Connection String Parameter Pollution Attacks

Chema Alonso[1], Manuel Fernandez[1], Alejandro Martín[1] and Antonio Guzmán[2]

[1]Informatica64, S.L.
[2]Universidad Rey Juan Carlos
[1]{chema,mfernandez,amartin}@informatica64.com, [2]antonio.guzman@urjc.es

Abstract. In 2007 the ranking of the top ten critical vulnerabilities for the security of a system established code injection as the top 2, closely following top 1 XSS attacks. The first release candidate of the 2010 version of the ranking has promoted code injection attacks to top 1. Actually, the most critical attacks are those that combine XSS techniques to access systems and code injection techniques to access the information. The potential damage associated with this kind of threats, the total absence of background and the fact that the solution to mitigate these vulnerabilities must be worked together with programmers, systems administrators and database vendors justifies an in-depth analysis to estimate all the possible ways of implementing this technique.

Keywords: Code injection attacks, connection strings, web application authentication delegation.

1 Introduction

SQL injections are probably the most known injection attacks to web applications by abusing its database architecture. Many different approaches and techniques have been studied and analyzed so far, and the published results conclude that to prevent these attacks from being successful, development teams need to establish the correct filtering levels on the inputs to the system.

In the case of the attack presented in this paper, responsibility lays not only on developers, but also on system administrators and database vendors. This attack affects web applications, but instead of abusing implementation flaws in the way database queries are crafted, which is the most commonly found scenario on other injection attacks, it abuses the way applications connect to the database.

According to OWASP [1], in 2007 the ranking of the top ten critical vulnerabilities for the security of a system established code injection attacks as the top 2, closely following top 1 XSS attacks. The first release candidate of the 2010 version of the ranking has promoted code injection attacks to top 1. Actually, the most critical attacks are those that combine XSS techniques to access systems and code injection techniques to access the information. This is the case for the so-called *connection string parameter pollution* attacks. Potential impact of this type of vulnerability and the total absence of background justify an in-depth analysis to estimate all possible attack vectors using this technique.

This paper is structured is in three main sections. The first is this short introduction where the foundations of the connection strings and existing mechanisms for the implementation of web applications authentication will be introduce. Section two proposes a comprehensive study of this new attack technique, with an extensive collection of test cases. The article concludes briefly summarizing the lessons learned.

1.1 Connections Strings

Connection strings [2] are used to connect applications to database engines. The syntax used on these strings depends on the database engine to be connected to and on the provider or driver used by the programmer to establish the connection.

One way or another, the programmer must specify the server and port to connect to, the database name, authentication credentials, and some connection configuration parameters, such as timeout, alternative databases, communication protocol or encryption options.

The following example shows a common connection string used to connect to a Microsoft SQL Server database:

```
"Data Source=Server,Port; Network Library=DBMSSOCN;
Initial Catalog=DataBase; User ID=Username;
Password=pwd;"
```

As the example shows, a connection string is a collection of parameters separated by semicolons (;), each parameter being a key-value pair. The attributes used in the example correspond to the ones used in the ".NET Framework Data Provider for SQL Server", which is chosen by programmers when they use the "SqlConnection" class in their .NET applications. Obviously, it is possible to connect to SQL Server using different providers such as:

- ".NET Framework Data Provider for OLE DB" (OleDbConnection)
- ".NET Framework Data Provider for ODBC" (OdbcConnection)
- "SQL Native Client 9.0 OLE DB provider"

The most common and recommended way to connect a .NET based application and a SQL server, is to use the framework default provider, where the connection string syntax is the same regardless the different versions of SQL Server (7, 2000, 2005 and 2008). This is the one used in this article to illustrate the examples.

1.2 Web Application authentication delegation

There are two ways of defining an authentication system for a web application: create an own credential system, or delegate it to the database engine.

In most cases, the application developer chooses to use only one user to connect to the database. Seen from the database side, this database user represents the entire web application. Using this connection, the web application will make queries to a custom users table where the user credentials for the application are stored.

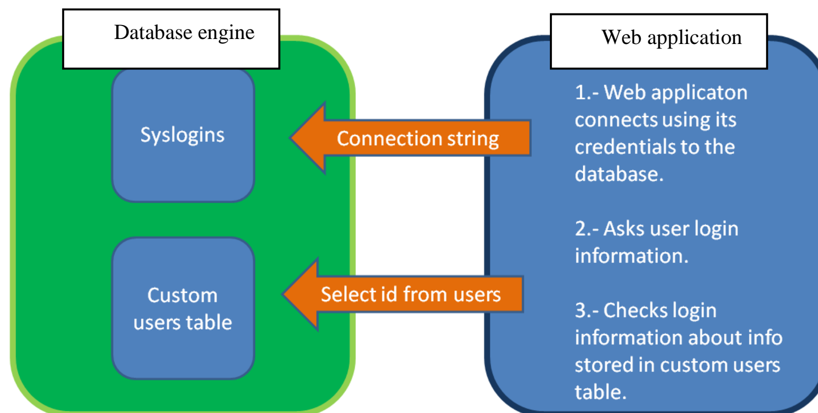

Fig. 1. Common web application authentication architecture

The web application is identified by a single database user with access to the entire application content in the database, thus it is impossible to implement a granular permission system in the database over the different object, or to trace the actions of each user in the web application, delegating these tasks to the web application itself. If an attacker is able to abuse some vulnerability in the application to access the database, the whole database will be exposed. This architecture is very common, and can be found in widely used CMS systems such as Joomla, Mambo and many others. Usually, the target of the attacker is to get the application users credentials from the users table in the database.

The alternative consists in delegating the authentication to the database engine, so that the connection string does not contain a fixed set of credentials, but will use those entered by the application user and it is the database engine responsibility to check them.

Database management applications always use this delegated authentication, so that the user connecting to the application will only be able to access and control those objects and actions for which he has permissions. With this architecture, it is possible to implement a granular permission system and to trace user actions in the database.

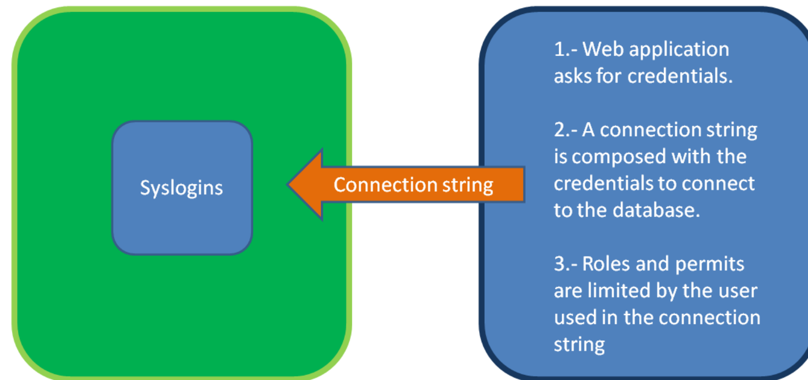

Fig. 2. Web application delegated authentication architecture.

Both methods offer different advantages and disadvantages, apart from the ones already mentioned, which are outside the scope of this article. The techniques described in this paper will focus on the second environment: web applications with delegated authentication to the database engine.

2 Connection String Injection

In a delegated authentication environment connection string injection techniques allow an attacker to inject parameters by appending them with the semicolon (;) character.

In an example where the user is asked to enter a username and a password to create a connection string, an attacker can void the encrypting system by entering a password such as *"pwd; Encryption=off"*, resulting in a connection string like:

```
"Data Source=Server,Port; Network Library=DBMSSOCN;
Initial Catalog=DataBase; User ID=Username;
Password=pwd; Encryption=off"
```

When the connection string is populated, the *Encryption* value will be added to the previously configured set of parameters.

2.1 Connection String Builder in .NET

Aware of this exploitation [3] of the connection strings, Microsoft included the "ConnectionStringBuilder" [4] classes on it's version 2.0 of the Framework. They are meant to create secure connection strings through the base class (DbConnectionStringBuilder) or through the specific classes for the different providers (SqlConnectionStringBuilder, OleDbConnectionStringBuilder, etc…), and they achieve this by allowing just key-value pairs for attributes and by escaping injection attempts.

The use of these classes when creating a connection string would prevent the injections. However, not every developer or application uses them.

2.2 Connection String Parameter Pollution

Parameter pollution techniques are used to override values on parameters. They are well known in the HTTP [5] environment but they are also applicable to other environments. In this example, parameter pollution techniques can be applied to parameters in the connection string, allowing several attacks.

2.3 Connection String Parameter Pollution (CSPP) Attacks

As an example scenario to illustrate these attacks, a web application where a user [User_Value] and a password [Password_Value] are required is served by a Microsoft Internet Information Services web server running on a Microsoft Windows Server. The application user credentials are going to be used to create a connection string to a Microsoft SQL Server database as follows:

```
Data source = SQL2005; initial catalog = db1;
integrated security=no; user id=+'User_Value'+;
Password=+'Password_Value'+;
```

This connection string shows how the application is connecting to a Microsoft SQL Server database engine. Knowing this, and attacker can perform a Connection String Parameter Pollution Attack. The idea of this attack is to add a parameter to the connection string with the desired value, regardless of if it already was in the string or the value with which was set up. The component used by .NET applications to craft the connection string will use the value of the last occurrence of the parameter in the connection string. If the connection string has two parameters which key is "Data Source", the value used will be the one of the last of the two pairs, which allows the following CSPP attack vectors:

2.3.1 CSPP Attack 1: Hash stealing

An attacker can place a Rogue Microsoft SQL Server connected to the Internet with a Microsoft SQL Server credential sniffer listening (In this exsmple CAIN [6] has been used). An attacker would perform a CSPP attack as follows:

```
User_Value:   ; Data Source = Rogue_Server
```

```
Password_Value:  ; Integrated Security = true
```

Resulting in the following connecting string:

```
Data source = SQL2005; initial catalog = db1;
integrated security=no; user id=;Data Source=Rogue
Server; Password=; Integrated Security=true;
```

The "Data Source" and "Integrated Security" parameters are being overridden so that the Microsoft SQL Server native drivers will use the last set of values ignoring the previous ones, and the application will try to connect to *Rogue_Server* with the Windows credentials it's running on, which can be either a system user or an application pool user.

2.3.1.1 Example 1: ASP.NET Enterprise Manager

This tool is an abandoned and unsupported Open Source tool, but still being used by some hosting companies and some organizations to manage Microsoft SQL Server databases via a web interface. The official web site, which used to be *aspnetenterprisemanager.com*, is today abandoned, but the tool can be obtained from several other web sites like SourceForge [7] or MyOpenSource [8]. This tool is being recommended in a lot of forums as a good ASP.NET alternative to PHPMyAdmin [9], even though the last version was published on the 3rd of January of 2003.

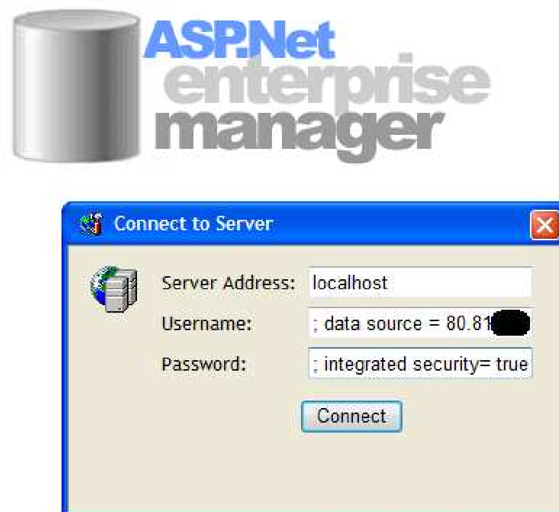

Fig. 3. CSPP in ASP.NET Enterprise Manager to steal the account information

The results are collected on the rogue server where the database connection sniffer has been installed giving access to the LM Hash of the account.

Sniffer | Cracker | Traceroute | CCDU | Wireless | Query

Timestamp	TDS server	Client	Username	Password	A
22/07/2009 - 13:52:53	80.81	217.130.	VE103$		N
22/07/2009 - 13:53:09	80.81.	217.130.	VE103$		N

AuthType	Domain	LM Hash	Domain	LM Has
NTLM Session S...	GRUPO_TRABAJO	5A932C2E11D567440000000000	GRUPO_TRABAJO	5A932
NTLM Session S...	GRUPO_TRABAJO	7447CA85CE589C320000000000	GRUPO_TRABAJO	7447C

Fig. 4. Hash collected in the rogue server with Cain

2.3.2 CSPP Attack 2: Port scanning

One of the valid parameters on a connection string is the port to connect to. An attacker can abuse an application vulnerable to this technique to network scan servers by trying to connect to different ports and see the error messages obtained:

```
User_Value:     ; Data Source =Target_Server,
Target_Port

Password_Value:   ; Integrated Security = true
```

This injection attack will result in the following connection string:

```
Data source = SQL2005; initial catalog = db1;
integrated security=no; user id=;Data Source=Target
Server, Target Port; Password=; Integrated
Security=true;
```

This connection string will ignore the first instance of the first "Data Source" parameter and will use the last one, meaning that the web application is going to try to connect to "Target Port" port on the "Target Server" machine. Observing the differences in the returned error messages, a port scan can be performed.

2.3.2.1 Example 2: myLittleAdmin and myLittleBackup

The tools myLittleAdmin [10] and myLittleBackup [11] are commercial tools developed by myLittleTools [12]. Both tools are vulnerable to CSPP attacks up to versions myLittleAdmin 3.5 and myLittleBackup 1.6.

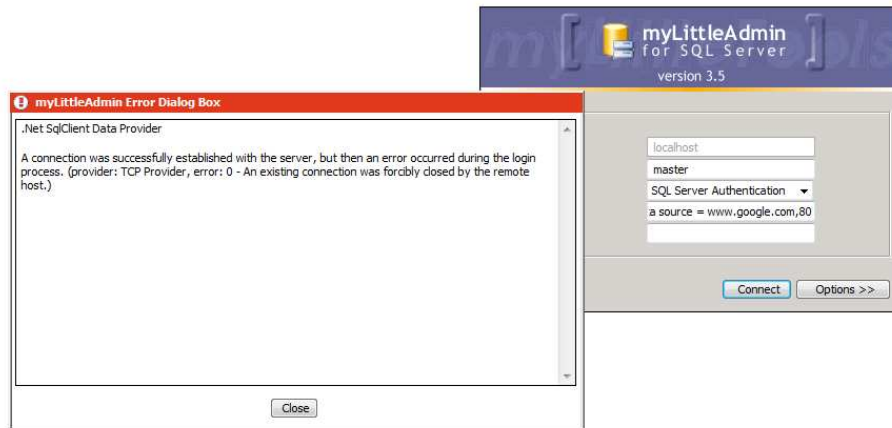

Fig. 5. A connection can be established through port 80 to www.gooogle.com

As shown in Fig. 5, when the port is listening (open) the error message obtained says that no Microsoft SQL Server is listening on it, but a TCP connection was established.

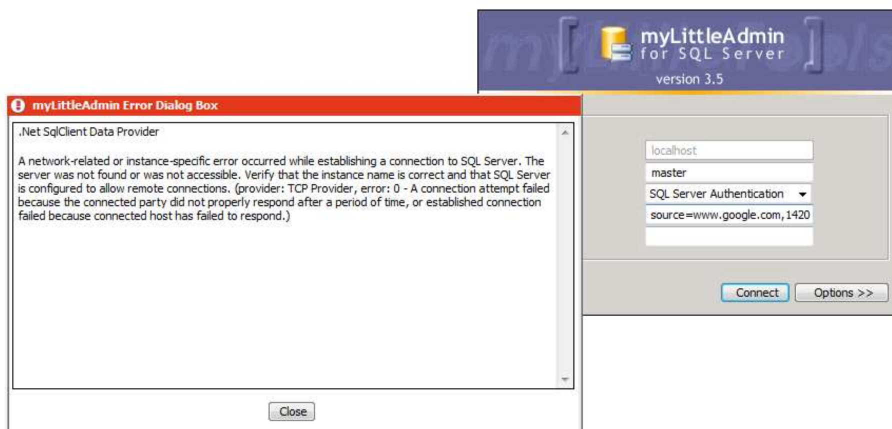

Fig. 6. A connection cannot be established through the XX port to www.google.com

When the port is closed, a TCP connection could not be completed and the error message is different. Using these error messages a complete TCP port scan can be run against a server. Of course, this technique can also be used to discover internal servers within the DMZ where the web application is hosted.

2.3.3 CSPP Attack 3: Hijacking Web credentials

This time the attacker tries to connect to the database by using the web application system account instead of a user provided set of credentials:

```
User_Value:   ; Data Source =Target_Server

Password_Value:  ; Integrated Security = true
```

These injected values will result in the following connection string:

```
Data source = SQL2005; initial catalog = db1;
integrated security=no; user id=;Data Source=Target
Server, Target Port; Password=; Integrated
Security=true;
```

This time is the "integrated security" parameter what is being overwritten with a "True" value. This means that the system will try to connect to the database with the system account which the tool is running with. In this case this is the system account used by the web application in the web server.

2.3.3.1 Example 3: SQL Server Web Data Administrator

This tool is a project, originally developed by Microsoft, which was made free as an Open Project. Today, it is still possible to download the last version that Microsoft released on 2004 from Microsoft Servers [13] but the latest one, released on 2007, is hosted in the Codeplex web site [14]. The version hosted in Codeplex is secure to this type of attacks because it is using the ConnectionStringBuilder classes to dynamically construct the connection string.

The version published on the Microsoft web site is vulnerable to CSPP attacks. The following screenshots show how it is possible to get access to the system using this type of attack.

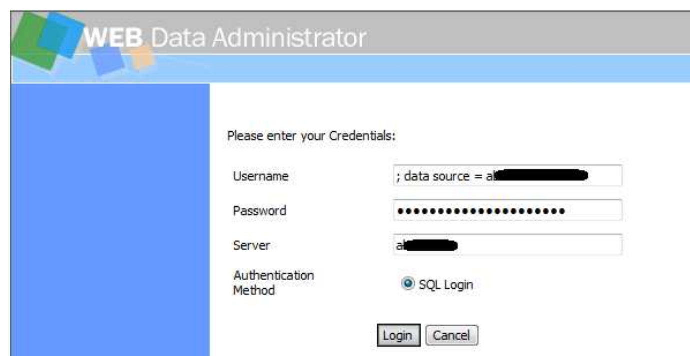

Fig. 7. Exploiting the credentials at the WEB Data Administrator

In Fig. 7, the password value is: *"; integrated Security=true"*, as described previously.

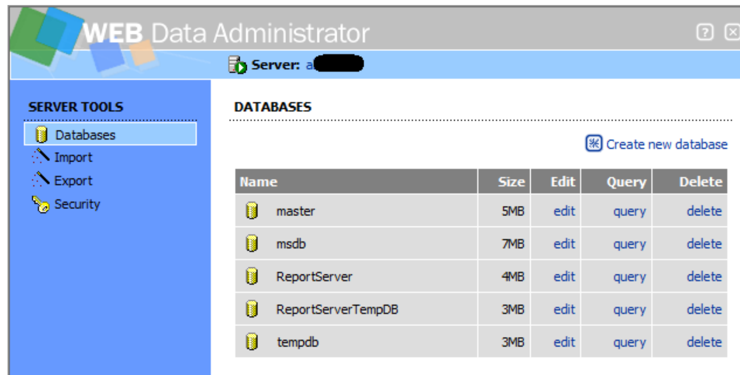

Fig. 8. Console access with the server account

The attacker can log into the web application to manage the whole system. As shown in Fig. 9, this is because all users and network services have access to the server.

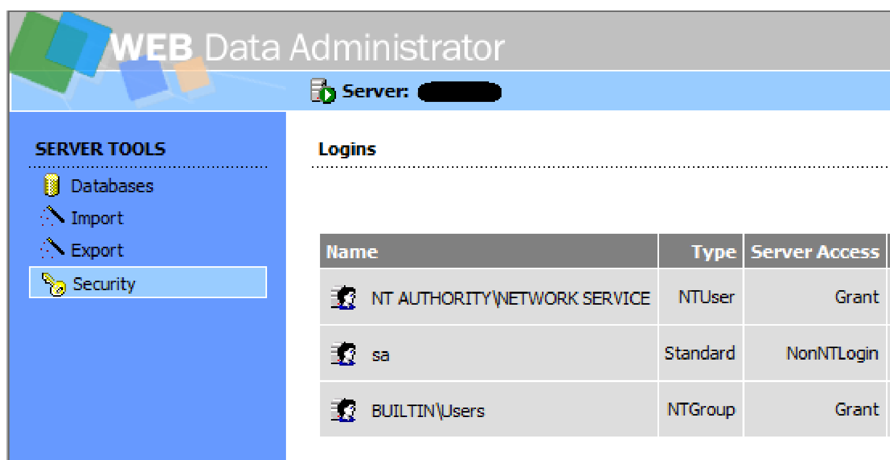

Fig. 9. System account access grant.

2.3.3.2 Example 4: myLittleAdmin and myLittleBackup

In mylittleAdmin and myLittlebackup tools, it is possible to check out the connection string used to get the access. Looking at it, the parameter pollution injected in order to obtain access to the system can be clearly seen.

Fig. 10. CSPP in myLittleAdmin.

Fig. 10 shows how the "Data Source" parameter, after the "User ID" parameter, has been injected with the *localhost* value. This parameter, "Data Source", is also the first one of the connection string. Their values are different but the one that is being used is the last one, i.e. the injected one.

Same happens with the "Integrated Security" parameter that appears initially with the *no* value but then is overriden by the one injected in the password value with value *yes*, resulting in a total access to the server with the system account the web application is running as.

Fig. 11. Querying the master..sysusers table.

2.3.3.3 Example 5: ASP.NET Enterprise Manager

The same attack also works on the latest public version of the ASP.NET Enterprise manager, so, as can be seen in the following login form, an attacker can perform the CSPP injection to get access to the web application.

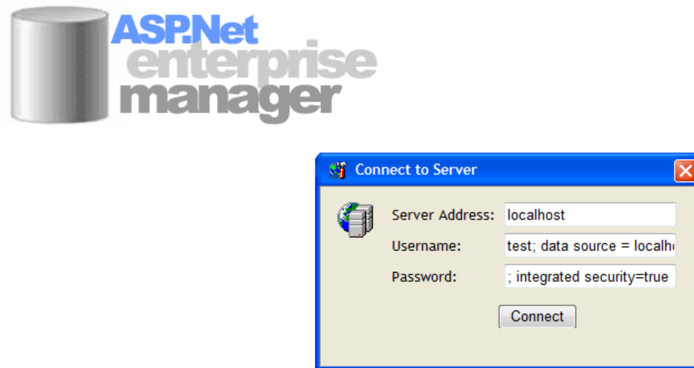

Fig. 12. CSPP in ASP.NET Enterprise Manager login form.

And as a result of it, full access can be obtained, just as can be seen in the following screenshot.

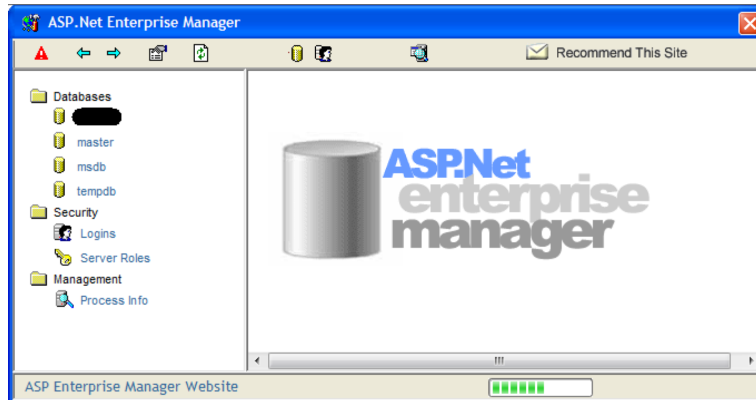

Fig. 13. Administration console in ASP.NET Enterprise Manager.

3 Conclusions

All these examples show the importance of filtering any user input in web applications. Moreover, these examples are a clear proof of the importance of maintaining the software updated. Microsoft released ConnectionStringbuilder in

order to avoid these kinds of attacks, but not all projects were updated to use these new and secure components.

These techniques also apply to other database engines such as Oracle databases, which allow administrators to set up Integrated security into the database. Besides, in Oracle connection strings it is possible to change the way a user gets connected by forcing the use of a *sysdba* session.

MySQL databases do not allow administrators to configure an Integrated Security authentication process. However, it is still possible to inject code and manipulate connection strings to try to connect against internal servers not exposed to the Internet.

In order to avoid these attacks the semicolon character must be filtered out, all the parameters sanitized, and the firewall be hardened in order to filter not only inbound connections but also prevent outbound connections from internal servers that are sending NTLM credentials to the internet. Databases administrator should also apply a hardening process in the database engine to restrict access by a minimum privilege policy.

References

1. The Open Web Application Security Project, http://www.owasp.org
2. Connection Strings.com: http://www.connectionstrings.com
3. Ryan, W.: Using the SqlConnectionStringBuilder to guard against Connection String Injection Attacks, http://msmvps.com/blogs/williamryan/archive/2006/01/15/81115.aspx
4. Connection String Builder (ADO.NET), http://msdn.microsoft.com/en-us/library/ms254947.aspx
5. Carettoni L., di Paola S.: HTTP Parameter Pollution, http://www.owasp.org/images/b/ba/AppsecEU09_CarettoniDiPaola_v0.8.pdf
6. Cain: http://www.oxid.it/cain.html
7. ASP.NET Enterprise Manager in SourceForge, http://sourceforge.net/projects/asp-ent-man/
8. ASP.NET Enterprise Manager in MyOpenSource: http://www.myopensource.org/internet/asp.net+enterprise+manager/download-review
9. PHPMyAdmin: http://www.phpmyadmin.net/
10. myLittleAdmin: http://www.mylittleadmin.com
11. myLittleBackup: http://www.mylittlebackup.com
12. myLittleTools: http://www.mylittletools.net
13. Microsoft SQL Server Web Data Administrator: http://www.microsoft.com/downloads/details.aspx?FamilyID=c039a798-c57a-419e-acbc-2a332cb7f959&displaylang=en
14. Microsoft SQL Server Web Data Administrator in Codeplex project: http://www.codeplex.com/SqlWebAdmin