EX.NO:	
	Implement Database Security

Aim:

To implementing database security mechanisms, specifically access control and authentication, on a Windows OS

Algorithm / Program:

1. **Install Microsoft SQL Server**: Ensure that Microsoft SQL Server is installed on your Windows machine. You can download it from the Microsoft SQL Server website.
2. **Install SQL Server Management Studio (SSMS)**: This is a graphical tool for managing SQL Server instances. You can download it from the SSMS download page.

Step-by-Step Guide :

This environment involves several steps. Here, we'll use Microsoft SQL Server as an example database management system to demonstrate how to set up and secure a database on Windows.

1. Understanding Database Security Threats

Common security threats to databases include:

- Unauthorized access
- SQL injection attacks
- Privilege escalation
- Data breaches due to weak authentication
- Insider threats

2. Implementing Access Control

Access control involves defining who can access the database and what actions they can perform. This is typically done using roles and permissions.

1. **Create Database and User Roles**:
   - Open SQL Server Management Studio (SSMS).
   - Connect to your SQL Server instance.
   - Create a new database:

     sql

     ```
     CREATE DATABASE SecureDB;
     GO
     ```

2. **Create Users and Roles**:
    - Create a user with administrative privileges:

      ```
      sql
      USE SecureDB;
      CREATE LOGIN admin_user WITH PASSWORD = 'StrongPassword';
      CREATE USER admin_user FOR LOGIN admin_user;
      EXEC sp_addrolemember 'db_owner', 'admin_user';
      GO
      ```

    - Create a read-only user:

      ```
      sql
      USE SecureDB;
      CREATE LOGIN read_only_user WITH PASSWORD = 'ReadOnlyPassword';
      CREATE USER read_only_user FOR LOGIN read_only_user;
      EXEC sp_addrolemember 'db_datareader', 'read_only_user';
      GO
      ```

3. **Verify Access Control**:
    - Log in as read_only_user and try to perform write operations to confirm that they are restricted.

3. *Implementing Authentication*

Authentication ensures that only authorized users can access the database.

1. **Enforce Strong Password Policies**:
    - Open SQL Server Management Studio (SSMS).
    - Connect to your SQL Server instance.
    - Set up strong password policies:

      ```
      sql
      Copy code
      ALTER LOGIN admin_user WITH CHECK_POLICY = ON,
      CHECK_EXPIRATION = ON;
      ALTER LOGIN read_only_user WITH CHECK_POLICY = ON,
      CHECK_EXPIRATION = ON;
      ```

2. **Enable Windows Authentication**:
    - SQL Server supports both SQL Server authentication and Windows authentication. Windows authentication is generally more secure as it integrates with Windows Active Directory.
    - In SSMS, navigate to **Security > Logins**.
    - Right-click and select **New Login**.
    - Choose **Windows authentication** and specify the Windows user or group.

3. **Configure SQL Server for Mixed Mode Authentication (if needed)**:
    - Open SQL Server Configuration Manager.
    - Navigate to **SQL Server Services**.
    - Right-click on the SQL Server instance and select **Properties**.
    - In the **Security** tab, choose **SQL Server and Windows Authentication mode**.
    - Restart the SQL Server service for the changes to take effect.

4. Testing and Verification

1. **Test Access Control**:
    - Ensure users have the correct permissions and cannot access or modify data beyond their privileges.
2. **Test Authentication Mechanisms**:
    - Attempt to log in with weak passwords and verify that access is denied.
    - Ensure that both SQL Server and Windows Authentication modes work as expected.
3. **Audit and Logging**:
    - Enable SQL Server auditing to monitor access and detect any unauthorized attempts.

      ```
      sql
      CREATE SERVER AUDIT AuditTest
      TO FILE (FILEPATH = 'C:\Audit\');
      ALTER SERVER AUDIT AuditTest WITH (STATE = ON);
      GO
      CREATE DATABASE AUDIT SPECIFICATION AuditSpec
      FOR SERVER AUDIT AuditTest
      ADD (SELECT ON DATABASE::SecureDB BY read_only_user),
      ADD (SELECT, INSERT, UPDATE, DELETE ON DATABASE::SecureDB BY
      admin_user);
      ALTER DATABASE AUDIT SPECIFICATION AuditSpec WITH (STATE =
      ON);
      GO
      ```

Output :

```
SELECT name FROM sys.databases;
name
----
master
tempdb
model
msdb
SecureDB
```

```
DatabaseRoleName  DatabaseUserName
---------------- ----------------
db_owner          admin_user
```

Result:

EX.NO:

Implement Encryption and Integrity Control-Database Security

Aim:

To implementing encryption and integrity controls for databases is crucial to protect sensitive data and ensure that it remains unaltered.

Algorithm /Program:

1. **Microsoft SQL Server**: Ensure that SQL Server is installed on your Windows system.
2. **SQL Server Management Studio (SSMS)**: Ensure SSMS is installed for managing the SQL Server instance.

Steps to Implement Encryption and Integrity Controls

1. Transparent Data Encryption (TDE)

Transparent Data Encryption (TDE) helps protect data at rest by encrypting the database files. This ensures that the database files are not readable if accessed directly from the disk.

1. **Create a Master Key**

   The master key is required to encrypt the database encryption key.

   ```
   sql
   USE master;
   GO
   CREATE MASTER KEY ENCRYPTION BY PASSWORD =
   'StrongPasswordForMasterKey';
   GO
   ```

2. **Create a Certificate**

   The certificate is used to protect the database encryption key.

   ```
   sql
   USE master;
   GO
   CREATE CERTIFICATE TDE_Cert WITH SUBJECT = 'TDE Certificate';
   GO
   ```

3. **Create a Database Encryption Key**

The database encryption key is used to encrypt the database.

sql

```
USE SecureDB;
GO
CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = AES_256
ENCRYPTION BY SERVER CERTIFICATE TDE_Cert;
GO
```

4. **Enable TDE on the Database**

   sql

   ```
   ALTER DATABASE SecureDB
   SET ENCRYPTION ON;
   GO
   ```

5. **Verify Encryption**

   sql

   ```
   USE SecureDB;
   GO
   SELECT name, is_encrypted
   FROM sys.databases
   WHERE name = 'SecureDB';
   GO
   ```

   Expected Output:

   - is_encrypted should be 1 for the SecureDB database.

2. Column-Level Encryption

Column-level encryption provides fine-grained control over the encryption of specific data within a table.

1. **Create a Symmetric Key**

   sql

   ```
   USE SecureDB;
   GO
   CREATE SYMMETRIC KEY SymmetricKey
   WITH ALGORITHM = AES_256
   ENCRYPTION BY CERTIFICATE TDE_Cert;
   GO
   ```

2. **Encrypt Data in a Table**
    - Create a table and insert some data:

      sql
      ```
      CREATE TABLE SensitiveData (
          ID INT PRIMARY KEY,
          SensitiveInfo VARBINARY(MAX)
      );
      GO

      OPEN SYMMETRIC KEY SymmetricKey
      DECRYPTION BY CERTIFICATE TDE_Cert;

      INSERT INTO SensitiveData (ID, SensitiveInfo)
      VALUES (1, ENCRYPTBYKEY(KEY_GUID('SymmetricKey'), 'Sensitive
      Information'));
      GO
      CLOSE SYMMETRIC KEY SymmetricKey;
      ```

3. **Decrypt Data for Viewing**

   sql
   ```
   OPEN SYMMETRIC KEY SymmetricKey
   DECRYPTION BY CERTIFICATE TDE_Cert;

   SELECT ID, CONVERT(VARCHAR(MAX), DECRYPTBYKEY(SensitiveInfo)) AS
   SensitiveInfo
   FROM SensitiveData;
   GO

   CLOSE SYMMETRIC KEY SymmetricKey;
   ```

   Expected Output:

    - The SensitiveInfo column should display the decrypted data.

3. Data Integrity Controls

Implementing data integrity controls ensures that the data is not tampered with and maintains its accuracy and consistency.

1. **Using Hashes for Data Integrity**
    - Create a table to store hashed data:

      sql
      ```
      CREATE TABLE DataIntegrity (
      ```

```
    ID INT PRIMARY KEY,
    OriginalData NVARCHAR(255),
    DataHash VARBINARY(64)
);
GO
```

- Insert data with a hash:

  sql

  ```
  INSERT INTO DataIntegrity (ID, OriginalData, DataHash)
  VALUES (1, 'Important Data', HASHBYTES('SHA2_256', 'Important Data'));
  GO
  ```

- Verify data integrity:

  sql

  ```
  DECLARE @OriginalData NVARCHAR(255);
  DECLARE @Hash VARBINARY(64);

  SELECT @OriginalData = OriginalData, @Hash = DataHash
  FROM DataIntegrity
  WHERE ID = 1;

  IF @Hash = HASHBYTES('SHA2_256', @OriginalData)
      PRINT 'Data integrity verified.';
  ELSE
      PRINT 'Data has been tampered with.';
  GO
  ```

2. **Expected Output:**
   - Data integrity verified. Should be printed if the data has not been altered.

Output:

```
    COLUMN_NAME     DATA_TYPE

       ------------          ----------

           ID                   int

      OriginalData        nvarchar

      DataHash             varbinary



ID        OriginalData              DataHash

-- --------------- ----------------------------------

 1         Important Data        0A6D3A6E1C5F4A12F8A4F5F6E7D8A9B7C6D7E8E7A9B7D6E8A9
```

Result: