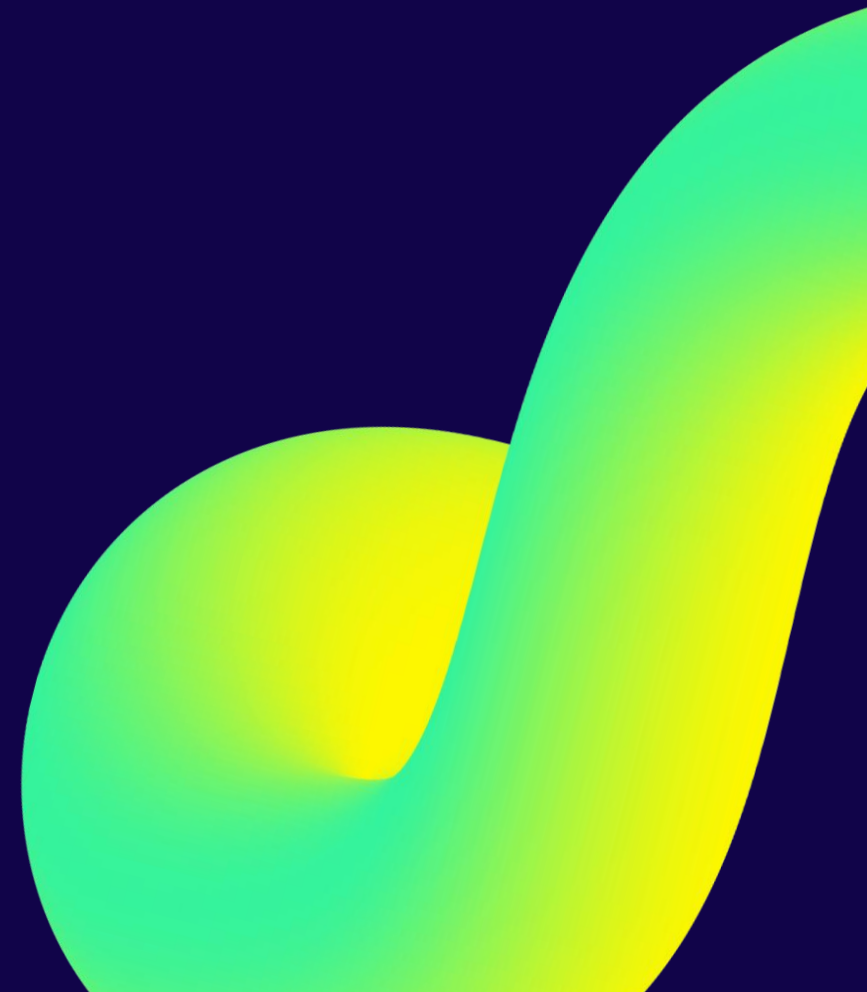


Don't Let Your Permissions Be **HIJACKED!**

Erland Sommarskog

Data Platform MVP



Erland Sommarskog

SQL Server MVP since 2001

Independent consultant based in Stockholm



esquel@sommarskog.se



<https://www.sommarskog.se>

Slides and scripts: <https://www.sommarskog.se/present>



What This is About

Permission Hijacking:

A person with less permissions than you have makes you unknowingly run code that (ab)uses your permissions to perform actions that are advantageous to the attacker.

Preamble: Isn't This Too Fantastic?

- When it comes to security, nothing is “too unlikely”.
- If an attack can be performed, it will be attempted sooner or later, some place.
 - You don't want that place be your workplace.
- It all depends on the stakes.
 - Perform advantageous updates.
 - Steal data.
 - Install ransomware.
 - Just cause a mess (e.g., a disgruntled employee).

Preamble: Not All Sites Are Equal

- In some places: only sysadmin and plain users and nothing in between.
 - Not very susceptible to the attacks I will discuss. (But see below.)
- Elsewhere: sysadmin, db_owner (application admin), db_ddladmin or similar (devs).
 - This is where you need to watch out!
- Don't overlook application logins.
 - They could have elevated permissions and SQL-injection holes.
- What about developers whose code gets deployed on the server?

The Evil DDL Trigger

`sysadmin.sql`

- An application admin could have installed a DDL trigger that performs malicious actions abusing sysadmin permissions when index and stats maintenance runs.
 - Adding a user to sysadmin is just one example.
 - It could be data theft, data manipulation etc.
- Line of defence: PoLP, [Principle of Least Privilege](#).
- In this case: `EXECUTE AS USER = 'dbo'`.
 - This way you sandbox yourself into the current database and renounce all your permissions on server level.

EXECUTE AS Commands

- EXECUTE AS USER: Run commands as a database user.
 - Sandboxed into the current database.
 - No server-level access or access to other databases.
 - Can still use temp tables.
- EXECUTE AS LOGIN: Run commands as a server login.
 - Sandboxed inside the instance.
 - Can access any database and server-level objects.
 - No access to file system or linked servers set up with self-mapping.
- To impersonate a user/login you need IMPERSONATE permission on that user/login.

About REVERT

- Impersonation is in force until the REVERT command is executed *in the same scope*.
 - REVERT in an inner scope has no effect towards an EXECUTE AS in outer scope.
- REVERT must be executed from the database where EXECUTE AS was executed.
- Extraneous REVERT are ignored without error.
- Tip: Always put REVERT in its own batch, so that is executed also if there is an error.

A Small Hiccup

`sysadmin.sql`

- `EXECUTE AS USER = 'dbo'` may produce error 15517:
Cannot execute as the database principal because the principal "dbo" does not exist, this type of principal cannot be impersonated, or you do not have permission.
- Happens easily with databases restored from other servers.
 - Due to SID mismatch between sys.databases and the database itself.
- Routine: set database owner when you restore a database.
- My opinion: a database should be owned by an SQL login that exists solely to own that database.

Running Database Maintenance

1. Ola Hallengren's Maintenance Solution.

- IndexOptimize accepts the parameter @ExecuteAsUser, which you set to 'dbo' (or another user of your choice).

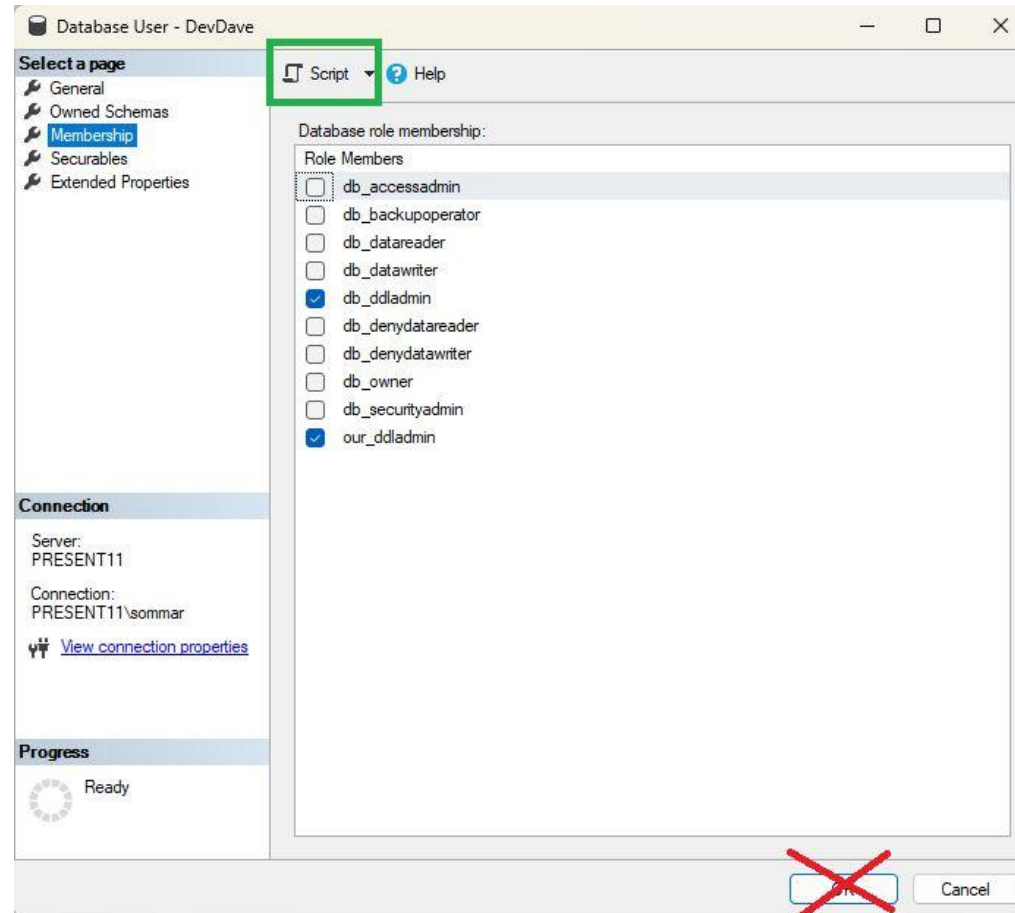
2. The maintenance plans in SSMS – no provision for EXECUTE AS USER.

- Only do backup and DBCC. Tell users to run index and stats maintenance themselves.
- Or for index and stats maintenance, have one task per database. Each task has its own login that is db_owner solely in that database. (Could be the database owner.)

DDL in General

- For the server DBA any DDL on database level is dangerous, when there is a local db_owner or similar: creating indexes, users, granting permissions etc.
- Must bracket all DDL with `EXECUTE AS USER = 'dbo'` in that case.
- Corollary: cannot use UI in SSMS directly. Must use SQL commands.
- Or use the script button, see next slide.

Script button in SSMS UI



Devs and DDL Triggers

`sysadmin.sql`

- A user in db_ddladmin could create a DDL trigger to hijack the permissions of a db_owner user.
- This role should not have rights to create DDL triggers!
- Take out all users from db_ddladmin.
- Create a new role, e.g. our_ddladmin (or ask for an AD group) that you add as member to db_ddladmin.
- Add devs to this role.
- Deny our_ddladmin ALTER ANY DATABASE DDL TRIGGER.
- Why not DENY db_ddladmin directly? SQL Server doesn't let us.

DML Triggers and Stored Procedures

- When you insert/update/delete data in tables, your permissions can be hijacked by a trigger.
- When running stored procedures, a developer may have hidden malicious code somewhere.
- If you only need to protect sysadmin (no one between db_owner and plain users) same as before: impersonate dbo.
- To get data from outside the database, read into temp tables before starting impersonation.

How to Protect db_owner?

- If there are users who can change stored procedures, triggers etc, they can hijack db_owner.
- Protection: PoLP, Principle of Least Privilege.
- Create a sandbox user with limited permissions and impersonate that user. For instance:

```
CREATE USER SandboxUser WITHOUT LOGIN
GRANT EXECUTE TO SandboxUser
GRANT SELECT, INSERT, UPDATE, DELETE TO SandboxUser
EXECUTE AS USER = 'SandboxUser'
```

Permissions for Sandbox User

- In many cases, a sandbox user with permission to run all stored procedures and full rights on all tables is enough.
- Specifically, it is sufficient with someone like DevDave who already has these permissions.
- But what about Bettie BusinessAnalyst who has her own schema where she can create tables, and you need to work with her tables?
- You may need a sandbox user with same permissions as Bettie.

Access Outside the Database

- With EXECUTE AS USER you are sandboxed into the current database.
 - What about stored procedures with access to other databases?
 - What if you need to join to big tables in other databases? Getting data into temp tables beforehand may be impractical.
- You need to use a sandbox *login* that you impersonate with EXECUTE AS LOGIN.
- Easy to create if you are sysadmin.
- A db_owner needs to ask the server DBA to create one and get IMPERSONATE rights on that login.

Notes on Sandbox Login

- Would typically be an SQL login, as that is easier to create.
- You can impersonate an SQL login, even if SQL authentication is disabled.
- ALTER LOGIN sandbox_login DISABLE is a good idea.
- ...Unless you prefer to log in as the sandbox login in a separate window.
 - But then you need keep track of password and all that.

But **SELECT** Should Be Safe?

`sysadmin.sql`

- A table could have a computed column that calls a UDF.
- Or user has renamed table and replaced it with a view that calls a UDF that returns the data in the table.
- UDF could call `xp_cmdshell` to connect back as `sysadmin`.
- Best practice: Keep `xp_cmdshell` disabled, enable and re-disable when you need it.
- But there are other extended stored procedures...
- There is also `OPENQUERY` & `OPENROWSET` (if enabled).
- So plain `SELECT` also needs `EXECUTE AS USER`.

A Change of Theme

- So far, we have discussed how hijacking attacks can be performed and how we can protect us.
- We will now make a break and discuss how we can reduce the need for giving out permissions that permit users to perform hijacking attacks.
- We will come back to more hijacking attacks later...

Application Users

- Application may log in users with integrated security or application may use its own login/password.
- Ideally users and application logins should only have permission to run stored procedures.
- If application submits direct SQL statements, they also need SELECT, INSERT, UPDATE and DELETE permissions.
- But they should never have more permissions. **Never!**
 - Minimises the damage of any SQL-injection hole.

Applications and Elevated Permissions

- What if an application needs permissions beyond EXECUTE and read/write on tables?
 - Example: show a count of users connected to the database; requires server-level permission to use DMV.
- Instead of granting the permission to an application login, you can package the permission inside a stored proc with full control over how the permission is used.
- You sign the procedure with a certificate, and from this certificate you create a user/ login which you grant the required permission(s).

Certificate Signing: Article and Video

- On my web site, there is an in-depth article about this technique: *Packaging Permissions in Stored Procedures*,
<https://www.sommarskog.se/grantperm.html>.
- Article includes stored procedure and script to automate the process.
- There is also a recording on [YouTube](#).

Certificate Signing: Key Facts

- When you change a signed procedure, signature and permissions disappear.
- Thus, it must be signed again.
- Example: Debbie Owner wants a procedure that shows user count.
- Server DBA reviews it and approves it by signing and granting permissions.
- If Debbie changes the procedure, she needs new approval.
- Thus, server DBA is in full control.

Certificate Signing and Permission Hijacking

- The packaged permissions apply inside dynamic SQL invoked by the signed procedure.
 - They also apply inside system procedures called by the SP.
- But they are removed when entering user-written modules: sub-procedures, triggers and functions.
- That is, the packaged permissions cannot be hijacked!
- But watch out for dynamic SQL! It could be constructed from data you have no control over.

Agent Jobs

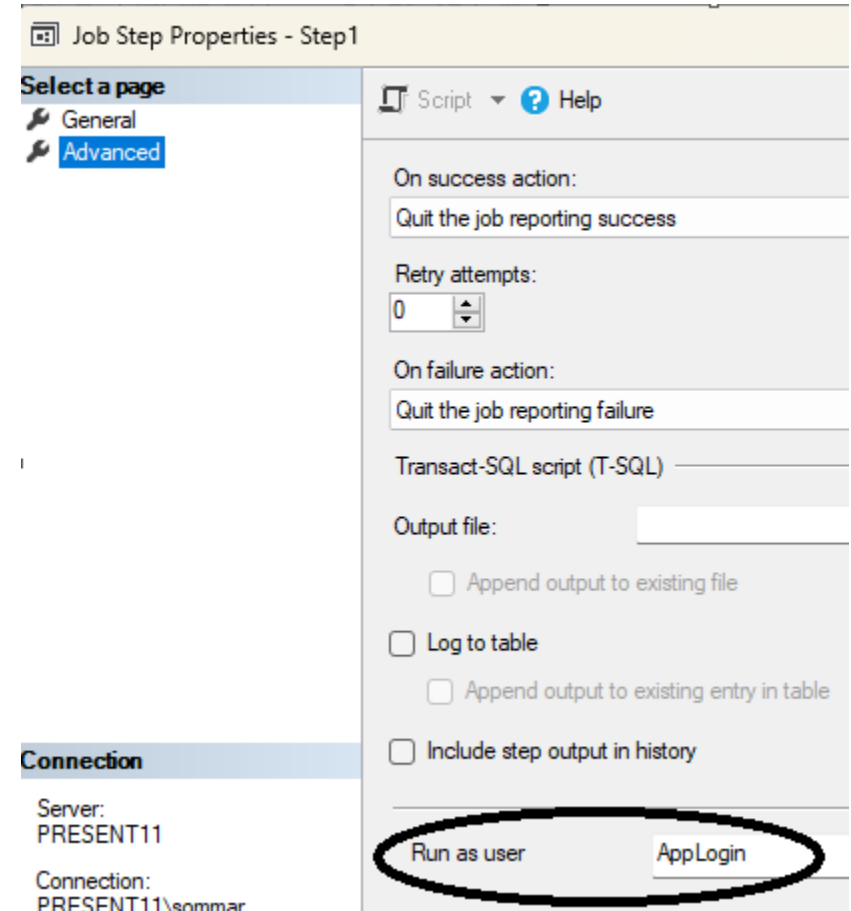
- DevDave approaches you and says: We need to run purges at night, can you create an Agent job to run this procedure?
- If you schedule it as sysadmin, Debbie and Dave can now make all sorts of changes to the procedure.
- First line of defence: Hey, why don't you schedule it from Task Scheduler on the application server?
 - After all, Agent is for management tasks, not application jobs, isn't it?
- But you may not win that battle...

Some Agent Principles

- Jobs owners should not be persons who can leave the company.
- Avoid adding users to msdb.
- From this follows that setting up an Agent job is always a sysadmin task.
- Application jobs in Agent should execute with same permissions as application users.
- If a job needs to perform actions that require higher permissions, use signed procedures.

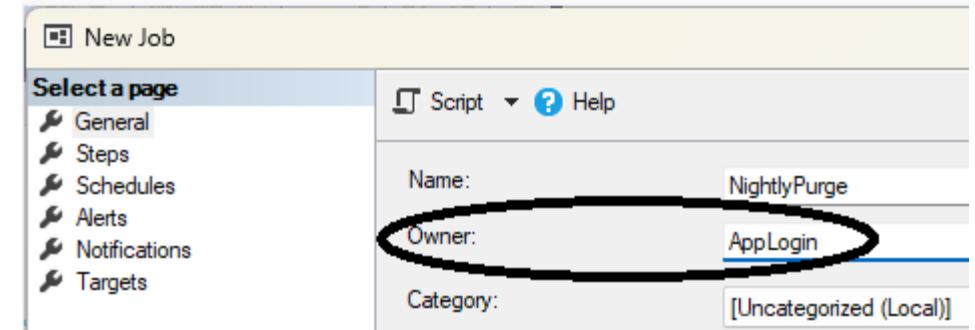
Alternative 2: Set User on Job Step

- Set user on the Advanced tab on the job step.
- Agent impersonates user with EXECUTE AS USER.
- Sandboxed into database, no cross-DB access or linked servers.
- Job must be owned by sysadmin.
 - If not, setting is ignored.



Alternative 3: Set Job Owner

- Set job owner on first page of job.
- SQL login or Windows login with “application permissions”
- Agent impersonates the owner with EXECUTE AS LOGIN.
 - No restriction with cross-database or server access.
 - Linked servers with login-mapping work. Those with self-mapping *do not*.
- Side effect: job owner gets added as a user to msdb and as member of SQLAgentUserRole on next install of a CU.



Alternative 4: CmdExec Job with Proxy

- More steps and involves more people.
- No impersonation, since Agent performs a real login in Windows.
- Thus, no restrictions with linked servers.
- While the most complicated, the most general.
- Service account for SQL Server Agent needs the Windows permission **Replace a process level token.**

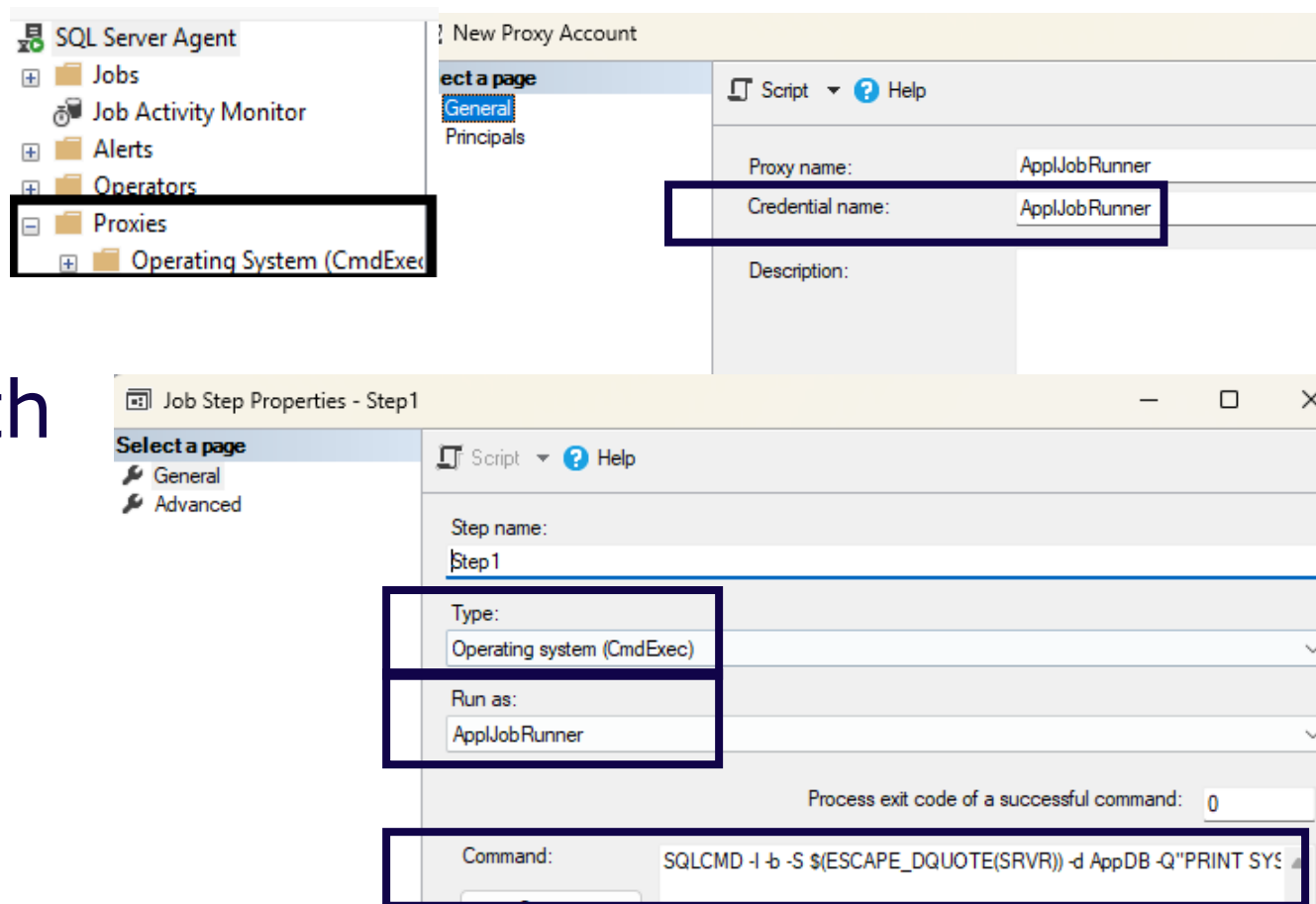
Alternative 4: CmdExec Job with Proxy

- First request to get an AD account.
- Create login in SQL Server.
 - For linked server access, this may include more than one instance.
- Create database user and grant application permissions.
 - For instance, add it to the ApplicationUsers role.
- Create a credential for this account in SQL Server:

```
CREATE CREDENTIAL App1JobRunner  
  WITH IDENTITY = 'DOMAIN\App1JobRunner',  
  SECRET = 'TheWindowsPassword'
```

CmdExec Job with Proxy

- Create proxy in Agent with access to CmdExec.
- Use credential from previous step.
- Create job in Agent with sysadmin as owner.
- Create job step.
 - **Type:** CmdExec.
 - **RunAs:** The proxy.
 - **Command:** SQLCMD.



CmdExec Job with Proxy: Running SQLCMD

```
SQLCMD -I -b -S $(ESCAPE_DQUOTE(SRVR)) -C  
-d AppDB -Q"EXEC somesp"
```

- I – For SET QUOTED_IDENTIFIER ON.
- b – Propagate error code, so job can be reported as failed.
- S – Use Agent token for server name to make it easier to move job to another instance.
- C – Trust Server Certificate, may be needed on SQL 2025.
- d – The database to connect to.
- Q – Command to run.

Notifications to Devs

- Since developers and application admins are not in msdb, they don't have access to Agent history.
- Set up mail notifications, so they get mail if job fails. (On success as well, if they want.)
- On the Advanced page of the job step, you can define an output file. (Does not work when you set job owner.)
- Direct output file to a share where application team has access.

Attacks Through Deployment

- You get a deployment package from the application team.
- It could include all sorts of nasty things:
 - Adding extra logins/users with elevated permissions.
 - Manipulate data (in any database).
- Or what about this:
 - Set up a temporary linked server to system B with sensitive data and where you also are sysadmin.
 - They set up another linked server to a test system C they have access to and dump the sensitive data there.
- How do you defend yourself against hijacking?

Defence Against Deployment Attacks

- For a plain script, one means of protection:
`EXECUTE AS USER = 'dbo' WITH NO REVERT`
- WITH NO REVERT causes any REVERT in the script to fail.
- Rather than dbo, use a sandbox user in our_ddladmin, or whatever permission you are prepared to grant the app team.
- Use a sandbox login, if actions are expected outside the database.
- First run in QA, to reveal unexpected but legit actions.

Deployment Attacks, Cont'd

- Deployment script may have legit actions that requires high permissions.
- Or deployment is through DACPAC or MSI install, with no options to control login.
- Auditing may be your only choice here.
- There are several options, but SQL Server Audit is the only that audits that it has been turned off.
- DDL Triggers? They can be turned off with DISABLE TRIGGER – which does not fire DDL triggers...

Attacks Through Applications

- You are using the time-reporting system or some other local application.
- Developer of that application has a hook that recognises you and sets up a connection to an instance with sensitive data, abusing your permissions for malicious actions.
- Possible defence: One Windows user for normal company work, and a second login for DBA stuff, maybe through jump servers.
- Rigid network control of which machines can access each other.

Summary: What This Has Been About

- Malicious users with some elevated permission can inject code that you unknowingly execute with your permissions, performing actions you don't agree to.
- Recall: Code can be DDL triggers, DML triggers, stored procedures and more.
- "Elevated permission" here includes access to the deployment pipeline!

Summary: Means of Protection

- Principle of Least Privilege, PoLP.
- Impersonate dbo to get rid of your server-level permissions.
- To protect db_owner from being hijacked, use sandbox users or logins.
- Never put people directly in db_ddladmin, but create a role our_ddladmin that you deny the rights to DDL triggers.

Summary: Means of Protection II

- Application users should have limited permissions, at most SELECT, INSERT, UPDATE and DELETE on tables.
- Application tasks in Agent jobs should run with application permissions, never elevated permissions.
- Use procedure signing when application users need to perform actions beyond their permission set.
- The application team may be in practice be equal to db_owner – even if they don't have access to the server.

Final Words

- Some attacks I have suggested may seem very contrived.
- Remember: if the stakes are high enough, a possible attack will be attempted, sooner or later at some place.
- Don't overlook the benevolent attacker.
 - That is, someone who performs an attack not out of malicious intent, but to circumvent corporate red tape to get their job done.
- You could have made those users sysadmin, but you decided to lock down their permissions. Certainly not a bad idea in itself.
- But if you don't watch out, they may find a way to "work around" that restriction.

Your feedback is important to us



Evaluate this session at:

passdatacommunitysummit.com/evaluations

Thank you

Erland Sommarskog

Don't Let Your Permissions be Hijacked:

<https://www.sommarskog.se/perm-hijack.html>.

Credits to **Baptiste Devigne** for pointing out the SELECT threat.



esquel@sommarskog.se



Slides and scripts:

<https://www.sommarskog.se/present>

Cleanup

sysadmin.sql