

Database Activity Monitoring: Intrusion Detection & Security Auditing

by Aaron Newman, Application Security, Inc. CTO & Founder

INTRODUCTION

At its core, security is all about risk reduction. One of the most effective database security practices, *defense-in-depth*, employs multiple layers of protection to reduce the risk of intrusion. It is analogous to the many defensive layers surrounding a medieval castle: drawbridge, moat, the outer wall, the inner keep, archers manning the wall, soldiers stationed outside the wall, etc. No single level of defense is infallible; and yet all of these layers cannot ensure the castle will be 100% impenetrable. However, these layers of protection can make the castle (and its crown jewels) less vulnerable to attackers.

Database security is quite similar. Protecting your organization's database (where its most sensitive data, its crown jewels lie) encompasses more than a set of permissions. There are many layers of protection to consider when safeguarding your organization's databases. The first layer of defense should always be along the network perimeter, which is typically protected by a firewall. Too frequently, however, even experienced IT security professionals consider the database's network 100% protected at that point. Unfortunately, nothing could be further from the truth. Perimeter security is necessary, but it is no longer sufficient. It is unreasonable to believe attackers cannot get behind, through, or around any firewall due to the dependant and interconnected nature of networks. It is necessary to start thinking about securing your organization's database with layers — not only at the perimeter, but also at the source of the data as well. A layered defense includes, but is not limited to: Vulnerability Assessment and Intrusion Detection/Security Auditing, both of which are described as follows.

Vulnerability Assessment is the Systematic examination of networks to determine the adequacy of security measures, identify security deficiencies, provide data from which to predict the effectiveness of proposed security measures, and confirm the adequacy of such measures after implementation.¹ Vulnerability assessment involves reviewing, analyzing, and even attacking your organization's own database to find security holes. This is considered a crucial piece in a layered defense methodology because it is necessary to identify what types of vulnerabilities are in existence and where they are so that they can be properly patched and fixed. However, fixing security holes on a timely basis is hard to realize when you consider assessing vulnerabilities on a larger scale.

Intrusion Detection/Security Auditing is an additional layer of necessary security. Security auditing is an independent review and examination of data processing system records and activities to test for adequacy of system controls, to ensure compliance with established security policy and operational procedures, to detect breaches in security, and to recommend any indicated changes in control, security policy, and procedures.² Intrusion detection/security auditing is a method of monitoring and responding to an attack when it occurs, and depending on the degree of severity, it will also allow you to respond to valid-yet-potentially malicious activity. In a worst-case scenario, should an attack happen, you will be alerted on the activity and be better enabled to counter with a quicker response before your database can be thoroughly exploited and your sensitive data compromised. In addition, you will also want to locate and fix the hole through which the attackers came through.

A recent storm of security legislation demands that extra steps are taken to secure sensitive data. Now, it is not only a matter of ethics or business savvy that compel us to secure information with a sound defense strategy — it's the law. Simply put, if the proper defenses are not put in place, there will be consequences. Information security is based on preserving the CIA (Confidentiality, Integrity, and Availability) of systems. Without upholding these basic tenets, a database will not measure up to the requirements of handling commercial data. Without real-time auditing and monitoring data, CIA is impossible to maintain. While discussions continue about the need to provide some level of auditing and monitoring, there is little information available to help define what is appropriate auditing and monitoring. Subsequently, the aim of this paper is to merge "theoretical best-practices" with "real-world practicality" in order to define a usable policy for database security auditing and monitoring. By following the policies outlined in this paper, you can properly implement a solution that will work well (and will not interfere) with other aspects of the system.

Through personal experience or third parties, there have been horror stories surrounding how to comply with new regulations that require an audit to data access. The legislation has good intentions, but there is little clarity behind what it actually means.

Consider the following example: An organization's auditor is told that they must be in compliance with regulation XYZ, which specifies all access to information type ABC must be audited. Since the vast majority of private and valuable information is stored in a database, the auditor will turn to the database administrator (DBA) to find out if they are in fact auditing access to all information of type ABC. The DBA will generally decline, but then request specifics on what needs to be audited. Unfortunately, the auditor is unclear on the exact details of the claim and will insist on auditing every 'read' and 'write' of sensitive data. The DBA will realize how often sensitive information is accessed on a daily basis and try to recommend another alternative, however, the auditor will insist on his/her original request, and the effort to record every 'read' and 'write' access to information type ABC begins, requiring 100 GB of archive space per day.

This white paper should clarify what types of activities we really need to examine, and how to keep the "noise" to a minimum. "Noise," throughout this white paper, is referenced as security audit scenarios where no value is gained from seeing an alert, and it only contributes to drowning out other more valuable audit data. If 10,000 audit records are received per day, it is going to be difficult to decipher the one record that really matters because it is buried in information overload.

NATIVE DATABASE AUDITING

Most database vendors provide some form of native auditing. Microsoft SQL Server and Oracle Database Server provide a fairly robust set of auditing capabilities. This is implemented as a system that writes activity to tables, log files, or even the Event Viewer on Windows. There are several ways that activity can be recorded within Microsoft SQL Server and Oracle Database Server, but along with each method, there are shortcomings. Below we explore each option.

MICROSOFT SQL SERVER CONNECTION AUDITING

Microsoft 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; and
- 3) If an attack was successful.

This form of auditing has little negative side effects since the amount of activity being audited is minimal. The auditing of connection attempts typically 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 and the minimal impact recording a login causes, it is rarely a bad decision to audit connection attempts in this method.

The possible settings for login auditing are:

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

The auditing information is written to the Microsoft 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 is needed to effectively monitor a database. There is also no facility to provide any intelligence that lets someone know whether malicious activity is or has happened in real-time.

MICROSOFT SQL SERVER 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. However, by default, it is not enabled.

When C2 auditing is enabled, Microsoft SQL Server will track C2 audit events and record them to a file in the `\mssql\data` directory. In accordance with the requirements for C2 security, the C2 auditing feature will cause the database to stop when the server is unable to write to the audit file for any reason. 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, the server must be restarted.

C2 auditing is ineffective because the amount of information being recorded is simply overwhelming. Enabling this feature can result in running out of disk space quickly and can harm performance on an already taxed system. There is also no facility to provide any intelligence to alert someone when malicious activity is or has happened.

MICROSOFT SQL SERVER SQL TRACE

Finally, database activity can be audited 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.

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

SUMMARY: MICROSOFT SQL SERVER NATIVE AUDITING BENEFITS AND SHORTCOMINGS

The ultimate shortcoming of native auditing is that there is no intelligence built into these interfaces. For C2 auditing, tuning or filtering criteria does not exist. There is simply just an on/off switch, and no adjustments can be made to what, when, or who is being audited. SQL Trace is slightly better, however the problem persists in that there is 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 switches. There is 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 evidence of malicious activity.

All of these audit systems result in data being written to a local file or table. This is inherently insecure since it means that the data is not well protected. The data is stored locally where the very person it is attempting to audit or monitor 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 simply delete individual records to hide their footprints. The same will happen for a DBA that wants to perform actions they are not authorized to carry out. They will simply remove any record of the malicious activity from the local audit logs

ORACLE DATABASE SERVER AUDITING

The Oracle Database Server provides a fairly robust set of security auditing capabilities "out of the box." This is implemented as a system, which writes activity to tables, log files, or even the Event Viewer on Windows. There are several ways you can record activity in Oracle. Each option is explored below:

Oracle's first form of auditing is a subsystem you can use to record failed and successful 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; and
- 3) If an attack was successful.

To enable auditing in Oracle, start by configuring the proper settings in the `init.ora` file:

```
audit_trail=true
```

This is not enabled by default. You may need to execute the script `ORACLE_HOME\rdbms\admin\cataudit.sql` using the `SYS` account if the auditing subsystem was not installed, or was uninstalled. This is not usually needed because all auditing tables, views, and procedures are installed by default. You can then control the Oracle auditing subsystem using system commands such as:

```
AUDIT ALL BY user1 BY ACCESS;
```

```
AUDIT SELECT TABLE, UPDATE TABLE, INSERT TABLE, DELETE TABLE BY user1 BY SESSION;
```

```
AUDIT EXECUTE PROCEDURE BY user1 BY ACCESS;
```

The `AUDIT` command is fairly flexible. You can use it to set auditing on specific objects, commands, or actions. You can also use it to set auditing based on the user taking an action. You can record events on every access. Or, you can record just the first access for a session. To disable auditing, the corresponding `NO AUDIT` command takes identical parameters to disable the auditing you configured with the `AUDIT` command. Records are typically stored in a table called `SYS.AUD$`. This can, however, also be stored at the operating system level. To view the values from the table, use one of the following auditing views:

- `DBA_AUDIT_EXISTS`
- `DBA_AUDIT_OBJECT`
- `DBA_AUDIT_SESSION`
- `DBA_AUDIT_STATEMENT`
- `DBA_AUDIT_TRAIL`
- `DBA_OBJ_AUDIT_OPTS`
- `DBA_PRIV_AUDIT_OPTS`
- `DBA_STMT_AUDIT_OPTS`

An audit trail also contains a variety of data types. The most usable information is likely:

- Username
- Terminal
- Timestamp
- Object Owner
- Object Name
- Action Name

Auditing at this level can have several shortcomings. First, since auditing is based in the database, it can detract from the system's performance. This is especially true when you attempt to record every access to certain data; the constant reading and writing of auditing can result in substantial disk I/O on the database server, creating a bottleneck that significantly slows down database performance. Another disadvantage: since auditing data is stored in the `SYS.AUD$` table, it ends up sharing disk space with user data, resulting in possible application downtime when log files fill up.

These two disadvantages merit consideration. The bigger issue is this: control of the database implies full control of the auditing system. There is no way to:

- Provide segregation of duties
- Limit the DBA from disabling the auditing
- Limit the DBA from deleting audit records
- Limit the DBA from changing auditing configuration

Segregation of duties is the key to meaningful security and regulatory compliance. The auditing subsystem must retain integrity, and must not be manipulated by the users it is meant to monitor. The "observer" and the "observed" cannot be the same person.

This same shortcoming also applies to database intrusions. For the audit trail to maintain an acceptable level of integrity, it must be able to withstand an attacker taking control of the database — and not lose the existing audit trail; its forensic evidence.

ORACLE AUDIT_SYS_OPERATIONS

The `AUDIT_SYS_OPERATIONS` parameter logs `SYS` user operations to the operating system file that contains the audit trail. This parameter was added to Oracle because, in earlier versions, these actions could not (and still cannot) be logged to the `SYS.AUD$` table. To configure the operating system file to log data, set the parameter `AUDIT_FILE_DEST` in the `init.ora` file. This parameter does not affect Microsoft Windows environ-

ments since, by default, all audit data is written to the event log. Nor does the parameter affect other parameters, such as AUDIT_TRAIL. This parameter is new as of Oracle9i release 2. By default, this value is set to false. You can enable this setting by adding the following line to the init.ora file:

```
AUDIT_SYS_OPERATIONS=true
```

After changing this value, you must stop and restart the database. (For additional details on this security settings, review the Oracle documentation at http://www.oracle.com/pls/db92/db92.drilldown?remark=&word=AUDIT_SYS_OPERATIONS&expand_all=1.)

The AUDIT_SYS_OPERATIONS also has shortcomings similar to those found in other Oracle native auditing methods. Auditing directly impacts system performance since it runs in the Oracle software. In addition, audited data is not protected against attackers who successfully break in and gain control of the database. Finally, the data is not protected against the DBA, who is the individual AUDIT_SYS_OPERATIONS is designed to track and monitor.

SUMMARY: ORACLE NATIVE AUDITING BENEFITS AND SHORTCOMINGS

Oracle provides the most comprehensive native audit functionality offered by any commercial database vendor. These built-in database functions allow administrators to roll out a granular auditing system, capable of monitoring and logging any and all database activity. While this functionality is extremely powerful, it is far from an ideal solution.

The ultimate shortcoming of native auditing is that there is no intelligence built into the interfaces. For AUDIT_SYS_OPERATIONS, tuning or filtering criteria does not exist. There is simply an on/off switch. No adjustments can be made to what, when, or who is being audited. Database Auditing is slightly better, however the problem persists in that there is no intelligence built into the auditing feature. It takes a significant amount of effort to turn on all the right switches and turn off all the wrong switches, plus this must be done individually for each database server. This creates a management nightmare, as any changes must be configured server-by-server, wasting time and introducing risk of human error.

Oracle Database Auditing includes no logic to detect and highlight malicious activity, nor can this be configured. Database auditing is great at amassing a huge amount of data, but is use-

less in finding the “needle in the haystack” that is evidence of malicious activity.

Additionally, all of these audit systems store data in a local file or table. This is inherently insecure since it means the data is not well protected. The data is stored locally where the very person it is attempting to audit or monitor 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 simply delete individual records to hide their footprints. The same will happen for a DBA that wants to perform actions they are not authorized to execute. They will simply remove any record of the malicious activity from the local audit logs.

WORKING TOWARD AN IDEAL DATABASE MONITORING AND SECURITY AUDITING SOLUTION

Now that the shortcomings of native database auditing options have been discussed, let's consider the characteristics of an ideal solution. We will start with the assumption that monitoring data is a complex task, and collecting data is simply part of the work. The trickiest aspects of monitoring a database are:

- 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

Presently, in most organizations, the DBA is the unrestricted owner of the database. An organization's most critical information is entirely exposed and controlled by this small handful of technologists. This leaves both the DBA, and the entire organization, in a precarious position. On one hand, the DBAs are afraid they will be blamed for any information leak, while on the other hand, the organization is forced to trust a small group of professionals in its technology group.

One way to mitigate risk is to audit and monitor DBA activities. It is necessary to limit the amount of work a DBA does on a production server. Security auditing and monitoring this data should not add significant overhead to any system.

How do you properly audit database activity? Not through native auditing, which fails here because it is fully under the control of the DBAs, who can turn off auditing, clear the audit logs, manipulate an audit record, or even reconfigure auditing to filter their own malicious activity. Auditing should ultimately enable a separation of duty. An ideal audit system is intelligent enough to distinguish database administration accounts, filter out "noise" and irrelevant events, and succinctly illustrate its activities. As well, the system should write audited data to a secure location where even the DBA would not have direct control over the recorded activity.

WATCHING TEMPORARY ACCOUNTS

Another type of activity that requires monitoring and security auditing 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 as required. For instance, the DBA will request that the operations team 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 an adequate system for reducing the exposure of a malicious database administrator. However, it still leaves some exposure in that it is difficult to track exactly what that administrator does during the period of time the temporary account exists. An ideal monitoring and security auditing system can provide real value in this situation. A system that can track the activity of the temporary database administrator can help you to easily review the activities and ensure that nothing malicious occurred.

AUDITING ACCESS TO SENSITIVE DATA

Your security auditing system should also monitor access to sensitive data in a subset of tables. A typical database contains massive amounts of data. Some of this data is not sensitive at all. However, if sensitive data falls into the wrong hands, the consequences could be disastrous. Auditing every database action can lead to information overload. For instance, if you have a lookup table to map a product to a product ID, there is not much value in auditing access to that table. That table may be accessed thousands of times a day, and auditing all those accesses to the table would result in so much "noise" it could bury a real attack. Other tables may include credit card numbers, payroll information, or social security numbers. Access to these tables should be audited and monitored closely.

This type of security auditing requires that DBAs or application owners decide beforehand what data is sensitive, and define it as such in the security auditing system. The security auditing system should be able to accept and configure the list of databases, tables, objects, and columns to monitor, and should also be easily configured to monitor specified actions on the table. For instance, if you have static data that is public information, you may not want to audit who performs a SELECT from the table. However, you definitely want to record who modifies the data. In that case, you need the ability to audit any UPDATE, DELETE, or INSERT made by any user.

FLEXIBILITY TO FILTER RESULTS

There exists a real need to filter how data is audited based on who is accessing the data. For instance, HIPAA (Health Insurance Portability and Accountability Act) regulations require strong accountability to access of patient records. If a system administrator accesses patient Jane Smith's medical records on June 15th, there must be a record of the action to ensure accountability for the data. On the other hand, if the patient's doctor accesses the data twenty times in a day, there is little value in recording this activity multiple times. Security auditing should record unauthorized users' attempts to access data — yet it should be flexible enough to minimize the "noise" level. Keeping "noise" level down can be accomplished by minimizing the recording of activity performed by authorized personnel. An ideal security auditing solution has the capability to filter auditing based on factors such as account name, source of activity, and the time of the activity.

ATTEMPTS TO CIRCUMVENT AN APPLICATION

Another common problem that should be monitored and audit-

is when users circumvent an application and connect directly to the database. This is especially problematic with two-tier 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 back-end database. Both architectures require you to monitor for the use of utilities such as Microsoft Access, Microsoft Excel, SQL*Plus, or even Query Analyzer to connect directly to the database. Users doing this may simply be trying to make their job easier, but they are, in effect, opening a security hole in the database. A frequent high-risk scenario involves users placing a linked Microsoft Excel spreadsheet on an open file share, allowing other users to see the results of their work in the database. Unfortunately, this linked spreadsheet can be manipulated to pull back other information from the schemas or tables.

There are two ways to audit for this particular problem. One way is to specifically watch for people using tools such as Microsoft Office to access the database. To do this effectively, you must be able to list the most common applications that a curious or well-meaning user might employ. The other strategy is to audit any connections not from the expected application name. For instance, consider a database connected to via an application called HRPayrollApp. You should be able to configure the security auditing system to raise an alert when someone connects to the database using any application other than HRPayrollApp.

SECURITY AUDITING EXCEPTIONS

Security audit systems should be able to “approve” traffic to prevent valid activity from continuing to trigger alerts. For instance, an application may legitimately access data in the database that is being monitored by the security auditing system. This is important to know during installation and set up of the security auditing system. However, the drawback is that it becomes “noise” after you see that data a few hundred times. This is why it is so important for an effective security audit system to reduce the number of items audited, and only catch the items an organization cares about. This goal is accomplished by allowing “exceptions.”

For example, an exception might be:

“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 security audit system should record the activity. A proper security auditing system should also be able to record any activity that does not

match specific criteria. For instance, the system should allow you to:

“Record all activity except for SQL statements from application XYZ.”

MANIPULATING WEB APPLICATIONS

Monitoring database activity originating from a web application is also important. An ideal security auditing solution should detect when someone manipulates a web application. How is this accomplished? To start with, the monitoring system has to learn what type of data the web application sends to the database. This activity would compile a complete list of each SQL statement executed by the web application. Then, when a query is executed, the security auditing solution should compare it against a list of “known” SQL statements. If the SQL statement has been modified — for example, by injecting additional SQL statements into the web application — the SQL statement will not match the list of “known” SQL statements.

The security auditing system will then trigger an alert, and the monitoring system will then send notification of the unauthorized activity.

How would a system go around getting a complete list of SQL statements generated by the web application? Three methods follow:

- 1) Run the monitoring system in learn mode during regular operations
- 2) Feed the application a list of the valid SQL statements
- 3) Enumerate the list of SQL statements

Recording the list of valid SQL statements requires a certain amount of intelligence. You cannot simply record all 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 one or two different SQL statements?

For the purposes of watching for the manipulation of a web application, these are the same statement, and the second example should not generate an alert. The monitoring and

security auditing 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, 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 and security auditing system is its ability to identify atypical activity (i.e., activity that is unusual, and may be in violation of corporate policy). An ideal tool should classify activity into patterns, and based on those activity patterns, identify usage patterns. This is useful in determining if unauthorized activities are taking place, or if corporate policies are being broken.

Consider mapping typical administrator activity. If a monitoring and security auditing system can detect when an administrator makes an uncharacteristic act, this can help ferret out wrongdoings. For example, an administrator breaks corporate policy by remotely administering the database from a home computer. Or perhaps the administrator logs into the network late at night from a remote office, raising a “red flag” for an attack from an internal employee.

A monitoring and security auditing tool should be able to properly characterize your system, then monitor for attacks, breaches in security, valid users performing unauthorized activities, and violations of corporate policies.

KNOWN ATTACKS

It is imperative that your database monitoring system detect and recognize attacks. Attacks come in many forms. When an attack occurs on your system, it should notify you that the system is under attack. Below is a sampling of the type of attacks a monitoring and security system should send alerts on:

- 1) Buffer overflows being executed from PL\SQL or 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

CONCLUSION

Monitoring and security auditing of your organization’s database applications is a critical component of achieving a strong defense-in-depth strategy around your sensitive data. However, to be efficient and effective you must use the right combination of tools. Monitoring and security auditing should never replace other necessary layers in the security stack, instead it should complement the existing pieces. Database intrusion detection and security auditing continues to grow in importance because of the rising volume of successful database breaches, and the resulting security legislation and regulations, including:

- Payment Card Industry Data Security Standard (PCI-DSS)
- Sarbanes-Oxley Act
- 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
- Federal Information Security Management Act

Clearly, database intrusion detection and security auditing comes with its complexities. Monitoring your databases is a useful tactic, but only if used in conjunction with a well-conceived and balanced security plan. Database monitoring should be a layer of defense augmenting your overall database security strategy. When used in conjunction with vulnerability assessment, encryption, and database integrity solutions, an extremely solid security solution can be implemented. When considering the use of database monitoring and security auditing solutions be sure to select a tool that will work well with other database security products. This will ensure an effective and holistic approach to security by incorporating and integrating all the different layers. By doing so, you will more effectively fortify your castle (database) and crown jewels (sensitive data) from modern day barbarians.

¹ Definition obtained from Bitpipe.com, <http://www.bitpipe.com/tlist/Vulnerability-Assessments.html>

² Definition obtained from Bitpipe.com, [http://www.bitpipe.com/tlist/Auditing-\(Computer-Security\).html](http://www.bitpipe.com/tlist/Auditing-(Computer-Security).html)

ABOUT APPLICATION SECURITY, INC. (APPSECINC)

AppSecInc is the leading provider of application security solutions for the enterprise. AppSecInc's products - the industry's only complete vulnerability management solution for the application tier - proactively secure enterprise applications at more than 400 organizations around the world. By securing data at its source, we enable organizations to more confidently extend their business with customers, partners and suppliers while meeting regulatory compliance requirements. 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.

**APPLICATION
SECURITY, INC.**

www.appsecinc.com