MSSQLTips Free SQL Server Conference Registration + Travel Expenses click to learn more

Understand the Limitations of SQL Server Dynamic Data Masking

Dungeon Fighter Online
The new Demonic Lancer advancements, Impaler and Dragoon are available now.

Free Webcast - Simplify SQL Server Administration

Problem

I recently published an introductory overview of Dynamic Data Masking (DDM), at the time based on SQL Server 2016 CTP 2.1. I tried to emphasize there that this is obfuscation, not encryption, and that there are easy ways to bypass the masking.

I am seeing multiple people with the expectation that DDM will unilaterally secure their data from lower-privileged users. While it's true that DDM can help to mask data from those outside of `sysadmin` and `db_owner` roles (or those explicitly granted `UNMASK`), there will be many ways for users to get around the obfuscation, intentionally or inadvertently.

Solution

To illustrate the problems with trying to use this as a true security feature, let's assume we have the following simple table and user:

```
-- for CTP 2.0 only: DBCC TRACEON(209,219,-1);
CREATE LOGIN peon WITH PASSWORD = 'peon', CHECK_POLICY = OFF;
GO

CREATE DATABASE DDM;
GO

USE DDM;
GO

CREATE USER peon FROM LOGIN peon;
GO

CREATE TABLE dbo.Employees
(
  ID INT PRIMARY KEY,
  Name SYSNAME,
  Salary INT
);
GO

INSERT dbo.Employees(ID, Name, Salary) VALUES
  (1,N'Wayne',100000),
  (2,N'Teemu',75000),
  (3,N'Aaron',20);
GO

ALTER TABLE dbo.Employees ALTER COLUMN Salary
ADD MASKED WITH (FUNCTION = 'default()');
GO

GRANT SELECT ON dbo.Employees TO peon;
```

Now, we've applied a default mask to the `Salary` column; so, we should see 0 for all those rows. And in fact, we do:

```
EXECUTE AS USER = N'peon';
GO
SELECT ID, Name, Salary FROM dbo.Employees;
GO
REVERT;
```

Results:

```
1    Wayne    0
2    Teemu    0
3    Aaron    0
```

In CTP 2.0, all that was necessary to bypass the masking was to convert to a compatible but different data type. For example:

```
EXECUTE AS USER = N'peon';
GO
SELECT ID, Name, CONVERT(CHAR(12), Salary) FROM dbo.Employees;
```

```
GO
REVERT;
```

Results:

```
1    Wayne    100000
2    Teemu    75000
3    Aaron    20
```

In CTP 2.1, they fixed that... you don't get the original masking, but you no longer see the actual values:

```
1    Wayne    xxxx
2    Teemu    xxxx
3    Aaron    xxxx
```

However, there were still other holes; for example, you only had to add an alias (several people assume this involved a join or putting the query in a view, but an alias that doesn't match the original name is all that's needed). Below I am using "e" as the alias for table "Employees" to get around this:

```
EXECUTE AS USER = N'peon';
GO
SELECT ID, Name, Salary FROM dbo.Employees AS e;
GO
REVERT;
```

Results:

```
1    Wayne    100000
2    Teemu    75000
3    Aaron    20
```

In CTP 2.2, they've fixed that one... but there are other holes remaining. This should not be considered an exhaustive list, but all of the following yield similar results (I'm going to stop showing the unmasked results now):

Populate a #temp table, @table variable, or use `SELECT INTO`

```
EXECUTE AS USER = N'peon';
GO
DECLARE @x TABLE(ID INT, Name SYSNAME, Salary INT);
INSERT @x(ID, Name, Salary) SELECT ID, Name, Salary FROM dbo.Employees;
SELECT ID, Name, Salary FROM @x;
GO
REVERT;
GO
```

Run the query from a second database, where the user *is* `db_owner`

First, create a second database and add peon to db_owner:

```
CREATE DATABASE DDM2;
GO
USE DDM2;
GO
CREATE USER peon FROM LOGIN peon;
ALTER ROLE db_owner ADD MEMBER peon;
```

Now, you won't be able to use `EXECUTE AS` directly, but if you establish a new session, log in as peon, connect to the DDM2 database, and run the following query, you'll see the unmasked values:

```
USE DDM2;
GO
SELECT ID, Name, Salary FROM DDM.dbo.Employees;
```

Infer or make deductions about the data based on filters

It's also quite easy to make inferences about the data based on where clauses and join criteria, for example both of these queries return two rows (Wayne and Teemu):

```
EXECUTE AS USER = N'peon';
GO

-- who makes more than $50K?
SELECT ID, Name, Salary FROM dbo.Employees
WHERE Salary > 50000;

-- who makes more then me?
SELECT ID, Name, Salary FROM dbo.Employees
WHERE Salary > (SELECT Salary FROM dbo.Employees WHERE Name = N'Aaron');

GO
REVERT;
```

Long story short

Dynamic Data Masking is not encryption. If you want to protect your data from prying eyes, and you can't prevent those prying eyes from writing their own ad hoc queries, you'll need to use other features, either in addition or as an alternative.

- **Transparent Data Encryption** - Protects the data if, for example, someone steals your backup, but doesn't protect the database running on the physical machine (and the sysadmin and anyone with sufficient access to the certs can get at the data).
- **Always Encrypted** - Protects the data in motion and at rest, and can be set to be decrypted only by the application, meaning it can be protected from the sysadmin as well.
- **Row-Level Security** - Protects prying eyes from seeing the data in specific rows, but there are potentially going to be similar holes here too (I'll take a closer look at this in a future tip).

I'm sure there are a lot of 3rd party tools that might suit your scenario as well. The point is, if you're Target, Home Depot, or Ashley Madison, Dynamic Data Masking is probably not the feature you're going to want to

use to restore trust in your user base (at least, not on its own). The use cases for DDM are:

- You want to shield data from users and can control their data access through queries in application code or stored procedures (no ad hoc query access).

That's it.

Even Microsoft's own documentation explicitly warns you against trying to use DDM to protect your data from users with ad hoc query rights:

> *The purpose of dynamic data masking is to limit exposure of sensitive data, preventing users who should not have access to the data from viewing it. Dynamic data masking does not aim to prevent database users from connecting directly to the database and running exhaustive queries that expose pieces of the sensitive data. Dynamic data masking is complementary to other SQL Server security features (auditing, encryption, row level security.:.) and it is highly recommended to use this feature in conjunction with them in addition in order to better protect the sensitive data in the database.*

Summary

Dynamic data masking provides a simple way to implement obfuscation of private data (and in my tests so far, the performance impact is negligible). It is important to understand the limitations, and keep in mind that it is not true encryption and that the data cannot be protected in all scenarios. In another tip I will talk about a more end-to-end way to protect sensitive data, even from the sysadmin role, in SQL Server 2016: Always Encrypted.

Next Steps

- Download the latest SQL Server 2016 CTP (or register for a trial of Azure SQL Database v12, where this feature appeared first).
- Try out Dynamic Data Masking in scenarios where it may seem useful.
- See these other security-related tips:
  - Use Dynamic Data Masking in SQL Server 2016 CTP 2.1
  - SQL Server Column Encryption
  - SQL Server Column Level Encryption Example using Symmetric Keys
  - Masking Personal Identifiable SQL Server Data
  - Natively Encrypting Social Security Numbers in SQL Server 2005
  - Identifying PII Data to Lock Down in SQL Server (Part 1 of 2)
  - Locking Down PII Data in SQL Server (Part 2 of 2)
  - SQL Server 2008 Transparent Data Encryption getting started
  - All Encryption Tips

Last Update: 2015-08-20

You're invited
to our next free
SQL Server webcast!

click to learn more

Next Tip

Click Here

About the author

Aaron Bertrand is a Senior Consultant at SQL Sentry, Inc., and has been contributing to the community for **about two decades**, first earning the Microsoft MVP award in 1997.

View all my tips

Related Resources

- More SQL Server DBA Tips...

Post a comment or let the author know this tip helped.

All comments are reviewed, so stay on subject or we may delete your comment. Note: your email address is not published. Required fields are marked with an asterisk (*).

*Name *Email Notify for updates

*** NOTE *** - If you want to include code from SQL Server Management Studio (SSMS) in your post, please copy the code from SSMS and paste the code into a text editor like NotePad before copying the code below to remove the SSMS formatting.

p

Send me SQL tips:

I'm not a robot

reCAPTCHA
Privacy - Terms

Save Comment

Friday, August 11, 2017 - 10:12:24 AM - Stéfano Back To Top

Nice tips.

Is there any way to deny columns for sysadmin users ?

Wednesday, June 08, 2016 - 6:13:58 AM - Perry Whittle Back To Top

Hi Aaron

very informative article, the feature is still fairly new and in it's infancy, I'm sure this will become a more seasoned feature in future versions

Regards Perry