

SQL Server 安全执行上下文和代码签名

SQL Server 决定主体是否有必要的执行代码权限的根本途径是其执行上下文规则。这一切都可能复杂一个主体有执行代码的权限，但是却没有访问代码中基础对象的权限，比如表中的数据。

执行上下文

当用户执行一个存储过程或其他数据库代码，SQL Server 检查以确保用户不但有执行过程权限，而且有代码访问的数据库对象的权限。如果没有这种权限检查，有人可以容易地创建读取表并使用其他对象的代码，用户没有访问权限却能执行该代码。这将导致一个重大的安全漏洞。

此权限检查过程的一个例外是当代码所有者同时是代码访问/涉及的所有相关对象的所有者。在这种共同所有权的情况下，SQL Server 只需验证调用者有执行代码权限，而不需继续检查访问对象的权限。

例如，如果存储过程中的代码访问三张表和四个视图，SQL Server 执行代码之前先执行下面步骤：

- 1、验证调用者对存储过程有执行权限。如果调用者没有，返回一个错误并且不执行该代码
- 2、检查代码的所有者是否拥有代码访问的所有对象。如果存在共同所有权，则停止检查权限并执行代码
- 3、如果共同所有权不存在，检查以确保调用者具有对代码访问的对象有权限。如果调用者对任何一个或多个对象没有权限，返回一个错误，并且不执行该代码

4、如果调用者有所有必要的权限，执行该代码；否则，返回一个错误，并且不执行该代码

This process of common ownership checking continues in the case where code calls other code or accesses other objects, which in turn calls other code or accesses other objects. 只有链中所有的对象都具有相同的所有者，就不需要进行权限检查。但只要访问中有一个对象在链中有不同的所有者，就要对该对象进行权限检查。

在这种情况下，对象的所有权称为所有权链接，缺少共同拥有权被称为一个断裂的所有权链接。这是理解 SQL Server 安全的一个非常强大的概念，允许 SQL Server 绕过不必要的权限验证来提高性能。

编写一个连续的所有权链接的代码通常更容易，因为你不必担心代码执行的安全上下文。这就是为什么在早期版本的 SQL Server 中有一个特殊的 dbo 角色拥有所有的数据库对象。但任何时候，你拥有共同所有权和权限访问一切，你违反了最小特权原则，并暴露你的数据给不必要的安全风险。

幸运的是，在 SQL Server 可以更改代码的安全执行上下文

更改执行上下文

你不总是希望在断裂的所有权链接下使用调用者的权限来验证权限。有时你想执行代码，好像它完全被另一个用户执行样，通过使用其他用户的权限来验证所有访问对象的权限。这就是所谓的转换执行上下文。这让你利用 SQL Server 的细粒度的权限，在基础对象严格控制权限，同时还让不同的用户执行代码的能力。

在 SQL Server 中，当你定义任何用户定义的函数(除内联表值函数)、存储过程、触发器，你可以使用 EXECUTE AS 子句为该对象定义，表示代码应该运行在指定用户的安全上下文。

T-SQL 提供四种 EXECUTE AS 选项：

EXECUTE AS CALLER : 向后兼容的默认值。代码在调用者的上下文中执行,他必须具有执行该代码并访问底层对象的权限。实际行为取决于所有权链接断裂或连续。

EXECUTE AS = 'username' and EXECUTE AS = 'loginname' :代码在指定的用户或登录的上下文中执行,因此指定的用户或登录必须具有所有相关对象的权限。在这种情况下,调用者必须满足任一条件:

- >具有执行代码的权限

- >是 sysadmin 或 db_owner,或在服务器或数据库有控制权限,或对指定用户有模拟权限

EXECUTE AS 登录名选项只适用于服务器范围 DDL 触发器和登录触发器。否则,提供的名称必须是有效的数据库用户名

EXECUTE AS SELF : 这是创建过程的用户的快捷方式。这是相当于 EXECUTE=[myusername]。SQL Server 目录存储了编写代码的实际用户 ID。

EXECUTE AS OWNER : 这是在特定用户的安全上下文下执行的另一个变化,in this case the owner of the code at the time of execution, not at the time of creation. 如果在数据库中创建代码的所有者更改过,这意味着该代码将在与第一次创建代码的原始用户不同的用户权限下执行。

当你在 SSMS 中运行代码,在一个会话中执行上下文的语句 EXECUTE AS 有两种变体, EXECUTE AS LOGIN= 'loginname'和 EXECUTE AS USER='username'。当一个用户登录到数据库实例开始一个会话,当时的执行上下文设置为登录用户以便进行权限检查。EXECUTE AS 会更改会话的执行上下文直到会话结束或使用 REVERT 语句恢复。

任何时候通过 EXECUTE AS 更改安全上下文，代码的创建者或会话用户必需对子句中的特定用户有模拟权限。如果是 EXECUTE AS SELF 则不需要有这个权限来模拟你自己。

使用 EXECUTE AS 子句

假设你在数据库中有一个 Vendor 表。这个表是定义在 SchemaUserTable 架构下，架构的所有者是 UserTable。代码 6.1 中创建一个访问该表的存储过程。该存储过程是在 SchemaUserProc 架构下，架构所有者是 UserProc。因为表和存储过程被定义在不同的架构中被不同的用户所拥有，存在断裂的所有权链接。

```
USE master;

GO

IF SUSER_SID('UserProc') IS NOT NULL DROP LOGIN UserProc;

IF SUSER_SID('UserTable') IS NOT NULL DROP LOGIN UserTable;

IF SUSER_SID('RealUser') IS NOT NULL DROP LOGIN RealUser;

GO

--Create the login

CREATE LOGIN UserProc WITH password = 'Y&2!@37z#F!11zB';

CREATE LOGIN UserTable WITH password = 'Y&2!@37z#F!11zB';

CREATE LOGIN RealUser WITH password = 'Y&2!@37z#F!11zB';

GO

--Create the database

IF DB_ID('ExecuteContextDB') IS NOT NULL DROP DATABASE ExecuteContextDB;
```

```

CREATE DATABASE ExecuteContextDB;

GO

USE ExecuteContextDB;

GO

--Create the users

CREATE USER UserProc;

CREATE USER UserTable;

CREATE USER RealUser;

GO

--Create the schemas

CREATE SCHEMA SchemaUserProc AUTHORIZATION UserProc;

GO

CREATE SCHEMA SchemaUserTable AUTHORIZATION UserTable;

GO

--Create a table and a proc in different schemas to ensure that there is no ownership chaining.

CREATE TABLE SchemaUserTable.Vendor

    (ID INT, name VARCHAR(50), state CHAR(2), phno CHAR

(12));

GO

SET NOCOUNT ON

GO

INSERT INTO SchemaUserTable.Vendor VALUES (1,'Vendor1',

'AK','123-345-1232');

INSERT INTO SchemaUserTable.Vendor VALUES (2,'Vendor2',

'WA','454-765-3233');

INSERT INTO SchemaUserTable.Vendor VALUES (3,'Vendor3',

'OR','345-776-3433');

INSERT INTO SchemaUserTable.Vendor VALUES (4,'Vendor4',

'AK','232-454-5654');

```

```

INSERT INTO SchemaUserTable.Vendor VALUES (5,'Vendor5',
'OR','454-545-5654');

INSERT INTO SchemaUserTable.Vendor VALUES (6,'Vendor6',
'HI','232-655-1232');

INSERT INTO SchemaUserTable.Vendor VALUES (7,'Vendor7',
'HI','453-454-1232');

INSERT INTO SchemaUserTable.Vendor VALUES (8,'Vendor8',
'WA','555-654-1232');

INSERT INTO SchemaUserTable.Vendor VALUES (9,'Vendor9',
'AK','555-345-1232');

GO

--Create the stored procedure in SchemaUserProc

CREATE PROC SchemaUserProc.VendorAccessProc @state CHAR
(2)
AS

    SELECT * FROM SchemaUserTable.Vendor WHERE state = @
state;

GO

```

代码 6.1 在一个架构中创建存储过程访问另一个架构中的表 ,两个架构有不同的所有者

将过程的执行权限授予给 RealUser 用户 , 如代码 6.2 所示 :

```

GRANT EXECUTE ON SchemaUserProc.VendorAccessProc TO RealUser;

```

代码 6.2 授予用户执行存储过程的权限

在 SSMS , 你可以在代码执行的查询窗口运行 EXECUTE AS 以临时更改安全上下文。使用代码 6.3 中更改安全上下文到 RealUser 然后运行存储过程来获取数据。

```

--Now try and access the proc as RealUser

```

```
EXECUTE AS USER = 'RealUser';  
  
EXEC SchemaUserProc.VendorAccessProc 'AK';
```

代码 6.3 切换执行上下文

执行上面代码抛出下面错误信息：

消息 229，级别 14，状态 5，过程 VendorAccessProc，第 4 行
拒绝了对对象 'Vendor' (数据库 'ExecuteContextDB'，架构 'SchemaUserTable') 的 SELECT 权限。

这个问题是因为所有权链接断裂，存储过程的所有者与表的所有者不同，并且 RealUser 用户在 Vendor 表上没有查询权限。下面是 SQL Server 从概念上分析步骤：

- 1、RealUser 是调用者，它有对过程的执行权限
- 2、过程的所有者是 UserProc，表的所有者是 UserTable。这表示所有权链接断裂，因此需要检验调用者 RealUser 是否有权限执行代码中的操作
- 3、RealUser 在 Vendor 表上没有查询权限，因此抛出错误，执行失败

你可以在存储过程的定义中使用 EXECUTE AS 子句来解决这个问题，假设你是存储过程的创建者，希望让 RealUser 在这种情况下执行代码。首先，使用代码 6.4 中的 REVERT 语句撤销 RealUser 安全上下文，返回到你自己的安全上下文：

```
REVERT;
```

代码 6.4 返回到原始安全上下文

接着修改存储过程，添加 EXECUTE AS 子句在 UserTable 安全上下文运行存储过程，此用户在表上有查询权限，如代码 6.5 所示：

```

ALTER PROC SchemaUserProc.VendorAccessProc @state CHAR
(2)

WITH EXECUTE AS 'UserTable'

AS

        SELECT * FROM SchemaUserTable.Vendor WHERE state = @
state;

GO

```

代码 6.5 修改存储过程使用 EXECUTE AS 在执行时切换执行上下文

提示：在本例中，UserTable 在 Vendor 表上有查询权限，UserTable 是架构 SchemaUserTable 的所有者。

修改执行上下文到 RealUser，然后重新运行存储过程，如代码 6.6:

```

EXECUTE AS USER = 'RealUser';

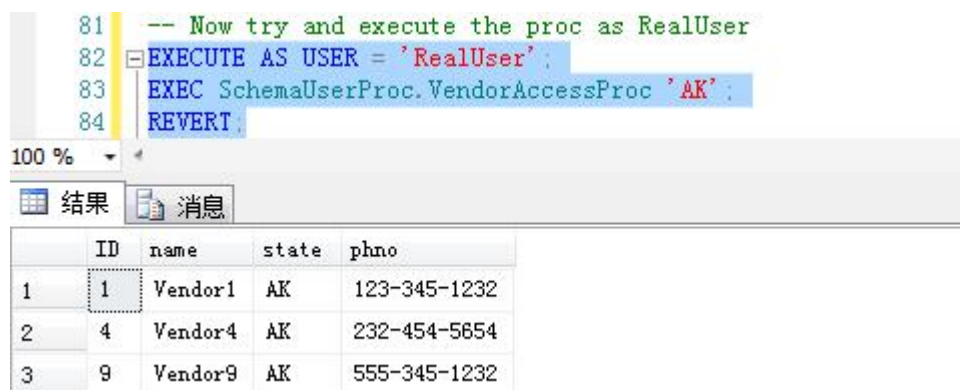
EXEC SchemaUserProc.VendorAccessProc 'AK';

REVERT;

```

代码 6.6 测试更新后的存储过程是否能被 RealUser 执行

这次执行成功，因为 SQL Server 检查所有权链接权限时，虽然还是断裂，但 UserTable 有必要的查询权限。执行结果如图 6.1 所示：



	ID	name	state	phno
1	1	Vendor1	AK	123-345-1232
2	4	Vendor4	AK	232-454-5654
3	9	Vendor9	AK	555-345-1232

图 6.1 在不同的用户安全上下文运行存储过程的结果

代码签名

一组 T-SQL 代码使用 EXECUTE AS 子句改变执行上下文只是一种避开断裂所有权链接问题的方法。另一个选择是使用证书或非对称密钥签名代码。此技术授予对代码本身的权限，而不是要求你更改执行上下文或依赖于调用者的权限。通过仔细地控制证书或非对称密钥的使用，你仍然可以控制哪些主体可以利用权限来执行代码。

该方法的方式是，你创建一个安全、加密证书或非对称密钥，然后创建一个与证书或密钥关联的用户。这是一个特殊类型的用户，它们没有关联登录名。你分配给用户执行存储过程所需的权限，然后使用 ADD SIGNATURE 语句将证书或密钥分配给存储过程。这个存储过程在与证书或密钥关联的用户安全权限中运行。

即使存储过程使用 EXECUTE AS 更改执行上下文，你也可以使用该技术。代码签名的一种常见方案是将执行上下文更改为具有大多数该代码需要执行的权限的用户，然后使用代码签名添加一个或多个附加权限。

像往常一样，举个例子助于理解这种技术。代码 6.7 创建了两个存储过程，从 ExecuteContextDB 数据库中的 Vendor 表中检索数据。UnsignedProc 过程不会签名，所以当 RealUser 执行时会失败。SignedProc 过程将签名，所以 RealUser 将正确执行。

```
CREATE PROC SchemaUserProc.UnsignedProc @state CHAR(2)
AS
    SELECT * FROM SchemaUserTable.Vendor WHERE state = @
state;
GO
CREATE PROC SchemaUserProc.SignedProc @state CHAR(2)
```

```

AS
    SELECT * FROM SchemaUserTable.Vendor WHERE state = @
state;

GO

--授予执行权限

GRANT EXECUTE ON SchemaUserProc.UnsignedProc TO RealUse
r;

GRANT EXECUTE ON SchemaUserProc.SignedProc TO RealUser;

GO

```

代码 6.7 创建相同的存储过程并授予执行权限给 RealUser

但这一次，不是通过改变执行上下文，我们将创建一个证书，如代码 6.8 所示。然后从该证书中创建用户，并授予该用户在 Vendor 表上的选择权限。最后，使用添加签名声明证书添加到 SignedProc 存储过程。注意，只有 SignedProc 有签名；UnsignedProc 仍未签名。

```

CREATE CERTIFICATE MyCertificate

    ENCRYPTION BY PASSWORD = 'SZ6T4O^ff&lKr3s?m\*'

    WITH SUBJECT = 'Certificate to sign SignedProc';

GO

CREATE USER MyCertificateUser

    FROM CERTIFICATE MyCertificate;

GO

GRANT SELECT ON SchemaUserTable.Vendor TO MyCertificate
User;

GO

ADD SIGNATURE TO SchemaUserProc.SignedProc BY CERTIFICA
TE MyCertificate

    WITH PASSWORD = 'SZ6T4O^ff&lKr3s?m\*';

```

代码 6.8 实施证书分配权限

最后，是测试这个代码签名方案的时候了，如代码 6.9 所示。结果如图 6.2 所示。UnsignedProc 存在断裂的所有权链接，并且 RealUser 没有对 Vendor 表的查询权限，所以执行失败。SignedProc 通过代码签名授予 SELECT 权限，执行成功返回三条记录。

```
--Test the code signing scheme

EXECUTE AS USER = 'RealUser';

--Can't run UnsignedProc

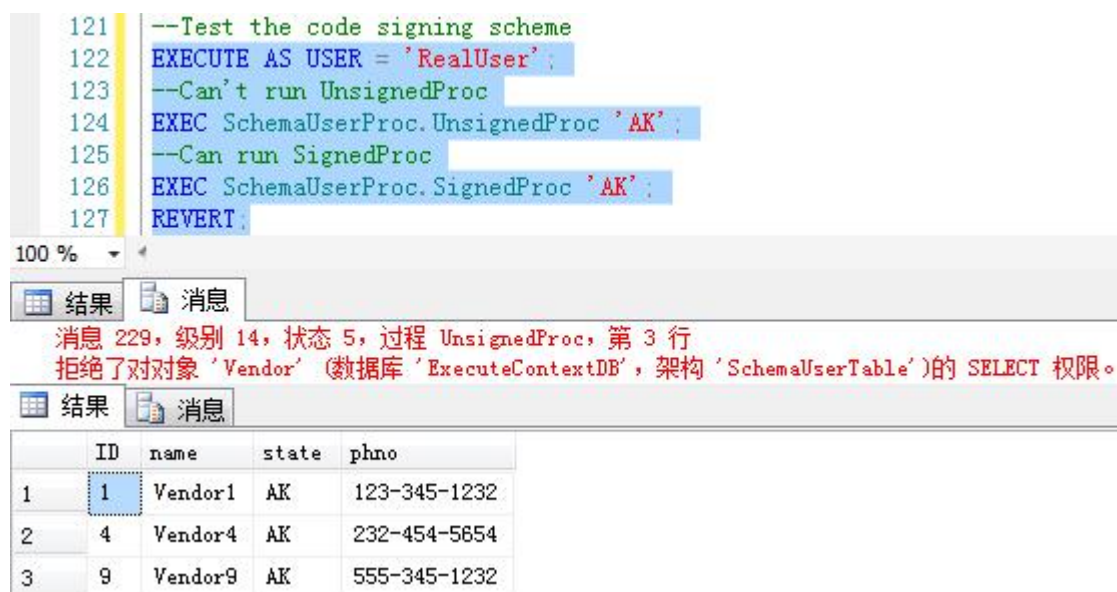
EXEC SchemaUserProc.UnsignedProc 'AK';

--Can run SignedProc

EXEC SchemaUserProc.SignedProc 'AK';

REVERT;
```

代码 6.9 测试签名证书



The screenshot shows a SQL query window with the following code:

```
121 --Test the code signing scheme
122 EXECUTE AS USER = 'RealUser';
123 --Can't run UnsignedProc
124 EXEC SchemaUserProc.UnsignedProc 'AK';
125 --Can run SignedProc
126 EXEC SchemaUserProc.SignedProc 'AK';
127 REVERT;
```

Below the query window, the 'Messages' tab is selected, showing the following error message:

消息 229, 级别 14, 状态 5, 过程 UnsignedProc, 第 3 行
拒绝了对对象 'Vendor' (数据库 'ExecuteContextDB', 架构 'SchemaUserTable') 的 SELECT 权限。

Below the error message, the 'Results' tab is selected, showing the following table:

	ID	name	state	phno
1	1	Vendor1	AK	123-345-1232
2	4	Vendor4	AK	232-454-5654
3	9	Vendor9	AK	555-345-1232

图 6.2 代码签名测试

设置所有起来有点复杂，但它是非常有价值的安全效益。正确地做了，该技术消除了用户需要对基础对象的选择权限，用户只需在存储过程上有执行权限就够。它可能不是你在存储过程或用户定义函数中广泛使用的东西，但它很好地解决了安全问题，在处理断裂所有链接时，没有一个主体具有所有所需的权限。