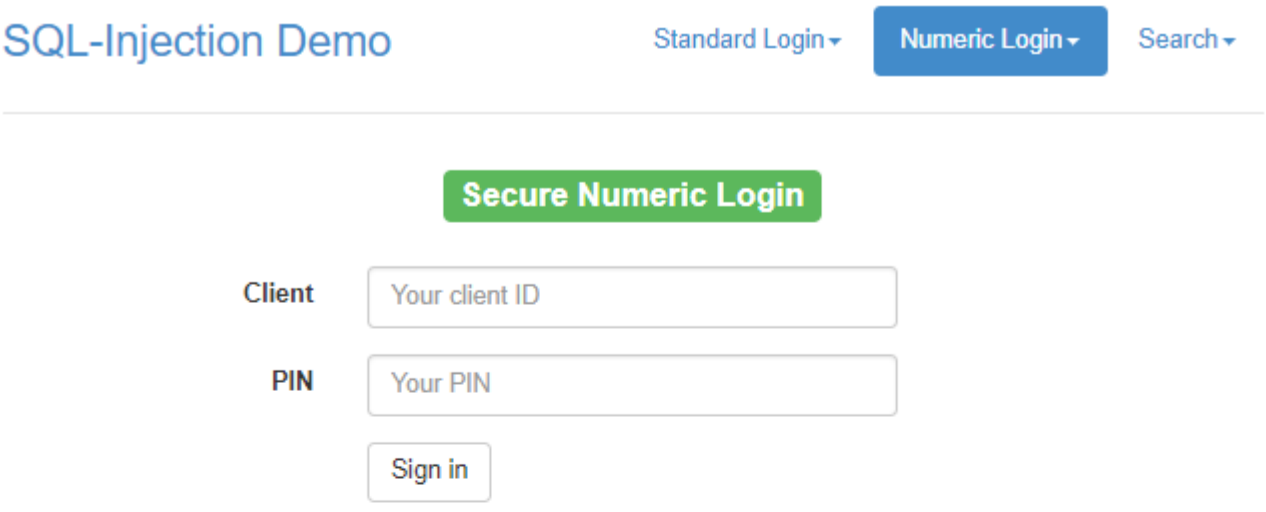
```
$client = $_POST['client'];
$pin = $_POST['pin'];

$query = sprintf("SELECT * FROM clients WHERE id = %s AND pin = %s;",
                 mysqli_real_escape_string($connection, $client),
                 mysqli_real_escape_string($connection, $pin));

$result = mysqli_query($connection, $query);

if ($result->num_rows > 0)
{
    echo "Authenticated as " . $client;

    // ...
    // $_SESSION['logged_user'] = $client;
    // ...
}
else
{
    echo "Wrong client/PIN combination.";
}
```



PHP Code:

```
$client = $_POST['client'];
$pin = $_POST['pin'];

if (is_numeric($client) && is_numeric($pin))
{
    $query = sprintf("SELECT * FROM clients WHERE id = %s AND pin = %s;",
                     mysqli_real_escape_string($connection, $client),
                     mysqli_real_escape_string($connection, $pin));

    $result = mysqli_query($connection, $query);

    if ($result->num_rows > 0)
    {
        echo "Authenticated as " . $client;

        // ...
        // $_SESSION['logged_user'] = $client;
        // ...
    }
```

SQL-Injection Demo

Standard Login ▾ Numeric Login ▾ Search ▾

Vulnerable Search

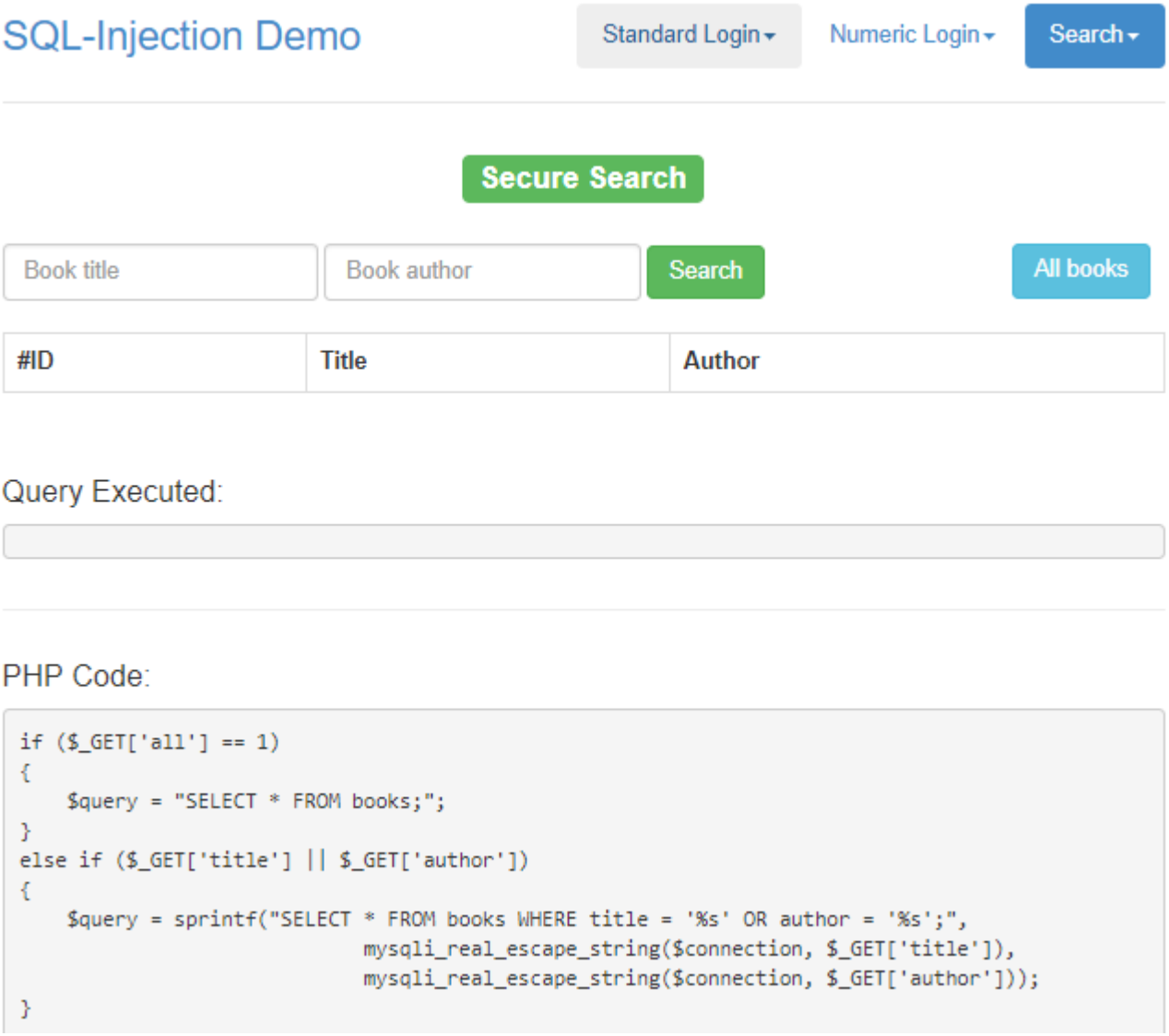
Book title | Book author | Search | All books

#ID	Title	Author

Query Executed:

PHP Code:

```
if ($_GET['all'] == 1)
{
    $query = "SELECT * FROM books;";
}
else if ($_GET['title'] || $_GET['author'])
{
    $query = sprintf("SELECT * FROM books WHERE title = '%s' OR author = '%s';",
                            $_GET['title'],
                            $_GET['author']);
}
```



Bibliography

- **Detection and Prevention of SQL Injection Attack: A Survey**, Zainab S. Alwan and Manal F. Younis2