PARASOFT®

Establishing a Continuous Process for PCI DSS Compliance

Visa, MasterCard, American Express, and other payment card companies currently require all U.S. merchants accepting credit card payments to comply with the Payment Card Industry Data Security Standard (PCI DSS), which is a global compliance initiative that outlines a set of comprehensive requirements to help organizations protect payment card account data from fraud, hacking, and other security vulnerabilities and threats. It requires organizations to not only perform many different tasks, but also to record who performed them and when. Parasoft Development Testing Platform (DTP) employs the Process Intelligence Engine (PIE) to help you establish a repeatable process that ensures your team members are conforming to your organization's prescribed PCI policy.

PCI DSS requirement 6 details how to "develop and maintain secure systems and applications." It promotes a proactive, preventative approach to building security into the application throughout the stages of software development, including continuous testing and continuous integration processes. This is in contrast to an ad-hoc approach that seeks to test security vulnerabilities out of the application one by one.

Parasoft is the industry leader in defect prevention—in fact, we wrote the book on it (*Automated Defect Prevention*, Wiley-IEEE, 2007). With over 25 years of experience helping over half of the Fortune 500 companies incorporate the PCI-mandated practices throughout the development process, Parasoft knows what it takes to rapidly bring organizations into compliance with PCI DSS.

Parasoft's PCI DSS Solution significantly reduces the time and cost of achieving PCI compliance by:

- **Delivering the industry's most comprehensive security vulnerability prevention and detection capabilities in an integrated solution:** Parasoft provides out-of-the-box automation of practices essential for achieving PCI DSS 6 compliance, including:
    - **Static analysis**– pattern-based coding standards, data flow analysis, code metrics.
    - **Dynamic analysis**– unit testing, integration testing, functional testing, memory error detection.
    - **Penetration testing**– runtime security policy validation (encryption, authentication, signatures).
    - **Peer code review** (and document review) process automation.
- **Providing out-of-the-box checking for the security issues referenced in PCI DSS requirement 6:** The solution is configured to deliver an instant assessment of compliance with PCI DSS requirement 6 security guidelines across Java, C/C++, .NET, Web language code, and other security-critical application artifacts (e.g., XML configuration files). This enables teams to rapidly assess the level of compliance—without spending time reading the PCI DSS specification and determining how the requirements translate to code.
- **Establishing an automated process that integrates security throughout the SDLC:** Parasoft's automated infrastructure facilitates continued compliance as the application evolves by making compliance to PCI-mandated practices an unobtrusive part of the team's existing workflow.
- **Facilitating issue remediation, not just issue detection:** Each issue detected is prioritized, automatically correlated to the developer who introduced it, then distributed to his or her IDE with direct links to the problematic code. Eventually, developers start writing compliant code as a matter of habit.
- **Delivering extensive reporting for documentation and process improvement:** Our centralized reporting system provides real-time visibility into overall security status and processes

Using Parasoft's integrated solution, organizations not only gain a fast track to PCI DSS 6 compliance, but also establish a process for ensuring that all of the mandated PCI DSS tasks are performed and documented as expected.

PARASOFT®

How Parasoft DTP Capabilities Facilitate Compliance with PCI DSS Requirements

PCI DSS Requirement	Parasoft DTP Component																										
	Policy Center			Process Intelligence Engine			Project Center			Test Engines															Report Center			
	PCI Policy	Policy Management	Policy Best Practices	Business Process Model	Business Rules/Notifications	Process Monitoring	Requirements Management	Task/Iteration Management	Task IDE Integration	Code Analysis - Pattern	Code Analysis - Flow	Coding Metrics	Automated Unit Testing	Functional Unit Testing	Runtime Error Detection	Message/Protocol Testing	Penetration Testing	Web Application Testing	End-To-End Testing	Manual Testing (UAT)	Business Process Testing	Load/Stress Testing	Change Based Testing	PCI Compliance Reporting	Policy Reporting	Process Visibility	Quality Visibility
6.1	■	■		■	■																			■	■		
6.2	■	■		■	■			■																■	■	■	
6.3	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
6.4	■	■	■	■	■	■	■	■	■															■	■	■	■
6.5	■	■	■				■	■	■	■	■			■	■	■	■	■		■			■	■	■		■
6.6	■	■	■	■	■	■	■	■	■	■	■						■						■	■	■	■	■
6.7	■	■		■	■	■		■	■															■	■	■	

Parasoft Support for PCI DSS 6

PCI DSS Requirements	Testing Procedures	Parasoft Capabilities
6.1 Establish a process to identify security vulnerabilities, using reputable outside sources for security vulnerability information, and assign a risk ranking (for example, as "high," "medium," or "low") to newly discovered security vulnerabilities. *Note:* R*isk rankings should be based on industry best practices as well as consideration of potential impact. For example, criteria for ranking vulnerabilities may include consideration of the CVSS base score, and/or the classification by the vendor, and/or type of systems affected.* *Methods for evaluating vulnerabilities and assigning risk ratings will vary based on an organization's environment and risk- assessment strategy. Risk rankings should, at a minimum, identify all vulnerabilities considered to be a "high risk" to the environment. In addition to the risk ranking, vulnerabilities may be considered "critical" if they pose an imminent threat to the environment, impact critical systems, and/or would result in a potential compromise if not addressed. Examples of critical systems may include security systems, public-facing devices and systems, databases, and other systems that store, process, or transmit cardholder data.*	**6.1.a** Examine policies and procedures to verify that processes are defined for the following: • To identify new security vulnerabilities • To assign a risk ranking to vulnerabilities that includes identification of all "high risk" and "critical" vulnerabilities. • To use reputable outside sources for security vulnerability information. **6.1.b** Interview responsible personnel and observe processes to verify that: • New security vulnerabilities are identified. • A risk ranking is assigned to vulnerabilities that includes identification of all "high" risk and "critical" vulnerabilities. • Processes to identify new security vulnerabilities include using reputable outside sources for security vulnerability information.	• The Parasoft DTP Engines routinely scan the code for new security vulnerabilities. • Parasoft automatically assigns a baseline risk ranking to vulnerabilities based on the inherent risk of the vulnerability • Users can re-prioritize and re-rank detected vulnerabilities in Parasoft DTP • Vulnerability rules are updated regularly, and updates can be applied

PCI DSS Requirements	Testing Procedures	Parasoft Capabilities
6.2 Ensure that all system components and software are protected from known vulnerabilities by installing applicable vendor- supplied security patches. Install critical security patches within one month of release. *Note: Critical security patches should be identified according to the risk ranking process defined in Requirement 6.1.*	**6.2.a** Examine policies and procedures related to security-patch installation to verify processes are defined for: • Installation of applicable critical vendor-supplied security patches within one month of release. • Installation of all applicable vendor-supplied security patches within an appropriate time frame (for example, within three months).	• Development tasks to check and implement patches are visible and reportable within the context of a project or development lifecycle. • Automated tests or notifications are created for manual verification of security patches.
	6.2.b For a sample of system components and related software, compare the list of security patches installed on each system to the most recent vendor security-patch list, to verify the following: • That applicable critical vendor-supplied security patches are installed within one month of release. • All applicable vendor-supplied security patches are installed within an appropriate time frame (for example, within three months).	• Policy enforcement notifies development to check for updated patches as part of the SDLC. • Development tasks to check and implement patches are visible and reportable within the context of a project or development lifecycle.
6.3 Develop internal and external software applications (including web-based administrative access to applications) securely, as follows: • In accordance with PCI DSS (for example, secure authentication and logging) • Based on industry standards and/or best practices. • Incorporating information security throughout the software-development life cycle *Note: this applies to all software developed internally as well as bespoke or custom software developed by a third party.*	**6.3.a** Examine written software-development processes to verify that the processes are based on industry standards and/or best practices.	Tracking written software development processes within a version control system allows PIE to detect changes and automatically assign tasks to perform examinations and interviews.
	6.3.b Examine written software-development processes to verify that information security is included throughout the life cycle.	
	6.3.c Examine written software-development processes to verify that software applications are developed in accordance with PCI DSS.	
	6.3.d Interview software developers to verify that written software-development processes are implemented.	
6.3.1 Remove development, test and/or custom application accounts, user IDs, and passwords before applications become active or are released to customers.	**6.3.1** Examine written software-development procedures and interview responsible personnel to verify that pre-production and/or custom application accounts, user IDs and/or passwords are removed before an application goes into production or is released to customers.	• Create and automate tests to identify test and custom account, user IDs, and passwords • Use Policy Center and PIE as quality gates to prevent production release when test failures occur.

PCI DSS Requirements	Testing Procedures	Parasoft Capabilities
6.3.2 Review custom code prior to release to production or customers in order to identify any potential coding vulnerability (using either manual or automated processes) to include at least the following: • Code changes are reviewed by individuals other than the originating code author, and by individuals knowledgeable about code-review techniques and secure coding practices. • Code reviews ensure code is developed according to secure coding guidelines • Appropriate corrections are implemented prior to release. • Code-review results are reviewed and approved by management prior to release. *Note: This requirement for code reviews applies to all custom code (both internal and public-facing), as part of the system development life cycle.* *Code reviews can be conducted by knowledgeable internal personnel or third parties. Public-facing web applications are also subject to additional controls, to address ongoing threats and vulnerabilities after implementation, as defined at PCI DSS Requirement 6.6.*	**6.3.2.a** Examine written software development procedures and interview responsible personnel to verify that all custom application code changes must be reviewed (using either manual or automated processes) as follows: • Code changes are reviewed by individuals other than the originating code author, and by individuals who are knowledgeable in code-review techniques and secure coding practices. • Code reviews ensure code is developed according to secure coding guidelines (see PCI DSS Requirement 6.5). • Appropriate corrections are implemented prior to release. • Code-review results are reviewed and approved by management prior to release. **6.3.2.b** Select a sample of recent custom application changes and verify that custom application code is reviewed according to 6.3.2.a, above.	• Automation and management of the code review workflow--including preparation, notification, and tracking of code reviews--address the known shortcomings of this very powerful inspection method. We automatically identify updated code by scanning the source control system or the local file system, matching the code with designated reviewers, and tracking the progress of each review item until closure. This allows teams to establish a bulletproof review process where all new code gets reviewed and all identified issues are resolved. • Automatically assign code review tasks based on code changes and verify that those tasks are completed. • Automatically scan code reviews to ensure secure coding guidelines were part of the review. • Track reviews in DTP for purposes of audit.
6.4.1 Separate development/test environments from production environments, and enforce the separation with access controls.	**6.4.1.a** Examine network documentation and network device configurations to verify that the development/test environments are separate from the production environment(s). **6.4.1.b** Examine access controls settings to verify that access controls are in place to enforce separation between the development/test environments and the production environment(s).	Application Virtualization can be used to provide access to production-like services and applications while maintaining a clear separation between development/test environments and production environments.
6.4.3 Production data (live PANs) are not used for testing or development	**6.4.3.a** Observe testing processes and interview personnel to verify procedures are in place to ensure production data (live PANs) are not used for testing or development. **6.4.3.b** Examine a sample of test data to verify production data (live PANs) is not used for testing or development.	Automated tests can check test PANs against live PANs and send reports to appropriate personnel when live PANs are detected in development or test environments.

PCI DSS Requirements	Testing Procedures	Parasoft Capabilities
6.4.4 Removal of test data and accounts before production systems become active	**6.4.4.a** Observe testing processes and interview personnel to verify test data and accounts are removed before a production system becomes active.	A combination of automated testing and application virtualization can both remove the need for test data to be hard-coded and verify that test data does not exist in code and properties files immediately prior to production. Quality gates can be used to block promotion to production for test failures in a continuous delivery scenario.
	6.4.4.b Examine a sample of data and accounts from production systems recently installed or updated to verify test data and accounts are removed before the system becomes active.	
6.4.5 Change control procedures for the implementation of security patches and software modifications must include the following:	**6.4.5.a** Examine documented change control procedures related to implementing security patches and software modifications and verify procedures are defined for: • Documentation of impact • Documented change approval by authorized parties • Functionality testing to verify that the change does not adversely impact the security of the system • Back-out procedures	• Automated change-based functional and unit testing. • Trigger back-out procedures automatically for any set of failure scenarios.
	6.4.5.b For a sample of system components, interview responsible personnel to determine recent changes/security patches. Trace those changes back to related change control documentation. For each change examined, perform the following:	
6.4.5.1 Documentation of impact.	**6.4.5.1** Verify that documentation of impact is included in the change control documentation for each sampled change.	Automatic change impact detection and reporting.
6.4.5.2 Documented change approval by authorized parties.	6.4.5.2 Verify that documented approval by authorized parties is present for each sampled change.	Define and automate business processes to require eSignature authorization under specified conditions.
6.4.5.3 Functionality testing to verify that the change does not adversely impact the security of the system.	**6.4.5.3.a** For each sampled change, verify that functionality testing is performed to verify that the change does not adversely impact the security of the system.	• Automated test runs are triggered by changes in a continuous integration and testing environment • Test failures are automatically reported to relevant developers or testers for immediate remediation
	6.4.5.3.b For custom code changes, verify that all updates are tested for compliance with PCI DSS Requirement 6.5 before being deployed into production.	
6.4.5.4 Back-out procedures.	**6.4.5.4** Verify that back-out procedures are prepared for each sampled change.	Parasoft PIE can interpret automated test results and trigger back-out procedures under user-defined conditions

PCI DSS Requirements	Testing Procedures	Parasoft Capabilities
6.5 Address common coding vulnerabilities in software-development processes as follows: • Train developers in secure coding techniques, including how to avoid common coding vulnerabilities, and understanding how sensitive data is handled in memory. • Develop applications based on secure coding guidelines. *Note: The vulnerabilities listed at 6.5.1 through 6.5.10 were current with industry best practices when this version of PCI DSS was published. However, as industry best practices for vulnerability management are updated (for example, the OWASP Guide, SANS CWE Top 25, CERT Secure Coding, etc.), the current best practices must be used for these requirements.*	6.5.a Examine software-development policies and procedures to verify that training in secure coding techniques is required for developers, based on industry best practices and guidance. **6.5.b** Interview a sample of developers to verify that they are knowledgeable in secure coding techniques. **6.5.c** Examine records of training to verify that software developers received training on secure coding techniques, including how to avoid common coding vulnerabilities, and understanding how sensitive data is handled in memory. **6.5.d** Verify that processes are in place to protect applications from, at a minimum, the following vulnerabilities:	Automated static analysis execution and assignment of findings trains or reinforces training relative to secure coding standards, including common coding vulnerabilities. Pre-configured PCI DSS test configurations can be leveraged in conjunction with task and process management to execute security tests through multiple layers of the application; this is accomplished through: • Penetration testing • Flow-based static analysis • Pattern-based static analysis • Unit/component testing • Functional/integration testing • User acceptance testing • Contextual peer code review
6.5.1 Injection flaws, particularly SQL injection. Also consider OS Command Injection, LDAP and XPath injection flaws as well as other injection flaws.	6.5.1 Examine software-development policies and procedures and interview responsible personnel to verify that injection flaws are addressed by coding techniques that include: • Validating input to verify user data cannot modify meaning of commands and queries. • Utilizing parameterized queries	• Static and flow analysis can find code that fails to validate potentially tainted input before passing it to vulnerable functions such as SQL query builders, LDAP, OS, XPath, and other accessors. • Penetration testing can verify that specific injection attacks are blocked for both internal and external applications
6.5.2 Buffer overflows	**6.5.2** Examine software-development policies and procedures and interview responsible personnel to verify that buffer overflows are addressed by coding techniques that include: • Validating buffer boundaries. • Truncating input strings.	• Static analysis and flow analysis can protect against buffer overflows • Runtime error detection can dynamically discover and report when overflows occur

PCI DSS Requirements	Testing Procedures	Parasoft Capabilities
6.5.3 Insecure cryptographic storage	**6.5.3** Examine software-development policies and procedures and interview responsible personnel to verify that insecure cryptographic storage is addressed by coding techniques that: • Prevent cryptographic flaws. • Use strong cryptographic algorithms and keys.	• Static analysis can enforce the use of specific algorithms throughout the application. • Dynamic testing can verify that encryption values match expectations.
6.5.4 Insecure communications	**6.5.4** Examine software-development policies and procedures and interview responsible personnel to verify that insecure communications are addressed by coding techniques that properly authenticate and encrypt all sensitive communications.	• Secure coding standards executed by static and flow analysis can enforce the use of appropriate authentication and encryption mechanisms when sending and receiving communications • Functional and unit testing can verify whether messages are encrypted and authenticated according to the defined strategy
6.5.5 Improper error handling	**6.5.5** Examine software-development policies and procedures and interview responsible personnel to verify that improper error handling is addressed by coding techniques that do not leak information via error messages (for example, by returning generic rather than specific error details.	Static analysis can detect and report when tainted or otherwise dangerous data is passed to a vulnerable interface.
6.5.6 All “high risk” vulnerabilities identified in the vulnerability identification process (as defined in PCI DSS Requirement 6.1).	**6.5.6** Examine software-development policies and procedures and interview responsible personnel to verify that coding techniques address any “high risk” vulnerabilities that could affect the application, as identified in PCI DSS Requirement 6.1.	Parasoft DTP and Parasoft PIE can automatically assign, distribute, track, and report all “high risk” findings of any kind, ensuring that they will be remediated and that management will have full visibility into unmitigated risks.
Note: Requirements 6.5.7 through 6.5.10, below, apply to web applications and application interfaces (internal or external):		Web applications, both internally and externally (public) facing, have unique security risks based upon their architecture as well as the relative ease and occurrence of compromise.

PCI DSS Requirements	Testing Procedures	Parasoft Capabilities
6.5.7 Cross-site scripting (XSS)	**6.5.7** Examine software-development policies and procedures and interview responsible personnel to verify that cross-site scripting (XSS) is addressed by coding techniques that include: • Validating all parameters before inclusion. • Utilizing context-sensitive escaping.	Multiple, complementary technologies automate a broad spectrum of security validation and verification practices for C/C++, Java, .NET, SOA, Web, and RIA– the most comprehensive in the industry. Practices supported "out-of-the-box" include: • Static code analysis - Coding standards, data flow, metrics–including preconfigured PCI DSS 6 , OWASP, and CWE / SANS Top 25 configurations. • Penetration testing - Message layer and web interface. • Runtime analysis - Buffer overflows and other memory errors. • Unit testing - Verification of input validation methods. • Runtime security policy validation - Validates encryption, authentication, signatures. • Although automated analysis can identify many vulnerabilities, it cannot detect instances where security controls are not designed properly or implemented as expected. Such high-level issues can only be detected when the human brain analyzes code in the context of its requirements and available test cases. This is facilitated through peer review workflow automation and management.
6.5.8 Improper access control (such as insecure direct object references, failure to restrict URL access, directory traversal, and failure to restrict user access to functions).	**6.5.8** Examine software-development policies and procedures and interview responsible personnel to verify that improper access control—such as insecure direct object references, failure to restrict URL access, and directory traversal—is addressed by coding technique that includes: • Proper authentication of users • Sanitizing input • Not exposing internal object references to users • User interfaces that do not permit access to unauthorized functions.	
6.5.9 Cross-site request forgery (CSRF)	**6.5.9** Examine software development policies and procedures and interview responsible personnel to verify that cross-site request forgery (CSRF) is addressed by coding techniques that ensure applications do not rely on authorization credentials and tokens automatically submitted by browsers.	
6.5.10 Broken authentication and session management *Note: Requirement 6.5.10 is a best practice until June 30, 2015, after which it becomes a requirement.*	**6.5.10** Examine software development policies and procedures and interview responsible personnel to verify that broken authentication and session management are addressed via coding techniques that commonly include: • Flagging session tokens (for example cookies) as "secure" • Not exposing session IDs in the URL • Incorporating appropriate time-outs and rotation of session IDs after a successful login.	

PCI DSS Requirements	Testing Procedures	Parasoft Capabilities
6.6 For public-facing web applications, address new threats and vulnerabilities on an ongoing basis and ensure these applications are protected against known attacks by either of the following methods: • Reviewing public-facing web applications via manual or automated application vulnerability security assessment tools or methods, at least annually and after any changes *Note: This assessment is not the same as the vulnerability scans performed for Requirement 11.2.* • Installing an automated technical solution that detects and prevents web-based attacks (for example, a web-application firewall) in front of public-facing web applications, to continually check all traffic.	**6.6** For public-facing web applications, ensure that either one of the following methods is in place as follows: • Examine documented processes, interview personnel, and examine records of application security assessments to verify that public-facing web applications are reviewed—using either manual or automated vulnerability security assessment tools or methods—as follows: »At least annually »After any changes »By an organization that specializes in application security »That, at a minimum, all vulnerabilities in Requirement 6.5 are included in the assessment »That all vulnerabilities are corrected »That the application is re-evaluated after the corrections. • Examine the system configuration settings and interview responsible personnel to verify that an automated technical solution that detects and prevents web-based attacks (for example, a web-application firewall) is in place as follows: »Is situated in front of public-facing web applications to detect and prevent web-based attacks. »Is actively running and up to date as applicable. »Is generating audit logs. »Is configured to either block web-based attacks, or generate an alert.	Automated security assessment can be conducted using multiple, complementary technologies that automate a broad spectrum of security validation and verification practices for C/C++, Java, .NET, SOA, Web, and RIA– the most comprehensive in the industry. Practices supported "out-of-the-box" include: • Static code analysis - Coding standards, data flow, metrics–including preconfigured PCI DSS 6, OWASP, and CWE/ SANS Top 25 configurations. • Penetration testing - Message layer and web interface. • Runtime analysis - Buffer overflows and other memory errors. • Unit testing - Verification of input validation methods. • Runtime security policy validation - Validates encryption, authentication, signatures. Manual review can be conducted with peer code review workflow automation and management—including preparation, notification, and tracking. We automatically identify updated code by scanning the source control system or the local file system, match the code with designated reviewers, and track the progress of each review item until closure. This allows teams to establish a bulletproof review process where all new code gets reviewed and all identified issues are resolved.

PCI DSS Requirements	Testing Procedures	Parasoft Capabilities
6.7 Ensure that security policies and operational procedures for developing and maintaining secure systems and applications are documented, in use, and known to all affected parties.	6.7 Examine documentation and interview personnel to verify that security policies and operational procedures for developing and maintaining secure systems and applications are: • Documented, • In use, and • Known to all affected parties.	Parasoft's automated assignment and reporting will ensure that systems are in use and known to affected parties.

About Parasoft's Application Security Solution

With over 25 years of experience helping top organizations implement PCI-mandated practices such as static analysis and code review, Parasoft knows what it takes to ensure adoption and establish a sustainable, repeatable, and predictable process. To achieve this, Parasoft's Application Security Solution establishes a continuous process that ensures security verification and remediation tasks are not only deployed across every stage of the SDLC, but also ingrained into the team's workflow. The solution automatically monitors policy compliance at all layers of the application stack, identifies vulnerabilities, and collects process metrics.

Developers secure code by simply responding to the reported tasks. To promote rapid remediation, each security issue detected is prioritized, automatically distributed to the developer who introduced it, then distributed to his or her IDE with direct links to the problematic code. Management gains real-time visibility into overall security status and processes, which allows teams to document improvements as well as determine what additional actions are needed to safeguard security.

For more information, visit http://www.parasoft.com/parasoft_security.

About Parasoft

For over 25 years, Parasoft has researched and developed software solutions that help organizations deliver defect-free software efficiently. By integrating Development Testing, API/cloud/SOA/composite app testing, and service virtualization, we reduce the time, effort, and cost of delivering secure, reliable, and compliant software. Parasoft's enterprise and embedded development solutions are the industry's most comprehensive—including static analysis, unit testing with requirements traceability, functional and load testing, dev/test environment management, and more. The majority of Fortune 500 companies rely on Parasoft in order to produce top-quality software consistently and efficiently.

Contacting Parasoft

USA

101 E. Huntington Drive, 2nd Floor
Monrovia, CA 91016
Toll Free: (888) 305-0041
Tel: (626) 305-0041
Fax: (626) 305-3036
Email: info@parasoft.com
URL: www.parasoft.com

Other Locations

See http://www.parasoft.com/contacts