Secure Database Credential Handling Policy

1.0 Purpose

This policy establishes the mandatory requirements for securely storing, retrieving, and managing database authentication credentials (usernames and passwords) used by software applications connecting to the organization's databases. Improper handling of these credentials poses a significant security risk, potentially leading to unauthorized database access, compromise of sensitive data, and broader system compromise. The purpose is to prevent credential exposure within source code, configuration files, or insecure storage locations.

2.0 Scope

This policy applies to all system implementers, software engineers, developers, administrators, and any personnel involved in the design, development, deployment, or maintenance of software applications (including programs, modules, libraries, scripts, or APIs) that access production databases operating on the organization's networks. While primarily focused on production environments, applying these principles to non-production and lab environments is strongly recommended due to the potential presence of sensitive data.

3.0 Policy Statements

Applications accessing organization databases must authenticate using credentials handled according to the following requirements:

3.1 Credential Storage Prohibitions

* Database credentials (usernames and passwords) **must not** be stored in clear text within the main executing body of an application's source code.
* Credentials **must not** be stored in files or locations directly accessible by a web server (e.g., within the web server's document root or browseable directories).
* Credentials **must not** be embedded directly within compiled application binaries where they could be easily extracted.

3.2 Approved Credential Storage Methods

Acceptable methods for storing database credentials include, but are not limited to:

* **Secure Configuration Files:** Storing credentials in a separate configuration file outside the application's source code and web-accessible directories. This file must have restricted file system

permissions (not world-readable or world-writeable) limiting access strictly to the application's service account or authorized processes. The credentials within this file should ideally be encrypted or protected using operating system mechanisms.
*   **Secrets Management Systems:** Utilizing dedicated secrets management solutions (e.g., HashiCorp Vault, Azure Key Vault, AWS Secrets Manager) designed for secure storage, retrieval, and rotation of credentials and secrets. This is the preferred method.
*   **Environment Variables:** Passing credentials via secure environment variables accessible only to the application process, configured through secure deployment mechanisms.
*   **Integrated Authentication / Service Accounts:** Leveraging integrated authentication mechanisms (e.g., Windows Authentication for SQL Server, Oracle OS Authentication [carefully configured], Kerberos) where the application authenticates using the operating system identity of its service account, eliminating the need to manage separate database passwords within the application context.
*   **Authentication/Entitlement Servers:** Utilizing centralized authentication services (e.g., LDAP, Active Directory) where database access may be granted based on user or service authentication managed by the central server, potentially reducing the need for direct credential handling by the application itself. (Note: Specific implementation details must be secure).

3.3 Secure Credential Retrieval and Handling in Code

*   When credentials must be read from a configuration file or other source, they should be retrieved immediately prior to establishing the database connection.
*   Once the database connection is established, the memory variables holding the clear-text credentials must be securely cleared, zeroed out, or released as soon as technically feasible within the programming language's capabilities.
*   Source code files dedicated solely to retrieving or managing credentials must be kept separate from the main application logic and protected with appropriate access controls.

3.4 Credential Uniqueness and Management

*   Each distinct application or service implementing a specific business function should utilize unique database credentials. Sharing credentials between different applications or services is prohibited.
*   Database passwords used by applications are considered system-level passwords and must comply with the complexity, rotation, and management requirements defined in the organization's Password Policy.
*   Development teams must implement documented processes for securely managing and rotating application database passwords, restricting knowledge of these passwords on a strict need-to-know basis.

3.5 Pass-Through Authentication

*   Database authentication mechanisms that rely solely on remote host authentication without additional database-level checks (e.g., some configurations of Oracle OPS$) are prohibited if they grant broad, unverified access. Implementations must ensure proper user mapping and authorization within the database.

3.6 Secure Coding Guidelines

*   Development teams must follow organization-approved secure coding guidelines specific to the programming languages and frameworks being used (e.g., Java, C#, Python, Perl). These guidelines should incorporate specific techniques for implementing the requirements of this policy.
(Reference to internal secure coding standard documents should be maintained here).

4.0 Compliance

4.1 Compliance Measurement

The designated IT authority (e.g., Precision Computer team, Information Security, Internal Audit) will verify compliance with this policy through methods such as code reviews, security architecture reviews, configuration audits, vulnerability scanning, penetration testing, and review of documentation and processes.

4.2 Exceptions

Any exception to this policy requires formal, documented justification outlining the technical constraints or business necessity, proposed compensating controls, and risk assessment. Exceptions must be approved in advance by the designated IT authority (e.g., Precision Computer team).

4.3 Enforcement

*   Applications or code found to be in violation of this policy must be remediated within a defined timeframe (e.g., 90 days) or risk being disabled.
*   Violations by employees may result in disciplinary action, up to and including termination of employment.
*   Violations by temporary workers, contractors, or vendors may result in the termination of their contract or assignment.

5.0 Definitions

*   **Credentials:** Authentication information, typically a username and password pair, used to verify identity and grant access.
*   **Executing Body (of code):** The primary source code files containing the main application logic, as distinct from separate configuration files or dedicated credential management modules.
*   **Hash Function:** A cryptographic function that converts an input into a fixed-size string of characters (the hash value). Used for integrity checks and password storage (storing the hash, not the password itself), but not directly applicable for storing credentials needed for active authentication by an application.
*   **LDAP (Lightweight Directory Access Protocol):** A protocol used for accessing and maintaining distributed directory information services, often used for authentication and authorization.
*   **Module:** A self-contained unit of software code that performs a specific task or set of tasks.
*   **Secrets Management System:** A dedicated tool or service designed to securely store, manage, and control access to sensitive information like API keys, passwords, and certificates.

Related Policies:

* Password Policy
* Secure Coding Standards/Guidelines
* Data Classification Policy

Revision #2
Created 28 August 2024 16:50:11 by Daniel O
Updated 16 September 2025 22:10:09 by Travis Woolery