Global Information Assurance Certification Paper

Securing SQL Connection String

Dmitry Dessiatnikov

January 8, 2004
GIAC Security Essentials Certification Practical Assignment -- version 1.4b
Option 1

Abstract

Securing authentication information used to establish connection between two applications is one of the most critical aspects of application security. This paper will focus on protecting connection strings used to authenticate communication between the web server and the back-end database. We will discuss and evaluate the vast array of options available for storage and protection of the connection strings. Because connection strings are dependent on the type of data source used, we will be specifically referring to the connection strings used to connect to the SQL Server in the Windows environment.

Introduction

Today, a distributed computing environment is an integral part of core business operations. Information system environments of most companies are complex and require the integrated functionalities of a large number of applications. Most of these applications need to communicate, pass data and exchange functionalities in order to accomplish a number of complex processes. In order to prevent unauthorized access or abuse of the established connections, communication between applications is established in the authenticated fashion. Connection strings contain authentication information used by the applications to connect to the data source, which in many cases is a database.

With the development and growth of the public Internet, the need to prevent unauthorized access through the Web enabled application has grown drastically. Most of the e-commerce websites collect or display some type of information to the end users. This information is commonly stored in the database that is connected to the web server. Thus a database in most cases is the depository of critical and often sensitive in nature information. It becomes critical to protect connection strings used to authenticate to the database from unauthorized access.

Connection String Properties

Connection strings contain vital information about the application itself and details about the type of connection established between the web application and the data source. Connection strings may differ by the type of data source, providers, or drivers used for the connection. Some of the critical properties of the connection specified in the connection string include:

- Hostname or IP address of the server housing the database
- The type of data source

- The type of technology used to communicate with the data source
- Name of the database containing the data
- Network libraries used for the connection
- The port number used for the connection
- Username and password for the account used to authenticate to the database (Andrews, p.262).

The basic format of a connection string is a list of keywords and values, which are separated by the semicolon. Values are assigned to the key names with an equal sign. The connection string will change greatly depending on the "provider" (OLE DB) or "driver" (ODBC) used because the provider/driver identifies the type of technology used to establish the connection (Andrews, p.262). Table 1 lists some of the common providers:

Connection Type	**Name/Value Pair**
SQL Server using ODBC	Driver={SQL Server}
Microsoft OLE DB Provider for Oracle	Provider=MSDAORA
Microsoft OLE DB Provider for Jet	Provider=Microsoft.Jet.OLEDB.4.0
Microsoft OLE DB Provider for IBM DB2	Provider=DB2OLEDB
Microsoft OLE DB Provider	Provider=SQLOLEDB
Microsoft Excel using ODBC	Driver={Microsoft Excel Driver (*.xls)}

Table 1 (Andrews, p. 263) – *Sample Providers/Drivers for Use in Connection Strings*

Consider sample connection string used for the OLE DB method of connecting to the SQL Server:

"Provider=sqloledb;Data Source=sqlservername;Uid=username;
Pwd=password;Database=databasename;"

As demonstrated, the connection string contains sufficient information to compromise the database server and to gain unauthorized access to the data stored in the database. Embedding the connection strings in the source code remains to be common coding practice, regardless of the associated risks.

SQL Authentication vs. Windows Authentication

The web server can authenticate to Microsoft SQL server by using either the Windows authentication mode or the SQL authentication mode. The Windows authentication does not require authentication information used to connect to SQL server to be stored in the connection string, thus making it more secure. In case of Windows authentication, SQL server has to validate user credentials with the Windows Domain Controller for domain level users or Windows server itself for local level accounts. This requires trusted connection between SQL server and the web server. However, the common security practice is to separate the web server and SQL server with the firewall and not to pass Windows credentials from one network to another through the firewall. Because for security reasons connection between the web server and SQL server is in 'nontrusted' context and SQL authentication

does not require trusted relationship, SQL authentication is more commonly used for connection between the web server and SQL server then Windows authentication.

There are more benefits to using Windows authentication than the fact that it does not require the storage of username and password in the connection string. Because Windows authentication relies on the Windows security, it provides such security features as: password encryption, password complexity enforcement, password expiration with account lockout threshold and extensive auditing.

Although SQL native authentication is inherently less secure than Windows authentication, the use of Windows authentication from the development standpoint is not very practical. SQL authentication allows for the application to be portable and capable of being used for connection with other databases that may not allow for integrated authentication. In addition, if Windows credentials were compromised, in order to compromise SQL database, an attacker would have to obtain SQL credentials. Thus, using SQL authentication would offer an additional layer of security that an attacker would have to bypass.

Storage of connection strings

The best way of protecting a secret is not to have one; however, secrets are a necessary evil when it comes to accomplishing certain functionalities, in this case establishing the communication channel between the web server and SQL server. Once SQL authentication is selected as the preferred method of authenticating the web server to the SQL server, protection of the connection strings becomes a critical issue. While a full proof protection of secrets using software is debatable, a number of techniques exist that accomplish the goal of safeguarding the connection string content. Existing methods can be grouped into three main techniques: hiding or security through obscurity, access control, and encryption.
(Davis,http://msdn.microsoft.com/msdnmag/issues/03/11/Prot ectYourData/default.aspx)

Security through obscurity

Very minimal, if any security is offered by hiding the connection string on the web server file system. Because application needs to know where to find it for the authenticated connection to be established, it is only a matter of time until unauthorized users will locate it as well. Security through obscurity offers a false sense of security and simply makes bad guys work harder for getting that crown jewel of information stored in the connection string. To that end there is no best solution, but the one that would be the most difficult for others to figure out. To mitigate this risk, place the connection string outside of the web server file system, because the most prevalent web server vulnerabilities are related to gaining unauthorized access to the file system.

The locations that are commonly used for connection strings include: compiled source code, Windows registry, configuration file and COM+

catalog. The list of alternatives continues to grow and by no means is limited to the above-mentioned options. The inherent security risks associated with each of these locations will be discussed below.

Compiled source code

Storing connection strings in the compiled source code offers a better solution then embedding them in a clear text in ASP pages that can be opened with such text editors as notepad. Often these pages are stored in the INETPUB directory on the web servers. An intruder who gained access to the file system of a web server through a number of directory traversal vulnerabilities would be able to read authentication information on the ASP page. In addition, web server flaws revealed a number of vulnerabilities that allowed attackers to view the ASP code through the browser, thus exposing connection strings in the source code.

The issue with compiled source code is that it can be decompiled or reverse engineered and the connection string can be read from the recreated source code file. A number of tools capable of reverse engineering source code are freely available for download from the Internet. One of these tools called Anakrino available from http://www.saurik.com/net/exemplar is used to reveal authentication information stored in the connection string in the Figure 1 below.

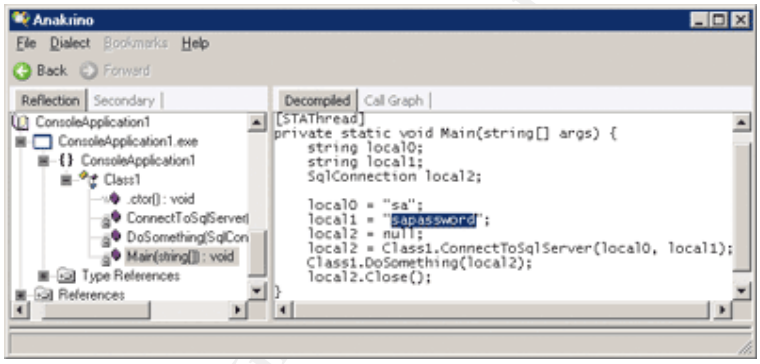

Figure 1 – Anakrino
(Davis, http://msdn.microsoft.com/msdnmag/issues/03/11/ProtectYourData/default.aspx)

Embedding connection strings in source code offers high performance because application does not have to reference an external source, but at the price of maintainability because changes in the connection string would require the application to be recompiled.

COM+ catalog

A worthy alternative to placing business logic in the presentation layer, which can be uncovered in a number of ways, is to create a true n-tier application. In an n-tier application, data, presentation and business logic are placed in 3 or more layers, thus creating more hoops for an attacker to pass through before they get to the crown jewel of SQL server authentication information. In

addition to the security benefits, an n-tier application offers the benefits of being more robust, flexible and scalable as data, presentation, and business logic can be copied or updated independently of each other.

There are a number of ways of implementing an n-tier application to hide the SQL connection string from prying eyes. One of them would be to hide all the details about the connection in a COM object and to simply call this object from the presentation layer. The COM object can even be stored on a separate system, thus cleaning the code stored on the web server from usernames and passwords used to connect to a SQL server. This COM object would be placed on the business layer separate from the presentation layer residing on the web server.

What is a COM object? From the Microsoft web site we learn: "The Component Object Model (COM) is a software architecture that allows applications to be built from binary software components." (Microsoft, http://www.microsoft.com/com/tech/com.asp). COM+ is an extension of COM architecture along with DCOM. COM+ catalog stores all the COM+ configuration data and during any kind of COM+ administration data is read and written to the COM+ catalog. COM+ catalog can be accessed through the Component Services administrative tool or through the COMAdminCatalog class. Only a user with Administrative rights can change data on the COM+ catalog. COM+ catalog stores data hierarchically in a number of collections such as applications and components. Connection string can be stored in the COM+ catalog for a specific component and then retrieved when an object of the component is activated.
(Aziz,http://www.csharphelp.com/archives3/archive482.html).

Windows Registry

Windows registry is a central hierarchical database used to store configuration information and other settings unique to the system. Registry is constantly referenced and updated while the system is being used. A number of applications store authentication information in the registry, so it would seem an appropriate placeholder for the connection string.

Storing the connection string in the registry would remove it from the file system, which has always been the target of attacks on the web server. The main drawbacks of using the registry as the connection string repository are that other people can access the registry and that the performance suffers when an application has to access the external source for authentication information.

Configuration files

In classic ASP applications, the global.asa file contains configuration information and it has been a common practice to place connection strings in global.asa file using the Application object.

Consider a sample global.asa file containing connection string:

```
<script language="vbscript" runat="server">
sub Application_OnStart
        Application("ConnectionString") = _
"Provider=SQLOLEDB.1; Data Source=sqlserver;Initial Catalog=sqldatabase;
Uid=sa;Pwd=password; "
end sub
</script>
```

In the above sample, Application_OnStart is an event handler that is invoked when the ASP page is first requested. Once the user accesses the first ASP page, Application_OnStart generates the connection string and passes it to the ASP source code. This poses a risk due to a large number of reveal source code vulnerabilities, which exposes the web servers' source code to the preying eye. Authentication information can be prevented from being passed to the source code. Instead of placing the connection string with the authentication information in the global.asa file as in the example above, the connection string can be only referenced in the global.asa file in the web server root directory by the name "strConnection". In this case, if an attacker gains access to the ASP code, he or she would see only reference to the connection string and no authentication information. (McLeavy,http://www.naspa.com/PDF/2002/0502%20PDF/T0205008.pdf).

Extract of ASP source code that does not reveal authentication information is demonstrated below:

```
<%
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open Application("strConnection")
Set RS=Conn.Execute("Select * from Table1")
>%
```
(McLeavy,http://www.naspa.com/PDF/2002/0502%20PDF/T0205008.pdf).

The actual authentication information would be located in the global.asa file and not passed to the ASP source code.

The ASP.NET web.config file serves the purpose of repository of the configuration settings that apply to the application as a whole. The Web.config file is one of the common storage locations for the connection strings in ASP.NET applications. Connection strings are places in the AppSettings section within the Configuration section and before the System.web section. (Smith, http://authors.aspalliance.com/stevesmith/articles/dotnetc onnectionstrings.asp).

A sample web.config file is shown below:

```
<configuration>
  <appSettings>
    <add key="ConnectionString"
      value="DataSource=sqlserver;InitialCatalog=sqldatabase;
```

```
Uid=sa;Pwd=password;"/>
  </appSettings>
  <system.web>
    <customErrors mode="Off"/>
  </system.web>
</configuration>
```

By default, web servers should not reveal the content of the global.asa and web.config files if a web user is requesting them through the browser. However, this layer of security can be bypassed if an attacker gains access to the source code due to source disclosure vulnerability or if he or she obtains read rights to the file system.

Control access to data

Solely hiding connection strings from the prying eye has proven to be insufficient for complete security of authentication information. Even though the number of locations used for storage of the connection strings continues to grow, because application needs to be able to find it, so will the bad guys only in a matter of time. What additional measures can be employed in order to help solve the limitations offered by the security through obscurity concept? How about controlling the data itself once its location has been identified?

Windows offers a number of ways to control access to the various locations used for connection string storage. Starting with the NTFS permissions on the individual files containing usernames and passwords, which would at least stop those with unprivileged access from accessing the files at the file system level. Registry keys can be protected with the Access Control Lists (ACLs), which can be used to tighten security at the granular level according to the privileges assigned.

Access can be restricted based on something the user requesting access knows or is. In other words, a legitimate user can be accepted on the basis of a verified caller's identity or on the basis of the knowledge of a common secret. Restricting access based on these criteria can be an effective measure in securing data if applied effectively, but it is not without its limitations. In the case of using knowledge of a common secret as a criterion for valid authentication, the issue of storing the common secret in a secure location raises a number of known security issues. Identitybased access control is not flawless neither. It will not be effective with such applications as ASP.NET, which run under the identity of an anonymous user, nor will it work for the application running under the multiple users, because they could access each other's data and data would not be set and retrieved by the same user. (Davis, http://msdn.microsoft.com/msdnmag/issues/03/11/ProtectYourData/default.aspx).

Encrypting data

From the above discussion, a conclusion can be drawn that both hiding the connection string and restricting access to it through identity based and

knowledge based access controls is less than secure. The final option is to store sensitive information in such a form that it would be meaningless to an unauthorized user. This way an attacker would not benefit from locating the placeholder for the connection string and bypassing access controls used to restrict access to the authentication information. Encrypting data offers the final obstacle, which can be an effective measure if it is implemented correctly.

The issue with using encryption to protect the connection string is that now the focus is passed from securing the connection string to securing the key used to decrypt the connection string. The concern with placing the encryption key in a secure location and then restricting access to that key still applies The question is who is responsible for storing the encryption key and how effective are the security measures surrounding the storage of that key.

A common method of encrypting and decrypting data is to use Microsoft Data Protection Application-Programming Interface (DPAPI) introduced with Windows 2000. DPAPI is a password-based data protection service on the Operating System (OS) level with no additional libraries required. Because DPAPI interfaces are implemented in crypt32.dll as part of Microsoft Cryptography Application-Programming Interface (CryptoAPI), most of the Windows systems have this functionality available to them. Every application on the system can take advantage of this encryption service without needing to handle any specific cryptographic code other than making function calls to DPAPI. The weakness of DPAPI lies in the fact that DPAPI encryption is based on the provided password. This, however is offset by the use of the Triple DES algorithm and strong encryption keys. (NAI Labs, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsecure/html/windataprotection-dpapi.asp).

DPAPI generates either user- or machine-specific encryption keys. The use of these keys is mutually exclusive, which means that only one type of encryption key can be used in one DPAPI call. When the user-specific key is used, only the user that encrypted the data can decrypt it. This requires the application to be running under the profile of a local or domain user and the DPAPI calls can be made on the system where the profile was created. User-specific keys cannot be used with the built-in system accounts used by such applications as ASP.NETbecause these accounts do not login interactively. With the machine-specific encryption keys, any application running on the same system can encrypt and decrypt the data. This capability can be further restricted to a particular caller by specifying an additional secret value called entropy. This way only applications with entropy can encrypt and decrypt particular data. An Additional secret can be used with both the user- and machine-specific encryption keys.
(Obviex,http://www.obviex.com/samples/dpapi.aspx).

The pros of using DPAPI include the fact that it is part of the operating system and thus freely available on most Windows systems. Overall it provides a relatively simple method for protecting the confidentiality of data and does not pass the responsibility of key handling to the end user. The disadvantages

include the inability to reuse the encryption key on the system other than the system, where the connection string was originally encrypted. The immobility of code ties back down to the fact that DPAPI is a password- and a OS-based cryptographic service.

Additional measures

None of the above measures are 100 percent fullproof. However, each of these measures, when implemented effectively reduces the risk of unauthorized access to the sensitive information stored in the database. In addition to hiding the connection string, restricting access to its content and encrypting it, a number of additional measures may be implemented to improve the security of an application and SQL server connected to it.

The principle of least privilege states that minimum access rights should be given to a user profile to accomplish its required functions. This principle should be applied to the account used for the connection between the web server and SQL server. If connection string were compromised, an attacker would have the SQL database rights that apply to that account. If the application does nothing but returns values from the database and SQL authentication is used for the connection, it must be ensured that this account has only read rights on the database. To achieve higher granularity in restricting access, this account should be assigned read only rights on the specific tables in that database. This granularity can be achieved in the MSSQL 2000 server through the Enterprise Manager.

In addition to restricting access to the data stored on the database, it must be ensured that an account used to connect to the database from a web server has execute rights only on the stored procedures that are required for it to accomplish its tasks. SQL server comes with many useful features available through the use of system and extended stored procedures. These stored procedures were originally designed to allow for extended support of such tools shipped with MSSQL server as Enterprise Manager and Query Analyzer. Misconfigured SQL server allows for these procedures to be executed by non-privileged accounts, thus exposing SQL server to unnecessary risks. If the connection string is compromised, an attacker may be able to login into the SQL server with the account stored in the connection string using such tools as SQL Analyzer. If this account has execute rights to such stored procedures as xp_cmdshell and xp_regread, an attacker may be able to execute commands on the operating system with the privileges of SQL service account and read the registry.

Avoid using an sa account when establishing connection between a web server and the database because of the high level of privileges that are assigned to that account. Since this is the superuser account in SQL database, by default it has complete control over the database, including the dangerous, stored procedures.

By default, MSSQL server is installed as a LocalSystem level account, which is a member of Local Administrators group on a Windows system and has

complete control over the system resources. In the scenario above, due to a number of SQL server misconfigurations, an attacker may be able to execute commands at the operating system level with the rights of SQL Service account. If this account has Administrative rights, so will the attacker. That is why SQL server service account should only have minimal rights and should help contain an attack to the server in case of a compromise. This account is required to run with the service rights only. A local user account is best for non-replicated servers and a domain user account is best for the servers that require replication or connections to remote servers.

To improve overall security, restrict access to the SQL server to the IP address of the web server. This way, if an attacker compromises the connection string, he or she would not be able to login from any other system but the web server.

Conclusion

Benjamin Franklin once said, “The three can keep the secret, if two are dead”. In this paper we discussed and critiqued from the security standpoint the vast array of options available for storage and protection of the SQL connection strings. There is no solution that would offer the complete protection of authentication information stored in the connection strings, however, the mix of the above mentioned measures would make compromise of sensitive information difficult. We pointed out that all the measures focusing on protection of connection strings can be grouped into three main areas: hiding connection strings, restricting access to them, and encrypting them. (Davis, http://msdn.microsoft.com/msdnmag/issues/03/11/ProtectYourData/default.aspx). Since there is no best solution, the least worst solution should be selected based on the application requirements and established security policy.

The need for securing applications is becoming more apparent as the tools and knowledge used to compromise them is becoming freely available on the Internet. The increasing rise of the automated tools lowers the technical expertise required for attack execution, thus causing the growth of people capable of exploiting unprotected applications.

References

1. Andrews, Chip; Litchfield, David and Grindlay, Bill. SQL Server Security. New York: McGraw-Hill/Osborne, 2003.
2. Davis, Alek. “Safeguard Database Connection Strings and Other Sensitive Settings in Your Code” MSDN Magazine. November 2003. URL: http://msdn.microsoft.com/msdnmag/issues/03/11/ProtectYourData/default.aspx (Jan 6, 2004)
3. McLeavy, Justin. “Using Global.asa to tighten ASP/SQL Security” Coding corner. May 2002. URL:http://www.naspa.com/PDF/2002/0502%20PDF/T0205008.pdf (Jan 7, 2004)

4. Smith, Steven. “ASP.NET: Connection Strings” ASPAllience.com. 8/11/2001. URL:http://authors.aspalliance.com/stevesmith/articles/dotnetconnectio nstrings.asp. (Jan 7, 2004)
5. Microsoft. Microsoft.com technologies. 3/30/99 URL:http://www.microsoft.com/com/tech/com.asp (Jan 7, 2004)
6. Aziz, Asad. “COM+ Automation using .NET C#” URL:http://www.csharphelp.com/archives3/archive482.html (Jan 8, 2004)
7. NAI Labs. “Windows Data Protection”. Network Associates, Inc. MSDN Library. October 2001. URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsecure/html/windataprotection-dpapi.asp (Jan 8, 2004)
8. Obviex “How To: Use DPAPI to Encrypt and Decrypt Data (C#)” 9/13/2003. URL:http://www.obviex.com/samples/dpapi.aspx (Jan 8, 2004)