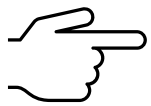
SQL Security Scripts for DBA

Asfaw Gedamu Haileselasie



Caution: Please use the commands with care, try them on test environments first.

Creating and managing user accounts

```
-- Create a new user account
CREATE USER username WITH PASSWORD 'password';

-- Grant the user permissions to a database
GRANT SELECT, UPDATE, DELETE ON database.table TO username;

-- Revoke permissions from the user
REVOKE SELECT, UPDATE, DELETE ON database.table FROM username;

-- Delete the user account
DROP USER username;
```

Granting and revoking permissions

```
-- Grant the user permissions to a database
GRANT SELECT, INSERT, UPDATE, DELETE ON database.table TO username;


-- Revoke permissions from the user
REVOKE SELECT, INSERT, UPDATE, DELETE ON database.table FROM username;
```

Backing up and restoring your database

```
-- Back up the database
BACKUP DATABASE database TO DISK = 'backup.sql';

-- Restore the database from a backup
RESTORE DATABASE database FROM DISK = 'backup.sql';
```

The first line of code backs up a database to a file called `backup.bak`. The second line of code restores a database from a file called `backup.bak`.

Implementing encryption

```
CREATE TABLE [table] (
    [column] [datatype] ENCRYPTED WITH (ALGORITHM = 'AES_256',
COLUMN_ENCRYPTION_KEY = [key_id]);
);
```

The first line of code creates a new table with a column that is encrypted using the AES_256 algorithm. The second line of code specifies the key ID that will be used to encrypt the column.

Monitoring database activity

```
CREATE TRIGGER [trigger_name] ON [table] AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    PRINT 'A row has been inserted, updated, or deleted from the [table]
table.';
END;
```

The first line of code creates a trigger named `trigger_name` that fires after a row is inserted, updated, or deleted from the `table` table. The second line of code prints a message to the console when the trigger fires.

Scanning for security vulnerabilities

To conduct vulnerability scan against a database we have to create two stored procedures `sp_dbcc_check_security` and `sp_dbcc_check_permissions`.

First: Create **`sp_dbcc_check_security`**

```
CREATE PROCEDURE sp_dbcc_check_security
(
   -- @dbname is the name of the database to scan.
   @dbname = NULL
)
AS
BEGIN
   -- Check the permissions on the master database.
```

```
    EXEC sp_dbcc_check_permissions @dbname = 'master';

    -- Check the permissions on all other databases.
    IF @dbname IS NULL
    BEGIN
        SELECT db_name
        FROM sys.databases
        WHERE db_name <> 'master'
        ORDER BY db_name;

        FOR EACH ROW
        BEGIN
            EXEC sp_dbcc_check_permissions @dbname = @row.db_name;
        END;
    END;
END;
```

The above stored procedure performs a security scan against the database. The first thing the stored procedure does is check the permissions on the master database. This is because the master database contains the system tables, which are the most critical tables in the database. If the permissions on the master database are not secure, then the entire database is vulnerable.

The next thing the stored procedure does is check the permissions on all other databases. This is because if the permissions on a database are not secure, then anyone with access to the database can potentially read, modify, or delete data in the database.

The stored procedure uses the `sp_dbcc_check_permissions` stored procedure to check the permissions on the databases. The `sp_dbcc_check_permissions` stored procedure takes a single parameter, the name of the database to check. The stored procedure returns a report of any security vulnerabilities that are found.

The `sp_dbcc_check_security` stored procedure is a powerful tool that can be used to help you secure your database. However, it is important to note that the stored procedure does not scan the entire database for security vulnerabilities. The stored procedure only scans the permissions on the databases. If you want to scan the entire database for security vulnerabilities, you will need to use a different tool.

Second: Create **sp_dbcc_check_permissions**

```
CREATE PROCEDURE sp_dbcc_check_permissions
(
    -- @dbname is the name of the database to scan.
    @dbname = NULL
)
AS
BEGIN
    -- Check the permissions on the dbo schema.
    EXEC sp_dbcc_check_permissions @dbname = @dbname, @schema = 'dbo';

    -- Check the permissions on all other schemas.
    IF @dbname IS NULL
    BEGIN
        SELECT schema_name
        FROM sys.schemas
        WHERE schema_name <> 'dbo'
        ORDER BY schema_name;

        FOR EACH ROW
        BEGIN
            EXEC sp_dbcc_check_permissions @dbname = @dbname, @schema =
@row.schema_name;
```

```
    END;
  END;
END;
```

This stored procedure performs a security scan against the database. The first thing the stored procedure does is check the permissions on the dbo schema. This is because the dbo schema is the default schema for the database, and it contains the most important tables in the database. If the permissions on the dbo schema are not secure, then anyone with access to the database can potentially read, modify, or delete data in the database.

The next thing the stored procedure does is check the permissions on all other schemas. This is because if the permissions on a schema are not secure, then anyone with access to the schema can potentially read, modify, or delete data in the schema.

The stored procedure uses the `sp_dbcc_check_permissions` stored procedure to check the permissions on the schemas. The `sp_dbcc_check_permissions` stored procedure takes two parameters, the name of the database to check and the name of the schema to check. The stored procedure returns a report of any security vulnerabilities that are found.

The `sp_dbcc_check_permissions` stored procedure is a powerful tool that can be used to help you secure your database. However, it is important to note that the stored procedure does not scan the entire database for security vulnerabilities. The stored procedure only scans the permissions on the schemas. If you want to scan the entire database for security vulnerabilities, you will need to use a different tool.

Finally: Conduct Security Scan

```
-- Run a security scan against the database
USE master;
GO
EXEC sp_dbcc_check_security;
GO
```

This script runs a security scan against the database. The first line of code, `USE master;`, specifies that the script will run in the `master` database. The second line of code, `GO`, is a delimiter that tells SQL Server to execute the previous line of code. The third line of code, `EXEC sp_dbcc_check_security;`, executes the `sp_dbcc_check_security` stored procedure. This stored procedure performs a security scan against the database and returns a report of any security vulnerabilities that are found.

The `sp_dbcc_check_security` stored procedure takes a number of parameters, but the most important one is the `dbname` parameter. This parameter specifies the name of the database that

you want to scan. In this case, the `dbname` parameter is not specified, so the stored procedure will scan the `master` database.

The `sp_dbcc_check_security` stored procedure returns a report of any security vulnerabilities that are found. The report includes the following information:

- The name of the vulnerability
- The severity of the vulnerability
- A description of the vulnerability

The report also includes a link to the Microsoft website where you can learn more about the vulnerability.

Grant delete previlege

The following script generates a SQL script that grants DELETE privileges on all tables for which the current user does not have DELETE privileges. The script then executes the generated SQL script.

```
SET PAGESIZE 0

SET FEEDBACK OFF

SET VERIFY OFF


SPOOL temp.sql


SELECT 'GRANT DELETE ON "' || u.table_name || '" TO &1;'

FROM   user_tables u

WHERE  NOT EXISTS (SELECT '1'

                   FROM   all_tab_privs a

                   WHERE  a.grantee    = UPPER('&1')

                   AND    a.privilege  = 'DELETE'

                   AND    a.table_name = u.table_name);

SPOOL OFF

@temp.sql
```

```
-- Comment out following line to prevent immediate run
@temp.sql

SET PAGESIZE 14
SET FEEDBACK ON
SET VERIFY ON
```

Grant execute previlege

The script below generates a SQL script that grants EXECUTE privileges on all packages, procedures, and functions for which the current user does not have EXECUTE privileges. The script then executes the generated SQL script.

```
SET PAGESIZE 0
SET FEEDBACK OFF
SET VERIFY OFF

SPOOL temp.sql

SELECT 'GRANT EXECUTE ON "' || u.object_name || '" TO &1;'
FROM   user_objects u
WHERE  u.object_type IN ('PACKAGE','PROCEDURE','FUNCTION')
AND    NOT EXISTS (SELECT '1'
                   FROM   all_tab_privs a
                   WHERE  a.grantee    = UPPER('&1')
                   AND    a.privilege  = 'EXECUTE'
                   AND    a.table_name = u.object_name);

SPOOL OFF
```

```
-- Comment out following line to prevent immediate run
@temp.sql

SET PAGESIZE 14
SET FEEDBACK ON
SET VERIFY ON
```

Grant insert previlege

The following script generates a SQL script that grants INSERT privileges on all tables for which the current user does not have INSERT privileges. The script then executes the generated SQL script.

```
SET PAGESIZE 0
SET FEEDBACK OFF
SET VERIFY OFF

SPOOL temp.sql

SELECT 'GRANT INSERT ON "' || u.table_name || '" TO &1;'
FROM   user_tables u
WHERE  NOT EXISTS (SELECT '1'
                   FROM   all_tab_privs a
                   WHERE  a.grantee    = UPPER('&1')
                   AND    a.privilege  = 'INSERT'
                   AND    a.table_name = u.table_name);

SPOOL OFF
-- Comment out following line to prevent immediate run
```

```
@temp.sql

SET PAGESIZE 14
SET FEEDBACK ON
SET VERIFY ON
```

Grant select previlege

Here is a script that generates a SQL script that grants SELECT privileges on all tables, views, and sequences for which the current user does not have SELECT privileges. The script then executes the generated SQL script.

```
SET PAGESIZE 0
SET FEEDBACK OFF
SET VERIFY OFF

SPOOL temp.sql

SELECT 'GRANT SELECT ON "' || u.object_name || '" TO &1;'
FROM   user_objects u
WHERE  u.object_type IN ('TABLE','VIEW','SEQUENCE')
AND    NOT EXISTS (SELECT '1'
                   FROM   all_tab_privs a
                   WHERE  a.grantee    = UPPER('&1')
                   AND    a.privilege  = 'SELECT'
                   AND    a.table_name = u.object_name);

SPOOL OFF
```

```
-- Comment out following line to prevent immediate run
@temp.sql

SET PAGESIZE 14
SET FEEDBACK ON
SET VERIFY ON
```

Grant update previlege

This script generates a SQL script that grants UPDATE privileges on all tables for which the current user does not have UPDATE privileges. The script then executes the generated SQL script.

```
SET PAGESIZE 0
SET FEEDBACK OFF
SET VERIFY OFF

SPOOL temp.sql

SELECT 'GRANT UPDATE ON "' || u.table_name || '" TO &1;'
FROM   user_tables u
WHERE  NOT EXISTS (SELECT '1'
                   FROM   all_tab_privs a
                   WHERE  a.grantee    = UPPER('&1')
                   AND    a.privilege  = 'UPDATE'
                   AND    a.table_name = u.table_name);

SPOOL OFF
```

```
-- Comment out following line to prevent immediate run
@temp.sql

SET PAGESIZE 14
SET FEEDBACK ON
SET VERIFY ON
```

Create package synonyms own by a user

Here, the script generates a SQL script that creates synonyms for all packages, procedures, and functions owned by the current user for which there are no corresponding synonyms. The script then executes the generated SQL script.

```
SET PAGESIZE 0
SET FEEDBACK OFF
SET VERIFY OFF

SPOOL temp.sql

SELECT 'CREATE SYNONYM "' || a.object_name || '" FOR "' || a.owner || '"."' || a.object_name
|| '";'
FROM   all_objects a
WHERE  a.object_type IN ('PACKAGE','PROCEDURE','FUNCTION')
AND    a.owner = UPPER('&1')
AND    NOT EXISTS (SELECT '1'
                   FROM   user_synonyms u
                   WHERE  u.synonym_name = a.object_name
                   AND    u.table_owner  = UPPER('&1'));
```

```
SPOOL OFF

-- Comment out following line to prevent immediate run
@temp.sql

SET PAGESIZE 14
SET FEEDBACK ON
SET VERIFY ON
```

Query users with write access to a schema

This SQL script is used to identify users who have write access to a schema. The script has three parts:

The first part of the script lists all users who have direct grants on the schema for the privileges of INSERT, UPDATE, or DELETE.

The second part of the script lists all users who have been granted a role that has INSERT, UPDATE, or DELETE privileges on the schema.

The third part of the script lists all users who have been granted system privileges that allow them to insert, update, or delete any table in the schema.

```
set verify off

-- Direct grants
select distinct grantee
from   dba_tab_privs
where  privilege in ('INSERT', 'UPDATE', 'DELETE')
and    owner = upper('&1')
union
-- Grants via a role
```

```
select distinct grantee
from   dba_role_privs
      join dba_users on grantee = username
where  granted_role IN (select distinct role
                        from   role_tab_privs
                        where  privilege in ('INSERT', 'UPDATE', 'DELETE')
                        and    owner = upper('&1')
                        union
                        select distinct role
                        from   role_sys_privs
                        where  privilege in ('INSERT ANY TABLE', 'UPDATE ANY TABLE', 'DELETE
ANY TABLE'))
union
-- Access via ANY sys privileges
select distinct grantee
from   dba_sys_privs
join   dba_users on grantee = username
where  privilege in ('INSERT ANY TABLE', 'UPDATE ANY TABLE', 'DELETE ANY TABLE');
```

Create synonyms for sequences in a schema

Use the following SQL script to create synonyms for all sequences in the schema that do not already have a synonym. The script generates a SQL script that creates synonyms for all sequences owned by the current user for which there are no corresponding synonyms. The script then executes the generated SQL script.

```
SET PAGESIZE 0
SET FEEDBACK OFF
SET VERIFY OFF


SPOOL temp.sql
```

```
SELECT 'CREATE SYNONYM "' || a.object_name || '" FOR "' || a.owner || '"."' || a.object_name
|| '";'
FROM   all_objects a
WHERE  a.object_type = 'SEQUENCE'
AND    a.owner       = UPPER('&1')
AND    NOT EXISTS (SELECT '1'
                   FROM   user_synonyms a1
                   WHERE  a1.synonym_name = a.object_name
                   AND    a1.table_owner  = UPPER('&1'));


SPOOL OFF

-- Comment out following line to prevent immediate run
@temp.sql

SET PAGESIZE 14
SET FEEDBACK ON
SET VERIFY ON
```

Create synonyms for all tables in a schema

Use the following SQL script to create synonyms for all tables in the schema that do not already have a synonym. The script generates a SQL script that creates synonyms for all tables owned by the current user for which there are no corresponding synonyms. The script then executes the generated SQL script.

```
SET PAGESIZE 0
SET FEEDBACK OFF
SET VERIFY OFF
```

```
SPOOL temp.sql

SELECT 'CREATE SYNONYM "' || a.table_name || '" FOR "' || a.owner || '"."' || a.table_name ||
'";'
FROM   all_tables a
WHERE  NOT EXISTS (SELECT '1'
                   FROM   user_synonyms u
                   WHERE  u.synonym_name = a.table_name
                   AND    u.table_owner  = UPPER('&1'))
AND    a.owner = UPPER('&1');

SPOOL OFF

-- Comment out following line to prevent immediate run
@temp.sql

SET PAGESIZE 14
SET FEEDBACK ON
SET VERIFY ON
```

Script: view_synonyms.sql

Use the SQL script below to create synonyms for all views in the schema that do not already have a synonym. The script generates a SQL script that creates synonyms for all views owned by the current user for which there are no corresponding synonyms. The script then executes the generated SQL script.

```
SET PAGESIZE 0
```

```
SET FEEDBACK OFF
SET VERIFY OFF

SPOOL temp.sql

SELECT 'CREATE SYNONYM "' || a.view_name || '" FOR "' || a.owner || '"."' || a.view_name ||
'";'
FROM   all_views a
WHERE  a.owner = UPPER('&1')
AND    NOT EXISTS (SELECT '1'
                   FROM   user_synonyms u
                   WHERE  u.synonym_name = a.view_name
                   AND    u.table_owner  = UPPER('&1'));

SPOOL OFF

-- Comment out following line to prevent immediate run
@temp.sql

SET PAGESIZE 14
SET FEEDBACK ON
SET VERIFY ON
```