

1. Entradas seleccionadas

Los dos métodos seleccionados fueron de la clase *UserHandler.cs* dado que de todo el sistema *backend* eran los más vulnerables:

```
public DataTable GetEmployeeInfo(ReciberModel id)
{
    string consult = "select Identification, FirstName, LastName, LastName2, Email, Country," +
        "State, City, ZipCode, Address, Phone from Users where Identification =" + "'" + id.id + "'";
    DataTable tableResult = CreateTableConsult(consult);
    return tableResult;
}
public DataTable ViewEmployeeInfo(string id)
{
    string consult = "select Identification, FirstName, LastName, LastName2, Email, Country," +
        "State, City, ZipCode, Address, Phone from Users where Identification =" + "'" + id + "'";
    DataTable tableResult = CreateTableConsult(consult);
    return tableResult;
}
```

Estos métodos **pertencen a las User Stories**, respectivamente:

- `public DataTable GetEmployeeInfo(ReciberModel id);` → TB-28: Como empleado quiero editar mi usuario en el sistema para poder actualizar datos.
- `public DataTable ViewEmployeeInfo(string id);` → TB-21 Como empleador quiero poder visualizar mi lista de empleados y su información relacionada en pantalla para tener fácil acceso a esta información.

Dada la concatenación de lo que se está recibiendo, en todo el sistema estos son los métodos más vulnerables a *SQL Injection*.

2. Implementación de controles necesarios

En nuestro sistema, basado en *.NET*, las consultas se hacían de forma que se tenía la consulta básica y se le pasaban los parámetros usando las funciones que este proveía; esta forma era la siguiente por ejemplo:

```
var consult = @"SELECT Identification, UserType, FirstName, LastName, LastName2
FROM Users
WHERE Email = @email AND Password = @password";
var queryCommand = new SqlCommand(consult, connection);

// Uses user's email to get their ID
queryCommand.Parameters.AddWithValue("@email", email);
queryCommand.Parameters.AddWithValue("@password", password);
```

Al principio desconocíamos de lo que era el *SQL Injection*, y esta forma de hacer las consultas fue prácticamente usada en todo el sistema *backend*, luego al entender lo que era *SQL Injection* y leer sobre las formas de prevenirlo descubrimos que precisamente esta forma como ya lo hacíamos era la forma de prevenirlo, así que habíamos hecho nuestro sistema protegido con *SQL Injection* desde el principio.

De todo el sistema solo hay dos métodos que no tenían esta forma establecida, los cuáles, como se mencionó anteriormente, están en la clase *UserHandler*:

```
public DataTable GetEmployeeInfo(ReciberModel id) {}
public DataTable ViewEmployeeInfo(string id) {}
```

Lo que se hizo fue convertir estos métodos de la forma incorrecta en que estaban, a la forma en que se realizaron los demás, utilizando parametrización, y además estableciendo la longitud de la variable a una cantidad de caracteres fija, pues el método siempre recibe un número de 10 caracteres:

```
public DataTable GetEmployeeInfo(ReciberModel id)
{
    string consult = "select Identification, FirstName, LastName, LastName2, Email, Country," +
        "State, City, ZipCode, Address, Phone from Users where Identification = @id";
    SqlCommand queryCommand = new SqlCommand(consult, connection);
    queryCommand.Parameters.AddWithValue("@id", id.id.Substring(0, 10));

    DataTable tableResult = CreateTableConsult(queryCommand);

    return tableResult;
}

public DataTable ViewEmployeeInfo(string id)
{
    string consult = "select Identification, FirstName, LastName, LastName2, Email, Country," +
        "State, City, ZipCode, Address, Phone from Users where Identification = @id";
    SqlCommand queryCommand = new SqlCommand(consult, connection);
    queryCommand.Parameters.AddWithValue("@id", id.Substring(0, 10));

    DataTable tableResult = CreateTableConsult(queryCommand);

    return tableResult;
}

private DataTable CreateTableConsult(SqlCommand queryCommand)
{
    SqlDataAdapter adaptadorParaTabla = new SqlDataAdapter(queryCommand);
    DataTable tableFormatConsult = new DataTable();
    connection.Open();
    adaptadorParaTabla.Fill(tableFormatConsult);
    connection.Close();
    return tableFormatConsult;
}
```

Resumidamente lo que se hizo fue parametrizar los datos que se concatenan a la consulta y limitar su longitud, para no concatenarlos directamente a la string sino **parametrizarlos**, eso se hizo por medio del método:

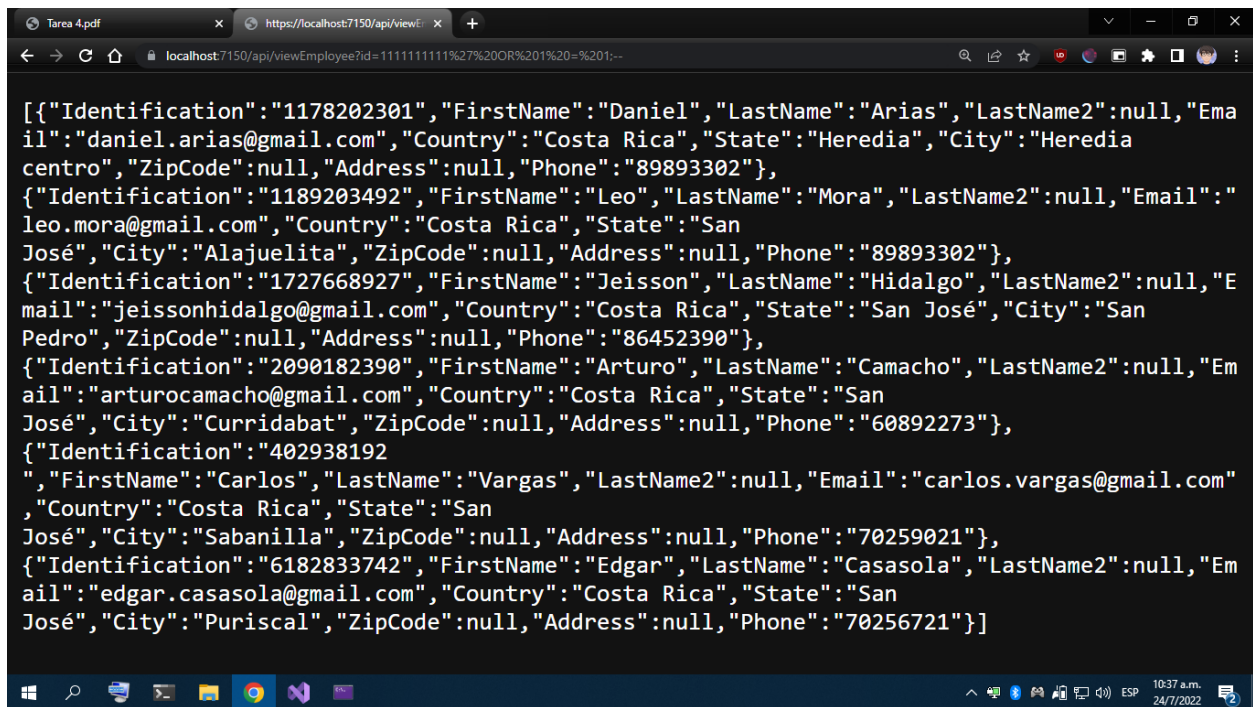
```
queryCommand.Parameters.AddWithValue("@id", id.Substring(0, 10)).
```

3. Realización de pruebas

Cabe reiterar que esto se hizo en lugares que el sistema casi no usa porque ya nuestro sistema estaba protegido con estas medidas contra *SQL Injection*, por lo que en todos los demás métodos ya se implementaba la parametrización de datos y el uso de *stored procedures*.

En la Figura 1 podemos ver que al acceder a la dirección: <https://localhost:7150/api/viewEmployee?id=1111111111'OR1=1;-->, la cual realiza una consulta por medio del URL, seleccionando no solo los datos de un usuario específico, sino que devolviendo todos los datos almacenados al hacer uso de *OR 1 = 1* el sistema **estaba desprotegido**; mientras que en la Figura 2 ya podemos ver que el sistema esta protegido gracias

a las medidas que se mencionaron antes. Cabe destacar que, las pruebas se realizaron mediante llamados a las APIs ya que del lado del **frontend**, las funcionalidades que realizan estas consultas se implementaron en botones, imposibilitando el ingreso de texto de manera gráfica.



```
[{"Identification": "1178202301", "FirstName": "Daniel", "LastName": "Arias", "LastName2": null, "Email": "daniel.arias@gmail.com", "Country": "Costa Rica", "State": "Heredia", "City": "Heredia centro", "ZipCode": null, "Address": null, "Phone": "89893302"}, {"Identification": "1189203492", "FirstName": "Leo", "LastName": "Mora", "LastName2": null, "Email": "leo.mora@gmail.com", "Country": "Costa Rica", "State": "San José", "City": "Alajuelita", "ZipCode": null, "Address": null, "Phone": "89893302"}, {"Identification": "1727668927", "FirstName": "Jeisson", "LastName": "Hidalgo", "LastName2": null, "Email": "jeissonhidalgo@gmail.com", "Country": "Costa Rica", "State": "San José", "City": "San Pedro", "ZipCode": null, "Address": null, "Phone": "86452390"}, {"Identification": "2090182390", "FirstName": "Arturo", "LastName": "Camacho", "LastName2": null, "Email": "arturocamacho@gmail.com", "Country": "Costa Rica", "State": "San José", "City": "Curridabat", "ZipCode": null, "Address": null, "Phone": "60892273"}, {"Identification": "402938192", "FirstName": "Carlos", "LastName": "Vargas", "LastName2": null, "Email": "carlos.vargas@gmail.com", "Country": "Costa Rica", "State": "San José", "City": "Sabanilla", "ZipCode": null, "Address": null, "Phone": "70259021"}, {"Identification": "6182833742", "FirstName": "Edgar", "LastName": "Casasola", "LastName2": null, "Email": "edgar.casasola@gmail.com", "Country": "Costa Rica", "State": "San José", "City": "Puriscal", "ZipCode": null, "Address": null, "Phone": "70256721"}]
```

Figura 1: Sistema **desprotegido** contra *SQL Injection*.

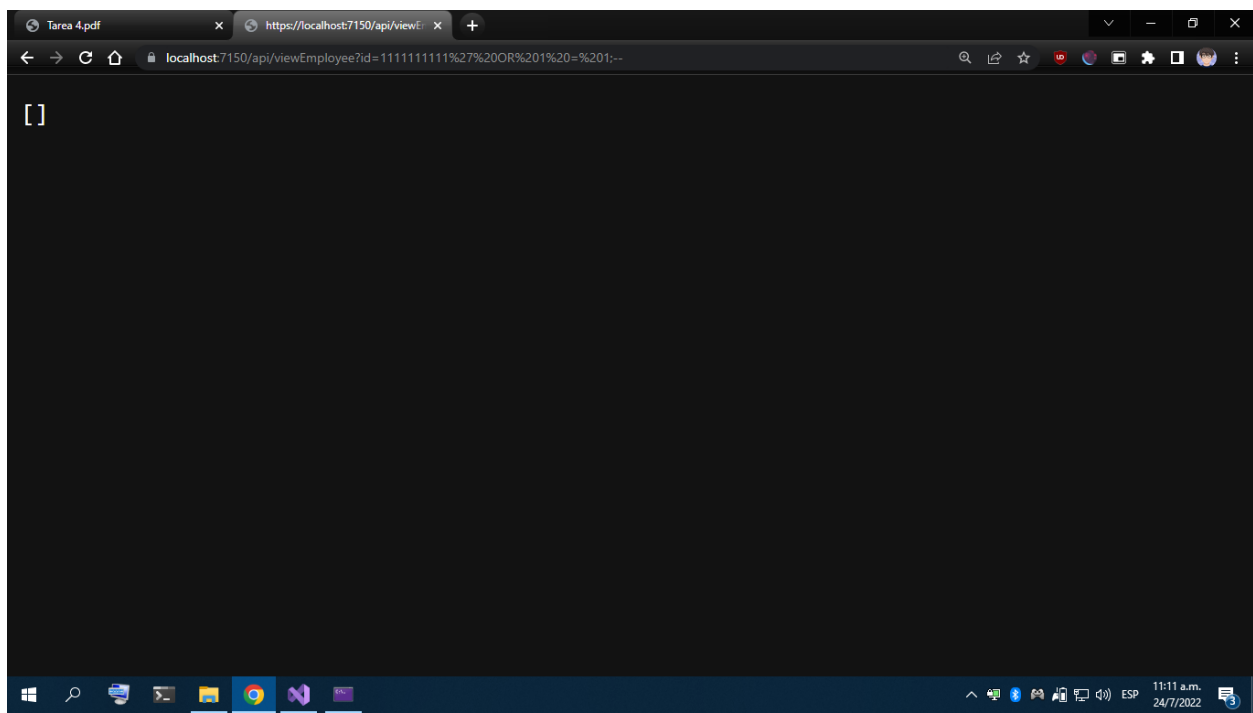


Figura 2: Sistema **protegido** contra *SQL Injection*.