Packaging Permissions in Stored Procedures

Erland Sommarskog

SQL Server MVP

Erland Sommarskog

Independent consultant based in Stockholm

SQL Server MVP since 2001

http://www.sommarskog.se

esquel@sommarskog.se

Slides and scripts are available on

http://www.sommarskog.se/present

and on the SQL Bits web site.

What This is All About

We want a user to be able perform a specific action that requires permission X.

But we don't want to grant the user that permission directly. (Because that would permit the user to do a lot more.)

What if we could package the permission inside a stored procedure with full control of what the user can do?

"*Hey, isn't that how it always have worked?*"

No, only in a special but common case.

In this session we will learn how to do it in the other cases.

Agenda

- **Ownership Chaining**
- **Certificate Signing**
- The EXECUTE AS Clause

Ownership Chaining

When a stored procedure (or a view, function or trigger) accesses an object, with *the same owner*, permissions are not checked for:

- DML (SELECT, INSERT, UPDATE, DELETE & MERGE).
- Execution of stored procedures and functions.

Does not apply to:

- Access through dynamic SQL.
- Access to metadata about the object.
- Special permissions such as ALTER.

01_ownershipcain.sql

Certificate Signing

What you use when ownership chaining does not apply.

The recipe:

1. Create a certificate.
2. Sign the procedure with the certificate.
3. Create a user from the certificate.
4. Grant the certificate user the permissions needed (which could be role membership).

Certificates and Signatures

A certificate is an asymmetric key that consists of:

- A private key that you keep secret and protect.
- A public key that you can share with anyone.
- Some metadata, including a signature. For our purposes, self-signed certificates are sufficient.

You can sign a document (email etc) with your private key.

The receiver can use the public key to verify the signature.

Proves that document is from you and has not been altered.

What is this User?

A special type of user that exists to connect permissions and certificate. It cannot log in or be impersonated.

You can only create one user per certificate.

When the procedure has a valid signature, the token of the certificate user is added to sys.user_token, just like a role.

Net effect: the permissions of the certificate user are added to the permissions of the current user.

02_certsigndb.sql

Observations on Certificate Signing

Procedure must be signed after each change.

The token and thus the permissions of the certificate user are carried on to dynamic SQL and system procedures.

But they are **not** carried on to nested procedures, triggers or functions.

How to Use This?

A few certificates with common permissions attached to them lying around?

Or one certificate per procedure with exactly the permissions needed?

The latter is better for both security and manageability.

Password nightmare? No, just throw the passwords away!

GrantPermsToSP

- Parameters: procedure name and a TVP with permissions.
- Drops any existing signature, certificate and user.
- Creates a new certificate with a random password.
- Signs the procedure.
- Creates a user and grants it permissions.

```
DECLARE @perms Permission_list
INSERT @perms (perm) VALUES ('SELECT ON PermTable')
EXEC GrantPermsToSP N'TestSP', @perms, @debug = 1
```

In many cases you put this in your deployment scripts.

04_procsignscript.sql

Server-Level Permissions

A scenario:

- A multi-application instance.
- For each database there are power users with db_owner permissions, but no server permissions.
- They need to see which users that are connected to their database.
- This requires VIEW SERVER STATE – but with that permission they would see too much.
- Certificates to the rescue!

Server-Level Recipe

1. Create a certificate in the master database.
2. Sign the procedure with certificate (if in master).
3. Create a login from that certificate.
4. Grant the login the required permissions.

While called “login”, this login cannot log in – it exists only to connect permissions and certificate.

Tokens can be inspected in sys.login_token.

05_certsignserver.sql

Server-Level Permission in User DB

1. Create a certificate in the master database.
2. Create a login from the certificate.
3. Grant the login the required permissions.
4. Copy certificate to user database.
5. Sign the procedure.
6. Drop the private key.

Last step ensures that local power users cannot use the certificate, even if they would know the password.

Copy Certificate

On SQL 2012 and up:
Get keys with certencoded() and certprivatekey(). Then run CREATE CERTIFICATE FROM BINARY in user DB.

In SQL 2005/2008:
Bounce the cert over disk with BACKUP CERTIFICATE and CREATE CERTIFICATE FROM FILE.
Delete file when done.

06_exportimport.sql

What About Availability Groups?

In an AG, certificate, login and permissions must exist on all nodes in the AG, so that things can work after a failover.

Big hassle? Don't worry, I have a script for you that:

- Creates cert and login and grants permissions in master.
- Copies the cert to the user database and signs the procedure.
- For AGs: Loops over the other nodes in the AG, using a temporary linked server to copy cert, login and permissions.
- You must specify: database, procedure and permissions.

07_procsignscript_server.sql

The Beauty of it All

The server-level DBA reviews the code before signing it, thereby adding the extra permissions.

If the power user changes the code, signature and permissions disappear.

Thus, DBA must sign again – and can review the changes.

That is, power users cannot use this to elevate their permissions behind the back of the DBA.

WITH EXECUTE AS

A popular alternative is to say something like this:

```
CREATE USER TestSP$Proxy WITHOUT LOGIN
GRANT SELECT ON PermTable TO TestSP$Proxy
go
CREATE PROCEDURE TestSP @id int, @title varchar(40)
  WITH EXECUTE AS 'TestSP$Proxy' AS
```

You grant the proxy user the permissions needed for the SP.

TestSP will run as if it had been started by TestSP$Proxy, i.e. impersonation.

WITH EXECUTE AS OWNER

Many skip the proxy user and just say:

```
CREATE PROCEDURE TestSP @id int, @title varchar(40)
  WITH EXECUTE AS OWNER AS
```

OWNER = typically dbo. That is, the procedure runs with the powers of dbo and can do anything in the database.

Violates the principle of granting minimum permissions.

Really bad if there is an SQL injection hole.

EXECUTE AS pitfalls

USER, SYSTEM_USER etc returns the name of the impersonated user. Exception: original_login().

Impersonation extends into everything you call (triggers, views etc).

Auditing, row-level security etc may not work as intended.

Need to use original_login() or one of session_context() / context_info(). Requires that you plan ahead.

If system is not ready for EXECUTE AS, use DDL trigger.

09_stopexecas.sql

EXECUTE AS for Server-Level

WITH EXECUTE AS does not work for server-level permissions!

EXCEUTE AS in an SP impersonates a *database user*.

You are sandboxed into the current database and cannot access things outside of it.

(Else someone who is db_owner could impersonate someone who is sysadmin and elevate their permissions.)

What About TRUSTWORTHY?

Yes, if database is TRUSTWORTHY **and** the *database owner* has the permission AUTHENTICATE SERVER, EXECUTE AS works for server permissions.

Big security risk!

A user with only db_owner rights can impersonate someone with sysadmin permissions and take over the server.

Recap: Ownership Chaining

What you use 95% of the time, for plain and simple DML in stored procedures.

- Does not work with dynamic SQL.
- Does not work with “advanced” permissions.
- Does not work with metadata.
- Does not work with server-level permissions.

Recap: Certificate Signing

Permits you to package about any database or server permission in stored procedures in a fine-grained way.

Easy to manage with GrantPermsToSP and the script for server-level permissions.

The preferred method for database permissions.

Always use certificates for server-level permissions!

It's Getting Very Near the End...

Erland Sommarskog – esquel@sommarskog.se

Scripts and slides on http://www.sommarskog.se/present
and on the SQL Bits site.

Packaging Permissions in Stored Procedures on the web:
http://www.sommarskog.se/grantperm.html
http://www.sommarskog.se/grantperm-appendix.html

Clean-up script

13_cleanupall.sql