Protecting Connection Information

- 03/29/2017
- 3 minutes to read

In this article

Protecting access to your data source is one of the most important goals when securing an application. A connection string presents a potential vulnerability if it is not secured. Storing connection information in plain text or persisting it in memory risks compromising your entire system. Connection strings embedded in your source code can be read using the Ildasm.exe (IL Disassembler)[1] to view Microsoft intermediate language (MSIL) in a compiled assembly.

Security vulnerabilities involving connection strings can arise based on the type of authentication used, how connection strings are persisted in memory and on disk, and the techniques used to construct them at run time.

Use Windows Authentication

To help limit access to your data source, you must secure connection information such as user ID, password, and data source name. In order to avoid exposing user information, we recommend using Windows authentication (sometimes referred to as *integrated security*) wherever possible. Windows authentication is specified in a connection string by using the `Integrated Security` or `Trusted_Connection` keywords, eliminating the need to use a user ID and password. When using Windows authentication, users are authenticated by Windows, and access to server and database resources is determined by granting permissions to Windows users and groups.

For situations where it is not possible to use Windows authentication, you must use extra care because user credentials are exposed in the connection string. In an ASP.NET application, you can configure a Windows account as a fixed identity that is used to connect to databases and other network resources. You enable impersonation in the identity element in the **web.config** file and specify a user name and password.

```
<identity impersonate="true"
        userName="MyDomain\UserAccount"
        password="*****" />
```

The fixed identity account should be a low-privilege account that has been granted only necessary permissions in the database. In addition, you should encrypt the configuration file so that the user name and password are not exposed in clear text.

Do Not Use Universal Data Link (UDL) files

Avoid storing connection strings for an OleDbConnection[2] in a Universal Data Link (UDL) file. UDLs are stored in clear text and cannot be encrypted. A UDL file is an external file-based resource to your application, and it cannot be secured or encrypted using the .NET Framework.

Avoid Injection Attacks with Connection String Builders

A connection string injection attack can occur when dynamic string concatenation is used to build connection strings based on user input. If the user input is not validated and malicious text or characters not escaped, an attacker can potentially access sensitive data or other resources on the server. To address this problem, ADO.NET 2.0 introduced new connection string builder classes to validate connection string syntax and ensure that additional parameters are not introduced. For more information, see Connection String Builders[3].

Use Persist Security Info=False

The default value for `Persist Security Info` is false; we recommend using this default in all connection strings. Setting `Persist Security Info` to `true` or `yes` allows security-sensitive information, including the user ID and password, to be obtained from a connection after it has been opened. When `Persist Security Info` is set to `false` or `no`, security information is discarded after it is used to open the connection, ensuring that an untrusted source does not have access to security-sensitive information.

Encrypt Configuration Files

You can also store connection strings in configuration files, which eliminates the need to embed them in your application's code. Configuration files are standard XML files for which the .NET Framework has defined a common set of elements. Connection strings in configuration files are typically stored inside the **<connectionStrings>** element in the **app.config** for a Windows application, or the **web.config** file for an ASP.NET application. For more information on the basics of storing, retrieving and encrypting connection strings from configuration files, see Connection Strings and Configuration Files[4].

See also

Links

1. https://docs.microsoft.com/en-us/dotnet/framework/tools/ildasm-exe-il-disassembler
2. https://docs.microsoft.com/en-us/dotnet/api/system.data.oledb.oledbconnection
3. https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/connection-string-builders
4. https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/connection-strings-and-configuration-files