Interested in learning more about security?

SANS Institute
InfoSec Reading Room

This paper is from the SANS Institute Reading Room site. Reposting is not permitted without express written permission.

SQL Server 2000: Permissions on System Tables Granted to Logins Due to the Public Role

Microsoft SQL Server 7.0 and 2000 make use of the concept of roles at the server level and within each database. The public role is one of the built-in roles in each database and this corresponds to everyone who has access to that database. While the use of the public role is frowned upon for user-created objects, the role itself has permission to system tables and views, especially within the master and msdb databases. In this paper I will cover the access rights to system tables the public role has in these two syste...

Copyright SANS Institute

Author Retains Full Rights

AD

SQL Server 2000: Permissions on System Tables Granted to Logins Due to the Public Role

K. Brian Kelley
GSEC Practical 1.4b (Option 1)
Submitted: 15 October 2003

© SANS Institute 2003, Author retains full rights

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

Table of Contents

Abstract 1

Introduction 2

The Public Role 4

Logins vs. Users 4

The dbo and guest Users 5

Ownership Chains 5

All Objects Have the Same Owner 6

Objects with Different Owners 6

How Cross-Database Ownership Chaining Works 8

Permissions on System Tables and Views 9

Master Database System Tables 10

The sysaltfiles Table (secure) 10
The syscacheobjects Table (secure) 10
The syscharsets Table (not sensitive) 10
The sysconfigures Table (insecure) 11
The syscurconfigs Table (insecure) 13
The syscursorcolumns Table (secure) 14
The syscursorrefs Table (secure) 14
The syscursors Table (secure) 14
The syscursortables Table (secure) 14
The sysdatabases Table (insecure) 15
The sysdevices Table (insecure) 17
The syslanguages Table (not sensitive) 19
The syslockinfo Table (insecure) 20
The syslocks Table (insecure) 21
The sysmessages Table (insecure) 22
The sysperfinfo Table (secure) 24
The sysprocesses Table (secure) 24
The sysservers Table (insecure) 24
The sysxlogins Table (secure) 25

User Databases 25

The syscolumns Table (insecure) 26
The syscomments Table (insecure) 28
The sysdepends Table (insecure) 31
The sysfilegroups Table (insecure) 32
The sysfiles Table (insecure) 33
The sysfiles1 Table (secure) 33
The sysforeignkeys Table (insecure) 33
The sysfulltextcatalogs Table (insecure) 34

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

The sysfulltextnotify Table (secure) 36
The sysindexes Table (insecure) 37
The sysindexkeys Table (insecure) 37
The sysmembers Table (insecure) 38
The sysobjects Table (insecure) 38
The syspermissions Table (insecure) 39
The sysproperties Table (secure) 40
The sysprotects Table (insecure) 40
The sysreferences Table (insecure) 42
The systypes Table (insecure) 42
The sysusers Table (insecure) 43

MSDB Database "System" Tables **44**
The Backup Tables (insecure) 46
The LogMarkHistory Table (insecure) 47
The MSWebTasks Table (insecure) 48
The Restore Tables (insecure) 48
The RTbl Tables (not sensitive) 49
The syscategories table (not sensitive) 49

The sp_help* System Stored Procedures ***50***

Determining What Stored Procedures Are Tied to System Tables **50**

The Purpose of the sp_help* Stored Procedures **52**

A Partial List **52**

And Then There was OpenHack 4 ***55***

What Microsoft Did **55**

Is It Really That Simple? **56**

Concluding Thoughts ***56***

References ***58***

© SANS Institute 2003, Author retains full rights

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

Abstract

Microsoft SQL Server 7.0 and 2000 make use of the concept of roles at the server level and within each database. The public role is one of the built-in roles in each database and this corresponds to everyone who has access to that database. While the use of the public role is frowned upon for user-created objects, the role itself has permission to system tables and views, especially within the master and msdb databases.

In this paper I will cover the access rights to system tables the public role has in these two system databases as well as in a typical user database. I'll also cover how the guest user adds to the conundrum, especially with respect to the system databases and cross-database ownership chaining. Finally, I'll look at what permissions can be revoked from the public role in each database and what the consequences are, both from a practical perspective (typical applications) to an extreme example (Microsoft's OpenHack 4 configuration).

© SANS Institute 2003, Author retains full rights

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

Introduction

SQL Server security is a growing field given the rapid adoption of Microsoft SQL Server.[1] Organizations are turning more and more to SQL Server to hold their critical data and as a result SQL Server has become a larger target for attackers everywhere. The two big SQL Server worms, SQLSpida[2] and SQL Slammer[3] [4], have proven how widespread SQL Server is in the field and how important it is to be secure. Whether or not the SQL Server in question serves internal applications or internet-facing web sites, security is now of paramount concern.[5]

One of the questions I see frequently with respect to SQL Server security is, "What permissions does an average user have?" Typically the responses that I see returned are based on what explicit rights a DBA has given a user. For instance, a user has been placed in a role and that role has been given permissions to execute a set of five stored procedures. However, what usually isn't documented is what rights the user has because of the public role, a role all users who have the ability to access the database belong to. Within an individual database the public role has access to system objects which an individual user has no need to access. These objects include those system tables and views containing information about the data schema itself. This is clearly a violation of the Principle of Least Privilege.

Complicating the situation is the guest user. The guest user is a special user in each database that can be turned on or off as needed. If a given login within SQL Server has no rights to a database but the guest user is enabled, the login is mapped to the guest user. Of course, the guest user is a member of the public role. So long as the guest user isn't enabled in user databases this shouldn't be an issue except for the fact the guest user is enabled for the master (access is required) and msdb (access can be removed in some cases) system databases.

Contained within the system database are numerous stored procedures a user could execute to gain more information about the underlying structure of a particular SQL Server. In addition, the public role has access to the system table sysdatabases. Contained within this table are all the databases for a given SQL Server as well as the file location of each database's primary database file. Such information is extremely valuable in the hands of an attacker seeking to obtain access to the database files themselves.

Also adding to the situation is the concept of cross-database ownership chaining. If the ownership of objects stays the same, permission checking is intentionally

---

[1] Regan, *Top Dog Oracle Losing Database Market Share*
[2] Knowles, *Digispid.B.Worm*
[3] Knowles, *W32.SQLExp.Worm*
[4] Moore, *The Spread of the Sapphire/Slammer Worm*
[5] Kelley, *SQL Server Security: Why Security Is Important*

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

skipped. While this is typically considered a security benefit, if not carefully watched it could also prove to be a security risk when permissions are skipped.

However, most of the permissions granted to the public role in the system databases as well as in a user database can be revoked, especially when dealing with custom applications. Applications such as Microsoft Access and the ODBC Administrator rely on querying particular system tables and revoking permissions can “break” these applications as a result. In my testing, these two applications were used to represent “typical” applications given the nature of Access and the prevalent use of Data Source Names (DSNs – often configured through ODBC Administrator) by third-party applications. The results of my testing, along with what other tests have revealed, show what can and cannot be revoked must be handled on a case-by-case basis.

Revoking table permissions from public isn’t enough, though, as Microsoft has provided the all too-helpful sp_help* stored procedures to return information contained in these system tables. Combined with cross-database ownership chaining, these stored procedures can be used by everyday users in order to reconnaissance SQL Server.

However, Microsoft has provided the bare minimums necessary for SQL Server to stay running in the OpenHack 2002 (OpenHack 4) configuration. We can use this as the “goal” in locking down SQL Server from the public role. The draconian measures Microsoft took may be beyond what can be supported when specific applications are considered, but they do provide insight into “how far we can go.”

We may not get “there,” but understanding what the public role has access to is key to determining what the risks are for Microsoft SQL Server. What can be tightened down should be. What can’t should be understood, documented, and checked periodically. Hopefully this paper will further an understanding of the extent to which the public role has fingers throughout SQL Server.

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

The Public Role

In every database for SQL Server 7.0 and 2000 there exists a special role called the public role[6]. Every user who has rights to this database is automatically a member of the public role. If the public role has permission to do something, all users have permission to do it. Likewise, if the public role is explicitly denied permission, all users are denied (with the notable exception of dbo).

The public role is a fixed-database role. It cannot be removed from the database. DBAs can grant, revoke, and deny permissions to the public role, though generally, granting or denying permissions to this role is not considered a best practice.[7] Rather, DBAs should create user-defined database roles and assign appropriate permissions to those roles. Users should be placed in these roles, allowing permissions to be managed according to the concepts of role-based security.

Even following this best practice, DBAs are often horrified to find what the typical user has access to do. Because of how SQL Server 2000 is typically configured, users have access to certain objects in the master and msdb databases, with master being the most significant. These objects allow for a great deal of reconnaissance on the SQL Server, more so than any security-conscious DBA would feel comfortable about. The reason users gain access is due to a special user account: guest.

Logins vs. Users

Before I delve into the guest user, I'll cover the difference between a login and a user in SQL Server. Unfortunately, DBAs (myself included) tend to use the terms login and user interchangeably, leading to a lot of confusion. However, the terms are not interchangeable.

A person has the ability to log on to SQL Server using a login. The login is at the server level. A login can either be a SQL Server login (held over from older versions but still heavily used by third-party applications) or a Windows account. That Windows "account" could even be a Windows group through which all Windows logins belonging to the group gain access.

A user, on the other hand, is specific to a database within SQL Server. Logins are mapped to users in a given database and it is the user who owns objects or has permissions to access database objects (whether explicitly defined against the user or through a role with permissions). A user does not cross databases

[6] SQL Server Books Online, Topic: public Role
[7] Warren, *Using the Public Role to Maintain Permissions*

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

because it is only defined at the database level. However, a given login could be mapped to multiple users, all within different databases.

In SQL Server there are two special users in every database. Those users are dbo and guest. The guest user is a particular problem child with respect to security.

The dbo and guest Users

The dbo user corresponds to the database owner. Whatever login owns the database maps in as dbo. In addition, all logins granted System Administrator rights within SQL Server (this is different from the local Administrators group for the operating system) also map in as dbo. The catch with dbo is this user bypasses all security checking. As a result, the typical user should never be mapped to dbo.

The other account, guest, is used for any login that isn't explicitly mapped to a user within the database. For instance, if a given login doesn't have access to the HR database, but the guest user is enabled, the login is mapped to the guest user. Any permissions the guest user has, the login has. Keep in mind all users belong to the public role. This includes guest. Therefore, if guest is enabled, all logins have whatever permissions the public role has.

The public role has direct access to quite a few sensitive system tables in SQL Server. This means a user with the ability to access a particular database could then issue queries directly against those system tables. While DBAs can partially mitigate this issue by ensuring the guest user isn't enabled in user databases, the guest user must be active in both the master and the tempdb databases. Master is of particular concern because there are quite a few "helpful" stored procedures a user can execute. These stored procedures can access system tables in user databases due to cross-database ownership chaining. But before I delve into cross-database ownership chaining, I'll first talk about ownership chains within the context of a single database.

> **Note:** Even if the guest user is disabled in a database, it'll still appear in the sysusers table within the database. This is by design. Under no circumstances should this row be deleted from sysusers. This isn't supported by Microsoft and moreover, can lead to an access violation.[8]

Ownership Chains

Ownership chains are a key part of SQL Server's security model[9] [10] but they can be a little difficult to comprehend at first. By using ownership chains, I can control

[8] Microsoft Knowledge Base Article 315523
[9] Kondreddi. *Overview of SQL Server Security Model and Security Best Practices*

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

how users access the data without giving them explicit rights to the tables that store that data. This is a crucial piece of the security puzzle.

For instance, if I create a stored procedure that carefully restricts how the data is modified and I understand ownership chains, I can force users through the stored procedure. Using this method I can ensure they don't go directly against the base tables, potentially causing a data integrity or data loss issue. I've been situation when a third party application requires such permissions and a user has blown away tables of critical data. Restoring data is tedious and ownership chains, when properly implemented, can reduce the frequency of restores in addition to the obvious benefits in securing information disclosure.

All Objects Have the Same Owner

Ownership chains come into play when we are dealing with the typical database objects: tables, views, stored procedures, and user-defined functions. Views depend on tables and sometimes other views. Stored procedures can depend on tables, views, and other stored procedures. Both of these can depend on user-defined functions and user-defined functions can depend on other user-defined functions as well as depending on tables and views. It's an interconnected web. Each of these objects has an owner. If our objects have the same owner, SQL Server will short-circuit the security check.

Using a stored procedure as an example, if the owner of the stored procedure owns all stored procedures, views, tables, and functions referenced by the stored procedure, security is only checked once, on the initial stored procedure. SQL Server will verify the user has permission to execute the stored procedure. SQL Server will make the assumption the owner intended for the user to have access to the other objects used in our "top" stored procedure declaration. I should remark here that there is an exception: dynamic SQL. Dynamic SQL under SQL Server 2000 executes under a different "batch" and is thus treated akin to a whole new process. Because of this, SQL Server rechecks security on a dynamic SQL query, even if the query is against objects with the same owner.

Objects with Different Owners

If I have nested objects like stored procedures calling stored procedures referencing views referencing other views referencing tables (just to build a long and complex chain), SQL Server will follow the chain as far as it can until it either reaches the end of the chain or hits an ownership change. Then it'll start the ownership chain over again. Figure 1 shows a complex chain.

[10] Microsoft, *SQL Server 2000 Security Model*

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

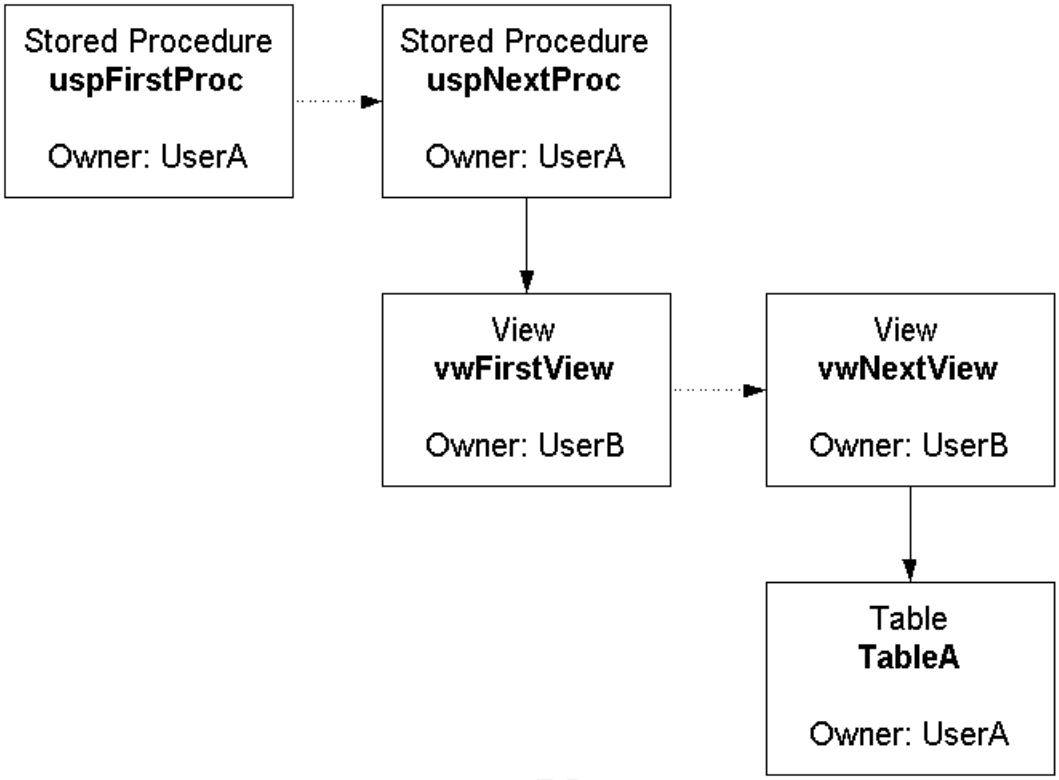

Figure 1. Complex ownership chain

I'll break this down step-by-step to show how SQL Server will handle this set of objects. First, SQL Server will check permissions on uspFirstProc since it is the initial object. Since uspNextProc is also owned by UserA, SQL Server doesn't do a permissions check. I've represented when SQL Server doesn't do a permission check with a dotted line. The next object in line is vwFirstView, which is owned by UserB, SQL Server will do a permissions check since the owner has changed. I've used a solid line to represent that a permission check does occur.

It'll then start the ownership chain over again, but this time with UserB. So when SQL Server hits vwNextView, also owned by UserB, SQL Server will consider it part of the new ownership chain. Since it's part of the chain, security isn't rechecked. Then SQL Server hits TableA. Even though I started with an object owned by UserA, SQL Server is now on UserB for its ownership chain.

Since this is a change in owners, SQL Server will recheck permissions on TableA. So to summarize, SQL Server does permission checks on uspFirstProc, vwFirstView, and TableA. The objects uspNextProc and vwNextView escape a permission check because they were part of already existing ownership chains. Putting this into a table, here's what SQL Server looks at:

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

Table 1. Security Checks on complex ownership chain.

Object	Owner in Chain	Owner of Object	Security Check
uspFirstProc	(none)	userA	Yes
uspNextProc	userA	userA	No
vwFirstView	userA	userB	Yes
vwNextView	userB	userB	No
TableA	userB	userA	Yes

Thus far, I've stayed in the same database. However, one of the reasons a user has so many permissions is due the permissions of the public role in various databases, to include master and msdb. If I have a user in the database for a particular application, that user can cross databases. Ownership chaining exists across databases, but it's slightly more complex.

How Cross-Database Ownership Chaining Works

Cross-database ownership chaining has always been present in SQL Server 7.0 and 2000. However, before SQL Server 2000 SP3, the existence of such a concept wasn't well-documented. Also, cross-database ownership chaining was on and could no be disabled. This represented a security issue, and with SP3, Microsoft added the capability to turn off cross-database ownership chaining across the entire server (which a couple of notable exceptions) and the ability to turn it on for particular databases.[11]

Normal database permissions work based on the user within a database. However, once you cross databases, you aren't dealing with the same user, since users are defined within a database. What's can be confusing is users own objects within databases. If I do a query against a user table and I check the owner, it's going to come back as the user. For instance, if my login MyNetworkLogin is mapped into a given database as the user MyDatabaseUser, a query as to who owns the table will turn up MyDatabaseUser. But users are tied to the database, so how does cross-database ownership chaining work? Something else needs to be used to tie objects together across-databases.

That something is the login. Since logins exist at the server level, they exist outside of any database. Furthermore, logins are mapped to users within a database. The system table *sysusers* ties the uid (User ID), which is used to identify the owner, to the sid (the unique ID for the login). Since I have a tie between users and logins for all databases the login is the natural choice to tie objects together across two or more such databases
.

[11] Kelley, *SQL Server 2000 SP3: What's New in Security*

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

Whenever a query crosses from one database to another, SQL Server compares the login from the first object with the login for the second object. Like with object chaining within a single database, if SQL Server finds the ownership a match, it will skip the permissions check. There is a catch, however, and that's based on who is executing the query.

When dealing with a single database, whoever is executing the query must already have access to the database. For example, if I'm executing a stored procedure in DatabaseA, my login must be mapped as a user within that database. If my login is not, then I can't get to the stored procedure in the first place, much less anything within that database called by the stored procedure. When crossing databases, a login must be mapped to a user in each database. If a query crosses from one database, say DatabaseA, to another database, DatabaseB, my login must have a mapping to a user in both DatabaseA and DatabaseB. While SQL Server may skip the permissions check between objects, it does not skip the check to see if a login has access to all databases concerned.

Keep in mind that the guest user, if enabled, is used whenever a logging doesn't have an explicit mapping. In other words, if the guest user is enabled and a login isn't mapped to a particular database user, SQL Server automatically completes the mapping and the login therefore has access.

Permissions on System Tables and Views

Recall that the guest account is enabled for both the master and msdb databases. All users, including guest, are members of the public role for a given database. What this means is that if someone has a login to SQL Server, they have access to objects within the master and tempdb databases at a minimum. Unless the guest user has been disabled in the msdb database, a login would have access to certain objects in that database as well. Finally, there are objects to which the public role has access in each user database as well.

Taking a look at the permissions, as well as the data each table, I've given three ratings: secure, insecure, and not sensitive. Objects marked secure contain information of interest to an attacker but by default the object is protected (at least with respect to direct access such as through a SELECT query). Objects marked insecure also have useful information but the public role has access. Objects marked not sensitive do not typically contain information outside of what can be found in a base install of SQL Server and thus do not represent a significant resource for gaining more knowledge about the SQL Server in question.

Naturally it's impossible to test permissions for all cases. My tests on whether or not anything "broke" centered primarily using the ODBC Data Source Administrator (sometimes shortened to ODBC) and Microsoft Access 2000 (MS

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

Access). If functionality was lost using these two applications, I've cited it. If a possible work-around exists, I've provided the information on how to utilize it. I'll start with the master database, then took at generic user databases, before finally covering the msdb database.

Master Database System Tables

The master database has system tables we won't find anywhere else. Some of these tables determine the locations for database files or provide information on how the server is performing.

The sysaltfiles Table (secure)

Microsoft doesn't provide a whole lot of documentation on the sysaltfiles table because this table is intended solely for SQL Server's use. One reason for the table is to store the location of the Temdb files. Every time SQL Server starts up, SQL Server retrieves the location for the Tempdb database and recreates the database files at the paths specified.

No permissions have been defined against the sysaltfiles table and as a result, only sysadmins have access to its contents. As a result, this system table is secure.

The syscacheobjects Table (secure)

SQL Server stores execution plans in the syscacheobjects table. Once again, this table is intended solely for SQL Server's internal use. However, if I'm trying to check if an execution plan for a particular stored procedure is getting cached, I can query directly against this table to find out.

Like the sysaltfiles table, no permissions have been set for the syscacheobjects table. As a result, the public role does not have access. Only sysadmins can look at this system table. It is also secure.

The syscharsets Table (not sensitive)

The syscharsets table contains all of the character sets, including case and accent sensitivity and sort order, supported by the particular instance of SQL Server. For almost all instances of SQL Server 2000, this table will contain exactly the same information. As a result, the contents of the table aren't of a sensitive nature.

Microsoft has given the public role SELECT permissions on this table, possibly because the contents aren't of particular importance to an attacker. The

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

syscharsets table contains collation information and while collation is important to display the data correctly, knowing all the collations a given SQL Server supports isn't going to give an attacker an advantage. Knowing the default collation might but since this table stores *all* collations supported by a given instance of SQL Server, it is not of much use to an attacker and I've indicated that by marking it as not sensitive.

In my testing I found I could create an ODBC connection (using a login that didn't have sysadmin privileges) when I revoked public access to syscharsets. However, I could not test the connection using the ODBC Data Source Administrator because when the test connection is made, a query is made against syscharsets:

```
select name
from master.dbo.syscharsets
where id = convert(tinyint,
            databasepropertyex ( db_name() , 'sqlcharset'))
```

As a result, the connection will succeed, but the query will fail. The ODBC Data Source Administrator takes the failure on the query against syscharsets as an overall failure of the test, but it does show that the connection is established successfully. I disregarded the error and was able to use the ODBC connection in MS Access.

> **Note:** In my testing, I used REVOKE and not DENY. REVOKE will simply remove the permission. DENY will place a block. If you use DENY, only members of the sysadmin role will ever be able to access the system table because that role doesn't go through a permissions check in the normal sense. By using REVOKE, you can prevent access to the system table, but you give yourself the flexibility if you do want to allow a role other than sysadmin to have access to the table. For more information on DENY and REVOKE, see SQL Server 2000 Books Online.

If you decide to lock down syscharsets, expect the error. Verify the connection succeeds because the error is a consequence of the query above failing. However, that has no effect on the connection itself.

The sysconfigures Table (insecure)

The sysconfigures table, as its name implies, contains configuration information about SQL Server. I consider the information in this table to be sensitive. Even seemingly innocent facts such as the number of users a particular SQL Server will allow can be used proactively. For instance, if I were an attacker I'd want to know how many connections I could create in order to brute force a password. That's just one example. Two other key bits of information I can gain from

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

sysconfigures is whether or not the SQL Server is in C2 Audit mode[12] and if this particular SQL Server allows updates to the system tables (not enabled by default, though this behavior can be overridden).

Since users have access to more information than they need to do their daily jobs, this clearly violates the Principle of Least Privilege. Certainly most users have no reason to know anything about the SQL Server configuration other than how they are supposed to login and what resources they have access to. I'm very uncomfortable with the public role having SELECT rights against this table.

When I REVOKE SELECT against the sysconfigures table for the public role, this is only partially effective. The system stored procedure sp_configure can return much of the same information. This stored procedure won't show the advanced configuration options unless you've toggled "show advanced options" on (a server or system administrator for SQL Server must toggle this setting - in which case it is on for all users), but a user could still get a bit of information about the server configuration from the use of sp_configure. As I'll talk about in the section on securing stored procedures, I could REVOKE EXECUTE on sp_configure for the public role, and this puts yet another barrier up.

Microsoft has built stored procedures for DBAs to use to gain information about the underlying schema of the databases. The intent with these stored procedure is to break a dependency on using the system tables to gain such information. Microsoft reserves the right to change the structure of these tables with any new release, to include a service pack. As a result, Microsoft provides support to the information stored within these tables using the stored procedures. From a security perspective that means not only do we need to look at the permissions on the tables themselves, but also on the stored procedures that have access to them. Since the stored procedures and the tables have the same owner (dbo), permissions are only checked on the stored procedure unless there is code within the stored procedure to do additional checking.

With that said, even I secure sysconfigures and the stored procedure sp_configure, this doesn't completely secure the configuration information. Figure 2 shows the use of the variable @@MAX_CONNECTIONS to get the maximum number of connections the SQL Server allows. Notice the failures to retrieve information via the sysconfigures table and the sp_configure stored procedure. Yet the user is still able to get the maximum number of connections.

[12] Microsoft. *Microsoft SQL Server 2000 C2 Evaluation*

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

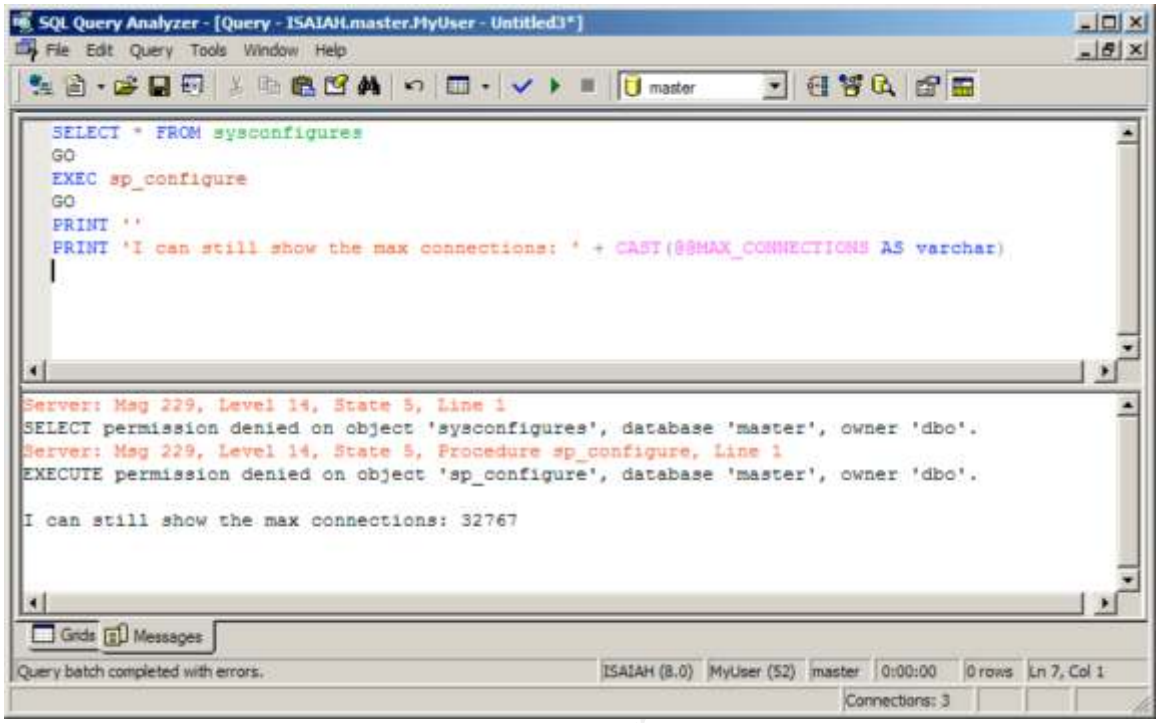

Figure 2. User still retrieves maximum number of connections.

Unfortunately, I can't prevent access to this variable. So an attacker still has some workarounds, but at least I can make it more difficult for the attacker to get to sensitive configuration information.

With that said, when I did revoke permissions to the public role and suffered no ill consequences with ODBC or MS Access. Unlike syscharsets, the sysconfigures table isn't called by the ODBC Data Administrator or by MS Access when the connection configuration is tested or a link table added.

The syscurconfigs Table (insecure)

Like sysconfigures, the syscurconfigs table contains configuration information on SQL Server. The syscurconfigs table is really a virtual table built each time it's queried by a user. The difference between this virtual table and the sysconfigures table is the sysconfigures table stores contains both the configuration information since startup as well as what the current values (running values) are while the syscurconfigs only holds the most current values.

Also like the sysconfigures table, the syscurconfigs table is not secure. The public role has SELECT rights against this table as well. Therefore, if you've decided to REVOKE SELECT for the public role of sysconfigures, you should do so on syscurconfigs as well. Since syscurconfigs has the current configuration parameters, an attacker could use this table to get at the server's configuration information, too.

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

When I revoked permission to syscurconfigs, I saw no ill effects with either the ODBC Data Administrator or MS Access. This matches what I saw with sysconfigures and since syscurconfigs contains similar to sysconfigures, it's logical that the effect is the same.

The syscursorcolumns Table (secure)

Though this is listed as a system table if you check out the sysobjects system table, you won't find any documentation on the syscursorcolumns (or any of the syscursor* tables) in Books Online. The syscursorcolumns table is undocumented, but it does exist.[13] This table's purpose is to store information about the columns of any cursors that happen to be active on the system.

Since it is an undocumented system table, no permissions have been defined. As a result, only members of the sysadmins role can query the contents on this table. The public role has no access rights. Therefore, this system table is secure.

The syscursorrefs Table (secure)

Like the syscursorcolumns table, the syscursorrefs table is undocumented. The syscursorrefs table stores the name of each cursor, whether it is global or local in scope, and a unique identifier (called cursor_handle) used by the server to identify the cursor. Also like the syscursorcolumns table, the syscursorrefs table has no access permissions defined. Therefore, only members of the sysadmins role can access this table. The public role cannot execute a query against this table.

The syscursors Table (secure)

Another undocumented system table, the syscursors table contains information about the cursors running on the system and their attributes such as whether or not they are static or keyset, forward only or scrollable, what the last operation was, and the like. Like the rest of the syscursor* tables, the syscursors table does not have permissions defined on it. Only members of the sysadmins role, who have permissions to everything, have the ability to access this system table.

The syscursortables Table (secure)

The syscursorstables system table keeps track of the base tables for each cursor. It also keeps track of any optimizer hints that may have been used as well

[13] Chigrik, *SQL Server 2000 Undocumented System Tables*

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

as what locking mechanism was specified. Like the rest of the system tables for cursors, there are no permissions defined. The public role does not have access.

The sysdatabases Table (insecure)

The sysdatabases system table is much like the sysconfigures table: most users don't need access to it. If a given user is accessing a database-backed application, the application should have the configuration information and the user has no reason to know the database or anything else. However, if the user is using a third-party reporting tool then the user has to have access to SQL Server. But a user shouldn't be presented with a menu of all the databases on a particular server. The user possesses just such a menu with the sysdatabases system table since the public role has select rights against this table.

Because the public role has access to sysdatabases, an attacker who has managed to get a login to SQL Server will be able to retrieve the following information: a list of all databases on the server and the location of the primary database file for each one. Figure 3 shows the information on the three system databases for a default installation of SQL Server.

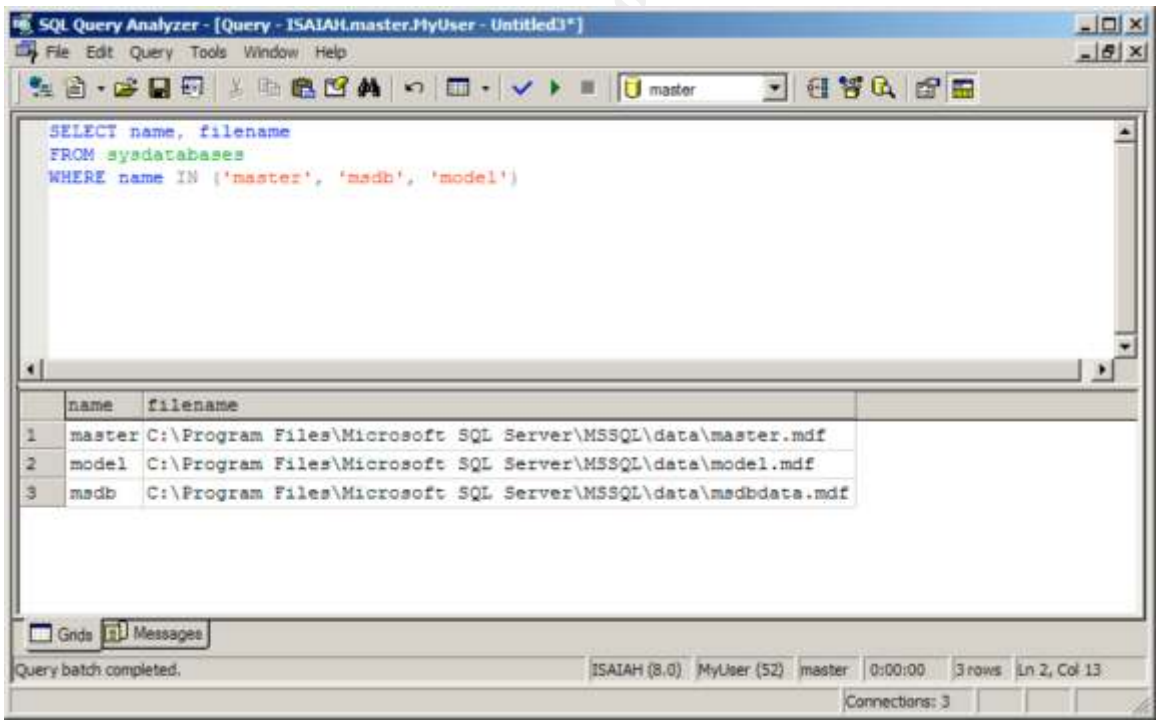

Figure 3. Database name and file locations for databases.

Having the filenames means having *exact* locations to attack. One of the lessons learned from securing Internet Information Server is if an attacker has a definite file path, he or she can then leverage that knowledge in further attacking the system. For instance, with an unpatched version of IIS 4.0 and Index Server 2.0,

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

an attacker could issue a bogus request to index server that would reveal the physical (*i.e.* C:\Inetpub\WWWRoot) path for the web server root directory.[14] An attacker could then use this directory in combination with other vulnerabilities to view source files on the server or even execute code.

Because this system table contains file path information I'd like to tell you to REVOKE SELECT to the public role, but unfortunately, I can't if you are using ODBC and MS Access. From my testing, the ODBC Data Source Administrator issues a query against sysdatabases if I check "Connect to SQL Server to obtain default settings for the additional configuration information":

```
select name
from master..sysdatabases
where has_dbaccess(name)=1
```

Without access to sysdatabases, ODBC Data Source Administrator can't validate whether or not any database other than whatever SQL Server selects as the default is a valid database. Even if I specify a valid database and then try and go on to the next screen, I receive the error stating, "The database entered is not valid." The key is not to check that checkbox beside "Connect to SQL Server to obtain default settings for the additional configuration information." The ODBC Data Source Administrator will assume I know what I'm doing and lets me enter in any values without checking. I can go through the configuration and even test successfully with no errors (provided I haven't locked down syscharsets, as above).

In my testing I found ODBC wasn't alone in accessing the sysdatabases table. Microsoft Access does as well if I try and create a link table, even if I have the ODBC connection created. All I did to generate this error was to select the defaults. As Access is in the process of bringing up the dialog window where I select what tables to link, an error indicating SELECT permissions have been denied comes back (Figure 4).

Figure 4. Access queries against sysdatabases.

Unfortunately, there isn't a work around on this. Access issues the following query no matter what:

[14] Microsoft Security Bulletin (MS00-006)

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

```
select substring('NY',status/1024&1+1,1)
from master..sysdatabases
where name=DB_NAME()
```

My consternation doesn't end there. If I try and create an Access project, Access uses an undocumented stored procedure, sp_catalogs_rowset;2, to retrieve the list of databases. Notice the version marking (;2). There is a version 1 and a version 5 as well. Access specifically calls version 2. Since sp_catalog_rows is owned by dbo and sysdatabases is also owned by dbo, the ownership chain is in effect and the public role is able to see the list of databases. The only option then is to REVOKE EXECUTE on sp_catalogs_rowset. Doing this means the user has to know exactly what database to connect to. But at least I can get a project created.

However, locking down sysdatabases will break the ability to link tables from a normal MS Access database. It also means I won't be able to connect to SQL Server through ODBC Data Source Administrator to get default settings, as with syscharsets.

The sysdevices Table (insecure)

The sysdevices system table contains information for all configured backup devices. If I've created a backup device, I'll find the information in this table. Again, this information includes the physical file paths. And not surprisingly, the public role has SELECT permissions against this table. In addition, since this table contains information on the data and log files for master, model, and tempdb databases due backward compatibility, the file path information is also there. I can get around the first issue, the file paths of my backups, because I can specify a file path as a backup device when I call a BACKUP DATABASE or BACKUP LOG operation. I don't have to use a pre-created device. But I can't get around the second issue, the file paths for the three system databases, because I didn't put them in sysdevices in the first place.

In my opinion, this table should have been secured but I understand why Microsoft did not. When SQL Server checks security, it does so against the database user. Cross-database ownership chains notwithstanding (remember, they can avoid permission checks when crossing databases), if the login doesn't have a user account with valid permissions in the database where the object resides, the login can't access the object. In other words, regardless of what database I'm in, if I'm using a login that isn't a member of the sysadmin fixed server role and I query sysdevices in master, I have to map to a user account in master that has permissions on sysdevices. By default, the guest user is enabled in master (don't remove it, it's required) and the public role has SELECT rights against sysdevices. However, if I were to REVOKE SELECT on sysdevices from public, I would have no way to SELECT against sysdevices.

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

One fixed server role that would potentially need access to sysdevices is diskadmin. So my first thoughts would be to assign the permission to diskadmin and all is well. Except it's not. I can assign permissions to a user or database role. I cannot assign permissions to a server role. If I were to try and execute the following T-SQL command, I would get an error saying there's no such user or group:

```
GRANT SELECT ON sysdevices TO diskadmin
```

I've chosen diskadmin because Disk Administrators have the ability to add and remove backup devices (for more information on the various server roles, see SQL Server Books Online). Logically, if I want a server role (other than sysadmin) to have access to this table, that role would be the diskadmin role. However, SQL Server will return an error because the security doesn't work for server roles. So this attempt falls flat.

Since I can't assign permissions against server roles, the easiest way to ensure diskadmin role members have the ability to view the contents of sysdevices is to grant SELECT permissions to the public role. Hence I understand Microsoft's approach but I'd still rather the sysdevices table is secure. To do this, I can develop a workaround if I create a specific database role to use for permissions. Here is a code sample:

```
-- drop public role's permissions
REVOKE SELECT ON sysdevices TO public

-- create a database role specifically to handle permissions
-- against sysdevices
EXEC sp_addrole 'db_diskadmin'
GRANT SELECT ON sysdevices TO [db_diskadmin]

-- Give a login access to the master database and put it in the role
EXEC sp_grantdbaccess 'MyUser'
EXEC sp_addrolemember 'db_diskadmin', 'MyUser'
```

I've created a role, db_diskadmin, within the master database to use to handle who has access to the sysdevices table. Since the public role doesn't have permissions, no one who isn't in my db_diskadmin role will able to retrieve the contents of the sysdevices table. In order to ensure a user in the db_diskadmin role can SELECT against sysdevices, I first have to grant SELECT rights to db_diskadmin. Then, I'll need to grant access to the master database for my user. Finally, I'll have to place the user in the db_diskadmin role.

If you are thinking that's a lot of work to keep the public out of sysdevices you are right. In reality, this only becomes necessary if I am using the diskadmin server role. If only sysadmins will be defining backup devices, then I can simply REVOKE permission to the public role and be done with it. Otherwise, a database role has to be created and users who need access to sysdevices have

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

to be added as users to the master database and then made rolemembers of the particular database role. I don't recommend doing this, even though I've provided a code sample on how to do so.

Though I've provided a code sample on how to get diskadmins access to sysdevices, I recommend *against* granting database access to master. I have provided a code sample to demonstrate the steps that are necessary to work around the issue. While the work around isn't terribly difficult, the security implications are such that I recommend specifying the backup path in the T-SQL statement, thereby eliminating the need to access sysdevices at all.

As far as testing went, neither ODBC nor MS Access have any normal reason to access sysdevices, so locking down this system table did not cause an issue.

The syslanguages Table (not sensitive)

Like the syscharsets table, the syslanguages table isn't sensitive. This table contains one row for every language supported by the particular SQL Server. As with syscharsets, this table is standard for most installations of SQL Server. In addition, knowing the languages doesn't help an attacker since the attacker always has the option of defaulting to US English. This language is always present in SQL Server, though it's not found in the syslanguages system table.

When I tested locking down the syslanguages table, I found ODBC Data Source Administrator could still connect to SQL Server to get the default settings without issue (so long as I didn't lock down sysdatabases). However, on the dialog box where there is an option to configure the languages in which system messages are returned, the checkbox to change the default language is disabled (Figure 5). This did not, however, affect testing the connection or creating the ODBC DSN. There were no problems with MS Access, either.

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

Figure 5. Change language option disabled.

The syslockinfo Table (insecure)

The syslockinfo system table is another virtual table that is generated by SQL Server when we query it. The syslockinfo table will return information on all current and pending locks within SQL Server. Locking information is great for DBAs trying to troubleshoot a blocking problem, but in reality an end user wouldn't need to know about locking. In a development environment I can see how experienced developers who understand locking and blocking could be more productive by having access to this table, but I certainly don't agree with such developers having that kind of access in production.

On the surface I would be likely to say that having access to locking information isn't that big of a help to an attacker, but it can be, because it can give an attacker more insight into what would make the best target. If an attacker is able to monitor locking information, the attacker is able to see what databases and what objects within those databases are frequently accessed. An attacker can use all of this information to get a picture of how the data is being used.

Consider the case where an attacker has access to a particular SQL Server. The attacker is a legitimate user (inside the organization). The attacker has access to a couple of databases on this SQL Server but there are many more the attacker doesn't have access to. Now the attacker is smart, he's only going to try and penetrate a database if it's used a lot. Otherwise, the payoff for the risk he's taking may not be anything at all. After all, he doesn't want to get caught going after a DBA's test database. There's no value in that! If he's going to take a chance in being discovered, he wants to be going after information he can sell for a good price.

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

By watching the syslockinfo table, the attacker can see what databases are used a lot. If the attacker also watches the system process IDs (SPIDs) of the users accessing the system, the attacker may be able to correlate locking to particular users. The queries of a CEO should generate more interest than those of the print plant. Since syslockinfo returns the requesting SPID, an attacker could then use the system stored procedure sp_who or the undocumented system stored procedure sp_who2[15] (which is why these stored procedures need to be locked down as well) to find out the user login belonging to a particular SPID. By correlating the user with the objects where locking is occurring, an attacker could have greater insight into what objects to try and go after.

Hopefully my example has demonstrated the sensitivity of the locking information on a production system. Since the average end user does not need access to such information, revoking SELECT permissions to public is a viable option. If you decide to lock down the syslockinfo table, some functionality for a non-sysadmin user will be impaired in Enterprise Manager. A non-sysadmin user will be unable to return information about current activity, with an error being return because of the syslockinfo lockdown. Figure 6 shows the error generated.

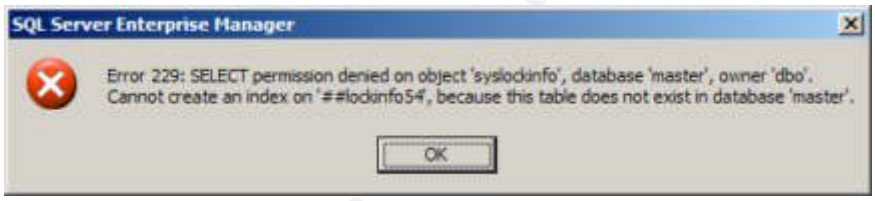

Figure 6. Non-sysadmin user returns error on syslockinfo.

If a non-sysadmin user needs access to the syslockinfo table, I have two options: (1) leave access as it is or (2) use a workaround similar to what I described under sysdevices. However, generally only DBAs should be monitoring current information on a database such as locking and processes, therefore securing syslockinfo is a good idea.

As far as testing is concerned, neither the ODBC Data Source Administrator nor MS Access uses syslockinfo. As a result, locking this table down does not break most applications. Exceptions, of course, are management application like SQL Server's Enterprise Manager.

The syslocks Table (insecure)

If I look in Books Online, I'll see that Microsoft has "removed" the syslocks table from SQL Server 2000. However, if I issue the following simple query against a SQL Server 2000 server, I find the syslocks table is still present:

[15] Chigrik, *SQL Server 2000 Useful Undocumented Stored Procedures*

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

```
SELECT * FROM syslocks
```

While the syslocks system table doesn't contain near the information available in syslockinfo, it does contain the database and object IDs in correlation with the SPIDs. As a result, it is just as sensitive as the syslockinfo table. However, the big difference is since Microsoft has said all references to the syslocks table should be removed for SQL Server 2000, securing it shouldn't be an issue at all. Once again, the public role has SELECT rights on this table, something we can do without. As a result, simply revoking rights for the public role will ensure an end user doesn't have the ability to use this table to return locking info. Again, since sysadmin role members bypass security checks, there aren't any issues if for some reason you want to use this system table. It's supposed to be "gone" but it's not. And as with syslockinfo, revoking public access to this table does not affect ODBC Data Source Administrator or MS Access.

The sysmessages Table (insecure)

The sysmessages table contains the preformatted error messages SQL Server returns. I've listed this table as insecure, but I only consider this particular system table insecure if custom error messages are used heavily for a particular SQL Server. With the use of the system stored procedure sp_addmessage, I have the ability to add my own custom error messages which I can use with the RAISERROR() command.

RAISERROR() doesn't have to use sysmessages. However, if I have a common set of error messages that are reused for a particular internal application, I would gain some advantage with adding custom error messages to sysmessages. If I have to change an error message, I only have to do so in one place and not in all the spots in code where you refer to that particular error. Hence the reason the sysmessages table exists.

Following that thinking, if I'm not careful about what type of custom messages I use, if any, I may give away information I didn't intend for everyone to see. For instance, I may have particular error messages indicating why a database operation failed written specifically for my application. For those users of the application, the error message is necessary. However, for general users who don't use the application, the error message shouldn't be seen. An example would be, "You don't have access to the loan. Call John Smith at x500 if you think you've gotten this message in error."

An attacker viewing this message would know John Smith has something to do with loans and also what extension he has. Aha! The attacker now has a target for a social engineering attempt. This particular error message reveals too much. In reality it's not a good message because John Smith will certainly not be an employee forever and his extension may not stay the same for the duration of his

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

tenure with the organization. However, this particular message illustrates how I could provide too much information in a basic error message.

If I'm not using the sysmessages table for custom error messages, then there isn't much an attacker can gain from this table. The messages would be the stock ones that are the same from install to install. Downloading the evaluation version of SQL Server 2000 from Microsoft and perusing the sysmessages table accomplishes the same thing. In other words: nothing.

If, however, I am using sysmessages for custom error messages, I can simply revoke SELECT rights to public. The only three stored procedures dependent on the sysmessages table are the system stored procedures designed to handle creating, modifying, and dropping these custom error messages.

Even with no permissions set for the public role, users will still get back error messages if they do something to warrant it. Figure 7 shows MyUser trying to SELECT against syslockinfo (which I secured previously) and sysmessages (secured as well). The error messages returned are the correct ones, indicating permission has been denied.

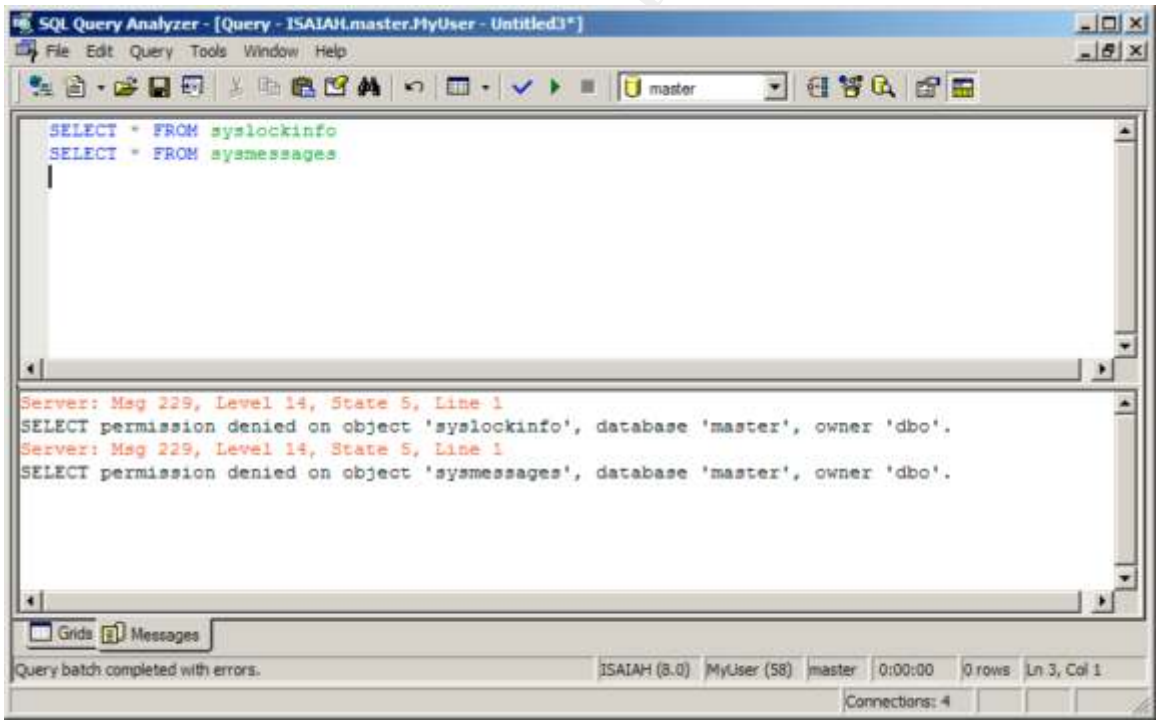

Figure 7. Error message even with sysmessages secured.

Even if I'm not using custom messages, locking down sysmessages doesn't appear to break any functionality. Also, if a future application does use custom error messages, the sysmessages will already be secured against a potential attacker. Once again, the ODBC Data Source Administrator and MS Access

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

don't query against sysmessages directly, so locking this table down doesn't have an impact on basic applications.

The sysperfinfo Table (secure)

The sysperfinfo table contains the performance counter values for SQL Server objects. Unlike most of the other system tables, the public role does not have access. As a result, no additional security measures are needed. No objects reference it, so a normal user isn't going to be able to get at the data contained in this table. This table is secure as is.

The sysprocesses Table (secure)

The sysprocesses table is secure by default but there are system stored procedures which report the information it contains. Returning to my example given in syslockinfo, if I can match up the SPID with the databases undergoing heavy locking, I can narrow down on targets to attack. Thankfully, the public role does not have the ability to SELECT against the sysprocesses table directly, so that means in order to secure the information contained within, we merely have to lock down any objects that reference it.

There are stored procedures requiring access to sysprocesses. The two biggest stored procedures that reveal too much information are sp_who and sp_who2. Since both reveal the SPID, the login, and the database the SPID is currently accessing, they represent a gold mine to a potential attacker. Also, since both the stored procedures and the sysprocesses table are owned by dbo, ownership chains are in effect. So although the public role doesn't have direct access to sysprocesses, it can get at the sysprocesses information because the public role does have EXECUTE rights on both sp_who and sp_who2. If I REVOKE EXECUTE on these two stored procedures, an attacker won't be able to determine who is on and accessing resources. I'll cover how to identify which stored procedures reference which tables later in the section on system stored procedures.

The sysservers Table (insecure)

One of the things attackers like finding is a list of valid computers within an organization. The sysservers system contains a list of all linked and remote servers known by a particular install of SQL Server. In other words, that ready-made list attackers love so much. Once again, the public role has SELECT permissions on this system table. This system table contains a row for every server, not necessarily SQL Server, that particular SQL Server can connect to via OLE DB. I say not necessarily SQL Server because a linked server connection to an MS Access database will show up in sysservers as well.

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

To make matters worse, sysservers has a whole host of stored procedures that refer to it. As a result, the sysservers table does serve an important role. However, whether or not an end user needs access to this table is a different question entirely. If I apply REVOKE against sysservers for the public role I've not done a whole lot because of the stored procedures that also return information on the remote and linked servers. There are quite a few stored procedures that depend on sysservers in some way. I issued a query against a SQL Server 2000 SP3 server and it returned 72 stored procedures that referred to sysservers in the text of the stored procedures. Most are add, change, or drop stored procedures dealing with replication. Some, like sp_helpserver, return the very information of a critical nature in sysservers.

Some of these stored procedures also include file paths and share points. Not only will these stored procedures reveal the name of the server, but also an access point. And all of this information is available to the public role by default. I am absolutely frightened by the amount and "quality" of information left unsecure in conjunction with this system table alone. As a result, if I can lock this table and the applicable stored procedures down, do so. However, I know to test rigorously, especially if I'm using replication. One of the catches with Microsoft's OpenHack 4 configuration[16] (which I'll discuss shortly) was that it did not use replication and thus replication support was not installed. Because replication wasn't covered in this "test," I am cautious about checking out a lock-down down this table and any objects related to replication. Replication can be highly customizable so it's difficult for me to make any generic recommendations on it except to test it heavily.

The sysxlogins Table (secure)

The sysxlogins table is an important system table because it contains the list of logins recognized (NT users and groups as well) for a particular SQL Server. Not only that, but it also has the hashes for the passwords of any SQL Server logins. Microsoft has not given any logins or roles permissions to this table, meaning a user has to a member of the sysadmin role to query against it. If an attacker were able to get at the password hashes, the attacker would be able to leverage brute force methods to attempt to crack the passwords. There are several articles[17] [18] available discussing weaknesses in the password hashes within SQL Server and this table should most certainly be kept out of the hands of the normal user.

User Databases

Just as there are important system tables in the master database, there are certain system tables that are present in every database. Like with the system

[16] *eWeek, OpenHack 2002 Downloads*
[17] Kelley, *SQL Server Security: Login Weaknesses*
[18] Litchfield, *Microsoft SQL Server Passwords (Cracking the password hashes)*

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

tables in the master database, most of the database-specific system tables are accessible by the public role.

The main issue with altering permissions on any of these system tables means the Information Schema views break. Though a particular Information Schema view only returns the information a user is allowed to see (for instance, only the columns a particular user has access to), the view does use the system tables. And since dbo owns the system tables and a "virtual owner," INFORMATION_SCHEMA, owns the Information Schema views, there is a breakdown in ownership chains and thus the Information Schema views also break.

Information Schema views contain metadata about the database. Microsoft has implemented them both to meet the SQL-92 standard that requires such views, but also to provide metadata information to users so they do not have to issue queries directly against the system tables. Microsoft recommends the use of these views because they should stay consistent across releases and Microsoft reserves the right to change the structure and availability of system tables at any time.

If I need the functionality of the Information Schema views or if I desire to allow users to use them, I won't be able to lock down most of the database-specific system tables. As a result, I have to tread carefully when it comes to locking down these particular system tables.

The syscolumns Table (insecure)

The syscolumns system table contains a listing of all the columns for all tables and views in a given database. In addition, it is also used to store the parameters for all the stored procedures for a given database as well. The main problem with the public role having permissions to this table is an attacker can find out about the structure of tables, views, and stored procedures he or she doesn't normally have access to.

For instance, consider a personnel system where the table Employee has fields for employees' emergency phone numbers, managers, departments, and salaries. Some personnel in Human Resources are allowed to see salaries, but not all members of this department are permitted to do so. After all, salaries are a sensitive item. As a result, personnel who aren't permitted to see salary are given access to the rest of the data through the use of a view that restricts the columns from the table so that the Salary column is excluded. An attacker from this group would be able to see that such a column does exist by querying directly against syscolumns. Not only that, but the attacker would also be able to see the base table that contains this field.

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

Because an attacker can learn more about the structure of a database than I want, locking down the syscolumns table by REVOKE SELECT to public seems like a good idea. Then a normal user won't be able to then view the contents of syscolumns, nor would the user be able to use the view INFORMATION_SCHEMA.Columns. A given user who has a legitimate need to see column information is also barred from doing so. The problem that arises, though, is with the db_datareader role.

The db_datareader role has implicit SELECT permissions against all tables and views in a given database, even if explicit SELECT permissions aren't set. The exception is if a particular user also has DENY as well, in which case the DENY trumps this implicit set of permissions. So if I simply revoke SELECT from public but I'm using the db_datareader role, a user in this role can still view the contents of syscolumns and the rest of the database-specific system tables. I don't recommend issuing DENY against the public role, nor am I big fan of the db_datareader role. Accessing system tables in databases is where these two approaches come head-to-head. I can't simply revoke permissions if I have users assigned as db_datareader. Then again, if I issue DENY to public and I have a normal user with a legitimate need (such as someone who is in the db_owner role), that user is broke. If possible, revoke permissions from public and don't use db_datareader.

Revoking public access to a system table in a database does not stop the db_datareader role. The db_datareader role has implicit SELECT rights against all tables and views. As a result, even if public doesn't have access to a particular table, a member of the db_datareader role does.

Another issue is that system stored procedures come into play like with system tables in master. If a user uses sp_help, a stored procedure the public role has EXECUTE rights on, the user can retrieve the column information for a given table even if the user is barred from syscolumns. Therefore, I'll have to lock down this stored procedure as well, but since it exists in the master database (and works because of cross-database ownership chaining), I'll effectively prevent the user from calling it in *any* database.

As might be expected, this issue of getting at the information found in a system table for legitimate use while preventing it when unauthorized is going to be a recurring theme as I go through the rest of the database system tables. Unfortunately, the options to securing database-specific system tables are more limited than with the system tables only found in the master database. Basically, I can revoke direct access against the base tables for a particular database, but in order to shut down public access to stored procedures that return the same information, I'll have to lock down the system stored procedures in master and that means affecting all databases.

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

As I'll cover when I get to the system stored procedures, locking down all the stored procedures isn't 100% possible if I am using applications like MS Access. In fact, locking down sp_columns, to prevent a user from getting column information on a given table, breaks MS Access' ability to link tables from SQL Server. Situations like this one are why I say that if you are using some of the basic tools, you may run into issues when you start locking things down..

The syscomments Table (insecure)

Of all the system tables to restrict at the database level, the syscomments table may initially be seen as one of front-runners. The syscomments table contains the code behind any view or stored procedure in the database. Such information is certainly dangerous in the wrong hands! If a user can get at the contents of a particular stored procedure, the user can then figure out how it works. I don't want this at all.

However, before restricting access to the public role on syscomments, one option we do have is to use the WITH ENCRYPTION option when creating our stored procedures and views. While the encryption mechanism on syscomments is relatively weak, a user either has to be a sysadmin of the system or be able to alter, create, and drop stored procedures and views in order to reveal the contents of them. Here is a stored procedure that allows a user with such rights to be able to peer into any stored procedure or view that is encrypted in such a fashion:

```
-- Decrypts encrypted stored procedures and views
-- Maximum length is ~ 4000
CREATE PROC sp_DecryptObject
  @Name nvarchar(50)
AS

SET NOCOUNT ON

DECLARE @SQL nvarchar(4000)
DECLARE @Type char(1)
DECLARE @ObjectWord nchar(4)

-- Verify the Object exists
IF EXISTS(SELECT Type FROM sysobjects WHERE name = @Name)
   BEGIN
     SET @Type = (SELECT Type FROM sysobjects WHERE name = @Name)
   END
ELSE
  BEGIN
    RAISERROR('Object Not Found', 16, 1)
    RETURN(-1)
  END

-- Check to see if it is a stored procedure or view
IF @Type = 'P'
   BEGIN
```


© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

```
   SET @ObjectWord = 'PROC'
   END
ELSE
  IF @Type = 'V'
    BEGIN
      SET @ObjectWord = 'VIEW'
    END
  ELSE
    BEGIN
      RAISERROR('Object not a view or stored procedure', 16, 1)
      RETURN(-1)
    END

-- Check to see if it is encrypted
IF NOT EXISTS(SELECT * FROM syscomments sc JOIN sysobjects so ON
sc.id = so.id
              WHERE sc.encrypted = 1 AND so.id = OBJECT_ID(@Name))
   BEGIN
     RAISERROR('Object not encrypted', 16, 1)
     RETURN(-1)
   END

CREATE TABLE ##Code (
  ID int,
  ctext nvarchar(4000)
)

SET @SQL = N'
INSERT INTO ##Code
SELECT 1, ctext FROM syscomments WHERE id = OBJECT_ID(''' + @Name +
''')'

EXEC(@SQL)

SET @SQL = N'ALTER ' + @ObjectWord + ' ' + @Name + ' WITH ENCRYPTION
AS SELECT
''0123456789012345678901234567890123456789012345678901234567890123456
789012345678901234567890123456789012345678901234567890123456789012345
678901234567890123456789012345678901234567890123456789012345678901234
5678901234567890123456789012345678901234567890123456789012345678901234
5678901234567890123456789012345678901234567890123456789012345678901234
5678901234567890123456789012345678901234567890123456789012345678901234
5678901234567890123456789Test'' Test'

EXEC (@SQL)

SET @SQL = N'
INSERT INTO ##Code
SELECT 2, ctext FROM syscomments WHERE id = OBJECT_ID(''' + @Name +
''')'

EXEC (@SQL)

DECLARE @OCryptString nvarchar(4000),
        @OPlainString nvarchar(4000),
        @NCryptString nvarchar(4000),
        @NPlainString nvarchar(4000),
        @Position int,
        @Length int
```


© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

```
SET @OCryptString = (SELECT ctext FROM ##Code WHERE ID = 1)
SET @NCryptString = (SELECT ctext FROM ##Code WHERE ID = 2)
SET @NPlainString = 'CREATE ' + @ObjectWord + ' ' + @Name + ' WITH
ENCRYPTION AS SELECT
''0123456789012345678901234567890123456789012345678901234567890123456
7890123456789012345678901234567890123456789012345678901234567890123
4567890123456789012345678901234567890123456789012345678901234567890
1234567890123456789012345678901234567890123456789012345678901234567890
123456789012345678901234567890123456789Test'' Test'

SET @Length = DATALENGTH(@OCryptString) / 2
SET @OPlainString = REPLICATE(N'A', @Length)

SET @Position = 1

WHILE (@Position <= @Length)
BEGIN
  SET @OPlainString =
    (SELECT STUFF(@OPlainString, @Position, 1,
        NCHAR( UNICODE( SUBSTRING( @OCryptString, @Position, 1 ) ) ^
        ( UNICODE( SUBSTRING( @NCryptString, @Position, 1 ) ) ^
        ( UNICODE( SUBSTRING( @NPlainString, @Position, 1 ) ) ) ) )
)
    )
  SET @Position = @Position + 1
END

SELECT @OPlainString

SET @SQL = N'DROP ' + @ObjectWord + ' ' + @Name
EXEC(@SQL)

EXEC(@OPlainString)

DROP TABLE ##Code
```

This stored procedure is adapted from a script posted by Shoeboy to BugTraq[19] that is now readily available all over the Internet. Other stored procedures very similar to this one are also available, and they cite Shoeboy's script as their source. However, all of these decrypting stored procedures suffer from a couple of limitations. First, it only works on stored procedures and views of less than approximately 4000 characters. The reason for this limitation is due to the fact that the field ctext is only 4000 characters long. As a result, when a stored procedure or view exceeds this length, it is "wrapped" into additional rows as needed. Second, since the object is being altered, dropped and then recreated, all permissions against the object will be lost. So the script can't be run if you need to maintain permissions on an object.

[19] shoeboy, *Some analysis of Microsoft SQL Server 2000 stored procedure encryption*

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

There is a small application called dSQLSRVD (dOMNAR's SQL Server Syscomments Decryptor)[20] is not bound by these limitations because it uses a mechanism to read and decrypt the stored procedure or view without having to alter, drop, and create it. However, because the only way to get a particular piece of information needed to decrypt requires sysadmin rights, you have to be a sysadmin to run it successfully.

I'll admit, however, that most stored procedures and views aren't created using the WITH ENCRYPTION option. As a result, locking down the syscomments table is definitely a good idea. However, users can still reveal the contents of unencrypted stored procedures and views using the sp_helptext stored procedure. Like with sp_help, sp_helptext is located in the master database. So if I restrict it, I restrict it across all databases. I don't have the option of only leaving it accessible for a particular database.

When testing with ODBC Data Source Administrator and MS Access, I found no problems when locking down this table. This is logical, as neither of those applications have any reason to query against syscomments. In fact, it is doubtful any application for the end user should have such a capability. Administrative tools for the DBA are a different story, but end user tools have no need to query syscomments.

The sysdepends Table (insecure)

The sysdepends system table is supposed to contain dependency information between objects. An attacker could use it to learn about more objects in a given database. How much information is present?

If objects like stored procedures have been created in the proper order, all dependency information should be stored in this table. However, it is possible to create a stored procedure that references another stored procedure that doesn't yet exist. As a result, the dependency information won't be found. Also, since sysdepends only stores dependency information for objects in the same database, any cross-database dependencies won't be recorded. So sysdepends won't contain information on objects from other databases and it won't contain information when objects have been created out of order. Also, if an object is "stand-alone" and doesn't depend on another object and also isn't depended upon (*i.e.* a table not referenced by a stored procedure or view) it won't show up in sysdepends. Note that foreign key constraints do not count as dependencies with respect to being stored in the sysdepends table. Even with that said, if I've made good use of stored procedures to secure our database, then a reference to most of the user objects in the database will be found in this table.

[20] dOMNAR, *dSQLSRVD*: http://www.geocities.com/d0mn4r/dSQLSRVD.html

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

Because an attacker can garner quite a bit of information from this table, it would be another good one to lock down. However, as with syscolumns and syscomments, a stored procedure in the master database gives users access to the same information as sysdepends even if the public role is denied access. The stored procedure in this case is sp_depends. The sp_depends stored procedure will return the name of the object and also what type of object it is. Again, unless I want to restrict sp_depends against all databases, I can't lock it down. With that said, another stored procedure I've already mentioned, sp_help, returns all of the objects in the database if I don't pass it a name. As a result, solely locking down sysdepends doesn't accomplish a whole lot since an attacker has so other avenues.

The big difference over syscolumns, however, is that MS Access doesn't use any stored procedures that query against it. Neither does the ODBC Data Source Administrator. As a result, I can lock this table down more often than is the case with syscolumns.

The sysfilegroups Table (insecure)

The sysfilegroups table is another system table accessible by the public role. However, the information contained within isn't as damaging at first glance as say the sysdatabases table from the master database. After all, the only information an attacker could glean is the name of the filegroups except there is the status field. By checking the status field, they could see if a filegroup is toggled for default (0x10 in hexadecimal or 16 in decimal) which means read/write or just read only (0x8 in hexadecimal or 8 in decimal). However, I've not seen a lot of installations that take advantage of using a read-only file group. As a result, the status field set for the default may not indicate anything at all.

The sysfilegroups table reveals the name of the filegroups but it doesn't contain any file path information. As a result, there's not a whole lot of information to garner from this particular table. But since an end user usually has no legitimate reason to know what the filegroups are for a particular database (after all, the DBA should be concerned about the physical structure of the database, but someone in accounting should not), the table shouldn't be open to the public role.

As with the other system tables, the sysfilegroups table does have a stored procedure that reveals the same information: sp_helpfilegroup. Once again, it sits in the master database. Unlike sp_help or sp_depends, sp_helpfilegroup isn't a stored procedure an end user should use. As a result, revoking EXECUTE rights on this stored procedure should not be an issue in most environments. As far as MS Access or ODBC Data Source Administrator are concerned there are no issues with locking this table down.

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

The sysfiles Table (insecure)

Like the sysdatabases table from the master database, the sysfiles system table contains file paths. Again, I think this is valuable information to an attacker. Certainly, no end user needs to know the specific file names for the database files. However, the public role does have SELECT rights against this table, too. Again, the end user doesn't have a reason to know physical file locations so I don't agree with the public role having access.

Like the other database-related system tables, the sysfiles table isn't the only mechanism available to find out file information. The system stored procedure sp_helpfile will return the primary database file for a given database. Also, I've already discussed the issues with trying to lock down the sysdatabases table in the master database. So even if I like down sysfiles for a particular database and restrict sp_helpfile, the issues with locking down sysdatabases means an end user will be able to find out the file path for the primary data file. Locking down sysfiles and sp_helpfile will put some roadblocks in the way of an attacker but unless my environment is such that I can lock down sysdatabases, an attacker will still be able to locate at least one file path for a given database.

Most user applications won't use the sysfiles table, so this table is up for consideration. End users should have no need for the file paths of the databases of your SQL Server. Protect them accordingly.

The sysfiles1 Table (secure)

Unlike the sysfiles system table, the sysfiles1 table is actually a virtual table that contains some but not all of the information sysfiles contains. Books Online doesn't discuss it but the table is present in all databases. It too has the file path information for a given database but the public role cannot SELECT against it. Even though it is a virtual table, I can assign permissions against it, just as with the syslockinfo table from the master database, though it isn't necessary unless I want to do so for the db_datareader role. Since the db_datareader role has implicit SELECT permissions on all tables, a member of this role can view sysfiles1 unless it is locked down.

Again, the same problems with locking down sysfiles apply to sysfiles1. If sysdatabases can't be locked down, locking down sysfiles1 won't ultimately stop an attacker. If, however, I can lock down sysdatabases and revoke access to sp_helpfile, locking down sysfiles and sysfiles1 should keep an attacker from discovering the file paths for a particular database.

The sysforeignkeys Table (insecure)

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

When I discussed the sysdepends table I stated it didn't contain information on foreign keys. Foreign keys have their own table, appropriately named sysforeignkeys. It too is accessible by the public role. Since foreign keys help keep relational integrity intact, they are valuable to the DBA or developer. However, the end user shouldn't be concerned with what foreign keys are defined. After all, they are part of the underlying schema and an end user should be interested in the data, not the structure.

Like with sysdepends, the sysforeignkeys table might give an attacker information about more objects in the database. As a result, an attacker would then have additional targets to try and get access to. Of course, like sysdepends, the sysforeignkeys table has its own system stored procedure: sp_helpconstraint. This system stored procedure, when given a table, will return all constraints on the table. As a result, the foreign key information is readily available unless I lock down sp_helpconstraint as well. Even if I do lock down both the sysforeignkeys table and the sp_helpconstraint stored procedure, my work is undone by the sp_help stored procedure. Therefore, I must be able to secure this system stored procedure or a user executing sp_help against a particular table gets the foreign key information anyway. So once again, locking down this system table is impossible in most environments.

As with sysdepends, locking down this table doesn't affect MS Access or ODBC Data Source Administrator. Neither these two, nor most end user applications would query against this table. Tools that map databases might, but likely only DBAs need such rights. After all, if a user (such as a developer) needs a database layout, the DBAs can always provide one if the need is legitimate.

The sysfulltextcatalogs Table (insecure)

The sysfulltextcatalogs table is only populated if full-text indexing is configured for a given database. If I'm not not using full-text indexing, this table isn't insecure at all for it is empty. However, if I am using full-text indexing for a particular database, I'll find a row per catalog on the database. Now if I'm using default file paths, all is well, for the path field will be NULL. However, if I chose to use something other than the default path, the path I've specified will show up. This is easy to demonstrate. First, I've created a local directory at C:\FullText\ which is not the default full-text catalog path. If I create two different full-text catalogs with the Pubs database, the first using my c:\FullText and the second not specifying a path (so it'll use a default), I'll have two rows in my sysfulltextcatalogs table:

```
USE Pubs
GO

sp_fulltext_catalog
  'SecurityExample',
  'Create',
  'C:\FullText'
```


© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

```
GO

sp_fulltext_catalog
  'DefaultPath',
  'Create'
GO
```

When I query the sysfulltextcatalogs, I'll see both catalogs (Figure 8). The first, SecurityExample, shows the path. The second, DefaultPath, does not. Obviously if I stick to just the default path, an attacker might be able to determine the full-text catalog name, but nothing more. However, if I have full-text catalogs distributed to non-default directories, the file paths for those directories will be stored in this table.

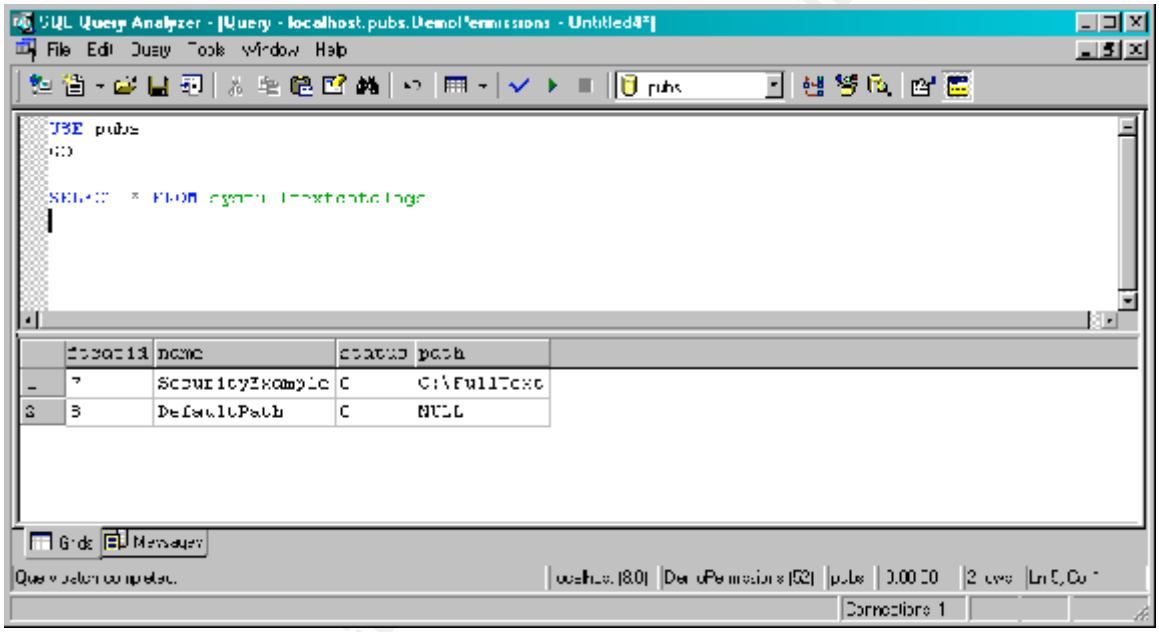

Figure 8. Paths in sysfulltextcatalogs

Since the structure and location of the full-text catalogs are not needed by a typical end-user, the public role does not need access to the sysfulltextcatalogs table. Revoking SELECT permissions against this system table will prevent direct access. However, there are two system stored procedures that have to be dealt with in order to completely secure the information contained within this system table. Those two stored procedures are sp_help_fulltext_catalogs and sp_help_fulltext_catalogs_cursor. The latter stored procedure returns the information in the form of a cursor, otherwise both stored procedures function the same. Both are located in the master database and since the public role doesn't really need access to these stored procedures I can revoke EXECUTE rights from the public role.

Some may be concerned about the Microsoft Search service (required for Full Text Indexing) because of the account it runs under. The Microsoft Search

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

service is supposed to run as LocalSystem. Even if I decide to revoke the login BUILTIN\Administrators, I still need to add another login to handle LocalSystem *first* and give this login sysadmin rights. As a result, there's not a lot that can be done in this case. Running Microsoft Search in any other context can result in a peaked-out processor state[21] or even an access violation. It is not a supported configuration by Microsoft.

The sysfulltextnotify Table (secure)

The sysfulltextnotify is an undocumented system table used for full-text operations. There are several system stored procedures that use this table, but it contains no real information useful to the average user. If I'm not using a feature called change tracking, this table won't be populated. However, if I have enabled change tracking, any time a change is made affecting a full-text indexed column, SQL Server will make an entry in the sysfulltextnotify table.

By default, SQL Server doesn't continually update full-text indexes as it does with clustered and non-clustered indexes. One method of keeping full-text indexes updated is to run an update job using SQL Server Agent during periods of low activity. Another method is to enable change tracking where SQL Server records all changes affecting full-text indexed rows as they occur. With change tracking enabled, updates to the full-text index can be made on a scheduled basis where only the rows affected are updated in the full-text index. The update can be set as a background operation. Enabling change tracking and performing updates based on change tracking recorded information is accomplishing using the sp_fulltext_table stored procedure. Such a discussion is beyond the scope of this paper, but you can find more information about Full-Text Indexing Change Tracking in Books Online.

If you aren't using change tracking, this table will be empty and as a result doesn't reveal any sensitive information. If, however, you are using change tracking, an attacker could get a picture of database activity by watching this table closely. This little bit of information may give an attacker of what particular table(s) to go after.

However, this table is like sysfiles1 in that the public role doesn't have explicit permissions to access the table. If, however, a user is assigned to the db_datareader role, the user will be able to SELECT against sysfulltextnotify. If I deny SELECT rights to public or to db_datareader, the user won't be able to peer in to the sysfulltextnotify system table. Unlike most of the other system tables, this one does not have a stored procedure that allows a user to peer into the contents of the table. This is probably because this table is intended for internal use only. So if I issue the DENY statement, I should be able to secure this system table without issue, even if I am using the db_datareader role.

[21] Microsoft Knowledge Base Article 295034

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

The sysindexes Table (insecure)

The sysindexes system table is another table that reveals physical structure information. Like the other database-specific tables that reveal physical structure, this one is accessible by the public role as well. The one main difference is the sysindexes table appears very cryptic at first glance. However, if an attacker is trying to determine which tables are being changed the most, the sysindexes system table represents the perfect place to look. The rowcnt and rowmodctr columns reveal the number of rows and the number of rows changed since the last time statistics were updated, respectively.

By restricting access to the sysindexes table, I can ensure the end user can't find out how many rows have been updated since the last time statistics were calculated. Of course, using stored procedures like sp_help will reveal what the indexes are on a particular table and what columns they reference, but sp_help doesn't reveal rowcnt or rowmodctr. The stored procedure sp_spaceused will reveal the number of rows and the amount of space the data uses, but again, it doesn't reveal any information like rowmodctr. There's one other stored procedure, sp_helpindex that also returns information on indexes, but it does reveal the number of rows or the number of rows that have been modified.

Taking all this into consideration, I can lock down sysindexes pretty tight and that will keep an attacker from being able to observe the row activity. Likewise, I can remove EXECUTE rights for the public role on both sp_helpindex and sp_spaceused because neither of these stored procedures would typically be used by end users. The stored procedure sp_help will reveal some information about the indexes on a given table, but at least it isn't everything that's found in sysindexes.

The sysindexkeys Table (insecure)

Like the sysindexes table, the sysindexkeys system table is open to the public role as well. This table contains information on what columns make up each index. This is the information returned by sp_help when we use it to examine a table in more detail. Like with most of the other physical structure tables, the sysindexkeys table contains very little information of value to a legitimate end user. While an attacker may find the information of some value, a representative from marketing would normally care less about this system table.

Locking down sysindexkeys will have a very limited effect if sp_help can't be locked down. In addition, both sp_help and sp_helpconstraint reveal information about the makeup of each index, though the latter stored procedure only does so if the index is tied to a constraint such as a primary key constraint.

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

The sysmembers Table (insecure)

The sysmembers system table is a join table to tie a database user to a database role. It is also open to the public role and represents a bonanza to an attacker. With this table I can find all the users who are members of db_owner or db_datareader and gain immediate access to any table in the database if I were to somehow manage to compromise one of those user accounts. For instance, the following query against sysmembers table reveals the users and their group memberships:

```
SELECT USER_NAME(memberuid), USER_NAME(groupuid)
FROM sysmembers
```

It really is that easy. End users shouldn't know users and role memberships, but they can find out such information via the sysmembers table. In addition, there are two stored procedures which let an end user return the members of a given group or role. These two stored procedures are sp_helpgroup and sp_helprolemember. So in addition to revoking SELECT rights against the sysmembers table (and barring or not using db_datareader), I'd also have to revoke EXECUTE rights for public on the sp_helpgroup and sp_helprolemember stored procedures.

One of the issues with revoking EXECUTE rights against these two stored procedures, though, is if a user is assigned to the db_securityadmin role for a given database. When I revoke EXECUTE permissions, the user no longer has the ability to use these two stored procedures. In order to enable the user to have access, I'll have to use a technique similar to what I described under sysdevices where a role is created in the master database with the ability to execute the two stored procedures. The user would then have to be granted access to the master database as a user and added to the role.

The sysobjects Table (insecure)

I've already discussed that sp_help without parameters reveals all of the objects, so right up front I realize that locking down sysobjects doesn't accomplish a whole lot in most environments. Sysobjects is a great table for the DBA because it contains every object. If I want to do something against a particular type of object such as all stored procedures, I can go to sysobjects. There are other methods as well, but such is the power of sysobjects. And therein lies the attraction for the would-be attacker. If something is useful for the DBA, chances are it is useful for the attacker.

As an attacker I can grab the names of the objects, what user owns them, and what they are. That's a lot of valuable information! Unfortunately, it is near impossible to lock all of this information away from the curious, much less the malicious. Just about every stored procedure starting with sp_help depends on

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

the sysobjects table. So even if I want to go about revoking SELECT permissions to the public role, I'd then have to go and revoke EXECUTE rights on a number of stored procedures, to include sp_help. What a nightmare! I've basically put sysobjects in the same class as sysdatabases. Some DBAs can lock it down because their environment permits it, the majority of others cannot.

If you have custom-built apps, you are probably in the category of the DBAs who can lock this table down (along with the applicable stored procedures). However, if you are supporting MS Access and other applications like it, you'll have the problem that these applications use sp_tables and similar stored procedures to build their lists. For instance, if you create an MS Access Project and successfully make a database connection, without sp_tables you won't be able to see any of the tables or views. Similarly, if you use a standard Access database and attempt a link table, without access to sp_tables you'll get the EXECUTE permissions denied.

The syspermissions Table (insecure)

My opinion on the syspermissions table matches that of the sysmembers table: end users shouldn't have open access to read what security is setup on a given database. However, the syspermissions table is available for the public role. I am assuming the rationale is something similar to the sysmembers table, where the public role has been granted rights to this table in order to proide support for the db_securityadminrole.

An attacker can use this table to figure out what permissions have been assigned to what user or role. Admittedly, the actadd column is cryptic and unless an attacker understood what the values were for this "internal use" column, the attacker won't be able to figure out what right a particular user or group has. However, by a process of matching actadd's values for rows that aren't known to rows that are known, an attacker could piece together what rights each user or group does have. A little tedious, perhaps, but it is worth it for someone intent on stealing data.

By revoking SELECT rights to the public role, end users won't be able to look at these rights and have the data to try and piece together actadd. Unlike most of the other system tables, syspermissions doesn't have any stored procedures that reveal similar information. As a result, when you lock down syspermissions, there isn't another mechanism to get at the information in syspermissions.

However, sysprotects also contains permissions information. As a result, it has to be locked down as well. I'll discuss sysprotects shortly, but suffice it to say if only one of the two tables is locked down the permissions can still be obtained by an attacker. Since this table isn't used by most user applications, I didn't find any issues with revoking access to it.

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

The sysproperties Table (secure)

If you are defining extended properties on your database objects, you'll find SQL Server stores them in the sysproperties table. Extended properties are new to SQL Server 2000. They are used to store metadata about the database objects and can be customized any way a developer or DBA sees fit. An application or user can use the fn_listextendedproperty function to return the extended property information on a given database object.

SQL Server doesn't have any permissions set against this undocumented system table meaning the public role doesn't have direct access to sysproperties. Because of this, a normal user will return the permission denied exception whenever an attempt is made to query against this particular table.

Because no explicit permissions are assigned, the sysproperties table is as secure as it can be. The problem is the function fn_listextendedproperty returns information contained within this table if the user knows how to query it properly. Since this is a system function, there's no mechanism for assigning permissions to it, meaning all users have the access specified by Microsoft. In the case of fn_listextendedproperty, that means the public role has SELECT rights. As a result, there's not anything else we can do about sysproperties except choose simply not to use it.

I'll be honest in saying that in my work environment we don't use extended properties and store them within the database. The main reason is we do a lot of transfers to and from a mainframe and other disparate (non-SQL Server systems). The big problem we face is identifying how a given field on the mainframe translates into our core warehouses, down to our data marts, and finally into our end-user applications with whatever they have for data stores (flat files, MS Access databases, etc.). As a result, we have to track our metadata at a more global level and have built systems to do this. Because we have to have an enterprise view of the data as it goes from one system to another, keeping extended properties up-to-date would represent a second place where we would have to maintain data. Therefore, we don't make use of this feature within SQL Server. If you do, keep in mind the system function that public will retain access to and judiciously place information about your objects into these extended properties.

The sysprotects Table (insecure)

Like the syspermissions table, the sysprotects table also stores permissions information. However, the sysprotects table comes into play especially when dealing with column-level permissions, which incidentally, is the granularity at which SQL Server verifies permissions. Also like the syspermissions table, the

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

sysprotects table is viewable by the public role. End users thus have rights to view what sort of security settings are on the various database objects/
In preparing the list of what objects were secure or insecure, I queried directly against this system table to view permissions information. The query I used was quite simple:

```
SELECT
    so.name [Table]
  , USER_NAME(sp.uid) [User],
    CASE action
      WHEN 26  THEN ' REFERENCES'
      WHEN 178 THEN ' CREATE FUNCTION'
      WHEN 193 THEN ' SELECT'
      WHEN 195 THEN ' INSERT'
      WHEN 196 THEN ' DELETE'
      WHEN 197 THEN ' UPDATE'
      WHEN 198 THEN ' CREATE TABLE'
      WHEN 203 THEN ' CREATE DATABASE'
      WHEN 207 THEN ' CREATE VIEW'
      WHEN 222 THEN ' CREATE PROCEDURE'
      WHEN 224 THEN ' EXECUTE'
      WHEN 228 THEN ' BACKUP DATABASE'
      WHEN 233 THEN ' CREATE DEFAULT'
      WHEN 235 THEN ' BACKUP LOG'
      WHEN 236 THEN ' CREATE RULE'
    END [Action]
  , [Columns]
FROM sysobjects so
  LEFT JOIN sysprotects sp
    ON so.id = sp.id
WHERE so.type = 'S'
ORDER BY so.name
```

The WHERE clause I've used, checking to see if the sysobject's column type is equal to 'S', is what restricts my result set only to system tables. However, an attacker could choose any value and in fact does not have to reference sysobjects at all. Simply using OBJECT_NAME(id) is sufficient to return what the object is called.

Once again, this is valuable information to an attacker. Not only would an attacker know what objects there are in the database but also the attacker is able to find who has what permissions. Obviously this is more information than what we want any end user gaining! However, simply denying SELECT rights on sysprotects is not enough. Unfortunately, there are several stored procedures that return the same information contained within sysprotects:

- sp_column_privileges
- sp_column_privileges_rowset
- sp_MSobjectprivs
- sp_table_privileges
- sp_table_privileges_rowset

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

All of these stored procedures are in the master database and the public role has EXECUTE rights against them. The sp_MSobjectprivs supports functionality in Enterprise Manager and the others are standard system stored procedures through sp_column_privileges_rowset and sp_table_privileges_rowset aren't documents in Books Online.

The main issue with locking down these stored procedures is once again with the db_securityadmin role. A user in this role would have a legitimate need to the information returned by these five stored procedures. In order to keep the public from having access but still giving access to security admins, the workaround of generating a role in the master database and assigning users to it would have to be implemented.

The sysreferences Table (insecure)

The sysreferences is another system table for foreign keys. Unlike sysforeignkeys, however, sysreferences records the columns that make up the foreign key relationship for both the referencing and referenced tables. The additional column information revealed by sysreferences might be of interest to an attacker, but once again, if the end user has access to sp_help, locking down sysreferences will do little good. Also, the sp_helpconstraint system stored procedure I cited under sysforeignkeys queries sysreferences as well.
The system stored procedure sp_helpconstraint returns the columns that make up the foreign key relationship, so if I'm going to lock down sysreferences, I need to lock down sp_helpconstraint as well. If I can lock down sp_help, then I can prevent an attacker from gaining any additional information via this system table. Without the ability to revoke execute rights on sp_help, however, and end user can get the information stored in sysreferences.

Once again, end-user applications typically have no need to access this type of information. Most applications try to determine relationships by looking at column names and then prompting the user to verify. As a result, I didn't find any issues locking down this table, with the exception of the system stored procedures that also returned similar information.

The systypes Table (insecure)

The systypes table contains a row for each system and user-defined data type. I debated about whether to mark this table insecure or not sensitive, but since I'm dealing with the underlying schema, there is some value to querying this table if user-defined data types are being used. Otherwise, this table contains the default 26 system-supplied data types present in every table. If I want to see what user-defined types are present in the current database, I can perform a comparison against the systypes table in the master database. Here's a query that does just that:

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

```
SELECT st.*
FROM dbo.systypes st
  LEFT JOIN master.dbo.systypes mst
    ON st.xusertype = mst.xusertype
WHERE mst.xtype IS NULL
```

Since user-defined data types are still based on system-specified data types, I don't think there is a whole lot of data to be obtained from this table. After all, whether or not I define the EmployeeID column as EmpID or int, I'm still dealing with an integer value. If a user has the ability to execute the sp_help table (it's back), not only can the user list all of the columns for a given table, but also can use the sp_help stored procedure to return information about a particular system stored procedure. Where this table can get me into trouble is if I have defined certain types with names like SSN or phone or MaidenName. Those types of eye-catching types will certainly cause an attacker to investigate further. After all, personal information is often a gold mine.

If I want to lock down the systypes table, I'll also need to be able to lock down the sp_help system stored procedure. Otherwise, an end user will be able to find out the definition of any particular user-defined data type.

The sysusers Table (insecure)

The sysusers system table contains all of the users and roles for a given database. Once again, the public role has access to this particular table. This means an end user can find out all of the usernames who have access to a given database. Needless to say, this is valuable information, as I've already discussed under sysmembers.

If I lock down sysusers, I can prevent end users and potential attackers from querying and finding out the users directly. One other thing I'll have to do is revoke execute rights on sp_helpuser. The system stored procedure sp_helpuser can return information on a specific user or it can return information and the public role has execute rights on it. Once again, if I revoke execute rights on sp_helpuser, I'll probably need to look at creating a role for db_securityadmin users who need access to sp_helpuser.

When I start looking at locking down system tables, system stored procedures follow. The problem that results is fixed database roles like db_securityadmin pay the price. These roles typically need to have access to the system stored procedures that are being locked down and this puts the DBA in a tough spot. Either the DBA locks down everything tight as can be and breaks db_securityadmin, or else the DBA leaves a door partially open by giving those personnel who will be db_securityadmin role members access to master with explicit access to the system stored procedures. Of course, if a particular person is a db_securityadmin in one database but not another, there's no way to restrict

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

the rights down to the database. The person could execute these system stored procedures in whatever databases the person has access to, regardless of database role.

In most of my work environments, roles like db_securityadmin weren't used at all. DBAs were considered responsible for security and this ruled out the need for such "lesser" roles. After all, who needs db_securityadmin when you're walking around with sysadmin, right? I understand the usefulness of these roles that don't have all the rights of db_owner or sysadmin, but I'll be honest in saying that if you use them, you have to carefully consider what you're going to lock down. I try not to use them because I know ultimately if there is a security breach, the DBA is whose head everyone is going to come looking for. As a result, delegating security is a dangerous business and typically I recommend that DBAs don't unless they are sure of the competence and attention to detail of the person they are giving those rights. After all, if I delegate security (unless I've got that management-directed order saying you have to and in that case I best keep it in a very safe place for later use), and the person I've delegated this responsibility to leaves a back door open someone exploits, I am the one on everyone's hit list.

MSDB Database "System" Tables

While the guest account is enabled by default in SQL Server, a DBA has the option of disabling it for msdb. Unlike master or tempdb, where the guest user is mandatory, msdb can be tightened down in this manner. However, there are repercussions, as might be expected.

Most of the tables in the msdb database used by the system are considered user tables based on their classification in sysobjects. For example, the backupfile table contains information on for each data or log file that has been backed up. However, if I check the type of this object in sysobjects, I find type equal to "U" for a user table. As a result, if I want to show the list of permissions for these tables, I'll have to modify my query I gave for sysprotects to look for a type of "U" instead of "S" in order to pick up these tables:

```
SELECT
    so.name [Table]
  , USER_NAME(sp.uid) [User],
    CASE action
      WHEN 26  THEN ' REFERENCES'
      WHEN 178 THEN ' CREATE FUNCTION'
      WHEN 193 THEN ' SELECT'
      WHEN 195 THEN ' INSERT'
      WHEN 196 THEN ' DELETE'
      WHEN 197 THEN ' UPDATE'
      WHEN 198 THEN ' CREATE TABLE'
      WHEN 203 THEN ' CREATE DATABASE'
      WHEN 207 THEN ' CREATE VIEW'
```


© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

```
        WHEN 222 THEN ' CREATE PROCEDURE'
        WHEN 224 THEN ' EXECUTE'
        WHEN 228 THEN ' BACKUP DATABASE'
        WHEN 233 THEN ' CREATE DEFAULT'
        WHEN 235 THEN ' BACKUP LOG'
        WHEN 236 THEN ' CREATE RULE'
      END [Action]
    , [Columns]
FROM sysobjects so
  LEFT JOIN sysprotects sp
    ON so.id = sp.id
WHERE so.type = 'U'
ORDER BY so.name
```

When I run the query, I find it returns over 200 rows but keep in mind I'm not returning individual tables, but permissions on those tables. Since DBA-created objects are rare in msdb, these are permissions on tables that ship with SQL Server. Though this query can return more than one row per table, there are still a large number of tables. Instead of going through the tables one-by-one, I'm going to look at them in groups. Related tables all share a common prefix, so figuring out the groups is easy. All the tables storing backup information begin with backup. All the tables dealing with log shipping begin with log_shipping.

If you run the query above, one of the first things you will likely notice is the public role doesn't have any rights on most of the tables. Unfortunately, some of the tables the public role does have access to do hold sensitive data, such as the backup and the restore tables. All told, there are 50 tables with permissions assigned to them, 46 of them with public role having access. Of the remaining 4, the TargetServersRole has rights, but not the public role. The groupings of these tables are (a * represents multiple tables with the prefix):

- backup*
- logmarkhistory
- mswebtasks
- restore*
- RTbl*
- syscategories
- sysdownloadlist
- sysjobs
- sysjobservers
- systargetservers

The last four tables are the ones where the TargetServerRole has rights. The reason the TargetServerRole has such permissions is for use with Multi-Server Administration. Since these four tables aren't accessible by public I won't discuss them further. Multi-Server Administration requires special configuration by the DBAs and thus the permissions and operations performed use privileged roles and users only. Another reason I won't cover these tables any further is because

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

in the OpenHack 4 configuration, Microsoft left these 4 tables as they were: with the TargetServerRole having access. As long as you don't explicitly grant a user access to msdb and place said user in the TargetServerRole, only sysadmins will have access to the tables.

Multi-Server Administration is a feature in SQL Server 7.0 and 2000 that allows a DBA to administer SQL Server Agent jobs across a farm of SQL Server installations all on just one or more SQL Servers. The SQL Servers that control jobs on "downstream" SQL Servers are known as master servers. The downstream servers are known as target servers. Jobs are created and managed on master servers and they then replicate the job information to the appropriate target servers. The target servers return events to the master server. All in all this seems like a good idea, however, in order to use Multi-Server Administration, prior to SP3 all servers must be in mixed mode because Multi-Server Administration uses SQL Server logins. With SP3 you can switch multi-server administration to use the Windows login the SQL Server Agent runs under, but the catch is that all SQL Servers have to be at least SQL Server 2000 SP 3. There's no mix and match allowed.

It's now time I delve into the 46 remaining tables existing in the msdb database. The first set of tables on the list is the backup tables.

The Backup Tables (insecure)

The four backup tables are all accessible by the public role. The backupfile table holds entries for each data and log file that has been backed up. The backupmediafamily table contains information of each of the media families on the server. A backup media family can consist of only one member and unless you're striping tapes or files for backups, and so if there is one row per family, that's not all that unusual. The backupmediaset table stores the media sets for the server. Media sets are made up of one or more media families. The backupset table has a row for each actual backup operation performed. There could be multiple backup sets on a particular backup media (such as 7 days of full backups on a single backup device).

I've talked a lot about physical file paths and both the backupfile and backupmediafamily tables contain file paths. An attacker could query these tables and learn the location of the backup files. Then, if the attacker can manage to break the security and retrieve the files, he or she may be able to restore the databases intact. There are additional methods to protect backup files including a new feature in SQL Server 2000 of password protecting backups. These additional measures make it that much harder for an attacker to successfully retrieve information from the backups I take to ensure my systems can be recovered. However, the password merely stops an attacker from restoring the data. There is no encryption in the process, meaning text strings

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

and the like are clearly visible. Therefore, I don't want regular users knowing where my backups are stored, much less having them access them.

Of the remaining two tables, the backupmediaset doesn't contain much usable information for an attacker. The backupset table, however, stores quite a bit of useful information. An attacker could retrieve when a database was backed up and who did it. This second piece of information, a user account, can be extremely valuable. After all, if I get a legitimate Windows login, I can then try standard password hacking techniques to try and get access to a system. The reason knowing a username is valuable is because if I attempt to login to Windows and I don't do so successfully, the error message I receive back tells me either the username or the password is invalid. In other words, unless I have a known good username, there's no guarantee the username I'm attempting to use is even valid. As a result, it wouldn't matter if I had all the computing power in the world trying to guess the password. If the user account doesn't exist, there's no access.

A good question here is "Why Microsoft chose to leave the backup tables open to everyone. It's a solution to the same problem we've seen before. Since db_owner and db_backupoperator roles need access to these tables in order to perform their backup duties, the only way to ensure a user has access to these tables is to grant rights to the public role. Otherwise, users would have to be explicitly added to the msdb database, a role would have to be created, and the role would have to be given access. This is the workaround I've discussed previously.

If I want to lock down this table, I must revoke SELECT rights from public against the four backup tables. There aren't any sp_help type stored procedures that reveal the same information contained in these four tables. So all permissions changes should be just on these four tables. Buf if I need to the db_owner or db_backupoperator backing up databases, I'll need to implement a workaround to the users who will be carrying out such activity. This means granting such users access to msdb and creating a role with rights to these tables. Then add those users to the role.

The LogMarkHistory Table (insecure)

The logmarkhistory table only comes into play if applications are using marked transactions. Otherwise this table stays blank. When a marked transaction is committed, the information about that marking is stored in this table and written to the transaction log. If need be, the transaction log can then be restored to the point of the mark. Do note, a marked transaction is different from a named transaction. The following is a named transaction:

```
BEGIN TRANSACTION MyTransaction
```


© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

A marked transaction uses the WITH MARK option:

```
BEGIN TRANSACTION AmarkedTransaction WITH MARK 'MyMark'
```

If I'm not using marked transactions, the LogMarkHistory table will remain empty and therefore secure. The public role only has SELECT permissions against this table. The user doesn't place data into the LogMarkHistory table. Instead this information is inserted automatically when the user commits a transaction using the WITH MARK option.

Looking back at the backup tables and forward to the restore tables, all the tables concerned with recovery of a database default to the public having permissions. After all, if a user has to restore a database to a marked point, it makes sense that the user might need to SELECT against the LogMarkHistory table. I'd be very cautious about applying any permissions changes to this table. Since restores tend to be rare and marked restores even rarer, I don't have a good recommendation on this particular table. However, if you aren't using marked transactions, there's no reason to secure the table.

The MSWebTasks Table (insecure)

The MSWebTasks table supports SQL Server's web tasks. A DBA can build web tasks by using the Web Publishing Wizard. Web tasks generate standard HTML web files from SQL Server data. If I want to produce static web pages (thus saving a trip back to the database server) when data changes, I can do so. By default, the public role has SELECT, INSERT, UDPATE, and DELETE rights against this system table. I'm not very keen on allowing people to see the web tasks I might have created. Likewise, I usually do not want that occurrence between two different users, each seeing the others web tasks. With the current model, all users can see all web tasks for a given system. I don't actively use web tasks so this table remains empty in my database installations. As a result, I am undecided on how best to secure this table.

The Restore Tables (insecure)

Like the backup tables, the restore tables are available to the public. The restorefile contains file path information indicating the exact location of the database files. An attacker could use this information to then target the actual files themselves, as I've discussed earlier. The restorefilegroup references the restorehistory table and lists the filegroups that have been restored. If I am only using the PRIMARY filegroup (in a high-performance system I might have multiple file groups), an attacker isn't going to get much information. The restorehistory table, like the backupset table, contains the Domain and User Acount for the user who performed the restore. Once again, this gives an

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

attacker known good usernames with which to use when trying to crack the password.

The reason the public role would need access to these tables is the check the restore history of a particular database. Unless a workaround is implemented, a given user, regardless of rights in another database, can't retrieve information from these tables. Those users who are db_owners trying to restore their database or checking to see if a DBA did so would need to go against this table.

The RTbl Tables (not sensitive)

The RTbl tables, as they are called in Books Online, are the tables that support Microsoft Repository within SQL Server. Their use and structure is completely dependent upon the users who have access to the SQL Server. The repository tables are not required for the operation of SQL Server. I don't know many who are actively using this feature of SQL Server because Microsoft Repository can be a difficult animal. However, since the repository tables aren't necessary for SQL Server, I don't consider them sensitive.

They can be sensitive depending on what information is stored in them. Service Pack 3 disables the ability to save DTS packages to Meta Data Services (the Repository). One of the reasons for this is that when a DBA saves a DTS package to Meta Data Services, the package cannot use password protection to encrypt and secure the contents. As a result, the DTS package can be opened up by anyone who can get to it using the Repository. If I choose to save a package to SQL Server or to a Structured Storage File (not a Visual Basic file, which also cannot have a password set), I have the option to specify a password. This is considered a best practice. If I do decide to save mission critical DTS packages into the repository, then I have placed something of value to an attacker in the repository. This isn't a good idea, especially considering the lack of a password. Also, packages saved to the repository appear to have a slightly higher corruption rate.[22]

Another point I'll make about Meta Data Services/Microsoft Repository is when Microsoft received the C2 Security Certification for SQL Server 2000, Meta Data Services was not part of the mix. All the objects that made up Meta Data Services were dropped for the C2 approved configuration.

The syscategories table (not sensitive)

The syscategories table contains the categories for grouping jobs, alerts, and operators. Since these categories are arbitrary and since they are used for the convenience of those that maintain a particular SQL server, their storage in the syscategories table in and of itself is not a security risk. After all, I've seen

[22] Knight, *Where should I save my DTS packages?*

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

numerous servers where no categories have been added or removed. The syscategories tables were populated with the same default values such as "Database Maintenance."

The syscategories table doesn't contain anything about file paths, users, internal structure, or anything else sensitive unless we create categories that reveal such information. As a result, the syscategories table isn't inherently sensitive unless we make it so.

The sp_help* System Stored Procedures

There are many, many system stored procedures. Some I've already discussed as I talked about locking down system tables. My goal in this section is to discuss some of the sp_help family of stored procedures. All told there are some 400 stored procedures in master alone that reference system tables which public has access to. Many are internal stored procedures built for Enterprise Manager and the other SQL Server client tools.

Determining What Stored Procedures Are Tied to System Tables

Most of the stored procedures I've talked about in the section on system tables revealed too much information to the end user. In order to identify what stored procedures access a particular system table, I've wrote a simple script to retrieve the information for me. However, since I'm doing a like comparison (remember, I've already indicated that sysdepends isn't always dependable), there may be some false positives. The script doesn't distinguish between the use of a system table name in a query or a comment. It just looks for a "hit" in the text.

As far as database system tables like sysobjects are concerned, I can't run in the database in question and get the system stored procedures because the query only looks at the syscomments table in the current database (meaning if I'm in Pubs, I won't query against syscomments in master and thus won't get the system stored procedures located there).

However, since master is a database with the same tables such as sysobjects, I can run the query in master to determine what system stored procedures access a particular system table. It can be startling how many stored procedures public has access to that access the system tables.

```
-- Change @TableName to the table you want to return the list of
-- stored procedures on. This also will work for user tables and
-- stored procedures.
DECLARE @TableName sysname
SET @TableName = 'sysobjects'

SELECT DISTINCT USER_NAME(so.uid) [Owner],
                so.name [Stored Procedure]
```


© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

```
FROM sysobjects so
  JOIN syscomments sc
    ON so.id = sc.id
  JOIN syspermissions sp
    ON so.id = sp.id
WHERE so.xtype = 'P' AND
      sc.text LIKE '%' + @TableName + '%' AND
      sp.grantee = USER_ID('public')
ORDER BY Owner, [Stored Procedure]
```

If I want to go ahead and figure out every stored procedure accessing system tables, I can wrap this query into a larger one. However, since the larger query involves the use of a cursor (and it has to cycle through each system table), it can take a bit of time to run. Here is that larger query:

```
-- The variable where the system table name will be stored
DECLARE @TableName sysname

-- Declare a cursor to cycle through all system tables
DECLARE cursSystemTables CURSOR FAST_FORWARD
FOR
SELECT DISTINCT name
FROM sysobjects
WHERE xtype = 'S'

-- Open the cursor
OPEN cursSystemTables

-- "Prime the pump"
FETCH NEXT FROM cursSystemTables INTO @TableName

-- Create a temporary table to hold our results
CREATE TABLE #SProcs (
  Owner sysname,
  SProc sysname
)

-- Cycle through the cursor
WHILE (@@FETCH_STATUS = 0)
BEGIN
  -- Insert results into temp table
  INSERT INTO #SProcs
  (Owner, SProc)
  SELECT DISTINCT USER_NAME(so.uid), so.name
  FROM sysobjects so
    JOIN syscomments sc
      ON so.id = sc.id
    JOIN syspermissions sp
      ON so.id = sp.id
  WHERE so.xtype = 'P' AND
        sc.text LIKE '%' + @TableName + '%' AND
        sp.grantee = USER_ID('public')

  -- Fetch next system table in the cursor
  FETCH NEXT FROM cursSystemTables INTO @TableName
```


© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

```
END

-- Clean up on cursor
CLOSE cursSystemTables
DEALLOCATE cursSystemTables

-- Retrieve stored procedures from temp table
SELECT DISTINCT Owner, Sproc
FROM #SProcs
ORDER BY Owner, SProc

-- Clean up on temp table
DROP TABLE #SProcs
```

The Purpose of the sp_help Stored Procedures*

The sp_help set of stored procedures was built to provide help on various aspects of SQL Server, hence the name. For instance, I can use the sp_helpdb stored procedure to get more information about a particular database.
Some of these stored procedures may not run successfully because of how I've chosen to lock down the system tables. For instance, when I locked down the sysfiles table in the Pubs database (the public role did not have permission to execute SELECT statements against it), sp_helpdb worked, but returned a SELECT denied error. The reason is that sp_helpdb does have some dynamic SQL in order to build the query to return information. Specifically, these lines from sp_helpdb refer to sysfiles.

```
select @exec_stmt = 'update #spdbdesc
        set dbsize = (select str(convert(dec(15),sum(size))* '
        + @low
        + '/ 1048576,10,2)+ N'' MB'' from '
        + quotename(@name, N'[') + N'.dbo.sysfiles)
WHERE current of ms_crs_c1'

execute (@exec_stmt)
```

As I discussed earlier, dynamic SQL will execute in a separate batch than the calling batch process. As a result, ownership chains no longer apply and permissions are re-evaluated. Even though public might have access to sp_help and sp_help is making the call to sysfiles, because this stored procedure uses dynamic SQL I'll get an access denied error.

A Partial List

With that said, a partial list of sp_help stored procedures (choices I've made in configuration such as SQL Server Notification Services, full-text indexing, and replication may alter the list) that are accessible by the public role are:

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

- sp_help
- sp_help_agent_default
- sp_help_agent_parameter
- sp_help_agent_profile
- sp_help_fulltext_catalogs
- sp_help_fulltext_catalogs_cursor
- sp_help_fulltext_columns
- sp_help_fulltext_columns_cursor
- sp_help_fulltext_tables
- sp_help_fulltext_tables_cursor
- sp_help_publication_access
- sp_helparticle
- sp_helparticlecolumns
- sp_helparticledts
- sp_helpconstraint
- sp_helpdb
- sp_helpdbfixedrole
- sp_helpdevice
- sp_helpdistpublisher
- sp_helpdistributiondb
- sp_helpdistributor
- sp_helpdistributor_properties
- sp_helpextendedproc
- sp_helpfile
- sp_helpfilegroup
- sp_helpgroup
- sp_helpindex
- sp_helplanguage
- sp_helplinkedsrvlogin
- sp_helplog
- sp_helplogins
- sp_helpmergealternatepublisher
- sp_helpmergearticle
- sp_helpmergearticlecolumn
- sp_helpmergecleanupwait
- sp_helpmergefilter
- sp_helpmergepublication
- sp_helpmergepullsubscription
- sp_helpmergesubscription
- sp_helpntgroup
- sp_helppublication
- sp_helppullsubscription
- sp_helpremotelogin
- sp_helpreplfailovermode

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

- sp_helpreplicationdb
- sp_helpreplicationdboption
- sp_helpreplicationoption
- sp_helprole
- sp_helprolemember
- sp_helprotect
- sp_helpserver
- sp_helpsort
- sp_helpsql
- sp_helpsrvrole
- sp_helpsrvrolemember
- sp_helpstats
- sp_helpsubscriberinfo
- sp_helpsubscription
- sp_helpsubscription_properties
- sp_helptext
- sp_helptrigger
- sp_helpuser

All told, there are over 60 sp_help stored procedures accessible by the public role in the master database. There are additional sp_help stored procedures in msdb as well.

- sp_help_category
- sp_help_job
- sp_help_jobhistory
- sp_help_jobschedule
- sp_help_jobschedule
- sp_help_jobserver
- sp_help_jobstep
- sp_help_jobstep
- sp_helphistory
- sp_helptask

I've already discussed some of them and what they reveal, but what you can lock down in your environment is going to differ from what I can lock down. In reality, what can and cannot be locked down may even differ depending on the role the SQL Server will play. For instance, it may be a good idea to allow developers on a SQL Server development box access to some of the sp_help stored procedures that I wouldn't want a typical user running in production. An sp_help stored procedure that fits this criteria is sp_helptrigger. Developers working on building a new application will probably need this stored procedure is they are building triggers for auditing, additional data validation, etc., but certainly no regular needs it in production.

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

My advice given the large number of stored procedures is to build a checklist for each environment. From that checklist, a DBA can develop scripts that carry out the appropriate REVOKE permission statements to remove the public role's access to the particular sp_help stored procedures. If I can successfully revoke public access to all system stored procedures as Microsoft did for OpenHack 4, I'm on my way to a much more secure system. This isn't usually possible, but it's the goal to shoot for.

And Then There was OpenHack 4

OpenHack is a hacking competition put on by the magazine eWeek. OpenHack 4, also known as OpenHack 2002, was the fourth time such a competition has been put on by eWeek (as the name would imply). The premise for OpenHack 4 was simple: break into a web application that was hardened to the best of two vendors' abilities. Those two vendors were Microsoft and Oracle.

One of the initial problems Microsoft faced in OpenHack 4 was the web application itself. It was written in Java Server Pages (JSP), a technology that doesn't come natively with Microsoft's Internet Information Server (IIS). So Microsoft rewrote the application in C# and deployed it using ASP.NET and IIS. Of course, Microsoft chose SQL Server 2000 as its database server. And this is where our interest as DBAs should perk up.

OpenHack 4 focused on application security because that had been shown to be a weakness in previous OpenHack competitions. As a result, the basic application was developed by eWeek, but then Microsoft and Oracle were left to their own devices to figure out how to harden the application. Microsoft's approach to hardening SQL Server was eye-opening.

What Microsoft Did

Microsoft revoked public access on all system tables in the master database. The same was true of all system stored procedures. Then, the public role was given access to the table spt_values and the system stored procedure sp_MShasdbaccess. Other than some of the functions in master that have permissions that cannot be removed from public, nothing else was spared.

Actually, the script that was used worked on all the databases on the server (with the exception of Northwind and Pubs, which were dropped). The exception was the script didn't touch the TargetServersRole in the msdb database. In Microsoft's documentation from OpenHack 4, they left the TargetServersRole permissions alone. The only other permissions added were specific to the application database. There access was restricted to a role called webuser. In

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

keeping with best practices, Microsoft only exposed a small set of stored procedures, no base tables, for access by this role.[23] [24]

So the bottom line is that Microsoft revoked all permissions from public and then added back: spt_values and sp_MShasdbaccess. That's it. At this point let me point out that both vendors did well, with Oracle suffering a small break and Microsoft coming through completely unscathed. What does this mean? It means Microsoft's approach to lock down its application was 100% successful. Since OpenHack is just that, open to anyone, that meant Microsoft's environment took on and defeated all comers. From a SQL Server perspective, that's exactly what I want to hear.

Is It Really That Simple?

In a word: no. Microsoft's OpenHack approach worked because they were coding a .NET application and didn't use tools like Microsoft Access or things like DSN connections that our users have to make their lives easier. When I go about locking tables down like what Microsoft did, functionality from these other pieces of a typical business environment are likely to have issues.

I also am obliged to point out touching permissions on system tables puts SQL Server in a non-standard configuration. Therefore, there's no guarantee that Microsoft will support the SQL Server should a problem arise if the default permissions are changed. Of course, this is true of any change that I make to any system object, but it behooves me to state this as a reminder.

Concluding Thoughts

The public role has permissions far and wide across the master, msdb, and individual user databases. Most of these permissions represent more rights than a normal user needs and is a violation of the Principle of Least Privilege. In some cases the system tables the public role has access to can be tightened down but there are potential consequences. As is typical, there has to be a compromise between security and functionality.

I've covered the system tables the public role has access to and explained how, through the guest user, a normal login has access to system tables in the master and msdb databases. The guest user can be disabled in msdb, but master requires guest. Since the guest user is a member of the public role (all users are), any login has the ability to access the system tables for which the public role has access.

---

[23] Knight, *10 Steps to Securing Your SQL Server*
[24] Andrews, *SQLSecurity Checklist*

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

Also, due to cross-database ownership chaining, certain sp_help stored procedures are available which can provide information contained in these same system tables. Again, the public role has the ability to execute a significant number of these, providing additional avenues for reconnaissance.

In both the cases of the system tables and the sp_help stored procedures, the public role's permissions can be revoked. Depending on the applications supported, however, crucial functionality may be lost. In the case of a custom application written specifically with security in mind, almost all permissions the public role has can be removed. Microsoft demonstrated this in its own configuration during the OpenHack 4 competition. In reality, however, most configurations probably will have a hard time duplicating such a setup. Even if they do, there is always the proviso that such a setup is not a recognized and supported configuration by Microsoft.

Whether or not we can revoke the permissions, it is important to understand the scope of the public role and what it grants a normal user. This scope is very broad and goes beyond whatever security a DBA may have set up for a specific application. Hopefully this paper has provided additional information in understanding the breadth of the public role in Microsoft SQL Server.

© SANS Institute 2003, Author retains full rights.

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

References

Andrews, Chip. "SQLSecurity Checklist." 2003. SQLSecurity.com. 14 October 2003. URL: http://www.sqlsecurity.com/DesktopDefault.aspx?tabindex=3&tabid=4

Chigrik, Alexander. "SQL Server 2000 Undocumented System Tables." 2003. MSSQLCity.com. 14 October 2003. URL: http://www.mssqlcity.com/Articles/Undoc/SQL2000UndocTbl.htm

Chigrik, Alexander. "SQL Server 2000 Useful Undocumented Stored Procedures." 2003. MSSQLCity.com. 14 October 2003. URL: http://www.mssqlcity.com/Articles/Undoc/SQL2000UndocSP.htm

"FIX: Microsoft Search Service May Cause 100 % CPU Usage if BUILTIN\Administrators Login Is Removed." 22 May 2003. Microsoft Knowledge Base. 14 October 2003. URL: http://support.microsoft.com/default.aspx?scid=kb;en-us;295034

Kelley, K. Brian. "SQL Server 2000 SP 3: What's New in Security." 5 Jun 2003. SQLServerCentral.com. 14 October 2003. URL: http://www.sqlservercentral.com/columnists/bkelley/sp3coresecurity.asp

Kelley, K. Brian. "SQL Server Security: Login Weaknesses." 14 August 2003. SQLServerCentral.com. 14 October 2003. URL: http://www.sqlservercentral.com/columnists/bkelley/sqlserversecurityloginweaknesses.asp

Kelley. K. Brian. "SQL Server Security: Why Security Is Important." 31 July 2003. SQLServerCentral.com. 13 October 2003. URL: http://www.sqlservercentral.com/columnists/bkelley/sqlserversecuritywhysecurityisimportant.asp

Knight, Brian. "10 Steps to Securing Your SQL Server." 3 April 2003. SQLServerCentral.com. 14 October 2003. URL: http://www.sqlservercentral.com/columnists/bknight/10securingyoursqlserver.asp

Knight, Brian. "Where should I save my DTS packages?" 15 November 2001. SQLServerCentral.com. 14 October 2003. URL: http://www.sqlservercentral.com/columnists/bknight/savingpackages.asp

Kowles, Douglas. "Digispid.B.Worm." 3 June 2002. Symantec Security Response Center. 13 Oct 2003. URL: http://securityresponse.symantec.com/avcenter/venc/data/js.spida.b.html

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

Knowles, Douglas. "W32.SQLExp.Worm." 4 February 2003. Symantec Security Response Center. 13 Oct 2003. URL: http://securityresponse.symantec.com/avcenter/venc/data/w32.sqlexp.worm.html

Kondreddi, N. Vyas. "Overview of SQL Server Security Model and Security Best Practices." 20 May 2003. Vyas Kondreddi's home page. 13 October 2003. URL: http://vyaskn.tripod.com/sql_server_security_best_practices.htm

Litchfield, David. "Microsoft SQL Server Passwords (Cracking the password hashes)." 24 June 2002. NGSSoftware. 14 October 2003. URL: http://www.nextgenss.com/papers/cracking-sql-passwords.pdf

OpenHack 2002 Downloads. 2 December 2002. eWeek. 14 October 2003. URL: http://www.eweek.com/article2/0,4149,743002,00.asp

Moore, David, *et. al.*, "The Spread of the Sapphire/Slammer Worm." 07 Feb 2003. CAIDA. 15 October 2003. URL: http://www.caida.org/analysis/security/sapphire/

"Microsoft Security Bulletin (MS00-006)." 31 March 2000. Microsoft TechNet. 14 October 2003. URL: http://www.microsoft.com/technet/security/bulletin/MS00-006.asp

"Microsoft SQL Server 2000 C2 Evaluation." 2003. Microsoft TechNet. 14 Oct 2003. URL: http://www.microsoft.com/technet/security/prodtech/dbsql/sqlc2.asp

"PRB: Removal of Guest Account May Cause Handled Exception Access Violation in SQL Server." 22 May 2003. Microsoft Knowledge Base Article 315523. 13 October 2003. URL: http://support.microsoft.com/default.aspx?scid=kb;en-us;315523

Regan, Keith. "Top Dog Oracle Losing Database Market Share." 11 March 2003. E-Commerce Times. 15 October 2003. URL: http://www.ecommercetimes.com/perl/story/20968.html

shoeboy. "Some analysis of Microsoft SQL Server 2000 stored procedure encryption." 17 December 2001. BugTraq Mailing List. 14 October 2003. URL: http://www.securityfocus.com/archive/1/246134

"SQL Server 2000 Security Model." 2003. Microsoft TechNet. 14 October 2003. URL: http://www.microsoft.com/technet/prodtechnol/sql/maintain/security/sp3sec/SP3SEC01.ASP

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

SQL Server Books Online (Updated for SP3). 17 January 2003. Microsoft. 14 October 2003. URL: http://www.microsoft.com/sql/techinfo/productdoc/2000/books.asp

Warren, Andy. "Using the Public Role to Maintain Permissions." 10 May 2001. SQLServerCentral.com. 13 October 2003. URL: http://www.sqlservercentral.com/columnists/awarren/sqlpermissionspublicrole.asp

© SANS Institute 2003, Author retains full rights

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

Last Updated: September 2nd, 2015

Upcoming SANS Training

Click Here for a full list of all Upcoming SANS Events by Location

SANS Crystal City 2015	Crystal City, VAUS	Sep 08, 2015 - Sep 13, 2015	Live Event
SANS Network Security 2015	Las Vegas, NVUS	Sep 12, 2015 - Sep 21, 2015	Live Event
SANS Seoul 2015	Seoul, KR	Sep 14, 2015 - Sep 19, 2015	Live Event
Data Breach Investigation Summit & Training	Dallas, TXUS	Sep 21, 2015 - Sep 26, 2015	Live Event
SANS Baltimore 2015	Baltimore, MDUS	Sep 21, 2015 - Sep 26, 2015	Live Event
SANS Perth 2015	Perth, AU	Sep 21, 2015 - Sep 26, 2015	Live Event
SANS Tallinn 2015	Tallinn, EE	Sep 21, 2015 - Sep 26, 2015	Live Event
SANS ICS Amsterdam 2015	Amsterdam, NL	Sep 22, 2015 - Sep 28, 2015	Live Event
SANS Bangalore 2015	Bangalore, IN	Sep 28, 2015 - Oct 17, 2015	Live Event
SANS DFIR Prague 2015	Prague, CZ	Oct 05, 2015 - Oct 17, 2015	Live Event
SANS Seattle 2015	Seattle, WAUS	Oct 05, 2015 - Oct 10, 2015	Live Event
SANS Tysons Corner 2015	Tysons Corner, VAUS	Oct 12, 2015 - Oct 17, 2015	Live Event
SOS: SANS October Singapore 2015	Singapore, SG	Oct 12, 2015 - Oct 24, 2015	Live Event
SANS Gulf Region 2015	Dubai, AE	Oct 17, 2015 - Oct 29, 2015	Live Event
SANS Cyber Defense San Diego 2015	San Diego, CAUS	Oct 19, 2015 - Oct 24, 2015	Live Event
SANS Tokyo Autumn 2015	Tokyo, JP	Oct 19, 2015 - Oct 31, 2015	Live Event
SANS South Florida 2015	Fort Lauderdale, FLUS	Nov 09, 2015 - Nov 14, 2015	Live Event
SANS Sydney 2015	Sydney, AU	Nov 09, 2015 - Nov 21, 2015	Live Event
SANS London 2015	London, GB	Nov 14, 2015 - Nov 23, 2015	Live Event
Pen Test Hackfest Summit & Training	Alexandria, VAUS	Nov 16, 2015 - Nov 23, 2015	Live Event
SANS Hyderabad 2015	Hyderabad, IN	Nov 24, 2015 - Dec 04, 2015	Live Event
SANS Cape Town 2015	Cape Town, ZA	Nov 30, 2015 - Dec 05, 2015	Live Event
SANS San Francisco 2015	San Francisco, CAUS	Nov 30, 2015 - Dec 05, 2015	Live Event
SANS Milan 2015	OnlineIT	Sep 07, 2015 - Sep 12, 2015	Live Event
SANS OnDemand	Books & MP3s OnlyUS	Anytime	Self Paced