Global Information Assurance Certification Paper

SQL Server Security

Stephen V. Arehart

Introduction

SQL Server is Microsoft's enterprise -level data access and data storage solution. Microsoft SQL Server is now at version 7.00.846, and has established a presence in the database market. Additionally, the tight integration between SQL Server, Microsoft Windows NT Server, and Microsoft Internet Information Server have made this combination a common platform for deploying web-enabled, data-driven applications. However, as more customers demand both ease of use and privacy of internet web sites running these dynamic applications, database security becomes an ever- increasing priority. In this article, we look at several security aspects of SQL Server, including:

- Security Facilities in SQL Server
- Existing SQL Server Vulnerabilities
- Hardening the default install of SQL Server
- Auditing SQL Server

This paper deals *exclusively* with SQL Server 7.0. SQL Server 6.5 and SQL Server 2000 are *not* covered.

Security Facilities in SQL Server

SQL Server has several facilities that can be used to control access to every object within a database, including multiple authentication modes, logins, users, roles, and object permissions.

Multiple Authentication Modes

SQL Server provides two authentication modes: **Mixed Mode** and **Windows NT Mode**. Mixed mode authenticates connections to the database through the *sysxlogins* table in the master database of an SQL server installation. [1,2,3] When SQL Server authenticates connections in mixed mode, a user wishing to connect to the database must have a valid login *and password* to the SQL Server installation running on the target machine. SQL Server checks the user-supplied password with the password stored in the *sysxlogins* table to make sure the user-supplied password is the correct one. Microsoft provided this mode primarily for backwards compatibility with older SQL Server installations and applications, and does not recommend this mode be used for authentication.

In contrast, Windows NT mode uses "Windows NT-based facilities" [1,2,3] to authenticate the user. For a user to be authenticated, they must have a valid Windows NT user account. However, the NT password is *not* stored in the database. According to Microsoft [4] Windows NT mode is more secure.

Logins, Users, Permissions, and Roles

SQL Server provides several mechanisms to control access to objects within the SQL Server database. Server *logins* are similar in purpose to Windows NT logins - they control who can authenticate to the server to gain access. SQL Server *users* are used to determine if an individual or group that has authenticated to the server (via a valid *login*) is authorized to access a particular database. Finally, the ability to `SELECT`, `INSERT`, `UPDATE`, `DELETE` and perform other actions on objects is determined through *object permissions*. That is, when a user decides to delete all entries in the *sysusers* table, SQL Server checks to make sure that the user actually has that right by consulting the *syspermissions* table to see if the user has the `DELETE` right on the *sysusers* table.

SQL Server also provides *roles*. "Roles allow users to be collected into a single unit against which permissions can be applied", [2] and are analogous to a Windows NT group. SQL Server provides a default set of roles for each database which satisfy the access needs of most situations, though it is possible to make custom roles for specific needs. Roles are not strictly necessary - it is possible to assign database permissions on a per-user basis. However, roles make managing object permissions much easier to do.

Existing SQL Server Vulnerabilities

Security vulnerabilities exist in SQL Server, and many of them relate to using the mixed-mode authentication model. Though a complete listing of vulnerabilities is beyond the scope of this article, it is instructive to examine some of the vulnerabilities that are known, and classify those vulnerabilities in terms of confidentiality attacks, availability attacks, and integrity attacks, since "these are the mechanisms by which threats are manifested."[5]

Confidentiality Attacks

Confidentiality attacks expose data that should not be exposed - the intruder sees what they should not see. Microsoft SQL Server has several confidentiality attacks, including:

- MS00-35. In this vulnerability, it is possible to find the 'sa' password for the sql server installation by inspecting temporary files created during service pack installation. An intruder could potentially use this to gain *administrative* access to SQL Server.
- MS00-041. Passwords can be obtained through DTS packages or through the SQL Server registration dialog.
- ISS Password Vulnerability. It is possible to extract passwords from the NTUSER.DAT file, and in particular, extract the "encrypted" password for an SQL Server installation.

All of these vulnerabilities are avoided if SQL Server uses *only* the Windows NT mode of authentication.

Availability Attacks

Attackers can affect the ability of a target system to reliably and timely provide user services, resulting in an availability attack:

- In MS99-059, it is possible to send malformed TDS packets to an SQL Server installation that can cause the system to crash.

Integrity Attacks

Attackers can subvert a trust relationship or alter data on a target system, producing an integrity attack:

- MS00-014, it is possible for a standard user to take advantage of poor input checking to bypass the SQL Server security model and execute arbitrary commands.

Hardening SQL Server Default Installation

As with most software products, the default installation of SQL Server is relatively relaxed. Mixed mode authentication, blank administrative passwords, and the "public" role having access to a number of stored procedures are examples of default settings that increase system usability but decrease system security. Listed below are steps one can take when installing and configuring an SQL Server installation to increase system security. Please note that this list is not exhaustive, and it is entirely possible that the there are additional settings one can configure to strengthen system security. Additionally, these changes should be tested in a development environment *before* deploying to production servers.

- **Physical Server Security** As always, all OS and software security is useless if someone can reboot the system with a floppy and read all the data off the computer with file system drivers. Make sure that appropriate physical security measures are applied for the system on which SQL server will reside.
- **Underlying OS Security** Since SQL Server runs on Windows NT, and uses some of NT's security features in the Windows NT security mode, it is advantageous to secure the base OS before securing the database server. Those who have completed the SANS GIAC LevelOne training received a free copy of "Securing Windows NT - Step By Step". We recommend this as a resource for securing the underlying OS.
- **Install SQL Server on a drive with NTFS.** When installing SQL Server, make sure all relevant files (.exes, .dlls, data files) are placed on a device that uses NTFS as the file system. Doing this allows the ability to apply appropriate access controls to those objects and audit those objects for illegitimate access.
- **Run services under a predetermined account.** As with all NT services, the services for SQL Server run within the security context of a chosen server (or domain)account. Choose this account wisely, give it a good password, and use NT ACLs to control what resources this account can access.
- **Change the 'sa' password(!)**. Inside SQL server, the 'sa' account can do everything. On installation, this password defaults to NULL, and this default has been exploited in a variety of attacks on SQL Server. Make sure that this password is long, follows good password creation practices, and is shared among very few people (if it is shared at all).
- **Once installation is complete, apply appropriate service packs.** As of this writing, SQL Server is at service pack 2. Make sure that when you apply service packs, you use the NT authentication mode to keep passwords from ending up in plaintext in temporary files.
- **Remove problematic default accounts and extended stored procedures**. By default, the guest account exists and is a member of the public role in all databases except the *model* database. We strongly suggest disabling the guest

account on NT, and removing the guest user from the *master*, *tempdb*, and *msdb* databases.

- **Drop all sample databases**. By default, the *Northwind* and *pubs* sample databases are created on install. Defaults on systems invariably have security-related problems, and we advise that these two default databases be removed.
- **Realize who ends up in the database by default**. On install, by default, the **BUILTIN\Administrators** NT group ends up as a login in the server. Make sure that this is what is desired.
- **Choose an appropriate network library**. SQL Server supports a number of protocol libraries for communication between clients and servers. Multiprotocol may be a good choice due to encryption and random TCP port assignment, but TCP/IP may be better in heterogeneous environments. SQLSecurity.com has a nice review of the strengths and weaknesses associated with each protocol.
- **For TCP/IP library, change default port**. When using TCP/IP as the network protocol, SQL Server listens on port 1433 by default. Changing this value is security by obscurity, but may prevent casual resource browsing and thwart the efforts of script-kiddies.
- **Drop problematic stored procedures**. SqlSecurity.com lists (among many other things) a number of stored procedures which could potentially be a problem. Examine this list, and explore the impact of dropping these stored procedures. However, we do recommend that the `xp_cmdshell` stored procedure be dropped.
- **Secure the MSSQL\ directories with NTFS permissions**. Make sure that the user selected for running services be able to have full control on these directories and subdirectories.
- **Use Windows NT Security**. Most current exploits are based on weaknesses within the mixed mode security mechanism. Use the NT mechanism. In addition to being safer, it is possible to apply all the NT password rules (including custom third-party password filters and account lockout policies) to SQL server as well, since authentication to SQL Server occurs through NT security facilities.

Auditing SQL Server

Once a system has been brought to a consistent and known state, it is important to periodically audit the system to make sure that the state is not altered. SQL Server uses the native OS logging facilities (the Event Viewer) and additional log files (found in MSSQL7\LOG in a default install) to log various events, which can be useful for auditing. Additionally, the SQL Server Profiler tool that comes with SQL Server can be used to track object access within the database. This can also be used for auditing[2].

However, there can be situations where auditing the database with these tools is either unacceptable or not possible. In this case, it is useful to develop SQL scripts that can be used to monitor system activity. For instance, to audit SQL Server on a server-wide level, one may want to:

- Check the version of the server to make sure appropriate service packs have been applied.
- Check for default databases on the server that may not be desirable.
- Check for logins with NULL passwords.
- Check for the last set of database backups.

Consider the following SQL script which would audit for those events:

```
USE master
GO

/* check to make sure they're up to the latest version
 */
SELECT @@VERSION

/* check to see if the "northwind" or "pubs" database exist.
 * if they do, warn the user to be proactive and remove
 * the default databases.
 */

IF (EXISTS (SELECT dbid from sysdatabases where name = 'pubs'))
PRINT "It looks like you have not removed the default database 'pubs'
from your system. You may want to remove this database."
PRINT ""

/* check for logins with null passwords or
 * existence of a 'guest' account
 */
select (substring(loginname,1,32))as loginname
from syslogins
where password is NULL
or loginname = 'guest'

/* determine if and when database backups have been performed
 */

select (substring(database_name,1,32)) as database_name,
backup_finish_date from msdb.dbo.backupset
```

This script could be placed on the server and executed interactively via the Query Analyzer, or in a batch process via the `osql` command-line utility, which could be used with the Schedule service to automatically execute on a regular schedule. Alternatively, a job could be scheduled in SQL Server that executes these sql commands and emails an administrator the results.

To aid SQL Server administrators, we have prepared several SQL server scripts that can be used to audit SQL Server at the server and database levels. Furthermore, these scripts can be used to obtain a "snapshot" of the database as it exists at some point, or they can be used to track what has changed within a set time frame. There are four scripts, and they are included in Appendix I of this document. We have included a number of comments in each script to explain what is being done and what base objects are being accessed. These scripts could serve as a basis for an auditing system. For instance, using

these scripts, along with Perl, the Perl DBI, and nmap could allow for dynamic detection of sql servers on a given network, followed by an audit of each of those servers. The possibilities are endless.

Conclusions

SQL Server has several facilities that can be used to control authentication to an SQL Server, and access to data within each database hosted on that server. There are known confidentiality, integrity, and availability attacks on SQL Server, and many of them capitalize on the weaker "mixed mode" authentication scheme. The default installation of SQL Server is weak, but there steps one can take to harden the installation. It is also possible to audit SQL server using several tools (Event Viewer, Profiler and custom SQL scripts) to monitor the state of the database. Several SQL scripts which can be used to audit the server are provided.

References

1. Microsoft Press, "SQL Server Books Online", 1998.

2. Microsoft Corporation, "Microsoft SQL Server Security", 1999.
URL: http://www.microsoft.com/sql/techinfo/securityWP.htm

3. Lewis, Morris "Creating a Manageable Security Plan", 1999.
URL: http://www.sqlmag.com/Articles/Index.cfm?ArticleID=15446

4. Microsoft Corporation, "Microsoft Security Bulletin (MS00-035): Frequently Asked Questions", 2000.
URL: http://www.microsoft.com/technet/security/bulletin/fq00-035.asp

5. Northcutt, Stephen. "Information Assurance Foundations. Core Issues and Challenges" (SANS GIAC LevelOne Seminar PDF) 2000, Version 1.3.

Appendix I - SQL Server Audit Scripts

```
/* SQL Server 7.0 Server Audit Script
 * Stephen V. Arehart
 *
 * This script audits the state of an SQL Server installation. It can
be
```

```
 * used to obtain a baseline configuration against which subsequent
audit
 * data can be compared.
 *
 * The script:
 *    checks the version of the database to make sure it has the
appropriate
 *    patches.
 *    generates a list of all databases on the server and warns if any
 *    of the default databases still exist.
 *    checks for logins with NULL passwords that aren't NT user or
group logins.
 *    checks for a guest login.
 *    determines when databases have been backed up.
 *    prints out the most recent SQL Server error log.
 */


SET NOCOUNT ON
GO
USE master
GO

DECLARE @sql_server_version VARCHAR(8)

SET @sql_server_version = '7.00.842'


PRINT "SQL Server 7.0 Audit Script"
PRINT "by Stephen V. Arehart"

PRINT "Performing server audit on " + @@SERVERNAME + "."
PRINT ""


/* Check to make sure they're up to the latest version.
 */
PRINT "As of Service Pack 2, part of @@VERSION should provide"
PRINT "the explicit version of the server as " + @sql_server_version +
"."
PRINT "If this is not the version you see below"
PRINT "this system *SHOULD* be patched."
PRINT "Here's the version of this SQL Server:"

SELECT @@VERSION

/* Get a list of all the databases on the system.
 */

PRINT "The following databases exist on this system:"
PRINT ""

SELECT SUBSTRING(name,1,30) AS database_name
  FROM sysdatabases

/* Check to see if the "northwind" or "pubs" database exist.
 * if they do, warn the user to be proactive and remove
```

```
 * the default databases.
 */

IF (EXISTS (SELECT dbid FROM sysdatabases WHERE name = 'pubs'))
PRINT "It looks like you have not removed the default databases 'pubs'
from
your system. You may want to do that."
PRINT ""

IF (EXISTS (SELECT dbid FROM sysdatabases WHERE name = 'Northwind'))
PRINT "It looks like you have not removed the default database
'Northwind' from
your system. You may want to do that."
PRINT ""

/* Check for logins with null passwords or
 * existence of a 'guest' account
 * (which would be bad).
 * note that NT logins will have NULL passwords, since
 * NT passwords are not stored in the database.
 */

PRINT "The following logins have null passwords"
PRINT "and ARE NOT NT logins:"
print ""
SELECT (SUBSTRING(loginname,1,32))AS loginname
  FROM syslogins
 WHERE password is NULL
   AND isntgroup = 0
   AND isntuser = 0
    OR loginname = 'guest'

/* Determine if and when database backups have been performed.
 */

PRINT "The following database backups have occurred:"
SELECT (SUBSTRING(database_name,1,32)) AS database_name,
        backup_finish_date
  FROM msdb.dbo.backupset

/* Output the most recent error log.
 */

PRINT "This is the most recent SQL Server Error Log"
PRINT ""
PRINT ""
EXEC sp_readerrorlog


/* SQL Server 7.0 Database Audit Script
 *
 * Stephen V. Arehart
 * This script can be used for a baseline audit of a given database.
Data
 * collected here can be stored and compared to subsequent audits.
 *
 * The script:
```

```
 *    generates a list of users in the database
 *    generates a list of roles in the database
 *    determines which stored procedures are executable and by which
user or
 *    role.
 *
 */

SET NOCOUNT ON
GO

DECLARE @database VARCHAR(50)

SET @database = DB_NAME()

PRINT "SQL Server 7.0 Audit Script"
PRINT "by Stephen V. Arehart"
PRINT "Performing a database audit on  the '" + @database + "'
database."
PRINT ""

/* Get a list of all the users in this database
 * from the sysusers system table.
 */

PRINT "The following users exist in this database:"
PRINT ""

SELECT SUBSTRING(name,1,30) AS name
  FROM sysusers
 WHERE issqluser =1
    OR isntuser = 1
    OR isntgroup = 1

/* Get all the roles in the current database.
 */
PRINT "The following user roles exist in this database:"
PRINT ""

SELECT SUBSTRING(name,1,30) AS role_name
  FROM sysusers
 WHERE issqlrole = 1

/* Figure out who has access to stored procedures. According
 * to Microsoft, all the permission information is actually
 * stored in the syspermissions table, but it's not very-well
 * documented, so I'm using the sysprotects view.
 */

PRINT "The following stored procedures are executable (action = 224)"
PRINT "by the people listed in user_name:"
PRINT ""

SELECT SUBSTRING(sysobjects.name,1,30) AS procedure_name,
       SUBSTRING(sysusers.name,1,30) AS user_name
  FROM sysobjects, sysprotects, sysusers
 WHERE sysobjects.id = sysprotects.id
```

```
   AND sysusers.uid = sysprotects.uid
   AND sysobjects.xtype = 'P'
   AND sysprotects.action = 224


/* SQL Server 7.0 Differential Server Audit Script
 *
 * Stephen V. Arehart
 * This script can be used to audit for changes that occured over a
 * configurable time period within the SQL Server installation.
 *
 * The script checks for the following:
 *    creation of new server logins.
 *    modifications to existing server logins.
 *    creation of new DTS packages on the server.
 *    execution of jobs on the server.
 *
 * This script could be used in conjunction with the system or SQL
Server
 * scheduler to run intermittently and email or log the results.
 */


SET NOCOUNT ON
GO

USE master
GO

/* These are used to determine various time intervals over which events
are
 * audited. For customized time windows, assign appropriate values to
 * each of these variables.
 */

DECLARE @time_interval           INT
DECLARE @login_creation_interval INT
DECLARE @login_update_interval   INT
DECLARE @dts_creation_interval   INT
DECLARE @job_execution_interval  INT

SET @time_interval = 24
SET @login_creation_interval = @time_interval
SET @login_update_interval = @time_interval
SET @dts_creation_interval = @time_interval
SET @job_execution_interval = @time_interval

PRINT "SQL Server 7.0 Audit Script"
PRINT "by Stephen V. Arehart"
PRINT "Performing a differential audit on server '" + @@SERVERNAME + "'
."
PRINT ""

/* Check for recently created logins.
 * The time over which logins have been created can be adjusted
 * by increasing the @login_creation_interval to a larger value.
 */
```

```
PRINT "The following logins have been created within the last " +
CONVERT(VARCHAR(10),@login_creation_interval) + " hours:"
PRINT ""
SELECT (SUBSTRING(loginname,1,30)) AS login_name, createdate AS
create_date
  FROM syslogins
 WHERE DATEDIFF(hour,createdate,GETDATE()) < @login_creation_interval

/* Check for logins that have been modified with the past
 * @login_update_interval hours.
 */

PRINT "The following logins have been modified within the past " +
CONVERT(VARCHAR(10),@login_update_interval) + " hours:"
PRINT""
SELECT SUBSTRING(loginname,1,30) AS login_name, updatedate as
update_date
  FROM syslogins
 WHERE DATEDIFF(hour,updatedate,getdate()) < @login_update_interval

/* Determine what DTS packages were created on the last
@dts_creation_interval
 * hours.
 * I currently don't know how to figure out if existing ones have
 * been modified.
 */

PRINT "The following DTS packages have been created within the past " +
CONVERT(VARCHAR(10), @dts_creation_interval) + " hours."
PRINT ""

SELECT SUBSTRING(name,1,30) AS package_name,
       SUBSTRING(owner,1,30) AS owner,
       createdate AS creation_date
  FROM msdb.dbo.sysdtspackages
 WHERE DATEDIFF(hour,createdate,GETDATE()) < @dts_creation_interval

/* Get list of jobs executed on system within the
 * past @job_execution_interval hours.
 * The CONVERT call is necessary because Microsoft decided to store
 * dates in the sysjobhistory table as an INT, but DATEDIFF can't work
on
 * an INT.
 */

PRINT "The following jobs have been run on this system:"
PRINT ""

SELECT SUBSTRING(name,1,30) AS job_name,
       run_date
  FROM msdb.dbo.sysjobs sysjobs, msdb.dbo.sysjobhistory sysjobhistory
 WHERE sysjobs.job_id = sysjobhistory.job_id
   AND DATEDIFF(hour, CONVERT(VARCHAR(10),run_date), GETDATE())
   < @job_execution_interval

/* SQL Server 7.0 Differential Database Audit Script
```

```
 * by Stephen V. Arehart
 *
 * This script can be used to audit for changes that occured over a
 * configurable time period within a given database.
 *
 * The script checks for the following:
 *    creation of new database users
 *    modifications to existing database users
 *    creation of new tables in the database
 *    creation of stored procedures in the database
 */

/* These are used to determine various time intervals over which events
are
 * audited.
 */

DECLARE @time_interval               INT
DECLARE @stored_proc_interval        INT
DECLARE @table_creation_interval     INT
DECLARE @user_creation_interval      INT
DECLARE @user_modification_interval  INT
DECLARE @database                    VARCHAR(50)

SET @time_interval = 24
SET @stored_proc_interval =  @time_interval
SET @table_creation_interval = @time_interval
SET @user_creation_interval = @time_interval
SET @user_modification_interval = @time_interval
SET @database = DB_NAME()

PRINT "SQL Server 7.0 Audit Script"
PRINT "by Stephen V. Arehart"
PRINT "Performing a differential audit on the database '" + @database +
"' ."
PRINT ""

/* Get a list of all users in the current database
 * that have been created over the past @user_creation_interval hours.
 */

PRINT "The following users have been created within the past "
+ CONVERT(VARCHAR(10),@user_creation_interval) + " hours:"
PRINT ""

SELECT SUBSTRING(name,1,30) AS user_name
  FROM sysusers
 WHERE DATEDIFF(hour,createdate,getdate()) < @user_creation_interval

/* Get a list of all users in the current database that have been
modified
 * over the past @user_modification_interval hours.
 */

PRINT "The following users have been modified with the past "
+ CONVERT(VARCHAR(10), @user_modification_interval) + " hours:"
PRINT ""
```

```
SELECT SUBSTRING(name,1,30) AS user_name
  FROM sysusers
 WHERE DATEDIFF(hour,updatedate,getdate()) <
@user_modification_interval

/* Get a list of tables that have been created within the past
 * @table_creation_interval hours.
 */

PRINT "The following tables have been created with the past " +
CONVERT(VARCHAR(10),@table_creation_interval) + " hours:"
PRINT ""

SELECT SUBSTRING(name,1,30) as table_name
  FROM sysobjects
 WHERE type = 'U'
   AND DATEDIFF(hour,crdate, getdate()) < @table_creation_interval

/* Get a list of stored procedures that have been created within the
 * past @store_proc_interval hours.
 */

PRINT "The following stored procedures have been created in the past "
+
CONVERT(varchar(10),@stored_proc_interval) + " hours:"
PRINT ""

SELECT SUBSTRING(name,1,30) AS procedure_name,
       crdate AS creation_date
  FROM sysobjects
 WHERE xtype IN ('X','P')
   AND DATEDIFF(hour,crdate, getdate()) < @stored_proc_interval
```