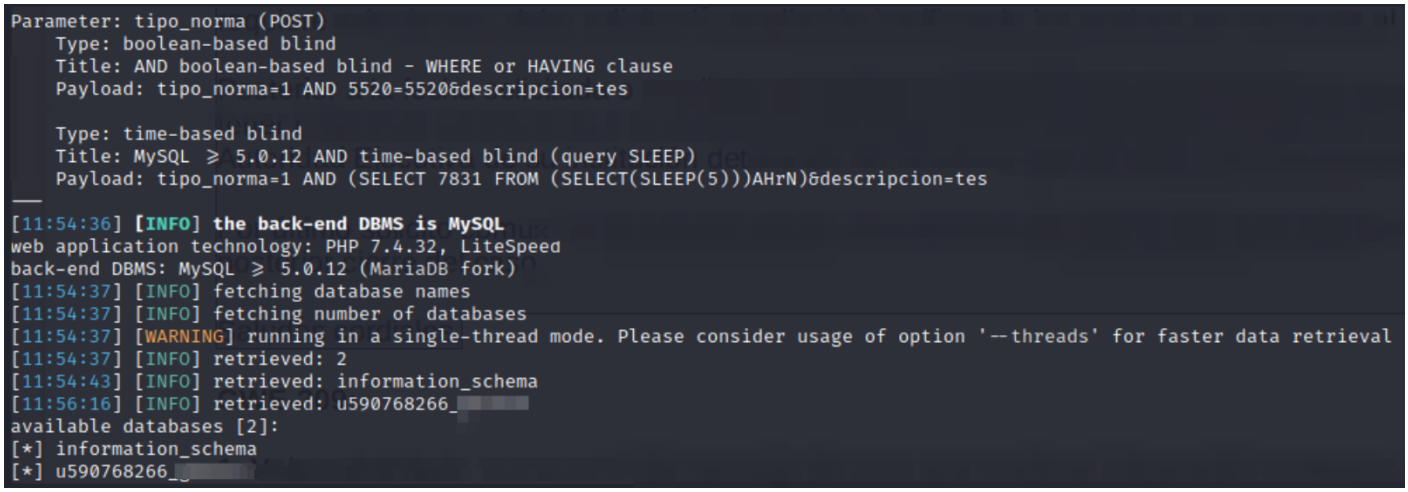
Inyección SQL

Sin la eliminación o citación suficiente de la sintaxis SQL en las entradas controlables por el usuario, la consulta SQL generada puede hacer que esas entradas se interpreten como SQL en lugar de datos de usuario ordinarios. Esto se puede usar para alterar la lógica de consulta para eludir las comprobaciones de seguridad o para insertar declaraciones adicionales que modifican la base de datos de back-end, posiblemente incluyendo la ejecución de comandos del sistema.

La inyección SQL se ha convertido en un problema común con los sitios web basados en bases de datos. La falla se detecta y explota fácilmente y, como tal, es probable que cualquier sitio o paquete de software con una base mínima de usuarios esté sujeto a un intento de ataque de este tipo. Esta falla depende del hecho de que SQL no hace una distinción real entre los planos de control y de datos.



Inyección SQL PHP - Solución

La solución aplica para aplicaciones que no usan ningún framework como base de desarrollo.

Opción a)

Usar esta opción en versiones de PHP 4 > = 4.3.0 hasta versiones anteriores a PHP 5.5.0.

```
$id=mysqlrealescapestring($GET["id"]); // sanitizando el parámetro id (ejemplo)
```

Opción b)

Ejemplo, valida el parámetro id

```
$id=$GET["id"];
$id=str_replace("'","",$id);
$id=str_replace('"','',$id);
$id=str_replace(')','',$id);
$id=str_replace('-','',$id);
$id=str_replace('%','',$id);
```

Opción c)

Usar consultas parametrizadas
https://www.php.net/manual/es/mysqli.quickstart.prepared-statements.php

También es importante realizar la verificación y validación en el resto de campos de entrada que existan en formularios o parámetros en URL de su sitio web.

Inyección SQL ASP.net - solución

Comience restringiendo la entrada en el código del lado del servidor para sus páginas web ASP.NET. No confíe en la validación del lado del cliente porque se puede omitir fácilmente. Use la validación del lado del cliente solo para reducir los viajes de ida y vuelta y mejorar la experiencia del usuario.

Si usa controles de servidor, use los controles de validación de ASP.NET, como los controles RegularExpressionValidator y RangeValidator para restringir la entrada. Si usa controles de entrada HTML regulares, use la clase Regex en su código del lado del servidor para restringir la entrada.

Opción a)

Puede restringir su entrada usando un control RegularExpressionValidator como se muestra a continuación:

```
<%@ language="C#" %>
<form id="form1" >
    <asp: TextBox ID="SSN" />
    <asp: RegularExpressionValidator ID="regexpSSN"
                                     ErrorMessage="Incorrect SSN Number"
                                     ControlToValidate="SSN"
                                     ValidationExpression="^\d{3}-\d{2}-\d{4}$" />
```

```
</form>
```

Si la entrada de SSN proviene de otra fuente, como un control HTML, un parámetro de cadena de consulta o una cookie, puede restringirla usando la clase Regex del espacio de nombres System.Text.RegularExpressions . El siguiente ejemplo asume que la entrada se obtiene de una cookie.

Opción b)

Usando System.Text.RegularExpressions;

```
if (Regex.IsMatch(Request.Cookies["SSN"], "^\d{3}-\d{2}-\d{4}$"))
{
    // access the database
}
else
{
    // handle the bad input
}
```

Opción c)

Usar parámetros con procedimientos almacenados:

El uso de procedimientos almacenados no impide necesariamente la inyección de SQL. Lo importante es usar parámetros con procedimientos almacenados. Si no usa parámetros, sus procedimientos almacenados pueden ser susceptibles a la inyección SQL si usan entrada sin filtrar como se describe en la sección "Descripción general" de este documento.

El siguiente código muestra cómo usar SqlParameterCollection al llamar a un procedimiento almacenado.

```
using System.Data;
using System.Data.SqlClient;


using (SqlConnection connection = new SqlConnection(connectionString))
{
  DataSet userDataset = new DataSet();
  SqlDataAdapter myCommand = new SqlDataAdapter(
              "LoginStoredProcedure", connection);
```

```
  myCommand.SelectCommand.CommandType = CommandType.StoredProcedure;
  myCommand.SelectCommand.Parameters.Add("@au_id", SqlDbType.VarChar, 11);
  myCommand.SelectCommand.Parameters["@au_id"].Value = SSN.Text;

  myCommand.Fill(userDataset);
}
```

En este caso, el parámetro @au_id se trata como un valor literal y no como un código ejecutable. Además, se comprueba el tipo y la longitud del parámetro. En el ejemplo de código anterior, el valor de entrada no puede tener más de 11 caracteres. Si los datos no se ajustan al tipo o la longitud definidos por el parámetro, la clase SqlParameter genera una excepción.

Opción d)

Usar parámetros con SQL dinámico:

Si no puede usar procedimientos almacenados, aún debe usar parámetros cuando construya sentencias SQL dinámicas. El siguiente código muestra cómo usar SqlParametersCollection con SQL dinámico.

```
using System.Data;
using System.Data.SqlClient;

using (SqlConnection connection = new SqlConnection(connectionString))
{
  DataSet userDataset = new DataSet();
  SqlDataAdapter myDataAdapter = new SqlDataAdapter(
         "SELECT au_lname, au_fname FROM Authors WHERE au_id = @au_id",
         connection);
  myCommand.SelectCommand.Parameters.Add("@au_id", SqlDbType.VarChar, 11);
  myCommand.SelectCommand.Parameters["@au_id"].Value = SSN.Text;
  myDataAdapter.Fill(userDataset);
}
```

Opción e)

Uso de procesamiento por lotes de parámetros:

Un concepto erróneo común es que si concatena varias sentencias SQL para enviar un lote de sentencias al servidor en un solo viaje de ida y vuelta, no puede usar parámetros. Sin embargo, puede usar esta técnica si se asegura de que los nombres de los parámetros no se repitan. Puede hacer esto fácilmente asegurándose de usar nombres de parámetros únicos durante la concatenación de texto SQL, como se muestra aquí.

```
using System.Data;
using System.Data.SqlClient;
. . .
using (SqlConnection connection = new SqlConnection(connectionString))
{
  SqlDataAdapter dataAdapter = new SqlDataAdapter(
       "SELECT CustomerID INTO #Temp1 FROM Customers " +
       "WHERE CustomerID > @custIDParm; SELECT CompanyName FROM Customers " +
       "WHERE Country = @countryParm and CustomerID IN " +
       "(SELECT CustomerID FROM #Temp1);",
       connection);
  SqlParameter custIDParm = dataAdapter.SelectCommand.Parameters.Add(
                                          "@custIDParm", SqlDbType.NChar, 5);
  custIDParm.Value = customerID.Text;

  SqlParameter countryParm = dataAdapter.SelectCommand.Parameters.Add(
                                      "@countryParm", SqlDbType.NVarChar, 15);
  countryParm.Value = country.Text;

  connection.Open();
  DataSet dataSet = new DataSet();
  dataAdapter.Fill(dataSet);
}
. . .
```

Inyección SQL JAVA - solución

La solución más simple es usar PreparedStatement en lugar de Statement para ejecutar la consulta.

En lugar de concatenar el nombre de usuario y la contraseña en la consulta, los proporcionamos para consultar a través de los métodos de establecimiento de PreparedStatement.

Ahora, el valor del nombre de usuario y la contraseña recibidos de la solicitud se tratan solo como datos, por lo que no se producirá una inyección SQL.

Veamos el código del servlet modificado.

```
String query = "select * from tbluser where username=? and password = ?";
Connection conn = null;
PreparedStatement stmt = null;
//Las credenciales utilizadas son de ejemplo, se recomienda utilizar contraseñas robustas
try {
    conn = DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/user", "root", "root");
    stmt = conn.prepareStatement(query);
    stmt.setString(1, username);
    stmt.setString(2, password);
    ResultSet rs = stmt.executeQuery();
    if (rs.next()) {
        // Login Successful if match is found
        success = true;
    }
    rs.close();
} catch (Exception e) {
    e.printStackTrace();
} finally {
    try {
        stmt.close();
        conn.close();
    } catch (Exception e) {
    }
}
```

Entendamos lo que está pasando en este caso.

Consulta : seleccione * de tbluser donde nombre de usuario =? y contraseña = ?

El signo de interrogación (?) en la consulta anterior se denomina parámetro posicional. Hay 2 parámetros posicionales en la consulta anterior. No concatenamos nombre de usuario y contraseña para consultar. Usamos métodos disponibles en PreparedStatement para proporcionar información de usuario.

Hemos configurado el primer parámetro usando stmt.setString(1, username) y el segundo parámetro usando stmt.setString(2, password). La API de JDBC subyacente se encarga de desinfectar los valores para evitar la inyección de SQL.

Revision #14
Created 1 febrero 2023 14:24:40
Updated 10 marzo 2023 11:56:15 by Vladimir Urquiola