THREAT MODELLING FOR SQL SERVERS

Designing a Secure Database in a Web Application

E.Bertino [1], D.Bruschi [2], S.Franzoni [2], I.Nai-Fovino [2], S.Valtolina [2]

[1]*CERIAS, Purdue University, West Lafayette, IN, USA*
[2]*DICO, Universita' degli Studi di Milano, Via Comelico 39, 20135 Milan, Italy*

Abstract: In this paper we present the results from an analysis focusing on security threats that can arise against an SQL server when included in Web application environments. The approach used is based on the STRIDE classification methodology. The results presented provide also some general guidelines and countermeasures against the different attacks that can exploit the identified vulnerabilities.

Key words: Database Systems, Web Services, Security, Threat Model

1. INTRODUCTION

In the last few years the use of the Internet has experienced an exponential growth and the World Wide Web has become the main instrument for information sharing. Such trends have pushed the development of a new kind of service architecture, specifically tailored at supporting data sharing among remotely connected clients, which is based on the concept of *Web Applications.* A web application can be essentially seen as a collection of different entities that collaborate in order to provide services to remote clients. In such an architecture, a client typically communicates with external entities using the HTTP protocol.

Various web application architectures have been devised and can be found in the literature. In this paper we will consider one of the most adopted, i.e. the architecture includes a database, positioned into a backend zone, storing all the information needed to provide the service. Since this data must be accessible from both the internal network and Internet, the database is exposed on the web and can thus become a target of possible attacks [1]. Many of these attacks can be prevented following some guidelines in the design and development of the web applications. In this paper, by using the STRIDE approach [2], we analyze the most frequent threats concerning the database in a generic model of web applications, and we describe the countermeasures to prevent those threats or mitigate the damages subsequent to a successful attack.

2. SQL SERVER

SQL Server [3] is a relational database management system which is part of the Microsoft family of servers. SQL Server was designed for client/server use and is accessed by applications using SQL. It runs on Windows NT version 4.0 or higher and is compliant with the ANSI SQL-92 standard. SQL Server supports symmetric multiprocessing hardware, ODBC, OLE DB, and major open standard communications protocols. It has Internet integration, data replication, transaction management and data warehousing features.

The main role of an SQL Server in a web-based architecture is to store and manage the data required by the authorized web applications. To be able to access data from a database, a user must pass through two authentication phases. The first phase is performed by the SQL Server and the other by the database management system. These two steps are carried out using logins names and user accounts respectively.

2.1 Authentication

A valid login name is required to connect to an SQL Server instance. A login could be a Windows NT/2000 login that has been granted access to SQL Server or an SQL Server login that is maintained within the SQL Server. These login names are stored in the master database. All permissions and ownership of objects in the database are controlled by the user account. SQL Server logins are associated with these user accounts. During a new connection request, SQL Server verifies the login name supplied to make sure that the login corresponds to a subject authorized to access SQL Server. SQL Server supports two authentication modes:

- Windows authentication mode: under this mode there is no need to specify a login name and password to connect to SQL Server. Instead the access to SQL Server is controlled by the user's Windows NT/2000 account (or the group to which the user's account belongs). Database administrators can grant access to the database to the user or the user group specified in the Access Control List provided by the operating system. Under this security mode, SQL Server tracks users by their individual SIDs (Security Identifiers) stored by the operating system itself.

- Mixed mode: users can establish a connection to an SQL server either using Windows authentication or SQL Server authentication. Under this authentication mode, the user must supply the SQL Server login and password when he connects to SQL Server. If the user does not specify an SQL Server login name and password, or request Windows Authentication, he/she is authenticated using Windows Authentication.

2.2 Access Control

Accesses to objects in the database are managed by granting the proper permissions to individual users or by defining user roles. A role is a group to which individual logins/users can be added, so that the permissions can be assigned to the group, instead of assigning them to all individual logins/users. There are three types of roles in SQL Server:

- Fixed server roles: these are server-wide roles. Logins can be added to these roles to gain the associated administrative permissions of the role. Fixed server roles cannot be altered and new server roles cannot be created. An example of a fixed server role is sysadmin, which is authorized to perform any activity in SQL Server.
- Fixed database roles: each database has a set of fixed database roles to which database users can be added. These fixed database roles are unique within the database. While the permissions of fixed database roles cannot be altered, new database roles can be created. An example of a fixed database role is db_owner, which has all permissions in the database.
- Application roles: after creating and assigning the required permissions to an application role, the client application needs to activate this role at run-time to get the permissions associated with that application role. By using application roles, the database administrator does not have to manage permissions at the individual user level; he/she simply needs to create an application role and assign permissions to it. The application that is connecting to the database activates the application role and inherits the permissions associated with that role.

2.3 System Prerequisites

The guidelines discussed in this paper are effective only if the SQL Server is properly installed, configured and patched. In this section we provide a list of actions that we assume have been already taken on the database.

2.3.1 Installation recommendations

We assume that the SQL Server has been installed with a least privilege account. In order to protect the domain hosting the database, the SQL server must not be installed on a primary or secondary domain controller. Instead, we recommend dedicating a machine to the database, without additional services (i.e. Upgrade tools, Replication support Script, Development tools Headers and library files used by C developers and Microsoft Data Access (MDAC), etc.) if they are not required. We assume that after the installation all available patches are applied.

2.3.2 Unused Services

During the installation phase three major services are set up, the SQLSERVERAGENT, the MSSQLServerADHelper and the Microsoft Search. These three services are optional and they must be disabled if they are not necessary. It is also important to notice that the presence of other services not related with SQL Server on the same machine can jeopardize the database security. The installation of such services is discouraged if they are not strictly required.

2.3.3 Unused Protocols

It is a good practice to configure SQL Server to support only clients that connect using the TCP/IP protocol. All the other unused protocols must be disabled. A TCP/IP stack hardening can be also taken into consideration.

2.3.4 Accounts

We assume that for all the accounts configured after the installation, the principle of least privilege has been adopted. The execution of the following actions should be considered:

- Secure the SQL Server service account
- Delete or disable unused accounts
- Disable the Windows guest account
- Rename the administrator account
- Enforce strong password policy (length and complexity, expiration time)
- Restrict remote logins
- Disable null sessions (anonymous logons)

2.3.5 File system and directory

An important prerequisite for a proper SQL Server installation is a strongly secure directory and file-system permission management. We take for granted the execution of the follow steps:

- Verification of permissions on SQL Server install directories.
- Verification that the Everyone group does not have permissions to SQL Server files.
- Secure setting up of log files.
- Securing or removing tools, utilities, and SDKs.

3. ASSETS

The first step that must be taken into account in a threat modelling process is the identification of the assets that need to be protected. In fact, they represent the value the attacker is looking for. In our particular case, the principal asset that we want to protect is the data stored in the database. It must be pointed out that not all the data stored have the same relevance. For example, a company can decide to publish on the Web only a partial database, while the whole enterprise data are kept offline. In such a case, the loss of the data accessible from the Web is much less serious than damages to the data stored in the backend database, as we discuss in Section 4. Data protection involves satisfying two main requirements: the integrity of stored data and their confidentiality. These two properties have both been taken into account in our threat modelling phase, as explained in Section 6.

In addition to data, the second fundamental asset that needs to be protected is the data management service, for which availability is crucial; a database should always be able to provide the data required to authorized users.

The data and the data management system can thus be considered the "crucial" assets of an SQL server, but there are some other assets not strictly connected with data stored in the database that can be valuable for an attacker. Some examples are the data accounts of authorized users, the database system, since it can be exploited for more sophisticated attacks, and the integrity of the host machines. Weaknesses in a host machine can be exploited to perform attacks against other machines in the enterprise network.

4. ARCHITECTURAL SCENARIOS

We consider two possible scenarios. In the first one, as described in the architecture overview, the SQL Server is located in the backend subnet. This is the most straightforward method for providing web access to the database. The web application server forwards the client requests to the database across the internal network.

The second scenario is characterized by a general architecture which is similar to the architecture of the first scenario. The main difference is that the backend database is not accessible from the web application. The client can only access a partial mirror database located in another subnet. The mirror database contains only the data to be accessed from the Internet. In this way, an attack exploiting the web application or the web server would compromise only the data stored in the database located in this partial database and not all the enterprise data. When adopting this configuration, this database must be synchronized with the backend SQL server, and these update operations need to be performed in a secure way.

When considering these scenarios, we must take into account that there is a non-negligible risk: we protect the database from an outside attacker, but we cannot say anything about an attacker that has already the control of other servers inside the backend subnet and that can use these servers in order to start an attack to the SQL Server. This issue can be addressed assuming that the backend subnet is trusted, or that the database is located in a dedicated subnet.

The threats concerning the two scenarios are in general the same; what changes is the difficulty with which these threats can be realized and the dimension of the final damage. For example, if an attacker can disrupt the data stored in the SQL server in the first scenario, the enterprise will lose all its data, with an enormous damage. If the same threat succeeds in the second scenario, the company will lose only the data published on the public mirror database, but not the sensitive data stored in the main SQL database.

5. ATTACK ENTRY POINTS

One of the most important issues in threat modelling activity is to identify the attack entry points of the system being analyzed. Based on the vulnerabilities of a SQL Server and the above architectural scenarios, we have identified three attack points:

- The client side of the web application: an attacker can use the normal web interface of a client in order to insert some malicious code or perform unauthorized or dangerous operations. It is very difficult to control this entry point. The main problem is that it is not possible to make any assumption about the client identity. Moreover we generally do not have control over the configuration and security of the client machine. Thus the client machine can be trojanized and controlled by a third malicious party. For all these reasons we assume that the client is not trusted. A general good practice is thus to perform strong input validation, to inhibit dynamic SQL, and to use an effective password management policy.

- Network: an attacker with a direct access to the network can intercept information or data flow between the client and web application or between the web application and SQL Server. It is important to note that this entry point is not only located on the external network, but it also involves the internal enterprise network. The attacker can also mount some complex attack like a man-in-the-middle attack. In order to protect this entry point a good practice is the use of a secure channel (IPsec, SSL etc.) between the different actors involved in the web application (client, web server, SQL server etc.)
- SQL Server port: an attacker can try to directly send requests and malicious code (Slammer worm for example acted in this way) to the SQL Server bypassing the web application. A strong access control policy and ad-hoc firewall rules can mitigate the vulnerability of this entry point.

6. SQL SERVER THREATS

As explained above, in order to identify the different threats on SQL Server, we have adopted the STRIDE classification that groups the different types of attack into six main classes (Spoofing identity, Tampering with data, Repudiation, Information disclosure, Denial of service and Elevation of privilege). Table 1 provides a mapping between the STRIDE classes and the common attacks we have identified.

In the remainder of this section, we describe in more detail the different types of attack and their respective countermeasures.

Table 1 Mapping between the STRIDE classes and the common attacks

	S	T	R	I	D	E
SQL injection		X		X		X
Unauthorized access	X	X		X		X
Network eavesdropping	X			X		X
Denial of service					X	
Timing analysis				X		
Error analysis				X		
Malicious Data Mining				X		

6.1 SQL Injection

This is a technique which exploits vulnerabilities in input validation to run arbitrary commands in the database [1]. It can occur when the application uses input to construct dynamic SQL statements to access the database. It can also occur if the code uses stored procedures that are passed strings containing unfiltered user input. The issue is magnified if the application uses an over-privileged account to connect to the database. In this instance it is also possible to use the database server to run operating system commands and potentially compromise other servers, in addition to being able to retrieve, manipulate, and destroy data.

Usually this attack affects applications that incorporate non-validated user input into database queries. Particularly susceptible is code that constructs dynamic SQL statements with unfiltered user input.

6.1.1 Countermeasures

In order to prevent this kind of attack the application should validate its input prior to sending a request to the database. Other preventive countermeasures are the use of type safe SQL parameters for data access, the execution of checks against parameter types and length, the use of injected code as literal data instead of executable code for the database. The use of restricted accounts can be useful too.

To discover if this type of attack has been performed, the only possible countermeasure that can be adopted is logging all requests sent to the database and then executing an off-line analysis. Because the logging and analysis activities can be very expensive, when designing the database and the database application it is important to identify which kind of data flow must be logged and analyzed in order to avoid overloading the logging system.

6.2 Network Eavesdropping

This threat is related to the unauthorized interception of information sent across a network [1]. This attack is usually carried out by means of a packet sniffer program which can monitor the traffic on the network. An attacker may exploit poorly configured network devices. Common vulnerabilities include weak default installation settings of the communication channels between client and web server and between web server and database server.

6.2.1 Countermeasure

There are different types of countermeasure that can be taken in order to protect against this type of threat. For example the use of Windows authentication to avoid sending credentials over the network can be useful to protect the system from the discovery of authorized user accounts. In this respect, the installation of a server certificate on the database server and the use of an encrypted channel like SSL or IPsec is a good practice to protect the integrity and confidentiality of the data exchanged on the network.

This type of attack assumes that a malicious user is able to capture the traffic between the different actors of the web application. Usually, as explained above such type of attacks is made possible by Sniffer tools. To detect the use of these tools on the enterprise network, the use of some "discovery sniffer" tools is suggested [4,5,6].

6.3 Error Analysis

This threat arises because of a general good practice concerning well designed code [1]. Indeed in a well designed software system, when an error occurs, a detailed error message is returned as feedback in order to understand where the problem is. Exception conditions can arise because of configuration errors, bugs in the code, or malicious input. Unfortunately, a malicious client can use these error messages in order to guess sensitive information about the location and nature of the data source in addition to valuable connection details.

Targets of this type of attack can be applications that incorporate non-validated user input into database queries and that do not use exception handling or implement it poorly. Particularly susceptible is code that constructs dynamic SQL statements with unfiltered user input or code that does not check input parameters.

6.3.1 Countermeasures

In order to avoid a malicious use of message errors, an approach is to filter them. Effective filtering can be achieved by ensuring that the database accepts connections only from the application (the application thus acts as a filter), by using exception handling throughout the application's code base and by returning a generic, harmless error messages to the client.

Log and traffic analysis is the only way for detecting this threat.

6.4 Denial of Service

Denial of service denies [1] legitimate users access to a server or services. It can focus on two targets: the former type of DoS arises because of the structure of the Internet. Therefore there are no effective countermeasures at the database server level. The latter type of DoS is performed to make the server unable to provide its services, by means like application crashing or resource consumption. Stored procedures executing non-optimized code, stored procedures executing code with weak controls over variable size and type, or stored procedures executing code with a bad resource allocation and management policy are examples of the possible target of this attack class.

6.4.1 Countermeasures

To avoid the execution of this type of attack the following operations should be considered. Strong input validation should be performed on the client side with the aim of avoiding the insertion of malicious or unusual requests.

Because most of the security holes with respect to the second class of DoS attacks are usually located in the stored procedures, a good practice is to allow the execution of only secure stored procedures. By secure stored procedures we mean stored procedure whose implementation is well known. Other suggested countermeasures include the adoption of a resource allocation policy combined with a service monitor tool and the profiling of the stored procedures also under stress conditions [7].

6.5 Unauthorized accesses

This threat is common to all systems in which there is no strong password management policy. Attackers typically try to guess the passwords of authorized users. This type of threat can also arise in all systems that are affected by buffer overflow vulnerability [8,9]. In this case once the attacker has obtained a remote root shell, he/she has gained access to the database.

6.5.1 Countermeasures

The best strategies for protecting against unauthorized accesses are the use of Windows authentication, the adoption of a strong password policy and a strong input validation to avoid the insertion of buffer overflow code. The only way to detect this type of threat is a log analysis.

6.6 Sensitive Information Disclosure

This threat can be posed even by users having limited access rights to the database. By manipulating the results of regular queries by means of data mining techniques an attacker can extract sensitive information.

6.6.1 Countermeasures

The only countermeasure to this threat can be database sanitization [10] or the publication of a partial database containing only non-sensitive information.

7. DESIGN GUIDELINES

We now summarize the major design guidelines that we have devised.

A first important protection measure is to adopt an architecture for protecting the SQL Server from exploit attempts and at the same time protecting the enterprise subnets from attacks performed using an eventually exploited SQL Server. A logical separation between the network hosting the SQL Server and the other networks is strongly recommended and also the presence of a mirror SQL Server is a good practice in order to guarantee the survivability of the system.

A second protection measure is the use of the Windows authentication whenever possible. With Windows authentication, it is possible to make use of the system account and password management policies, so there is no need to store database connection strings with embedded credentials and to have to transmit these credentials across the network. Windows and SQL Server must both recognize the account from which the application runs. The account must be granted a login to SQL Server and the login needs to have associated permissions to access a database.

If SQL authentication must be adopted, it is necessary to take additional precautions in order to secure the database connection string, since it contains the user login and password. The connection string must not be sent over the network in clear text, but it must be encrypted.

The application login to the database must be properly authorized and restricted. The application should use a least privileged account that has limited permissions in the database; this can limit the potential damage if the account is compromised or malicious code is injected.

A third relevant protection measure is to never connect to an SQL Server using the **sa** account or any account that is a member of the SQL Server **sysadmin** or **db_owner** roles. If the connection is established using an over-privileged account, for example an account provided with the SQL Server **sysadmin** role, the attacker can perform any operation in any database on the server.

Other important protection measures are based on the use of parameterized stored procedures that should be used for data access where possible. Stored procedures can enhance data access security in several ways. Database users can be given permissions to execute a stored procedure without being granted permissions to directly access the database objects on which the stored procedure operates. Besides, stored procedures can validate user input, and their parameters cannot be treated as executable code. All this helps mitigate the risk posed by SQL injection attacks. However, only stored procedures whose origin and behavior is well known should be used.

If parameterized stored procedures cannot be used for some reason and the application needs to construct SQL statements dynamically, it is crucial to use typed parameters and parameter placeholders to ensure that input data are checked with respect to their length and type.

Another protection technique is related to error management by the application software. It is important to trap all the exceptions the application may raise and return only generic error messages which do not reveal details about the inner database structure.

Finally sensitive data should be encrypted when stored in the database and protected, when being transmitted across the network, by using an SSL connection between the Web server and database server and/or an IPSec encrypted channel.

8. ACKNOWLEDGEMENTS

This work reported in this paper was developed as part of the *Designing Secure Applications* (DeSecA) project, funded by Microsoft. Partners within this project are the Universita' degli Studi di Milano, the Technical

University of Ilmenau, the University of Salford, and the COSIC and DistriNet research groups of the Katholieke Universiteit Leuven.

9. REFERENCES

[1] J.D. Meier and others, Improving Web Application Security. Threats and countermeasures, Microsoft press, 2003

[2] M. Howard and D. LeBlanc, Writing secure code 2nd edition , Microsoft press, 2003

[3] SQL Server 2000, Resource kit, Microsoft press, 2001

[4] http://sniffdet.sourceforge.net/download.html

[5] http://www.atstake.com/antisniff/

[6] http://www.packetfactory.net/Projects/sentinel/

[7] E. Whalen, M. Garcia and others, SQL Server 2000 Performance tuning, Microsoft press, 2001

[8] A. One, Smashing The Stack For Fun And Profit, .oO Phrack 49 Oo. Volume Seven, Issue Forty-Nine File 14

[9]] M. Howard, Reviewing Code for Integer Manipulation Vulnerabilities, Secure Windows Initiative, 2003

[10] V. S. Verykios, E. Bertino, I. Nai Fovino, L. Parasiliti Provenza, Y. Saygin, Y. Theodoridis, State-of-the-art in privacy preserving data mining, ACM SIGMOD Record, Volume 33 Issue 1, 2004