Measuring and Optimizing Database Security Operations: An Open Model

Findings from the Database Security Quant Research Project

Version 1.0
Released: April, 2011

Securosis, L.L.C. 20930 North Tatum Blvd Suite #110-116 Phoenix, AZ 85050 **T** 602-412-3051 info@securosis.com www.securosis.com

Author's Note

The content in this report was developed independently of any sponsors. It is based on material originally posted on the Securosis blog <http://securosis.com/blog>, but has been enhanced, reviewed, and professionally edited.

Special thanks to Chris Pepper for editing and content support.

Licensed by Application Security Inc.

About AppSec:

Founded in 2001, Application Security, Inc. (AppSec) has pioneered database security, risk, and compliance solutions for the enterprise. AppSec empowers organizations to assess, monitor and protect their most critical database assets in real time, while simplifying audits, monitoring risk, and automating compliance requirements.

As the leading provider of cross platform solutions for the enterprise, AppSec's products – AppDetectivePro for auditors and IT advisors, and DbProtect for the enterprise – deliver the industry's most comprehensive database security solution. With over 2,000 customers in 42 countries, AppSec is headquartered in New York City and has offices throughout North America and the United Kingdom.

For more information, please visit www.appsecinc.com.

Contributors

The following individuals contributed significantly to this report through comments on the Securosis blog and follow-on review and conversations:

'ds'

Russell Thomas

Copyright

Executive Summary

Developing an Open Database Security Metrics Model

The Database Security Operations Quant research project, Database Quant for short, was initiated to develop an unbiased metrics model to describe the costs of securing database platforms. Our hope is to provide organizations with a tool to better understand the security costs of configuring, monitoring and managing databases. By capturing quantifiable and precise metrics that describe the daily activities database administrators, auditors and security professionals, we can better understand the costs associated with security and compliance efforts. Database Quant was developed through independent research and community involvement, to accurately reflect all substantive efforts that comprise a database security program.

The DB Quant Challenge

What does it cost to secure a database? Database security encompasses a large number of processes managed by different teams- from DBAs, to security operations, to IT operations. Also, these processes aren't ever applied to all databases equally in the real world, and are tuned to deal with specific application and database needs.

We have built specific processes for all common database security tasks, and placed them in a superset process following a logical order appropriate for most database security operations. These steps are by no means gospel, but the metrics within the steps are likely to be where you spend a majority of time and money. The key to deriving value out of this project is to use the provided framework as a reference to improve your own processes, while picking the metrics that make the most sense for you

Key Findings

1. At the time this project started, there were no standardized processes for database security in the industry. Assessment, auditing, monitoring and related activities are, historically, ad-hoc. To frame the metrics discussion we needed to understand the daily activities of DBA's, auditors and IT managers and settle on the activities common amongst these groups. We collected process guidelines from several large enterprises in the financial and retail sectors, as well as feedback from dozens of small firms and practitioners to form a set of security processes for every common database security task. We believe that this is the first comprehensive set of database security processes ever openly published.
2. Staff time represents the majority of costs. While several tasks, such as monitoring, involve up front investment into monitoring tools, employee time for setup tasks and policy management tend to be more substantive. Continued management of systems, collection of information on threats, and verification of reports quickly outstrip sunk costs for automation software.
3. We found during our research that auditors and operations management personnel - responsible for regulatory mandates and industry compliance - followed the same set of security processes as described in the Discover &

Assess phases of this report. Compliance and security operations processes, and consequently the metrics that describe the costs associated with both efforts, are essentially the same.

4. Our results also show a great divide in the depth and complexity of the processes use by mid-market companies (those with less that $1B revenue) and large enterprises. Further, the number of participants in the process grows considerably with the size of the company. For example, database security efforts for mid-sized firms was almost the sole responsibility of the database administrator, with some assistance from general IT management teams. In large enterprise, internal auditing and security groups managed the process and requirements, with implementation being performed by database administrators (DBAs), IT and security administrators. Similarly, the depth of the processes used by larger enterprises to govern inter-departmental tasks, and tracking software to manage and automate the process was in stark contrast with mid-market forms. As a result, every process described has specific recommendations to reflect the differences between the two audiences.
5. While the processes vary by company size, key metrics that embody the majority of costs tend to be the same. And while there are many activities, there are really only a couple key metrics for every process that consume a majority of time or investment. That means even when you don't follow a formal process, the basic work to accomplish a task - more often than not- is the most resource intensive. For those looking to maintain 'ballpark' cost estimates with minimal amount of tracking overhead, it's fairly easy to identify and capture the handful of quantifiable metrics appropriate to your operations.

Key Processes and Metrics

The following represent the key tasks and associated metrics from the DB Quant Metrics project. We feel these tasks within the overarching process offer both

1. **Plan**:

Process Step	Key Metric
Configuration standards	Time to determine configuration standards requirements
AAA	Time to map business functions to logical roles
	Time to determine necessary administrative roles
Classification	Time to adjust or develop classification scheme
Monitoring	Time to identify which events/activities to monitor

2. **Discover & Assess:**

Process Step	Key Metric
Enumerate	Time to run active scan or manually discover databases
Identify Applications	Time to define patterns, expressions, and signatures
	Time to identify applications using the database
Vulnerabilities	Time to scan databases for vulnerabilities and configurations
AAA	Time to enumerate groups, roles, and accounts

3. **Secure:**

Process Step	Key Metric
Patch	Costs for maintenance, support, or additional patch management tools
	Time to test and install patch
Configure	Time to identify policy/standards violations and incorrect settings
Restrict Access	Time to implement new groups and roles, and to adjust memberships
	Time to reconfigure service accounts

4. **Monitor:**

Process Step	Key Metric
Audit	Time to review logs for policy violations and security anomalies
Monitor	Cost of DAM tool
	Time to monitor for alerts
	Time to generate compliance reports

5. **Protect:**

Process Step	Key Metric
DAM/Blocking	Time to manage incidents
Encrypt	Time to deploy, encrypt data, and set authorization rights
Mask	Cost to acquire masking products
	Time to create masking/transformation plan and configuration

6. **Manage:**

Process Step	Key Metric
Configuration Management	Time to determine required changes, includes understanding side effects
Patch Management	Time to test and deploy patches
Change Management	Validate: Time to validate change control occurred properly

How to use DB Quant

The value of any research is in how you use it to improve your operations and day to day activities. In this paper you will find a very detailed set of process steps, each of which may or may not be relevant to database security within your organization. Use what makes sense and forget the rest. In terms of the metrics, we had an excellent comment on one of blog posts from our *Network Security Operations Quant* project which puts this initiative into the proper context.

> *Who is the intended audience for these metrics? [Metrics] are part of the job, but I'm not sure what the value is. To me the metrics that are critical around process [focus on whether] the number of changes align with the number of authorized requests. Do the configurations adhere to current policy requirements, etc...*
>
> *Just thinking about [my last] presentation to the CIO, I spent 3 hours getting consensus and 2 hours on prioritizing. [How do these metrics] get me much traction?*

One of the pillars of our philosophy on metrics is that there are really three types of metrics that security teams need to worry about. This comment is about the first type: the stuff you need to substantiate what you are doing for audit purposes. Those are key issues and things that you must be able to prove.

The second bucket is numbers that are important to senior management. These tend to focus around incidents and spending. Basically how many incidents happen, how that is trending, and how long it takes to deal with each one. On the spending side, senior folks want to know about percentage of expenditure relative to total IT spending, relative to total revenues, and how that compares to peers.

Then there is the third bucket, which are **the operational metrics that we use to improve and streamline our processes**. It's the old saw about how you can't manage what you don't measure — well, the metrics defined within DB Quant represent much of what we can measure. That doesn't mean you *should* measure everything, but the idea of this project is to decompose the processes as much as possible to provide a basis for useful measurement. Again, not all companies do all the process steps. Actually most companies don't do much from a process standpoint — besides fight fires all day.

Gathering this kind of data requires a significant amount of effort and will not be for everyone. But if you are trying to understand operationally how much time you spend on things, and then use that data to analyze and improve your operations, you can get payback. Or if you want to use the metrics to determine whether it even makes sense for you to be performing these functions (as opposed to outsourcing), then you need to gather the data.

Clearly the CIO and other C-level folks aren't going to be overly interested in the amount of time it takes you to develop database authentication, authorization, and access control policies. They care about outcomes, and most of the time you spend with these executives needs to be focused on getting buy-in and updating status on commitments you've already made. Which is the way it should be.

But if you don't measure and tune your internal processes, odds are you'll be less efficient — eating up budget and being forced to rely on FUD (fear, uncertainty, and doubt) to justify future spending. Which is most definitely how it *shouldn't* be. These metrics provide the fundamental tools for you to optimize your processes, even if you only use a fraction of them.

Table of Contents

Introduction 9

Background of the Project 9

Project Assumptions 10

The Database Security Process 11

DB Quant Metrics 14

Plan Phase 16

Configuration Standards 18

Authentication, Authorization, and Access Control Policies 20

Classification Policies 23

Monitoring Policies 25

Discover and Assess Phase 28

Enumerate Databases 30

Identify Applications, Owners, and Data 33

Assess Configurations and Vulnerabilities 37

Assess Authentication, Authorization, and Access Controls 40

Secure Phase 43

Patch 44

Configure 47

Restrict Access 49

Shield 52

Monitor Phase 54

Audit 55
Monitor Activity 57
Protect Phase 59
Block (DAM) 60
Encrypt 62
Deploy WAF 65
Mask Data 67
Manage Phase 69
Manage Configurations 70
Manage Patches 72
Manage Changes 75
Conclusion 77
About the Analysts 79
About Securosis 80

Introduction

Background of the Project

Few areas of IT tend to fall through the gaps as consistently as database security. Databases hold massive amounts of critical data and drive our most important applications and business processes, but responsibility for security is split among multiple constituencies that don't always speak the same language, never mind get along. Database administrators are the ultimate authorities for their systems, yet their primary responsibility is keeping systems running. Security is, at best, a secondary priority for them. Security practitioners are charged with the overall security of an organization, yet few have a solid understanding of how database management systems work.

The result is that those most responsible for security have limited domain knowledge, and rely on a hodgepodge of external tools (if they're lucky) and insufficient privileges to verify database security. Database administrators, while responsible for managing the databases, suffer from of lack security knowledge both with the database and supporting systems.

As a result, both groups approach problems from extremely different angles, often with divergent goals, and with only limited cooperation. The situation is so bad that we don't even have consistent database security models to show how to blend and balance these responsibilities. DBA-focused models are limited to basic configuration settings, while security tends to be limited to collecting audit logs and checking user permissions and patch levels - which are rarely up to date due to the problems of patching most database platforms.

To be fair, this isn't the fault of DBAs, security professionals, or anyone else. It's merely the logical outcome of the 'economics' of the situation, and splitting up responsibilities between teams with limited contact. For most enterprises, these practitioners don't even work in the same organizations. But in recent years, in response to compliance requirements and database-focused attacks, we have finally started to see these walls erode and the practice of database security improve.

Despite these improvements, we still lack a comprehensive framework *and the metrics to measure our efforts.* There are so many ways to approach database security, with so many technical and process options, that it's hard to pull these together into a consistent process-oriented framework. And until we have the framework it's nearly impossible to determine the kinds of metrics we need to measure the cost and effort (and thus efficiency) of our database security program.

This problem isn't limited to database security- operational efficiency metrics are generally lacking in the security industry, where we tend to focus more on risk/threat metrics models — which seem to rarely be accurate. That's where *Project Quant* comes in. Beginning in 2008, Securosis initiated a series of projects to create operational metrics models for major

areas of security. They are designed to help you better understand your security processes in terms of their costs, efficiency, and effectiveness so you can both drive specific improvements and better communicate your value to non-security management.

The formal objective and scope of this project are:

> *The objective of Database Security Quant is to develop a cost model for implementing and managing database security that accurately reflects the associated financial and resource costs.*

By providing a detailed performance metrics model we hope to help organizations improve their internal processes, as well as overall efficiency and effectiveness. The model should help identify specific areas of inefficiency and guide users towards specific improvements. Project Quant is also a quantified cost model, and provides a way to measure patch management costs in different areas across their entire programs. We have used surveys and interviews to inform and support our findings, and (as with the model) all data is being made completely public. We hope this helps organizations better understand the state of patching in the industry, and their own maturity.

It's time to remove the guesswork, begin understanding the real costs of patch management decisions, and provide the open frameworks, models, metrics, and data to optimize our processes.

Our design goals for this project were to:

- Build the model in a manner that supports usage as an operational efficiency model to help organizations optimize their network security monitoring and management processes, and compare costs of different options.
- Produce an open model, using the Totally Transparent Research process.
- Advance the state of IT metrics, particularly operational security metrics.

As you read through this report, it's useful to keep the philosophy of Quant in mind: the high level process framework is intended to cover **all** the tasks involved. That doesn't mean you need to *do everything*, but does mean this is a fairly exhaustive list. Individual organizations then pick and choose those steps which are appropriate for them. As such, this model is really an exhaustive framework that can kickstart your efforts to optimize database security processes.

Project Assumptions

To achieve our DB Quant goals, we made certain assumptions:

- *This should be a quantified metrics model, focused on costs:* All the metrics or variables in the model should be measurable with accuracy and precision. "Qualified" metrics, such as risk and threat ratings, are not included. This model is designed only to measure the costs of database security processes, and to identify operational efficiencies or deficiencies in specific process areas. It relies on measurable, quantifiable inputs, rather than assessments or other unquantifiable values based on human judgement.
- *The model should apply to all relevant activities in scope:* The scope includes planning your database security, implementing security controls, monitoring database security operations, collecting needed activities and audit logs, and the ongoing management of database security. Due to the wide scope, the metrics don't necessarily include every

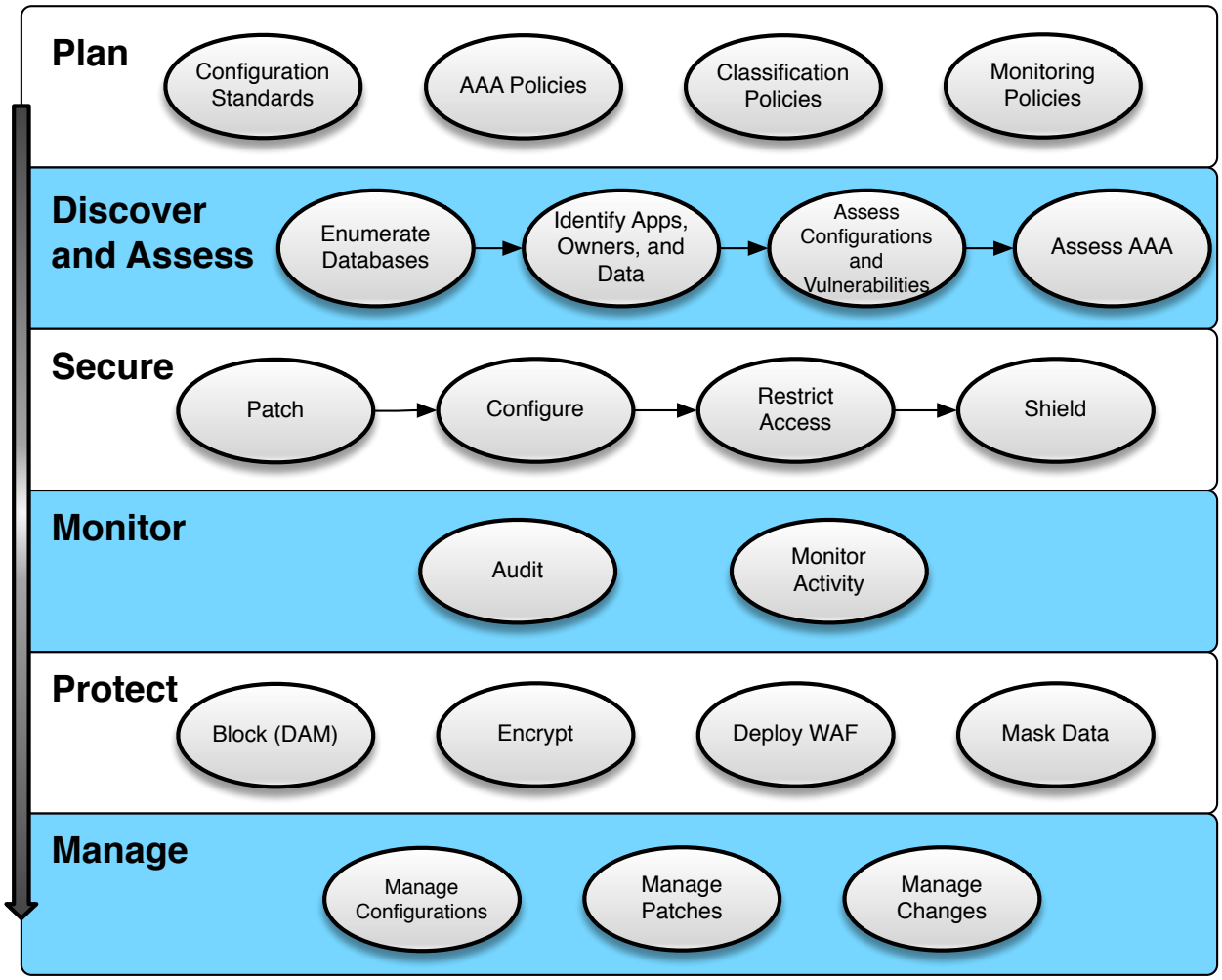
possible cost; we have focused on those that encompass the majority of database security spending in most situations.

- *The model should apply to organizations of any size or vertical:* This is not designed only for large organizations in particular vertical markets. Although smaller organizations work with fewer resources and different processes, the model still provides a functional framework.
- *The model represents a superset of database security activities:* To achieve the dual goals of covering every activity in scope, and applying to organizations of differing sizes and verticals, the model was designed as a superset of any one organization's activities. *We do not expect users to utilize the entire model, and you are encouraged to adapt it for your own particular needs.* We understand collecting all this data could actually cost more than managing the databases. Over time we hope that more and more of these metrics will be available through automation, included in support tools , and from service providers.
- *The model should break out costs by process to support optimization:* One reason for the extensive detail in each process is to support identification of specific operational efficiencies and problems. Our goal is to help organizations identify and correct problem areas; so this project defines all aspects of each process in gory detail to enable data collection, analyses on process efficiency, and trending.
- *The model cannot measure the costs of not securing your databases:* Clearly, the easiest way to reduce your network security operational costs to zero is to do nothing. While there are many ways to 'measure' the business impact of not protecting your networks, they are not part of this model. In this project we are concerned only with measuring the costs when you *do* protect your networks. This is primarily due to our strict focus on quantified metrics: addressing the impact of *not* monitoring or managing network security devices would require us to include predictive and more subjective elements.
- *Not all databases require the same security:* All organizations use databases of different value, and not all of them need to be secured the same. Just as this is a superset of processes for your entire database security program which you can pick and choose from, even within your organization you can select controls to best meet the needs of individual databases.

The Database Security Process

With that preamble to provide context, let's go over how we have broken up a very large set of operational processes. We divided the macro process into six subprocesses which are typically performed in sequence. These should work for both new and existing database programs, and new and existing databases.

1. Plan
2. Discover and Assess
3. Secure
4. Monitor
5. Protect
6. Manage



Before we discuss each phase, we need to acknowledge that this is a lot of information. As we mentioned before, our philosophy is to build out a large framework with many options, so individual organizations can then pick and choose just what they need. We know not everyone performs all these steps, but this is the best way to build something that works for organizations of different sizes and verticals. Most of you will apply a subset of the model to a couple critical tasks to assess effectiveness.

Plan

In this phase we establish our standards and policies to guide the rest of the program. This isn't a one-time event, because technology and business needs change over time. Standards and policies should be considered for multiple audiences and external requirements,

1. *Configuration standards:* Identify sources for configuration standards (DB vendor, NIST, CERT, etc). Develop internal security and configuration standards for all supported database platforms.

2. *Authentication, authorization, and access control policies:* Policies around user management and use of accounts -- including connection mechanisms, DBA account policies, DB vs. domain vs. local system accounts, and so on.

3. *Classification policies:* Set policies for how data will be classified. Note that we aren't saying you need complex data classification, but you do need to establish general policies about the importance of different kinds of data (*e.g.,* PCI related, PII, health information) to properly define security and monitoring requirements.

4. *Monitoring policies:* Develop security auditing and monitoring policies, which are often closely tied to compliance requirements.

Discover and Assess

Here we find our databases, determine which applications use them, what data they contain, and who owns the system and data; then assess the databases for vulnerabilities and secure configurations. One of the more difficult problems in database security is finding and assessing all the databases in the first place.

1. *Enumerate databases:* Find all the databases in your environment. Determine which are relevant to your task.

2. *Identify applications, owners, and data:* Determine who is responsible for the databases, which applications rely on them, and what data they store. A primary goal here is to use the applications and data to classify the database by importance and sensitivity of information. You will also need access for future tasks.

3. *Assess vulnerabilities and configurations:* Perform a configuration and vulnerability assessment on the databases.

4. *Assess authentication, authorization, and access controls:* Collect and evaluate the allowed authentication methods, entitlements for user accounts, and user/role authorizations.

Secure

Based on the results of the configuration and vulnerability assessments, next update and secure the databases. Also lock down access channels and look for any entitlement (user access) issues. All these requirements vary based on the policies and standards defined in the Plan phase.

1. *Patch:* Update the database and host platform to the latest security patch level.

2. *Configure:* Securely configure the database in accordance with your configuration standards. This might also include ensuring the host platform meets security configuration requirements.

3. *Restrict access:* Lock down access channels (*e.g.,* review ODBC connections and ensure communications are encrypted), and check user entitlements for any problems, such as default administrative accounts, orphan accounts, or users with excessive privileges.

4. *Shield:* Many databases have their own network security requirements, such as firewalls or VPNs. Although directly managing firewalls is outside the domain of a database security program, you should still engage with network security to make sure systems are properly protected.

Monitor

This phase consists of Database Activity Monitoring (DAM) and database auditing. Monitoring tends to be focused on granular user activity and real time policy enforcement, while auditing is more concerned with traditional auditing and forensic analysis. Both technologies enforce policies defined in the Plan phase.

1. *Audit:* Collection, management, and evaluation of database, system, and network audit logs (as relevant to the database).

2. *Monitor Activity:* Granular monitoring of database user activity with Database Activity Monitoring.

Protect

In this phase we apply preventative controls to protect the data as users and systems interact with it. This includes using Database Activity Monitoring for active alerting, encryption, data masking for data moved to development, and Web Application Firewalls to limit database attacks via web applications.

1. *Block with Database Activity Monitoring:* In the Monitor phase we use DAM to track activity; in this phase we create active policies to generate alerts on violations or even block activity.

2. *Encrypt:* Activities to support and maintain encryption/decryption of database data.

3. *Deploy Web Application Firewalls:* Many database breaches result from web application attacks — typically SQL injection — so we have included WAFs to block those attacks. WAFs are one of the only post-application-deployment tools available to directly address database attacks at the application level. We considered additional application security options, but aside from secure development practices, which are well beyond the scope of this project, WAFs are pretty much the only tool designed to actively protect the database.

4. *Mask data:* Conversion of production data into less sensitive test data for use in development environments, and a method to obfuscate data as it is read from a database.

Manage

The triumvirate of ongoing systems and application management: configuration management, patch management, and change management.

1. *Manage configurations:* Keeping systems up to date with configuration standards, including standards that change over time due to new requirements and threats.

2. *Manage patches:* Keeping systems up to date with the latest patches.

3. *Manage changes:* Databases updates on a regular basis; including structural/schema changes, data cleansing, and so on.

DB Quant Metrics

For each database security process we have laid out a set of metrics to quantify the cost of performing the activity. We designed the metrics to be as intuitive as possible while still capturing the necessary level of detail. The model collects an inclusive set of potential security operations metrics, and as with each specific process we strongly encourage you to use what makes sense for your own environment.

Because this model includes so many metrics, we have color coded the metrics to help you prioritize:

Key	The most important metrics in a given category. Using only key metrics will provide a rough but reasonably accurate overview of costs. These are the most useful metrics for determining costs and operational efficiency, and can be reasonably collected by most organizations.
Valuable	Metrics that are valuable but not critical for determining costs and efficiency. They provide greater accuracy than key metrics alone, but require more effort to collect.
Standard	Detailed metrics to help with deep quantification of a process, but these are either less important or more difficult to quantify. They may be more difficult to collect, or might involve complex interdependencies with other metrics.

Using Key metrics alone will provide a reasonable picture of database security operations costs and a basis for improving operational efficiency and program effectiveness. Including Valuable metrics, or Valuable and Standard metrics, provides greater detail.

How to Use the Metrics

We recommend most organizations start at the process level. That involves matching each process in use within your organization against the processes described in this research, before delving into individual metrics. This serves two purposes:

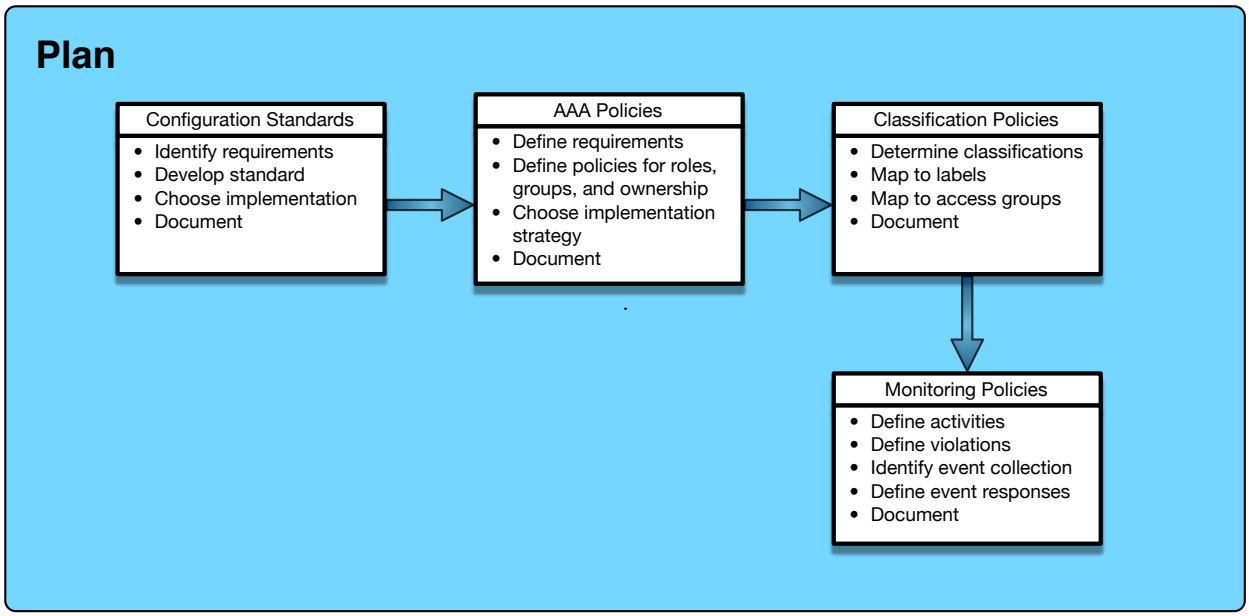
- First, it helps document your existing process or lack thereof. Since all the metrics in the model correlate with steps in the DB Quant processes, you'll need this to begin quantifying your costs.
- Second, you may find that this identifies clear deficiencies in your current process, even before evaluating any metrics. This provides an opportunity for a quick win early in the process to build momentum.

We include the applicable metrics for each specific process and subprocess, which can be built up to quantify your entire database security program. Thus you make detailed measurements for all the individual processes and then combine them, subtracting out overlapping efforts. Most of the metrics in this model are in terms of staff hours or ongoing full-time equivalents; others are hard costs (*e.g.,* licensing fees, test equipment, etc.).

It's important to keep the purpose of these metrics (and the entire Quant research program) in context. **The precision of the measurement is less important than the consistency and completeness of your efforts.** If you do have the ability to fully quantify costs for each step in the process you'll get a more accurate result, but this isn't realistic for most organizations. That said, with the right tools and automation you may be able to come extremely close for certain processes, so *think in terms of the average cost (or time) for any given step*. As long as your method of estimation is consistent, you'll get useful metrics.

Plan Phase

As with any development project, your motivation and goals should be documented up front and later used to gauge the success of your effort. For security efforts, like most IT projects, gathering requirements is a large part of the work. We initially thought a single process applied to each effort (configuration management, auditing & monitoring, access control, and data protection) would work, as there is considerable overlap between them, but when we dug into specific projects we started seeing important differences. The result is four planning subprocesses.



Many of you are probably saying "Holy @&!^@! Just planning is a huge effort! Where do I begin?" Identifying requirements for database security, or PCI, or anything else can be lengthy and complex; and it's not always clear where to find this information. While our focus in this project is identifying and quantifying costs to secure databases, we can't totally ignore what it takes to do the work, and we need to provide a some pragmatic advice along the way. We'll help steer you in the right directions as much as possible through this document.

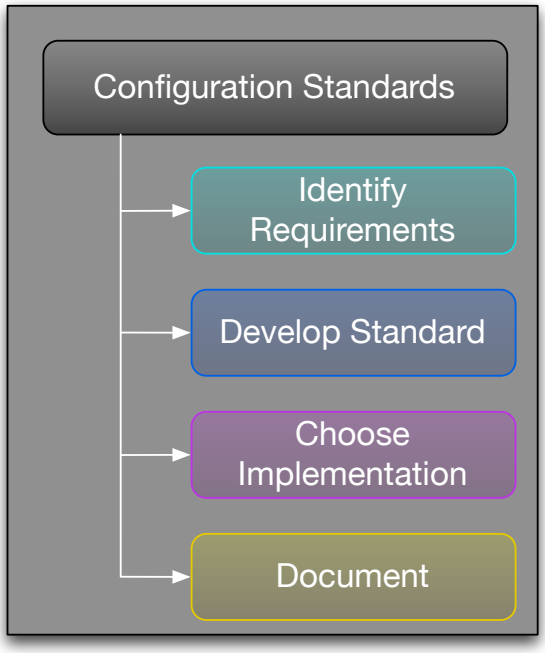
For now, consider that every other database administrator has the same set of security challenges. Ask peers what they are doing to meet security requirements. Database vendors are also a good place to start, as they provide recommend setup and configuration, and list recent security notifications. Leverage security and operations personnel within your company to highlight security issues. Look to local DBA groups for advice on how they set up databases securely. As far

as compliance, you can wade through the law doing your best to understand it, but if you have co-workers who specialize in audit and compliance, ask for assistance. If you have acquired 3rd party security tools, ask the vendor for recommendations (all of them provide some sort of guidance for using their tools for major compliance initiatives, which help sell their products). If your company has security guidelines in place you are lucky, so use them to help scope your tasks.

These high-level policies are designed to guide the rest of your program and will save time and costs later, because instead of having to start from scratch for every database, you have a base to either directly comply with, or to adjust as necessary for specific systems. Just make sure you document any deviations from the baseline, especially if the database is within a compliance scope.

Configuration Standards

There are four major subprocesses in developing configuration standards:



1. **Identify Requirements:** Requirements include everything from adopting database security best practices to PCI compliance. They may originate from external or internal sources. Vendor security configuration guides, NIST, CERT, and CIS benchmarks are common sources; as are compliance regulations such as PCI-DSS. Requirements — especially for industry and regulatory compliance — are generic and require some interpretation. Requirements such as "implement separation of duties" and "secure the database from SQL injection" are common, but there are many ways to meet them. The objective in this phase is to identify what needs to be done — we'll deal with *how* later.

2. **Develop Standard:** Starting from security or compliance requirements, which portions are relevant to you? This is where you specify your standards as a subset of the requirements which apply to your organization. Select settings, controls, and standards as necessary, pulling from the sources and matching against your requirements.

3. **Choose Implementation Strategy:** Most database security functions can be accomplished in more than one way. For example, "capture failed logins" can be satisfied with external monitoring or internal auditing. Satisfying a requirement on Oracle may be accomplished differently than on SQL Server. Don't get bogged down in specifics, but select a strategy that meets you standard and fits your operational model.

4. **Document Standard:** Record your findings and your decisions. If you are going through this process, odds are there are other people involved who will need to understand and adhere to the standard. A written outline is necessary to share policies.

Large vs. Small Company Considerations

One of the difficulties of building generic process maps is trying to factor in every potential scenario, then reflect them all in the process. But in the real world many of the steps in a given process are built to support scaling for large enterprise environments. So for each process we will try and highlight the major differences for organizations of different sizes.

Smaller organizations typically manage fewer databases, and generally run on less formal policies. Those of you in medium and small organizations need to rely more on off the shelf standards, of which many are available for the different database platforms. The Center for Internet Security and NIST are good resources, as are database platform vendors. Building off one of these standards, and adapting it for your own needs, is usually fairly straightforward.

A large organization may have many thousands of databases scattered across different business units. Requirements building in this environment is tougher, but the goal is to start with a baseline approved by security and then go through the tedious task of negotiating exceptions with all the various scattered teams. Rather than making the initial policy generation a negotiation exercise, focus on working with a few DBAs managing major systems of different types, then deal with exceptions and general policy changes once you start measuring compliance and verifying configurations.

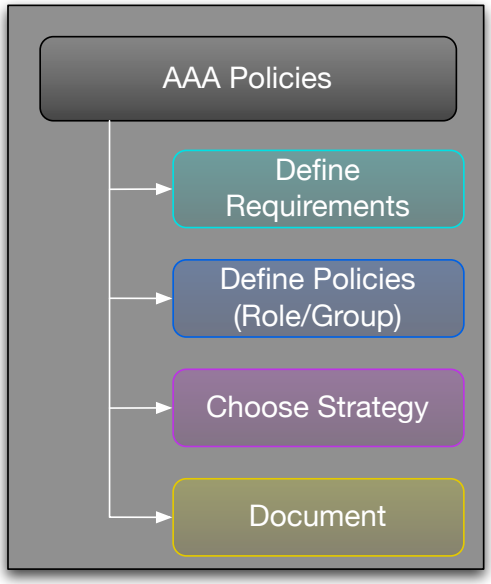
Configuration Standards Metrics

Process Step	Variable	Notes
Identify Requirements	Time to identify and collect configuration sources	*e.g.,* CIS benchmarks, NIST guidelines, vendor configuration guides
	Time to locate any existing internal standards	
	Time to identify/gather internal security requirements	
	Time to identify/gather compliance requirements	
	Time to research practices to meet requirements	
Develop Standard	Time to determine standards requirements	Define generic requirements and specifics for major platforms
Choose Implementation Strategy	Time to determine settings, controls, and configurations to meet standard	Will vary by platform
	Time to determine controls priorities	
	Time to determine responsible party	Who implements/verifies?
	Time to determine verification method	
Document Standard	Time to document standards	
	Time to obtain management approval	

Authentication, Authorization, and Access Control Policies

Crafting access strategies is time-consuming, and it is difficult to provide data security without imposing overly burdensome setup and management tasks. Compliance requirements and segregation of duties to prevent fraud make the process even more demanding. Given the fluidity of users and rules the one most likely to create security issues by varying from the specification. Databases have three classes of users: administrators, database programmers, and application users — each with very different needs. It is important to plan for additional users and roles, as database use cases change. It is very important to have a plan for revoking permissions quickly without impairing general usage. We hate to say "expect the unexpected", but with database access control planning, it's particularly important to plan for a flexible, easy to manage authorization model.

The plain truth is that you can drive yourself insane by delving too deep into these policies. Understand the needs of your organization and only go as deep as you must. It's impossible to define every possible permutation for every possible database — in this phase we are focused on the broad strokes to guide more detailed implementation and analysis on individual systems.



1. **Define Requirements:** What are the access control guidelines? Determine which business functions are being supported, which systems support those functions, who needs access to the systems, and which facilities they are allowed to use. For administrative roles, determine what tasks are performed. Identify additional security and compliance requirements (*e.g.,* separation of duties).

2. **Define Policies for Roles, Groups, & Ownership:** Based on the requirements, develop roles and groups to support business functions and enforce security constraints. Determine object and data ownership and formulate a permissions model for the database, schemas, and tables. Plan how users will obtain and lose permissions, and make some provision for use cases not included in the model. Identify service account usage.

3. **Choose Implementation Strategy:** Database permissions are established both within the database and externally. Define which facilities are responsible for policy enforcement. Define the method for verification of policy. Remember, this is a strategic planning exercise — don't get bogged down in the details.

4. **Document:** Document requirements. Clarify database use models from administration. Train administrative staff on policy.

Large vs. Small Company Considerations

As we mentioned, defining in-depth AAA policies can be time consuming for organizations of any size. All organizations should first focus on administrative access and approved authentication methods (including the account type — database vs. system vs. domain).

This might be as far as a smaller organization goes, but the next step is typically to document the users and roles for major systems — particularly accounting/finance, HR, and a few other major business systems.

A large organization with complex compliance requirements should perform this entire exercise within the scope of a larger identity management initiatives — which most have thanks to satisfy our friend, compliance. Smaller organizations may stick to a few high-level policies, then mostly focus on specific schemes for a few critical systems.

Authentication, Authorization, and Access Control Policies Metrics

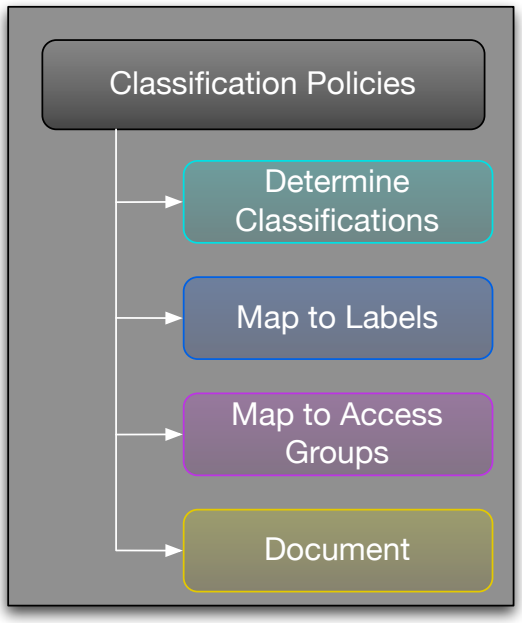
Process Step	Variable	Notes
Define Requirements	Time to identify business groups and functions	
	Time to locate internal business requirements for Access/ Authentication/Authorization	
	Time to identify/gather external security and compliance requirements	
Define Policies for Roles, Groups, and Ownership	Time to specify business functions	Only major business functions, *e.g.,* accounting for General Ledger access vs. AR
	Time to map business functions to logical roles	
	Time to determine object and data ownership	Again, only for major applications. Typically this is ERP, CRM, and HR
	Time to determine necessary administrative roles	The different DBA accounts needed to support segregation of duties
Choose Implementation Strategy	Time to identify which organizations will be supported	Both internal and external
	Time to define approved authentication mechanisms	
	Time to define allowed access control mechanisms	
	Time to identify legitimate and undesirable access methods	The different methods of connecting to the DB — *e.g.,* ODBC over SSL with approved port numbers
	Time to define database administrator roles	
Document	Time to document standard	
	Time to obtain approval, distribute standard, and educate team members	

Classification Policies

Data classification for databases is a necessary step for many compliance and data privacy regulations. In practical terms this often devolves into a giant data labeling or classification project that wastes time and effort. You will need to investigate requirements and best practices, but we recommend you avoid using an overly detailed model that nobody will actually use. Figure out what needs to be secure, but be general and pragmatic in your data security approach.

Keep in mind that this is all strategic planning. At this stage of analysis you will not be examining specific statements or policies. During this planning, there is a tendency to begin delving into implementation specifics that are simply not helpful at this stage. Focus on the big picture: how data moves and is used within the organization.

Here are the four steps in the process to create data classification policies:



1. **Determine Classifications:** What is your high level scheme? What is considered sensitive, and how will you define it?
2. **Map to Labels:** Labeling is the process of applying a classification label to data within your database. It can be performed at many levels, such as rows or columns, using different techniques. For now determine what labels will be standard in your environment and which techniques are approved.
3. **Map to Access Groups:** What is your classification model? Siloed, hierarchical, and labeling are all common options. How will your access control system implement the data security model?These models are implemented on top of access controls, but in some cases underlying data features such as labeling support more granular control.
4. **Document:** Document your classification scheme and the various label and access policies.

Large vs. Small Company Considerations

Smaller organizations tend to use fewer classifications — often limited to strict compliance requirements like PCI or HIPAA. A larger organization may have a more complex and generic scheme, with nonspecific labels such as ‘sensitive’. Note that we are not saying this is better good practice, but it is common.

In terms of applying labels, few organizations of any size get down to the row level. At minimum, consider classifying the entire database, and in larger organizations it is often useful to apply column or row-level labels on the most critical enterprise systems with the most sensitive data as you can use these to later help enforce access controls.

Classification Policy Metrics

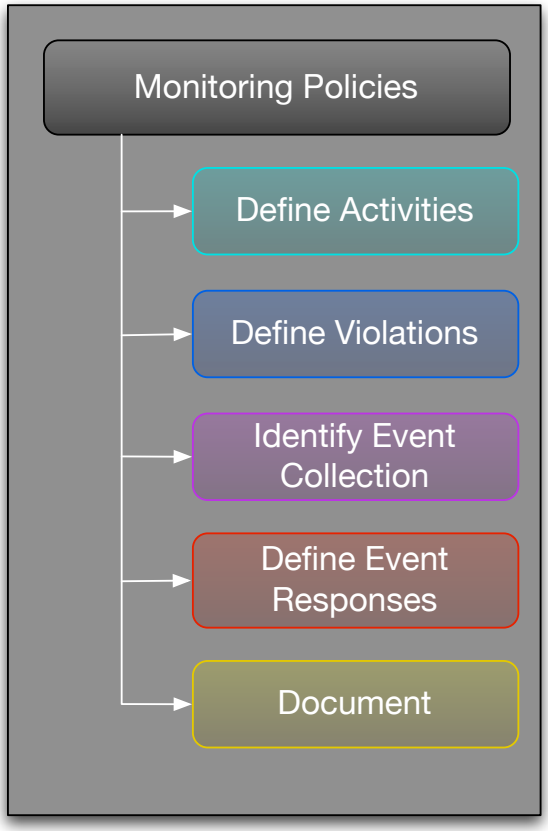
Process Step	Variable	Notes
Determine Classifications	Time to identify existing classification scheme	
	Time to adjust or develop scheme	
Map to Labels	Time to determine label techniques for DBMS platforms	Labeling is implemented differently in the various database platforms. In some cases, implementation may need to be manual
	Time to map labels to classification levels	
Map to Access Groups	Time to map groups to data labels	Which groups from the AAA planning should be allowed access to data, by label
Document	Time to document standard	This is a key step due to its role in compliance. Although still important in organizations without compliance mandates, it can be reduced to Valuable in such cases.
	Time to distribute standard and educate team members	Consensus now avoids disagreement later.

Monitoring Policies

This phase includes both Database Activity Monitoring, and database auditing. DAM (Database Activity Monitoring) verifies database usage — it can provide near-real-time analytics to track user behavior and anomaly detection. Monitoring is different from auditing in that it analyzes all activity in near-real-time, and is based more on individual queries than on transactions. Database auditing is also useful — and common — but provides fewer details and misses some major categories of usage (*e.g.,* `SELECT` queries). For monitoring systems to work, we need to define what we consider suspect behavior or abnormal use. Think of it as black and white lists for database transactions. But to build those lists you need an idea of what to accomplish, and what activities should never occur. As every database is used differently, you must define what is appropriate and what isn't. Identify events you are interested in, then define acceptable behaviors and outcomes. For auditing, you need to determine what information to collect and where to store it, both of which are influenced by compliance requirements.

The subprocesses are:

1. **Define Activities:** Investigate business processes. Define critical operations and functions. What activities does the system support, and what subset are you interested in monitoring. Identify security and compliance in relation to data privacy, fraud detection, and system misuse.
2. **Define Violations:** Determine which events indicate problems. Consider users, time of day, function, data volume, and other available attributes that can help identify suspicious transactions. Identify criticality of events and specify desired responses. Consider periodic review of general database usage in order to refine policy.
3. **Identify Event Collection:** How will you capture events? Determine what event collections are available. Map policies to event collection for misuse detection.
4. **Define Event Responses:** When a policy violation is discovered, how will you react? Specify how event notification will occur and who will be responsible.
5. **Document:** Ensure all concerned parties are aware of their responsibilities and coordination points with other groups.



Large vs. Small Company Considerations

A smaller organization with more limited resources may not even develop much of a formal monitoring policy- they'll simply define a few requirements for how to configure auditing within their database platforms. But for compliance it is important that they at least map any of these compliance requirements to what they are actually collecting. Even a small retailer has *some* database monitoring and auditing requirements if they accept payments.

Clearly this is a more time consuming task for larger organizations, and although it's one we see skipped a lot and handled on a per-database level, we strongly recommend defining at least some high-level policies to guide individual instances. Especially (again) to meet compliance requirements.

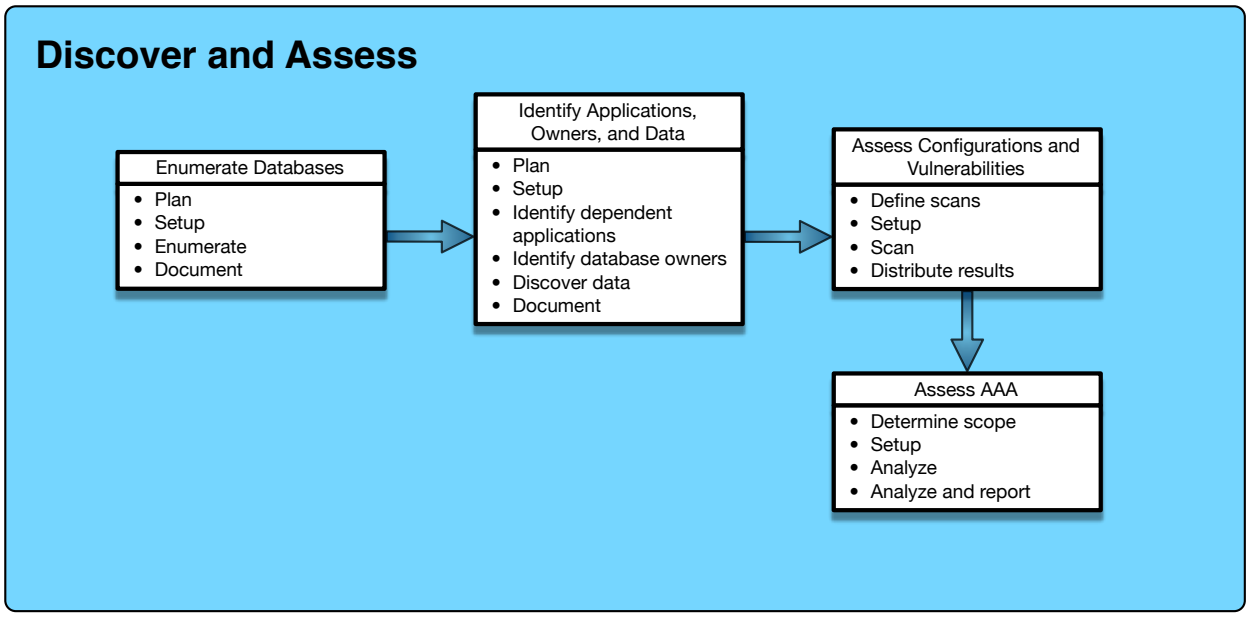
It's also important, particularly in larger organizations, to define collection and retention requirements for monitoring and auditing. Pay attention to separation of duties and where the files are stored, and who has access to them, as these are common areas for audit deficiencies and problems in real breaches — making it easy for the bad guys to cover their tracks.

Monitoring Policy Metrics

Process Step	Variable	Notes
Define Activities	Time to identify covered business processes	
	Time to gather security and compliance requirements	
Define Violations	Time to define suspect behavior	What should *not* happen
	Time to define abnormal use cases	Map which behaviors indicate security failure or a compliance event
	Time to update existing standards and policies	
	Time to determine appropriate response and criticality	Per rule violation
Identify Event Collection	Time to identify major applications (Optional)	The major data sources to be used; often ERP, financial systems, and payment processing
	Time to identify which events/activities to collect	Map which events to collect, analyze, and report
	Time to establish priority (Optional)	Define order of event processing or filtering if necessary
	Time to determine ownership	Who owns the policy?
	Time to specify notification	How suspect events and information are recorded and distributed
Define Event Responses	Time to define event response per rule	Who gets notified?
	Time to determine ownership	Who responds to the event?
	Time to specify criticality per rule	How important the event is, in terms of IT priorities
	Time to define verification method/workflow	Must provide clear evidence of completion
Document	Time to document standard	
	Time to distribute standard and educate team members	

Discover and Assess Phase

One of the most common failures in database security is treating it as a series of one-off projects, rather than evaluating and managing overall risk. Most organizations, especially large ones, tend to have a reasonable handle on their primary databases such as the ones handling corporate financials and customer transactions, but little to no visibility into all the other systems scattered throughout the organization — enterprise databases you did not know about, small 'personal' databases embedded within applications, and production data sets on test servers. And of course organizations rarely have itemized lists of configuration errors such as administrative functions open to the public and external stored procedures. In this phase we identify databases; determine what applications and business units they support; assess them for vulnerabilities; and evaluate authentication, authorization, and access controls.



If you know you have a few critical databases you need to start with, as is common when dealing with an audit deficiency, you might skip the enumeration step of this phase to focus on those systems. The other steps are generally necessary for any database security project.

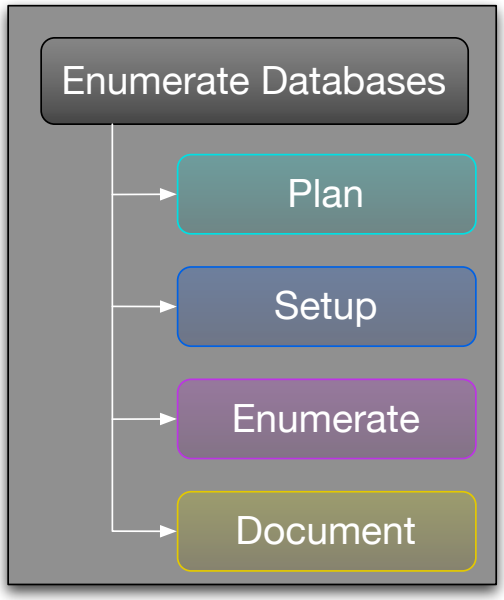
Enumeration is really the linchpin step which differentiates a series of one-off projects from a database security *program*. It's what allows you to prioritize and manage deficiencies based on data, rather than assumptions.

The rest of the steps are fairly standard for even individual database security projects, although the level of depth you go into varies with the importance of the systems.

Enumerate Databases

Database discovery can be performed manually or automated. Segmented networks, regional offices, virtual servers, multi-homed hosts, remapping of standard ports, and embedded databases are all examples of common impediments you need to consider. If you choose to automate, most likely you will use a tool that examines network addresses and interrogate network ports, which may not identify all database instances but should capture most database installations. If you are using network monitoring to discover databases you will miss some, at least in the first sweep, so consider scanning more than once. For a manual process you will need to work with business units to identify databases, and perform some manual testing to identify unreported databases. Understand what data you need to produce in this part of the process, as your results from database discovery will be used to feed data discovery and assessment.

There are four main subprocesses:



1. **Plan:** How will you scan the environment? Determine what parts of the process are automated vs. manual and make sure you have clear guidelines. You may refine the scope to portions of your environment or database types of interest. Also ascertain what you need to collect (database name, IP address, port number, database type, subnet, etc.) with your scans so that you can identify the owner or function. Finally note that the person who creates the plan may not be the person who runs the scan, so document what is expected.

2. **Setup:** Acquire and install tools to automate the process, or map out your manual process. Then configure tools (if necessary) for your environment, specifying acceptable network address and port ranges. Don't forget to account for network segregation and multiple database connection options.

3. **Enumerate:** Run your scan, manually find databases, and/or schedule repeat scanning. Capture the results and filter out unwanted information to keep the data in scope for the project based on your planning requirements. Record as you baseline for future trend reports, keeping in mind that in practice you will run this step more than once. As you discover databases you did not know existed, you'll also need to determine whether you have sufficient credentials for further analysis. If you are using a manual process, this consists of contacting business units to identify assets and manually assessing each system.

4. **Document:** Format data, generate reports, and distribute. Use results to seed the next data discovery and assessment tasks.

Large vs. Small Company Considerations

The key difference is the sheer complexity of navigating the network. This is one of those processes that is likely easier the smaller you are, since the amount of effort generally corresponds to the size of your environment. Note that 'size' in

this context has more to do with number of databases than headcount; and is of course also heavily affected by security sensitivity, number of sites, and complexity of the data and database environment.

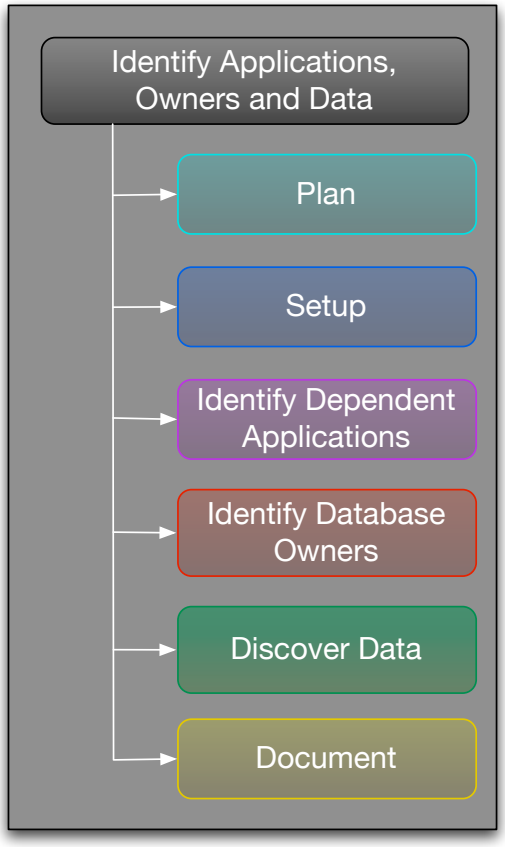
In a large company, this task is effectively impossible to initiate and maintain with any level of accuracy without automated tools. While you can keep a handle on your known critical systems, without some level of automation you won't be able to find unknown databases — including copies or versions of the known systems.

Enumerate Databases Metrics

Process Step	Variable	Notes
Plan	Time to define scope and requirements	What databases/networks are in scope; what information to collect
	Time to identify supporting tools to automate discovery	
	Time to identify business units & network staff	Who owns the resources and provides information
	Time to map domains and schedule scans	
Setup	Capital and time costs to acquire tools for discovery automation	Optional
	Time to contact business units & network staff	
	Time to configure discovery tool	Optional
	Time to contact database owners and obtain credentials and access	As needed, depending on the tool and process selected
Enumerate	Time to run active scan	If using a tool
	Time to manually discover databases	Optional, if automated tool not used. May be a technical process, or contact with business units
	Time to run scan/passive scan	Automated port scan; network flow analysis
	Time to contact business units	Identify databases discovered
	Time to manually login, confirm scan, and filter results	Optional
	Time to repeat steps	As needed
Document	Time to save scan results	
	Time to generate report(s)	
	Time to generate baseline of databases for future comparisons	Cataloged by type, version, location, and ownership

Identify Applications, Owners, and Data

The primary role of most enterprise databases is to support other applications. To achieve this databases may provide access controls, secure network communications, parameter filtering, label security, masking, and encryption services. Data and database security is therefore a function of the application & database relationship. Since database security changes are materially affected by — and in turn influence — applications and business units, it is absolutely essential to determine dependencies, ownership, and where critical/sensitive data is used and (potentially) exposed.



1. **Plan:** Develop a plan to identify the application dependencies, data owners, and data types/classifications for the databases enumerated in the previous stage. Determine manual vs. automated tasks. If you have particular requirements, specify and itemize required data and assign tasks to qualified personnel. Define data types that require protection. Determine data collection methods (monitoring, assessment, log files, content analysis, etc.) to locate sensitive information.
2. **Setup:** Databases, data objects, data, and applications have ownership permissions that govern their access and use. For data discovery create regular expression templates, locations, or naming conventions for discovery scans. Test tools on known data types to verify operation.
3. **Identify Dependent Applications:** For applications, catalog connection methods and service accounts where appropriate.
4. **Identify Database Owner(s):** List database owners. Database owners provide credentials and accounts for dedicated scans, so determine who owns database installations and obtain credentials.
5. **Discover Data:** For data discovery return location, schema, data type, and other metadata.
6. **Document:** Generate reports.

In essence, this is three separate discovery processes: discovering the applications that attach to a database, who manages them (and which business units own them), and what is stored within that database. All can potentially be performed with a credentialed investigation of the platform and system, or by observing network traffic, plus a little manual effort to tie back to the business unit owner. Credentialed scans complete provide results at the expense of requiring logins to access the database systems, while passive network scanning is easier but incomplete.

Identification of applications, owners, and data provides information necessary to determine overall security and regulatory controls — as well as the potential business impact of any changes. Woe be unto the security manager who

locks down a database before determining application and business unit dependencies. This sub-phase defines not only the scope of the scanning in the next task, but also monitoring and reporting efforts in subsequent phases.

Large vs. Small Company Considerations

As with database enumeration, this is another phase where the workload is usually more manageable for a small company, assuming the environment isn't overly large or complex, but keep in mind that small companies often have similar data proliferation problems; especially with data extracts placed into small databases such as Microsoft Access. If you handle highly sensitive or regulated information you might need some sort of automated scanning tool just like a large enterprise.

In large enterprises this step can be extremely difficult. Many teams focus on critical systems they know need security changes, and the primary goal is to discover application dependencies and owners. It can actually be harder to determine these dependencies than enumerate databases in the first place — there are entire specialized tools designed purely to map out complex applications. Mapping out the first level direct connections is usually fairly straightforward, but determining the entire application architecture and full dependencies can be difficult and time consuming if it has never been done before.

Identify Applications, Owners, and Data

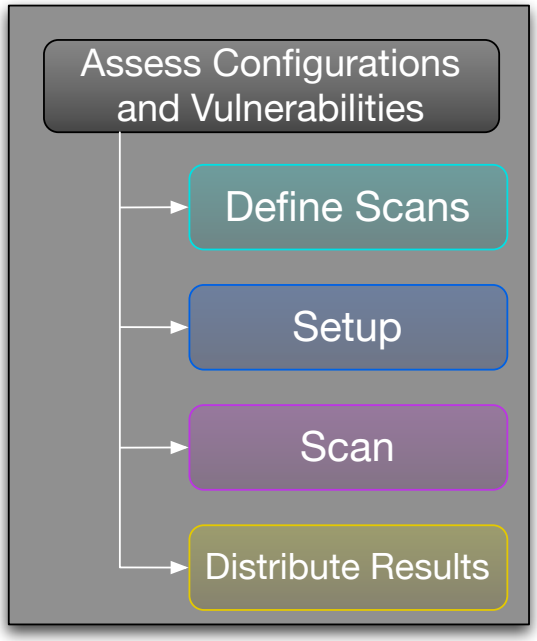
Process Step	Variable	Notes
Plan	Time to assemble list of databases	Feeds from the Enumerate Databases step
	Time to define data types of interest	The sensitive data you want to discover, such as credit card numbers
	Time to map locations and schedule scans/analysis	Databases will reside on different domains, subnets, etc. This is the time to develop a scanning plan based on location
Setup	Capital and time to acquire tools for discovery automation	Optional — DB discovery tools from previous phase may provide this
	Time to define patterns, expressions, and signatures	What sensitive data looks like
	Time to contact business units & network staff	
	Time to configure discovery tool	Optional
Identify Dependent Applications	Time to schedule and perform review/ run scan	
	Time to identify applications using the database	Based on connections and/or service account credentials
	Time to catalog application dependencies and connection types	Most items can be discovered without DB credentials
	Time to repeat steps	As needed
Identify Database Owners	Time to identify database owners	The real-world owner, not just the DBA account name
	Time to obtain access and credentials	Usually a dedicated account is established for this analysis
Discover Data	Time to schedule and run scan	For automated scans
	Time to compile table/schema locations	For manual discovery
	Time to examine schema and data	For manual discovery
	Time to adjust rules and repeat scans	For automated scans

Process Step	Variable	Notes
Document	Time to filter results and compile report	Gather data names, types, and locations
	Time to generate report(s)	

Assess Configurations and Vulnerabilities

Assessing databases is more difficult than non-database configuration and vulnerability assessment due to key differences in how database management systems store configuration settings. Rather than storing information in a configuration file, most database settings are stored within the structures of the database itself, requiring tools or processes capable of accessing these settings via SQL. Database assessment includes the analysis of database configuration, patch status, and security settings within the database as well as the host platform. It is accomplished by examining the database system both internally and externally — in relation to known threats, industry best practices, and IT operations guidelines. And while scans themselves aren't overly time consuming, the process of setting up the rules/policies to scan can be demanding.

The four steps are:



1. **Define Scans:** This is where you define what you want to accomplish. Compile a list of databases that need to be scanned and determine requirements for different database types. Investigate best practices, and review security and compliance against both internal and external requirements.

2. **Setup:** Determine how to accomplish your assessment goals. Which functions will be automated and which will be manual? Are these credentialed scans or passive? Download updated policies from tools and database vendors, and create custom policies where needed. Create scripts, if needed, to collect the information, determine priority, and suggest remediation steps for policy violations.

3. **Scan:** Scans are an ongoing effort, and most scanning tools provide scheduling capabilities. Collect results and store.

4. **Distribute Results:** Scan results will spotlight critical issues, variations from policy, and general recommendations. Filter unwanted data by audience, then generate reports. Reporting includes feeding automated trouble ticket and workflow systems.

Database discovery, data discovery, and database security analysis are conceptually simple. Find the databases, determine what they are used for, and figure out whether they are secure. In practice they are much harder. If you run a small IT organization you probably know where your one or two database servers are located, and should have the resources to find sensitive data.

When it comes to security policies, databases are so ,complex and the threats evolve so rapidly, that definition and setup tasks comprise the bulk of work for this entire phase. Good documentation, and a method for tracking threats in relation to policies and remediation information, are critical for managing assessment.

Large vs. Small Company Considerations

A large organization will definitely need some sort of automation to assess configurations and vulnerabilities. There's no way to do this manually. Many large organizations already have some form of vulnerability scanning, but unless these tools include database-specific features (especially credentialed in-DB scans) they won't provide all the necessary information. Most large organizations with mature database security processes mix up a portfolio of baseline assessments for every database in the environment, with more in-depth assessments for key systems.

Small organizations may manage much of this process manually if they have the right knowledge internally and only a small number of databases to assess. However, manual analysis can be very time consuming — there are free tools that can help with at least basic scans. And it's important to keep in mind that knowing about a CVE listed threat, and knowing how to create a security policy to address that threat, are outside the core skill set of most DBAs. Following published security hardening guidelines from your DBMS vendor and organizations like NIST and the Center for Internet Security is the best place to start and provides something you can assess against, but you really need to figure out whether the cost of this manual process is greater than using a tool.

Keep in mind that few of you assess system and server configurations and vulnerabilities manually, so it's hard to justify avoiding investment in database-specific assessment technologies.

Assess Configurations and Vulnerabilities Metrics

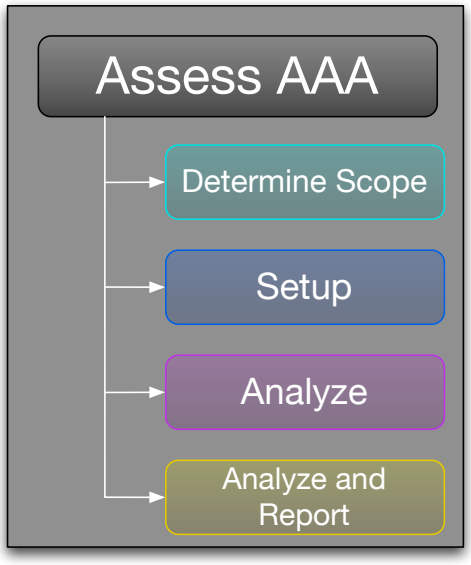
Process Step	Variable	Notes
Define Scans	Time to list databases	This may be a subset of databases, preferably prioritized, from the Enumerate phase.
	Time to gather internal requirements	Security, operations, and internal audit groups. These should feed directly from the standards established in the Plan phase
	Time to identify tasks/workflow	Should be a one-time effort
	Time to collect updated vulnerability lists	CERT or other threat alerts
	Time to collect configuration requirements	You should have this from the Plan phase, but may need to update or refine. Additionally, these need to be updated regularly to account for software patches. This includes patch levels, security checklists from database vendors, and checklists from third parties such as NIST and the Center for Internet Security.
	Capital and time costs to acquire and install tools for automated assessments	Optional
	Time to contact database owners to obtain access	

Process Step	Variable	Notes
Setup	Time to update externally supplied policies and rules	Policy is the high-level requirement; rule is the technical query for inspection. These come with the tools but may requiring tuning for your internal requirements and environment.
	Time to create custom rules from internal and external policies	Additional policies and rules not provided by an outside party
Scan	Time to run active scan	
	Time to scan host configuration	This is the host system for the database
	Time to scan database patches	
	Time to scan database configuration	Internal scan of database settings
	Time to scan database for vulnerabilities (Internal)	Access settings, admin roles, use of encryption, etc.
	Time to scan database for vulnerabilities (External)	Network settings, external stored procedures, etc.
	Variable: Time to rerun scans	
Distribute Results	Time to save scan results	
	Time to filter and prioritize scan results by requirements	Divide data by stakeholder (security, ops, audit)
	Time to generate report(s) and distribute	

Assess Authentication, Authorization, and Access Controls

Database authentication, authorization, and access controls are the front line of defense for data privacy and integrity, as well as providing control over database functions. Reviewing these controls is the most demanding of these tasks in terms of time, as the process is multifaceted -- needing to account not only for the settings inside the database, but how those functions are supported by external host and domain level identity management services. This exercise is typically split between users of the database and administrators, as each has very different security considerations. Password testing can be time-consuming, and, depending upon the methods employed, may require additional database resources to avoid impact on production servers.

The steps are:



1. **Determine scope:** Determine the list of databases you need to assess AAA controls for, then discover how they are implemented; which functions are available at the host and domain levels, and how they are linked to database permissions. Determine what password checks should be employed.

2. **Setup:** For automated scans: the cost to acquire, install and configure the tools. Then the time to obtain host/database permissions needed for manual or automated scans. You will need to collect documented roles, groups, or service requirements for users of the databases in later analysis. You will also need to generate report templates for stakeholders who will act upon scan results — these are often used for compliance auditing.

3. **Analyze:** Run scans for database users showing group and role memberships, and then scan groups, roles, and service account membership for each database. Collect domain and host user account information and settings if these are used in the AAA scheme.

4. **Analyze & Report:** Administrative roles must be reviewed for separation of duties, both between administrative functions and between DBAs and IT administrators. Service accounts used by applications must be reviewed. User accounts must be reviewed for group memberships and roles. Groups and roles must be reviewed to verify permissions are appropriate for business functions. And compliance reports must be generated for all concerned parties.

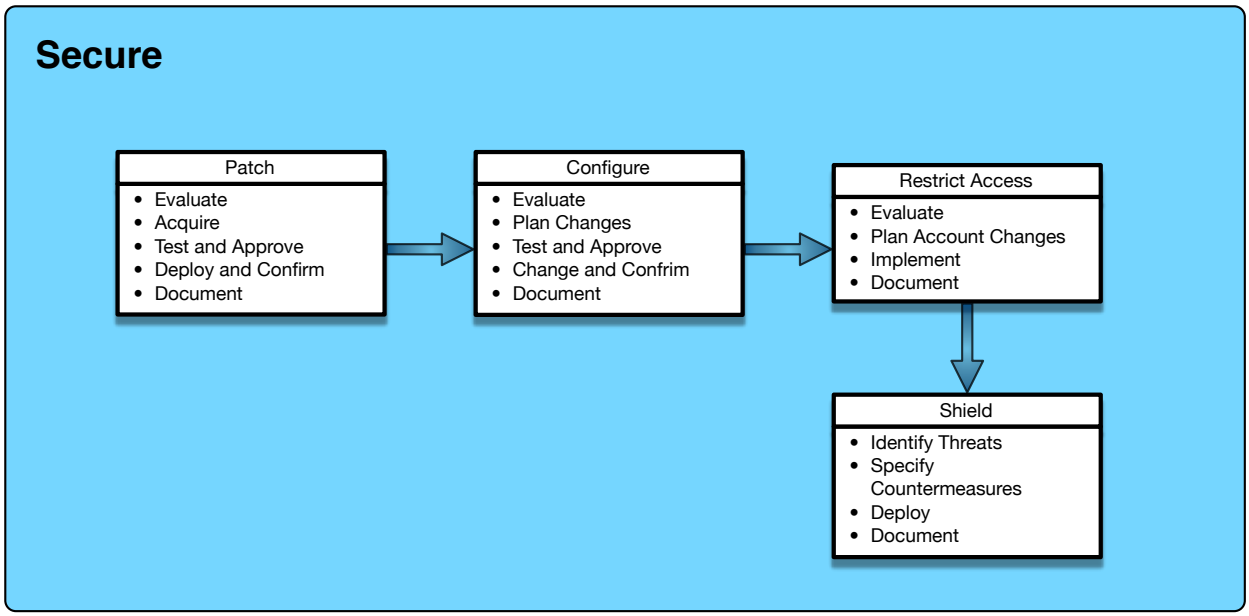
Assess AAA Metrics

Process Step	Variable	Notes
Determine Scope	Time to list databases	This may be a subset of databases, preferably prioritized, from the Enumerate phase
	Time to determine authentication methods	Database, domain, local, and mixed mode are common options
Setup	Capital and time costs to acquire and install tools for automated assessments	Optional
	Time to contact database owners to obtain access	
	Time to establish baselines for group and role configurations	Policy is the high-level requirement; rule is the technical query for inspection. Provided with the tools (if you use them), but they may require tuning for your internal requirements and environment.
	Time to create custom rules from internal and external policies	Data privacy, operational control, and security require different views of settings to verify authorization settings
Assess	Time to enumerate groups, roles, and accounts	
	Time to assess entitlements by user/role	
	Time to scan database and domain access configuration	
	Time to scan password configuration	Review aging and reuse policies, failed login limits, and inactivity lockouts
	Time to scan passwords for compliance	Optional
	Time to record results	
Analyze and	Time to map admin roles	Verify DBA permissions are divided across separate roles
	Time to review service account and application access rights	Time to verify DB system mapping to domain access

Process Step	Variable	Notes
Report	Time to evaluate user accounts and privileges	Verify users are assigned the correct groups and roles, and groups and roles have reasonable access

Secure Phase

In the Secure phase we fix whatever problems we found in the Discover and Assess phase: missing database patches, configuration changes, access control settings, and shielding the database from known attacks. For each of these preventative security tasks we've included a simplified process and only the critical relevant metrics. Due to the potential scope of some of these processes we restricted ourselves to just the most pertinent metrics for sake of simplicity and practicality.



As with many of the other phases, you need to prioritize based on the values of different databases. But database patching and configuration failures are a very common point of entry for attackers, especially for systems backing web applications. So you need to prioritize based not only on the value of the system, but also on its accessibility to the outside world.

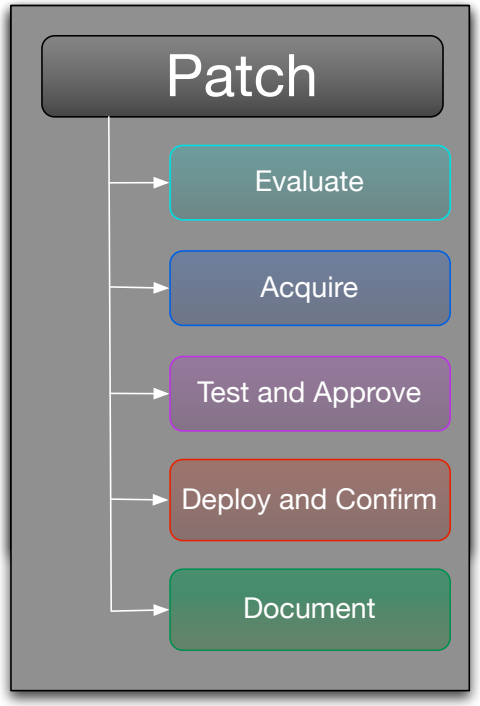
For example, in the Heartland Payment Systems breach, it wasn't the transaction database/application that was initially compromised, but a lower-value web application on a different network segment. The compromise of that low value system gave the attackers access to the internal network, which they then used to compromise additional systems, eventually finding one with a part-time VPN connection to the credit card transaction network.

Patch

Security patches are a little different than general product updates to fix other bugs. If you are experiencing a functional problem with an application, you know for certain that you need a certain patch and already possess some understanding of how critical that issue is to your firm. With security, most DBAs may not be fully cognizant of the risks known exploits pose, or what will happen if they fail to patch. If you don't have a security group helping with the analysis the evaluation process is often based on matching critical weaknesses to database features used within the environment, which isn't always effective or efficient. Security patches also must be balanced between immediate deployment for critical vulnerabilities on high-value systems where workarounds or shielding aren't available, vs. those patches you can delay until the next scheduled cycle.

Database vendors make it easy to locate and obtain patches. Security patches are well publicized and alert notices are commonly emailed to DBAs when they become available. Keep in mind that some database patches require updates to the underlying operating system kernel, libraries, or modules; and the evaluation process needs to cover those updates as well.

There are five steps in database patching:



1. **Evaluate:** Monitor sources for security advisories, which may be as simple as subscribing to an email list. When an advisory is released, determine if it is relevant to your environment and systems. Evaluate the criticality of the patch and determine your risk exposure, potential shielding or workarounds, and the priority of the patch.

2. **Acquire:** Locate and acquire the patches. This often involves costs for a maintenance contract and patches may be surprisingly difficult to obtain, depending on your vendors.

3. **Test and Approve:** Build deployment packages or scripts, develop test cases and criteria, establish a test environment, and then test the patch and any system/application/functional dependencies. Analyze the results, determine which systems and/or configurations to deploy it on, and approve for deployment.

4. **Deploy and Confirm:** Schedule the patch for deployment. Prep the target systems for maintenance (*e.g.,* backup and put application into maintenance mode), install the patch, and verify successful deployment. This may involve specific functional testing because it isn't uncommon for patching tools/scripts to report success without applying the actual patch.

5. **Document:** Document successful deployments and update configuration documentation with the updated patch levels.

Additional Considerations

There are typically two factors that affect database patching: the organization size, and the database platforms involved. Some platforms are much more difficult to patch than others and require complex scripts and series of manual steps. Some databases aren't supported by any patch management tools, exacerbating the problem by requiring a laborious manual process for every patch to be installed. Some database vendors supply unreliable automated patch deployment tools, which require manual validation.

So although a larger organization is more likely to have automated tools available, these may or may not be applicable depending on the database platform and other factors.

Smaller organizations may be more likely to use database platforms with better patch deployment options, such as SQL Server vs. DB2 on an IBM mainframe. On the other hand, they are less likely to have a proper test environment available.

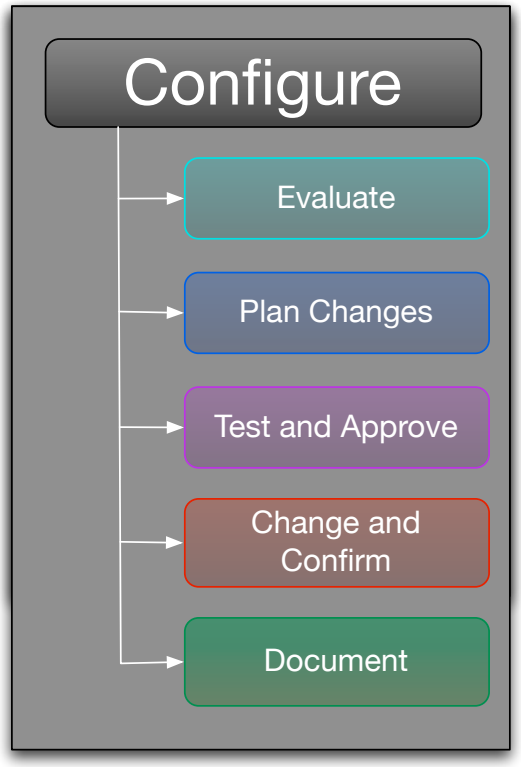
Large organizations should categorize their database platforms based on patch difficulty, and leverage automation wherever possible, even if only one database at a time will be updated. We can't express how much time manually installing a patch can take on some platforms. You should also schedule quarterly maintenance windows for critical security updates, even if you end up not needing the time.

Patch Metrics

Process Step	Variable	Notes
Evaluate	Time to monitor for advisories per database type	Vendor alerts and industry advisories announce patch availability
	Time to identify appropriate patches	
	Time to identify workarounds	Identify workarounds if available, and determine whether they are appropriate
	Time to determine priority	Is this a critical vulnerability? If so, when should you apply the patch? Are there other factors which affect importance?
Acquire	Time to acquire patches	
	Costs for maintenance, support, or additional patch management tools	Optional: Updates to vendor maintenance contracts, if required
Test and Approve	Time to create regression test cases and acceptance criteria	How will you verify the patch does not break your applications, etc.?
	Time to set up test environment	Obtain servers, tools, and software for verification; then set up for testing
	Time to run test	Variable: may require multiple cycles, depending upon test cases
	Time to analyze results	
	Time to create deployment packages	Optional — if not using stock patches. Approve, label, and archive the tested patch.
Deploy and Confirm	Time to schedule and notify	Schedule personnel & communicate downtime to users
	Time to install	Take DB offline, back up, patch database, and restart
	Time to verify	Verify patch installed correctly and database services are available
	Time to clean up	Remove temp files
Document	Time to document updated systems	
	Time to update configuration documentation	

Configure

The next task in the Secure phase is to configure the databases. In the Plan phase we gathered industry standards and best practices, developed internal policies, and defined settings to standardize on. We also established the relative importance of policy violations, so we can distinguish critical alerts which require action from from purely informational notifications. Then, in the Discovery phase we gathered a list of databases, gained access to those systems, and implemented our rules (generally in the form of SQL queries), which instantiate policies from the Plan phase. Now we take the resulting information and update our database configurations to satisfy our requirements.



1. **Evaluate:** Map the results from the configuration assessment to the configuration requirements. Identify non-compliant databases and settings for remediation.
2. **Plan Changes:** Prioritize issues and determine what configuration changes are required. Develop a plan for deploying the changes and assign/schedule.
3. **Test and Approve:** Build deployment packages or scripts, or document manual changes. Develop test cases and criteria, establish a test environment, and then test the change and any system/application/functional dependencies. Analyze the results, determine what systems and/or configurations to deploy it on, and approve for deployment.
4. **Change and Confirm:** Schedule the change for deployment. Prep the target systems for maintenance (*e.g.,* back up and put applications in maintenance mode), implement the change, and verify successful deployment.
5. **Document:** Document the changes and, if needed, update configuration policies.

Additional Considerations

Managing configuration changes for databases is very similar to managing patches, with the same issues around automation vs. manual management.

While there are few patch management tools for databases, there are even fewer which address anything but the highest-level configuration changes. Managing database configurations is almost always a manual process, but use of configuration/vulnerability scanning tools can at least help determine when systems fall out of compliance or if changes are implemented outside approved processes.

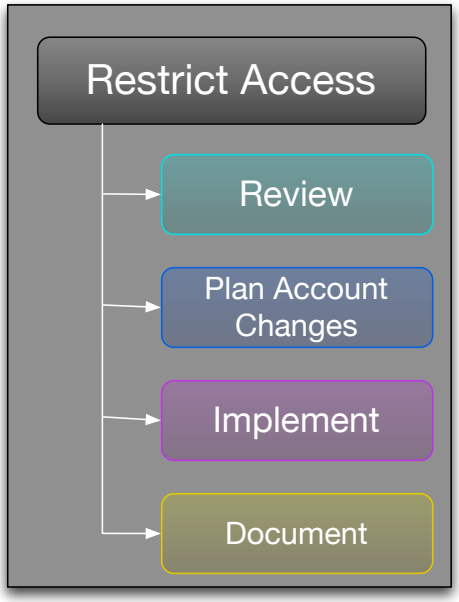
Configure Metrics

Process Step	Variable	Notes
Evaluate	Time to review assessment reports per database	Assessment scans from Discover and Assess phase, etc.
	Time to identify policy/standards violations and incorrect settings	
Plan Changes	Time to prioritize	
	Time to itemize issues	For tracking/change management
	Time to select remediation option	What changes to make and how
	Time to allocate resources, create work order, and create change script as needed	
Test and Approve	Time to create regression test cases and acceptance criteria	How will you verify the change does not break your applications?
	Time to set up test environment	Obtain servers, tools, and software for verification; then set up for testing
	Time to run test	Variable: may require multiple cycles, depending upon test cases
	Time to analyze results	
	Time to create deployment packages/ change script	
Change and Confirm	Time to schedule and notify	Schedule personnel & communicate downtime to users
	Time to install	Taking DB offline, back up, patch database, and restart
	Time to verify	Verify patch installed correctly and database services are available
	Time to clean up	Remove temp files
Document	Time to document updated systems	
	Time to update configuration documentation	If needed, should include any approved variances from standards

Restrict Access

In this phase we adjust access controls and authorizations to meet our security requirements. Setting -- or resetting as the case may be -- database access control and account authorization is a major task. Most of the steps within this phase are self explanatory, but for databases with hundreds to thousands of users, the amount of time spent on review can be significant. We need to see what is in place, compare that against documented polices, and return users and groups to their intended settings. Many users have elevated permissions granted 'temporarily' to get a specific task done with data or database functions outside their normal scope, or due to job function changes, but such permissions are often left in their 'temporary' state rather than being reset when no longer needed or appropriate. This "permissions creep" is a common problem. For permissions put in place to avoid breaking application functionality or still required for certain users to perform temporary tasks, document the variance.

The four steps are:



1. **Review:** Review the database entitlement and authentication requirements. Then determine if the current authentication methods align with those requirements and what changes, if any, need to be made.

2. **Plan Account Changes:** Using the data collected in the Discover and Assess phase, identify any discrepancies in account entitlements and other access control and authorization settings. Then identify any required user/role/group entitlement and permission changes. Also determine whether you need any password changes, especially for service accounts. This will be the most time consuming step, especially if you manage it manually tools are now on the market to help identify entitlement issues within databases.

3. **Implement:** Notify users, then adjust authentication methods, permissions, and other user account settings (including removing or disabling orphan accounts). Pay particular attention to service accounts.

4. **Document:** Document any changes and generate compliance reports (often required when dealing with user accounts).

Additional and Large vs. Small Company Considerations

Digging into database user accounts to identify entitlement issues is one of the most dreaded database security tasks, perhaps second only to installing security updates on legacy systems. It is also increasingly required to support compliance efforts.

This tends to be much more complex in larger organizations due to the number of users and roles involved — especially for ERP and other major internal applications. External-facing systems are often easier to manage, even if there are more users, due to the more limited set of roles.

In a large internal application the key area of focus is user roles, as that is where we tend to see the most problems. Mapping these to users and required entitlements may require use of a tool and/or extensive resources if you have never gone through the process before.

After “permissions creep”, the next most common issue we see is poorly managed service accounts — this problem spans organizations of all sizes. With service accounts we frequently see weak passwords and, especially, static passwords stored in plain text configuration files. This is a common source of audit deficiencies; one way to help manage this problem is to move to multi-factor authentication for these accounts, typically adding in a digital certificate tied to the application server IP address.

But the reality is that this is a tough and time consuming step, which is also one of the most important in the entire database security program.

Restrict Access Metrics

Process Step	Variable	Notes
Review	Time to review users and access control settings	Should have been completed in Review phase
	Time to identify authentication method	
	Time to compare authentication method against policy	Method might be domain, database, mixed mode, etc.
Plan Account Changes	Time to identify user permission changes	
	Time to identify group and role membership adjustments	
	Time to identify changes to password policy	
	Time to identify dormant or obsolete accounts	
Implement	Time to alter authentication settings/methods	Global settings
	Time to reconfigure and remove user accounts	
	Time to implement new groups and roles, and to adjust memberships	
	Time to reconfigure service accounts	Such as generic application and DBA accounts
Document	Time to document changes	
	Time to document accepted configuration variances	
	Time to generate compliance reports	

Shield

Threats against databases and the information stored therein are not always conventional -- SQL injection, for example, is often more a factor of how you've coded your database and application than any particular vendor vulnerability. There will be instances where patches for specific threats are unavailable or security risks are simply inherent to the database features in use. Other exploits leverage weaknesses in database trust relationships, such as Oracle database links, DB2 remote command service, Sybase remote server access, and SQL Server trusted servers. Still others exploit flaws in underlying network security, such as insecure communication or improperly implemented SSL connections. This task within the Secure phase is intended to account for cases where the database is incapable of protecting itself without functional modification or "work arounds", or to deal with new vulnerabilities you can't immediately patch. Or *ever* patch, as some applications are only certified for a legacy database version for which patches aren't available, or patching will break the application or support contract.

We advocate a "Patch and Shield" model to protect the database when patching comes up short. The approach might entail disabling database features, or further refinement to the database configuration. Virtual patching can also be accomplished through firewall, application firewall, or activity monitoring capabilities that block malicious requests. This process is not typically discussed in database vendor recommendations or "best practices", as it directly addresses platform deficiencies and remediation through third party vendors, but is an important step for 0-day protection..

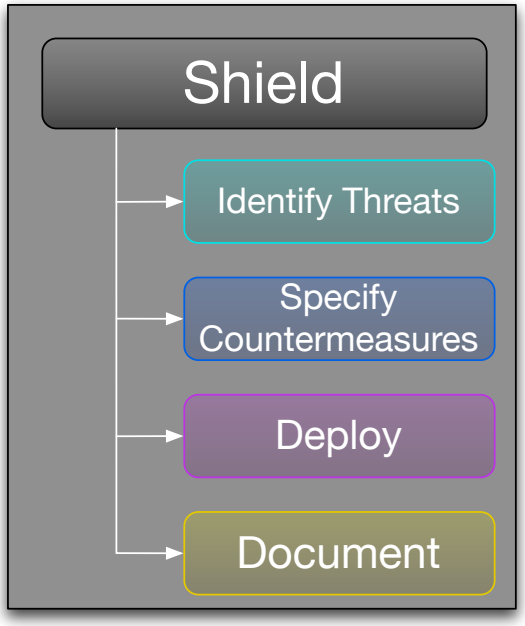
The steps are:

1. **Identify Threats:** Determine the threat (*e.g.,* a new vulnerability or SQL injection in general) and at-risk databases. This entails evaluating the network connections and routes to the database, as well as exploitable trust relationships deeper in the environment.

2. **Specify Countermeasures:** Determine the method to shield the database from the threat — such as a workaround, network filter/segmentation change, or database-specific tool.

3. **Deploy:** Deploy the tool, workaround, or rule/policy change within an existing external security control.

4. **Document:** Document the shielding action and schedule for removal after you update the database to remove the fundamental flaw/vulnerability.



Shield

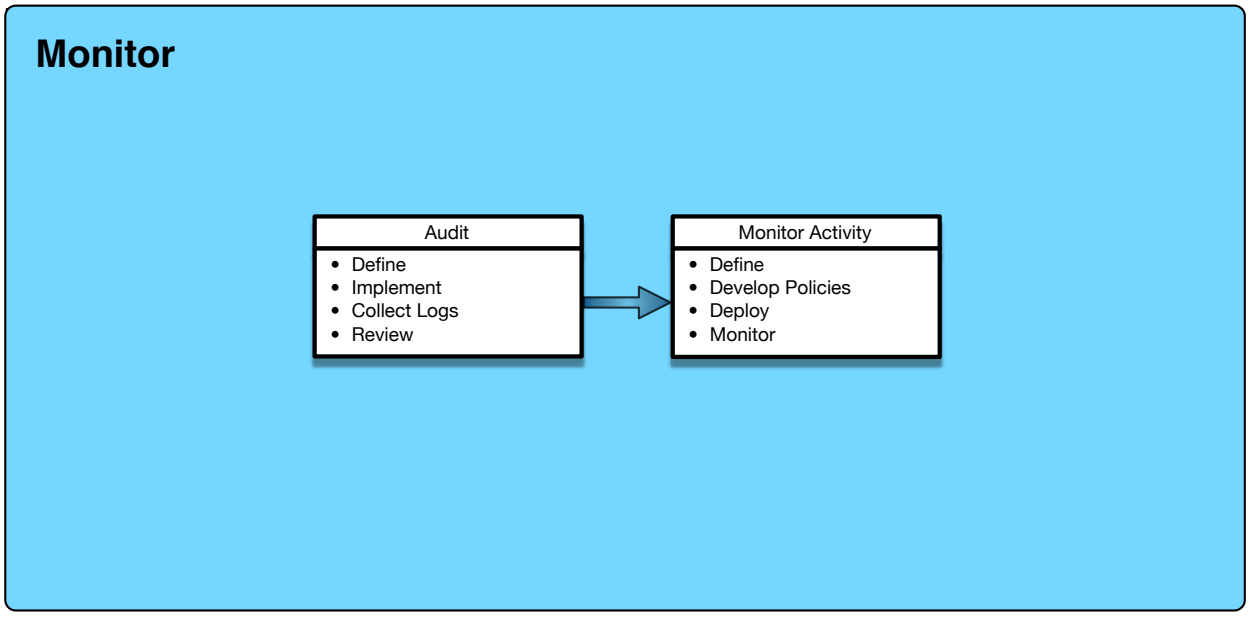
Process Step	Variable	Notes
Identify Threats	Time to identify threat and at-risk databases	
	Time to review ingress/egress points and network protocols	Such as external stored procedures and multiple network connections
	Time to identify exploitable trust relationships	Such as mixed mode authentication
Specify Counter-measures	Time to identify shielding method	Workaround, security tool, security rule/signature, or other external change
	Time to develop specific change	The actual change to support the shielding method — such as network configuration change, security tool rule/policy, or database configuration change.
	Time to test countermeasure	
	Cost of countermeasures	Capital costs (*e.g.,* cost of security tool)
Deploy	Time to deploy countermeasure	
	Time to confirm deployment	
Document	Time to document	Verify DBA permissions are divided across separate roles
	Time to schedule removal of countermeasure	Optional: only if a later patch/fix will eliminate the problem

Monitor Phase

The Monitor phase serves two purposes: to satisfy compliance requirements, and to improve security. While we have always audited our databases to some degree, the Sarbanes-Oxley Act and other regulations dramatically altered both what we need to monitor, and how we perform monitoring. Over the same time period we have also seen massive growth of database security issues such as SQL injection, that can be extremely difficult to manage entirely with preventative controls.

As a result, we have seen improvements in native auditing and increased adoption of Database Activity Monitoring. Native auditing (the auditing functions built into databases) has shown very significant performance improvement. Even 1-2 versions back, enabling anything but the most basic auditing on some database platforms would cripple performance; but most vendors have largely resolved these issues.

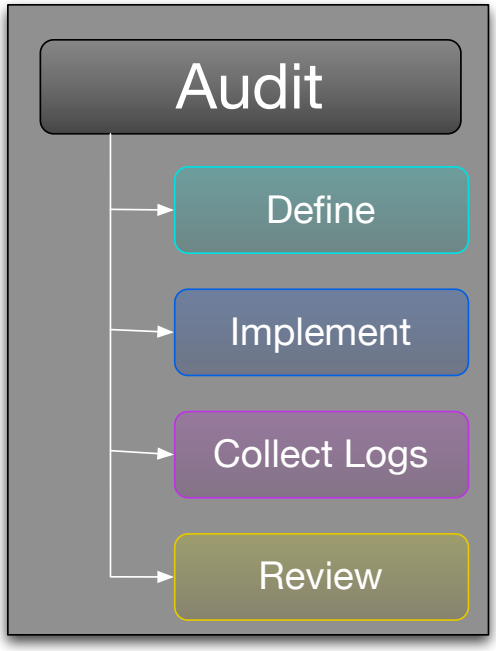
Database Activity Monitoring not only provides much more granular auditing, but low-impact monitoring of legacy systems and real-time security alerts based on centralized policies.



Audit

Database auditing is the examination of audit or transaction logs to track changes to data or database structure. Database auditing is not specifically listed as a requirement of most compliance initiatives, but in practice it fills an essential role by providing an accurate and concise history of business processes, data usage, changes, and administrative tasks -- all necessary for policy enforcement. As such, most audit requirements center on tracking a specific set of users, objects, or data elements within the database. Auditing capabilities are built into all relational database platforms, and most major platforms offer more than one way to collect transaction information. You may choose to supplement native database auditing with external audit data sources, but for the scope of this project we will stick with the more common built-in auditing.

Gathering metrics for database auditing requires first scoping the project to understand which databases need which controls, determining how to configure auditing capabilities to satisfy your requirements, and then periodically collecting the audit trails generated. Day to day management of the audit trails is often an issue, depending upon how many transaction types you track. On high-volume transaction severs the data files grow quickly, requiring archival of the audit files so data is not lost, and configuring the database to truncate logs if necessary to avoid filling disk drives to capacity.



Auditing includes four steps:

1. **Define:** Define which databases require auditing, and which activities on those databases to audit. Determine additional requirements, such as how audit logs will be stored and secured, and who has access.
2. **Implement:** Enable auditing on the databases. Integrate with any log management, SIEM, or external storage tools.
3. **Collect Logs:** Collect the logs, which may be a manual or automated process. Clean up log files after collection.
4. **Review:** Review the log files for security issues. Generate any required compliance reports.

Large vs. Small Company Considerations

It is very common to manage auditing manually in organizations of all sizes — particularly at mid-sized and smaller companies — despite the availability of inexpensive and free log collection tools. Logs should never be stored on the same system as the database, and tools help with collection, the analysis, and reporting.

Successful large companies tend to use a mix of Database Activity Monitoring (which we will discuss in the next section) and log management tools. This is definitely an area where tools result in cost savings compared to manual processes, and when compliance is involved, a tool becomes pretty much mandatory.

Audit Metrics

Process Step	Variable	Notes
Define	Time to identify databases	
	Time to determine auditing requirements	Partially completed in Plan phase
	Time to identify users, objects, and transactions to audit	
	Time to specify filtering	
	Time to determine collection/review requirements	
	Cost of storage and/or log management tool	
Implement	Time to set up and configure auditing	
	Time to integrate with existing systems	*e.g.,* SIEM, log management
Collect Logs	Time to collect and/or migrate logs	
		Remove from primary storage
Review	Time to review logs for policy violations and security anomalies	
	Time to clean up logs	

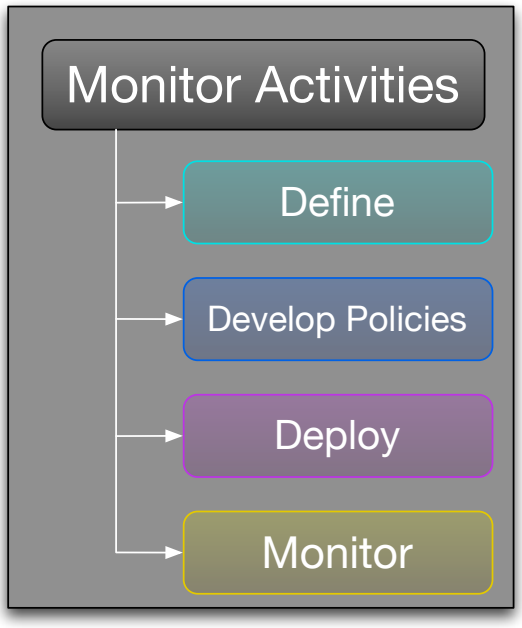
Monitor Activity

Database monitoring is distinctly different from auditing: it provides near-real-time detection, heterogeneous database support, aggregation and correlation, and secure event storage; it also offers more forms of event collection than audit and transaction log files. Securosis has our own definition of Database Activity Monitoring. Databases do not have monitoring built in, so this function is provided through other products — typically from third parties.

The two primary use cases are security and compliance. The policies to support each will be different; each option suggests different methods of data collection, and require integration with different applications used by different stakeholders in the security process. The first step is to identify your goals and outline how the product is to be used. Later you will move on to the selection of a product, development of policies to enforce, and final deployment and integration. In this phase we are only covering the monitoring of systems and alert generation.

Monitoring consists of four steps:

1. **Define:** Determine which databases require monitoring, security requirements, and which activities to monitor. If a tool is not currently in use, select one.
2. **Develop Policies:** Determine the rule set for the monitored database, including which activities to monitor and which security policies will generate alerts.
3. **Deploy:** Deploy the monitoring tool and any required collection agents. Configure monitoring according to the requirements and policies developed in steps 1 and 2.
4. **Monitor:** Monitor the database, collect activity, and manage policy violations and incidents. Generate any required compliance and security reports.



Additional Considerations

Database Activity Monitoring is, by definition, something provide by party tools outside the database, generally from third parties, although some database vendors offer DAM products for their databases.

Most organizations tend to focus their DAM deployments on high-value and regulated databases, then slowly expand coverage.

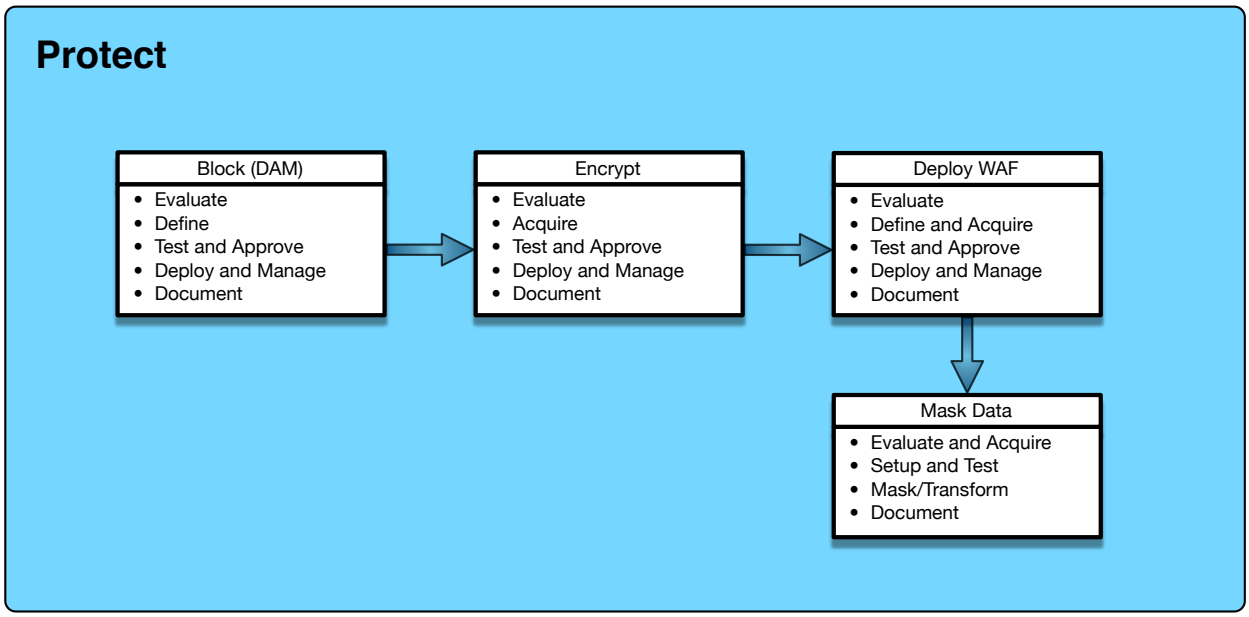
Monitor Activity Metrics

Process Step	Variable	Notes
Define	Cost of DAM tool	
	Time to identify and profile monitored database	Identify the database to monitor and its configuration (*e.g.,* DBMS, platform, connection methods)
Develop Policies	Time to define security rules	
	Time to define monitoring policies	
Deploy	Time to deploy DAM tool	Optional- may not be required after the initial deployment
	Time to deploy and configure collection agent	
	Time to test deployment	
	Time to record configuration changes	Some collection agents require system configuration changes within the database
	Time to deploy/enable policies	
Monitor	Time to monitor for alerts	
	Time to review activity	If manual review is used
	Time to generate compliance reports	

Protect Phase

The purpose of the Protect phase is to implement active security controls to block attacks, protect stored data, and scrub/obfuscate production data for use in testing environments. Until now we have focused on finding and configuring databases to reduce security exposure, managing users, and monitoring (for unusual activity/abuse and to support compliance). Those processes create a secure baseline for our systems, and help find and fix problems.

In this phase we start implementing active security controls that change the functioning of the database and can interfere with business process if not implemented properly. These controls are more active in nature than merely securely configuring a system and managing users, and are designed to directly stop attacks.

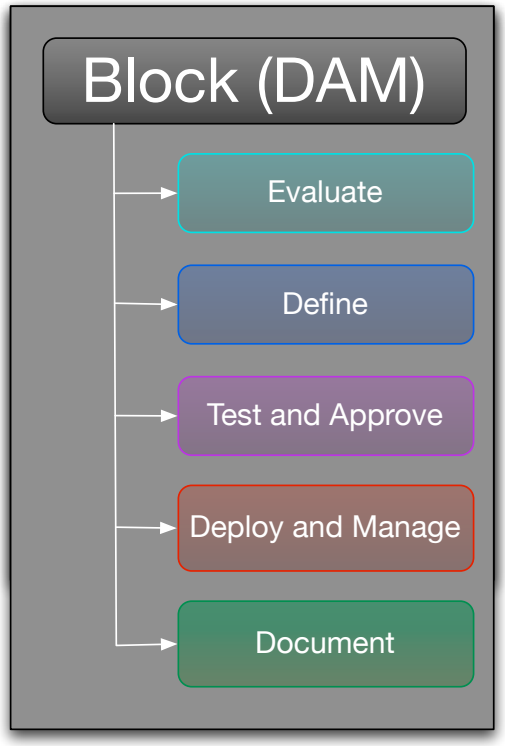


Of these controls, we most commonly see encryption, then masking, then Web Application Firewalls. Very few users are deploying DAM in active blocking mode, although many of the products support it and we're seeing growing interest — largely to deal with patching issues.

Block (DAM)

By now some of you have deployed DAM to help with the Monitor phase of the program. In monitoring mode, Database Activity Monitoring platforms are deployed "out of band", collecting activity and generating alerts as a third party observer. But DAM can also be used, like a firewall, to block dubious queries in real time and enforce proper database use. The various tools function differently, but in general they bridge or proxy SQL traffic and drop or block queries that violate policy. Some tools are unidirectional (inbound requests only) while others are bidirectional or fully integrated within the database via an agent and can even detect problems such as 'bad' stored procedures.

The capabilities of the various products vary greatly, and like other security tools generally offer options to deploy in a white list (only allow approved queries) or black list (block bad queries based on signatures) mode.



The steps in DAM blocking are:

1. **Evaluate:** Determine which activities to block for which databases, and whether any architectural or platform modifications are required.
2. **Define:** Create rules and policies for blocking. Determine the deployment mode, perform analysis to generate the rules, and select the blocking technique.
3. **Test and Approve:** Test policies in monitoring mode and approve them for deployment.
4. **Deploy and Manage:** Deploy the DAM tool if necessary and configure the database for it (*e.g.,* install an agent or set up a proxy). Enable policies/rules in blocking mode and monitor results. Manage ongoing incidents and database changes, and tune policies as needed over time.
5. **Document:** Document the policies and any changes to the database and DAM tool. Generate ongoing reports on activities and incidents to evaluate effectiveness.

Additional Considerations

Logically it might make sense to include blocking under the Monitor phase because the same tool is used, but we do it this way because blocked events are critical items with a different review process. It's much easier to account for the time and resources by splitting blocking into a separate task from passive activity monitoring. The sequence of events is pretty straightforward: you will have something specific to address, such as *ad hoc* database connections or SQL injection. Identify the databases, create the policy that describes the goal, and then specify the DAM rule or rules that perform the work. DAM tools often provide a level of abstraction so you can set the pre-defined policy, and the rules to form that policy are implemented on your behalf.

- Blocking is a more advanced DAM feature that can have serious side effects, and is typically employed only after monitoring policies are successfully in place.
- Policies are typically based on information discovered through monitoring.
- Blocking rules are commonly predicated on comparison to a known behavioral profile, with the profile built over time from monitoring activity.
- Blocking warrants more carefully crafted rules to enforce business policies; and on a more practical level additional routine maintenance; as application queries, database structures, and use cases evolve.

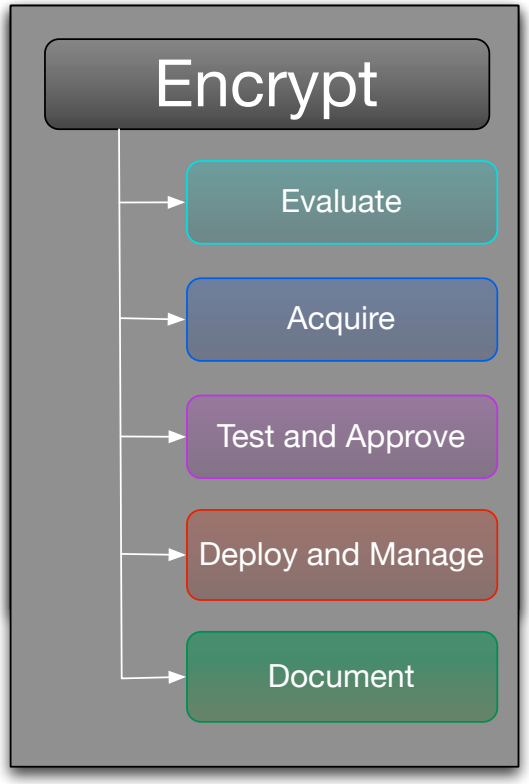
Block (DAM) Metrics

Process Step	Variable	Notes
Evaluate	Time to identify databases	
	Time to identify activity to block	Partially addressed in the Plan phase
	Cost of DAM tool for blocking	May already be accounted for
	Time to determine database changes needed to support DAM integration	
Define	Time to select blocking method	
	Time to create rules and policies	
	Time to specify incident handling and review	
Test and Approve	Time to configure and test rules	Rules are typically deployed in monitoring mode and the results evaluated.
Deploy and Manage	Time to deploy DAM and/or integrate database	
	Time to deploy rules	
	Time to manage incidents	
	Time to tune policies	Based on effectiveness/issues as well as changes to the database that could affect enforcement
Document	Time to document rules/policies	
	Time to generate ongoing reports	

Encrypt

Several forms of encryption are available for databases. Each is different in its level of security, ease of deployment, cost, and performance impact on transaction processing -- all of which make the selection process difficult. Furthermore, security and compliance requirements pertaining to encryption are often murky. They key to this process is understanding the requirements and mapping them to available technologies.

Pay close attention to operations and integration efforts to ensure no hidden obstacles are discovered *after* deployment. Examples include finding that tape archiving no longer works, or that user account recovery fails to recover encrypted data. This type of thing is common, so we've included it in the process.



1. **Evaluate:** Determine encryption requirements and identify preferred option. This means determining which information needs to be encrypted by which general method (*e.g.,* transparent vs. column-level vs. application), external requirements (such as backups and batch jobs), supported encryption options for the platform, and key management and reporting requirements.
2. **Acquire:** Evaluate the encryption tools or native features for deployment, select, and acquire.
3. **Test and Approve:** Establish a test environment and test the implementation plan, as well as performance and dependencies.
4. **Deploy and Manage:** Implement encryption on the production systems. Manage access, keys (including rotation, if needed), and requirements changes.
5. **Document:** Document and generate ongoing reports (often needed for compliance).

Additional Considerations

Database encryption can be as simple as changing a few configuration settings, or as complex as completely redesigning the database and all connected applications. Once in place, encryption doesn't tend to be overly time consuming, but depending on the design and age of your database the initial implementation can be incredibly intensive.

In terms of cost, the design of the existing database to encrypt is the primary factor — more than the size of the data or the platforms involved. We recommend transparent encryption in most cases, as it is easy to deploy and secures stored data.

Encrypt Metrics

Process Step	Variable	Notes
Evaluate	Time to confirm data security requirements	
	Time to identify encryption method and tools	*e.g.,* native database, OS
	Time to identify integration requirements and dependencies	*e.g.,* external key management, key rotation, disaster recovery considerations
Acquire	Time to evaluate encryption tools	
	Cost to acquire encryption products	
	Optional: Cost to acquire key management system	
	Variable: Cost of maintenance and support licenses	Native transparent encryption cost is likely to be additional to base database license
Test and Approve	Time to establish test environment	
	Time to install and configure encryption tool	Test environment configuration
	Time to test	Verify data is encrypted, backup procedures still work, etc.
	Time to establish disaster recovery process and procedures	Keys and supporting services need to be accounted for, etc.
	Time to collect sign-offs and approval	Verify efficacy of encryption, and that systems pass test cases
	Time to create database archive	Archive & verify production backup
Deploy and Manage	Time to install encryption in production environment	
	Time to install key management server (if used) and generate keys	Keys must to be generated regardless
	Time to deploy, encrypt data, and set authorization rights	

Process Step	Variable	Notes
	Time to integrate with applications, backup, and authentication systems	
	Ongoing time to manage keys (rotation, generation) and access changes	
Document	Time to document updated systems	
	Time to generate ongoing security and compliance reports	

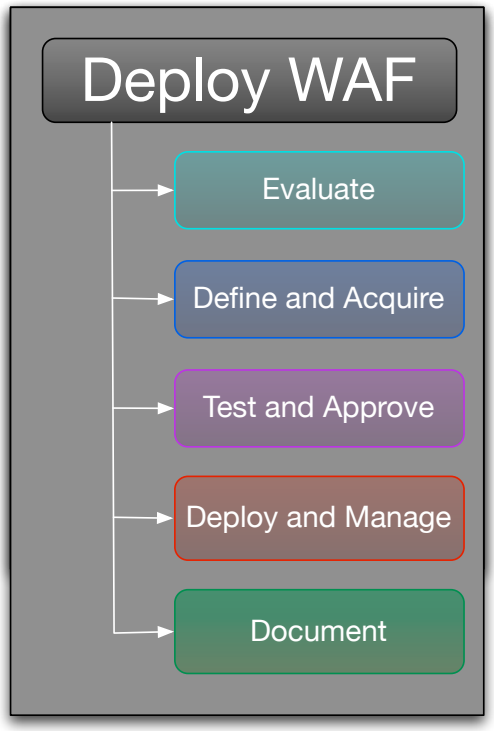
Deploy WAF

Deploying a WAF does not fall on the shoulders of database administrators. And it's not really something one normally thinks of when itemizing database security tasks, but that is beginning to change. With database support for web and mobile applications, and PCI requirements overshadowing most commerce sites, WAF is an important consideration for guarding the symbiotic amalgamation of web and database applications.

At this phase of the program we are not fully fleshing out a process for WAF deployment, but picking those tasks where database administrative experience is relevant. For some of you this step will be entirely optional. Others will be working with security and application developers to refine rule sets based on query and parameter profiles, and verifying transactional consistency where blocking is employed.

The steps are:

1. **Evaluate:** Identify the database to protect, the applications above it, and the security requirements.
2. **Define and Acquire:** Define protection requirements, including specific rules to deploy. Acquire a WAF or access to an existing WAF to implement the rules with.
3. **Test and Approve:** Test the rules in monitoring mode and approve for deployment. Not that testing will be performed by whoever manages the WAF, rather than the database security manager, with approval required for both teams.
4. **Deploy and Manage:** While the WAF team/manager is responsible for deploying the rules, the database team will monitor the database for any performance, functional, or security issues that require WAF configuration or rule changes.
5. **Document:** Document the rules.



Additional and Large vs. Small Company Considerations

Companies of all sizes deploy WAFs, but *who manages the WAF* varies a great deal depending on industry, company size, and internal organizational structure. In a smaller company all web application and database security might be managed by a single person, while this is often distributed across teams in larger companies. For of database security metrics you might limit time and cost estimates to only those handled on the database side, because much of the cost of a WAF is borne by other teams.

But there's nothing wrong with lumping everything together if the primary driver for the WAF is the database. How deeply you measure these metrics depends on your project goals, but don't feel obliged to include all WAF costs in your database program.

Deploy WAF Metrics

Process Step	Variable	Notes
Evaluate	Identify database to be protected by WAF and the involved applications	
	Determine database-level security requirements	Only those where the WAF can help, such as SQL injection
Define and Acquire	Time to gather application query and parameter profiles	For example, what does the web application send to the database? Provide to the WAF team to generate rules/policies.
	Time to define WAF rules for database protection.	
	Time for WAF team to create and approve rules	
	Optional: Time to acquire and deploy WAF	
Test and Approve	Time to establish test environment or deploy rules in monitoring mode	
	Time to evaluate results	
	Time to approve for deployment	Will involve both WAF and database teams
Deploy and Manage	Deploy rules to WAF	In blocking mode
	Time to manage incidents and alerts	
	Time to adjust rules for database/ application changes or performance issues	
Document	Time to document policies and deployment	
	Time to generate ongoing security and compliance reports	

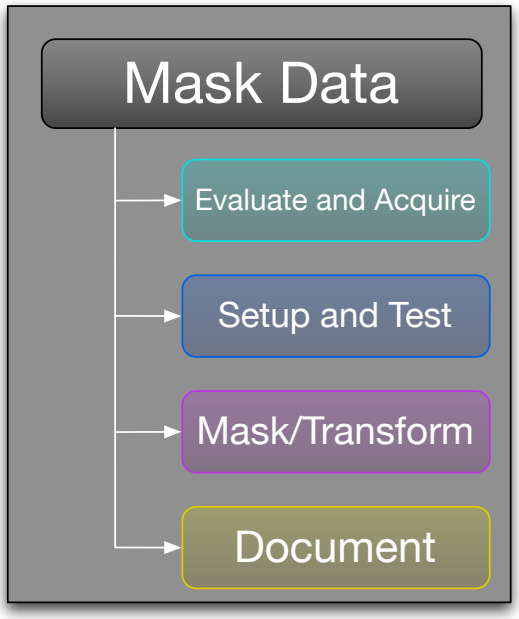
Mask Data

The last step in the Protect phase is to transform production data for use in test environments to limit risk exposure. In a nutshell, masking is applying a function to data in order to obfuscate sensitive information, while retaining its usefulness for reporting or testing. Common forms of masking include randomly re-assigning first and last names, and creating fake credit card and Social Security numbers. The new values retain the format expected by applications, but are not sensitive in case the database is compromised.

Masking has evolved into two different models: the traditional Extraction, Transformation, Load (ETL) model, which alters copies of the data; and the dynamic model, which masks data in place. The conventional ETL functions are used to extract real data and provide an obfuscated derivative to be loaded into test and analytics systems. Dynamic masking is newer, and available in two variants. The first overwrites the sensitive values in place, and the second variant provides a new database 'View'. With views, authorized users may access either the original or obfuscated data, while regular users always see the masked version. Which masking model to use is generally determined by security and compliance requirements.

The steps are:

1. **Evaluate and Acquire:** Determine masking requirements, evaluate tools and manual processes, and acquire a tool or the resources for manual masking.
2. **Setup and Test:** Create the masking environment, define masking rules, and test with samples of production data.
3. **Mask/Transform:** Convert the data and provide to the requestor. Repeat as needed to maintain consistency between the two environments.
4. **Document:** Document the masking rules and details to support future transformations.



Additional Considerations

Masking is typically an ongoing process used to support developers. Whether or not you use tools, you should prepare the process to be repeated as updated snapshots of the production environment are needed by developers. Masking is sometimes seen as a one-off project, but if handled that way the two environments end up diverging so much that the masked data is no longer representative of production.

One trick to help track down logic issues is to create fictional test cases that are not masked/transformed between production and development. That allows developers to test the application logic in both environments and directly compare results, which is otherwise not possible using fully masked data.

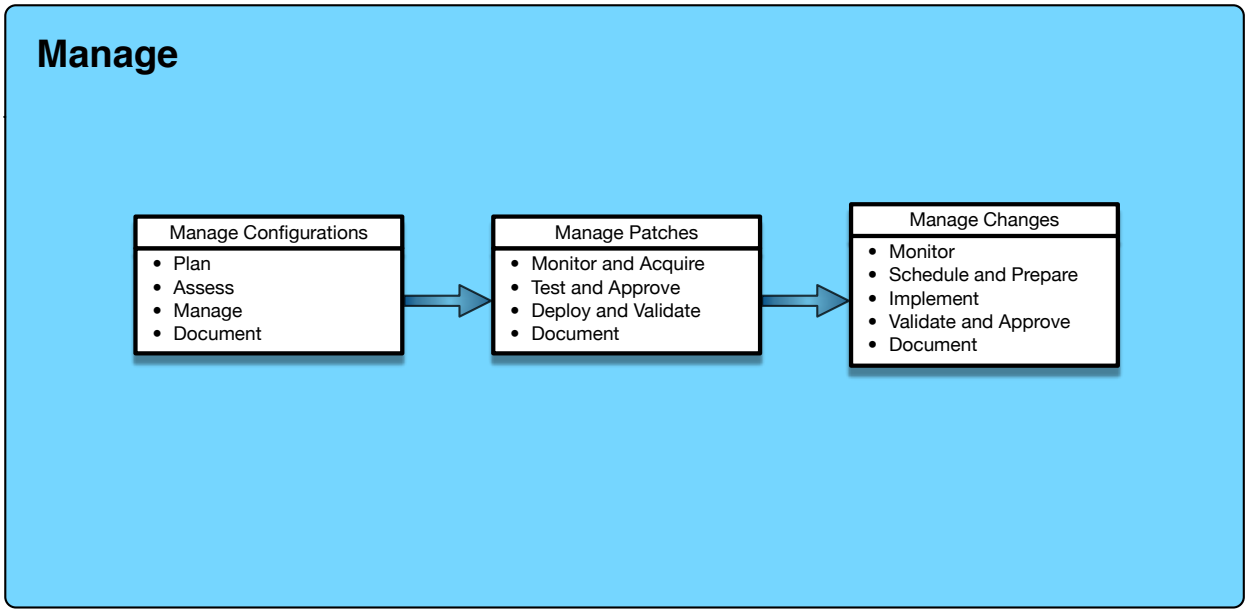
Mask

Process Step	Variable	Notes
Evaluate and Acquire	Time to confirm data security and compliance requirements	
	Time to identify preservation requirements	*e.g.,* last 4 digits of credit card, format, data type
	Time to specify masking model	*e.g.,* ETL, in place, etc.
	Time to generate baseline	*e.g.,* gather sample data and formats for testing
	Time to evaluate masking products	
	Cost to acquire masking products/packages	
	Time to acquire access and authorization to data systems	Credentials to implement masking on sensitive data
Setup and Test	Time to identify integration requirements	*e.g.,* authorization, ETL, disaster recovery considerations, etc.
	Time to install masking tool	
	Time to create masking/transformation plan and configuration	Specific process for masking, which is often created in the masking tool
	Time to test masking and adjust plan	Mask the production data, then review for issues and adjust the plan or configuration accordingly
Mask/Transform	Time to mask	i.e. ETL.
	Time to confirm masking and distribute resulting database to requestors	
Document	Time to document masking plan	

Manage Phase

Many of the databases you're dealing with throughout this process probably started in a reasonably manageable, or even secure, state. But time is the great killer of security: configurations drift; entitlements expand; and all the little tweaks to keep a system running slowly degrade security, and often performance.

The Manage phase is an ongoing process used to keep systems compliant with policy. These three sub-phases help us keep our systems configured securely and up to date, and allow us to track changes ranging from settings to user accounts.



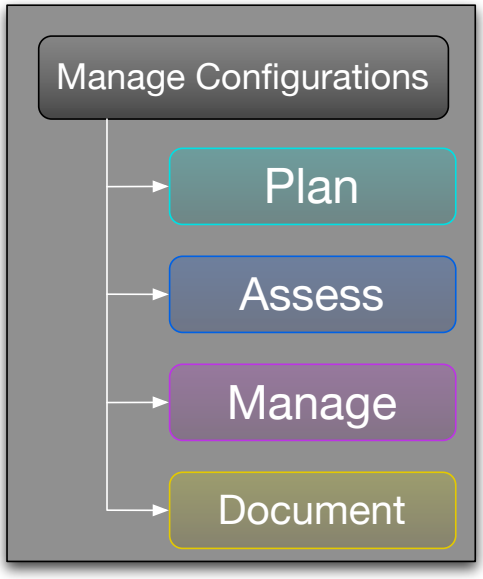
In some ways this phase is the most important in the entire program. It's the only way to keep control over databases and minimize the inevitable drift over time — we're not naive enough to think any process is perfect.

The good news is that we have seen plenty of organizations successfully implement these management tasks on at least some of their databases — typically the most important ones — and that unlike some other security tasks, these come with clear performance and stability benefits and are well understood by database administrators.

Manage Configurations

In the Plan phase we determined our configuration standards, in the Discover and Assess phase we determined which systems were in and out of compliance, and in the Secure phase we fixed non-compliant systems.

All those were essentially one-time processes to realign our resources. Now it's time to convert them into an ongoing, repeatable process. All these steps overlap with previous phases, but the focus is on managing configurations over time and keeping them compliant, as opposed to bringing them into compliance.



The steps are:

1. **Plan:** Determine which systems are part of the configuration management program and map them to configuration standards. As configuration standards change, determine which systems are affected. If a system needs a change that violates the existing standard, evaluate and approve (or deny) the request.
2. **Assess:** Evaluate systems to see which are still in compliance. This should be done on a rolling, scheduled basis to detect configuration drift.
3. **Manage:** Create a remediation plan and implement changes. Verify that changes were implemented successfully.
4. **Document:** Document any database updates, as well as changes to configuration standards.

Large vs. Small Company Considerations

If you are part of a small IT organization, this is a pretty straightforward process. If your work as part of a larger enterprise team, you'll have stakeholders in database administration, audit, IT operations, and security; which makes information collection, distribution, and record-keeping far more complicated.

The most important consideration in configuration management is also the most obvious; get systems compliant and keep them there. This involves a cultural change in many organizations, but current configuration management tools do a much better job of monitoring databases than even a few years ago.

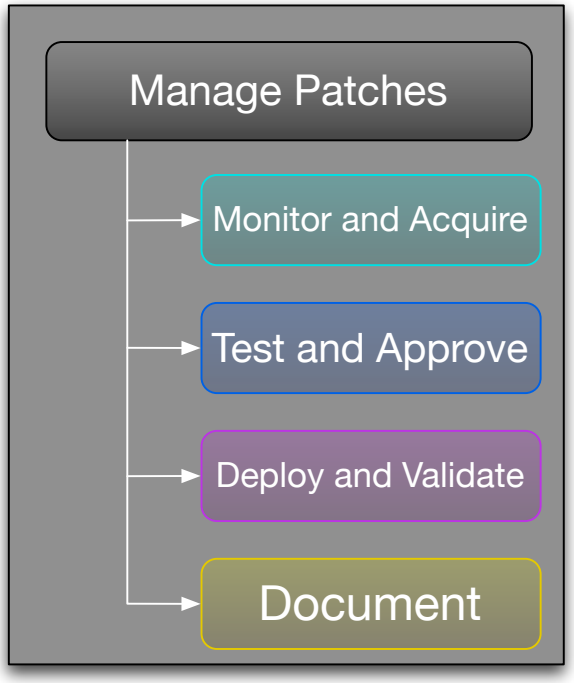
Manage Configurations Metrics

Process Step	Variable	Notes
Plan	Time to identify databases	Should come from Discover and Assess phase
	Time to determine appropriate configuration standard	Should come from Secure phase
	Time to assess changes to standards, identify affected systems	Ongoing process when standards change
Assess	Time to assess/scan configurations	Manual or automated, on a scheduled basis
	Time to determine required changes	What settings, review potential side-effects
Manage	Time to update configuration	
	Time to verify configuration change succeeded	
Document	Time to document configuration change	What changed, and variations to standard
	Time to update configuration standards	If needed
	Time to generate required compliance reports	

Manage Patches

Database Patch Management is still shockingly difficult. Due to all the application dependencies common in enterprise databases, even simple patches can break core functions. Not all database platforms offer reliable installers for patches, resulting in complex processes prone to failure. Even finding a maintenance window to install a patch can be a political battle. Despite all that, keeping systems up to date is still one of the most important tasks for ensuring database security.

At times you may have workarounds, or be able to mask flaws with third party security products, but the vendor is the only way to really 'fix' fundamental database security issues. That means you will be patching on a regular basis to address 0-day attacks, just as you do with 'Priority 1' functional issues. Database vendors have security teams dedicated to analyzing attacks against their databases, and small firms must leverage their expertise. But you still need to manage the updates in a predictable fashion that does not disrupt business functions.



1. **Monitor and Acquire:** Monitor vendor feeds/sources for patch releases. When a patch is released, determine which systems in your environment are affected and acquire the patch. This will likely include capital costs for maintenance contracts, as some database vendors (such as Oracle) only release security updates to clients with current contracts.
2. **Test and Approve:** Create a test environment and test the patch and installation process. Determine whether the patch will result in issues for dependent systems. Once ready, approve the patch for distribution and installation.
3. **Deploy and Validate:** Install the patch on required systems. verify that the patch installed correctly and dependent systems are functioning properly.
4. **Document:** Document and generate ongoing reports (often required for compliance).

Additional Considerations

Database security patching is often a political battle between the conflicting interests of DBAs (keep it running if “nothing is broken”) and security (update vulnerable systems). Many users we talk with patch on an annual basis, and even the best often patch only quarterly. The complexity is compounded by the poor patch installers offered by some vendors. We have heard many reports of installers reporting success even though the patch didn’t actually apply properly.

Some vendors follow regular release schedules and provide pre-notification for impending patches. We suggest you use these to plan maintenance windows, or at least evaluate the patches to determine your risk exposure and any shielding or workarounds.

Configuration and vulnerability management tools can be very helpful in evaluating successful patch installation as the better ones don't rely on self reporting (such as version numbers), but look deeper at the system and individual components.

Manage Patches Metrics

Process Step	Variable	Notes
Monitor and Acquire	Time to monitor for advisories	
	Time to determine systems affected by patch release	
	Time to acquire patch	
	Capital costs of maintenance/support license	
Test and Approve	Time to create test environment	
	Cost of test environment	
	Time to test and evaluate	
	Time to generate installation script and approve	
Deploy and Validate	Time to install patch	
	Time to validate installation	
	Time to clean up patch installation	Remove residual components and reset system to functional state — *e.g.,* remove installer and release from maintenance mode
Document	Time to document updated systems	
	Time to generate ongoing security and compliance reports	

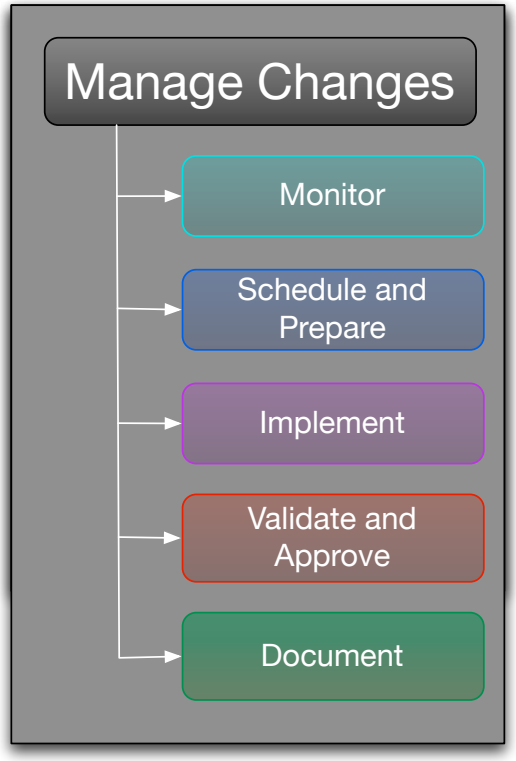
Manage Changes

Change management is the lynchpin of keeping control over our systems. With databases a change could be something as simple as a one-line SQL query, where other systems or applications would require code changes and installers. Knowing how your systems change over time, and allowing only approved changes, is absolutely critical.

The good news is this is one area where security, database, and application teams tend to agree. Most organizations experience the pain of downtime due to unapproved or untracked changes, so it isn't very hard to get a change management program started.

The steps are:

1. **Monitor:** Gather change requests and map to affected databases.
2. **Schedule and Prepare:** Schedule the change, prepare documentation and change scripts, and assign in your change management system. If necessary, test.
3. **Implement:** Apply the change.
4. **Validate and Approve:** Confirm that the change implemented correctly and approve that it is complete.
5. **Document:** Close the ticket in your change management system and document the change.



Additional Considerations

One of the toughest problems in database change management is monitoring for unapproved changes. One option is to combine DAM (or auditing) with your configuration management system. Monitor all changes by administrators and require them to set a session variable matching the change management ticket number. You can then scan for these to find unapproved activity and to track changes back to approved tickets.

Manage Changes Metrics

Process Step	Variable	Notes
Monitor	Time to monitor for change requests	
	Time to determine affected databases	
Schedule and Prepare	Time to schedule the change	*i.e.,* What does the web application send to the database? Provide to the WAF team to generate rules/policies.
	Time to prepare	Including creation of change & validation scripts
	Time to test	
	Time to approve change for implementation	
Implement	Time to implement change	
	Time to clean up	
Validate and Approve	Time to validate change occurred properly	
	Time to approve that change is complete	
Document	Time to close in change management system	
	Time to generate documentation	
	Time to generate ongoing security and compliance reports	

Conclusion

Old Divisions, New Metrics

Throughout our research for Database Quant we consistently struggled with the historical disconnect between security and database management. The authors of this report have both worked across the aisle in their careers; serving as both database administrators and security professionals and experienced this disconnect first-hand. In most organizations DBAs manage everything involved with databases, and security is rarely involved. Also, few security professionals posses enough database experience to handle anything other than the highest-level issues. While ongoing compliance needs are eroding these walls, they are far from crumbled.

While it wasn't out initial goal when we started this project, we realized that this document may be more valuable for laying out a clear set of database security processes than for the specific metrics. We feel it bridges the divide between the two professions by focusing on key areas and using terminology familiar to both. Since the primary Securosis audience is security professionals, we hope they can use this as a tool to open a dialog with DBAs and to help define their program requirements.

And while the metrics are not as detailed as some of our other Quant projects, they still provide a solid baseline to measure the efficiency of your program.

The one potential weakness in this report is that we had to rely extensively on our own experiences. While we did receive public feedback and test our content with various security and database professionals, the number of people that understand both worlds is still fairly small and significantly limited our potential sources for feedback. Thus we plan to continue to update this document as we receive ongoing feedback and processes and tools change.

Where to Start?

There are many operational steps and associated metrics in this project. We recommend organizations start small, and likely focus on areas involved in compliance. Database discovery, assessment (especially configuration and vulnerability), auditing/monitoring, and encryption tend to be the top compliance concerns and are likely activities you are already involved with. Another common area is assessing and managing user entitlements. All of these also happen to be areas where security and database teams tend to have to work together.

The steps to introduce this approach to your organization are pretty straightforward and very replicable.

1. Pick a place to start.
2. Map the process.
3. Choose the metrics.
4. Collect the data.

5. Analyze the data.
6. Adapt the process.

Then go back to Step 1, with another subset of your database security operational processes. We don't mean to oversimplify things, but it's not hard. Your organization just needs the commitment to systematically collect data and adapt the processes based on what the data tells you.

Finally, the authors of this report would like to encourage additional open, independent, community research and analysis projects in IT and security metrics. Utilizing a transparent research process enables new kinds of collaboration capable of producing unbiased results. We are investigating other opportunities to promote open research and analysis, particularly in the areas of metrics, frameworks, and benchmarks. If you have any suggestions as to additional research opportunities, feel free to drop us a line at info@securosis.com.

About the Analysts

Rich Mogull, Analyst/CEO

Rich has twenty years of experience in information security, physical security, and risk management. He specializes in data security, application security, emerging security technologies, and security management. Prior to founding Securosis, Rich was a Research Vice President at Gartner on the security team. Prior to his seven years at Gartner, Rich worked as an independent consultant, web application developer, software development manager at the University of Colorado, and systems and network administrator. Rich is the Security Editor of *TidBITS*, a monthly columnist for *Dark Reading*, and a frequent contributor to publications ranging from *Information Security Magazine* to *Macworld*. He is a frequent industry speaker at events including the RSA Security Conference and DefCon, and has spoken on every continent except Antarctica (where he's happy to speak for free — assuming travel is covered).

Prior to his technology career, Rich also worked as a security director for major events such as football games and concerts. He was a bouncer at the age of 19, weighing about 135 lbs (wet). Rich has worked or volunteered as a paramedic, firefighter, and ski patroller at a major resort (on a snowboard); and spent over a decade with Rocky Mountain Rescue. He currently serves as a responder on a federal disaster medicine and terrorism response team, where he mostly drives a truck and lifts heavy objects. He has a black belt, but does not play golf. Rich can be reached at rmogull (at) securosis (dot) com.

Adrian Lane, Analyst and CTO

Adrian Lane is a Senior Security Strategist with 22 years of industry experience, bringing over a decade of C-level executive expertise to the Securosis team. Mr. Lane specializes in database architecture and data security. With extensive experience as a member of the vendor community (including positions at Ingres and Oracle), in addition to time as an IT customer in the CIO role, Adrian brings a business-oriented perspective to security implementations. Prior to joining Securosis, Adrian was CTO at database security firm IPLocks, Vice President of Engineering at Touchpoint, and CTO of the security and digital rights management firm Transactor/Brodia. Adrian also blogs for Dark Reading and is a regular contributor to Information Security Magazine. Mr. Lane is a Computer Science graduate of the University of California at Berkeley with post-graduate work in operating systems at Stanford University. Adrian can be reached at alane (at) securosis (dot) com.

About Securosis

Securosis, L.L.C. is an independent research and analysis firm dedicated to thought leadership, objectivity, and transparency. Our analysts have all held executive level positions and are dedicated to providing high-value, pragmatic advisory services.

Our services include:

- *Primary research publishing:* We currently release the vast majority of our research for free through our blog, and archive it in our Research Library. Most of these research documents can be sponsored for distribution on an annual basis. All published materials and presentations meet our strict objectivity requirements, and follow our Totally Transparent Research policy.
- *Research products and strategic advisory services for end users:* Securosis will be introducing a line of research products and inquiry-based subscription services designed to assist end user organizations in accelerating project and program success. Additional advisory projects are also available, including product selection assistance, technology and architecture strategy, education, security management evaluations, and risk assessment.
- *Retainer services for vendors:* Although we will accept briefings from anyone, some vendors opt for a tighter, ongoing relationship. We offer a number of flexible retainer packages. Example services available as part of a retainer package include market and product analysis and strategy, technology guidance, product evaluations, and merger and acquisition assessments. Even with paid clients, we maintain our strict objectivity and confidentiality requirements. More information on our retainer services (PDF) is available.
- *External speaking and editorial:* Securosis analysts frequently speak at industry events, give online presentations, and write and/or speak for a variety of publications and media.
- *Other expert services:* Securosis analysts are available for other services as well, including Strategic Advisory Days, Strategy Consulting engagements, and Investor Services. These services tend to be customized to meet a client's specific requirements.

Our clients range from stealth startups to some of the best known technology vendors and end users. They include large financial institutions, institutional investors, mid-sized enterprises, and major security vendors.

Additionally, Securosis partners with security testing labs to provide unique product evaluations that combine in-depth technical analysis with high-level product, architecture, and market analysis.