Use trusted context in DB2 client applications

Greater security in multi-tier environments for DB2 CLI, XA, and JDBC applications

Samantha Tran
Manoj Mohan

September 28, 2006

Learn how to implement trusted context on DB2® in a call level interface (CLI), XA, or Java™ Database Connectivity (JDBC)-based application. Using a trusted context in an application can improve security by placing accountability at the middle-tier, reducing over granting of privileges, and auditing of end-user’s activities.

Introduction

In a three-tiered application model, where the middle tier (for example, WebSphere Application Server or Domino) is responsible for authenticating the users running the client applications and managing the interactions with the database server. The middle tier’s authorization ID needs to have all the privileges associated with end-users in order to perform any operations that the end users may require. While the three-tiered application model has many benefits, having all interactions with the database server (such as a user request) occur under the middle tier’s authorization ID raises several security concerns like the following (Also refer to Figure 1.):

- **Loss of user identity:** Some enterprises prefer to know the identity of all users accessing the database for access control purposes.
- **Diminished user accountability:** Accountability through auditing is a basic principle in database security. The database should be able to distinguish the transactions performed by the middle tier for its own purpose, from those performed by the middle tier on behalf of some users.
- **Over-granting of privileges:** The middle tier’s authorization ID must have all the privileges needed to execute all the requests from all the users. This has the security issue of enabling users who do not need access to certain information to obtain access to them.
- **Weakened security:** In addition to the over-granting privilege issue, the current approach requires that the authorization ID used by the middle tier to connect must be granted privileges on all resources that might be accessed by user requests. If that middle-tier authorization ID is ever compromised, then all those resources are exposed.
Clearly, there is a need for a mechanism whereby the actual user's identity and database privileges are used for database requests performed by the middle tier on behalf of that user. The most straightforward approach of achieving this goal is for the middle-tier to establish a new connection using the user ID and password, and then direct the user's requests through that connection. Although simple, this approach suffers from several drawbacks. Many middle-tier servers do not have the user's authentication credentials needed to establish a connection. There's the obvious performance overhead associated with creating a new physical connection for each user at the database server. Therefore, creating an overhead for maintenance of user's credentials at both the server and middle tier.

A better approach is needed for ensuring that each user's database identity and database privileges are used for any database requests performed by a middle tier on behalf of that user. In order to maximize performance, the approach is to allow for the middle tier to reuse the same physical connection without the need to re-authenticate the user at the database server. This introduces the idea of a trusted connection.

**Use a trusted connection**

In order to establish a trusted connection, a new object called a trusted context must be created on DB2 in order to establish a trusted relationship between DB2 and an external entity, such as a middleware server. The definition of a trusted context includes criteria that must be met for a specific connection to use the trusted context and be considered a trusted connection.
When there is an attempt to establish a trusted connection, a series of trust attributes are evaluated to determine if a specific context is to be trusted. The relationship between a connection and a trusted context is established when the connection to the server is first created and remains for the duration of that connection. When a trusted connection is established, it is possible that the connection can be used for a different authorization ID by allowing the middle tier to specify a new user ID, possibly without the need to authenticate the user ID. (Refer to Figure 2.)

**Figure 2. Three-tiered application model with trusted context**

![Three-tiered application model with trusted context](image)

**Defining a trusted context**

A trusted context is a new object that is defined based upon a system authorization ID, and one or more sets of connection trust attributes. Each trusted context is identified based on an associated system authorization ID and one or more sets of connection trust attributes, where each set defines at least one connection trust attribute:

- **System authorization ID:** The primary trust attribute is the authorization ID used for the connection. There is always a clear mapping between any given system authorization ID used to establish a connection and a specific trusted context.
- **Connection trust attributes:** A set of connection trust attributes identify a set of characteristics that must be met by a connection for that connection to be considered a trusted connection based on that trusted context. All of the conditions that are defined for a set of attributes for the trusted context must be satisfied for a connection to be considered a trusted connection using that set of attributes for the trusted context.
• **PROTOCOL:** The communication protocol trust attribute. This is used to control which network communication protocols can use the trusted context.

• **ADDRESS:** The network address trust attribute. This is used in conjunction with the PROTOCOL attribute to control which addresses the trusted context can be used with. This is the actual client's IP address or domain name, used by the connection to communicate with the database manager.

• **ENCRYPTION:** The network encryption trust attribute. This specifies the minimum level of encryption of the data stream (“network encryption”) for the connection.

• **AUTHENTICATION:** The authentication trust attribute. The attribute specifies the level of authentication required to be performed on the system authorization ID during the establishment of the connection.

Assume that an administrator wants any connection to be considered trusted when the system authorization ID is NEWTON and the TCP/IP address attribute is 9.26.146.201. The administrator could define such trusted context as follows:

**Example 1. Sample trusted context definition**

```sql
CREATE TRUSTED CONTEXT ctxName1
BASED UPON CONNECTION USING SYSTEM AUTHID newton
ATTRIBUTES ( PROTOCOL 'TCPIP',
               ADDRESS '9.26.146.201',
               ENCRYPTION 'NONE' )
ENABLE
ALLOW USER zurbie
```

If a connection is established using TCP/IP from IP address 9.26.146.201 with authorization ID NEWTON, there is a match between this connection’s attributes and the trusted context ctxName1 defined above, and encryption is ignored.

The administrator can also alter and drop the trusted context object by using `ALTER TRUSTED CONTEXT` and `DROP TRUSTED CONTEXT` statements.

**Trusted connection in CLI application**

A trusted connection can be established and switched for another user by using the following:

- SQLConnect API for CLI applications
- SQLSetConnectAttr and SQLGetConnectAttr API for CLI applications

The following new connection attributes will be introduced for the SQLSetConnectAttr API in CLI applications:

- **SQL_ATTR_USE_TRUSTED_CONTEXT:** A value indicating whether the client is requesting a trusted connection. This value can only be specified before the connection is established or after a disconnection.

- **SQL_ATTR_TRUSTED_CONTEXT_USERID:** A string indicating the user ID to be used on the current trusted connection.
• **SQL_ATTR_TRUSTED_CONTEXT_PASSWORD**: A string indicating the password that may be set by application for authentication purposes. This attribute will have no effect unless the attribute, SQL_ATTR_TRUSTED_CONTEXT_USERID is also set.

The example below shows how to establish a trusted connection for user ID "newton" to testdb database in a CLI application. Before establishing a trusted connection, the application must set the SQL_ATTR_USE_TRUSTED_CONTEXT attribute using the SQLSetConnectAttr API. Once the attribute has been set, a trusted connection can be established by using the SQLConnect API. After the trusted connection has been established, the application can then switch the user to an allowed user defined in the trusted context. In this example, the application switches the connection for user ID "zurbie" by setting the attribute SQL_ATTR_TRUSTED_CONTEXT_USERID.

**Example 2. Using trusted connection in CLI program**

```c
int main(int argc, char *argv[])
{
    int rc = 0;
    SQLHANDLE henv; /* environment handle */
    SQLHANDLE hdbc; /* connection handle */

    printf("\nTHIS SAMPLE SHOWS\n");
    printf("HOW TO CONNECT TO AND DISCONNECT FROM A DATABASE.\n");

    /* allocate an environment handle */
    SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);

    /* allocate a database connection handle */
    SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc1);

    /* set attribute to enable a trusted connection */
    SQLSetConnectAttr(hdbc1,
        SQL_ATTR_USE_TRUSTEDCONTEXT,
        SQL_TRUE,
        SQL_IS_INTEGER);

    /* Establish a trusted connect to a testdb with SQLConnect() */
    /* as user newton */
    SQLConnect( hdbc1, "testdb", SQL_NTS, "newton", SQL_NTS, "xxxxx", SQL_NTS );

    // Perform some work like creating objects, inserting data etc.
    // All the work is performed as user newton

    /* Switch the user from newton to zurbie on a trusted connection */
    SQLSetConnectAttr( hdbc1,
        SQL_ATTR_TRUSTEDCONTEXTUSERID,
        "zurbie",
        SQL_IS_POINTER );
    SQLSetConnectAttr( hdbc1,
        SQL_ATTR_TRUSTEDCONTEXTPASSWORD,
        "xxxxx",
        SQL_NTS );

    /* Perform new work using userid zurbie */

    /* Disconnect from testdb */
    SQLDisconnect( hdbc1 );
    SQLFreeHandle( SQL_HANDLE_DBC, hdbc1 );
}
```
Trusted connection in an XA application

The application program will be able to utilize the trusted connection in an distributed transactions. The XA distributed transaction process starts up one application server per process. In each application-server process, the connection can be established using the XA API (xa_open). This section describes the environment configurations and considerations for running DB2 CLI applications under this environment with trusted context.

A trusted connection can be established and switched for another user by:

- Setting the TCTX parameter in the xa_open string to either true or false to indicate to that client whether the client is running in trusted context mode or not.
- Then, the application must call SQLConnect() to associate the Transaction Manager(TM)-opened connections with the CLI/Open Database Connectivity (ODBC) connection handle. The application can specify the user ID and password in the SQLConnect string.
- Subsequently, the application may call xa_start to pass a transaction ID (XID) to a Resource Manager (RM) to associate the calling thread with a transaction branch.
- In order to switch a user on a trusted connection, the application must first call xa_end (TM_SUCCESS) and specify the new user ID and optional password by calling SQLSetConnectAttr.

This example shows how to enable a trusted context and switch the user ID in an XA application environment. The application must call xa_open with TCTX=TRUE, in order to establish a trusted connection with the database server. After a trusted connection has been established using the SQLConnect string, the application can switch the user ID by calling SQLSetConnectAttr and setting SQL_ATTR_TRUSTED_CONTEXT_USERID to newton. Once the application calls xa_start to begin the transaction, the work is done under the trusted user ID newton. Once the xa_close has been performed, there is no underlying trusted connection. If the CLI handle still exists, it is no longer marked as a trusted connection since the trusted context setting was performed by the XA code during xa_open, when the connection was created. The xa_close disconnected the trusted context and so any new connections established with CLI that are not using XA are not trusted.

Example 3. Using trusted connection in XA application

```sql
#-- Allocate the environment handle
sqlallocenv 1

#-- Set the Trusted Context bit, System Authid and Password
xaopen 10 "DB=st1ec1, sreg=t, SPM=domino, TCTX=TRUE, uid=zurbie, PWD=xxxxxxxx" TMNOFLAGS

#-- Allocate the connection handle
```
Trusted connection in a JDBC application

The IBM DB2 Driver for JDBC and SQLJ provides methods that allow you to establish and use trusted connections in Java programs. To avoid vulnerability to security breaches, an application server that uses these trusted methods should not use un-trusted connection methods.

The DB2ConnectionPoolDataSource class provides several versions of the getDB2TrustedPooledConnection method, and the DB2XADataSource class provides several versions of the getDB2XAConnection method, which allow an application server to establish the initial trusted connection. You choose a method based on the types of connection properties that you pass and whether you use Kerberos security. When an application server calls one of these methods, the IBM DB2 Driver for JDBC and SQLJ returns an Object[] array with two elements:

- The first element contains a connection instance for the initial connection.
- The second element contains a unique cookie for the connection instance. The cookie is generated by the JDBC driver and is used for authentication during subsequent connection reuse.
The DB2PooledConnection class provides several versions of the getDB2Connection method, and the DB2Connection class provides several versions of the reuseDB2Connection method, which allow an application server to reuse an existing trusted connection on behalf of a new user. The application server uses the method to pass the following items to the new user:

- The cookie from the initial connection.
- New connection properties for the reused connection.

The JDBC driver checks that the supplied cookie matches the cookie of the underlying trusted physical connection, to ensure that the connection request originates from the application server that established the trusted physical connection. If the cookies match, the connection becomes available for immediate use by this new user, with the new properties.

Example 4. Using trusted connection in JDBC application

```java
/#------------------#------------------#------------------#------------------#
#-- JDBC example
#-- Test a Trusted Context on the connection
#------------------#------------------#------------------#------------------#

/* The first item that was obtained from the previous */
getTrustedPooledConnection

/* Call is a connection object. Cast it to a PooledConnection object. */
javax.sql.PooledConnection pooledCon = (javax.sql.PooledConnection)objects[0];
properties = new java.util.Properties();

// Set new properties for the reused object using
// properties.put("property", "value");
// The second item that was obtained from the previous
getTrustedPooledConnection

/* call is the cookie for the connection. Cast it as a byte array. */
byte[] cookie = ((byte[])(objects[1]);

/* Supply the user ID for the new connection. */
String newUser = "newuser";

// Supply the name of a mapping service that maps a workstation user
// ID to a z/OS RACF ID
String userRegistry = "registry";

/* Do not supply any security token data to be traced. */
byte[] userSecTkn = null;

/* Do not supply a previous user ID. */
String originalUser = null;

// Call getDB2Connection to get the connection object for the new
// user.
java.sql.Connection con = ((com.ibm.db2.jcc.DB2PooledConnection)pooledCon).
getDB2Connection(cookie,newuser,password,userRegistry,userSecTkn,originalUser,properties);
```

Conclusion

In the e-commerce world, many applications rely on the security aspects of middleware servers like Domino and WebSphere Application Server, and trusted context is an ideal way to ensure a secure environment for such applications without much performance degradation. The
security attributes of trusted context can be tuned to ensure air-tight access to the database servers. Furthermore, the ability to switch users with minimum connection resources and without authentication, makes trusted context an ideal choice for business applications.
Related topics

- Download a free trial version of DB2 Enterprise 9.
- Now you can use DB2 for free. Download DB2 Express-C, a no-charge version of DB2 Express Edition for the community that offers the same core data features as DB2 Express Edition and provides a solid base to build and deploy applications.
- Build your next development project with IBM trial software, available for download directly from developerWorks.
- Visit the DB2 9 Documentation page for more details about DB2 for Linux, UNIX, and Windows latest release.
- "DB2 Connect User's Guide" (developerWorks): Find more details on how to use a trusted connection with DB2 Connect.
- "An Overview of DB2 and Java DataBase Connectivity (JDBC)" (developerWorks, March 2002): Learn the basic JDBC APIs for programming in DB2 environment.
- Visit the developerWorks resource page for DB2 for Linux, UNIX, and Windows to read articles and tutorials and connect to other resources to expand your DB2 skills.
- Learn about DB2 Express-C, the no-charge version of DB2 Express Edition for the community.