SAy our Security Right (or Not)

Neil Hambly, Founder, CIO & Principal Consultant, DataMovements

Moderated By: Narinder Sharma

NeilHambly@datamovements.co.uk

@Neil_Hambly

neilhambly

Neil Hambly

Founder and Principal Consultant of DataMovements
A Microsoft Partner Data & Analytics Consulting company, with DBA Services and Data Analytics focus
UK Clients include NHS, Royal Mail, Safetykleen

https://www.datamovements.co.uk

DATA MOVEMENTS
BRINGING DATA INTO FOCUS

Previous roles @ SQL Server Practice lead @ Northdoor PLC (MSFT Gold partner), Accenture, ABN AMRO, ASOS, BBC, Confio Software & others with 18+ years in SQL Server roles

Presented at 150+ events, user groups and PASS events such as PASS Summit, SQL Saturdays and more.

SAy our Security Right (or Not)

Session Agenda

24HOP - 45 Minutes Sessions with Q&A (10-15 minutes)

- Security – Where do we Start ?
    - Who are you ? What do you want to ?
- Terms
    - Ultra Quick Security Jargon recap
- Windows Login & SQL Server Logins
    - When & Why's of using one login vs other
- 'sa' account
    - Please come this way *and learn more about this using this very special account (or for not using it)*
- Permissions
    - What are these and should I be allowed to do something ?
    - Principle of least privilege design (do we know how to achieve this)
- Ownership
    - This is yours.. Make sure you take could care of it
- Impersonation
    - For my next appearance I will be Tom Jones
    - EXECUTE AS Clause
- Questions & Answers
    - OK what did not makes sense (please don't say all of it)
- Next Session
    - Hacking SQL Server (André Melancia)

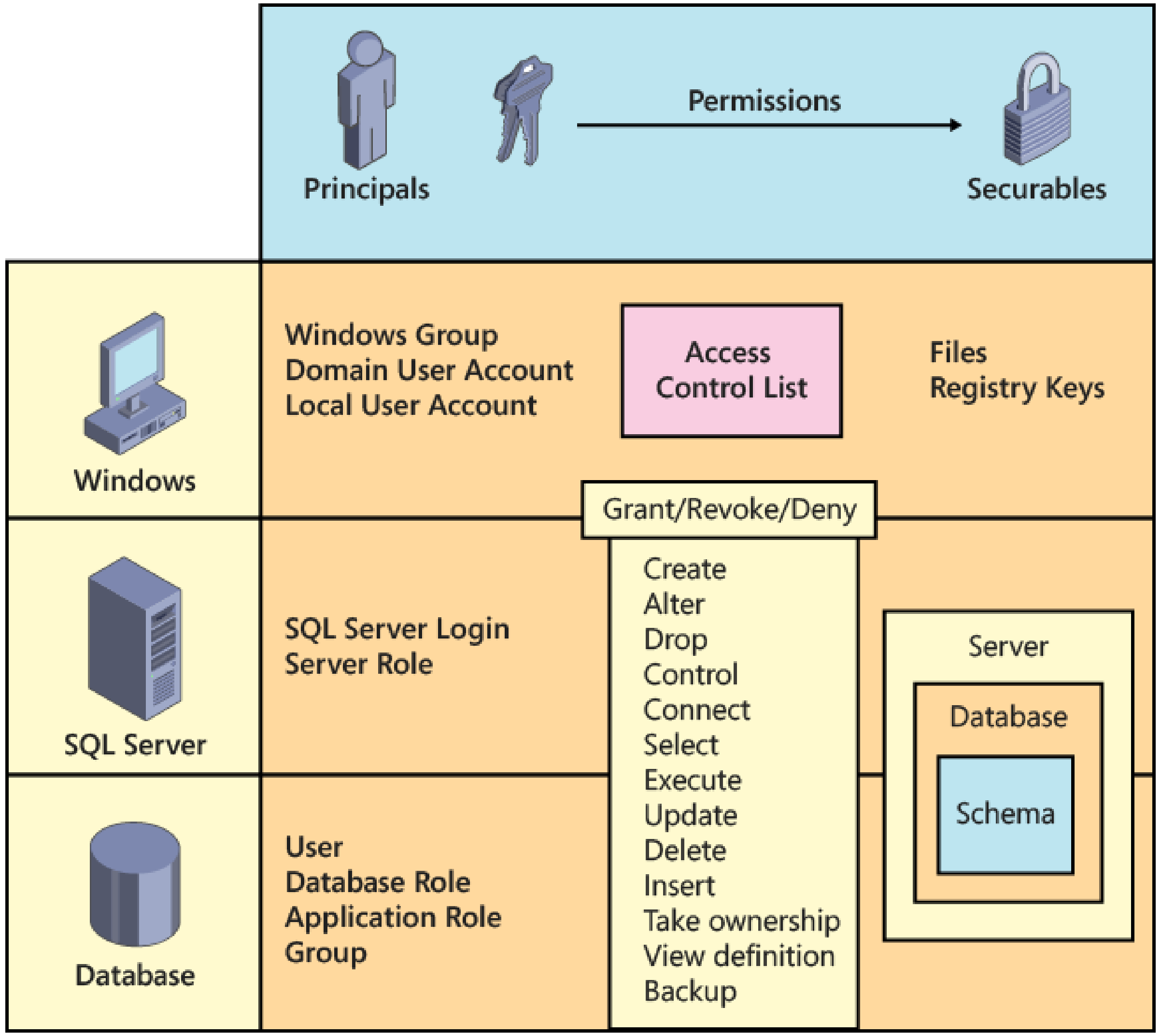
Security .. Where do we Start ?

We are going need to have at a minimum a basic grasp of the different **security groups** that exist within SQL Server, who controls security, who has what permissions and many other fundamental security concepts.

Let's start by examining the diagram on the right, it shows some different groupings, we cover these in more detail as we move through the session, for now take a few seconds to absorb it.

Welcome to the Security REALM

Now, who are you ? give me your credentials



Security Terms

We will start with a quick re-cap here of **Key terms** used when discussing security

- **Users**
  - You, me and well everyone else I guess who wants to visit
  - Database users
- **Logins**
  - How do we recognize you - *Neil will know be known as KOTD ("KingOfTheDancers")*
  - Server Login Permissions (View Any Definition, View Server State, Alter Trace)
- **Sever level Roles**
  - dbmanager, dbcreator , loginmanger, public, sysadmin, processadmin, bulkadmin, setupadmin, securityadmin, diskadmin, serveradmin
- **Database Level Roles**
  - db_owner , db_ddl_admin, db_backoperator, db_datareader, db_denydatareader, db_datewriter, db_denydatawriter, db_secuirtyrole, db_accessadmin, db_ddladmin
- **Authorization**
  - GRANT (*Why Yes you can do that*) & WITH GRANT OPTION (*and you can now also grant*)
  - REVOKE (*Hmm on seconds thoughts*)
  - DENY (*Sorry Not on my watch*)
- **Permissions**
  - What exactly you can do once inside (*and remember to keep all arms and legs inside the ride throughout*)
  - Server Login Permissions (View Any Definition, View Server State, Alter Trace
- **Ownership**
  - Ownership Chains
- **Securables**
  - Server Level
    - Endpoint, Login, Database
  - Database Level
    - User, Role, Application role, Assembly, Message Type, Route, Service, Remote Service Binding, Fulltext Catalog, Certificate, Asymmetric Key, Symmetric Key, Contract, Schema
  - Schema Level
    - Type, XML Schema Collection, Object (Aggregate, Function, Procedure, Queue, Synonym, Table, View)
- **Principals**
  - sys.server_principals, sys.database_principals (*we check out some queries with these DMV's*)

SQL & Windows Logins

There are 2 different type of Logins we can use
{Depending if you have configured SQL Server to allow one or both}

Windows Login

This is the most secure and preferred way {for many reasons} to access SQL Server, with your current (or supplied Windows Credentials) via a local account [ServerName\NeilHambly] or domain login [Datamovements\NeilHambly]

Leveraging Windows AD groups to grant or remove access for a specific account [Login] or group of accounts

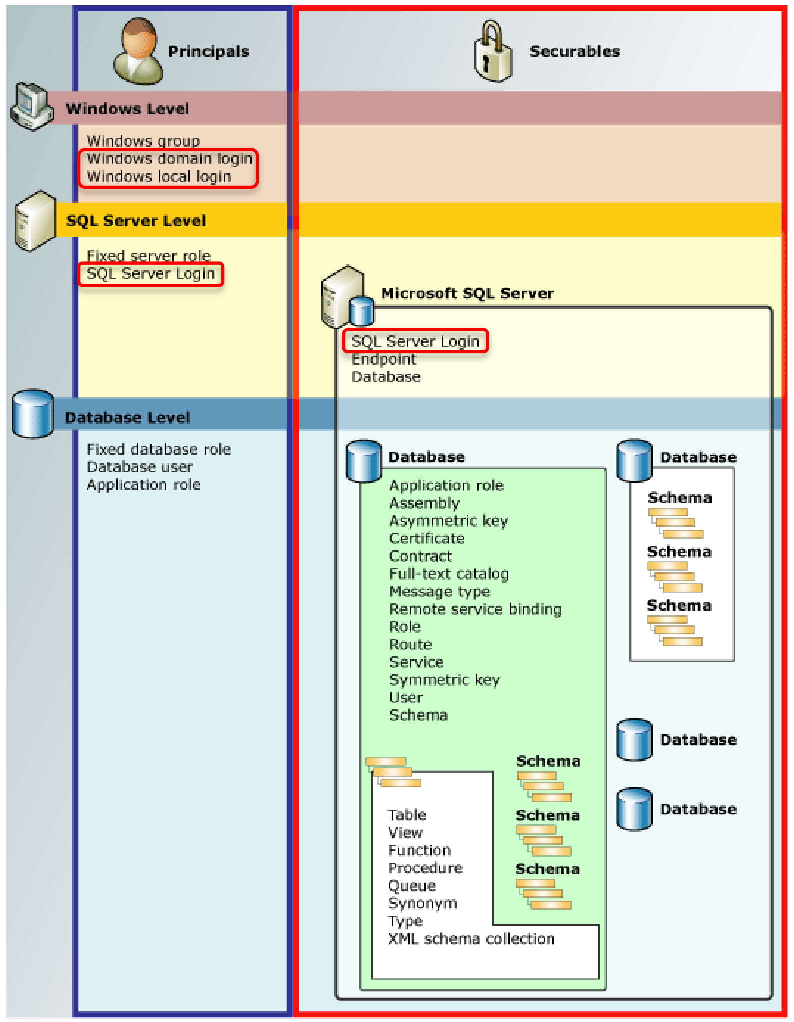
SQL Server Login

This was and still is used heavily, but is less secure and requires considerable more admin to control especially across multiple servers

'sa' account is widely misused, we will cover this in a lot more depth in the session.

When we are finished you will want to disable it and out in place a much stronger security strategy.

Azure Database, only SQL Login was allowed but now can also be via Azure AD



Logins Login(less) & Guest Login

A login only can only be granted authorization to objects in a database
<u>if a database user has been mapped to the login.</u>
A special user [**guest]** exists to permit access to a database for logins that are not mapped to a specific database user.

Any login can use the database through the **guest** user, although it is suggested that the **guest** user <u>not be enabled</u> except in the MSDB database, in order for some SQL Server features to work, the guest user must be enabled in MSDB.

In SQL Server 2005 they introduced a new type of user, a user which is not mapped to any login.
Users not mapped to logins can provide an alternative to using application roles.

Invoking impersonation using the EXECUTE AS statement to allow that user only the privileges needed

You create a user without a login using following DDL

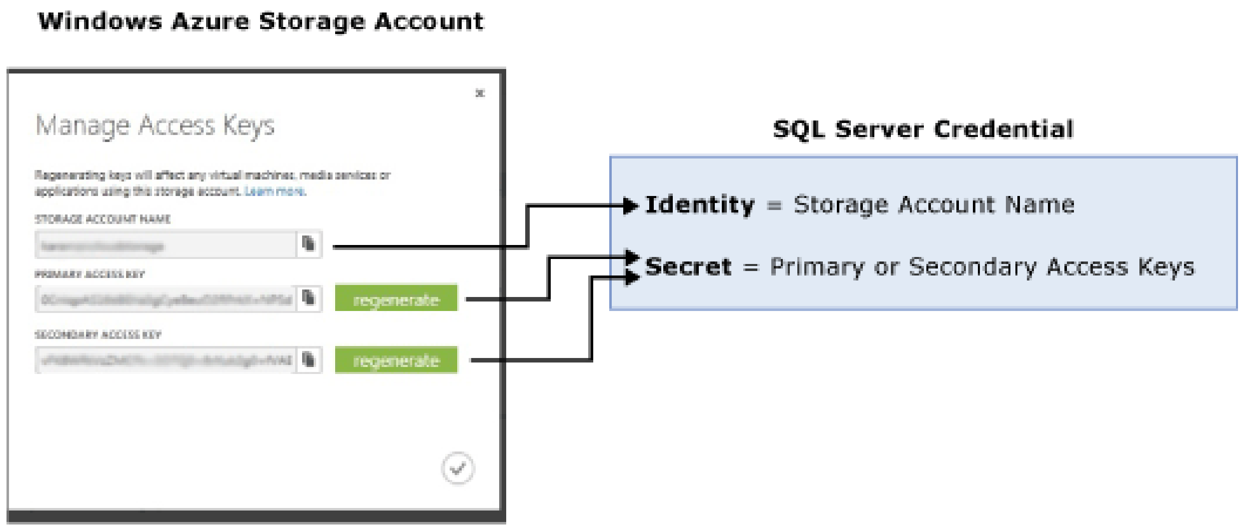
```
-- Creates a user “InternalUser” without a login

CREATE LOGIN InternalUser WITHOUT LOGIN
GO
```

User With Security Credentials

We can create User accounts and associate them to a security credential
Normally credentials would be a Windows Account with the right permissions
Another reason for using credentials could be to access Azure blog storage for storing / retrieving backups

```
CREATE CREDENTIAL AzureBlobCred WITH IDENTITY= ‘<storage account name>'
-- this is the name of the storage account you specified when creating a storage account
, SECRET = '<storage account access key>'
-- this should be either the Primary or Secondary Access Key for the storage account
```



Regenerating keys will affect any virtual machines, media services or applications using this storage account. Learn more.

Identity = Storage Account Name
Secret = Primary or Secondary Access Keys

Hello ‘sa’

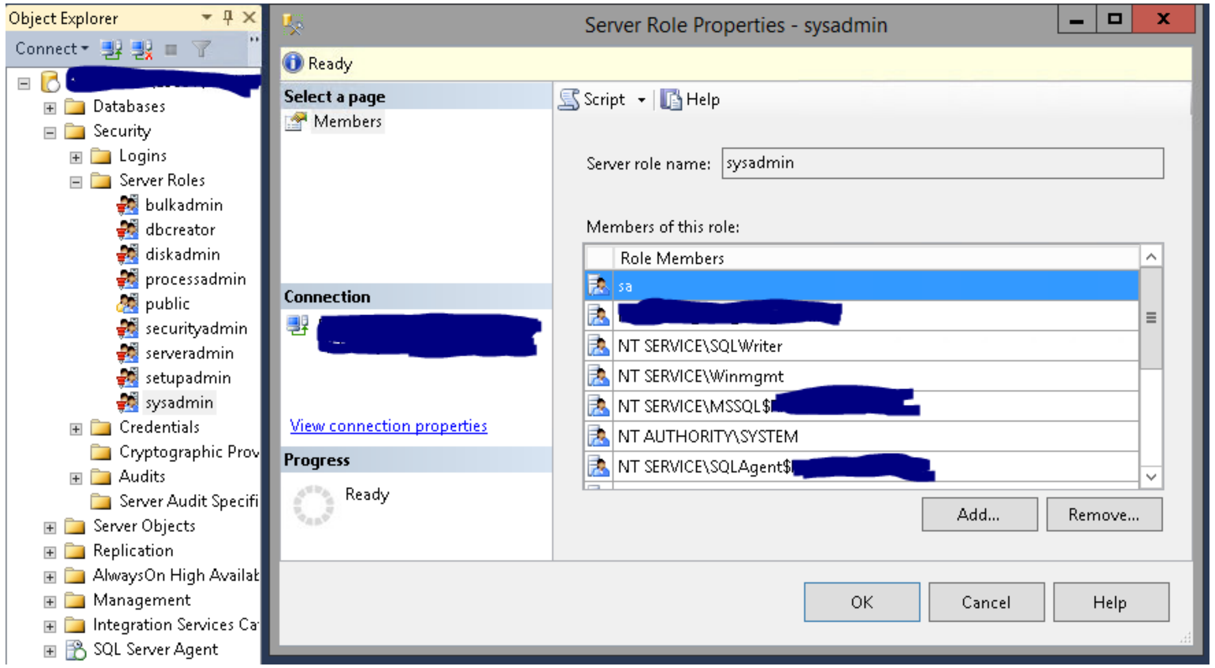
Please come this way

Anyone want to come play ‘lets admin the server’ game

‘sa’ account

Let’s take a more detailed look at this account ‘sa’ [System Administrator]
Doesn’t take a genius to work out this could be a security breach due it’s the level of rights it has
So who has [sysadmin] right’s is really important
Out-of-the-box we have one account with that special privilege ..{you guessed right} the ‘sa’ account
It comes with the sysadmin role (that can’t be revoked)

https://docs.microsoft.com/en-us/sql/relational-databases/security/authentication-access/principals-database-engine
The SQL Server sa login is a server-level principal.
By default, it is created when an instance is installed.
Beginning in SQL Server 2005, the default database of sa is master.
This is a change of behavior from earlier versions of SQL Server.
The sa login is a member of the sysadmin fixed database role.
The sa login has all permissions on the server and cannot be limited.
The sa login cannot be dropped, but it can be disabled so that no one can use it.



Object Explorer
Connect
- [redacted]
  - Databases
  - Security
    - Logins
    - Server Roles
      - bulkadmin
      - dbcreator
      - diskadmin
      - processadmin
      - public
      - securityadmin
      - serveradmin
      - setupadmin
      - sysadmin
    - Credentials
    - Cryptographic Prov
    - Audits
    - Server Audit Specifi
  - Server Objects
  - Replication
  - AlwaysOn High Availat
  - Management
  - Integration Services Ca
  - SQL Server Agent

Server Role Properties - sysadmin

Ready

Select a page
Members

Script Help

Server role name: sysadmin

Members of this role:

Role Members
sa
[redacted]
NT SERVICE\SQLWriter
NT SERVICE\Winmgmt
NT SERVICE\MSSQL$[redacted]
NT AUTHORITY\SYSTEM
NT SERVICE\SQLAgent$[redacted]

Add... Remove...

Connection
[redacted]
View connection properties

Progress
Ready

OK Cancel Help

Usage of ‘sa’ should only be used as a LAST RESORT and not in ANY regular use, those with sysadmin rights should be using their account (Windows Login) then we know who did what. [sys_user]
The ‘sa’ account could be used by anyone who had the password (or could access that somehow reset it) **renaming** or **disabling** it are good ideas.

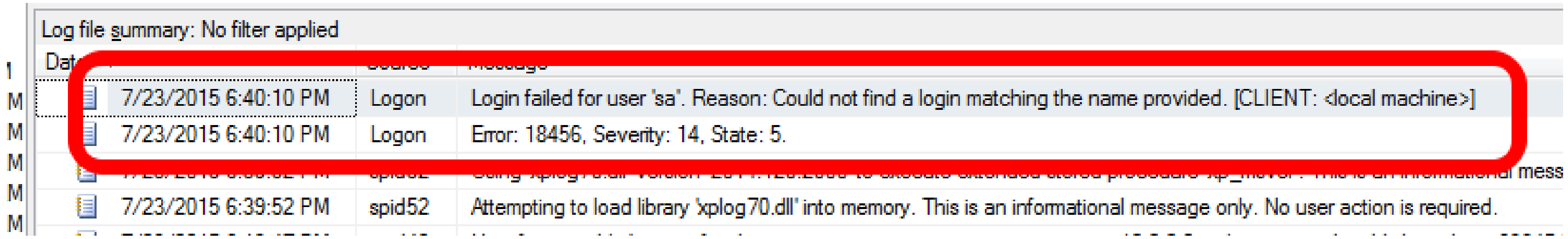
Renaming the ’sa’ login

Renaming the ‘sa’ account is another option, however this **can’t** be done via the Login dialog page

Right-click, Rename is an option

Using T-SQL is another way to do this

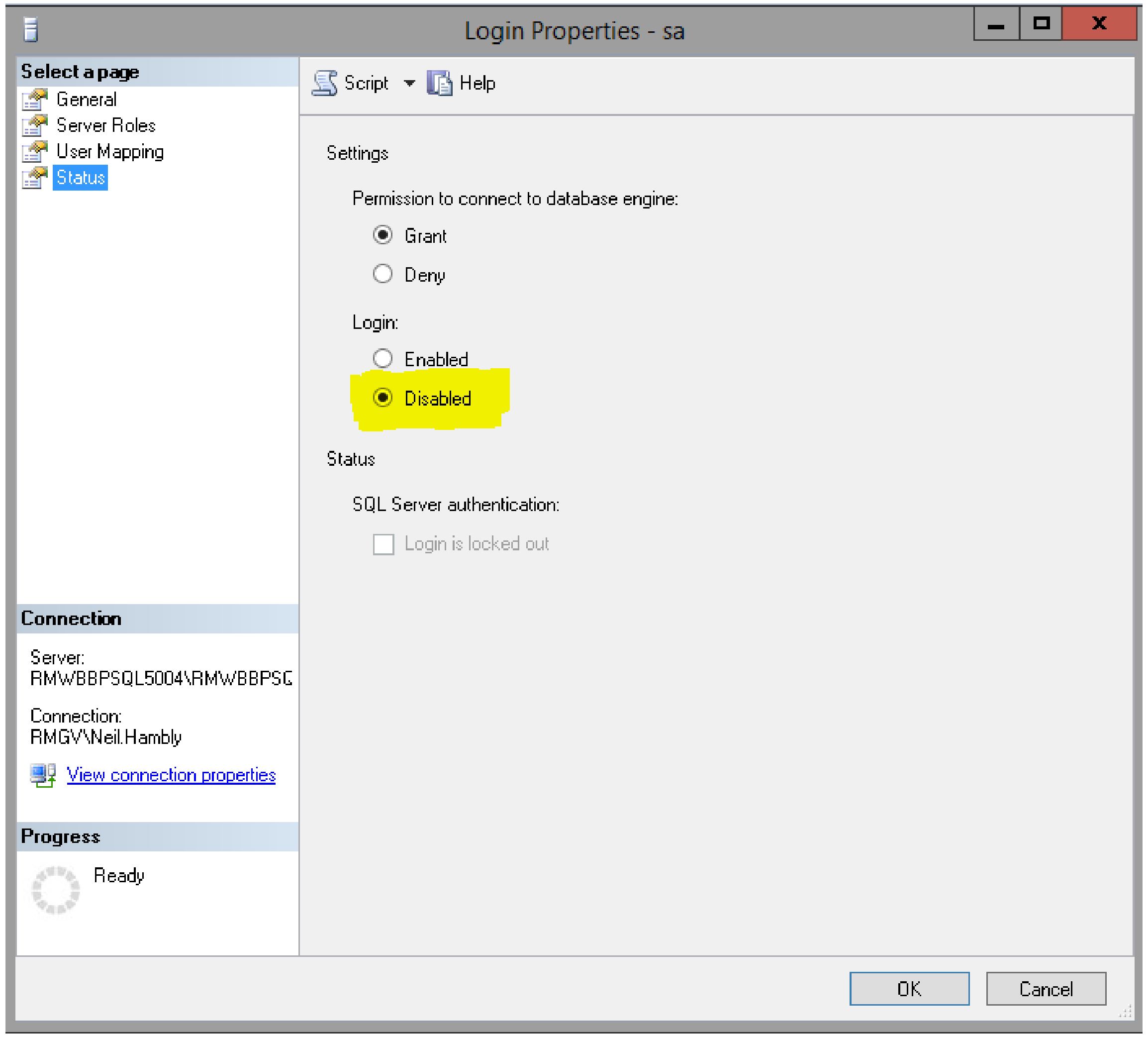
Tracking usage of ‘sa’ can then be achieved with **Failed Logins**



Log file summary: No filter applied

```
7/23/2015 6:40:10 PM   Logon    Login failed for user 'sa'. Reason: Could not find a login matching the name provided. [CLIENT: <local machine>]
7/23/2015 6:40:10 PM   Logon    Error: 18456, Severity: 14, State: 5.
7/23/2015 6:39:52 PM   spid52   Attempting to load library 'xplog70.dll' into memory. This is an informational message only. No user action is required.
```

Disabling the ’sa’ login


Login Properties - sa

Select a page
- General
- Server Roles
- User Mapping
- Status

Script | Help

Settings

Permission to connect to database engine:

- [✓ Grant]
- Deny

Login:

- Enabled
- [✓ Disabled]

Status

SQL Server authentication:

- [ ] Login is locked out

Connection

Server:
RMWBBPSQL5004\RMWBBPSC

Connection:
RMGV\Neil.Hambly

View connection properties

Progress

Ready

OK | Cancel

Disabling the ‘sa’ login is an option, this can be done in the Login Properties dialog page or via T-SQL

Easy Way

SQL Server has an *undocumented* system stored procedure named **sp_SetAutoSAPasswordAndDisable**.
Executing this stored procedure **resets** the password to a random GUID for the ‘sa' login and then disables it.
{Requires account executing this sproc to to be a sysadmin}

Permissions

Permissions are what governs everything you can or can’t do in SQL Server, understanding what are the ‘effective’ permissions are is useful and important.

But it can be confusing at times, with the complexity of all those permissions.

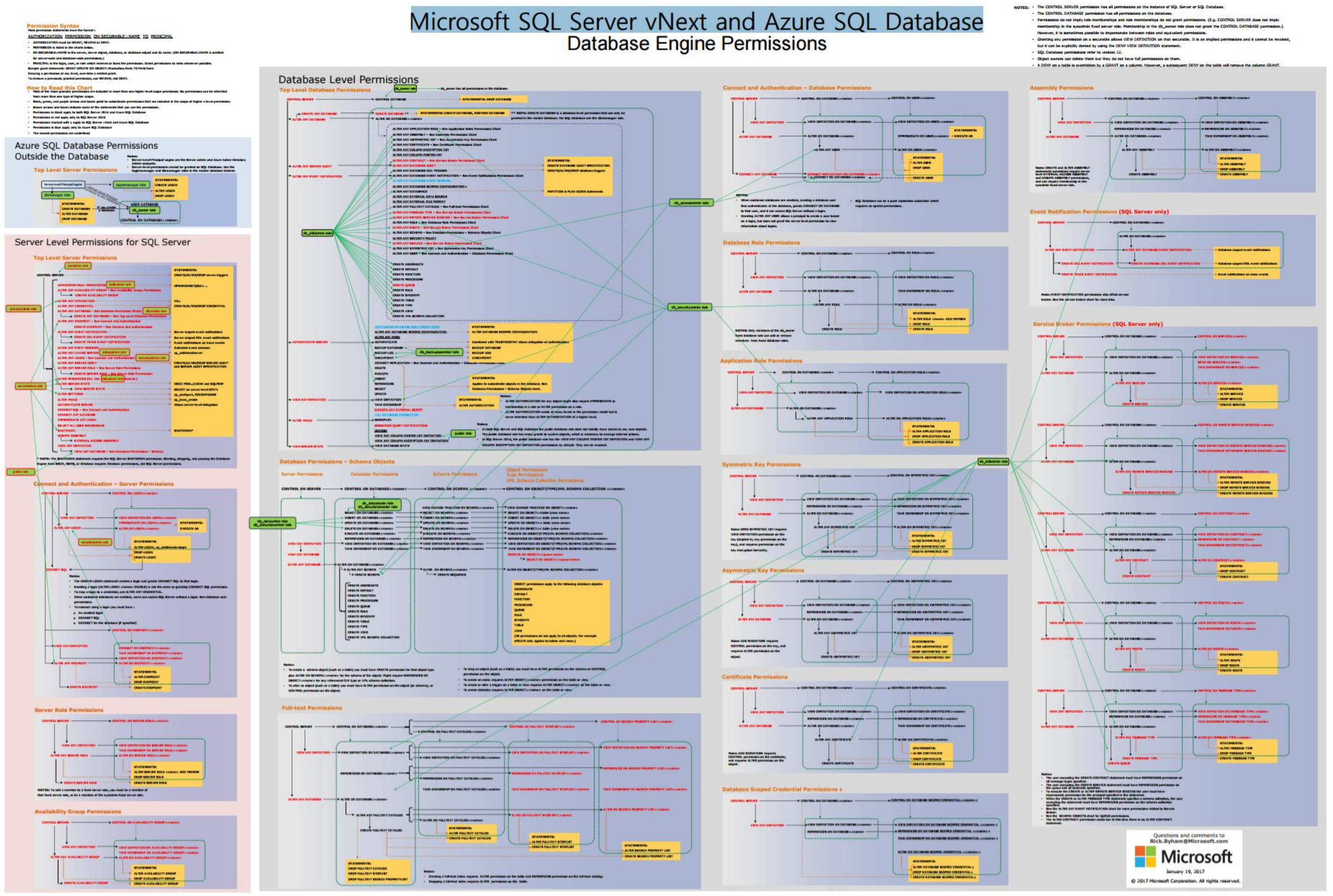
Your ‘effective’ permissions are the combined result of all the various permissions granted, or through a role or any revoked or deny permissions.

Visual guidance can be useful, well this Microsoft PDF guide sownload can help some in understanding the permissions and their hierarchies.

http://go.microsoft.com/fwlink/?LinkId=229142

Effective Permissions

https://docs.microsoft.com/en-us/sql/relational-databases/security/authentication-access/determining-effective-database-engine-permissions


Microsoft SQL Server vNext and Azure SQL Database
Database Engine Permissions
Permission Syntax
How to Read this Chart
Azure SQL Database Permissions Outside the Database
Top Level Server Permissions
Server Level Permissions for SQL Server
Connect and Authentication – Server Permissions
Server Role Permissions
Availability Group Permissions
Database Level Permissions
Top Level Database Permissions
Database Permissions – Schema Objects
Full-text Permissions
Connect and Authentication – Database Permissions
Database Role Permissions
Application Role Permissions
Symmetric Key Permissions
Asymmetric Key Permissions
Certificate Permissions
Database Scoped Credential Permissions
Assembly Permissions
Event Notification Permissions (SQL Server only)
Service Broker Permissions (SQL Server only)
NOTES:
Microsoft

Granting Database Permissions

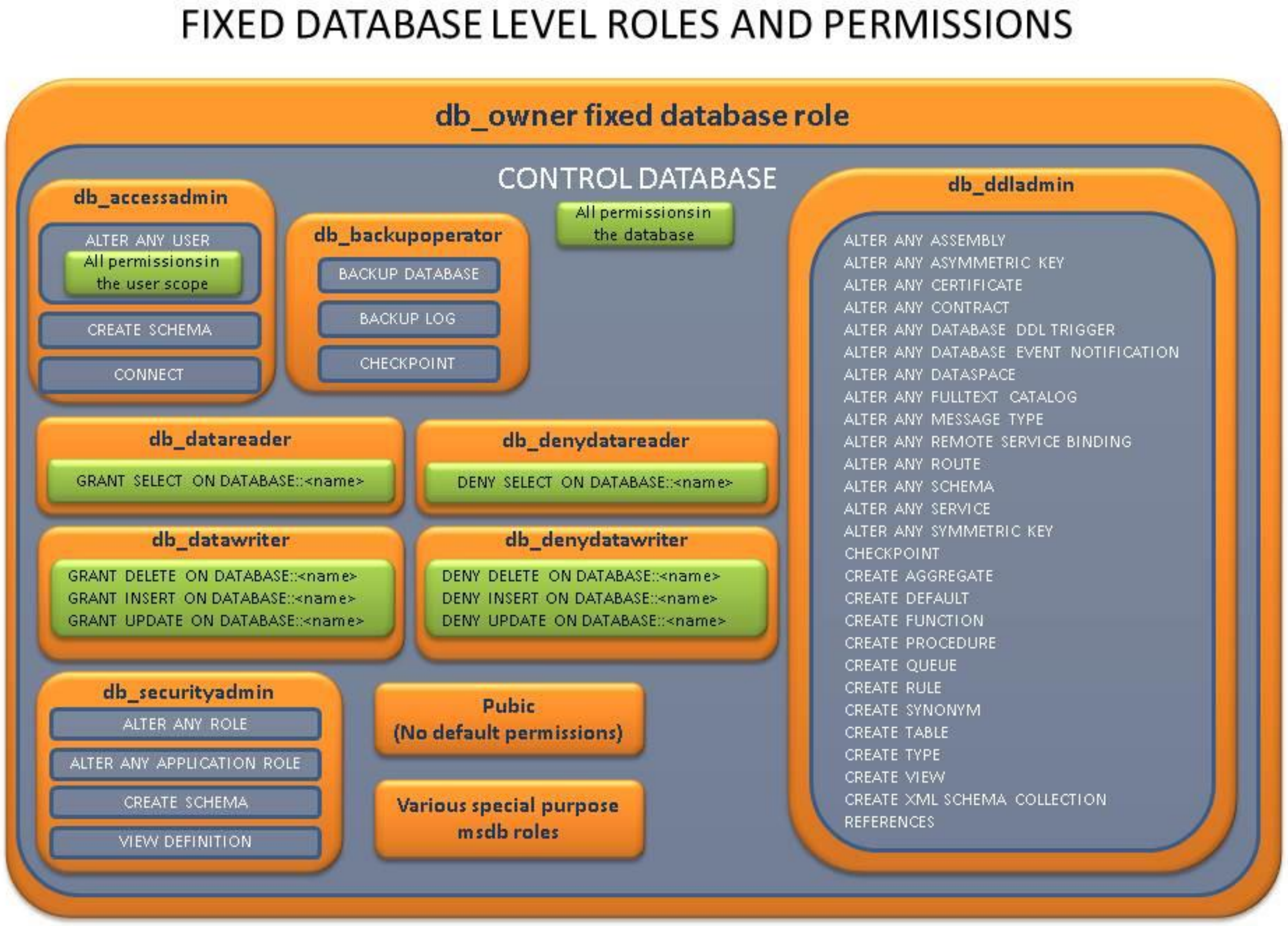
The grantor (or the principal specified with the AS option) must have the permission itself with GRANT OPTION, or a higher level permission
Grantees of CONTROL permission on a database, such as members of the **db_owner** fixed database role, can grant any permission on any securable in the database.
For a more detailed and expanded explanation of GRANT then reference https://docs.microsoft.com/en-us/sql/t-sql/statements/grant-transact-sql

: ALL
This option does not grant all possible permissions, using GRANT ALL is equivalent to granting the following permissions:
BACKUP DATABASE, BACKUP LOG,
CREATE DATABASE, CREATE DEFAULT, CREATE FUNCTION, CREATE PROCEDURE, CREATE RULE, CREATE TABLE, CREATE VIEW.

: AS
Specifies a principal from which the principal executing this query derives its right to grant the permission.

AS *granting_principal*	Additional permission required
Database user Database user mapped to a Windows User / Group Database user not mapped to any server principal	IMPERSONATE permission on the user, membership in the db_securityadmin fixed database role, membership in the db_owner fixed database role, or membership in the sysadmin fixed server role.
Database user mapped to a certificate or asymmetric key	Membership in the db_securityadmin fixed database role, membership in the db_owner fixed database role, or membership in the sysadmin fixed server role.
Database role or Application role	ALTER permission on the role, membership in the db_securityadminfixed database role, membership in the db_owner fixed database role, or membership in the sysadmin fixed server role.

Fixed Database Roles | User Created



```
-- Creating roles, adding members

CREATE ROLE Managers;
ALTER ROLE Sales ADD MEMBER NeilH;
ALTER ROLE Sales DROP MEMBER NeilH;
```

https://docs.microsoft.com/en-us/sql/relational-databases/security/authentication-access/credentials-database-engine

Database **User / Role / Application** Permissions

DATABASE USER | DATABASE ROLE | APPLICATION ROLE

A Database User / Database Role / Application Role is a **database-level securable** contained by the database that is its parent in the permissions hierarchy.
The most specific and limited permissions that can be granted on a **database** are listed in the following table, together with the more general permissions that include them by implication.

Role permission	Implied by role permission	Implied by database permission	Database User | Database Role | Application Role
CONTROL	CONTROL	CONTROL	Database User, Database Role, Application Role
IMPERSONATE	CONTROL	CONTROL	Database User
TAKE OWNERSHIP	CONTROL	CONTROL	Database Role
ALTER	CONTROL	ALTER ANY USER ALTER ANY ROLE ALTER ANY APPLICATION ROLE	Database User Database Role Application Role
VIEW DEFINITION	CONTROL	VIEW DEFINITION	Database User, Database Role, Application Role

```
-- Finding information on users, Example lists users with db_securityadmin role

EXEC sp_helpuser 'db_securityadmin';
GO
```

Database Permissions Query

```
-- The following query returns a list of the permissions that have been granted or denied at the server level.

USE Master;

SELECT      pr.type_desc,
            pr.name,
            isnull (pe.state_desc, 'No permission statements') AS state_desc,
            isnull (pe.permission_name, 'No permission statements') AS permission_name
FROM sys.server_principals AS pr
LEFT OUTER JOIN sys.server_permissions AS pe
            ON pr.principal_id = pe.grantee_principal_id
WHERE is_fixed_role = 0 -- Remove for SQL Server 2008
ORDER BY pr.name, type_desc;
```

Database Permissions Query

```
--  The following query returns a list of the permissions that have been granted or denied at the database level.

USE <Set Database>;

SELECT      pr.type_desc,
            pr.name,
            isnull (pe.state_desc, 'No permission statements') AS state_desc,
            isnull (pe.permission_name, 'No permission statements') AS permission_name
FROM sys.database_principals AS pr
LEFT OUTER JOIN sys.database_permissions AS pe
            ON pr.principal_id = pe.grantee_principal_id
WHERE pr.is_fixed_role = 0
ORDER BY pr.name, type_desc;
```

```
--  Use the HAS_PERMS_BY_NAME function to determine if a particular user (in this case UserY) has a permission.

EXECUTE AS USER = 'UserY';
SELECT HAS_PERMS_BY_NAME ('dbo.TableX', 'OBJECT', 'SELECT');
REVERT;
```

Database Ownership & ’sa’ account

The system databases (master, model, msdb, tempdb) will have ‘sa’ as it’s db_owner

Even if this has been renamed as internally it still exists (sid 0x01) and that is what is associated with the ownership, We just prefer to use the more friendly version to see what sid has ownership.

Renaming of the login just changes the label we have given it, the sid will be unchanged

Microsoft reference to SQL Server Principals

https://docs.microsoft.com/en-us/sql/relational-databases/security/authentication-access/principals-database-engine

```
--  Determine who is a member of a fixed database role, execute the following query in each database..

USE <Set Database>;

SELECT      DP1.name AS DatabaseRoleName,
            isnull (DP2.name, 'No members') AS DatabaseUserName
FROM sys.database_role_members AS DRM
RIGHT OUTER JOIN sys.database_principals AS DP1     ON DRM.role_principal_id = DP1.principal_id
LEFT OUTER JOIN    sys.database_principals AS DP2     ON DRM.member_principal_id = DP2.principal_id
WHERE DP1.is_fixed_role = 1
ORDER BY DP1.name;
```

Impersonation

What exactly is **IMPERSONATION** the demo used EXECUTE AS ?

Good Question, so glad you asked

Say I need to perform some action or investigate an issue but under a different user's account

So I'll need to be doing this in the context of that user's security but without having them doing all the login's etc..

Granting the IMPERSONATE right to a specific user to act as if they are a different user or perhaps an application.

I would then be able to execute statements as if I was that account or role, troubleshoot / investigate any issues with security, once done can simply revoke the IMPERSONATE rights.

```
-- Granting Impersonation rights to a user for a user database
-- on a specific application role

USE <Set Database>;

GRANT IMPERSONATE ON USER:: UserY TO UserX;
GO
```

Impersonation EXECUTE AS

By default, a session starts when a user logs in and ends when the user logs off.
All operations during a session are subject to permission checks against that user.

When an **EXECUTE AS** statement is run, the execution context of the session is switched to the specified login or user name.
After the context switch, permissions are checked against the login and user security tokens for that account instead of the person calling the **EXECUTE AS** statement.
In essence, the user or login account is **impersonated** for the duration of the session or module execution,
or the context switch is explicitly reverted.

```
-- Give IMPERSONATE permissions on UserY to UserX , UserX can successfully set the execution context to UserY.

GRANT IMPERSONATE ON USER:: UserY TO UserX;
--Display current execution context.
SELECT SUSER_NAME(), USER_NAME();
-- Set the execution context to loginX (mapped to userX).
EXECUTE AS LOGIN = 'loginX';
--Verify the execution context is now loginX.
SELECT SUSER_NAME(), USER_NAME();
REVERT;
```

https://docs.microsoft.com/en-us/sql/t-sql/statements/execute-as-clause-transact-sql

QUESTIONS?

THANK YOU
FOR ATTENDING

Microsoft redgate