

Giving Permissions through Stored Procedures

Erland Sommarskog
SQL Server MVP



A photograph of a dirt path winding through a lush green forest. The path is covered in fallen leaves and ferns. On the left side of the path, there is a small blue and white signpost. The trees are tall and dense, with sunlight filtering through the canopy.

Erland Sommarskog

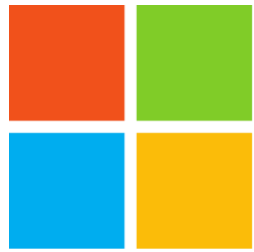
Independent consultant based in Stockholm

SQL Server MVP since 2001

<http://www.sommarskog.se>

esquel@sommarskog.se

Primary sponsors



Microsoft



Informator

A SOPRANO COMPANY



The others...



Giving Permissions through Stored Procedures

If a user has permission to run a stored procedure, he does not need permission for any action performed inside the stored procedure – True or False?

True in some common cases –

but the general answer is **False**.

In this session we will learn how we can make the statement become True in general.

Agenda

- Ownership Chaining
- Certificate Signing
- The EXECUTE AS Clause
- The Dangers of TRUSTWORTHY

Slides and all scripts are available on <http://www.sommarskog.se/present>

Ownership Chaining

No permissions are checked when a module (procedure, function, view or trigger) accesses an object with the same owner. Applies to:

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

Does not apply to:

- Dynamic SQL.
- Metadata access.
- Special permissions such as ALTER.

[01_ownershipcain.sql](#)

Certificate Signing

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).

When something is signed with a certificate, this permits you to verify that the contents is unchanged using only the public key.



What is this User?

This is a special type of user that exists only to connect permissions and certificate. It cannot log in or execute.

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 user's own permissions.

[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.

Keep in mind: DENY always trumps GRANT.

Certificate Management

Have a few certificates tied to “standard” permissions? Possible; a procedure can be signed with multiple certs.

But better is to have one certificate per procedure, granted exactly the permissions needed. This is easier – and more secure.

You can have a special stored procedure that performs the signing. If developers are trusted, this procedure can be called from the deployment script for the stored procedures.

The password? Throw it away!

[04_procsignscript.sql](#)



Server-Level Permissions

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. Export certificate.
5. Move to user database.
6. Import certificate.
7. Sign the procedure.
8. Optional: drop private key.

Export/Import Certificate

In all versions (from SQL 2005 and up):

- Export: **BACKUP CERTIFICATE** (to disk).
- Import: **CREATE CERTIFICATE FROM FILE** (and delete the file).

In SQL 2012 and later:

- Export: **certencodetd()** and **certprivatekey()**.
- Import: **CREATE CERTIFICATE FROM BINARY**.

[06_exportimport.sql](#)

[07_procsignscript_server.sql](#)

EXECUTE AS and DB Permissions

Proper version:

1. Create a proxy user `WITHOUT LOGIN` with the name derived from the procedure.
2. Grant the proxy user the required permissions.
3. Add the clause `WITH EXECUTE AS 'SPName$Proxy'`.

User creation and granting are in the same file as the procedure.

Lazy version:

1. Use `WITH EXECUTE AS OWNER` and no proxy user.

[08_executeasdb.sql](#)

EXECUTE AS, cont'd

A lot simpler than certificates. ...but!

- Lazy version breaks the principle of granting minimum permissions.
- Breaks schemes for row-level security and auditing based on `SYSTEM_USER`, `USER` etc.
- This can be mitigated by using `original_login()` or `context_info/session_context` – requires planning ahead.
- If your system is not ready for `EXECUTE AS` – you can stop it with a DDL trigger.

[09_stopexecas.sql](#)

Server-Level Permissions and EXEC AS

Create a proxy login, grant permissions, add EXECUTE AS clause?

[10_execasserver.sql](#)

When impersonating a **user**, we are sandboxed into the current database and cannot access things outside it, unless two doors are opened:

1. The database owner must have been granted the permission **AUTHENTICATE SERVER**. (Which is the case if owner = sa.)
2. The database must be set **TRUSTWORTHY**.

[11_trustworthy1.sql](#)

TRUSTWORTHY is a Security Risk

With certificate signing, the DBA can request to review the code every time an application admin wants to change an SP with server-level access.

EXECUTE AS + TRUSTWORTHY gives the application admin carte blanche to change the SP to his/her own liking.

But that is not all. Danger alert!

[12_trustworthy2.sql](#)

Combined with AUTHENTICATE SERVER, a person with **db_owner** rights (or rights to create and impersonate users) can elevate to **sysadmin**.

TRUSTWORTHY, cont'd

Recommendation: the owner of each database should be a unique SQL login which exists solely to own that database.

Hopefully, you think twice before you grant that login `AUTHENTICATE SERVER` to open the second door.

It is OK to open both doors, IF everyone with permission to create users in the database also are sysadmin.

...and this will never change. (Think: consultants.)



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 grant about any database or server permission through stored procedures in a fine-grained way. (But cannot overcome an explicit DENY.)

Seems to be a bit of hassle at first, but with organisation and throw-away passwords it's not a big deal.

The preferred method for database permissions.

Always use certificates for server-level permissions!



Recap: EXECUTE AS

Good for the lazy and casual. :-)

Implications for auditing and row-level security.

Requires that you plan ahead.

Can be OK for database permissions if conditions are relaxed (or you need to overcome DENY).

Always use certificate signing 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>

Giving Permissions through Stored Procedures on the web:
<http://www.sommarskog.se/grantperm.html>

...and beware of TRUSTWORTHY!

[13_cleanupall.sql](#)



Don't miss

- **Evaluations**
- **The Raffle**