Walkthrough of Programming Databases Using Microsoft ADO

Version 0.5

Introduction to the Sample Application

The sample application is designed to show how to interact with a DBMS using relational SQL and simple call-level interface that uses connections, commands, and cursors. The application is not meant to solve a specific problem but rather to provide a test bed for understanding how to interact with database.

This application is written as a client-only an "HTML Application" (`.hta` file) in HTML and JavaScript. While this application connects to the extended form of Connolly/Begg *Hotel* database, the application could connect to other DBMSs such as MySQL or Microsoft SQL Server by changing the connection string. The concepts implemented in the sample application should transfer to other approaches for writing database applications.

This application depends on using Microsoft ActiveX extensions to the web browser, hence the application requires the use of Microsoft Internet Explorer running on Microsoft Windows platform.

Both safe (secure) code and unsafe (vulnerable) code is included to understand how to prevent SQL injection style attacks.

The Core HTML Page

This HTML application uses a simple structure for interaction consisting of three parts implemented as `<DIV>` elements in body of the web page:

- `MenuSection` – List of menu options to execute functionality in the application
- `FormSection` – Placeholder to display <FORM> elements to gather user input for interaction with the database such as search criteria
- `OuptutSection` – Placeholder to update with status information and output data retrieved from the database

JavaScript functions replace contents of the `FormSection` and OutputSection dynamically using web browser Document Object Model (DOM) interface.

```
<!--  Core HTML Page -->
<body>
    <a name="home"/>
    <h1>ADO Programming Example</h1>
    <h2>Select an Option</h2>
    <div id="MenuSection">
        <a href="#home" onclick="showHotel(&quot;OutputSection&quot;);" >
            Show List of Hotels</a>
        <br/>
        <a href="#home" onclick="formSearchHotel(&quot;searchHotelUnsecure&quot;);" >
            Search for a Hotel using SQL Injection Vulnerable Code</a>
        <br/>
        <a href="#home" onclick="formSearchHotel(&quot;searchHotelSecure&quot;);" >
            Search for a Hotel using Secure Code</a>
    </div>
    <div id="FormSection">
    </div>
    <h2>Output</h2>
    <div id="OutputSection">
    </div>
</body>
```

Overview of the JavaScript Code

All of the JavaScript code is located in the `<HEAD>` (header) element of the web page. The JavaScript code consists of discrete functions listed in the source code. The current application has five different functions:

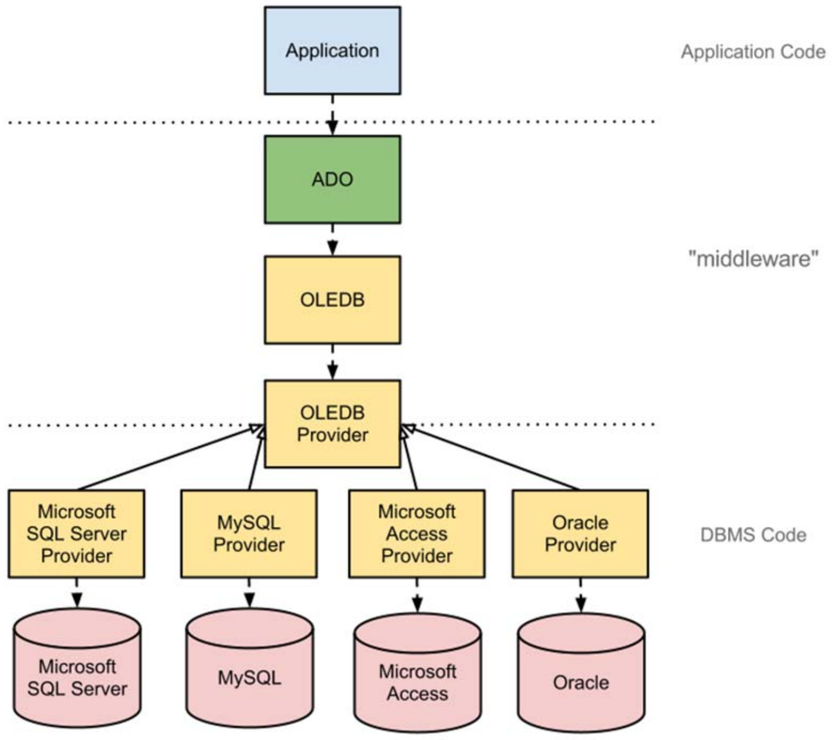
- `getPath()` – return relative file path
- `formSearchHotel()` – displays a `<FORM>` to get user input
- `searchHotelSecure()` – secure code approach when including user input as part of database interaction
- `searchHotelUnsecure()` – SQL injection vulnerable approach when including user input as part of database interaction
- `showHotel()` – simple `SELECT` statement execution

The Microsoft ADO Database Interface

This application depends on the Microsoft ADO (ActiveX Data Objects) call-level interface (CLI) for interacting with a DBMS. Call-level interfaces for DBMSs like ADO provide functions that allow you to connect to databases, execute SQL statements, and retrieve data.

Middleware Architecture

ADO implements a middleware architectural style for the database interface. The middleware architecture style allows an application to communicate to a variety of different underlying database engines without the need to port the application for unique aspects of each interface. Database engine specific interaction is isolated internally with the middleware architecture in database "providers" that implement an abstract class interface than then can be called by the higher level functionality in ADO. The "provider" is determined by configuration string passed to the ADO call-level interface. Middleware architectural approaches do have a performance overhead (assume about 5-10%) compared with calling a proprietary DBMS interface directly.

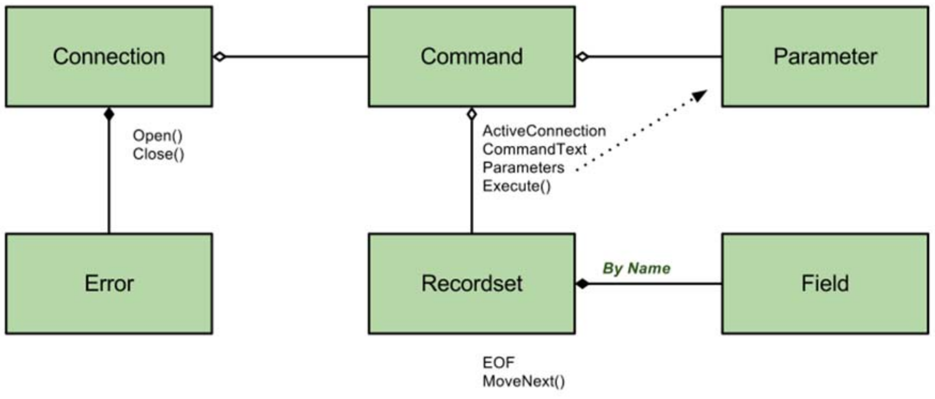


ADO Architecture

Similar middleware call-level interfaces include Microsoft's ODBC and ADO.Net interfaces well as Java's JDBC.

ADO Object Model

ADO is an object-oriented call-level programming interface based on common concepts associated with direct interfaces for interacting with databases (versus interfaces based on disconnected datasets). The ADO interface focuses on three key conceptual objects—connections, commands, and record sets.



ADO Object Model (with functions and attributes used in sample application)

Connection

The Connection object holds the channel for communicating to underlying DBMS. The Connection object configures the appropriate "provider" or driver to use, manages authentication, provides context for transactions, and holds recent errors.

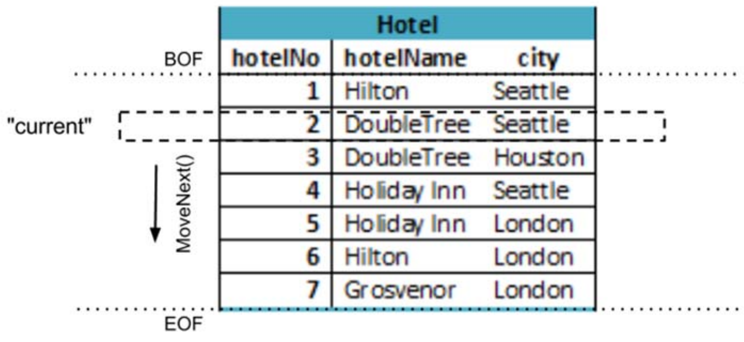
Command

The Command object processes a provided SQL statement as a command. The Command objects implements a standard SQL syntax that may not have the rich extensions available at the proprietary DBMS level. You specify an SQL statement as a string to the Command object. The SQL statement can contain parameters (arguments) in the form of `[`*`parameter-name`*`]` within the string for later replacement by supplying a Parameter object at run-time with the appropriate value.

You should use parameterized SQL statements with Command objects rather than concatenating together a SQL statement at run-time with user-supplied arguments in order to prevent SQL injection attacks. This will be detailed in the comparison between the secure and vulnerable code examples described later.

Recordset

The Recordset object provides a file-like interface (commonly called a *cursor* in database programming) for accessing the data retrieved from a database using an SQL `SELECT` statement or other output producing command.



Hotel		
hotelNo	hotelName	city
1	Hilton	Seattle
2	DoubleTree	Seattle
3	DoubleTree	Houston
4	Holiday Inn	Seattle
5	Holiday Inn	London
6	Hilton	London
7	Grosvenor	London

A database cursor acts similar to a file pointer when reading a file. The cursor allows you to access data retrieved from a database on a row by row basis. At any given time, you can only access one row's column values. The simplest form (read only server-side) cursors can only be moved in forward direction (using `MoveNext()`) toward the last row in the Recordset. The Recordset has an `EOF` flag that becomes true when the cursor has been moved past the last row in the Recordset.

Individual column values for a row in the Recordset can be accessed by name or by relative zero-based position. Let's assume a Recordset object named `Results` contained the retrieved output from `SELECT hotelNo, hotelName, city FROM Hotel`. For each row, the value for customer name can be accessed by column name using `Results("hotelName")` or by relative position `Results(1)`.

Skeleton for Simple Database Retrieval

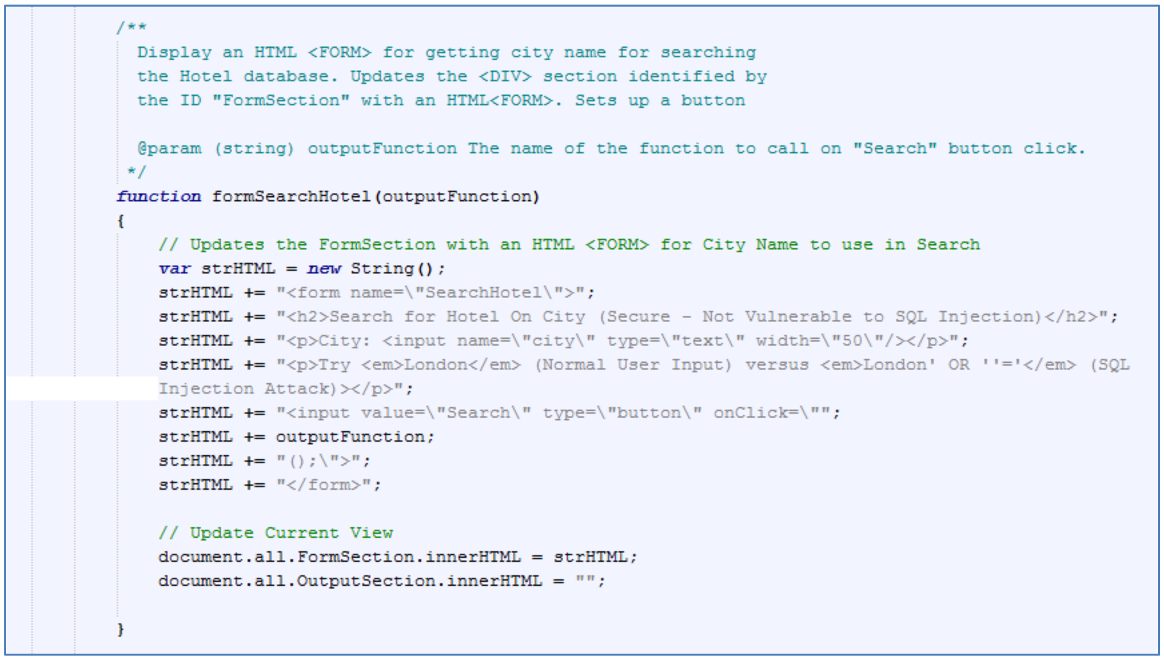
The simple skeleton code for database retrieval is:

1) Establish a Connection to a database
2) Prepare a Command to execute an SQL `SELECT` statement
3) Execute the Command and get the resulting Recordset
4) Loop through the Recordset until EOF is reached using MoveNext()

Function by Function Details

formSearchHotel

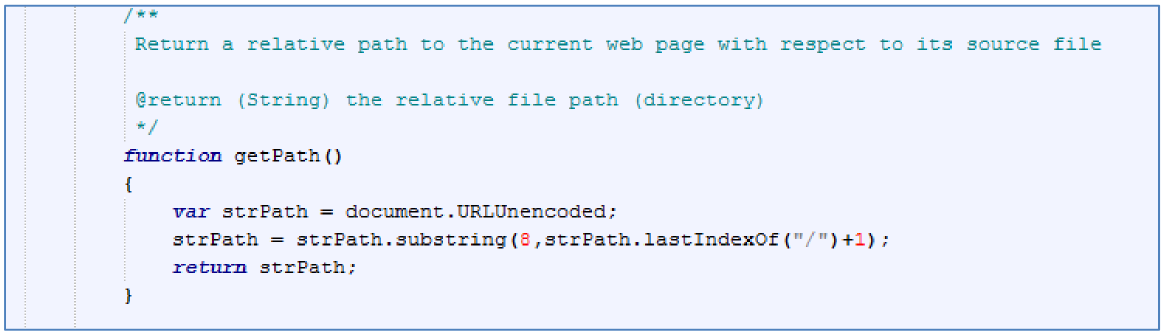
`formSearchHotel()` generates a `<FORM>` element in the `FormSection` of the webpage to get the input to search on the *City* column of the *Hotel* table. The `outputFunction` argument specifies the search function to call when the user selects the *Search* button.



```
/**
  Display an HTML <FORM> for getting city name for searching
  the Hotel database. Updates the <DIV> section identified by
  the ID "FormSection" with an HTML<FORM>. Sets up a button

  @param (string) outputFunction The name of the function to call on "Search" button click.
 */
function formSearchHotel(outputFunction)
{
    // Updates the FormSection with an HTML <FORM> for City Name to use in Search
    var strHTML = new String();
    strHTML += "<form name=\"SearchHotel\">";
    strHTML += "<h2>Search for Hotel On City (Secure - Not Vulnerable to SQL Injection)</h2>";
    strHTML += "<p>City: <input name=\"city\" type=\"text\" width=\"50\"/></p>";
    strHTML += "<p>Try <em>London</em> (Normal User Input) versus <em>London' OR ''='</em> (SQL
    Injection Attack)</p>";
    strHTML += "<input value=\"Search\" type=\"button\" onClick=\"";
    strHTML += outputFunction;
    strHTML += "();\">";
    strHTML += "</form>";

    // Update Current View
    document.all.FormSection.innerHTML = strHTML;
    document.all.OutputSection.innerHTML = "";

}
```

getPath

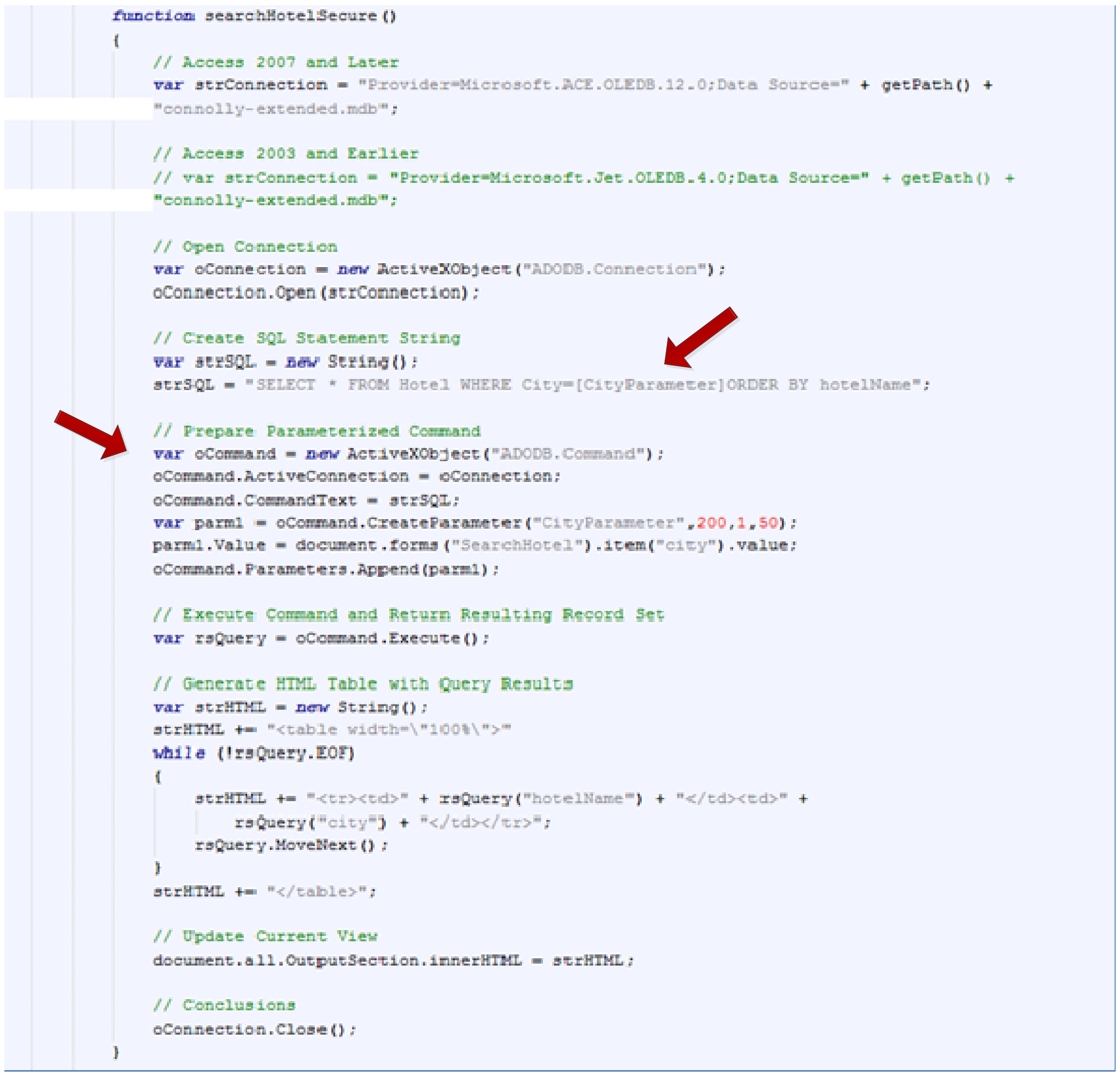
`getPath()` is a utility function that returns the relative file path (directory path) to the current web page. The application assumes that the `connolly-extended.mdb` (Microsoft Access database) is located in the same directory. This allows the application to be executed more easily on any machine.



```
/**
 Return a relative path to the current web page with respect to its source file

 @return (String) the relative file path (directory)
 */
function getPath()
{
    var strPath = document.URLUnencoded;
    strPath = strPath.substring(8,strPath.lastIndexOf("/")+1);
    return strPath;
}
```

searchHotelSecure

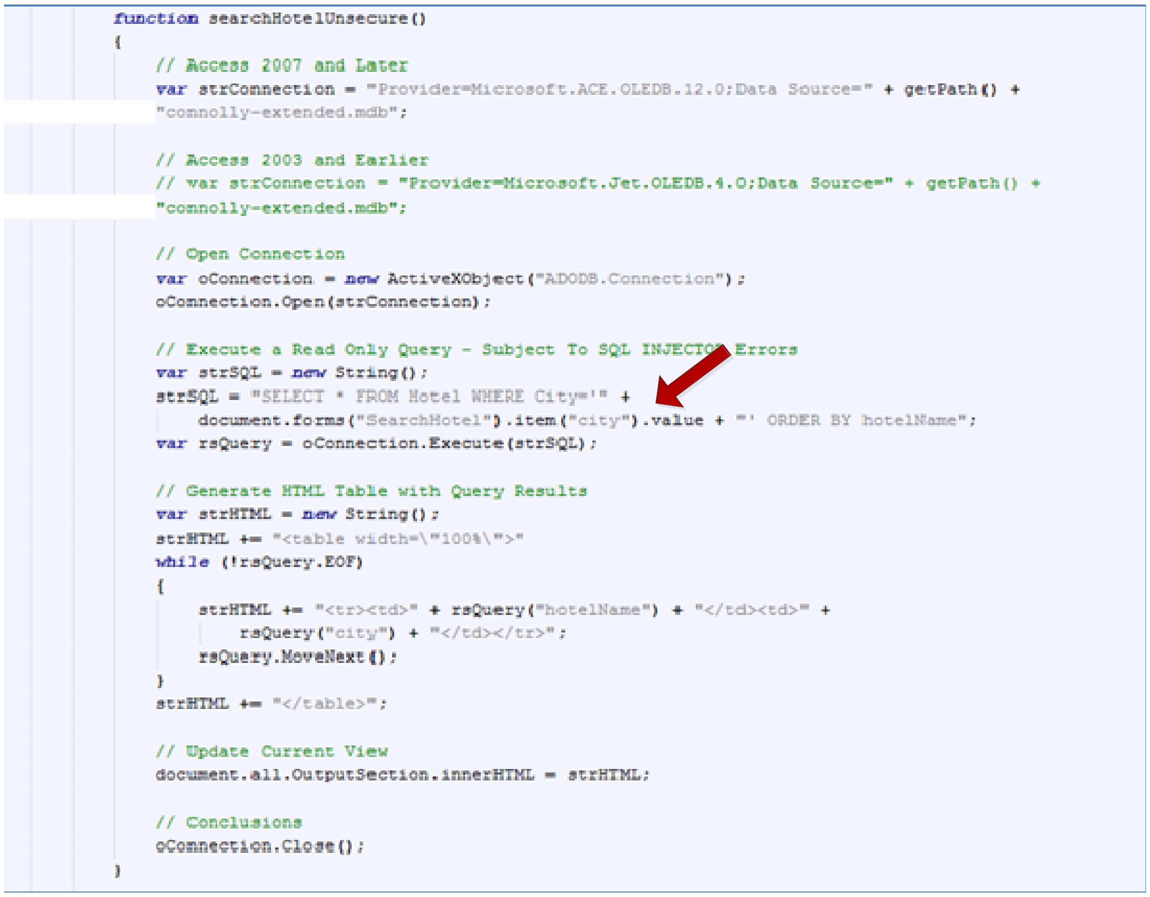
`searchHotelSecure()` implements a secure coding approach for calling parameterized database operations—in this case, a `SELECT` statement where the user is providing a parameter for the city name. The key to the secure coding approach is to create a Command object for executing the SQL statement. `[CityParameter]` is specified in the SQL statement text where we want to supply the value for the city name. A Parameter object is created to hold the parameter value using `CreateParameter()`. You will want to refer to ADO document for details on the parameters to function. The parameters specify the parameter name, data type, parameter type, and length. The parameter with its value assigned is attached to the Command object (using `Parameters.Append()`).



```
function searchHotelSecure()
{
    // Access 2007 and Later
    var strConnection = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" + getPath() +
"connolly-extended.mdb";

    // Access 2003 and Earlier
    // var strConnection = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" + getPath() +
"connolly-extended.mdb";

    // Open Connection
    var oConnection = new ActiveXObject("ADODB.Connection");
    oConnection.Open(strConnection);

    // Create SQL Statement String
    var strSQL = new String();
    strSQL = "SELECT * FROM Hotel WHERE City=[CityParameter]ORDER BY hotelName";

    // Prepare Parameterized Command
    var oCommand = new ActiveXObject("ADODB.Command");
    oCommand.ActiveConnection = oConnection;
    oCommand.CommandText = strSQL;
    var parm1 = oCommand.CreateParameter("CityParameter",200,1,50);
    parm1.Value = document.forms("SearchHotel").item("city").value;
    oCommand.Parameters.Append(parm1);

    // Execute Command and Return Resulting Record Set
    var rsQuery = oCommand.Execute();

    // Generate HTML Table with Query Results
    var strHTML = new String();
    strHTML += "<table width=\"100%\">"
    while (!rsQuery.EOF)
    {
        strHTML += "<tr><td>" + rsQuery("hotelName") + "</td><td>" +
            rsQuery("city") + "</td></tr>";
        rsQuery.MoveNext();
    }
    strHTML += "</table>";

    // Update Current View
    document.all.OutputSection.innerHTML = strHTML;

    // Conclusions
    oConnection.Close();
}
```

searchHotelUnsecure

`searchHotelUnsecure()` implements an SQL injection vulnerable coding approach for calling parameterized database operations—in this case, a `SELECT` statement where the user is providing a parameter for the city name. The key to vulnerability is the creation of the SQL statement to be executed using only string concatenation. This is further exacerbated by no input validation on the string provide by the user. While the best method for writing secure code includes input validation, this may not be sufficient because we may not be aware of all string canonical forms that user might specify for a string.



```
function searchHotelUnsecure()
{
    // Access 2007 and Later
    var strConnection = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" + getPath() +
    "connolly-extended.mdb";

    // Access 2003 and Earlier
    // var strConnection = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" + getPath() +
    "connolly-extended.mdb";

    // Open Connection
    var oConnection = new ActiveXObject("ADODB.Connection");
    oConnection.Open(strConnection);

    // Execute a Read Only Query - Subject To SQL INJECTION Errors
    var strSQL = new String();
    strSQL = "SELECT * FROM Hotel WHERE City='" +
        document.forms("SearchHotel").item("city").value + "' ORDER BY hotelName";
    var rsQuery = oConnection.Execute(strSQL);

    // Generate HTML Table with Query Results
    var strHTML = new String();
    strHTML += "<table width=\"100%\">"
    while (!rsQuery.EOF)
    {
        strHTML += "<tr><td>" + rsQuery("hotelName") + "</td><td>" +
            rsQuery("city") + "</td></tr>";
        rsQuery.MoveNext();
    }
    strHTML += "</table>";

    // Update Current View
    document.all.OutputSection.innerHTML = strHTML;

    // Conclusions
    oConnection.Close();
}
```

showHotel

`showHotel()` shows the simple form of using ADO to retrieve rows from a database. The function defines a connection string to configure the ADO database provider to use for a specific DBMS (in this case Microsoft Access).

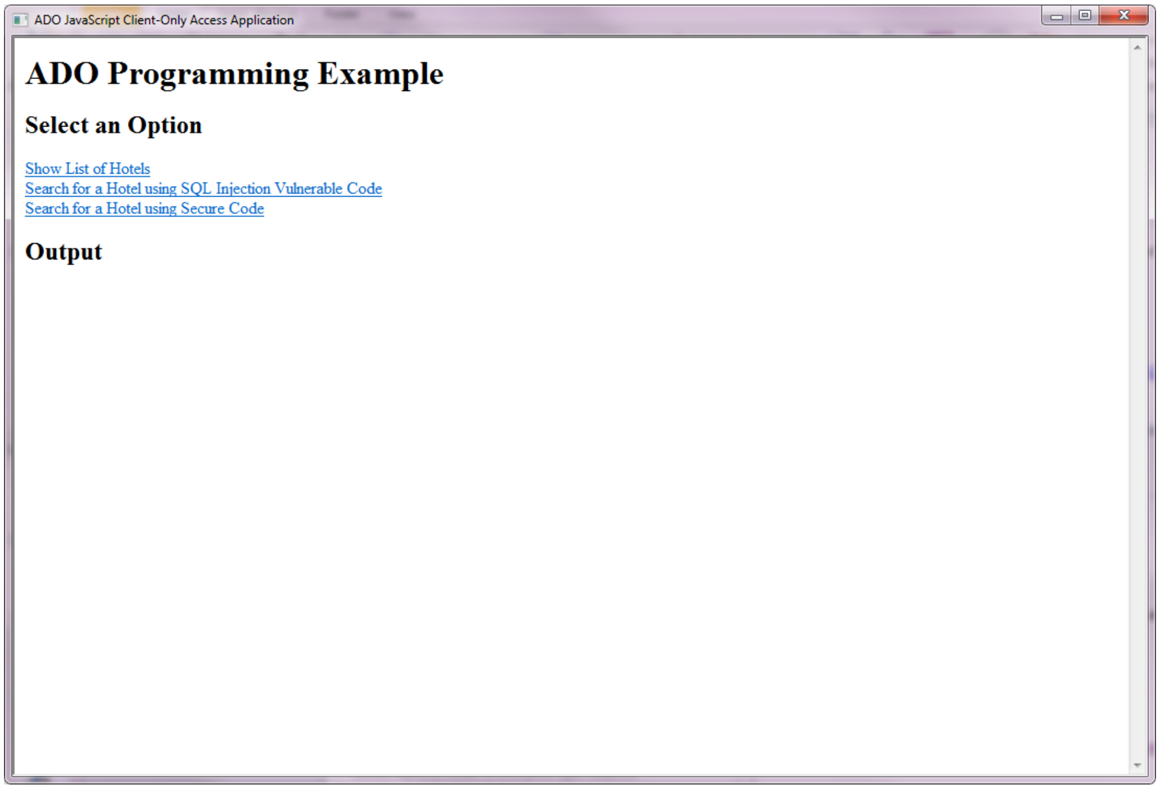
```
// showHotel - Output a List of Hotels to Output Window
function showHotel()
{
    // Access 2007 and Later
    var strConnection = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" + getPath() +
    "connolly-extended.mdb";

    // Access 2003 and Earlier
    // var strConnection = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" + getPath() +
    "connolly-extended.mdb";

    // Open Connection
    var oConnection = new ActiveXObject("ADODB.Connection");
    oConnection.Open(strConnection);

    // Execute a Read Only Query
    var rsQuery = oConnection.Execute("SELECT * FROM Hotel ORDER BY hotelName,city");

    // Generate HTML Table with Query Results
    var strHTML = new String();
    strHTML += "<table width=\"100%\">"
    while (!rsQuery.EOF)
    {
        strHTML += "<tr><td>" + rsQuery("hotelName") + "</td><td>" +
            rsQuery("city") + "</td></tr>";
        rsQuery.MoveNext();
    }
    strHTML += "</table>";

    // Update Current View
    document.all.FormSection.innerHTML = "";
    document.all.OutputSection.innerHTML = strHTML;

    // Conclusions
    oConnection.Close();
}
```

Executing the Application

The application can be executed by open the `connolly-extended-secure.hta` file using Microsoft Internet Explorer web browser. The browser may warn about security concerns which should be accepted with this application.

Startup Screen

The application will show a home page with three menu options. These menu options will execute JavaScript functions using web browser DOM `onclick=''` events.


ADO JavaScript Client-Only Access Application

ADO Programming Example

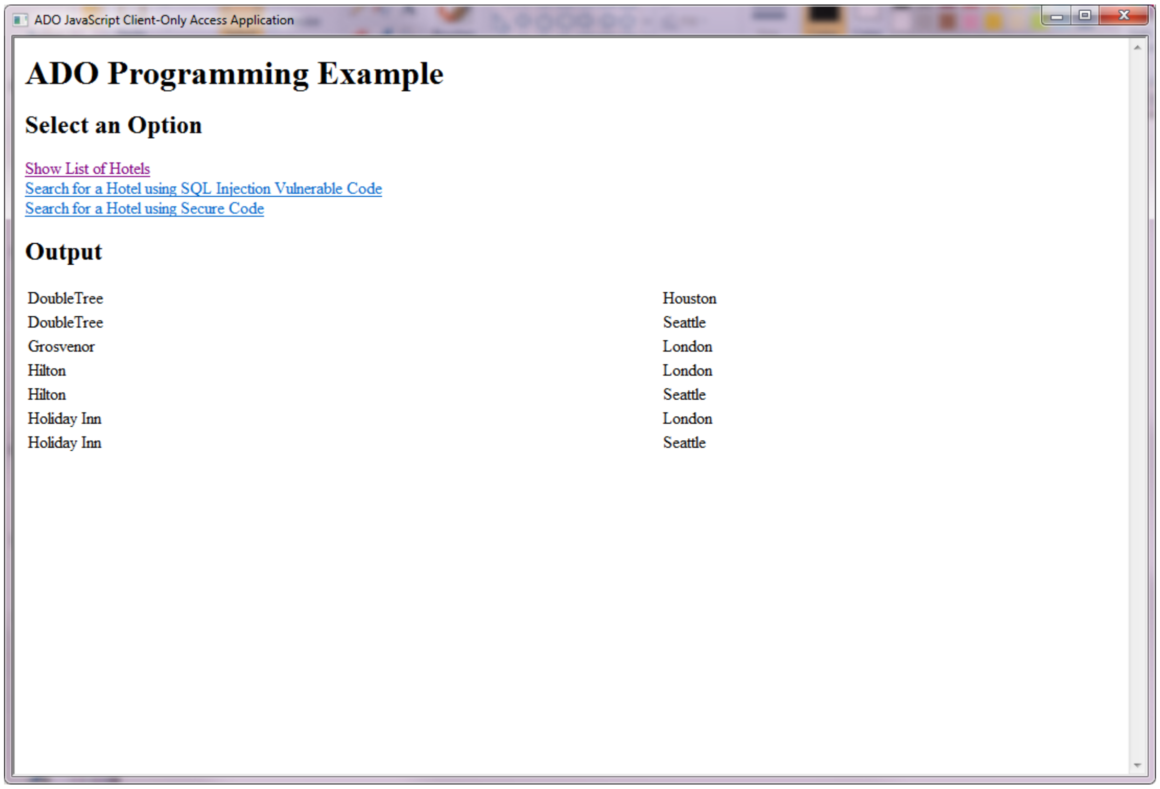
Select an Option

Show List of Hotels
Search for a Hotel using SQL Injection Vulnerable Code
Search for a Hotel using Secure Code

Output

Show List of Hotels

This option executes the `showHotel()` function where executes the SQL statement `SELECT * FROM Hotel ORDER BY hotelName,city` and produces an HTML table with resulting set.



ADO Programming Example

Select an Option

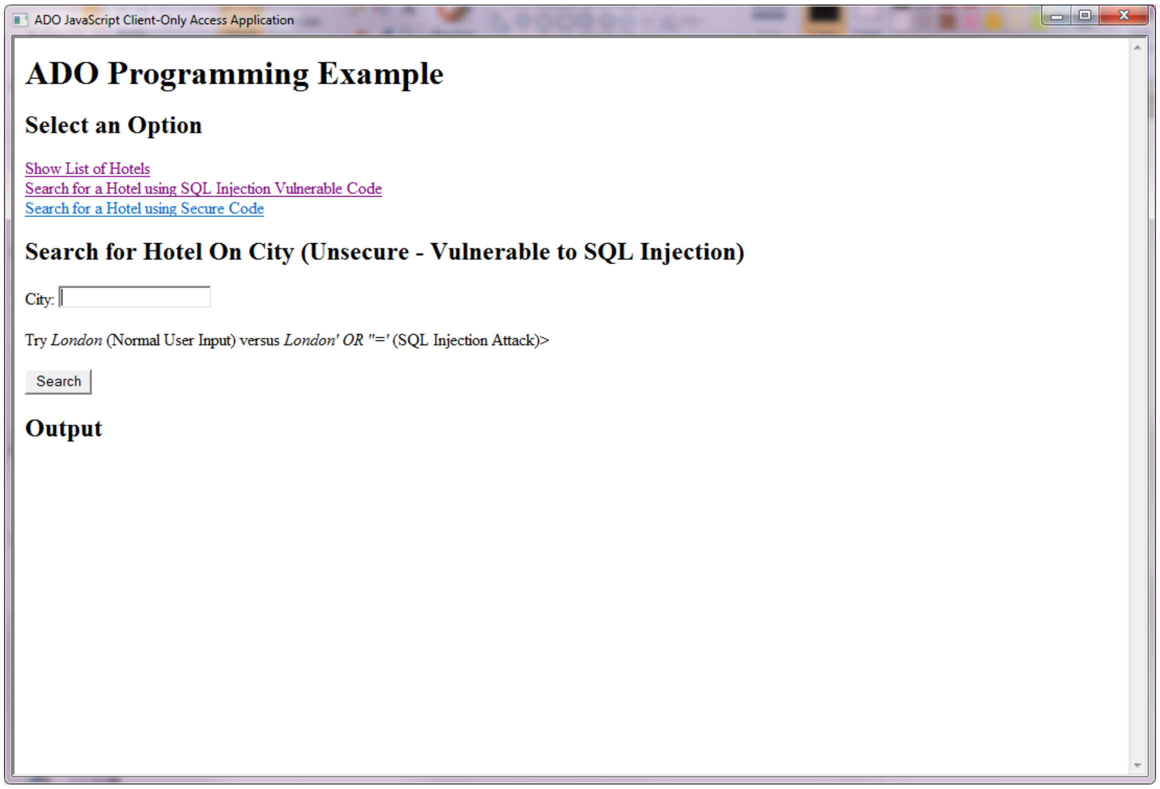
Show List of Hotels
Search for a Hotel using SQL Injection Vulnerable Code
Search for a Hotel using Secure Code

Output

DoubleTree	Houston
DoubleTree	Seattle
Grosvenor	London
Hilton	London
Hilton	Seattle
Holiday Inn	London
Holiday Inn	Seattle

Search for a Hotel using SQL Injection Vulnerable Code

This option calls the `formSearchHotel()` function to display a HTML form to get name of a city to produce a report with a subset of hotels. The option sets up a call to `searchHotelUnsecure()` to show the results.


ADO JavaScript Client-Only Access Application

ADO Programming Example

Select an Option

Show List of Hotels
Search for a Hotel using SQL Injection Vulnerable Code
Search for a Hotel using Secure Code

Search for Hotel On City (Unsecure - Vulnerable to SQL Injection)

City:

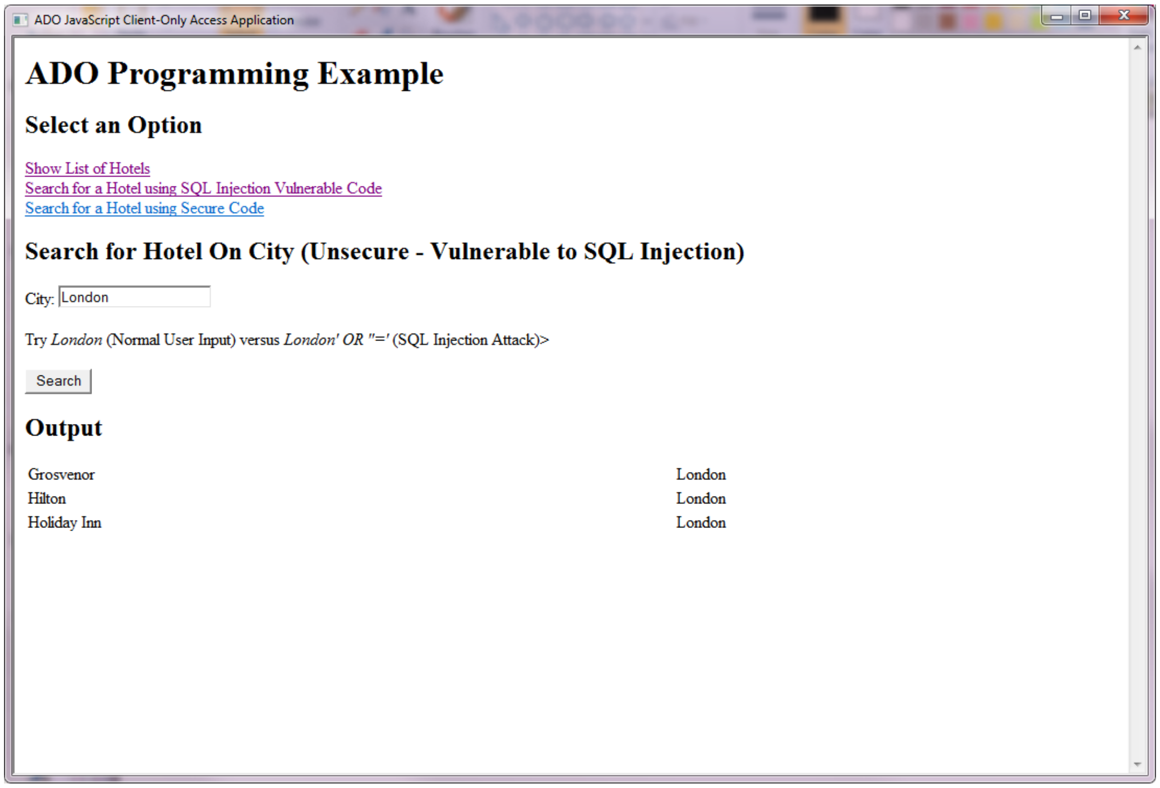
Try *London* (Normal User Input) versus *London' OR ''='* (SQL Injection Attack)>

Search

Output

Normal User

The normal user types the name of a city, selects the Search button, and gets an appropriate report only those Hotel located in that city.


ADO JavaScript Client-Only Access Application

ADO Programming Example

Select an Option

Show List of Hotels
Search for a Hotel using SQL Injection Vulnerable Code
Search for a Hotel using Secure Code

Search for Hotel On City (Unsecure - Vulnerable to SQL Injection)

City: London

Try *London* (Normal User Input) versus *London' OR ''='* (SQL Injection Attack)>

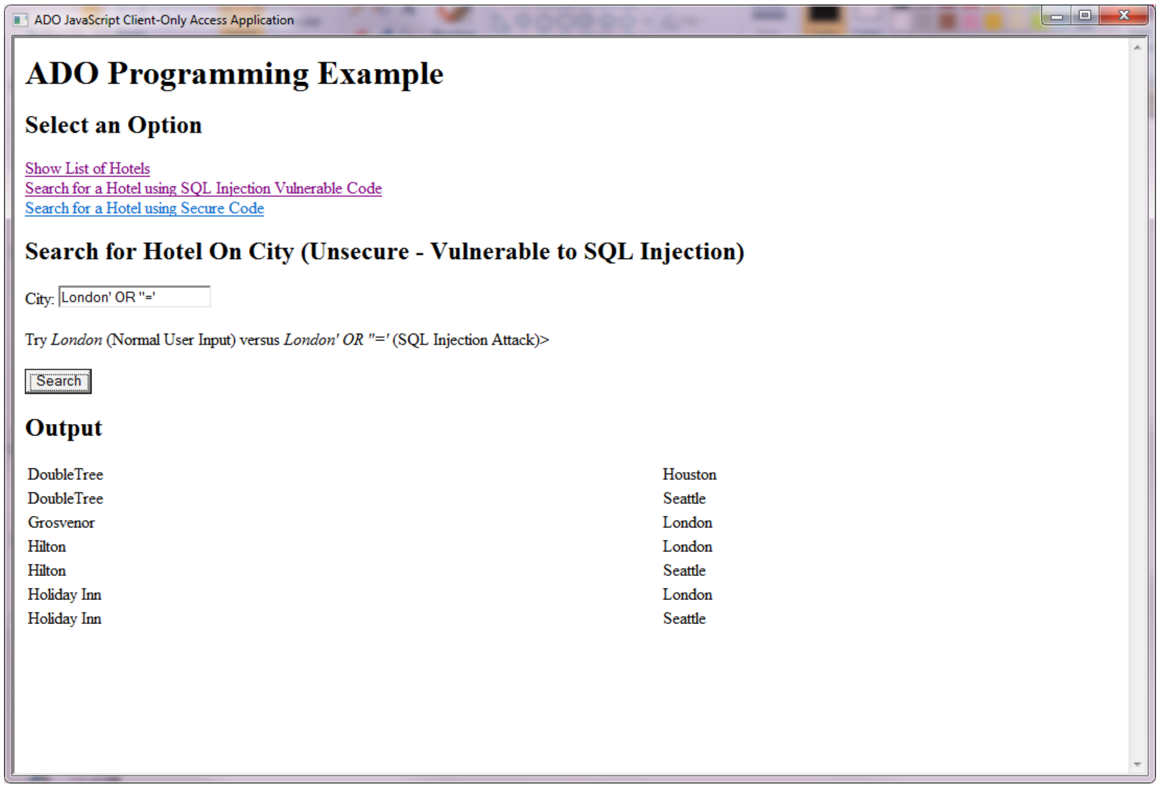
Search

Output

```
Grosvenor                                                  London
Hilton                                                     London
Holiday Inn                                                London
```

Malevolent User

The malevolent user assumes that the web page uses string concatenation to create the SQL `SELECT` statement. The malevolent user exploits this weakness by typing valid SQL clauses as part of the input string. Here the malevolent user makes so that the entire table will be displayed whether or not this user should have had access to all data in the table. This is a type of *information disclosure* attack. Depending on the database programming interface, a malevolent user could do much more serious harm. You should notice that the output from the malevolent user's input is all roles in the table.


ADO JavaScript Client-Only Access Application

ADO Programming Example

Select an Option

Show List of Hotels
Search for a Hotel using SQL Injection Vulnerable Code
Search for a Hotel using Secure Code

Search for Hotel On City (Unsecure - Vulnerable to SQL Injection)

City: London' OR ''='

Try *London* (Normal User Input) versus *London' OR ''='* (SQL Injection Attack)>

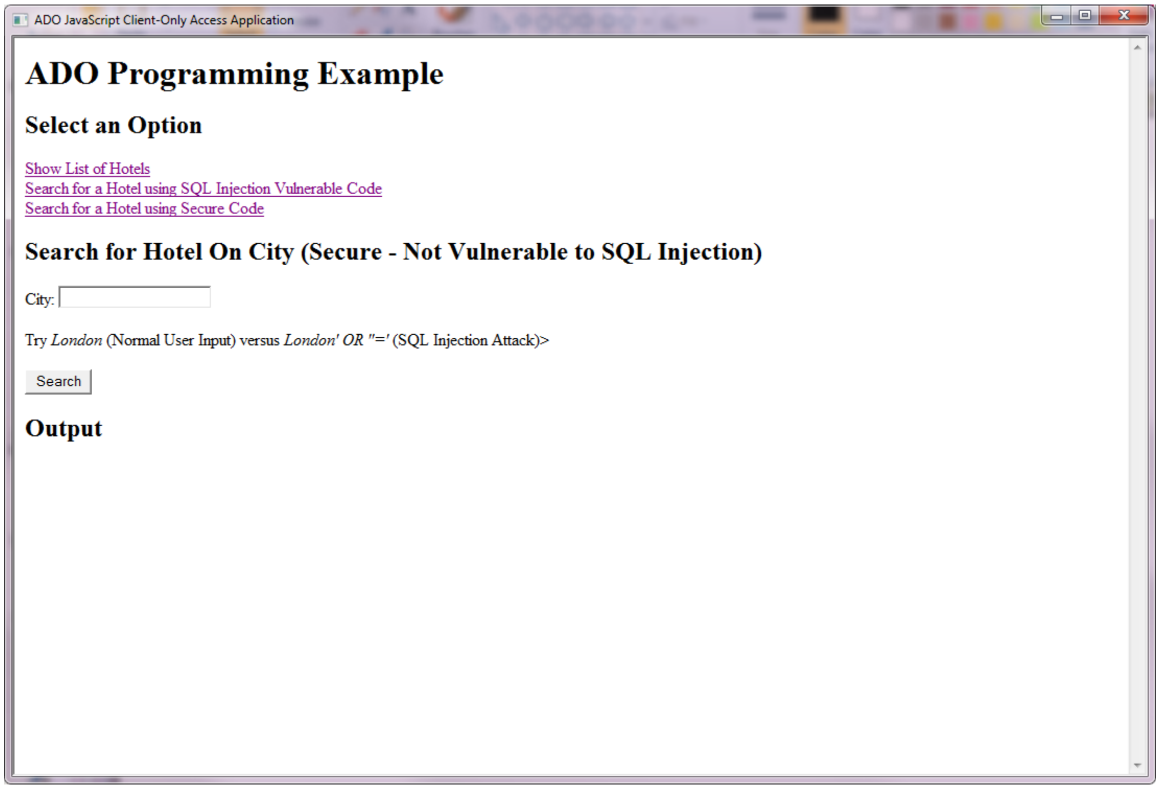
Search

Output

DoubleTree	Houston
DoubleTree	Seattle
Grosvenor	London
Hilton	London
Hilton	Seattle
Holiday Inn	London
Holiday Inn	Seattle

Search for a Hotel using Secure Code

This option calls the `formSearchHotel()` function to display a HTML form to get name of a city to produce a report with a subset of hotels. The option sets up a call to `searchHotelSecure()` to show the results.


ADO JavaScript Client-Only Access Application

ADO Programming Example

Select an Option

Show List of Hotels
Search for a Hotel using SQL Injection Vulnerable Code
Search for a Hotel using Secure Code

Search for Hotel On City (Secure - Not Vulnerable to SQL Injection)

City:

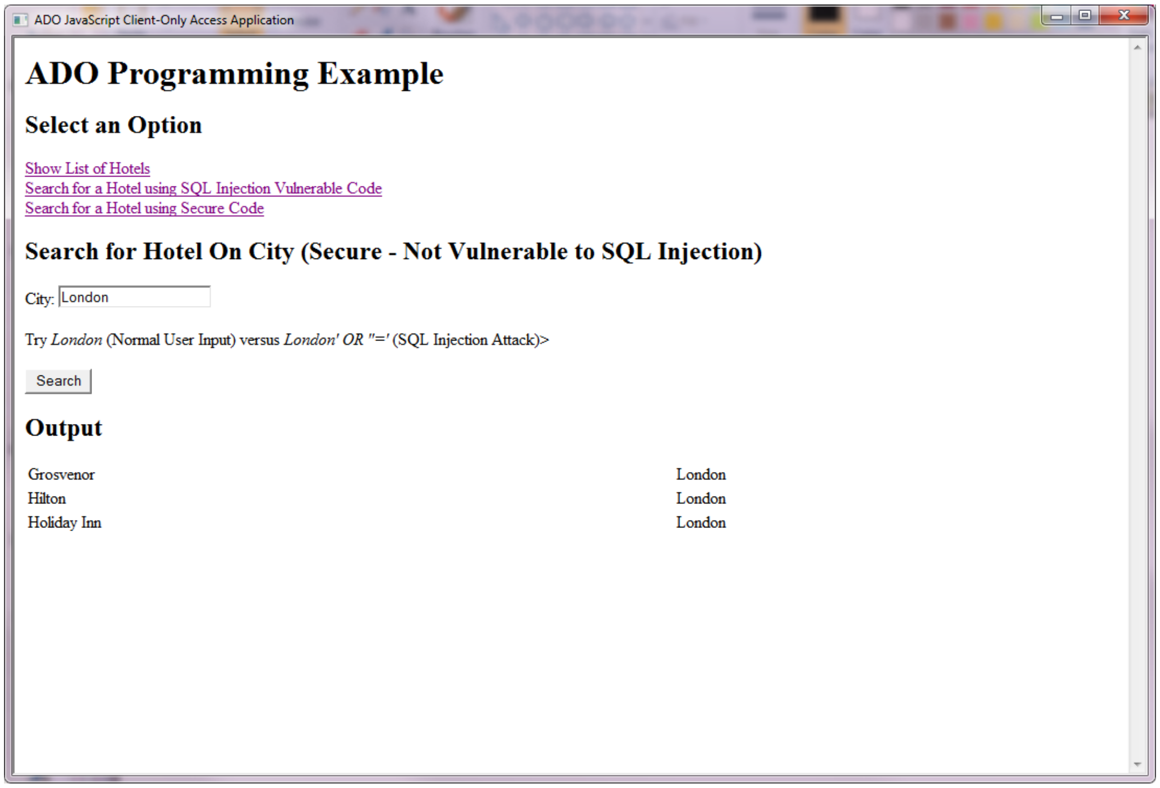
Try *London* (Normal User Input) versus *London' OR ''='* (SQL Injection Attack)>

Search

Output

Normal User

The normal user types the name of a city, selects the Search button, and gets an appropriate report only those Hotel located in that city.


ADO JavaScript Client-Only Access Application

ADO Programming Example

Select an Option

Show List of Hotels
Search for a Hotel using SQL Injection Vulnerable Code
Search for a Hotel using Secure Code

Search for Hotel On City (Secure - Not Vulnerable to SQL Injection)

City: London

Try *London* (Normal User Input) versus *London' OR ''='* (SQL Injection Attack)>

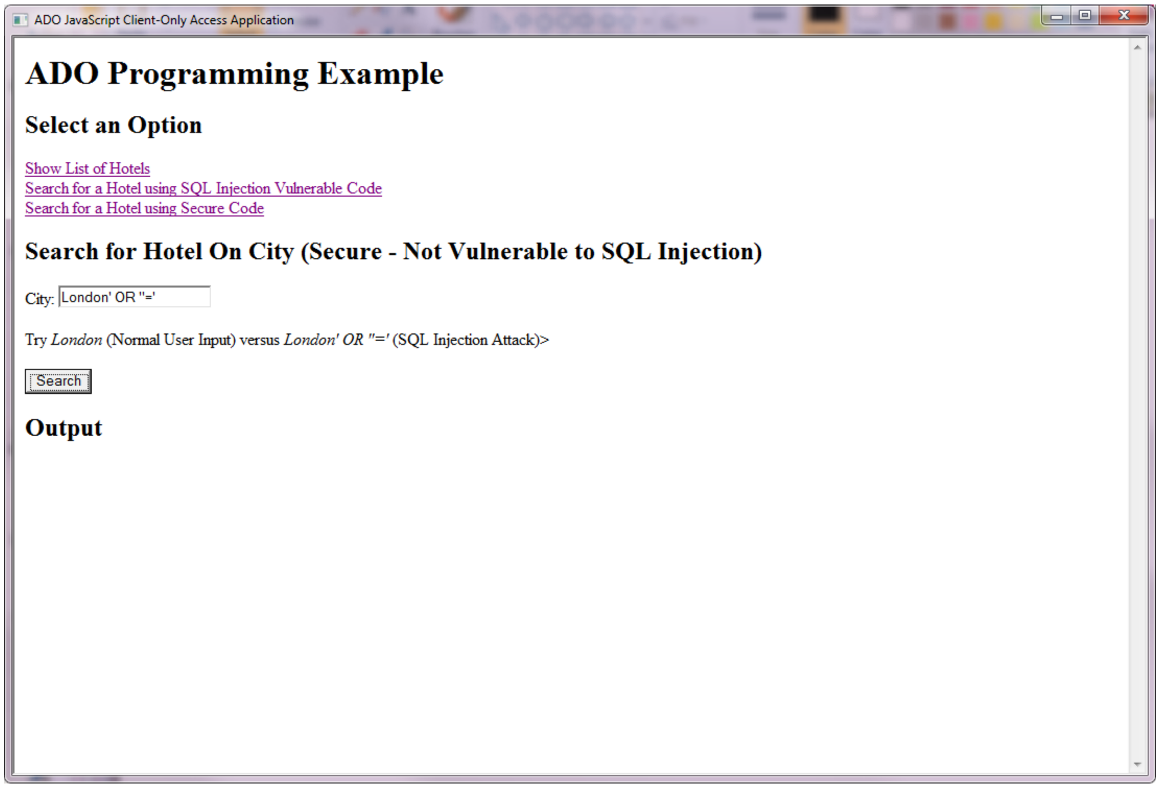
Search

Output

```
Grosvenor                                   London
Hilton                                      London
Holiday Inn                                 London
```

Malevolent User

The malevolent user assumes that the web page uses string concatenation to create the SQL SELECT statement. The malevolent user attempts to exploit that weakness by typing valid SQL clauses as part of the input string; however, the application using secure coding methods and the attack is prevented.



ADO Programming Example

Select an Option

Show List of Hotels
Search for a Hotel using SQL Injection Vulnerable Code
Search for a Hotel using Secure Code

Search for Hotel On City (Secure - Not Vulnerable to SQL Injection)

City: London' OR ''='

Try *London* (Normal User Input) versus *London' OR ''='* (SQL Injection Attack)>

Output