SQL Server – Stored Procedure Attacks

INFORMATION IN THIS CHAPTER

- How Stored Procedure Attacks Work
- Dangers Associated with a Stored Procedure Attack
- The Future of Stored Procedure Attacks
- Defense against Stored Procedure Attacks

The acronym SQL actually stands for Structured Query Language, which is the standard programming language utilized to access and manipulate databases. For example, from a security perspective, you probably have heard of “SQL Injection” as a form of attack against SQL databases. Because of the name SQL Server, you may think that this is a Microsoft-specific vulnerability; however, the SQL in SQL Injection is actually referring to the language rather than Microsoft’s product. This makes it a valid attack against all databases that allow SQL queries rather than a vulnerability specific to the Microsoft product.

Microsoft’s SQL Server application has been around for a long time and has become more secure with each new release. Although SQL Server has had many versions, there are really only five versions that you may run into today; these are versions 6.5, 7.0, 2000, 2005, and 2008. As you would expect, each version has its own quirks, which include both features to use and vulnerabilities that can be exploited. In all cases, the Microsoft developers have included the ability to leverage reusable code to perform functions through the use of procedures stored within the database application itself. In the SQL Server world, these pieces of reusable code are known as stored procedures.

Stored procedures are a series of SQL statements that perform predefined tasks. This programming style is based on creating programming code to perform some specific task or function and storing it for use by your programs. This saves the

\(^{A}\)SQL Injection is discussed in detail in Mike Shema’s *Seven Deadliest Web Application Attacks* (Syngress, ISBN: 978-1-59749-543-1) and Clarke’s, *SQL Injection Attacks and Defense* (Syngress, ISBN: 978-1-59749-424-3) as well as in conjunction with stored procedures later in this chapter.
developer’s time and effort when writing new programs because instead of having to repetitively write all of the code to perform some task, they are able to call existing functions to get the desired results.

Think about it in terms of your real life. Washing clothes used to be a time-consuming and boring task. To wash your incredibly prolific T-shirt collection featuring the characters of Star Trek: The Next Generation (or “TNG” as the cool insiders call it), you would have had to fill up a tub with water and soap; drop in your “Picard > Kirk,” “What happens on the Holodeck, stays on the Holodeck,” and “Just say NO to assimilation” T-shirts and wash them in the soapy water (usually by rubbing each one against a wash board to get out all of the dirt, grime, and salsa stains); then refill the tub with clean water and rinse each individual T-shirt to get out the soap. Today, you just drop these clothes into a machine that performs all of the washing functions by just turning it on. Not only does this save you the effort of having to wash the clothes yourself, it also provides you with a repeatable process that you can now use for your set of Battlestar Galactica gym shorts.

By implementing stored procedures, the developer is not only able to perform a specific task or function with a single call, but also able to increase the performance of their applications. This is the case because instead of sending a long query string to the database over a network, the developer sends a short statement, which executes the stored commands locally on the server. Finally, since stored procedure calls are embedded into many precompiled programs, the developer can change the results of many programs by just changing the programming of the stored procedure itself.

In addition to providing the ability for developers to create and store their own procedures for reuse, SQL Server comes prepackaged with stored procedures from Microsoft that allows a user to administer the database itself. These well-known procedures should specifically concern you as a security practitioner rather than custom-stored procedures created by your own database administrators (DBAs) or developers. Although custom procedures can be just as powerful as those provided by Microsoft (or well-known applications that run on top of SQL Server), attackers generally don’t want to waste time figuring out what these functions are until all other avenues of attack have failed. Discovering you are running SQL Server, however, or an application that relies on SQL Server and stored procedures for its own use, the attacker may identify an attack vector he can use to either steal data directly from the database or escalate his privileges.

NOTE
Like so many other Microsoft products, SQL Server did not begin its life with Microsoft. Sybase was the original author of SQL Server and Microsoft was brought in with Ashton-Tate as partners to assist in porting it to OS/2. Ashton-Tate eventually stepped aside and Microsoft ended up porting the product to Windows NT on its own. In 1993, the partnership agreement between Microsoft and Sybase ended. Sybase continued development for UNIX, eventually renaming it to Adaptive Server Enterprise (ASE) with Microsoft keeping the original name for its Windows-only product.


**HOW STORED PROCEDURE ATTACKS WORK**

As you would hope from a security perspective, stored procedures are not always available for attackers to use right out of the box. For example, SQL Server may not have stored procedures available for you to utilize (an administrator may have removed them or they may be disabled by default), and it does require you to have appropriate permissions when accessing these procedures. Certain conditions, therefore, may need to exist before initiating an attack utilizing SQL's stored procedures.

**Initiating Access**

The first step in the attack methodology is to obtain access to accounts or applications with proper permissions to interact with the stored procedures. A common SQL Server account that is fruitful for attackers to gain access to and leverage is the pre-built administrator account that is named System Administrator or “sa” by default. This account is created as part of the initial installation for SQL Server; however, any account with appropriate permissions will do.

**WARNING**

“sa” is the legacy account that acts as an administrator-level account for managing SQL Server tasks and also provides full control over the database instance and its data. The “sysadmin” fixed server role is designed to provide accounts assigned to the role full control over all aspects of the SQL Server instance it is a part of. By default, the sa account is assigned to the sysadmin role, making it a prime target for attackers.

Access to a valid account can be accomplished through several methods depending on the access an attacker already has to the network or the database instance. One of the most common methods for gaining access to a sysadmin fixed server role account is to perform password guessing or dictionary attacks against the default sa account. All too often, administrators fail to configure accounts with strong passwords (or any password at all for that matter). Depending on what version of SQL Server is implemented and what password policies are implemented, account lockouts may or may not be enabled to limit these attacks. Finally, DBAs may have turned off auditing for failed logon attempts because of “performance” reasons or the events are created, but there is no monitoring of the logs. This type of configuration will allow attackers to conduct password attacks against the SQL Server that may go unnoticed.

In SQL Server 2008, the sa account is present whether mixed mode authentication or Windows authentication is selected as the authentication mode. However, in the case of Windows authentication mode, the sa account is left disabled. In order to ensure compatibility with legacy applications and database interaction, many administrators will configure servers to use mixed mode authentication and enable the sa account.

In SQL Server 2005 and 2008, administrators are forced to provide a password for the account; however, this was not the case with earlier versions. After the initial configuration of these early versions, sysadmins are able to set a password with a null
value. In a security-conscious world, the ability to leave the password blank wouldn’t be a big deal, because anybody who cares about security would never set it that way. Unfortunately, in most cases, it is actually DBAs who handle the security within an SQL Server, and that means it is possible that the convenience of a blank password will trump security (this situation almost always means that performance trumps security, which has its own implications).

**Accessing Stored Procedures**

Once an attacker has administrative control over the SQL Server instance, attacks can be leveraged against the stored procedures implemented on the server. Stored procedures come in different flavors and provide different functionalities. For Microsoft’s SQL Server, three main categories of stored procedures exist:

- User-defined stored procedures are implemented to maximize code reuse and user-defined operations via Transact-SQL (T-SQL) statements or using the .NET framework Common Language Runtime (CLR).
- Extended stored procedures allow database developers to create reusable code in languages such as C. This is a legacy method and will be removed at some point in the future.
- System-stored procedures provide administrative interfaces for some of the administrative management of the SQL Server instance.

Accessibility of stored procedures will depend on the version of SQL Server installed and the configuration of the server. In the last several versions of SQL Server, Microsoft has slowly implemented controls and configuration changes to the

---

**TIP**

The sqlcmd utility is new as of SQL Server 2005 and provides additional features and options as compared to the osql utility. In some cases, the osql utility may not be compatible with all of the features found in SQL Server 2005 and 2008. Microsoft recommends using the sqlcmd utility to ensure compatibility with the new features found in these versions. In this chapter, we will be using the sqlcmd utility for our examples, as many of the commands are identical in comparison to the legacy osql utility.

Executing stored procedures interactively using the sqlcmd utility is a fairly straightforward task. Once a valid account is obtained, an administrator may use the sqlcmd utility to connect to the SQL Server and execute commands to access data or perform functions. Successful connection to the SQL Server with the sqlcmd utility will enable you to execute commands in a command-line environment.

---

How Stored Procedure Attacks Work

The database engine stored procedure “sp_configure” allows configuration of many options globally on the SQL Server instance. Using sp_configure to reenable the stored procedure will allow the administrator to continue on with the task at hand.

The default implementation of SQL Server in an attempt to reduce the exploitation of some of the more well-known vulnerabilities associated with SQL Server.

Depending on the SQL Server version and the implemented configuration, stored procedures may or may not be enabled. Figure 3.1 provides an example of an administrator connecting to the SQL Server and attempting to leverage the functionality of the xp_cmdshell extended stored procedure. The initial error message indicates that the requested stored procedure is disabled and the administrator is not able to successfully complete the command as requested; however, if the stored procedure has not been fully removed, the administrator can reenable the stored procedure with a few simple commands, assuming that the administrator has appropriate permissions to do so.

FIGURE 3.1
Enabling xp_cmdshell Stored Procedure
DANGERS ASSOCIATED WITH A STORED PROCEDURE ATTACK

The question you may be thinking right now is, what is the point of using a stored procedure attack if you already require sysadmin-level privileges prior to executing it? This is a valid question because if you already have sysadmin-level privileges, then you have the ability to create and manage privileges within the database, the ability to manipulate any part of the databases stored within SQL, and access to all of the data. Therefore, the point of the attack cannot be to gain administrative privileges within the database itself. If you already have everything you need to walk in through the front door of a building, the question becomes, what do you get by using the service entrance?

In this case, the service entrance gives you the authority to roam the whole building instead of just the common areas that visitors see. The combination of stored procedures and your sysadmin role access allows you to utilize SQL Server as your attack platform to defeat the server and any additional applications running on a shared server (this could mean owning the domain, if the SQL Server application is installed on a Domain Controller). In addition, stored procedures attacks can be used in conjunction with other SQL Server attacks, such as SQL injection, to gain this same authority without requiring sysadmin-level access prior to the beginning of the attack.

Understanding Stored Procedure Vulnerabilities

Historically, there have been numerous vulnerabilities identified in Microsoft SQL Server stored procedures. Some of the vulnerabilities are directly related to the code implemented to support the stored procedures, while other vulnerabilities stem from the functionality some of the stored procedures provide. A few of the categories for attacks against stored procedures experience over time include excessive privileges, buffer overflows, and trojanted stored procedures.

- **Excessive privileges** Some of the stored procedures preinstalled on SQL Server allow the execution of commands on the underlying operating system. This type of relationship between the SQL Server and the operating system allows attackers to leverage system commands that can cause an immediate impact on the security of the SQL Server and the supporting operating system.

- **Buffer overflows** In the past, several stored procedures have experienced issues with exception handling for receiving parameters in the context of a stored procedure causing the return address of the call to be overwritten. A buffer overflow condition can allow attackers to take control of the next instruction performed on the system and subsequently allow for arbitrary commands to be executed. These conditions may allow for attackers to interact with the core operating system and may also include causing denial of service conditions.

- **Trojans** Attackers who are able to gain access to the underlying operating system have been able to replace legitimate Dynamic-Link Libraries (DLLs), applications,
and executable files with files that appear to be the legitimate but have been modified. Stored procedures are sourced from a series of DLLs and modification of the stored procedure functions within the DLLs can allow execution of code that runs under the context of the SQL Server.

Microsoft has done a fairly good job at documenting stored procedures and the capabilities they provide. Not all of the stored procedures available, however, are documented by Microsoft and administrators may not fully understand some of the security issues implementing stored procedures could cause.

Some of the notable stored procedures that allow attackers to interact with and glean information from the SQL Server include:

- **xp_cmdshell** This extended stored procedure allows members of the sysadmin fixed server role to execute commands in the context of the permissions associated with that of what account the SQL Server service is running under.
- **xp_enumgroups** As the name of the stored procedure indicates, this extended stored procedure allows members of the sysadmin and db_owner fixed server roles to enumerate group membership information from the local or domain groups specified in the stored procedure call.
- **sp_addlogin** This is a system stored procedure that creates a new user account that can be used for authentication to the SQL Server. However, Microsoft documentation indicates that this stored procedure will be removed in a future version of SQL Server. In addition, Microsoft recommends using Windows authentication as an alternative to this method.
- **sp_addsrvrolemember** This adds an existing account to a specified group within the SQL Server instance.
- **xp_grantlogin** This stored procedure assigns the appropriate permissions that allow the defined Windows security group or account to connect to the SQL Server.
- **xp_logininfo** This provides information about a specific account or a group of accounts and the level of access the account has. The stored procedure can also return information about accounts and group membership.
- **xp_regread** This stored procedure returns the values associated with registry keys found on the SQL Server.
- **xp_regenumvalues** This provides a list of all the values located under a specific registry key.
- **xp_regwrite** This stored procedure is used to write entries to the system registry.
- **xp_msver** This provides information about the version of the SQL Server instance, as well as the underlying operating system.
- **xp_servicecontrol** This controls the state of the operating system services. This stored procedure can be used to start, stop, pause, continue, and query state any service the sa or sysadmin fixed server role has permissions for.

Examples of some of the common attacks against stored procedure implementations are provided to help illustrate some of the concepts discussed. Although a few examples are provided for clarity of what an attacker may do, the sky is the limit if you
have a good imagination and think like an attacker. The following scenarios assume that the stored procedures have already been enabled as previously discussed.

**Scenario 1: Adding a Local Administrator**

One of the most common attack scenarios leveraged today involves using stored procedures to add user accounts to the SQL Server host operating system. This scenario involves an attacker successfully authenticating and connecting to an SQL Server using the sa account with a weak password. Unfortunately, in the field, it is fairly common to find SQL Server databases using SQL Server authentication and allowing access via the sa or other application accounts assigned to the sysadmin fixed server role.

**WARNING**

Although this chapter focuses on the risks stored procedures can create, it should also be obvious to readers that poorly implemented passwords for databases will allow access to the contents of the database. This may include viewing contents of the database or dropping tables of the database as well. Always ensure strong passwords are used to protect critical assets.

Once an attacker authenticates successfully, stored procedures can be leveraged to execute further attacks against the SQL Server and the underlying operating system.

Figure 3.2 illustrates an attacker connecting to the SQL Server using the sqlcmd utility and authenticating with valid credentials. Upon successful connection, the attacker can leverage the use of the `xp_cmdshell` stored procedure to add a user account to the local system.

**FIGURE 3.2**

Adding a User to the Local Administrator Group
DBAs and attackers can utilize the *xp_cmdshell* stored procedure to interact with the operating system to perform administrative duties usually reserved for administrators of Windows itself. As seen in Figure 3.2, the attacker executes a few simple commands to add a user to the operating system hosting the SQL Server. In our target farm, the attacker has connected to an SQL 2008 Server that is running on Windows Server 2008. After connecting, the attacker issues a *net user* command to add a new user to the server’s local Security Accounts Manager (SAM) database. Once the attacker has created the new account, “t800” in our example, he then uses the *xp_cmdshell* stored procedure to execute the *net localgroup* command to add the new account to the Administrators group on the server. It does not take much imagination to think of what types of malicious activities can be performed when an attacker has access to a local account that is part of the administrators group.

**Scenario 2: Keeping Sysadmin-Level Access**

In some cases, attackers may consider adding an additional account to maintain access in the event the primary sysadmin account password is changed or the account used for access by the attacker is disabled. Shamefully, DBAs may not actually notice the additional account unless auditing for the account creation is enabled and there is monitoring and alerting for this type of activity.

While working in the field doing penetration tests, we have added an administrator-level account once we compromised a system in order to maintain access during the assessment process. At the end of the assessment, accounts are usually removed as part of the cleanup process. Prior to cleanup, this administrator-level account may have resided on the system for days or weeks, depending on the scope of the assessment, without the true administrators identifying the new account. Where are we going with this? Well, since our real-world experience shows this occurs regularly during these controlled tests, it is only natural to assume that attackers could use the same methods to insure extended access to the system.

Figure 3.3 shows our attacker connecting to the SQL Server and using the *sp_addlogin* stored procedure through the *sqlcmd* utility to create a new account named “backdoor” with a password “1337P@ss.” For the sake of clarity, we are using an account named backdoor in this example to place some emphasis what we
are doing. However, it is likely that an attacker would try to choose an account name that blends in. Naming the account “backup,” “service_account,” or “admin” are good choices because they seem like the kind of accounts that could possibly be in an administrator group. After the attacker has added the account to the SQL Server, the account is then added to the sysadmin fixed server role by invoking the \textit{sp_addsrvrolemember} stored procedure, and our backdoor account now has the same level of access the default sa account.

Figure 3.4 shows the outcome of the particular attacks perpetrated in Figure 3.3. The Server Role Properties window on our SQL Server 2008 target shows the backdoor account as one of the accounts belonging to the sysadmin fixed server role. Access is verified by connecting to the SQL Server with the \textit{sqlcmd} utility and using the \textit{xp_msver} extended stored procedure.

\textbf{FIGURE 3.4}
Backdoor Account Using Stored Procedures

\section*{Scenario 3: Attacking with SQL Injection}

This chapter has mainly focused on security issues related to the implementation and availability of stored procedures on Microsoft SQL Server. Many of the examples provided thus far have assumed that the sa or another sysadmin fixed admin role had
been previously compromised. This example describes leveraging stored procedures by using SQL injection attacks. Before we jump into how SQL injection can be used to leverage stored procedures, let’s spend a few paragraphs going over the basics of how SQL injection works.

SQL injection provides attackers a method for interacting with a Web application and its back-end database. These attacks are based on the manipulation of form fields, URLs, or cookies and posting a request to the Web server. The Web server logic evaluates the submission and returns the results based on the interpretation of the request. By modifying a legitimate request, an attacker may be able to cause unexpected results resulting in an SQL error or successful execution of the request.

Depending upon what account and context the SQL Server backend is provisioned with, an attacker may be able to perform a wide range of tasks. A classic example of an SQL injection attack consists of an attacker taking advantage of a Web site login page that contains user name and password fields as well as a submit button. Legitimate users will most likely have a user name and password that allows access to the Web site based on the permissions assigned to their account. However, an attacker can bypass authentication by entering specially crafted SQL statements into the user name and password fields.

For instance, if an attacker entered the following SQL statement into the user name field on the login form and clicked the Login button, the attacker may be able to trick the application logic into allowing access to the application even though no authentication with a legitimate account actually occurred.

\[
pwned' \text{ OR } 1=1'--
\]

The query when processed will use an SQL statement to verify the submitted credentials. In the example provided, the final query sent to the Web and SQL Server may look similar to the following statement.

\[
\text{SELECT * FROM users WHERE userID = 'pwned' OR 1=1--}
\]

This previous statement will always return “true” based on the condition that 1 is equal to 1 (the “--” is an SQL comment delimiter that tells the server to ignore code or values that follows the evaluation of 1 = 1). Since a reply of “true” usually means that the username/password combination has been authenticated, this may trick the application into believing that the user has valid credentials and allow access.

In addition, an attacker may be able to enumerate table and column names, allowing the attacker to construct a query to INSERT or DELETE records from a database table. The attacker may also be able to DROP entire tables from the database, which could cause denial of service to legitimate users. Microsoft provides some additional information on the general mechanics behind SQL injection attacks and mitigation measures on the MSDN site, and these attacks are discussed in detail in Mike Shema’s Seven Deadliest Web Application Attacks (Syngress, ISBN: 978-1-59749-543-1). Now that a quick overview of SQL injection basics has

been provided, let’s expand the topic to include how we can use SQL injection to leverage the stored procedures this chapter has been focusing on.

By slightly modifying the approach, we showed for attacking the Web application, the attacker can try to pass SQL commands that call on stored procedures. Using the following SQL stored procedure call may result in the SQL Server’s host operating system sending an Internet Control Message Protocol (ICMP) ping packet to the IP address identified in the ping command (which should be the address of the attacker’s computer).

```
'; exec master..xp_cmdshell 'ping 192.168.204.128'--
```

Access to the stored procedure would be validated by starting a packet capture using tcpdump or Wireshark on the attacker’s computer, then listening for ICMP packets to be returned from the source address of the SQL Server where the stored procedure was executed. If the SQL Server’s host operating system replies, then access to the stored procedure is verified and the attacker may move on to further attacks using stored procedures.

A similar attack involves the attacker again using the `xp_cmdshell` stored procedure, however, using the appropriate commands to add a user to the local system. This is similar to what was illustrated in our first scenario; however, this time, the attacker is executing the command from a Web form.

```
'; exec master..xp_cmdshell 'net user attacker P@ssw0rd /add'--
```

Some of these attacks have been around for quite some time and will most likely be relevant for years to come. It is important to remember that applications that interact with SQL Server should be closely scrutinized and follow best practices for ensuring applications as secure as possible before deployment.

**THE FUTURE OF STORED PROCEDURE ATTACKS**

The good news about SQL Server is that Microsoft has started taking steps to reduce the attack surface of the default installation and has turned its focus onto ensuring a secure development environment that should limit the amount, impact, and scope of vulnerabilities in the future. The bad news is that this really doesn’t have anything to do with disallowing the abuse of code or leveraging SQL’s authority to escalate your privilege beyond the application itself.

Microsoft may cut off the attack vectors shown or even remove the particular pieces of code that were presented as valid attacks, but others will certainly take their place since this powerful flexibility is one of the core features of the product and the Microsoft philosophy. Even if you could somehow convince Microsoft to remove the raw convenience of stored procedures (or whatever they might choose to rename it for marketing reasons), Microsoft would still have to deal with the heavy bondage that is “backwards compatibility.”

SQL Server’s success and use in the field today is really based on what DBAs and developers have created on top of the SQL Server database application itself. This means that Microsoft must keep in mind that major changes to the functionality of
the product will have a severe impact on the applications that run on it. As we stated in the section “How Stored Procedure Attacks Work,” stored procedures are not available for attackers to utilize right out of the box. That statement, however, is only referring to the newer versions of SQL Server.

In versions of SQL Server before SQL Server 2005, the stored procedures we are concerned with were installed by default; therefore, Microsoft developers must assume that somebody actually utilizes these stored procedures as part of the applications they have created. Microsoft was willing to pull these procedures from the default install, but that doesn’t mean that they are willing to permanently break applications developed on top of SQL Server.

For this reason, the features that drive the sales of SQL Server are those that serve to make development of applications that run on the platform as easy as possible. Since stored procedures are one of those features, and they need to continue to be available for reasons of backwards compatibility, don’t expect these attacks to change very much in the near future.

DEFENSES AGAINST STORED PROCEDURE ATTACKS

From a defensive point of view, we consider stored procedure attacks to be a “second layer” attack because it requires that you have already penetrated the first layer and gained a level of authority prior to being able to execute. When developing a defensive plan to protect against a second-layer attacks, the general rules are as follows:

1. Try to protect the second layer by ensuring that the second layer is secure.
2. Eliminate the vulnerabilities that are exploited by typical second-layer attacks.
3. Limit the attack surface as much as possible.
4. Log/monitor for attacks and have an active and effective alert system.
5. Do your best to limit the impact and effectiveness of the attacks.

This approach is an important part of a defense-in-depth strategy. The concept of defense-in-depth was covered earlier in this book in Chapter 1, “Windows Operating System – Password Attacks.” The goal is to make it as difficult as possible (or hope-fully impossible) for an attacker to execute the attacks we have demonstrated.

Stored procedures provide a good example of this idea. In the following sections, you will see multiple strategies that fall into the same defensive layer, but you will not see any that would fall into the second defensive layer (eliminating the second-layer vulnerabilities). Part of the reason that the stored procedures attacks are the subject of this chapter is that it is not possible to completely eliminate the vulnerability.

First Defensive Layer: Eliminating First-Layer Attacks

Executing many of the stored procedure attacks we have explained requires that you already have sysadmin-level access within SQL Server application. It is obvious that you can gain this level of authority by directly defeating SQL Server’s security, but it
can also come from attacking and defeating Windows itself. The key to eliminating first-layer attacks, therefore, is actually just following good security practices in regards to both Windows and SQL.

**NOTE**

Prior to SQL Server 2008, administrator-level access within SQL Server was automatically provided to the local administrators group of the Server (if it was set for either mixed mode or Windows authentication). This means that if you gain local administrator membership on a machine running SQL Server 2005 or earlier, the database is automatically yours. Of course, this does not mean that you won't be able to easily find a way to gain sysadmin-level access in SQL Server 2008 if an attacker “Owns” the machine, especially because the built-in administrator account is still provided this authority, but it is no longer automatic.

The subject of securing either your Windows operating system or your SQL Server implementation is covered in many other books that are much larger than this one, so we obviously cannot go into all of the details surrounding how to do this. We can, however, hit some of the high points related to blunting general attacks.

**Implement the Strongest Authentication Possible**

We feel that this is so important that “password attacks” is the subject we chose as the most dangerous attack against Windows itself as covered in Chapter 1, “Windows Operating System – Password Attacks.” Weak passwords on default accounts are often one of the things both attackers and penetration testers go after, and it is scary how many times this works, even in environments that are supposedly “high security.” The need for strong authentication is important regardless of the account type or authorization level, but it is doubly important when you are looking at privileged accounts that have administrative rights within an application.

Attackers have many tools at their disposal today that allow the automation of dictionary and brute force password attacks against Microsoft SQL servers. The tools are used by attackers and penetration testers and are usually easy to configure. Some currently available applications are listed in Table 3.1; however, these are just an example as there are many other similar tools.

SQL Server has built-in integration with Windows security and you should use this whenever you can. This is especially true when the Windows server is part of a domain and account credentials and passwords are stored in active directory (AD) rather than the local machine. Regardless of where the credentials are stored

<table>
<thead>
<tr>
<th>Table 3.1 SQL Server password attack tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hydra</td>
</tr>
<tr>
<td>SQLBrute</td>
</tr>
<tr>
<td>SQLPing</td>
</tr>
<tr>
<td>Cain and Abel</td>
</tr>
<tr>
<td>Metasploit framework</td>
</tr>
</tbody>
</table>
implementing strong password policies such as minimum lengths, complexity, and lockout periods is critical to limiting the effectiveness of password based attacks. As long as your SQL Server is running on Windows Server 2003 or later, you should also always select the Enforce password policy option within SQL Server. This option automatically enforces all of the same password policies of the computer against the SQL logins, which includes the sa account.

Even better than having strong passwords is requiring two-factor authentication mechanisms for all privileged accounts. Windows natively supports mechanisms such as biometric scanners, smart cards, and tokens. Since Windows supports these, you can easily use them for all of your integrated accounts. In addition, SQL Server 2008 running on a Windows 2008 platform fully supports two-factor for biometric and smartcard certificates. Although two-factor systems can have their own problems and vulnerabilities, generally they are more secure than even a 100-character password with upper- and lowercase letters, numbers, and symbols.

Implementing End-Point Security Mechanisms

Although some end-point security solutions (such as an antivirus solution) are given in today’s world, many security administrators think of end-point security solutions only in terms of their workstations rather than their servers. In our opinion, this is a mistake. Relying on network systems such as firewalls and intrusion detection system/intrusion protection system (IDS/IPS) to protect the server infrastructure provides an attacker with only one system to defeat. In addition to network systems only providing perimeter security, most of the administrators of these systems have concerns regarding performance that will preclude you from being able to define detailed access control lists (ACLs) and policies for every server.

By adding desktop firewalls and host-based IDS/IPS to the server running SQL Server, you are able to prevent certain actions, or alert someone to these actions, based on different kinds of activities that occur on the server during normal operating conditions. Using and tuning these solutions properly can even make it so that certain actions can only be performed from a management subnet or the internal network. This may not stop every attacker, but it would definitely slow one down (at least one who isn’t using an internal zombie that they already own).

In addition to traditional IDS and IPS implementations, administrators may choose to deploy IDS/IPS systems that detect and alert administrators of attacks against SQL server instances. This will provide advanced knowledge of pending attacks and other suspicious activities to network administrators and security personnel.

Employ an Efficient and Well-Defined Patching Process

Some things seem obvious to an attacker, if Code Red will still work against an SQL Server (and it is scary that even today we still see this in the wild) then the “administrator” is more likely to be Bob from accounting than an IT security professional. The unfortunate reality is that no developer can anticipate every possible attack and no software company can afford to make their application 100% bulletproof before
they release it; therefore, it is vitally important to ensure that vendor patches for the operating system and applications running on a system are applied as quickly as possible. This may seem pretty straightforward; just have Windows and all of the applications automatically download and install patches from the vendor as soon as they are released. Sadly, things don’t get to work this easily in the real world.

In a working environment, SQL Server is generally a part of the backbone of some business processes and therefore concerns about issues such as performance and downtime are valid. With this in mind, most updates and/or patching must actually occur during regularly scheduled support windows rather than when the update or patch is first released. The design of the patching process must understand this and balance the criticality of the patch with the risk of downtime for this server. This balancing act can mean that the most critical servers are actually the servers that get patched the least, and this should not be acceptable from a security perspective.

One way to combat this situation is to define a solid business and technical process related to patching your SQL Servers. This core process should start with defining categories for the criticality and priority of each update or patch (the number of categories an organization defines is up to them). The process should then evaluate the criticality of each system and define timelines and procedures for each of the categories previously defined. Once these guidelines are in place, each update or patch should be evaluated when it is released from the vendor and immediately assigned to a category. Once the patching category is defined, the process and timeline for the implementation of the update or patch on each system should already be defined. You must actively monitor the criticality of an update or patch until it is fully implemented and you should reevaluate the category it is assigned to, if the situation changes. For example, if a vulnerability is found in Windows and a patch is released on a Tuesday, but there is no exploit code in the wild, then you may assign the patch into your “standard priority” category. Everything sounds good at this point, but let’s say that on Wednesday someone releases an exploit for the vulnerability the patch addresses.

From a security perspective, the vulnerability has now gone from a theoretical to an active risk and you must be able to act accordingly. This change in circumstances doesn’t automatically mean that you have to change the designation from “standard priority” to “critical priority,” nor would a change in category necessarily mean that you would apply the patch to your SQL Server any sooner. The crucial element here is that your process must allow you to actively reevaluate the criticality of the patch based on the change in circumstances and act according to the new evaluation.

**Second Defensive Layer: Reduce the First-Layer Attack Surface**

Reducing the attack surface for Windows means following the basic Windows security precautions that you will find in any security best practice guide. Eliminate or disable all unnecessary applications, services, and network protocols (Minesweeper is not a necessary application on an SQL Server no matter how bored you get waiting for a data-mining query to complete). Rename, disable, and/or delete unnecessary accounts (including the built-in administrator account once you have created an
alternative account and assigned it administrator group membership). Limit the user rights, privileges, and group membership of accounts to only what they need to perform the function they are designed for.

From an SQL Server perspective, reducing the first-layer attack surface means removing any unnecessary accounts from the sysadmin server role and locking down the sa account. Assuming you chose Windows authentication mode during setup (or have switched over to that mode since then), your first step is to create a local account with a strong password within Windows and then add that account to the sysadmin role within the SQL Server security. Once this is done, you would need to log in to Windows as that account and delete the local administrator account or group (depending upon the version of SQL Server you are using) from the sysadmin role.

Locking down the sa account is also a multistep process, you need to start by setting an extremely strong password then disabling the account. If you are running SQL 2005 Server or higher, then you should also rename the sa account to something unique.

```
ALTER LOGIN sa DISABLE;
ALTER LOGIN sa WITH NAME = [ZeroCool];
```

The “ALTER LOGIN” statements shown above will first disable the “sa” account and then rename it to “ZeroCool.”

**Leverage Microsoft Knowledge**

Microsoft deserves a lot of credit for providing in-depth technical documentation, tools, and recommendations at no charge to allow you to tighten up the security to the level you want. Microsoft’s “Threats and Countermeasures” guide for Windows 2008 lists every security item that can be managed by group policy and includes information about the vulnerability, countermeasures, and potential impact of each particular setting. There are other earlier guides available, but each guide is completely backwards compatible and includes information about what versions each setting is applicable to, so there is no reason not to download the newest one.

In addition to the “Threats and Countermeasures” guides, both Windows 2003 and 2008 have Security Compliance Management Toolkits that include preconfigured security baselines that you can apply to a Windows server utilizing the tools provided in the toolkit. Besides the tools for implementing preconfigured security baselines, each toolkit includes a security guide and some settings guides that explain what each baseline does and its impacts, as well as links to much more documentation on that particular subject.

Beyond these particular items, Microsoft actually provides its entire knowledge base to the public (the only difference between what is available to you online and Microsoft support personnel is some extra tagging) along with an incredible amount of information about the inner workings of the operating system and SQL Server on the msdn.microsoft.com site. They also have resources dedicated to basic SQL Server security and many of the basic security provisions of Windows (eliminating unnecessary accounts from the SQL Server application database, like the built-in administrator, and having strong authentication policies) also apply to securing the application.
Finally, many security organizations, books, and magazines provide publicly available recommendations to help you secure both your Windows and SQL Servers. The point we are trying to get across here is that you should actively leverage all of this information to determine the best way to secure both the Windows and SQL Server against the initial compromise that will provide an attacker with sysadmin-level authority and thus the ability to use stored procedure attacks.

**Third Defensive Layer: Reducing Second-Layer Attacks**

Unless there is a specific reason that you need a stored procedure (especially all of the “xp_” procedures), these should all be completely removed from the server. If there is some circumstance where you do need these procedures, but don’t need them to always be active, then you should disable the procedures (if they are not already disabled by default).

**Fourth Defensive Layer: Logging, Monitoring, and Alerting**

Throughout this chapter, we have shown many different ways that SQL, and by extension Windows, security can be compromised by different attacks. Stopping these attacks is an ongoing battle that unfortunately will never end, but the best way to mitigate the impact of these attacks is by responding as effectively as possible. The crucial element involved in responding to any attack is to first recognize that something is going on.

The purpose of all of the stored procedure attacks described in the section “Dangers Associated with a Stored Procedure Attack” is to actually create accounts and gain membership in groups that provide privileged access to either SQL Server or the Windows operating system. In both cases, audits can be defined that will capture information about these events when they occur, and these will be stored in either the SQL Server or Windows event logs. Once the events are created, they can be actively monitored by a Microsoft solution such as System Center Operations Manager (SCOM) or a third-party service management system such as Tivoli, or moved across the network to specialized logging servers among many other choices for a monitoring infrastructure. Once this infrastructure is created, any solution you utilize should be configured to send alerts to administrators if different events set off the triggers you define and they should have policies and procedures surrounding the investigation of the alert and responses.

**Identifying Vital Attack Events**

The problem with auditing is that so much information gets put into event logs that it is difficult to sort out what is significant and what isn’t. This gets even more difficult if you are trying to set up alerting policies because although you need certain information, too many false-positives means that the alerts will actually get ignored by your security personnel. If you understand the way an attack is perpetrated,
however, you should be able to identify either a single vital element, or a series of vital elements, that must occur as part of the attack. By identifying these elements, you can do some security testing with the attack and understand what information will only be entered into an event log when this vital attack element occurs.

**EPIC FAIL**

In 2008, Countrywide Home Loans reported the loss of over 2.4 million customer records including social security and mortgage loan information. The insider who performed the attacks confessed to downloading approximately 20,000 files per week over a 2-year period and selling them for a total of approximately $70,000.

Implementing controls to audit data access may be able to detect large queries and provide early warning about potential data loss. Insider threats are just as dangerous as external threats, in many cases, more dangerous due to the access already provided to employees.

If you have followed the recommendation to ensure that the `xp_cmdshell` stored procedure is disabled, you have set yourself to catch the vital element of the deadly attacks we have described in this chapter because they all require this single action. When we used the `sp_configure` command to enable the `xp_cmdshell` stored procedure in the section “Accessing Stored Procedures” (Figure 3.1 shows this action), SQL 2008 actually logged the event shown in Figure 3.5 (this type of event is logged

![FIGURE 3.5](image)

Stored Procedures Enabled Event Message

---

within SQL 2008 by default). This event provides a message that partially states, “Configuration option ‘xp_cmdshell’ changed from 0 to 1.” Because this message is so specific to this particular event, it makes it simple to set up an alert to security personnel if an attacker actually has enabled this stored procedure in order to try to carry out the stored procedures attacks discussed in each of the scenarios presented in the section “Dangers Associated with a Stored Procedure Attack.”

In this case, we got lucky because logging for this type of event was enabled by default in our test environment, and the message was so specific to the action we were protecting against that all we have to do is define the alert in whatever service we are using to actively monitor the logs. In most cases, making sure that an event is generated in your logs that is specific to your vital attack element and is precise enough to only occur in conjunction with that element may take some work; however, the added level of security you get from taking the time to do this is well worth the effort.

**Fifth Defensive Layer: Limiting the Impacts of Attacks**

The approach here is to look at what barriers you can put in place to stop an attacker from escalating their privilege at each point of a successful attack. One area to look at is limiting the access of the service accounts that SQL utilizes. Where possible, you should use named accounts rather than system, and these should be created as local service accounts rather than normal user accounts. If you take a look at Figure 3.4 from Scenario 2, “Keeping Sysadmin-Level Access,” you will see that in SQL 2008 these security precautions are there by default. However, that is not the case in all earlier versions.

In addition, you need to run SQL Server as its own server rather than sharing it with other applications. If this is an issue because of limited server resources within your environment, then you should utilize virtualization to separate the applications as different server instances running on the same physical device. Finally, you should never allow SQL to run on the same server as a domain controller. This is probably self-evident to you, but think about a backoffice server that may run SQL, Exchange, and a Domain Controller on the same server. Although this may seem like a more efficient use of resources, the impact of any of the successful stored procedure attacks we have shown here means that the attacker now owns your domain.

**SUMMARY**

As part of the SQL Server code base, Microsoft has provided a way for prebuilt pieces of code to be stored within SQL Server itself and leveraged over and over again by DBAs and developers to perform many functions through a simple call to these procedures. Many of the functions that come with SQL Server from Microsoft are procedures that are meant to provide hooks into many of the administrative tasks that DBAs have to perform, but that also makes them prime targets for attacks.
Microsoft has recognized this vulnerability and deploys its newest versions of SQL Server with these procedures disabled by default; however, they also provide very simple ways to enable them.

This chapter was able to explain how Microsoft SQL Server utilizes stored procedures and the purpose of each of the default system stored procedures. It should also have given you an understanding of how attackers can utilize these stored procedures and how dangerous they can be. Finally, you should now be able to grasp how to build the strongest possible defenses against SQL stored procedures attacks.

---

**Endnotes**

1. [http://go.microsoft.com/fwlink/?LinkId=148532](http://go.microsoft.com/fwlink/?LinkId=148532)