PAYMENT CARD INDUSTRY DATA SECURITY STANDARDS APPLICATION DEVELOPMENT AND MAINTENANCE PROCEDURES

Purpose: The Department of Information Technology (DoIT) is committed to developing secure applications. DoIT's System Development Methodology (SDM) and Application Development requirements ensure that security is paramount when developing an application.

Additional security is mandated for applications involved in credit card processing. This is any application included in cardholder environment that processes, stores or transmits credit card data. All application development shall comply with Payment Card Industry Data Security Standards (PCI DSS) Requirement 6: Development and maintain secure systems and applications. . . .

This procedure shall be used in conjunction with, and not as a replacement to, the System Development Methodology and the Application Development requirements.

Procedure: This document outlines all the steps and components that must be in place and validated as correct prior to moving an application that is part of the transmission, processing or storage of credit card transactions

The following must be in place prior to a new application being moved into production or any signification changes to an existing application are promoted to production.

1. Develop software applications securely and in accordance with PCI DSS Requirements relative to
   a. Secure authentication and logging
   b. Based on industry standards and best practices
   c. Incorporating information security through the software development lifecycle
2. Establish a separate development/test environment with appropriate segmentation from the production environment.
3. Remove development, test and any custom application accounts, user accounts and passwords prior to promoting the application to production for access by end users
4. Restrict test data to development/test environments and prohibit use of live cardholder data for use in the development/test environment.
5. Review of Custom Code prior to release to production
6. Penetration Testing for any significant change, or annually, prior to release to production
7. Follow all ICR policies and procedures relative to promoting application to production environment which includes separation of responsibilities for activities in development/test versus production environments.
8. Implement secure coding techniques that address items in the OWASP Top Ten as well as any common coding vulnerability identified explicitly in PCI DSS Requirement 6.5 et seq.

Guidance: PCI DSS requires that applications are tested specifically for the vulnerabilities listed in the procedures. This section provides guidance on testing procedures to be followed by developers.

PAYMENT CARD INDUSTRY DATA SECURITY STANDARDS APPLICATION DEVELOPMENT AND MAINTENANCE PROCEDURES

1. **Injection flaws, particularly SQL injection. Also consider OS Command Injection, LDAP and XPath injection flaws as well as other injection flaws**

   Testing procedure: Validate input to verify user data cannot modify meaning of commands and queries, utilize parameterized queries, etc.

   Guidance: Validate input to verify user data cannot modify meaning of commands and queries. Injection flaws, particularly SQL injection, are a commonly used method for compromising applications. Injection occurs when user-supplied data is sent to an interpreter as part of a command or query. The attacker's hostile data tricks the interpreter into executing unintended commands or changing data, and allows the attacker to attack components inside the network through the application, to initiate attacks such as buffer overflows, or to reveal both confidential information and server application functionality. This is also a popular way to conduct fraudulent transactions on commerce-enabled web sites. Information from requests should be validated before being sent to the application - for example, by checking for all alpha characters, mix of alpha and numeric characters, etc.

2. **Buffer Overflow**

   Testing procedure: Validate buffer boundaries and truncate input strings.

   Guidance: Ensure that applications are not vulnerable to buffer overflow attacks. Buffer overflows happen when an application does not have appropriate bounds checking on its buffer space. To exploit a buffer overflow vulnerability, an attacker would send an application a larger amount of information than one of its particular buffers is able to handle. This can cause the information in the buffer to be pushed out of the buffer's memory space and into executable memory space. When this occurs, the attacker has the ability to insert malicious code at the end of the buffer and then push that malicious code into executable memory space by overflowing the buffer. The malicious code is then executed and often enables the attacker remote access to the application and/or infected system.

3. **Insecure Cryptographic Storage**

   Testing procedure: Prevent cryptographic flaws and use strong cryptographic algorithms and keys

   Guidance: Prevent cryptographic flaws. Applications that do not utilize strong cryptographic functions properly to store data are at increased risk of being compromised and exposing cardholder data. If an attacker is able to exploit weak cryptographic processes, they may be able to gain clear-text access to encrypted data.

4. **Insecure communications**

   Testing procedure: Verify that insecure communications are addressed by coding techniques that properly authenticate and encrypt all sensitive communications

Guidance: Properly encrypt all authenticated and sensitive communications. Applications that fail to adequately encrypt network traffic using strong cryptography are at increased risk of being compromised and exposing cardholder data. If an attacker is able to exploit weak cryptographic processes, they may be able to gain control of an application or even gain clear-text access to encrypted data.

5. **Improper error handling**

Testing procedure: Verify that improper error handling is addressed by coding techniques that do not leak information via error messages

Guidance: Do not leak information via error messages or other means. Applications can unintentionally leak information about their configuration, internal workings, or violate privacy through a variety of application problems. Attackers use this weakness to steal sensitive data, or conduct more serious attacks. Also, incorrect error handling provides information that helps a malicious individual compromise the system. If a malicious individual can create errors that the application does not handle properly, they can gain detailed system information, create denial-ofservice interruptions, cause security to fail, or crash the server. For example, the message "incorrect password provided" tells them the user ID provided was accurate and that they should focus their efforts only on the password. Use more generic error messages, like "data could not be verified."

6. **All “High” Vulnerabilities**

Testing procedure: Verify that coding techniques address any “high risk” vulnerabilities that could affect the application.

Guidance: Any high vulnerabilities noted per Requirement 6.1 that could affect the application should be accounted for during the development phase. For example, a vulnerability identified in a shared library or in the underlying operating system should be evaluated and addressed prior to the application being released to production.

7. **Cross site Scripting**

Testing procedure: Validate all parameters before inclusion, utilize context-sensitive escaping, etc.

Guidance: All parameters should be validated before inclusion. XSS flaws occur whenever an application takes user supplied data and sends it to a web browser without first validating or encoding that content. XSS allows attackers to execute script in the victim's browser which can hijack user sessions, deface web sites, possibly introduce worms, etc.

8. **Improper Access Control**

Testing procedure: Verify that Improper Access Control, such as insecure direct object references, failure to restrict URL access, and directory traversal is addressed through secure coding. Properly authenticate users and sanitize input. Do not expose internal object references to users.

Guidance: Do not expose internal object references to users. A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, database record, or key, as a URL or form parameter. Attackers can manipulate those references to access other objects without authorization. Consistently enforce access control in presentation layer and business logic for all URLs. Frequently, the only way an application protects sensitive functionality is by preventing the display of links or URLs to unauthorized users. Attackers can use this weakness to access and perform unauthorized operations by accessing those URLs directly. Protect against directory traversal. An attacker may be able to enumerate and navigate the directory structure of a website thus gaining access to unauthorized information as well as gaining further insight into the workings of the site for later exploitation

9. **Cross-site Request Forgery**

Testing procedure: Validate that the application does not rely on authorization credentials and tokens automatically submitted by browsers.

Guidance: Do not reply on authorization credentials and tokens automatically submitted by browsers. A CSRF attack forces a logged-on victim's browser to send a pre authenticated request to a vulnerable web application, which then forces the victim's browser to perform a hostile action to the benefit of the attacker. CSRF can be as powerful as the web application that it attacks.

10. **Broken Authentication and Session Management**

Testing procedure: Verify that broken authentication and session management are addressed via coding techniques that commonly include flagging session tokens/cookies as secure, not exposing session IDs in the URL, and incorporating appropriate time-outs and rotation of session IDs.

Guidance: Secure authentication and session management prevents unauthorized individuals from compromising legitimate account credentials, keys, or session tokens that would otherwise enable the intruder to assume the identity of an authorized user.

Accountability: This procedure applies to all applications developed internally as well as bespoke or custom software developed by a third party.

It is the responsibility of each DoIT Division Director and Bureau Chief or their designee to enforce this policy. Employees who do not comply with this policy shall be subject to disciplinary action as outlined in the Administrative Rules of the Division of Personnel.

Description: This procedure provides a common approach to application security.

PAYMENT CARD INDUSTRY DATA SECURITY STANDARDS APPLICATION DEVELOPMENT AND MAINTENANCE PROCEDURES

Reference: System Development Methodology (SDM)
Payment Card Industry Data Security Standards Requirements 6.x
Application Security Procedures
Application Security Policy
Application Security Scan Request Form
SDM Security Guidelines
Administrator Account and Password Policy
User Account and Password Policy
User Account Maintenance Policy
IT Standards Exception Policy