IDERA® | SQL Server Security Whitepaper

TOP 5 ITEMS TO AUDIT IN SQL SERVER

by k. Brian Kelley

Learn auditing methods to identify key database security concerns

TABLE OF CONTENTS

INTRODUCTION

SQL Server provides robust capabilities to monitor itself, and it can be easy to be overwhelmed with the choices presented through catalog or dynamic management views, extended events, server-side traces (for older versions of SQL Server), or performance counters. That is true on the security side as well. This whitepaper presents the top five items you should be auditing on all of your SQL Servers, and how to do them. Keeping an eye on these items will help you verify database security and access to your environment. They are:

1. Who has administrative access to SQL Server?
2. Who has been granted CONTROL and IMPERSONATE permissions?
3. Who are the database owners?
4. What are the respective database permissions?
5. What failed logins are you getting on your SQL Servers?

SECTION ONE
ADMINISTRATIVE ACCESS TO SQL SERVER

Traditionally, we have focused on two areas:

- The system administrator account
- Members of the sysadmin server role

However, we are going to add a third one: the securityadmin role. First, though, we will look at system administrator.

HARDENING *system administrator*

As a general rule, the system administrator account should never be used. It is just like the Administrator account at the operating system level. The best practice for Windows is to rename and disable the Administrator account. While it is possible to discover the renamed account, renaming the account protects against scripts and attempts which target the old name. As a second step, if we disable system administrator, it cannot be used to connect to SQL Server.

Within SQL Server, the system administrator account always has the same principal ID (1). We can use that to execute a simple query to check to see if it has been renamed and disabled.

```
SELECT name, is_disabled
FROM sys.sql_logins
WHERE principal_id = 1;
```

	name	is_disabled
1	sa	0

This simple audit script should be run regularly (preferably via automation) if you do not have a third party means of monitoring this account. That is true of all the scripts provided in this whitepaper.

AUDITING THE *sysadmin* AND *securityadmin* ROLES

The sysadmin role has been on the audit list because a member of that role can do anything within SQL Server. However, securityadmin should be added because a member of that role has the potential to create a login with equivalent rights as a member of the sysadmin role. Before SQL Server 2005, this was not possible. Some guidance still reflects how this role functioned in SQL Server 2000 and older.

Starting with SQL Server 2005 to the latest version of SQL Server 2019, securityadmin gained the ability to grant the CONTROL permission at the server (meaning SQL Server) level, which is equivalent to what a member of the sysadmin server role can do. We will get to auditing for CONTROL permissions next. Concerning auditing for the two server roles, there are two ways to proceed.

The Older Way, Using *sp_helpsrvrolemember:*

A system stored procedure, sp_helpsrvrolemember, is available that will report the members of a fixed server role. For the two roles we are interested in, here are the commands:

```
EXEC sp_helpsrvrolemember 'sysadmin';
EXEC sp_helpsrvrolemember 'securityadmin';
```

For the fixed server roles, the ones that come with Microsoft SQL Server, this method works fine. However, this system stored procedure does not report on user-defined server roles, which were available in the latest versions of SQL Server. Therefore, there is a second way to get role membership.

The New Way, Using Catalog Views:

There are two catalog views we will need to use to get the same information as with sp_helpsrvrolemember. They are:

- sys.server_principals
- sys.server_role_members

The advantage to this method is we can query for members of both roles in a single query:

```
SELECT R.name AS 'Role', L.name AS 'Login'
FROM sys.server_principals AS L
        JOIN sys.server_role_members AS RM
                ON L.principal_id = RM.member_principal_id
         JOIN sys.server_principals AS R
                 ON R.principal_id = RM.role_principal_id
WHERE R.name IN ('sysadmin', 'securityadmin')
ORDER BY R.name, L.name;
```

	Role	Login
1	sysadmin	NT Service\MSSQL$SQL19
2	sysadmin	NT SERVICE\SQLAgent$SQL19
3	sysadmin	NT SERVICE\SQLWriter
4	sysadmin	NT SERVICE\Winmgmt
5	sysadmin	QUICK\
6	sysadmin	sa

SECTION TWO

CONTROL AND IMPERSONATE PERMISSIONS

SQL Server does not just provide the ability to grant administrative rights through roles. It also provides the ability to assign permissions against what are called securables. One of those securables is Server, which corresponds to the SQL Server as a whole. Other securables include traditional objects like tables, views, and stored procedures. SQL Server also considers logins and users as securables. The complete list can be found in the documentation of SQL Server, Books Online. However, for now, we are primarily concerned with four securables in particular: Server, Database, Login, and User.

THE POWER OF CONTROL

For the first two securables, Server and Database, we are concerned with when the CONTROL permission has been granted. CONTROL gives complete control over the securables. Some securables are also called scopes. Scopes are simply securables that can contain other securables, which leads to a hierarchy much like a Windows folder can contain subfolders and files. The Server securables is a scope, and it contains Databases and Logins as well as other securables. Databases are also scopes and contain Users, Schemas, and other database-level objects. There is one more scope, the Schema, which is what contains the traditional objects like tables, views, stored procedures, and functions. Schemas themselves are within the Database scope.

What is important to note about scopes is that when permission is granted to a scope, it carries down to everything contained by that scope, following the hierarchy. If a login has CONTROL permission against Server, that means it has CONTROL permission over everything in SQL Server. Having CONTROL against a Database means having CONTROL over everything contained in the database. That is why auditing for CONTROL is important. Here is how to do it at the Server level:

```
-- For the codes used in class and type
-- See the Books Online entry for sys.server_permissions
SELECT L.name, P.state_desc, P.permission_name
FROM sys.server_permissions AS P
        JOIN sys.server_principals AS L
         ON P.grantee_principal_id = L.principal_id
WHERE P.class = 100
        AND P.type = 'CL'
ORDER BY L.name;
```

You will also want to audit each database. This query needs to be run in the database.

```
-- For the codes used in class and type
-- See the Books Online entry for sys.database_permissions
SELECT U.name, P.state_desc, P.permission_name
FROM sys.database_permissions AS P
        JOIN sys.database_principals AS U
        ON P.grantee_principal_id = U.principal_id
WHERE P.class = 0
        AND P.type = 'CL';
```

Having CONTROL against a schema is also important, but we will discuss permissions below the Database scope shortly.

IMPERSONATE FOR PRIVILEGE ESCALATION

If a login does not have permission to do something, but if it can impersonate someone who does, then the login effectively has the same permissions as the impersonated login. That is also true at the database level with users. Therefore, auditing for IMPERSONATE is important, especially on logins and users with elevated privileges such as system administrator or database owner. Here is how to do so at the server level with the second name being the login that can be impersonated:

```
-- For the codes used in class and type
-- See the Books Online entry for sys.server_permissions
SELECT L.name, P.state_desc, P.permission_name, I.name
FROM sys.server_permissions AS P
        JOIN sys.server_principals AS L
        ON P.grantee_principal_id = L.principal_id
        JOIN sys.server_principals AS I
        ON P.major_id = I.principal_id
WHERE P.class = 101
        AND P.type = 'IM'
ORDER BY L.name, I.name;
```

And here is the query for the database level:

```
-- For the codes used in class and type
-- See the Books Online entry for sys.database_permissions
SELECT U.name, P.state_desc, P.permission_name, I.name
FROM sys.database_permissions AS P
        JOIN sys.database_principals AS U
        ON P.grantee_principal_id = U.principal_id
        JOIN sys.database_principals AS I
        ON P.major_id = I.principal_id
WHERE P.class = 4
        AND P.type = 'IM'
ORDER BY U.name, I.name;
```

There are legitimate uses for IMPERSONATE, which is why it is provided as permission within SQL Server. Therefore, if you encounter it on any of your servers, verify that the occurrence is valid.

SECTION THREE

DATABASE OWNERSHIP AND THE *db_owner role*

Ownership implies control, and that is undoubtedly true with SQL Server. With regards to database ownership, we are concerned with two things:

- The actual owner of the database
- Members of the db_owner fixed database role

AUDITING THE ACTUAL DATABASE OWNER

If a login owns a database, it maps into the database as a database owner (dbo) by default. So database administrator members of the sysadmin role, by the way. The reason this is important is that the database owner user bypasses security checks. Whoever owns a database can do anything he or she wants within it. Therefore, auditing for database owners is essential. The following query may result in the Owner showing as NULL. That can happen if a login owned a database, but the login was subsequently dropped from SQL Server, and the database ownership was never transferred. Any NULL listings should be updated appropriately.

```
SELECT D.name AS 'Database', L.name AS 'Owner'
FROM sys.databases AS D
        LEFT JOIN sys.server_principals AS L
        ON D.owner_sid = L.sid
ORDER BY D.name;
```

AUDITING THE db_owner AND db_securityadmin FIXED DATABASE ROLES

We are also concerned with members of the db_owner role within a database. Members of the db_owner role, unless explicitly blocked by a DENY, also can do anything within the database. That is different from database owner (dbo), for which the DENY will not apply. Although a DENY can block a member of db_owner, that user does have the ability to revoke the DENY, thereby gaining access. As a result, this level of permission should be audited, too. Once again, we have two methods, similar to what we had with the server roles.

Using sp_helprolemember:

There is a system stored procedure at the database level called sp_helprolemember, and it functions as the server level sp_helpsrvrolemember. Therefore, we just need to specify the fixed database role, db_owner, to audit:

```
EXEC sp_helprolemember 'db_owner';
```

Using the catalog views:

The other method is to use catalog views. Again, the view structure at the database level mirrors the server level catalog views.

```
SELECT U.name AS 'User', R.name AS 'Role'
FROM sys.database_principals AS U
        JOIN sys.database_role_members AS RM
        ON U.principal_id = RM.member_principal_id
        JOIN sys.database_principals AS R
        ON R.principal_id = RM.role_principal_id
WHERE R.name = 'db_owner'
ORDER BY U.name;
```

SECTION FOUR
DATABASE PERMISSIONS

Applications use databases, and naturally, we are concerned about database permissions. There are many catalog views around database objects, all of which are necessary to get a complete view of permissions within the database. Let us look at the most typical securables

DATABASE SECURABLES

We have already covered these securables concerning CONTROL permission, but it is important to see all the permissions at the database level. For instance, the CREATE TABLE permission is at the database level. Keep in mind that this permission also grants the ability to ALTER and DROP tables as well. Therefore, here is how to audit everything at the database level:

```
SELECT U.name, P.state_desc, P.permission_name
FROM sys.database_permissions AS P
        JOIN sys.database_principals AS U
         ON P.grantee_principal_id = U.principal_id
WHERE P.class = 0
        AND NOT P.type = 'CO' -- Excluding Connect permission
ORDER BY U.name, P.permission_name;
```

SCHEMA SECURABLES

Because a schema is what contains tables, views, stored procedures, and more, it is important to query the schema permissions as well because remember that SQL Server treats the securables scopes as a hierarchy. Therefore, if I have SELECT permission on a scope, I have permission for all objects within that hierarchy. That carries into other scopes as well. Therefore, if a user has SELECT at the database level, it has SELECT on every table and view in the database. If it has to EXECUTE at the schema level, it has to EXECUTE on every stored procedure within the schema. Here is how to audit the permissions:

```
SELECT U.name AS 'User', P.state_desc, P.permission_name, S.name AS 'Schema'
FROM sys.database_permissions AS P
        JOIN sys.database_principals AS U
        ON P.grantee_principal_id = U.principal_id
        JOIN sys.schemas AS S
        ON P.major_id = S.schema_id
WHERE P.class = 3
ORDER BY U.name, S.name;
```

OBJECT SECURABLES

A user can have permissions directly on tables, views, and other objects. Therefore, this should be audited as well:

```
SELECT U.name AS 'User', P.state_desc, P.permission_name,
        S.name + '.' + O.name AS 'Object'
FROM sys.database_permissions AS P
        JOIN sys.database_principals AS U
        ON P.grantee_principal_id = U.principal_id
        JOIN sys.objects AS O
        ON P.major_id = O.object_id
        JOIN sys.schemas AS S
        ON O.schema_id = S.schema_id
WHERE P.class = 1
ORDER BY U.name, O.name, P.permission_name;
```

I am not accounting for permissions specifically against columns, but the main object. So if a user has permissions against specific columns in a table or view, this query will not show them. You will have to go an additional join deep, joining to sys. columns and using the major_id to identify the object and minor_id to identify the specific column in your ON syntax.

OTHER SECURABLES

There are other securables to check for permissions on such as the ones related to encryption and cryptography. They follow the same format as the queries above, only joining to a different catalog view and using a different class ID for sys.database_ permissions. You can find information on these in Books Online

SECTION FIVE

AUDITING FAILED LOGINS

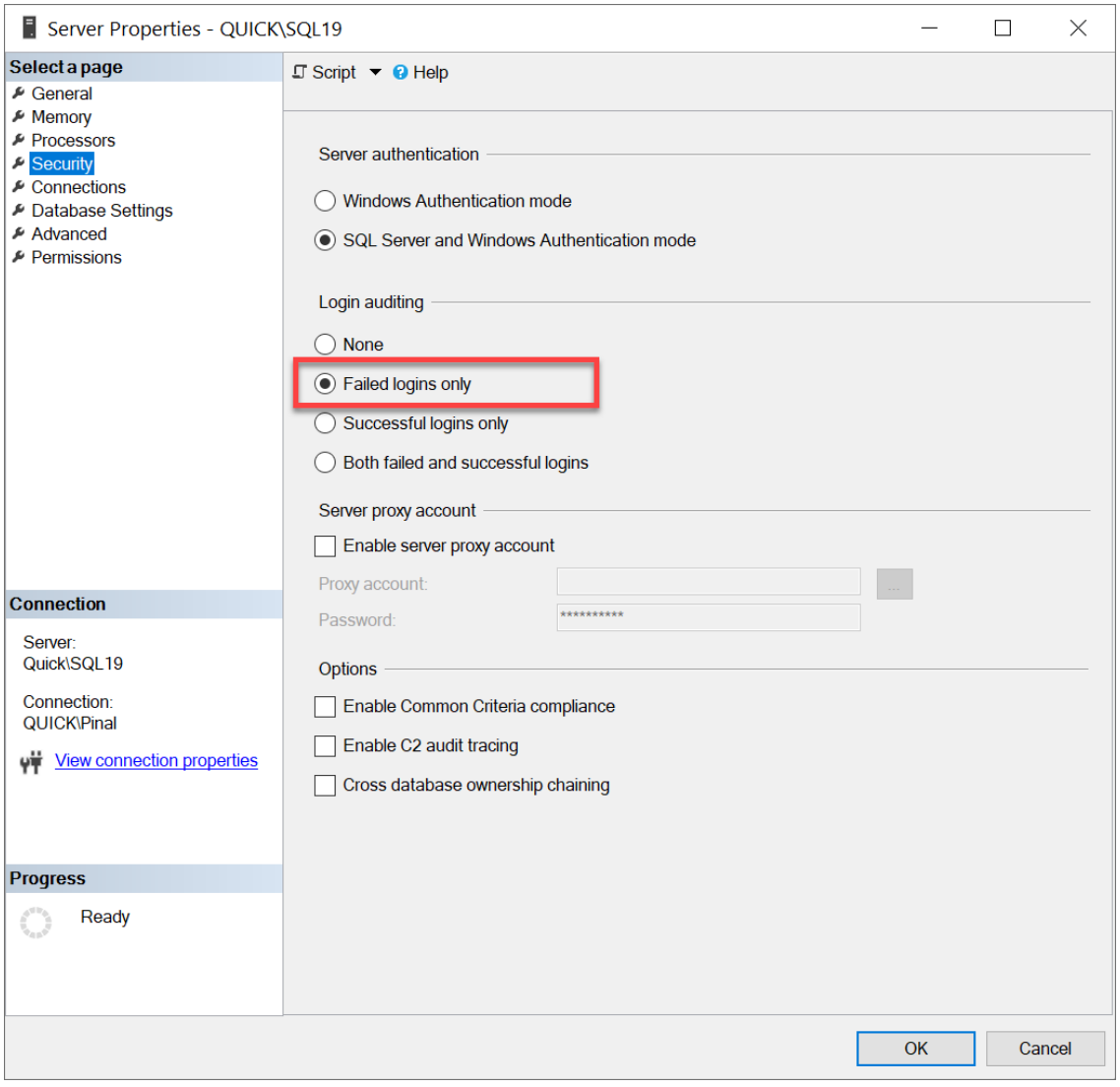
There are two reasons to audit for failed logins:

- To detect when someone is trying to get into SQL Server improperly.
- To detect when a person or application with a legitimate reason to access SQL Server cannot connect.

It is not usual to find that the second reason is the most prominent one for auditing for failed logins. When an application is not working, the failed login will typically pinpoint what is wrong. For instance, if someone mistyped a password that the application will use to connect if you are auditing for failed logins, you will see the failed login and the reason (for example, a bad password). The lack of a failed login when you are auditing for failed logins is telling as well. For instance, if you do not see a failed login and the application is failing to connect, this could point to the connection string having the wrong server, the firewall on one or both servers interfering, or some network issue preventing the communication from occurring. Therefore, auditing for failed logins is essential not just for security, but as a good operational practice.

CONFIGURING AUDITING ON FAILED LOGINS THE REGULAR WAY

To configure an audit for failed logins, right-click on the server in SQL Server Management Studio (SSMS) and choose Properties from the pop-up menu. Then click on the Security page. Look for this part of the dialog window and ensure that Failed logins only are marked. If you have a reason to do, you can mark the entry for Both failed and successful logins but realize that this will put many entries into the SQL Server log and the Application event log if you have a busy SQL Server.



This setting is read when SQL Server starts up. Therefore, for it to take effect, you will need to restart the SQL Server service.

FINDING THE FAILED LOGIN EVENTS IN THE SQL SERVER LOG

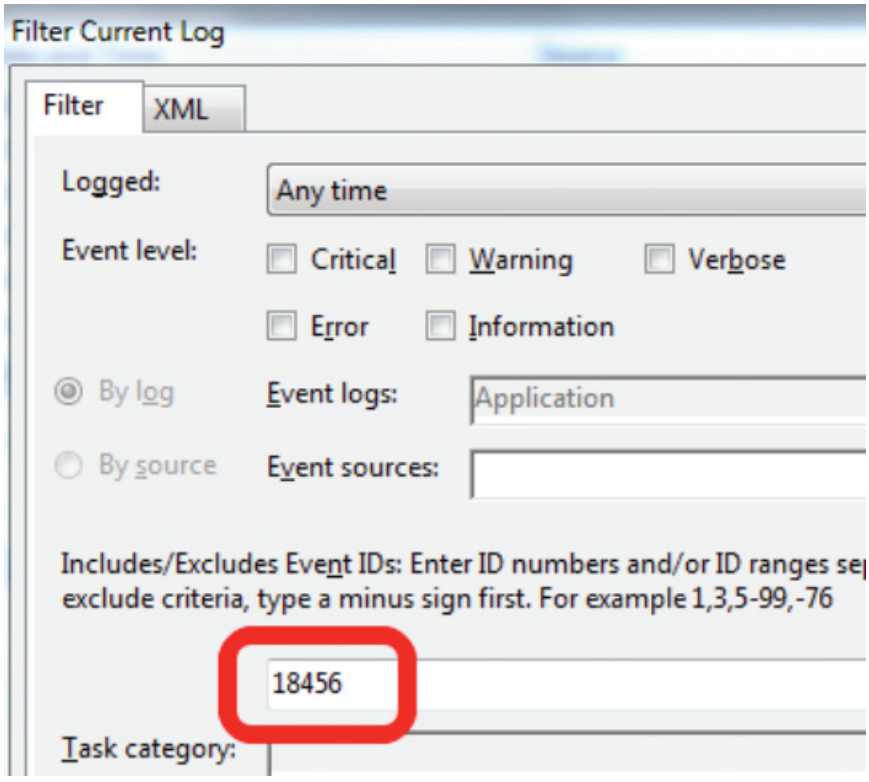
There are two reasons to audit for failed logins:

If you are auditing for failed logins, open up your SQL Server log and look for entries with the Source of Logon. If you are using SSMS, you can even choose to Filter on this source. Typically for failed logins, you will see a couple of entries like so:

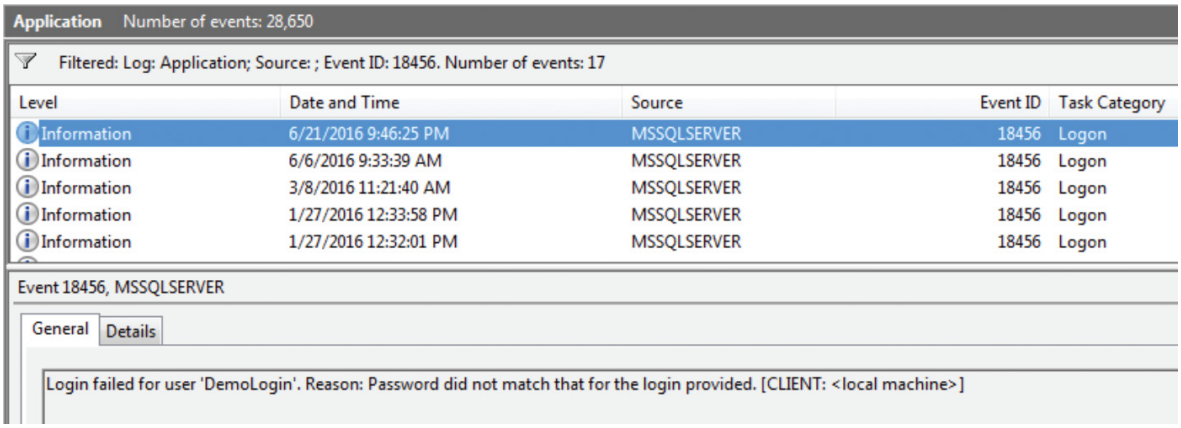
	Source	Message
	Logon	Login failed for user 'DemoLogin'. Reason: Password did not match that for the login provided. [CLIENT: <local machine>]
	Logon	Error: 18456, Severity: 14, State: 8.

FINDING THE AUDIT EVENTS IN THE APPLICATION EVENT LOG

When SQL Server is auditing for logins, it will also write events to the OS Application event log. There are often many events in the Application event log, so the best way to look for these events is to filter the log. If you are in the Event Viewer or another tool (such as Computer Management), which gives you access to the event logs, drill down until you see Application under Windows Logs. Right-click on Application and choose Filter Current Log. In the dialog window, enter the event ID, such as 18456, like so, which will filter the Application log for just those events. We could specify a source, but it will be different depending on if you have a named instance or not.



Once the filter applies, the number of events you will see will have been reduced. Here is an example of what you will see for failed logins. The only event IDs showing are 18456. The details of the event tell us what login failed and why the failure occurred.



If your organization has a security information and event management tool, and it is taking events from the Application event log, it can pick up the failed logins. That would help correlate a path of an adversary in the event of a real security incident. That is another reason to audit failed logins.

SECTION SIX

A WORD ABOUT AUTOMATION TOOLS

We have intentionally included the scripts and information you need to audit the top five audit concerns we have listed. While you can collect all of it manually, it is best if you are collecting this information through automation, such as via scheduled tasks or SQL Server Agent jobs. If you can also process the information gathered automatically, that is even better. Keep in mind that there are third-party applications that do the grunt work for you, including reporting and alerting. Among them are tools of IDERA like SQL Compliance Manager, SQL Secure, and a free tool, SQL Permissions Extractor. They are worth looking into to reduce the amount of time you have to spend auditing the top five items we have presented here.

ABOUT THE AUTHOR

K. Brian Kelley is a SQL Server author, columnist, and Microsoft Most Valued Professional, focusing primarily on SQL Server and Windows security. In addition to being a database administrator, he has served as an infrastructure and security architect encompassing solutions with Citrix, virtualization, and Active Directory. Brian is also a Certified Information Systems Auditor and has been the head of a computer incident response team of a financial organization. Brian is active in the information technology community, having spoken at DevConnections, SQL Saturdays, code camps, and user groups.

SQL COMPLIANCE MANAGER

Improve Any SQL Server Audit

- **Audit Sensitive Data** - see who did what, when, where, and how
- **Track and Detect** - monitor and alert on suspicious activity
- **Satisfy Audits** - for PCI, HIPAA, FERPA and SOX requirements
- **Generate Reports** - 25 built-in reports to validate SQL Server audit trails
- **Minimize Overhead -** lightweight data collection agent minimizes server impact

Start for FREE