http://www.sqlservercentral.com/articles/Security/sqlserversecurityprosandconsofapplicationroles/1116/
Printed 2008/07/25 11:28AM

SQL Server Security: Pros and Cons of Application Roles

By Brian Kelley, 2003/08/24

Application roles aren't talked about frequently when database roles are discussed. The idea behind applications roles is a valid one: if a user comes through a particular application (and that application only), there should be a mechanism where the user gains privileges specific to that application. For instance, if I have a call tracking system, all users may need SELECT rights against tables for ad hoc queries. However, the users should not have any means to edit the tables unless they come through my special call tracking system. I expect the data to be handled in a certain way and I don't want anyone manipulating data outside of my application. But how can I give users the ability to only read data in one context yet modify data in another? One answer is with the application role.

If I utilize application roles, I can grant a user the ability to read the tables and that's that. When the user comes through the application, the application executes a special stored procedure to activate the application role and the application's connection (this is an important point I'll discuss a bit later) has the new permissions of the application role. I haven't had to grant the user two different logins (such as with a service account or an additional SQL Server login specifically for the application), but I have managed to build two different contexts so far as permissions are concerned for my given user. And if you aren't very familiar with application roles you're probably thinking, "What's the catch?" Of course there is one (actually more than one). That's what this article is about: the pros and cons of using application roles.

Now for the disclaimer: I'm biased. I don't like application roles. While I agree with the concept behind them, I've not found many uses for them where the pros outweigh the cons. While I've considered their use in several major applications my organization has written, each time the application roles have proven themselves unworkable. I'll try and be fair and balanced in this article, but I've provided this disclaimer in case I'm not.

Pro: Application-specific Permissions

This pro is the whole reason application roles exist. If I want to tailor permissions for a particular application, I can do so using application roles. I create the application role, assign permissions to the application role, and then when the application connects to SQL Server, I execute a stored procedure to activate the application role. Immediately I have all the permissions of the application role, completely replacing any previous permissions I might have had. The star players for using application roles are a pair of stored procedures:

Stored Procedure	Purpose
sp_addapprole	Creates an application role in the existing database.
sp_setapprole	Activates an application role for the current connection

I'll go back to my example. In my Call Tracking System, I have a table TroubleTickets. Because management wants to be able to produce reports against this table and they could come up with any number of queries using various third-party reporting applications, I've had no choice but to grant SELECT rights for certain users against the TroubleTickets table. Of course, some of these users are also using the Call Tracking System to manage tickets. If I grant INSERT and UPDATE rights to the users via a normal database role, that means these particular users could make changes in TroubleTickets through Microsoft Access, Query Analyzer, Enterprise Manager, or some other tool. They wouldn't be making changes through my application. I could go a step further and force all inserts and updates to the table through various stored procedures and at least I've stopped users from directly modifying the table. Are we in the clear yet? Not necessarily. What if my call tracking system has business rules that we've found too complex to model with stored procedures and still get a reasonable amount of performance? Then I'll build those business rules into my application and the rules will be applied before any stored procedure is called. I'm relying heavily on the application, aren't I? But if a regular user has direct access to the stored procedures, the user can bypass all the business rules I have in place. This is a problem. Here's where an application role can be helpful because I can give a regular user different permissions depending on context. A bit of code shows this advantage:

```
-- Role to handle those who have to report against the
EXEC sp_addrole 'CallManagementReporting'
GO
```

```
-- Grant SELECT access to TroubleTickets for this role
GRANT SELECT ON dbo.TroubleTickets TO CallManagementReporting
GO

-- Application Role to handle data manipulation through the use of stored procedures
EXEC sp_addapprole 'CallTrackingApplication', 'SomePa$$word'
GO

-- Grant EXECUTE right on a stored procedure to the application role
GRANT EXEC on dbo.usp_AddTicket TO CallTrackingApplication
GO
```

I've created a database role called CallManagementReporting to handle queries against TroubleTickets. I've also created an application role titled appropriately CallTrackingApplication and I've granted it the ability to execute dbo.usp_AddTicket. Only this particular application role has permission to execute the stored procedure and you have to know the password (which we won't give out) in order to active the role. With the application role I've been able to create different permissions based on context.

Pro: Potentially Fewer Logins Required

I use the word potentially cautiously. The use of application roles can reduce the number of logins if personnel already have valid logins to SQL Server. If I have to maintain separate security contexts and I don't use application roles, then I have to use a service account or some other sort of shared login for the application. One more login may not seem like much, but it's one more account that potentially can be compromised. If I'm trying to control access, I want as few accounts as possible. It's much easier to keep an eye on three or four logins than twenty. If I set up the application role, users log on to SQL Server using their normal logins and the application activates the role. But while an application role in this context can reduce the number of logins, it could also cause the opposite effect. And that leads me to a discussion of the cons.

Con: Potentially More Logins Required

If I have the situation where users don't normally have access to SQL Server except through the application, then they probably don't have logins to the SQL Server. If I stick with using application roles and I throw out the idea of a shared account, I have to enable more logins for the SQL Server than I would previously. I can reduce the administrative overhead with the use of Windows groups (and as far as SQL Server is concerned I grant access to the group and not the individual users) but what I don't prevent is the ability to do reconnaissance.

For instance, a user with a valid login has the ability to query against SQL Server and issue a statement like SELECT @@VERSION. Issuing this one statement tells the user two valuable pieces of information: the version of SQL Server and the version of the operating system (including service pack). I can look at these versions and get an idea of what vulnerabilities the system has. If I see a SQL Server version prior to SP2, I know I can exploit the SQL Server listener service on UDP/1434 as Slammer did. If I see a Windows 2000 version prior to SP2, I know the RPC DCOM vulnerability Blaster and its variants have been exploiting is present. And while we try to ensure all of servers are at the latest service packs and hotfixes, the truth of the matter is often this isn't the case.

Obviously we run this risk with any login. If a user can logon to SQL Server, the user can execute this query. As a result, minimizing the number of logins is always preferred. Therefore, if there are a number of users who don't log into SQL Server except through the application it may be a better practice to use a service account or single login and have the application handle the security. A good example is an intranet application used by the entire company. In my organization's intranet application, a single login is used because the majority of users only access the SQL Server due to the intranet site. Therefore, there is no reason to grant them each logins. Instead, a single login is used and the intranet application handles the security.

Con: OLE DB Resource Pooling Issues

From a performance standpoint, opening and closing database connections are resource intensive and thus the concept of connection or resource pooling has come about. My application may make a call to close a database connection, but I'll leave it open for a set period of time in case I receive a call to open a new connection with the exact same credentials. I've therefore saved the resources required to close the connection and then open a new one. While one connection here or there may not seem like a lot, if I have applications making lots of connections, I can see a performance improvement by using connection or resource pooling. However, if I chose to use application roles, I won't be able to use resource pooling.

Application roles take effect for the remainder of a given connection. If I'm maintaining connections in order to improve performance that means my application role will apply to any "new" connection if the credentials are the same. At first glance

we might say, "Oh, the horror! Application roles mean we take a hit to performance." But that's usually not the case, and this may not be a con in most configurations using application roles. Keep in mind the whole reason we'd implement application roles is to give our users different security contexts. We expect users to still logon to SQL Server using their own "personal" credentials. The application role would override their credentials and replace them with its own. So if a user is coming through an application which activates an application role, the user is still initially logging on to SQL Server using his or her individual logon. Connection and resource pooling works by reusing connections if the credentials match those of a previously open connection. Coming through Query Analyzer, if I've already set the application role for a connection and I call sp_setapprole again, I will receive the following error:

> Server: Msg 2762, Level 16, State 1, Procedure sp_setapprole, Line 41
> sp_setapprole was not invoked correctly. Refer to the documentation for more information.

You could also see a "General Network Error" instead of this one and both are related to the Resource Pooling issue. The workaround given by Microsoft is to disable resource pooling altogether. Since ADO uses Resource Pooling by default, you'll have to add this to your connection string:

```
OLE DB Services = -2
```

This will disable Connection Pooling and you'll therefore avoid the error. I've included a link in the Additional Resources section to the Microsoft Knowledge Base article on this issue.

Con: Crossing Databases Requires the guest User

If I have a multiple database application, I probably don't want to use an application role. When an application role takes effect, it overrides any database and login credentials for the connection. That means if I try to access another database using the three-part naming convention like Northwind.dbo.Customers, I won't map into the database based on my original login. The only user I can use is guest, if it is defined. If guest is *not* defined, I run into problems. While SQL Server remembers who I am, it doesn't let me map into the database. If I try, I'll receive an error like so:

> Server: Msg 916, Level 14, State 1, Line 1
> Server user 'MyDomain\MyUser' is not a valid user in database 'SecondDatabase'.

Not only do I not map into a second database based on my login, but I'm also restricted from using the USE command to switch databases. If the guest user isn't enabled in the second database, I receive an error like the one above. If I have enabled the guest user in the database I'm trying to switch to, I'll receive a different error which makes it clear the application role is stopping me:

> Server: Msg 8194, Level 16, State 1, Line 1
> Cannot execute a USE statement while an application role is active.

Therefore, if my application requires the use of more than one database I have to enable the guest user in the database(s) where the application role doesn't reside. Once I invoke the application role, I can only get to them using the guest user.

Con: Server Roles Are Overridden

Just as my ability to access databases is restricted, so too is my ability to use anything granted to me by a server role when I enact an application role. On most servers I control I have sysadmin fixed server role rights, meaning I can do anything I want on the given SQL Server. I can certainly execute sp_addlogin to add a login to SQL Server. But once I enact an application role, my server role permissions are gone. If I were to try and execute sp_addlogin after activating an application role, I'll receive the following error:

> Server: Msg 15247, Level 16, State 1, Procedure sp_addlogin, Line 17
> User does not have permission to perform this action.

Being a member of sysadmin means I'm automatically a member of securityadmin. However, if I look at the block of code in sp_addlogin that generated the error, I see the following:

```
-- CHECK PERMISSIONS --
IF (not is_srvrolemember('securityadmin') = 1)
begin
   dbcc auditevent (104, 1, 0, @loginame, NULL, NULL, @sid)
```

```
   raiserror(15247,-1,-1)
   return (1)
end
ELSE
begin
   dbcc auditevent (104, 1, 1, @loginame, NULL, NULL, @sid)
end
```

I can't add the user because I no longer register as a member of the securityadmin fixed server role. This can be a problem but I have to look at this con from a realistic perspective. If I'm coming through an application which is going to enact an application role, it is very unlikely I'm going to use any permissions related to a fixed server role. I'm going to use Enterprise Manager, Query Analyzer (the most likely in my case), or some third-party database administration tool, not some application which will change my security context. This con shouldn't be a show-stopper but it is one we need to be aware of as DBAs.

Con: Protecting the Password Is a Concern

Application roles should have passwords. Otherwise, a user could just enact an application role any time he or she wanted to using some means of connecting to the database other than through the application. With additional passwords comes additional management responsibilities. But we face this same issue when dealing with service accounts and shared logins. Typical techniques used to protect these passwords work for application roles with a couple of exceptions.

The first exception is it is possible to enact an application role without encrypting the password. While encrypting the password is a relatively simple affair, I can't prevent someone from activating the application role without encryption. For instance, I can make the following call:

EXEC sp_setapprole 'CallTrackingApplication', 'SomePa$$word'

This request is perfectly legal so far as SQL Server is concerned but as you can plainly, see, the password is in clear view. However, as Figure 1 shows, Profiler won't reveal the password or even the application role being invoked. Rather, it records that an sp_setapprole call was made but provides no additional information.

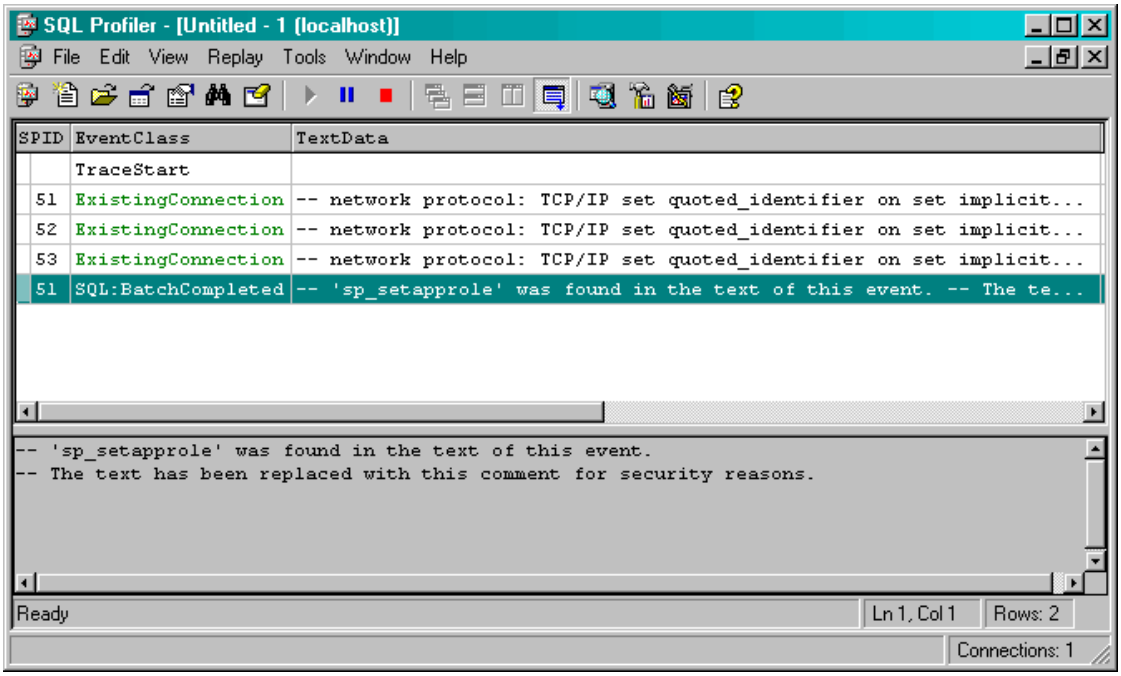

Figure 1: SQL Profiler hides details about the sp_setapprole call

If you're running SQL Server 2000 patched to SQL Server SP3 or higher, the only possibility for getting the password is if it travels across the network and the communication between SQL Server and the client system is unencrypted. As of SP3, DBCC INPUTBUFFER() only returns the name of the stored procedure if the event was an RPC call. The function fn_get_sql() will automatically filter out security related stored procedures like sp_setapprole as well. However, if you're running a SQL Server version prior to SQL Server 2000 SP3, DBCC INPUTBUFFER() can reveal the password so there is a method to

getting method without sniffing network traffic. Keep in mind the execution rights on DBCC INPUTBUFFER, though. Regular users can execute DBCC INPUTBUFFER() for their SPID only. Sysadmins can execute DBCC INPUTBUFFER() on any SPID, but then again they have all rights on the SQL Server anyway. If you still want the password encrypted, here's how to force the password through the ODBC Encrypt function:

EXEC sp_setapprole '<*Application Role*>', {Encrypt N'<*password*>'}, 'odbc'

Using the example I've already given:

```
EXEC sp_setapprole 'CallTrackingApplication', {Encrypt N'SomePa$$word'}, 'odbc'
```

The second issue is the strength of the encryption. The encryption method is extremely weak. If the communication between the application and the SQL Server travels across the network and that communication isn't encrypted, someone can sniff the packets and get the encrypted password for the application role. However, the encryption method is the same as with a SQL Server login, so we've not incurred any greater risk when we talk about the network. Unlike, SQL Server logins, however, I could potentially catch the sp_setapprole request using DBCC INPUTBUFFER() if I'm accessing a SQL Server system prior to SQL Server 2000 SP3 as I've previously discussed. But then again, I have to be a sysadmin to do this and therefore I'd say this risk is relatively small. If you're unfamiliar with how SQL Server login passwords are encrypted as they travel across the wire, you can read up on the encryption algorithm in a previous article I've written. I've added it to the Additional Resources section of this article.

Depending on how application role passwords are stored for a given application, application roles may present a third issue. If application role passwords are hard-coded into a compiled application, then the password can never be reset. That means if someone should compromise the application role password, there is no option to reset the password. Of course, if we're dealing with SQL Server logins this can also be the case. I still remember my dismay when I installed a third-party application that had hard-coded the SQL Server login and password into the application. Of course, the database scripts to create the user were contained in a text file meaning if you had a copy of the application, you knew the username and password for any installation of that application. Application roles can suffer from the same problem. While password management can be a big issue, I don't think the answer is to hard-code them into applications.

Concluding Thoughts

Application roles certainly have their place. If we need to generate different security contexts for a user based on application, the application role comes to the forefront. Provided our users already have logins, the use of an application role can therefore eliminate the need for service accounts or other shared logins. On the other hand, if not all of our users have logins for a given SQL Server and we stick with application roles, we'll end up creating more logins. In this case it may be better to use a shared login. There are some other issues with application roles, the most prominent being with resource pooling. Also, if we need to use more than one database or if we need to use permissions granted to a server role, application roles pose a problem because they override all other login and user contexts. This isn't as big a concern for server roles since we wouldn't tend to use them in an application that would enact an application role, but losing user context when crossing databases means our only option is the guest user in these other databases. Finally, when we use application roles we have to deal with the normal issues regarding password management. The same issues with handling passwords with SQL Server logins also apply to application roles.

I've tried to give fair coverage of application roles and present both the pros and cons. Their validity as a security option is different from application to application and as with any reasonable security solution there are cases where application roles make sense and other cases where they do not. Therefore, blanket statements like "Application roles are bad," or "Application roles are always the way to go," aren't helpful nor are they reasonable. You'll have to make the call on any application you're designing the database security on.

Additional Resources

- Books Online: Establishing Application Security and Application Roles
- Microsoft Knowledge Base Article - PRB: SQL Application Role Errors with OLE DB Resource Pooling
- SQL Server Security: Login Weaknesses

© 2003 by K. Brian Kelley. http://www.truthsolutions.com/
Author of Start to Finish Guide to SQL Server Performance Monitoring (http://www.netimpress.com).