

SQL Server Separation of Duties
SQL Server White Paper

**Author:** Lara Rubbelke

**Contributor:** Kathi Kellenberger

**Reviewers:** Jack Richins, Darmadi Komo

**Published:** April 2011

**Applies to:** SQL Server 2008, SQL Server 2008 R2

**Summary:**

SQL Server 2008 R2 provides the full breadth of tools to support restrictive compliance and security requirements. This whitepaper will associate the features and options available in SQL Server 2008 R2 to meet the varying separation of duties requirements to enable the DBA to be productive and responsive while protecting applications and sensitive data.

© 2011 Microsoft Corporation. All rights reserved. This document is provided "as-is." Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

Contents

[Why Separation of Duties? 3](#_Toc292120996)

[Applying Security Principles to the DBA 4](#_Toc292120997)

[The role of sysadmin 4](#_Toc292120998)

[The role of sa 5](#_Toc292120999)

[The role of db\_owner 5](#_Toc292121000)

[Granular Server Permissions 6](#_Toc292121001)

[Database Roles 7](#_Toc292121002)

[Auditing and Monitoring Separation of Duties 8](#_Toc292121003)

[Protecting Database and Data Access 10](#_Toc292121004)

[Database and Schema Ownership 10](#_Toc292121005)

[Ownership Chaining 10](#_Toc292121006)

[Authorizing elevated Privileges for Specific DBA Tasks 11](#_Toc292121007)

[Signed Modules 11](#_Toc292121008)

[Separation of Duties Framework 14](#_Toc292121009)

[Execute As and Impersonation 18](#_Toc292121010)

[Conclusion 20](#_Toc292121011)

# Why Separation of Duties?

In the face of internal and external regulatory requirements, organizations require that the security and privileges of users and administrators must be minimized for certain types of data categorized as sensitive. At its core, separation of duties restricts the authority of individuals to minimize opportunities for internal threats and fraud. Many organizations who are implementing separation of duties are now requiring that security is minimized for those types of SQL Server users who have previously defaulted to full control and sysadmin: the DBA.

SQL Server 2008 R2 provides the full breadth of tools to support restrictive compliance and security requirements. This whitepaper will associate the features and options available in SQL Server 2008 R2 to meet the varying separation of duties requirements to enable the DBA to be productive and responsive while protecting applications and sensitive data.

Separation of Duties is not a standard set of rules. Security and compliance requirements will differ in every organization. Achieving separation of duties will begin by identifying and documenting the requirements and restrictions, properly stating the maximum level of authority for sensitive server and database objects. This whitepaper assumes each organization has:

* Classified the different groups of users
* Classified the different groups of application databases
* Categorized the security requirements for server level and database level objects
* Identified the rules and processes that will be necessary to temporarily elevate privileges to complete predefined tasks

The concepts laid out in this whitepaper can be applied to those production application servers and databases which fall under the guidelines for strict separation of duties. This guide is designed for data platform managers and application owners who need to understand the SQL Server concepts and tools that will support compliance requirements related to Database Administrators.

# Applying Security Principles to the DBA

## The role of sysadmin

Do not use the sysadmin role as a general purpose DBA role. Database Administrators should be granted individual permissions on the server and in the databases, and membership in the sysadmin role should be limited to logins which will be used in emergencies or for the rare circumstances which require sysadmin authentication. Every installation of SQL Server should have at least one login in the sysadmin role for use in emergency circumstances or when this level of authority is required. Membership in the sysadmin role should be minimized, and the sysadmin should not be treated as the default role for the DBA.

Sysadmin is a fixed server role in SQL Server that allows members to perform any activity on the instance. SQL Server will bypass the Permission Check Algorithm[[1]](#footnote-1) if the login is a member of the sysadmin fixed server role, which means the sysadmin will never be denied any activity on the instance. Because the sysadmin role carries extremely high privileges, it should be considered a protected role, and should not be used for regular day-to-day management and administration.

Individuals who require management authority at the instance level should be granted individual permissions on the server and databases. CONTROL SERVER often corresponds to sysadmin, although there are some limited processes which will not function unless run under sysadmin. CONTROL SERVER will grant full server level permissions and automatically grant full access to all databases. Unlike the sysadmin role membership, applying individual permissions of CONTROL SERVER will allow a granular approach to granting and denying access to individual securables. For example, a company may want to allow the DBA team to manage the instance without access to logins or sensitive tables. In this case, the permissions GRANT CONTROL SERVER and DENY ALTER ANY LOGIN are applied to the DBA logins. In addition, access to individual tables which store sensitive data may be denied to this login.

use [master]

GO

GRANT CONTROL SERVER TO [CORPDOMAIN\DBA\_Tier2\_Team]

GO

DENY ALTER ANY LOGIN TO [CORPDOMAIN\DBA\_Tier2\_Team]

GO

use [AdventureWorks2008R2]

GO

DENY CONTROL ON [HumanResources].[EmployeePayHistory] TO [CORPDOMAIN\DBA\_Tier2\_Team]

GO

## The role of sa

The sa account should never be used for managing SQL Server, and it is a best practice to rename and disable this account. The sa login is a special login created when SQL Server is installed. If SQL Server was installed with Mixed Mode Authentication, the sa login is enabled and a password is required. The sa login is not mapped to any individual, is in the highly privileged sysadmin role, and sa account privileges cannot be reduced.

The following example will disable and rename the sa account. These statements require CONTROL SERVER and ALTER ANY LOGIN permissions.

use [master]

GO

ALTER LOGIN sa DISABLE;

ALTER LOGIN sa WITH NAME = [SQLsa];

## The role of db\_owner

Each database has a fixed database role named db\_owner, and there will always be at least one member in this role in each database. The db\_owner role has one automatic member, which is DBO, and sysadmins automatically map to DBO[[2]](#footnote-2). Members of the db\_owner fixed database role are similar to users which have been granted CONTROL permissions on the database, however these are not synonymous. The db\_owner fixed database role has the CONTROL DATABASE permission, but granting the CONTROL DATABASE permission does not make a user a member of the db\_owner fixed database role. Members of the db\_owner role have rights to execute processes which may only be executed by members of db\_owner, such as some DBCC operations.

While the db\_owner fixed database group may be considered similar to a database-scoped sysadmin server role, there is an important difference. As mentioned above, permissions may not be reduced for those members of the sysadmin role. In contrast, the members of db\_owner may have granular permissions granted and denied. For example, the following group is added to the db\_owner role in the AdventureWorks2008R2 database. This group may be successfully restricted from managing database users through the DENY ALTER ANY USER statement.

USE [AdventureWorks2008R2]

GO

CREATE USER [CORPDOMAIN\DBA\_Tier2\_Team] FOR LOGIN [CORPDOMAIN\DBA\_Tier2\_Team]

GO

EXEC sp\_addrolemember N'db\_owner', N'CORPDOMAIN\DBA\_Tier2\_Team'

GO

DENY ALTER ANY USER TO [CORPDOMAIN\DBA\_Tier2\_Team]

GO

## Granular Server Permissions

SQL Server 2008 R2 supports granting permissions at a granular level. For example, users may be granted rights to view metadata without any access to view the data. Achieving separation of duties requires an approach to assigning permissions that are needed and denying permissions where risk has been assessed.

Prior to SQL Server 2005, administrator permissions at the server level were often granted by using the server-level roles. As a best practice*, do not use fixed server-level roles*. Take advantage of the more than 120 granular permissions available at the server level to manage administrator rights. Determine which permissions are required to accomplish the assigned tasks and grant only those permissions to the administrators. See SQL Server Books Online for the [syntax to grant permissions](http://msdn.microsoft.com/en-us/library/ms178569.aspx) and a detailed list of [permissions applicable to specific securables](http://msdn.microsoft.com/en-us/library/ms191291.aspx#_securables).

As discussed, the most powerful of the SQL Server fixed server roles is the sysadmin role, however other fixed server roles may also exceed the permissions an organization wishes to grant. For example, a DBA may have been added to the SQL Server securityadmin role so they may unlock SQL Server logins. The rights of the securityadmin role will far exceed ALTER ANY LOGIN permissions and the need for the DBA to unlock SQL Server logins such that the DBA may grant any server-level or database-level permission to any other login, up to and including sysadmin. To test the functionality of this role, perform these steps on a test server.

1. Create a new login named “user\_admin” and add the login to the securityadmin role.
2. Create a second login named “user\_test”. Do not add the user any server roles or map to any databases.
3. Connect to your SQL Server with the user\_admin login. Execute the following script to grant CONTROL SERVER to the new user\_test login.

GRANT CONTROL SERVER TO user\_test;

1. While connected to SQL Server as the user\_admin login, impersonate user\_test with EXECUTE AS to gain full control over the SQL Server instance.

EXECUTE AS LOGIN = 'user\_test';

SELECT \* FROM AdventureWorks2008R2.HumanResources.EmployeePayHistory;

REVERT;

The user\_test login now has complete control of the instance granted by an account in the securityadmin role.

Some of the permissions, such as CONTROL SERVER, are effectively a combination of multiple more granular permissions. Run this script to see the list of permissions that roll up to CONTROL SERVER.

SELECT \* FROM fn\_builtin\_permissions(default)

WHERE parent\_covering\_permission\_name = 'CONTROL SERVER'

SQL Server Books Online extends this function with the dbo.ImplyingPermissions function (<http://msdn.microsoft.com/en-us/library/ms177450.aspx>) to return a hierarchy of permissions which may be implied when parent permissions are granted. Create the dbo.ImplyingPermissions function in any database on the server. The following example will use the function to return all of the permissions which will imply ALTER SCHEMA rights.

SELECT \* FROM dbo.ImplyingPermissions('schema', 'alter')

|  |  |  |  |
| --- | --- | --- | --- |
| permname | class | height | rank |
| ALTER | SCHEMA | 0 | 0 |
| CONTROL | SCHEMA | 0 | 1 |
| ALTER ANY SCHEMA | DATABASE | 1 | 1 |
| ALTER | DATABASE | 1 | 2 |
| CONTROL | DATABASE | 1 | 3 |
| ALTER ANY DATABASE | SERVER | 2 | 3 |
| CONTROL SERVER | SERVER | 2 | 4 |

Active Directory (AD) is the most straightforward method of managing permissions across a group of individuals. Follow these steps:

1. Create AD security groups for the DBA team. Each organization may have multiple tiers of DBAs, and a different group will be necessary for each DBA tier that represents different permissions.
2. Add the member accounts of the administrators to the AD groups.
3. Create a Windows login in SQL Server and map to the AD security group.
4. Finely tune the permissions of the new login using the GRANT and DENY commands.

As new administrators are moved into the department, just add them to the appropriate AD security groups.

## Database Roles

Several built-in database roles exist at the database level. Some database roles allow members to peform an administrator task such as backing up the database (db\_backupoperator), while others grant permissions to the database objects. The db\_owner role is the most powerful role at the database level, giving the members full control of the database. These roles are very flexible and it is possible to deny specific persmissions to override database role permissions.

Run this script to see a list of built-in database roles and their permissions.

EXEC sp\_helpdbfixedrole;

EXEC sp\_dbfixedrolepermission;

An additional option at the database level is mapping a user to the dbo user. The dbo user owns the database and it is not possible to remove any permission from the dbo user. Dbo at the database is synonomous to the sa account at the server.

Add user-defined database roles to control permissions for database-wide tasks or privileges at the schema or object level. Object permissions across a large number of users and groups are easier to manage with user-defined database roles. For example, an organization has three tiers of DBAs, and there are three different Active Directory security groups created for these three groups of administrators. The top tier DBA group is the only group with permissions on all tables in a database. Tier 2 and Tier 3 DBA groups must be denied access to sensitive tables. Individual permissions are applied to a single user-defined database group, and the Tier 2 and Tier 3 DBA groups are added as members of the database group.

## Auditing and Monitoring Separation of Duties

As stated earlier in this document, every installation of SQL Server will have at least one sysadmin login with elevated privileges, and the sysadmin logins may open potential insider threats. Those systems which require strict security principles will demand that compensating controls are enforced to monitor the use of restricted and highly privileged logins that may access sensitive server and database objects. These compensating controls are typically considered audits and policies.

Auditing will capture instance level and database level events and write these events to a target. These events may be written to the Windows Security event log, Windows Application event log, or a file. The whitepaper [Auditing in SQL Server 2008](http://msdn.microsoft.com/en-us/library/dd392015.aspx) and [SQL Server Books Online](http://msdn.microsoft.com/en-us/library/cc280386.aspx) are excellent resources to learn auditing concepts and how to implement auditing in SQL Server 2008 R2.

As a practical example, an organization may want to set up an audit to track highly privileged users who manage security. The company requires an audit of occurrences where logins are created, existing logins are added to roles or granted privileges to sensitive objects. This organization also needs to monitor access to a sensitive table Sales.Customer by these highly privileged users. The organization would like these events written to the Windows Application log, which will be monitored with Microsoft System Center. Note that this is one example, and will not provide a comprehensive audit for every scenario. Please review audit requirements and map the server level and database level Audit Events to meet these requirements.

The following will create the audit and audit specifications for the above example.

USE master

GO

--Create a SQL Server Audit to write the audit results to the target

CREATE SERVER AUDIT Admin\_Audit

 TO APPLICATION\_LOG

 WITH ( QUEUE\_DELAY = 1000, ON\_FAILURE = CONTINUE);

--Create a Server Audit Specification for the server audit

CREATE SERVER AUDIT SPECIFICATION Login\_Server\_Spec

FOR SERVER AUDIT Admin\_Audit

 ADD (SERVER\_PRINCIPAL\_CHANGE\_GROUP),

 ADD (SERVER\_PERMISSION\_CHANGE\_GROUP),

 ADD (SERVER\_ROLE\_MEMBER\_CHANGE\_GROUP),

 ADD (DATABASE\_PRINCIPAL\_CHANGE\_GROUP),

 ADD (DATABASE\_ROLE\_MEMBER\_CHANGE\_GROUP)

 ;

--Create a Server Audit Specification for the server audit

USE AdventureWorks2008R2

GO

CREATE DATABASE AUDIT SPECIFICATION Customer\_Database\_Spec

FOR SERVER AUDIT Admin\_Audit

 ADD (DATABASE\_OBJECT\_PERMISSION\_CHANGE\_GROUP),

 ADD (SELECT ON Sales.Customer BY dbo),

 ADD (INSERT ON Sales.Customer BY dbo),

 ADD (UPDATE ON Sales.Customer BY dbo),

 ADD (DELETE ON Sales.Customer BY dbo)

GO

--Enable the Audit Specification

ALTER DATABASE AUDIT SPECIFICATION Customer\_Database\_Spec

WITH (STATE = ON);

--Enable the Audit Specification and Audit

USE master

GO

ALTER SERVER AUDIT SPECIFICATION Login\_Server\_Audit

WITH (STATE = ON);

ALTER SERVER AUDIT Admin\_Audit

WITH (STATE = ON);

In addition to monitoring highly privileged logins and sensitive database activity, it is important to set up policies to maintain the proper configuration requirements. SQL Server 2008 R2 Policy Based Management allows you to specify intent in a policy, store the policy on a server and have the server managed by that policy. In the context of Separation of Duties, intent may be defined as security role membership, audit configuration, and other standards that must be enforced to meet security and compliance goals. SQL Server Policy Based Management is covered in great detail in [SQL Server Books Online](http://msdn.microsoft.com/en-us/library/bb510667.aspx), the [SQL Server Compliance Guide](http://www.microsoft.com/downloads/en/details.aspx?FamilyId=6E1021DD-65B9-41C2-8385-438028F5ACC2&displaylang=en), and a detailed [Policy Based Management whitepaper](http://msdn.microsoft.com/en-us/library/dd938891%28SQL.100%29.aspx). In addition, the [Enterprise Policy Management Framework](http://epmframework.codeplex.com/) is a solution to extend Policy Based Management to all versoins of SQL Server and provide a centralized dashboard of enterprise policy compliance.

As a practical example, an organization has a standard that the sa account is renamed and disabled for every installation. The Database Administrator may create a policy to report on the state of each server to determine if there are any servers that do not comply with this standard. A sample of this policy is available in the policy exmples that are available with the [sample materials](http://download.microsoft.com/download/e/d/3/ed3cdde1-0ffc-4abc-8bc2-d62f206d1a3d/SQLComplianceLab.exe) that complement the SQL Server Compliance Guide.

Individuals with VIEW SERVER STATE permission or users in the PolicyAdministratorRole database role in the msdb database may view Policy evaluation history. Users in the PolicyAdministratorRole database role may also create and edit policies. Care must be taken to only grant the PolicyAdministratorRole to users who should create and manage policies.

# Protecting Database and Data Access

## Database and Schema Ownership

One of the more important elements in securing a database environment is the design of schemas, and schema and object ownership. Schemas are containers of database objects. Each schema and object will have an owner, which may be any database principle including a user, role, or applications role.

Database objects are often created in the default schema dbo, which can make separation of duties difficult. Applications should not use the dbo schema. All objects owned by dbo are chainable to all objects in the dbo schema and all other schemas owned by dbo. Every application should be isolated to individual schemas, and these schemas should have unique owners that align with the security objectives. The separation of ownership from schemas will provide more control over the security of database objects and minimize the unintended implications of ownership chaining.

## Ownership Chaining

An ownership chain will exist when a database object refers to another object and both are owned by the same user. For example, a stored procedure may reference a table in a database. SQL Server will check the permissions of the stored procedure, but SQL Server will skip the permission check of the table if the stored procedure and the table are owned by the same principle. Note that explicitly denying permissions on the table will not be honored in this scenario. If these objects are owned by different principles, known as breaking the ownership chain, permissions for all objects will be checked against the original caller.

Ownership chaining may be deliberate, but it is important to take care that any implicit security permission that is granted is implemented intentionally. While ownership chaining may appear to simplify the management of security and may slightly improve performance, it may unintentionally grant permissions to sensitive objects. When securing database objects from highly privileged users, validate that the security intentions are not broken through ownership chaining. For example, owners of schemas may grant rights to underlying table objects in a way that may not accurately reflect the security intentions.

# Authorizing elevated Privileges for Specific DBA Tasks

To achieve separation of duties, the DBA permissions may be minimized to limit the DBA’s direct access to certain SQL Server objects. At a minimum, the DBA is not in the sysadmin role which means the DBA is no longer granted rights to execute every process on the server. Some server and database diagnostic or management processes and/or views will require elevated privileges, up to and including sysadmin. Organizations may temporarily grant these privileges by using EXECUTE AS or signed modules.

In most cases, signed modules will be more secure and a preferred avenue for achieving the goal of granting privileges. Both EXECUTE AS and signed modules will fill the requirement of enabling specific tasks while minimizing permissions, but these features achieve this objective through different paths. EXECUTE AS will temporarily change the execution context of the caller to an elevated user account to provide the privileges needed to complete a task. signed modules will temporarily add needed permissions to the caller in order to complete a task *without changing the primary identity of the execution context*. Environments that need an auditable process that directly ties a user to each executed command will prefer signed modules to achieve the requirements.

## Signed Modules

Signed modules provide an auditable and secure mechanism to enable permission to execute specific tasks. A module may be a function, trigger, assembly, or stored procedure. Signed modules will enable organizations to maintain the proper level of security while ensuring the DBA is empowered to be responsive with minimal permissions. Initially setting up signed modules may seem more complex than the alternative EXECUTE AS, but the process will be easier to configure for cross-database or server level processes and the result will be a tamper resistant procedure that ensures auditable integrity. The next section will discuss the [Separation of Duties Framework](http://sqlserversod.codeplex.com/) which will ease the process of setting up signed modules for administrative purposes. SQL Server Books Online includes a [tutorial for creating a signed module](http://msdn.microsoft.com/en-us/library/bb283630.aspx).

We will revisit our earlier example which stated the DBA must be able to control the server, but they are not allowed to alter logins. An organization may prefer even more granular permissions which allow a DBA to unlock a login but not have any other access to alter or add logins. The permission ALTER ANY LOGIN does not allow for this level of granularity and will grant more access to the logins than wanted by the organization. This organization may create a signed stored procedure for the specific task of unlocking a login. The signed stored procedure will assign the additional privileges needed to complete the task within the execution of the stored procedure without changing execution context during the execution.

1. **Configure the environment**

Open a new query in SQL Server Management Studio and execute the following statement to create a database that will be used for management procedures. The statement will also create a database master key in the master database if one does not exist. A schema will be created in the DBA\_Admin database which will be used for the management procedures.

USE master

GO

CREATE DATABASE DBA\_Admin

GO

IF NOT EXISTS (SELECT \* FROM sys.symmetric\_keys WHERE name LIKE '%databasemasterkey%')

 BEGIN

 CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'ASDECd2439587y'

 END

GO

USE DBA\_Admin

GO

CREATE SCHEMA [dba] AUTHORIZATION [CORPDOMAIN\DBA\_Tier1\_Team]

GO

IF NOT EXISTS (SELECT \* FROM sys.symmetric\_keys WHERE name LIKE '%databasemasterkey%')

 BEGIN

 CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'pass@word1'

 END

GO

1. **Create a certificate**

The following statement will create a certificate. Backup the certificate and the private key. The certificate backup will then be imported to the DBA\_Admin database.

USE master

GO

CREATE CERTIFICATE cert\_DBAProcedures

 WITH SUBJECT = 'Sign DBA procedures';

GO

BACKUP CERTIFICATE cert\_DBAProcedures TO FILE = 'c:\CertBackup\cert\_DBAProceduresCert.cer'

 WITH PRIVATE KEY ( FILE = 'c:\CertBackup\cert\_DBAProcedureskey.pvk',

 ENCRYPTION BY PASSWORD = 'pGFD4bb925DGvbd2439587y');

GO

USE DBA\_Admin

GO

IF NOT EXISTS (SELECT \* FROM sys.certificates WHERE name = 'cert\_DBAProcedures')

 BEGIN

 CREATE CERTIFICATE cert\_DBAProcedures FROM FILE = 'c:\CertBackup\cert\_DBAProceduresCert.cer'

 WITH PRIVATE KEY

(FILE = 'c:\CertBackup\cert\_DBAProcedureskey.pvk',

 DECRYPTION BY PASSWORD= 'pGFD4bb925DGvbd2439587y')

 END

GO

1. **Create a certificate account using the certificate**

A login is created from the certificate. The login does not have rights to connect directly to SQL Server, and will only be allowed to alter logins.

CREATE LOGIN login\_cert\_DBAProcedures FROM CERTIFICATE cert\_DBAProcedures

GO

REVOKE CONNECT SQL FROM login\_cert\_DBAProcedures;

GO

GRANT ALTER ANY LOGIN TO login\_cert\_DBAProcedures

GO

1. **Create and sign a stored procedure using the certificate**

Create a procedure that will unlock a login. Add a signature to the procedure to sign the procedure with the certificate.

CREATE PROCEDURE dba.sysproc\_UNLOCK\_LOGIN @Login sysname

AS

 DECLARE @cmd nvarchar(max)

 SET @cmd = 'ALTER LOGIN ' + QUOTENAME(@Login)

+ ' WITH CHECK\_POLICY = OFF; ALTER LOGIN '

+ QUOTENAME(@Login) + ' WITH CHECK\_POLICY = ON;'

 EXEC sp\_executesql @cmd

GO

ADD SIGNATURE TO dba.sysproc\_UNLOCK\_LOGIN

BY CERTIFICATE cert\_DBAProcedures

GO

1. **Grant rights to execute the procedure**

The DBA Tier 2 team is granted rights to execute the procedure.

GRANT EXECUTE ON dba.sysproc\_UNLOCK\_LOGIN

TO [CORPDOMAIN\DBA\_Tier2\_Team]

GO

The signed module dba.sysproc\_UNLOCK\_LOGIN will meet the organization requirements to allow the DBA team to unlock the login without any additional privileges on logins. This process will preserve execution context.

Signed modules are a secure and auditable method to temporarily elevate privileges. While signed modules provide many benefits, they may be complex to setup. The Separation of Duties Framework was designed to reduce the effort to set up signed modules.

## Separation of Duties Framework

When SQL Server Granular Permissions do not provide enough granularity, rely on SQL Server signed modules. When signed modules appear too complex or must be replicated across multiple systems, rely on the Separation of Duties Framework. In short, the Separation of Duties Framework will ease the setup of signed modules. In addition, the framework will create one or many tiers of users to support even the more challenging and complex security requirements.

For example, an organization may have a team of junior SQL Server DBAs who must execute tasks that require high privileges. In addition, the organization may have a team of experienced DBAs who must execute additional processes. In both cases, the organization must ensure that these DBAs do not have the rights to view sensitive data stored in the database. The Separation of Duties Framework will ease the implementation of these processes and rules, helping the organization meet their security requirements while authorizing the tasks for each of their DBA teams.

The Separation of Duties Framework was originally designed to address the separation of DBA from sysadmin, but this framework may also be used to temporarily grant users elevation of privileges in a controlled and auditable environment. The [SQL Server Separation of Duties Framework](http://sqlserversod.codeplex.com/) will ease the process of setting up a restrictive environment while providing a predefined set of processes a DBA may use to manage restricted instances and sensitive databases. The Separation of Duties Framework is designed to empower the DBA team to be productive and responsive with processes that are auditable, secure, and extensible while being easy to implement and manage.

The framework will create or use an existing database and a dba schema designed for DBA processes. In this DBA database, the framework will create database roles, signed stored procedures, and the securables needed to support the environment. Full documentation to set up the [Separation of Duties Framework](http://sqlserversod.codeplex.com/) is available in the download files from the codeplex project. The following steps walk through setting up the framework:

1. **Define the roles and tasks.**

Each organization will have different regulations that stipulate the security boundaries for individuals and groups. Prior to installing the Separation of Duties Framework, it is necessary to define the types of roles that will engage with SQL Server and the tasks that each role is permitted to execute.

1. **Create folders to represent the defined roles.**

Create folders in a Procedures directory that will mimic the security roles you identified in the previous step. Remember that these folders are hierarchical, and each folder level will inherit the privileges of the parent folders. The Separation of Duties Framework will create roles based on the folder structure under the Procedures directory.

For example, the sample organization has two tiers of DBA support. DBA Security Low is allowed to unlock logins, and DBA Security High is allowed to unlock logins and issue DBCC CheckDB. Figure 1 represents the folder structure for the Separation of Duties Framework for this organization.



Figure

In this example, the Separation of Duties installation process will create two database roles: DBA Security Level High and DBA Security Level Low. The installation process will create and sign the stored procedures in each of these folders and grant EXECUTE permissions for the roles associated with each folder.

1. **Add stored procedures sql files to the folders created in the previous step.**

Create procedures or use existing example procedures available in the framework that represent the tasks each role is allowed to execute. Place these in the appropriate folder which represents the users who are permitted to execute the task. The Separation of Duties Framework install script will create each procedure, sign the procedures with a certificate, and grant EXECUTE permissions to the appropriate roles.

A number of sample procedures are included in the download folder Sample Administrator Procedures. These procedures represent the types of tasks the DBA team may be permitted to execute, but each organization must determine if these tasks are permitted based on the overall security guidelines. Please carefully review each procedure to ensure they will not expose any risks to the security goals of the organization.

**In the example, the DBA Security Low is only permitted to unlock logins and the DBA Security High is permitted to unlock logins and execute DBCC CheckDB. The organization will copy the sample dba.sysproc\_DBCCCheckDB procedure from the Sample Administrator Procedures to the DBA Security High folder. Using the template in the folder Additional Tools the organization will create a procedure to unlock logins. This new procedure is placed in the folder DBA Security Low as a .sql file. The following is an example of this new procedure.**

USE $(DatabaseName)

GO

--Create a procedure to execute ALTER LOGIN

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID(N'[dba].[sysproc\_UNLOCK\_LOGIN]') AND type in (N'P', N'PC'))

DROP PROCEDURE dba.sysproc\_UNLOCK\_LOGIN

GO

CREATE PROCEDURE dba.sysproc\_UNLOCK\_LOGIN

 @Login sysname

AS

SET NOCOUNT ON

 IF( db\_name() <> N'$(DatabaseName)' )

 BEGIN

 DECLARE @ErrorMessage NVARCHAR(4000);

 DECLARE @ErrorSeverity INT;

 DECLARE @ErrorState INT;

 SET @ErrorMessage = N'This module must be called

from an authorized database. Please contact

the administrator.';

 SET @ErrorSeverity = 16;

 SET @ErrorState = 1;

 RAISERROR (@ErrorMessage, @ErrorSeverity, @ErrorState )

 RETURN;

 END

 IF (@Login IS NULL)

 BEGIN

SET @ErrorMessage = N'Procedure or function

sysproc\_UNLOCK\_LOGIN expects parameter @Login,

which was not supplied.';

 SET @ErrorSeverity = 16;

 SET @ErrorState = 1;

 RAISERROR (@ErrorMessage, @ErrorSeverity, @ErrorState )

 RETURN;

 END

 ELSE

 DECLARE @cmd nvarchar(max), @passwordOptions nvarchar(max), @setOptions nvarchar(max)

 SET @cmd = 'ALTER LOGIN ' + QUOTENAME(@Login) +

' WITH CHECK\_POLICY = OFF;

ALTER LOGIN ' + QUOTENAME(@Login) +

' WITH CHECK\_POLICY = ON;'

 EXEC sp\_executesql @cmd

GO

--Add signature to procedure

ADD SIGNATURE TO dba.sysproc\_UNLOCK\_LOGIN

BY CERTIFICATE cert\_DBAProcedures

GO

--Grant execute to Role

IF '$(RoleName)' <> 'None'

GRANT EXECUTE ON dba.sysproc\_UNLOCK\_LOGIN to [$(RoleName)]

GO

--Add extended properties to describe the objects.

IF NOT EXISTS (SELECT \* FROM ::fn\_listextendedproperty(N'MS\_Description' , N'SCHEMA',N'dba', 'PROCEDURE','sysproc\_UNLOCK\_LOGIN', NULL,NULL))

 BEGIN

 EXEC sys.sp\_addextendedproperty @name=N'MS\_Description', @value=N'Issue ALTER LOGIN command.' , @level0type=N'SCHEMA',@level0name=N'dba', @level1type=N'PROCEDURE',@level1name=N'sysproc\_UNLOCK\_LOGIN'

 END

GO

1. **Execute the PowerShell install script.**

**The Separation of Duties Framework is installed using a PowerShell script.**

Open a PowerShell session. If starting a PowerShell session from SQL Server Management Studio[[3]](#footnote-3), the session will automatically include the SQL Server PowerShell snapins. If starting a Windows PowerShell session, be sure to add the SQL Server PowerShell snapins. SQL Server Books Online includes an excellent reference for working with the SQL Server PowerShell snapins: <http://msdn.microsoft.com/en-us/library/cc281962.aspx>.

Issue a Set-Location to the Install directory of the Separation of Duties Framework. In the example shown in Figure 2 the Separation of Duties Framework install files are located in C:\SQLSoD\Install. Execute the InstallScript.ps1 script.



Figure

This script will ask for a number of values, as depicted in Figure 3. These variables are explained in the Separation of Duties Framework documentation. It is important to note that the InstallScript.ps1 will ask for a database name to use for the signed modules and User-Defined Database Roles. If the user database does not already exist it will create the database. Many organizations may choose to use an existing database that exists for DBA purposes.



Figure

**The PowerShell script will execute a SQLCMD script AdminSetup.sql. This command will create a login from a certificate, but this login cannot be used to connect to SQL Server and will only be used to provide the needed permissions for the signed modules.**

1. **Place the appropriate users and groups into the newly created Database Roles.**

**Open the new DBA database in SQL Server Management Studio and note the two User Defined Database Roles: DBA Security High and DBA Security Low. Add the appropriate users to each role based on the previously defined security requirements. These users will have rights to the appropriate procedures necessary for their roles. Note that these users do not require any additional permissions to execute these procedures. An organization may determine that their DBA team may only interface with a server through signed modules. In this scenario, the organization will add the DBA Active Directory Logins to these roles and will not grant any other permissions on the server or the databases.**

**In our example, the members of the DBA Security Low will execute the procedure dba.sysproc\_UNLOCKLOGIN to when they are notified of SQL accounts that are locked. DBA Security High members will execute dba.sysproc\_DBCCCheckDB to issue the CheckDB command against a database. These procedures will maintain execution context, ensuring the database administrators are able to proceed with their tasks while maintaining compliance requirements.**

The Separation of Duties Framework supports multiple tiers of database administrators and users for a flexible and granular approach to enabling tasks across multiple teams. The framework will ease the process of setting up signed modules for tasks that require elevated privileges.

## Execute As and Impersonation

EXECUTE AS is an alternative to signed modules. When granted the rights, a user may impersonate another login with EXECUTE AS in a session or within a module. It is very important to note that EXECUTE AS will *switch execution context.* This means that the execution context will not appear as the caller on any underlying audit process.

The session level EXECUTE AS is primarily useful for DBAs and developers who may want to test a statement or procedure under the context of another user. For example, a DBA may want to validate a view or stored procedure will produce the expected result for a particular user. To leverage EXECUTE AS in a session the caller must have the rights to impersonate the target user. This must be granted with care as this may undermine security goals.

Impersonation may be enabled on a login or a user. Impersonation privileges on logins are automatically granted to sysadmin and users with CONTROL SERVER. DBO and users with CONTROL DATABASE are automatically granted rights to impersonate any user in the database. Impersonation should be denied to logins if security goals will be compromised for highly privileged users. For example, an organization may have a SQL Server with an application login that has rights to a table HumanResources.EmployeePayHistory which stores sensitive information. The DBA team has been denied privileges on the table, preventing them from directly selecting from the table. If the DBA team has CONTROL SERVER, these individuals may issue the following statement to view the sensitive information.

EXECUTE AS LOGIN = 'AppUser'

GO

SELECT \*

FROM HumanResources.EmployeePayHistory

GO

REVERT

The security team must explicitly deny impersonation to logins with CONTROL SERVER permissions. As stated earlier, sysadmin role members may not be denied any permissions.

use [master]

GO

DENY IMPERSONATE ON LOGIN::[AppUser] TO [CORPDOMAIN\DBA\_Tier2\_Team]

GO

EXECUTE AS may be used in a module to switch the execution context and temporarily grant elevated privileges to the caller. We will discuss EXECUTE AS in the following scenario, but it should be emphasized that signed modules are a better solution for temporary elevation of privileges.

For example, our sample organization has a process which requires the DBA team to occasionally truncate the Sales.CustomerStage table. A stored procedure is created to issue the truncate command, leveraging the EXECUTE AS OWNER so the caller may successfully complete the tasks in the procedure without additional rights on the sensitive table.

USE AdventureWorks2008R2

GO

CREATE PROCEDURE dba.TruncateCustomerStage

WITH EXECUTE AS OWNER

AS

 TRUNCATE TABLE Sales.CustomerStage;

GO

GRANT EXECUTE ON dba.TruncateCustomerStage TO SQLUser2

GO

This procedure will fulfill the technical requirements, but the strategy for meeting these requirements must also align with compliance goals of the organization. As stated earlier, EXECUTE AS will change the execution context, which directly impacts auditing, trace, Profiler , or any tools that collect Login and statement level information. Figure 4 illustrates the trace captured in SQL Server Profiler when a login “SQLUser2” executed the dba.TrucateCustomerStage procedure. Note the SQL:BatchStarting event is executed by Login SQLUser2, but the next command is executed by DBAAdmin (the owner of the dba schema).



Figure

This represents the impact when the execution context is switched using EXECUTE AS. Organizations which require an auditable process to chain the execution of a process to the user must understand these ramifications of switching execution context. Note that this is not the behavior with signed modules which maintains the primary identity of the execution context.

# Conclusion

SQL Server provides the tools necessary to meet and support separation of duties goals. Organizations must begin by defining the security requirements to clearly identify the restrictions necessary to meet their compliance and security goals. Database Administrators should be granted individual permissions on the server and in the databases, and membership in the sysadmin role should be limited to logins which will be used in emergencies or for the rare circumstances which require sysadmin authentication. When more granularity is needed for specific tasks, leverage signed modules and the Separation of Duties Framework.

For more information:

* <http://www.microsoft.com/sqlserver>
* <http://www.microsoft.com/sqlserver/en/us/solutions-technologies/mission-critical-operations/security-and-compliance.aspx>
* <http://www.microsoft.com/about/twc/en/us/default.aspx>

Did this paper help you? Please give us your feedback. Tell us on a scale of 1 (poor) to 5 (excellent), how would you rate this paper and why have you given it this rating? For example:

Are you rating it high due to having good examples, excellent screen shots, clear writing, or another reason?

Are you rating it low due to poor examples, fuzzy screen shots, or unclear writing?

This feedback will help us improve the quality of white papers we release.

Send feedback.

1. See Permission Check Algorithm in SQL Server Books Online <http://msdn.microsoft.com/en-us/library/ms191291.aspx#_algorithm> [↑](#footnote-ref-1)
2. Like sysadmin, DBO will bypass the permission check algorithm. [↑](#footnote-ref-2)
3. PowerShell is available in SQL Server 2008 Management Studio and later. PowerShell is not available in SQL Server 2005 Management Studio. [↑](#footnote-ref-3)