

Security Auditing In Microsoft SQL Server

White Paper

By Aaron C. Newman, CTO & Founder

**APPLICATION
SECURITY, INC.**

www.appsecinc.com

Tel: 1-866-9APPSEC

E-mail: info@appsecinc.com

Introduction.....	3
The Ideal Monitoring Solution	4
Watching the Database Administrator	4
Watching Temporary Accounts.....	4
Auditing Access to Sensitive Data.....	5
The Flexibility to Filter Results.....	5
Attempts to Circumvent an Application	5
Auditing Exceptions.....	6
Manipulating Web Applications	7
Identifying Unusual Activity	8
Known Attacks.....	8
Native Database Auditing.....	9
Connection Auditing.....	9
C2 Auditing.....	10
SQL Trace.....	10
Summary of Shortcomings	11
AppRadar – The Leading Solution.....	11
Auditing.....	12
Attack Signatures.....	13
Buffer overflows	13
Web Application Attacks.....	13
Privilege Escalation.....	14
Accessing OS Resources.....	14
Password attacks	14
System Events	14
Conclusion	15
About Application Security, Inc.	15

INTRODUCTION

The database community has begun to realize its role in a robust information security infrastructure. Traditionally there has been little concern for the security of relational database systems, and for the vast amounts of data they house. In the early days of relational databases, gaining access to a database was so difficult that the need for complex security features was irrelevant. Databases were housed in mainframes not accessible directly from the network. Slowly they were ported to other networked systems such as UNIX, Linux, and Windows, but even when this happened, the databases were kept far behind the firewall out of the reach of the typical Black Hat surfing the Internet.

As digital information has become more and more critical to how businesses and customers interact, firewalls, which once served as the walls of the fortress, have now become an archaic means of defense providing little real-world protection. Even the smallest of organizations have numerous entry points into their networks - wireless access, remote employees connecting through VPN, customers connecting through web applications, etc.. As well, employees take home laptops, hook up to a home cable modem connection, get infected with a Trojan, and then return to work. This becomes a virtual “store and forward” attack that effectively gives the Black Hat a backdoor into an organization’s internal network.

As the threats have changed, the responses must change as well. Rather than focusing protection solely on perimeter security, it is imperative we start looking at protecting data at the source – inside the database. The goal is to provide protection where it is most effective and cannot be circumvented. The database has come to be the de facto standard for housing an organization’s most valuable information. The logical extension of these facts is that the database is where your strongest security needs to be in place.

Given the flurry of new information security related legislature that has arisen, there is a strong call to take extra steps to provide strong security at the database level. The need to protect sensitive information no longer needs to be driven by corporate strategy, federal laws are forcing organizations to put the proper defenses in place, with consequences for those who do not comply. Information security is focused on preserving the confidentiality, integrity, and availability (CIA) of the information system. Without maintaining these basic tenets, a database does not measure up to the requirements of handling commercial data.

Without real-time auditing and monitoring of data, CIA is impossible to maintain. While there have been many discussions on the need to provide some level of auditing and monitoring, there is little information to help organizations define what the appropriate auditing and monitoring strategy is for them. This paper is based on “theoretical best-practices” combined with “real-world practicality” to define a usable policy for auditing and monitoring databases. By following the policies we outline in this paper, you can properly implement a database system that will work well, and provide adequate security for the data it houses.

THE IDEAL MONITORING SOLUTION

Monitoring can be a complex task. Collecting data is the simple part of the job. An ideal solution needs to handle the tricky aspects of the job, as well as the simple ones. Some of the more difficult issues that need to be addressed by a monitoring solution include:

- 1) Deciding what to monitor for
- 2) Handling the volume of data that needs to be monitored
- 3) Detecting when something malicious has occurred
- 4) Ensuring the integrity of the audit data

WATCHING THE DATABASE ADMINISTRATOR

Modern database technology is designed to be managed by database administrators, who are the unrestricted owners of the database. This architecture leaves an organization's most critical information entirely exposed to and controlled by a small handful of technologists, the Database Administrators or DBAs. This leaves both the DBA and the entire organization in a precarious position. The DBA is afraid that she or he will be blamed for any information leak, while the organization is forced to almost blindly trust a small number of people in the technology group.

One way to mitigate this risk is by properly auditing and monitoring the activities of the DBA. The amount of work that a DBA does on a production server is very limited. Auditing and monitoring this activity does not add significant overhead to the system.

How do we properly audit activity in a database? Native auditing fails here because it is fully under the control of the DBA. He or she can easily turn off auditing, clear the audit logs, manipulate an audit record, or even reconfigure auditing to filter out their own malicious activity. Auditing should ultimately provide a separation of duties. An ideal audit system would be intelligent enough to distinguish database administrative accounts, filter out noise and irrelevant events, and succinctly illustrate their activities. Auditing data should be written to a secure location, where even an administrator of the database would not have direct control over the recorded activity.

WATCHING TEMPORARY ACCOUNTS

Another type of activity that requires monitoring is the use of temporary and special accounts. Many companies have procedures through which the database administrator can request a temporary account for others or for themselves to manage databases if required. For instance, the DBA will request the operations team to create a temporary account for which to logon and manage the database when the database goes down or when backups need to be recovered. This account will be set to expire in several hours after which the account will be deleted.

This is a reasonable system for reducing exposure to a malicious database administrator. However, a window of exposure remains, in that it is difficult to track exactly what that administrator does while using the temporary account. An ideal monitoring and auditing system can provide real value in this situation. A system that can track the activity of the DBA's temporary account can be used to ensure that nothing malicious has occurred.

AUDITING ACCESS TO SENSITIVE DATA

An auditing system also needs to be able to monitor access to sensitive data in a subset of tables. A typical database contains massive amounts of data, some of which is not sensitive at all. Other pieces of data however could lead to disaster, if they fell into the wrong hands. Auditing all database activity is impractical, and can lead to information overload. For instance, suppose you have a lookup table to map products to product IDs. This table will likely be accessed regularly, possibly thousands of times each day. Tracking access to that lookup table won't catch any malicious activity; in fact the amount of noise this tracking would generate may very well bury a real attack in a sea of essentially meaningless data. While some tables are not sensitive, others probably are. Those that contain credit card numbers, payroll information, or social security numbers should be monitored, and the audit trail should be regularly reviewed.

In order to facilitate this type of auditing, database administrators or applications owner must decide what data is sensitive and define that in the auditing system. The auditing system should be able to accept and configure the list of databases, tables, objects, and columns that can be monitored, as well as being able to configure which actions on various objects to monitor. For instance, if you have static data that is public information, you will likely have no need to audit who SELECTs from the table. At the same time, you may have a strong need to record who modifies the data. In that case you need the ability to audit any UPDATE, DELETE, or INSERT to the TABLE from any user.

THE FLEXIBILITY TO FILTER RESULTS

It is important to have the ability to filter out how data is audited based on who is accessing the data. For instance, there is a requirement within the HIPAA regulations stating that access to patient records be strongly accounted for. If an administrator in the system accessed patient Jane Smith's medical records on June 15th, there should be a record of the action to provide accountability for that data. On the other hand if the patient's doctor accesses the data twenty times in a day, there's little value in recording that activity multiple times. Auditing needs to be able to record unauthorized users attempting to access data, yet also needs to be flexible enough to keep the level of noise to a minimum. This can be accomplished by minimizing the recording of activity performed by authorized personnel. An ideal auditing solution should have the capability to perform granular filtering based on factors such as account name, source of activity, and the time of the activity.

ATTEMPTS TO CIRCUMVENT AN APPLICATION

Another common issue that needs to be monitored is users circumventing an application, and connecting directly to the database. This can be a problem if you are using two-tiered architectures, such as a Visual Basic executable connected directly to a database, or if you are using a three-tiered architecture, such as a web application connected to the backend database via an application server. Either way, there is a need to monitor for people using utilities such as Microsoft Access, Microsoft Excel, or even Query Analyzer to connect directly to the database. Someone doing this may simply be trying to make their job easier but are in effect opening up a security hole in the database. On many occasions, users have been discovered to have placed a linked Microsoft Excel spreadsheet on an open file share to allow other users to see the results of their work in the database. Unfortunately this linked spreadsheet can be manipulated to pull back information from the database other than the data for which it was intended.

There are two ways to audit for this problem. One way is to watch specifically for people using tools such as Microsoft Office to access the database. To do this effectively, you need to be able to list the most common applications that a curious, or well-meaning user might employ. Another strategy is to audit any connections that do not come from the expected application. For instance, let's consider a database where users connect via an application called HRPayrollApp. An ideal auditing solution should be configurable to alert you when someone connects to the database using any application other than HRPayrollApp.

AUDITING EXCEPTIONS

An audit system needs to provide the ability to easily “approve” traffic in order to prevent valid activity from continuing to raise alerts. For instance, an application may legitimately access data in the database that is being tracked by the auditing system. That’s good to know when you first install and setup the auditing system. It however becomes “noise” after you see that data a few hundred times. By noise, we mean that you get no value out of seeing the alerts, so they contribute to drowning out other more valuable audit data. If we get 10,000 audit records it’s going to be hard to see the one record that really matters because it’s buried in an information overload. This is why it is so important for an effective audit system to be able to reduce the number of items audited, and to only catch the activities that we care about. This is accomplished by allowing “exceptions” to be created. For instance, an exception would say - do not record access to data when a specific SQL statement comes from user XYZ from machine ABC. When any other access to the data occurs, the audit system should record the activity. A proper auditing system should also have the ability to record any activity that does not match specific criteria. For instance, the system should allow you to say record all activity except for SQL statements from application XYZ.

MANIPULATING WEB APPLICATIONS

Monitoring database activity originating from a web application is very important. An ideal audit solution must be able to detect when someone manipulates a web application. How can this be accomplished? To start with, the monitoring system would have to learn how the web application tends to use the database. This involves compiling a complete list of each SQL statement that is executed by the web application. Then, when a query is executed, it should be compared to the list of “known” SQL statements. If any SQL statement has been modified, for example by injecting additional SQL statements through the web application, the resultant SQL statement will not appear on the “known” list, causing the monitoring system to alert on the unauthorized activity.

How would a system go about gathering a complete list of SQL statements generated by the web application? Three methods are enumerated below:

Method #1: Running the monitoring system in a “learning mode” during regular operations

Method #2: Manually feed the application a list of the valid SQL statements

Method #3: Use an automated tool to enumerate the list of SQL statements

Recording the list of valid SQL statements requires a certain amount of intelligence. You cannot simply record all the SQL statements and then use that as the list because these SQL statements are parameterized. For instance, you may have a SQL statement to look for all the orders for a customer. This would generate the following SQL statement:

```
SELECT * FROM ORDERS WHERE CUSTOMER_ID = 5
```

Searching on a different customer would yield:

```
SELECT * FROM ORDERS WHERE CUSTOMER_ID = 6
```

Are these 1 or 2 different SQL statements? For the purposes of watching for a web application being manipulated these are the same statement and the second example should not generate an alert. The monitoring solution should be able to normalize these SQL statements into representations such as the following:

```
SELECT * FROM ORDERS WHERE CUSTOMER_ID = ?
```

Then as each statement is reduced to its normalized form, it can be compared and effectively evaluated as the same SQL statement.

On the other hand, the following SQL statement should not be evaluated as the same as those above, because the normalized form is different:

```
SELECT * FROM ORDER WHERE CUSTOMER_ID = 10 UNION SELECT USER, PASSWORD FROM master.dbo.syslogins
```

IDENTIFYING UNUSUAL ACTIVITY

Another important aspect of a monitoring system is the ability to identify activity that is not typically seen. This is meant to locate and isolate activity that is unusual and may possibly be in violation of corporate policy or may be a previously unknown attack. An ideal tool would be able to classify activity into patterns, and based on those patterns be able to identify normal traffic. This is a critical component in the ability to determine that an unauthorized or unexpected activity has taken place.

A good example demonstrating the need for this feature involves mapping typical administrator activity. If a monitoring system can detect when an administrator does something unusual, it can help ferret out wrongdoings. The auditing solution must have the ability to catch an administrator breaking corporate policy by remotely administering the database from home, or perhaps logging into the network late at night from a remote office, which could indicate an attack from an internal employee.

KNOWN ATTACKS

It is imperative that your database monitoring system can detect and recognize known attacks. Attacks can come in many forms. When one occurs, the system should be able to notify someone right away. Below is a sampling of the broad categories of attacks that a monitoring system should be able to pick up on:

- 1) Buffer overflows being executed from Transact SQL
- 2) Web application attacks
- 3) Privilege escalations
- 4) Accessing OS resources
- 5) Password attacks
- 6) Pen Testing or hacker tools used against the database
- 7) Database starting and stopping

Within each category are a multitude of specific attacks to look for.

NATIVE DATABASE AUDITING

Most database vendors provide some form of native auditing. This is typically implemented as data written to tables, log files, or even the Event Viewer on Windows. As a major player in the database market, Microsoft has provided native auditing functionality in their SQL Server product. There are actually several ways that you can record activity in Microsoft SQL Server. Each method has its strengths and weaknesses. We explore each option below:

CONNECTION AUDITING

MS SQL Server's first form of auditing is a subsystem that can be used to record failed and successful login attempts on the server. Recording connection attempts is useful in being able to discover:

- 1) Who is attempting to connect to the database
- 2) When an attack is taking place
- 3) If an attack was successful

This form of auditing has little negative side effects, as the amount of activity being audited is minimal. Auditing connection attempts does not result in a significant performance impact on the database, and rarely creates an excessive amount of data written to the log. Because of the importance of knowing what logins are connecting to the database, coupled with the minimal impact recording a login causes, it is rarely a bad decision to audit connection attempts with this method.

The possible settings for login auditing are:

- 1) None - logs no auditing information
- 2) Success - causes only successful logins to be logged
- 3) Failure - causes only failed logins to be logged
- 4) All - causes successful and failed logins to be logged

The auditing information is written to the SQL Server error logs and to the Windows event log. To enable auditing of logins, perform the following actions:

- 1) Open Enterprise Manager and connect to the database.
- 2) Click the right-mouse button on the instance and select Properties from the popup menu.
- 3) Open the 'Security' tab.
- 4) Under 'Audit Level' choose 'All'.
- 5) Click the OK button

Unfortunately, this is only a small subset of the audit information that we need to effectively monitor a database. There is also no facility that provides the intelligence to alert us when something bad is happening in real-time.

C2 AUDITING

Microsoft SQL Server also provides an option to audit successful and failed attempts to access statements and objects. This option was designed to meet the standards of the C2 Security Evaluation criteria. This feature provides a valuable means of monitoring for system misuse. By default, it is not enabled.

When C2 auditing is enabled, SQL Server will track C2 audit events and record them to a file in the `\mssql\data` directory. A drawback of this approach is that the C2 auditing feature will cause the database to stop when the server is unable to write to the audit file for any reason. This is in accordance with the requirements for C2 security. Care should be taken when enabling C2 auditing since excessive use of audit counters could have a significant performance impact on the server.

Before attempting to set the 'c2 audit mode' configuration option, you must enable the 'show advanced options' configuration option. This is performed using the following command:

```
USE master
EXEC sp_configure 'show advanced option', '1'
RECONFIGURE
```

To enable the feature, set 'c2 audit mode' to 1 using the following command:

```
sp_configure 'c2 audit mode', 1
```

After setting the value, you must stop and restart the server.

C2 auditing is generally considered ineffective because the amount of information being record is simply overwhelming. Enabling this feature can result in running out of disk space quickly and can harm performance on an already taxed system. Again, this method provides no means to alert someone when something bad is happening.

SQL TRACE

The last method Microsoft offers for tracking database activity is through a SQL trace. This is an interface made available through extended stored procedures. It is used extensively to identify poorly running SQL statements, and to debug other performance problems. Ultimately, events can be collected and viewed through an application called SQL Profiler.

There are a number of issues related to relying on SQL Trace to monitor your database. The biggest shortcoming is the impact on performance. These native interfaces are difficult to configure in a way that performance does not suffer significantly. Even Microsoft discourages the use of SQL Trace on a production system because when it is enabled, it can consume memory, CPU cycles, and disk space. While interfaces such as SQL Trace can be tuned for negligible performance impact, out of the box they are not designed to audit or monitor systems on an on-going basis.

NATIVE AUDITING SUMMARY

The ultimate issue with native auditing is that there is no built-in intelligence that allows the tools to make smart decisions, or to fire off alerts. With C2 auditing, tuning or filtering criteria do not exist. There is just an on/off switch. No adjustments can be made to who or what is being audited. SQL Trace is slightly better, however there is still no intelligence built into the auditing. It takes a significant amount of effort to turn on all the right switches and turn off all the wrong ones. There is also no logic built-in to detect and highlight malicious activity. SQL Trace is great at amassing a huge amount of data, but is useless in finding the “needle in the haystack” that is the evidence of malicious activity.

All of the native audit systems rely on storing their output in a local file or table. This is inherently insecure, since it means the audit trail is not well protected. The data is stored locally, where the very people it is attempting to audit can access it. If an attacker breaks into a system, the first action he or she will take is to clear or truncate any audit trail, or even delete individual records to hide their footprints. The same will happen with a database administrator that wants to perform actions they are not authorized to perform. They will simply remove any record of the malicious activity from the local audit logs.

APPRADAR – THE LEADING SOLUTION

AppRadar is a patent-pending revolutionary new monitoring and auditing solution for protecting data at the source – in the database. It is complementary to the many monitoring and intrusion detection tools available for the network, as AppRadar is focused squarely on database applications. Intrusion Detection engines such as Snort and Internet Security System’s RealSecure™ are very useful at detecting attacks from the perimeter, but do nothing to identify attacks within the database. These engines are not designed to see a malicious user connecting to the database executing SQL attacks, SQL Injection, or even an unauthorized user trying to access password hashes. Traditional Intrusion Detection Systems focus on perimeter security, while AppRadar is designed to detect misuse (whether from authorized or unauthorized users) of your most valuable assets – within the database.

AppRadar is a highly flexible solution designed to work effectively out-of-the-box with the ability to be customized to handle the unique constraints of any environment. AppRadar is a hybrid system, providing a combination of rule-based signatures and anomaly detection. By combining the two into a holistic approach, the highest levels of protection can be achieved.

AUDITING

AppRadar provides the ability to audit database activity. This includes monitoring for changes to objects, system tables, permissions, configuration systems, and many others options. AppRadar can be configured to audit specific objects or entire object classes. This can be customized to suit your environment by determining what data is sensitive and then specifically setting AppRadar to monitor those objects.

Below are some of the audit events AppRadar can monitor for:

Accessing list of logins	CREATE DEFAULT	Login added - Windows
Adhoc queries enabled	CREATE FUNCTION	Login attempt - failed
Adhoc updates to system catalogs	CREATE INDEX	Login attempt - successful
Allow direct updates to system tables	CREATE PROCEDURE	Login created - standard
Allow remote access to server	CREATE RULE	Login denied - Windows
ALTER FUNCTION	CREATE TABLE	Login dropped - standard
ALTER PROCEDURE	CREATE TRIGGER	Login revoked - Windows
ALTER TABLE	CREATE VIEW	Object permission denied
ALTER TRIGGER	Cross DB ownership chaining	Object permission granted
ALTER VIEW	Database backup	Object permission revoked
Application role enabled	Database restore	Server role - login added
Auditing disabled	Database role - created	Server role - login dropped
Authentication mode changed	Database role - dropped	SQL statements
C2 audit disabled	Database role - user added	Statement permission denied
Connect from ISQL Utility	Database role - user dropped	Statement permission granted
Connect from MS Access 2000	Database user created	Statement permission revoked
Connect from Office 2000 Component	Database user revoked	Stored procedure executed
Connect from Office XP Component	DBCC command	System table - DELETE
Connect from OSQL Utility	Direct access to system tables	System table - INSERT
Connect from SQL Profiler	DROP DATABASE	System table - SELECT
Connect from SQL Query Analyzer	DROP DEFAULT	System table - UPDATE
Connect from SQL Server Enterprise Manager	DROP FUNCTION	Trace started
Connect from TOAD for SQL Server	DROP INDEX	Trace stopped
CREATE DATABASE	DROP PROCEDURE	User defined function executed
	DROP RULE	User table - DELETE
	DROP TABLE	User table - INSERT
	DROP TRIGGER	User table - SELECT
	DROP VIEW	User table - UPDATE
	Extended stored procedure executed	
	Failed object access	

AppRadar also provides the ability to monitor the activity of specific accounts. For instance, you can configure AppRadar to monitor the activities of the SA or other DBA logins. AppRadar can also be configured to capture the activity of all users other than the Web Application.

ATTACK SIGNATURES

AppRadar provides the industries most comprehensive library of database attack signatures. AppRadar covers the following attacks out of the box:

BUFFER OVERFLOWS

AppRadar monitors for attacks that take advantage of buffer overflow vulnerabilities. Numerous extended stored procedures in Microsoft SQL Server 2000 contain buffer overflows which result in either a database crashing or the memory in the stack being overwritten (including the return address of the calling function). This can result in an exception being thrown, or worse yet, an attacker taking full control of the system. AppRadar Sensors can pick up on these attack patterns:

BULK INSERT buffer overflow	xp_createqueue buffer overflow
DBCC addextendedproc buffer overflow	xp_decodequeuecmd buffer overflow
DBCC BUFFER buffer overflow	xp_deleteprivatequeue buffer overflow
DBCC checkconstraints buffer overflow	xp_deletequeue buffer overflow
DBCC CLEANABLE buffer overflow	xp_displayparamstmt buffer overflow
DBCC INDEXDEFRAG buffer overflow	xp_displayqueuemsgs buffer overflow
DBCC PROCBUF buffer overflow	xp_dsninfo buffer overflow
DBCC SHOWCONTIG buffer overflow	xp_execresultset buffer overflow
DBCC showtableaffinity buffer overflow	xp_mergelineages buffer overflow
DBCC UPDATEUSAGE buffer overflow	xp_oledbinfo buffer overflow
General access violation	xp_peekqueue buffer overflow
OpenDataSource buffer overflow	xp_printstatements buffer overflow
Openrowset buffer overflow	xp_proxiedmetadata buffer overflow
pwdencrypt buffer overflow	xp_readpkfromqueue 1st param overflow
pwdencrypt exploit	xp_readpkfromqueue 3rd param overflow
RAISERROR buffer overflow	xp_readpkfromvarbin 1st param overflow
sp_OACreate buffer overflow	xp_readpkfromvarbin 3rd param overflow
sp_OAGetProperty buffer overflow	xp_resetqueue buffer overflow
sp_OAMethod 5th param overflow	xp_showcolv buffer overflow
sp_OAMethod 6th param overflow	xp_sprintf format string overflow
sp_OASetProperty buffer overflow	xp_sqlinventory buffer overflow
Well-known xp_peekqueue exploit	xp_unpackcab buffer overflow
xp_controlqueueservice buffer overflow	xp_updatecolvbm buffer overflow
xp_createprivatequeue buffer overflow	

WEB APPLICATION ATTACKS

Rules within this category can be enabled to monitor against possible access-related attacks. Attacks may include attempts to elevate privileges and gain access to powerful resources within a Microsoft SQL Server database.

Comments - SQL Injection	Error - SQL Injection
Cross-site scripting	Mismatched quotes - SQL Injection
Data Thief attempted	SET NOEXEC - SQL injection
Data Thief successful	UNION clause - SQL Injection

PRIVILEGE ESCALATION

It is possible for a low-privileged user to exploit vulnerable extended stored procedures to effectively bypass access controls. This category of rules alerts on the exploitation of these kinds of stored procedures.

- Elevate privilege using XPs
- OPENROWSET privilege escalation
- Run-time patch exploit
- SQL Agent privilege escalation
- SQL injection in sp_MSdropretry

ACCESSING OS RESOURCES

Microsoft SQL Server provides extended stored procedures that allow operating system commands or registry changes to be executed from Transact-SQL as if from a command-line prompt. This category focuses on the execution of these stored procedures.

- Generic use of xp_cmdshell
- Injection in sp_attachsubscription
- Read sensitive OS files
- Registry backdoor installed
- SAM database in registry accessed
- Using Adhoc queries with Jet

PASSWORD ATTACKS

Attempts to guess passwords by trying likely combinations of characters or exploiting certain Microsoft vulnerabilities are simplistic attacks that can be used against a database.

- OPENROWSET used to find service account
- Password guessing
- Password read from files
- Scripted password attack

SYSTEM EVENTS

These rules uncover system level events such as the starting and stopping of the database being monitored and the starting and stopping of the AppRadar Sensor.

- Agent registered
- Agent started
- Agent stopped
- Agent unregistered
- Database started
- Database stopped

CONCLUSION

Monitoring of databases within an organization has become a critical component of a proper layered security approach, however to be useful, the monitoring must be done using the right combination of tools and best practices. Monitoring is not a replacement for the other layers in the security stack – it is a complementary piece that greatly decreases the likelihood of successful attacks. A major driver for the need to monitor and audit databases is the rise of government and industry regulations on how sensitive data is handled. These regulations include:

- ❑ VISA standards for credit card vendors
- ❑ Sarbanes-Oxley
- ❑ HIPAA (Health Insurance Portability and Accountability Act)
- ❑ European Union Data Protection Directive
- ❑ California’s Database Security Breach Notification Act (California Senate Bill 1386)
- ❑ Gramm-Leach-Bliley Act

There are security elements that go beyond monitoring which must be considered when creating a data security policy. Vulnerability assessment, encryption, and database integrity solutions help to ensure a solid security foundation for storing sensitive data. To provide effective and holistic security, be sure to incorporate defense mechanisms at all the different layers.

ABOUT APPLICATION SECURITY, INC.

AppSecInc is the leading provider of database security solutions for the enterprise. AppSecInc products proactively secure enterprise applications at more than 300 organizations around the world 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.