IDERA® | Database Tools Whitepaper

NUTS AND BOLTS OF PERMISSIONS AND SECURITY

BY PINAL DAVE

CONTENTS

INTRODUCTION

Being in the computer industry is one of the most challenging things anyone can get into. When upgrading one's knowledge to keep the data secure, there are many hurdles to cross. In this industry, where we look for quick, fast responses from development to deployment to sales, everyone is expected to deliver without compromising any of the business parameters. As we chase around the strict deadlines for delivery, the team tries to take shortcuts. That, for most cases, involves cutting corners and compromising on designing for secure systems. That is a concern for many organizations. Database administrators love to build a secure system to deploy applications delivered from the Application team. In this whitepaper, I would like to get back to the basics of security. We want to cover some of the fundamental building blocks that were added with SQL Server since the 2005 version that is worth noting till today in SQL Server 2019.

PRINCIPALS

Let us discuss one of the most important keywords, which often confuses people – Principals. Principals are entities that can request SQL Server resources. In SQL Server, all login and database user accounts as Principals. That is consistent with the Windows concept of a security principal. A security principal is an entity that can be identified and verified via authentication. The concept has not changed much from earlier versions of SQL Server. It is more of a terminology change.

The capabilities of principals depend on their scope of definition (server or database) and whether they represent a single principal (Primary) or a collection of principals (Secondary). Each Principal is uniquely identified by a security identifier.

Server-level primary principals

Dynamic management views are the heart of finding what type of login is currently being used. At a high-level, to learn about the Primary Principals, we can use the following dynamic management view query

```
SELECT *
FROM master.sys.server_principals;
```

Editor | Results | Messages

	name	principal_id	sid	type	type_desc	is_disabled
1	sa	1	0x01	S	SQL_LOGIN	0
2	public	2	0x02	R	SERVER_ROLE	0
3	sysadmin	3	0x03	R	SERVER_ROLE	0
4	securityadmin	4	0x04	R	SERVER_ROLE	0
5	serveradmin	5	0x05	R	SERVER_ROLE	0
6	setupadmin	6	0x06	R	SERVER_ROLE	0
7	processadmin	7	0x07	R	SERVER_ROLE	0
8	diskadmin	8	0x08	R	SERVER_ROLE	0
9	dbcreator	9	0x09	R	SERVER_ROLE	0
10	bulkadmin	10	0x0A	R	SERVER_ROLE	0
11	##MS_SQLResourceSigningCertificate##	101	0x0106...	C	CERTIFICATE_MAPPED_LOGIN	0
12	##MS_SQLReplicationSigningCertificate##	102	0x0106...	C	CERTIFICATE_MAPPED_LOGIN	0
13	##MS_SQLAuthenticatorCertificate##	103	0x0106...	C	CERTIFICATE_MAPPED_LOGIN	0
14	##MS_PolicySigningCertificate##	105	0x0106...	C	CERTIFICATE_MAPPED_LOGIN	0
15	##MS_SmoExtendedSigningCertificate##	106	0x0106...	C	CERTIFICATE_MAPPED_LOGIN	0
16	##MS_PolicyTsqlExecutionLogin##	257	0x2757...	S	SQL_LOGIN	1
17	QUICK\Pinal	259	0x0105...	U	WINDOWS_LOGIN	0
18	NT SERVICE\SQLWriter	260	0x0106...	U	WINDOWS_LOGIN	0
19	NT SERVICE\Winmgmt	261	0x0106...	U	WINDOWS_LOGIN	0
20	NT Service\MSSQL$SQL19	262	0x0106...	U	WINDOWS_LOGIN	0
21	NT AUTHORITY\SYSTEM	263	0x0101...	U	WINDOWS_LOGIN	0
22	NT SERVICE\SQLAgent$SQL19	264	0x0106...	U	WINDOWS_LOGIN	0
23	NT SERVICE\SQLTELEMETRY$SQL19	265	0x0106...	U	WINDOWS_LOGIN	0
24	##MS_PolicyEventProcessingLogin##	274	0x1546...	S	SQL_LOGIN	1
25	##MS_AgentSigningCertificate##	275	0x0106...	C	CERTIFICATE_MAPPED_LOGIN	0

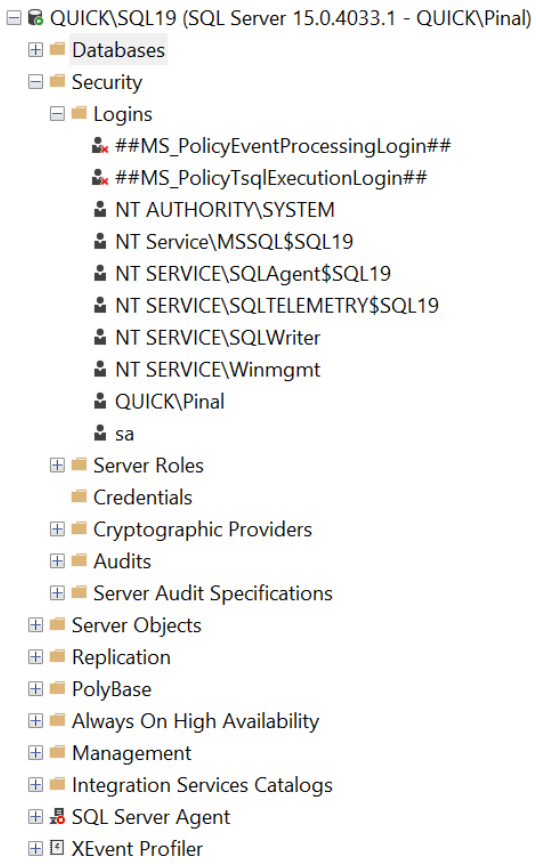
Windows login

Windows logins are defined in the local machine or the Active Directory forest (called a domain login) of which the SQL server instance is a part of. The management of these principals is not within the scope of the SQL Server. Each Windows login is uniquely represented with a security identifier, provided by Windows itself. These can be obtained using the following query:

```
SELECT *
FROM master.sys.server_principals
WHERE type = 'U'
```

	name	principal_id	sid	type	type_desc	is_disabled
1	QUICK\Pinal	259	0x01050000000...	U	WINDOWS_LOGIN	0
2	NT SERVICE\SQLWriter	260	0x01060000000...	U	WINDOWS_LOGIN	0
3	NT SERVICE\Winmgmt	261	0x01060000000...	U	WINDOWS_LOGIN	0
4	NT Service\MSSQL$SQL19	262	0x01060000000...	U	WINDOWS_LOGIN	0
5	NT AUTHORITY\SYSTEM	263	0x01010000000...	U	WINDOWS_LOGIN	0
6	NT SERVICE\SQLAgent$SQL19	264	0x01060000000...	U	WINDOWS_LOGIN	0
7	NT SERVICE\SQLTELEMETRY$SQL19	265	0x01060000000...	U	WINDOWS_LOGIN	0

You can also see all the logins in the SQL Server Management Studio right under Security >> Logins Folder. In the image below, we can also see the SQL Logins which we will discuss next.



SQL login

SQL logins are defined in a particular SQL Server instance. They are undefined outside the scope of that SQL Server. They are used for authorization of resources within that server. However, they are visible across all databases in the server. They can be used for authorization of resources in any database in the server. As with Windows logins, each SQL login is uniquely represented by a security identifier (which in this case is a globally unique identifier as it has no Windows-provided security identifier).

```
SELECT *
FROM master.sys.server_principals
WHERE type = 'S'
```

	name	principal_id	sid	type	type_desc	is_disabled
1	sa	1	0x01	S	SQL_LOGIN	0
2	##MS_PolicyTsqlExecutionLogin##	257	0x27578D...	S	SQL_LOGIN	1
3	##MS_PolicyEventProcessingLogin##	274	0x1546A0...	S	SQL_LOGIN	1

Server-level secondary principals

As we discuss the fine print, these are two kinds of Server level secondary principals as described below.

Windows groups

Windows groups are again defined in the local machine or the Active Directory forest (called domain login), of which the SQL Server instance is a part. The management of these principals is not within the scope of the SQL Server. Based on the nature of the domain, there can be different types of these. A security identifier uniquely represents each Windows group. We can query them using:

```
SELECT *
FROM master.sys.server_principals
WHERE type = 'G'
```

SQL role

SQL roles are specifically more widely known as fixed server roles. They are defined in each SQL Server instance, and membership is undefined outside the scope of that SQL Server. They are used for authorization of resources within that server. As with Windows logins, each SQL role is uniquely represented by a security identifier (which in this case is a globally unique identifier as it has no Windows-provided security identifier).

```
SELECT *
FROM master.sys.server_principals
WHERE type = 'R'
```

Database-level primary principals

Database user

Any server-level principal which requires access to a database needs to be mapped to a Database User in the relevant database. A Database User is a principal at the database level. Every database user is a member of the public role. SQL Server includes a guest database user account that has limited access rights. With the secure by default strategy, the latest versions of SQL Server keep this account in a disabled status.

```
SELECT *
FROM [DatabaseName].sys.database_principals
WHERE type in ('S','U','C')
```

	name	principal_id	type	type_desc	default_schema_name
1	dbo	1	S	SQL_USER	dbo
2	guest	2	S	SQL_USER	guest
3	INFORMATION_SCHEMA	3	S	SQL_USER	NULL
4	sys	4	S	SQL_USER	NULL

Application role

Application roles are database scoped principals that are like users but have no mapped login. They are restricted to a database and are identified by a password.

```
SELECT *
FROM [DatabaseName].sys.database_principals
WHERE type = 'A'
```

We also have something called a CONTAINED DATABASE USER, which is worth a mention here. We are keeping it out of the scope of this document because it is all by itself a larger topic to discuss.

Database-level secondary principals

Database role

Database roles are an aggregation mechanism for collections of database users. Since they are defined within the database, they cease to exist outside the database. Members of the DB_OWNER and DB_SECURITYADMIN fixed database roles can manage fixed database role membership. Every database user is a member of the public database role. When a user has not been granted or denied permissions on a securable, it inherits the permissions granted to the public on that securable.

```
SELECT *
FROM [DatabaseName].sys.database_principals
WHERE type = 'R'
```

LOGINS AND USERS

As we have discussed, the concept of a login as an object has been replaced by the principal object. However, we still have to be able to provide the ability for SQL Server to recognize a principal and authorize it for login access to SQL Server and its databases. To do this, we create a login in SQL Server for the relevant principal, and we create a user in each required database and map it to the principal.

When we create a login, we are adding an entry into the SYS.SERVER_PRINCIPALS view in master in much the same way as used to occur with the sysxlogins system table in earlier versions of SQL. There is a new syntax, CREATE LOGIN, to replace the previous SP_ADDLOGIN procedure.

When we create a database user, we are adding an entry into the SYS.DATABASE_PRINCIPALS view in a specific database, in much the same way as used to occur with the sysusers table in earlier versions of SQL Server. One of these views exists in each database, as database users are scoped at the database level. There is a new syntax, CREATE USER, to replace the previous SP_ADDUSER procedure.

It is essential to be aware of several new features, though, such as the ability to map a SQL Login to a Certificate or Asymmetric key. Additionally, it is also possible to assign credentials to a SQL login. That is used for the mapping of a SQL login to external credentials such as a Windows login for authentication and authorization purposes when accessing external resources. It is also now possible to ENABLE and DISABLE a login.

Password policy

There are also many new options for SQL logins, which are worth pointing out, CHECK_EXPIRATION and CHECK_POLICY being two of those. These specify whether a password expiration policy should be enforced on this login. They also specify also whether the Windows password policies of the computer on which SQL Server is running should be enforced on this login.

To strengthen the security of SQL Login accounts, since SQL Server 2005, we can enforce strong passwords, account lockout, password age mechanisms, and enforcement of domain or local level password policy.

Group Policy security settings can be examined by running GPEDIT.MSC, and expanding Computer Configuration ➔ Windows Settings ➔ Security Settings ➔ Account Policies ➔ Password Policy. If I do this on my machine, I see the following settings (your values may have different settings):

Enforce password history	Eight passwords remembered
Maximum password age	90 days
Minimum password age	One day
Minimum password length	Eight characters
Password must meet complexity requirements	Enabled
Store password using reversible encryption for all users in the domain	Enabled

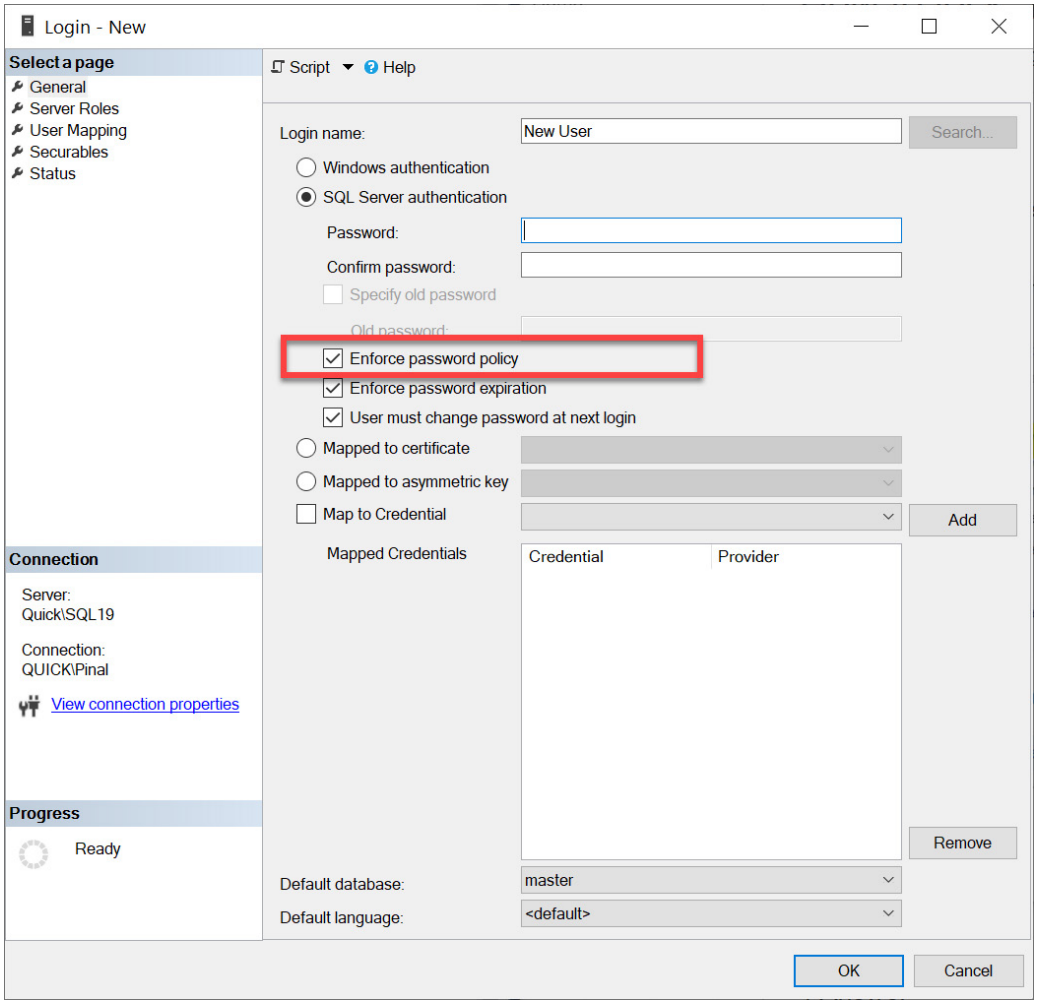
Local security policy for a non-domain computer can be examined using the Local Security Settings tool under Administrative Tools.

It is these settings that Password Policy is enforcing for SQL logins. Of course, Windows logins already have this enforced by the system, so in effect, we are only bringing SQL in line with established best practice. Note that Password Policy does not apply to Application Roles in this release. CHECK_POLICY is enabled by default, although CHECK_EXPIRATION is not.

- CHECK_EXPIRATION enforces minimum and maximum password age.
- CHECK_POLICY enforces all other policies.

When you run afoul of any of the policy settings, your account is locked. It must be unlocked by a Sysadmin, using the UNLOCK option of ALTER LOGIN.

In simple words, when the user enables the Enforce password policy option in the SQL Server Login window, it instructs the SQL Server to adhere either to the local security policy or to the policy defined in the domain.



Password policy troubleshooting

Most of the error messages that cause login failures due to Password Policy enforcement are fairly clear, however, there may be circumstances under which the error is not correctly propagated to the user, due to legacy applications, poor error handling, etc. In these cases, the EventLog and Errorlog should contain the relevant message.

SQL logins may expire after the Maximum Password age. To address this and identify accounts which are affected by the policy, a Sysadmin can run the following script to provide the current state of SQL logins:

```
DECLARE @max_age int
SET @max_age = 90
SELECT name,
 is_policy_checked,
 is_expiration_checked,
 LOGINPROPERTY(name, 'IsExpired') AS is_expired,
 LOGINPROPERTY(name, 'IsLocked') AS is_locked,
 LOGINPROPERTY(name, 'IsMustChange') AS is_mustchange,
 LOGINPROPERTY(name, 'HistoryLength') AS history_length,
 LOGINPROPERTY(name, 'BadPasswordCount') AS bad_password_count,
 DATEDIFF(day, GETDATE(), DATEADD(day, @max_age,
 CAST(LOGINPROPERTY(name, 'PasswordLastSetTime')
 AS datetime))) AS days_until_expiration,
 LOGINPROPERTY(name, 'PasswordLastSetTime') AS
 password_last_set_time,
 LOGINPROPERTY(name, 'BadPasswordTime') AS bad_password_time,
 LOGINPROPERTY(name, 'LockoutTime') AS lockout_time
FROM sys.sql_logins
```

At every organization where I go for SQL Server Performance Tuning consultancy, I always recommend them to run the above script as my clients must know where they are standing with regards to SQL Server security. The logins are the most vulnerable area of any server.

Note that we are required to set the MAX_AGE variable manually. That should be set to the value for maximum password age as per the Global Policy/Local Policy currently in force on the SQL Server machine, which can be obtained via GPEDIT.MSC or Local Security settings.

To unlock an account, a Sysadmin should run the following statement to unlock the affected user account:

```
ALTER LOGIN Pinal WITH PASSWORD = 'password' UNLOCK
GO
```

SPECIAL USER SYSADMIN

If SQL is installed in Windows Authentication mode, we assign a random password to the system administrator account and disable it by default. That is to prevent a situation where we have a system with an active system administrator account with a blank password. If the authentication mode is changed, the system administrator account needs to be re-enabled, and a password must be assigned to it.

```
ALTER LOGIN sa ENABLE;
ALTER LOGIN sa WITH PASSWORD = 'myC0mplexPa$$w0rd';
```

To do this, you must be a member of the SYSADMIN role. If you have dropped the BUILTIN\Administrators group from the SYSADMIN role and have no other Sysadmins other than system administrator, then there is an obvious problem. To overcome this, it should be possible to connect via a dedicated admin connection as a member of the server group BUILTIN\Administrators (using Windows Authentication) and add a SYSADMIN account or to re-enable the system administrator account and reset its password. Any other action that this Admin attempts inside SQL Server will work only if that Admin account has been granted access to perform that action.

I suggest that you keep the system administrator account disabled on your production environment and also do not share the password of it with anyone.

CONCLUSION

Security is a core area and non-negotiable when it comes to mission-critical applications. As DBAs, knowing the fine print of working with security is very important. In this whitepaper, we got a chance to explore and learn some of these basics so that our foundation is strong. I highly recommend we keep our eyes open on our deployment server to confirm our environments are configured correctly. Look for possible security loopholes and try to plug them before they become a compliance issue.

ABOUT THE AUTHOR

Pinal Dave is a SQL Server performance tuning expert and an independent consultant. He has authored twelve SQL Server database books, and 33 Pluralsight courses. Pinal has written over 5100 articles on the database technology on his blog at a https://blog.sqlauthority.com. Along with more than 17 years of hands-on experience, he holds a Master's of Science degree and some database certifications.

IDERA'S SOLUTION

SQL Secure

Manage SQL Server Security & Permissions

IDERA SQL Secure discovers security vulnerabilities and user permissions for SQL Server instances deployed on physical, cloud, and virtual hosts. Find out who has access to what and identify each user's effective rights across all SQL Server and Azure SQL Database objects. Alert on violations of your corporate policies, monitor changes made to security settings, and generate security audit reports as well as recommendations on how to improve your security model.

Start a free, fully-functional, 14-day trial today!

Start for FREE