Payment Card Industry (PCI)

Software Security Framework

Secure Software Requirements and Assessment Procedures

Version 1.2.1

May 2023

Document Changes

Date	Version	Description
January 2019	1.0	Initial release
April 2021	1.1	Update from v1.0. See *PCI Software Security Framework – Summary of Changes from Secure Software Requirements and Assessment Procedures Version 1.0 to 1.1* for details of changes.
October 2022	1.2	Update from v1.1. See *PCI Software Security Framework – Summary of Changes from Secure Software Requirements and Assessment Procedures Version 1.1 to 1.2* for details of changes.
May 2023	1.2.1	Update from v1.2 to address errata.

Table of Contents

Introduction

To ensure reliable and accurate electronic payment transactions, the systems and software used as part of the payment transaction flow must be designed, developed, and maintained in a manner that protects the integrity of payment transactions and the confidentiality of all sensitive data stored, processed, or transmitted in association with payment transactions. This document, the *Payment Card Industry (PCI) Secure Software Requirements and Assessment Procedures* (hereafter referred to as the "PCI Secure Software Standard" or "this standard") provides a baseline of security requirements with corresponding assessment procedures and guidance for building secure payment software.

The *PCI Secure Software Standard* is intended for use as part of the PCI Software Security Framework (SSF). Entities wishing to have their payment software validated under the PCI SSF would do so to this standard.

Terminology

A list of applicable terms and definitions is provided in the *PCI Software Security Framework Glossary of Terms, Abbreviations, and Acronyms,* available in the PCI SSC Document Library: https://www.pcisecuritystandards.org/document_library/.

Additionally, definitions for general PCI terminology is provided in the PCI Glossary on the PCI SSC website at: https://www.pcisecuritystandards.org/pci_security/glossary/.

Related Publications

In addition to the security requirements and assessment procedures for payment software defined in this standard, there are additional documents available to support the use of this standard. For more information, refer to the latest versions of (or successor documents to) the following PCI SSC publications in the PCI SSC Document Library:

Document Name	Description
PCI Software Security Framework – Technical FAQs for Secure Software Standard ("Secure Software Technical FAQs")	Technical Frequently Asked Questions (FAQs) provide a mechanism to address questions related to the interpretation and application of the associated PCI Standard and Program. Technical FAQs are considered 'normative,' and they must be fully considered within the scope of assessment activity for the associated Standard.
PCI Software Security Framework – PCI Secure Software Lifecycle Standard ("Secure SLC Standard")	Additional security requirements for software development organizations to ensure they develop and maintain software securely throughout the entire software lifecycle.
PCI Software Security Framework – Glossary of Terms, Abbreviations, and Acronyms ("SSF Glossary")	Describes important terms, abbreviations, and acronyms used throughout the Secure Software Standard and supporting documentation.
PCI Software Security Framework – Secure Software Program Guide ("Secure Software Program Guide")	Describes the program requirements for entities to validate their payment software for compliance to the Secure Software Standard and have their software listed and maintained on the PCI SSC's List of Validated Payment Software.
PCI Software Security Framework – Secure Software Template for Report on Validation ("Secure Software ROV Reporting Template")	The mandatory template that qualified SSF Assessors must use to document the results of a Secure Software Assessment and report those results to PCI SSC.
PCI Software Security Framework – Secure Software Attestation of Validation ("Secure Software AOV")	A template document provided by PCI SSC that Secure Software Assessor Companies and Vendors must use to attest to the results of a Secure Software Assessment.
PCI Software Security Framework – Qualification Requirements for Assessors ("SSF Qualification Requirements")	Describes the minimum capability and related documentation requirements that SSF Assessor Companies and their Assessor-Employees must satisfy to be qualified to perform Secure Software Assessments.
PCI PIN Transaction Security (PTS) Point-of-Interaction (POI) Modular Security Requirements ("PCI PTS POI Standard")	Security requirements that must be met for payment acceptance devices to obtain Payment Card Industry (PCI) PIN Transaction Security (PTS) Point of Interaction (POI) device approval.

Document Name	Description
Vendor Release Agreement ("VRA")	Establishes the terms and conditions that Vendors of Validated Payment Software must meet to participate in PCI Programs.

Stakeholder Roles and Responsibilities

There are numerous stakeholders involved in maintaining and managing PCI standards. The following describes the high-level roles and responsibilities of these stakeholders as they relate to the PCI Software Security Framework:

PCI SSC – Responsible for maintaining the standards, supporting programs, and related documentation associated with the PCI Software Security Framework including, but not limited to:

- Maintaining the PCI Secure Software Standard (this document).
- Maintaining all supporting documentation including reporting templates, attestation forms, frequently asked questions (FAQs), and guidance to assist entities implementing and assessing to this standard.
- Providing instructions and guidance for SSF Assessors in accordance with the requirements and assessment procedures in this standard.
- Maintaining a list of all SSF Assessors qualified to perform assessments to this standard (on the PCI SSC Website).
- Maintaining a quality assurance program for SSF Assessors.

Participating Payment Brands – Responsible for developing and enforcing their respective compliance programs related to PCI standards including, but not limited to:

- Defining and enforcing requirements, mandates, and deadlines for compliance to the PCI Secure Software Standard (this document).
- Determining the entities that are required to comply with this standard.
- Specifying the validation methods and frequency.
- Identifying and enforcing any fines or penalties for non-compliance.

SSF Assessor Companies – Responsible for maintaining the required knowledge, expertise, and equipment necessary to execute all assessment activities, adhering to all SSF Assessor Qualification Requirements, performing assessments to this standard, and generating the assessment report documenting the results. Note that not all SSF Assessor Companies are qualified to perform assessments to this standard. For more information on assessment activities and assessor qualification requirements, refer to the *PCI Secure Software Program Guide* and *Qualification Requirements for SSF Assessors*, respectively.

Payment Software Vendors / Providers / Developers – Responsible for developing, distributing, maintaining, and operating (where applicable) payment software, and ensuring that their payment software meets all applicable security requirements defined in this standard.

Overview of PCI Secure Software Standard

The security requirements defined within the *PCI Secure Software Standard* ensure that payment software is designed, engineered, developed, and maintained in a manner that protects payment transactions and data, minimizes vulnerabilities, and defends against attacks.

Scope of Security Requirements

The security requirements defined in this standard describe the security characteristics, controls, features, and capabilities that payment software must possess to protect the integrity of payment functions and the confidentiality of sensitive payment data. The payment software features that are in scope for these requirements include, but are not limited to:

- All end-to-end payment software functionality, including:
  - All payment functions.
  - Inputs and outputs.
  - Handling of error conditions.
  - Interfaces and connections to other files, systems, and/or software.
  - Data flows.
  - Security mechanisms, controls, and countermeasures, such as authentication, authorization, validation, parameterization, segmentation, logging, and so on.
- Processes used by the software vendor, provider, or developer to identify and support software security controls.
- Guidance that the software vendor, provider, or developer is expected to provide to stakeholders that describes:
  - How to implement and operate the payment software securely.
  - All configuration options available that can impact the security of payment software, including those of the execution environment and related system components.
  - How to implement security updates.
  - How and where to report security issues to the software vendor, provider, and/or developer.

  Note that the software vendor, provider, or developer may be expected to provide such guidance even when the specific settings:

  - Cannot be controlled by the payment software vendor, provider, or developer after the software is installed in a production environment.
  - Are the responsibility of the implementing entity and not the software vendor, provider, or developer.

- Any other software, software functionality, or services necessary for a full implementation of the payment software, including but not limited to:
  - Third-party and open-source software functions, libraries, packages, components, services, and dependencies embedded in or relied upon by the payment software to provide its intended function.
  - Features and functions of a supported platform or the execution environment relied upon by the payment software for security purposes.
  - Third-party or custom tools and functions relied upon by the payment software to satisfy security requirements in this standard.

Requirement Modules

The PCI Secure Software Standard includes the concept of requirement "modules," which are distinct groups of requirements relating to a specific topic or type of software. Modules are intended to clarify how and when specific requirements apply to a given payment application or function.

The requirements in this standard are organized into the following four requirement modules:

- **Core Requirements ("Core Module")**: General security requirements that apply to all types of payment software regardless of software function, design, or underlying technology.
- **Module A – Account Data Protection Requirements ("Account Data Protection Module")**: Additional security requirements for payment software that store, process, or transmit account data.
- **Module B – Terminal Software Requirements ("Terminal Software Module")**: Additional security requirements for payment software specifically designed for deployment and operation on PCI-approved POI devices.
- **Module C – Web Software Requirements ("Web Software Module")**: Additional security requirements for payment software that uses Internet technologies, protocols, and languages to initiate or support electronic payment transactions.

Requirement Module Applicability

Each requirement module includes its own applicability criteria. It is expected that software assessed to this standard will include assessment to all applicable modules. At a minimum, payment software must be assessed to the Core Module. Additional modules are included in the assessment when the software meets the applicability criteria for those additional modules. Refer to the "Purpose and Scope" section within each additional module for more information on module applicability criteria.

Be aware that some requirements defined within individual modules are extensions of Core Module requirements. Where such relationships are noted, the requirements in modules should be assessed in conjunction with their associated "Core" requirements.

Also note that there may be certain requirements defined within a module that are similar to requirements in other modules or that may have broader applicability beyond the module(s) where they are defined. Unless otherwise noted, such requirements are required to be assessed only in the context of that module. With that said, such requirements are likely to be consolidated and/or applied more broadly in future updates to this standard. Entities are encouraged to identify and apply requirements that may be applicable to an entity's payment software regardless of whether the entity is required to assess to the module where such requirements are defined.

Objective-Based Approach to Requirements

The PCI Software Security Framework has adopted an "objective-based" approach to defining the security requirements in this standard. The PCI SSC acknowledges that there is no "one size fits all" approach to software security and that software vendors need flexibility to determine the software security controls and features most appropriate to address their specific business needs and risks.

An "objective-based" approach is one that states security requirements as a desired security goal or outcome without necessarily specifying the method(s) to be used to achieve the desired goal. This approach enables entities to implement software security controls based on the risks identified by the software vendor for a given software application. For this approach to be successful, software vendors must possess a robust risk-management practice as an integral part of their software development lifecycle (SDLC) and be able to demonstrate how the implemented security controls are supported by the results of their risk identification and management practices. Without a robust risk-management practice in place and evidence available to support risk-based decision making, adherence to the requirements defined in this standard may be difficult to validate.

Requirement Frequency and Rigor

Given the nature of PCI SSC's objective-based approach to security requirements, many security requirements do not specify the level of rigor or frequency for periodic or recurring activities, such as the maximum period in which a security update must be provided to fix known vulnerabilities. In such cases, the software vendor may define the level of rigor or frequency appropriate for its business needs. The level of rigor or frequency chosen, however, must be supported by documented risk assessments and the resultant risk management decisions. Additionally, the software vendor must demonstrate that its implementation provides ongoing assurance that the software security controls and activities are effective and satisfy all relevant control objectives.

Requirement Structure

The security requirements defined in this standard are as follows:

- **Control Objectives** – The high-level security objectives that must be met. Control objectives are broadly stated to provide software vendors the flexibility to determine the best method(s) to achieve the stated objective. Regardless of the method(s) chosen, it is expected that the software vendor be able to produce clear and unambiguous evidence to demonstrate that the chosen method(s) is/are appropriate, sufficient, and properly implemented to satisfy the objective.

- **Test Requirements** – The expected assessment activities to be performed by an assessor to determine whether a specific control objective has been met. Test requirements are intended to provide both the software vendor and assessor with a common understanding of the tasks expected to be carried out by the assessor during testing. The specific method(s) used, the item(s) examined, and the personnel interviewed must be appropriate for the control objective being validated and for the software being assessed.
- **Guidance** – Additional information to help payment software vendors and assessors understand the intent of each control objective. The guidance may also include best practices that should be considered and examples of controls or methods that may be used to satisfy the control objective. Guidance is not intended to preclude other methods that a software vendor may use to meet a control objective, nor does it replace or amend the control objective to which it refers.

Testing Methods

To support the validation of their software to the requirements in this standard, software vendors are expected to produce evidence that they have satisfied the stated control objectives. The test requirements identified for each control objective describe the activities to be performed by the assessor to confirm that the software and/or software vendor have met the control objective(s). Test requirements include the following testing activities:

- **Examine**: The assessor critically evaluates data evidence. Common examples include software design and architecture documents (electronic or physical), source code, configuration and metadata files, bug tracking data and other output from software development systems, and security-testing results. The choice of evidence that may be used to meet an "examination" requirement is deliberately left open for the tester to determine. However, it is a requirement of this standard that the software source code be made available for review as part of the assessment. It is not acceptable for an assessment report to be provided where no source code was examined or used in the process of performing the testing.
- **Interview**: The assessor converses with individual personnel. The purposes of interviews include determining how an activity is performed, whether an activity is performed as defined, and whether personnel have particular knowledge or understanding of applicable policies, processes, responsibilities, or concepts.
- **Test:** The assessor evaluates the software operation to analyze its characteristics and behavior in various scenarios. Unless otherwise stated, software "testing" must include functional testing using forensic tools and techniques. Examples of such tools and techniques include the use of automated static analysis security testing (SAST), dynamic analysis security testing (DAST), interactive application security testing (IAST), and software composition analysis (SCA) tools. Where adversarial testing is explicitly referenced, fuzzing and other penetration testing tools and techniques must be used to try and bypass software security controls or to cause the software to behave in unintended ways.

The specific items or processes to be examined or tested, and the personnel to be interviewed should be appropriate for the control objective being validated and for each entity's organizational structure, culture, business practices, and software products. It is at the discretion of the assessor to determine the suitability or adequacy of the evidence provided by the entity to support each test requirement. Where bullets are specified in a control objective or test requirement, each bullet is expected to be validated as part of the assessment.

When documenting the assessment results, the assessor identifies the testing activities performed and the result of each activity. While it is expected that an assessor performs all test requirements defined for each control objective, it may also be possible for a control objective to be validated using different or additional testing methods. In such cases, the assessor is expected to document why alternative testing methods were used and how those methods provide at least the same level of assurance as the stated test requirements. Where terms such as "periodic," "appropriate," and "reasonable" are used in the test requirement, it is the software providers responsibility to define and defend its decisions regarding the frequency, robustness, and maturity of the implemented controls or processes.

Reliance on Third-Party Testing

All test requirements are expected to be performed by the assessor. An assessor may choose, however, to rely on testing performed by a third-party to satisfy a test requirement, including the software provider. The assessor retains full responsibility for the testing activities and results, regardless of whether the testing is performed by the assessor, by the software provider, or by a third-party. Where third-party testing is relied upon by the assessor, the assessor must document and justify the following:

- How the evidence provided by the third-party supports the same level of rigor as testing performed by the assessor, and
- How the assessor verified that the third-party testing relied upon by the assessor is appropriate.

Where an assessed entity's testing is to be used for the purposes of satisfying test requirements, the assessor must first verify the software vendor is Secure SLC-qualified[1] before software vendor testing can be relied upon.

Use of Sampling

Where appropriate, the assessor may utilize sampling as part of the testing process in accordance with a documented sampling methodology. The assessor's sampling methodology must detail how samples are chosen and must be provided to PCI SSC upon submission of the Report on Validation (ROV).

Sample selection must include a representative sample of all people, processes, and technologies in scope for the PCI Secure Software assessment. Sample sizes must be sufficiently large to demonstrate that the sample accurately reflects the characteristics of the larger population.

In instances where the assessor's findings are based on a representative sample rather than the complete set of applicable items, the assessor must explicitly note this fact in the ROV, detail the items chosen as samples for the testing, and provide references to the applicable sections of the assessor's sampling methodology provided with the ROV. Where the assessor selects samples that do not align with the assessor's documented sampling methodology, the assessor must provide justification in the ROV for each instance where such samples are used.

[1] Refer to the PCI Secure SLC Standard and its associated Program Guide for more information on Secure SLC qualification.

Use of a Test Platform

To ensure that software testing complies with this standard, it may be necessary for the software vendor to provide a test platform. A test platform is special test functionality that is either separate or absent from production-level code. The test platform must rely on as much of the intended production-level functionality as possible. The test platform serves only to provide a test framework that allows for software functionality to be exercised outside of a production-level deployment environment to verify the software's compliance to this standard. For example, elevated privileges or access capabilities may need to be granted for the purpose of providing run-time visibility into various facets of the software operation. Other examples include providing a test function to initiate a test transaction or to perform authentication functions. It is at the assessor's discretion to request any test functionality deemed necessary to verify the software's compliance with applicable requirements in this standard.

Core Requirements

Minimizing the Attack Surface

The attack surface of the software is minimized. Confidentiality and integrity of all software critical assets are protected, and all unnecessary features and functions are removed or disabled.

Control Objectives	Test Requirements	Guidance
Control Objective 1: Critical Asset Identification All software critical assets are identified and classified.		
1.1 All sensitive data stored, processed, or transmitted by the software is identified.	**1.1.a** The assessor shall examine evidence to confirm that information is maintained that details all sensitive data that is stored, processed, and/or transmitted by the software. At a minimum, this shall include all payment data; authentication credentials; cryptographic keys and related data (such as IVs and seed data for random number generators); and system configuration data (such as registry entries, platform environment variables, prompts for plaintext data in software allowing for the entry of PIN data, or configuration scripts).	Software security controls are designed and implemented to protect the confidentiality and/or integrity of critical assets. To make sure these controls are effective and appropriate, the software vendor should identify all sensitive data the software collects, stores, processes, or transmits, as well as all sensitive functions and resources it either provides or uses.
	1.1.b The assessor shall examine evidence to confirm that information is maintained that describes where sensitive data is stored. This includes the storage of sensitive data in temporary storage (such as volatile memory), semi-permanent storage (such as RAM disks), non-volatile storage (such as magnetic and flash storage media), or in specific locations or form factors (such as with an embedded system that is only capable of local storage).	
	1.1.c The assessor shall examine evidence to confirm that information is maintained that describes the security controls that are implemented to protect sensitive data.	
	1.1.d The assessor shall test the software to validate the evidence obtained in Test Requirements 1.1.a through 1.1.c.	

Control Objectives	Test Requirements	Guidance
	1.1.e The assessor shall examine evidence and test the software to identify the transaction types and/or card data elements that are supported by the software, and to confirm that the data for all of these is supported by the evidence examined in Test Requirements 1.1.a through 1.1.c.	
	1.1.f The assessor shall examine evidence and test the software to identify the cryptographic implementations that are supported by the software (including cryptography used for storage, transport, and authentication), and to confirm that the cryptographic data for all of these implementations is supported by the evidence examined in Test Requirements 1.1.a through 1.1.c, and that the evidence describes whether these are implemented by the software itself, through third-party software, or as functions of the execution environment.	
	1.1.g The assessor shall examine evidence and test the software to identify the accounts and authentication credentials supported by the software (including both default and user-created accounts) and to confirm that these accounts and credentials are supported by the evidence examined in Test Requirements 1.1.a through 1.1.c.	
	1.1.h The assessor shall examine evidence and test the software to identify the configuration options provided by the software that can impact sensitive data (including those provided through separate files or scripts, internal functions, or menus and options), and to confirm that these are supported by the evidence examined in Test Requirements 1.1.a through 1.1.c.	
1.2 All sensitive functions and sensitive resources provided or used by the software are identified.	**1.2.a** The assessor shall examine evidence to confirm that information is maintained that details all sensitive functions and sensitive resources provided or used by the software. At a minimum, this shall include all functions that are designed to store, process, or transmit sensitive data and those services, configuration files, or other information necessary for the normal and secure operation of those functions.	

Control Objectives	Test Requirements	Guidance
	1.2.b The assessor shall examine evidence to confirm that information is maintained that clearly describes how and where the sensitive data associated with these functions and resources is stored. This includes the storage of sensitive data in temporary storage (such as volatile memory), semi-permanent storage (such as RAM disks), and non-volatile storage (such as magnetic and flash storage media). The assessor shall confirm that this information is supported by the evidence examined in Test Requirement 1.1.a through 1.1.c.	
	1.2.c Where the sensitive functions or sensitive resources are provided by third-party software or systems, the assessor shall examine evidence and test the software to confirm that the software correctly follows available guidance for the third-party software. ***Note:*** *For example, by reviewing the security policy of a PTS or FIPS140-2 or 140-3 approved cryptographic system.*	
1.3 Critical assets are classified.	**1.3** The assessor shall examine evidence to confirm that: • The software vendor defines criteria for classifying critical assets in accordance with the confidentiality, integrity, and resiliency requirements for each critical asset. • An inventory of all critical assets with appropriate classifications is maintained.	Critical assets represent the sensitive data, functions, and resources that have business value and require confidentiality, integrity, or resiliency protection. There are numerous analysis techniques that can be used to identify critical assets, including Mission Impact Analysis (MIA), Functional Dependency Network Analysis (FDNA), and Mission Threat Analysis. Additional information and techniques can be found in publications such as the appendices of *NIST Special Publication 800-160* or in other publications from industry standards bodies such as EMVCo, ISO or ANSI.

Control Objectives	Test Requirements	Guidance
Control Objective 2: Secure Defaults Default privileges, features, and functions are restricted to only those necessary to provide a secure default configuration.		
2.1 All functions exposed by the software are enabled by default only when and where it is a documented and justified part of the software architecture.	**2.1.a** The assessor shall examine evidence and test the software to identify any software APIs or other interfaces that are provided or exposed by default upon installation, initialization, or first use. For each of these interfaces, the assessor shall confirm that the vendor has documented and justified its use as part of the software architecture. Testing shall include methods to reveal any exposed interfaces or other software functionality (such as scanning for listening services where applicable). ***Note:*** *This includes functions that are auto-enabled as required during operation of the software.*	Software often contains functionality (for example, web services, administrative interface, application heartbeat, etc.) that is optional and is generally unused by many users. This functionality typically does not receive the same attention as standard or essential software functions and services, and often contains security weaknesses that can be exploited by malicious users to bypass security controls. To ensure a secure software deployment, the software's default configuration should only expose functionality that has been reviewed, justified, and approved. This should include the default configuration for all software APIs, protocols, daemons, listeners, components, etc. Any unnecessary services, protocols, or ports should be disabled or removed. For guidance on services, protocols, or ports considered to be insecure, refer to industry standards and guidance (for example, NIST, ENISA, etc.).
	2.1.b The assessor shall test the software to determine whether any of the interfaces identified in Test Requirement 2.1.a rely on external resources for authentication. Where such resources are relied upon, the assessor shall examine evidence to confirm that methods are implemented to ensure that proper authentication remains in place and that these methods are included in the assessment of other applicable requirements in this standard.	
	2.1.c The assessor shall test the software to determine whether any of the interfaces identified in Test Requirement 2.1.a rely on external resources for the protection of sensitive data during transmission. Where such resources are relied upon, the assessor shall examine evidence to confirm that methods are implemented to ensure proper protection remains in place and that these methods are included in the assessment of other applicable requirements in this standard.	

Control Objectives	Test Requirements	Guidance
	2.1.d The assessor shall test the software to determine whether any of the interfaces identified in Test Requirement 2.1.a expose functions or services that have publicly disclosed vulnerabilities by conducting a search on the exposed protocols, methods, or services in public vulnerability repositories such as that maintained within the National Vulnerability Database.	
	2.1.e Where known vulnerabilities in exposed interfaces exist, the assessor shall examine evidence and test the software to confirm the following: • Methods are implemented to mitigate the exploitation of these weaknesses. • The risks posed by the use of known vulnerable protocols, functions, or ports are documented. • Clear and sufficient guidance on how to correctly implement sufficient security to meet applicable control objectives in this standard is provided to stakeholders in accordance with Control Objective 12.1. ***Note:*** *The assessor should reference the vendor threat information defined in Control Objective 4.1 for this item.*	
	2.1.f The assessor shall examine evidence to identify any third-party modules used by the software and to confirm that any such functions exposed by each module are disabled, unable to be accessed through mitigation methods implemented by the software, or formally documented and justified by the software vendor. Where access to third-party functions is prevented through implemented protection methods, the assessor shall test the software to confirm that it does not rely on a lack of knowledge of such functions as a security mitigation method by simply not documenting an otherwise accessible API interface, and to confirm that the protection methods are effective at preventing the insecure use of such third-party functions.	

Control Objectives	Test Requirements	Guidance
2.2 All software security controls, features, and functions are enabled upon software installation, initialization, or first use. ***Note:*** *Specific software security controls required to protect the integrity and confidentiality of sensitive data, sensitive functions, and sensitive resources are captured in the Software Protection Mechanisms section.*	**2.2.a** The assessor shall examine evidence and test the software to identify all software security controls, features and functions relied upon by the software for the protection of critical assets and to confirm that all are enabled upon installation, initialization, or first use of the software.	As previously noted earlier in guidance, software security controls are designed and implemented to protect the confidentiality and integrity of critical assets. Examples of such software security controls include authentication and authorization mechanisms, cryptographic controls, and controls to prevent leakage of sensitive data. Default software settings should result in a secure software configuration and should not rely on the end-user being a subject-matter expert to ensure a secure configuration. To that effect, all available software security controls should be active upon software installation, initialization, or first use, depending upon how the software is deployed.
	2.2.b Where any software security controls, features and functions are enabled only upon initialization or first use, the assessor shall test the software to confirm that sensitive data is processed only after this initialization process is complete.	
	2.2.c Where user input or interaction is required to enable software security controls, features, or functions (such as the installation of certificates), the assessor shall examine evidence to confirm that clear and sufficient guidance on configuring these options is provided to stakeholders in accordance with Control Objective 12.1.	
2.3 Default authentication credentials or keys for built-in accounts are not used after installation, initialization, or first use.	**2.3.a** The assessor shall examine evidence to identify the default credentials, keys, certificates, and other critical assets used for authentication by the software. ***Note:*** *The assessor should refer to evidence obtained in the testing of Control Objectives 1, 5, and 7 to determine the authentication and access control mechanisms, keys, and other critical assets used for authentication.*	To protect against unauthorized access, payment software should prevent the use of built-in accounts until the default authentication credentials can be changed. *(continued on next page)*

Control Objectives	Test Requirements	Guidance
	2.3.b The assessor shall test the software to confirm that all default credentials, keys, certificates, and other critical assets used for authentication by the software are supported by the evidence examined. ***Note:*** *It is expected that this analysis will include, but not necessarily be limited to, the use of entropy analysis tools to look for hardcoded cryptographic keys, searches for common cryptographic function call and structures such as S-Boxes and big-number library functions (and tracing these functions backwards to search for hardcoded keys), as well as checking for strings containing common user account names or password values.*	Built-in accounts with known credentials such as default or empty passwords, or default keys are often overlooked during installation, initial configuration, or use, and can be used by a malicious user to bypass access controls. Therefore, the software should not use or rely on the default credentials for its operation upon installation, initialization, or first use.
	2.3.c Where user input or interaction is required to disable or change any authentication credentials or keys for built-in accounts, the assessor shall examine evidence to confirm that guidance on configuring these options is provided to stakeholders in accordance with Control Objective 12.1.	
	2.3.d The assessor shall test the software to confirm that default authentication credentials or keys for built-in accounts are not used by the authentication and access control mechanisms implemented by the software after software installation, initialization, or first use. ***Note:*** *The assessor should refer to evidence obtained in the testing of Control Objective 5 to determine the authentication and access control mechanisms implemented by the software.*	
	2.3.e The assessor shall test the software to confirm that cryptographic keys used for authentication are not used for other purposes, such as protecting sensitive data during storage and transmission. ***Note:*** *The assessor should refer to evidence obtained in the testing of Control Objective 6 to determine the software security controls implemented to protect sensitive data.*	

Control Objectives	Test Requirements	Guidance
2.4 The privileges and resources requested by the software from its execution environment are limited to those necessary for the operation of the software.	**2.4.a** The assessor shall examine evidence to identify the privileges and resources required by the software and to confirm that information is maintained that describes and reasonably justifies all privileges and resources required, including explicit permissions for access to resources, such as cameras, contacts, etc.	In many attacks on software or underlying systems, the software is often used to execute functions on the underlying operating systems or to abuse accessible external resources. When the software requires excessive permissions, such permissions may be exploited by a malicious user. To minimize the software's attack surface, the software should only request and be granted the minimum required privileges for its intended operation. For example, system service accounts that the software uses to operate, or accounts used by the software to access underlying components such as a database or invoke web-services calls should not require permissions that exceed the minimum necessary for the software to perform its operations. The same concept applies to resources used by the software. The software should be granted access to only the minimum required resources for its expected operation. For example, mobile applications that do not require access to the camera or photographs should not request such access unless they are a necessary part of the software architecture. Similarly, software should not have access to sensitive files (for example, /etc/passwd) unless there is a legitimate need for the software to access those files.
	2.4.b Where limiting access is not possible due to the architecture of the solution or the execution environment in which the software is executed, the assessor shall examine evidence to identify all mechanisms implemented by the software to prevent unauthorized access, exposure, or modification of critical assets, and to confirm that guidance on properly implementing and configuring these mechanisms is provided to stakeholders in accordance with Control Objective 12.1.	
	2.4.c The assessor shall test the software to confirm that access permissions and privileges are assigned according to the evidence examined in Test Requirement 2.4.a. The assessor shall, where possible, use suitable tools for the platform on which the software is installed to review the permissions and privileges of the software itself, as well as the permissions and privileges of any resources, files, or additional elements generated or loaded by the software during use. ***Note:*** *Where the above testing is not possible, the assessor shall justify why this is the case and that the testing that has been performed is sufficient.*	
	2.4.d Where the software execution environment provides legacy features for use by older versions of the software, the assessor shall examine evidence and test the software to confirm that these are not used, and that only recent and secured functionality is implemented. For example, software should "target" the latest versions of APIs provided by the environment on which they run, where available.	

Control Objectives	Test Requirements	Guidance
2.5 Default privileges for built-in accounts are limited to those necessary for their intended purpose and function.	**2.5.a** The assessor shall examine the evidence to identify all default accounts provided by the software and to confirm that the privileges assigned to these accounts are justified and reasonable.	In support of the principle of "least privilege," built-in accounts should only have the privileges required for the intended function of the account, including access to sensitive data and resources as well as the ability to execute sensitive functions. For example, a built-in administrator account may require the ability to configure the software and associated user accounts, but not the ability to access areas containing sensitive data. Applying the principle of least privilege to user accounts helps prevent users without sufficient knowledge about the software from incorrectly or accidentally changing the software configuration or its security settings. Enforcing least privilege also helps to minimize the effects of unauthorized access to software user accounts. To limit access to sensitive data, functions, and resources to only those accounts that require such access, the level of privilege and access required should be defined and documented for each built-in account in an access matrix, such that its assigned functions may be performed, but no additional or unnecessary access or privileges are granted.
	2.5.b The assessor shall test the software to confirm that all default accounts provided or used by the software are supported by the evidence examined in Test Requirement 2.5.a.	
	2.5.c The assessor shall examine evidence and test the software to confirm that exposed interfaces, such as APIs, are protected from attempts by unauthorized users to modify account privileges and elevate user access rights.	

Control Objectives	Test Requirements	Guidance
Control Objective 3: Sensitive Data Retention Retention of sensitive data is minimized.		
3.1 The software only retains the sensitive data absolutely necessary for the software to provide its intended functionality.	**3.1.a** The assessor shall examine evidence to identify the sensitive data that is collected by the software for use beyond any one transaction, the default time period for which it is retained, and whether the retention period is user-configurable, and to confirm that the purpose for retaining the sensitive data in this manner is justified and reasonable. ***Note:*** *The assessor should refer to evidence obtained in the testing of Control Objective 1.1 to determine the sensitive data retained by the software.*	To prevent the unauthorized disclosure of sensitive data to unauthorized parties, the software should retain sensitive data only for the duration necessary to perform the specific operation for which sensitive data is collected. Retaining sensitive data longer than required presents opportunity for the data to be mishandled, misused, or accidentally disclosed. This control objective differentiates between transient sensitive data retained temporarily and persistent sensitive data that is retained on a more permanent basis. Examples of transient sensitive data include the retention account data in memory until payment authorization is received. Examples of persistent sensitive data include the storage of account data on disk to support recurrent payment transactions.
	3.1.b The assessor shall test the software to confirm that all available functions or services designed for the retention of sensitive data are supported by the evidence examined in Test Requirement 3.1.a. ***Note:*** *The assessor should refer to evidence obtained in the testing of Control Objective 1.2 to determine the sensitive functions and services provided or used by the software.*	
	3.1.c The assessor shall examine evidence and test the software to determine whether the software facilitates the storage of persistent sensitive data for the purposes of debugging, error finding or testing of systems, and to confirm that such data is protected during storage in accordance with Control Objective 6.1. Any function that allows for the storage of sensitive data for these purposes must be explicitly enabled through an interface that requires interaction and authorization by the user and retains the data only for the duration necessary in accordance with reasonable vendor criteria. Closure of the software must result in termination of this debugging state, such that it requires explicit re-enablement when the software is next executed, and any sensitive data is securely deleted per Control Objective 3.4.	

Control Objectives	Test Requirements	Guidance
	3.1.d Where user input or interaction is required to configure the retention period of sensitive data, the assessor shall examine evidence to confirm that guidance on configuring these options is provided to stakeholders in accordance with Control Objective 12.1.	
3.2 Transient sensitive data is retained only for the duration necessary to fulfill a legitimate business purpose.	**3.2.a** Using information obtained in Test Requirement 1.1.a, the assessor shall examine evidence to identify all sensitive data that is retained by the software for transient use, what triggers the secure deletion of this data, and to confirm that the purposes for retaining the data are justified and reasonable. This includes data that is stored only in memory during the operation of the software.	Sensitive data elements collected in conjunction with software operations should only be retained for as long as required to complete that operation or related transaction. After payment processing is complete, all transient sensitive data should be securely deleted from all locations where it has been retained such that any subsequent process, component, function, application, or user within the environment may not access or capture the sensitive data. Software vendors should also be aware of and account for how other aspects of the software architecture (such as the software-development language and operating environment) may affect how and where transient sensitive data is retained. For example, operating-system usage of swap partitions or virtual memory files can cause information that should have been transient to persist longer than intended. If any sensitive data must be used for debugging or troubleshooting purposes, the software should only capture the minimum amount of data necessary and store it securely in a known location.
	3.2.b Using information obtained in Test Requirement 1.2.a, the assessor shall test the software to confirm that all available functions or services that retain transient sensitive data are supported by evidence examined in Test Requirement 3.2.a and do not use immutable objects.	
	3.2.c The assessor shall examine evidence and test the software to determine whether the software facilitates the storage of transient sensitive data for the purposes of debugging, error finding or testing of systems, and to confirm that such data is protected in accordance with Control Objective 6.1. Any function that allows for the storage of transient sensitive data for these purposes must be explicitly enabled through an interface that requires interaction and authorization by the user. Closure of the software must result in the termination of this debugging state, such that it requires explicit re-enablement when the software is next executed, and any transient sensitive data is securely deleted in accordance with Control Objective 3.5.	

Control Objectives	Test Requirements	Guidance
	3.2.d Where the retention of transient sensitive data requires user input or interaction, the assessor shall examine evidence to confirm that guidance on configuring these options is provided to stakeholders in accordance with Control Objective 12.1.	
3.3 The software protects the confidentiality and integrity of sensitive data (both transient and persistent) during retention. ***Note:*** *The Software Protection Mechanisms section includes several specific software security controls that are required to be implemented to protect sensitive data during storage, processing, or transmission. Those software security controls should be analyzed to determine their applicability to the types of sensitive data retained by the software.*	**3.3.a** The assessor shall examine the evidence to identify the methods implemented to protect sensitive data during storage.	The software should maintain security controls and mechanisms to protect all sensitive data while it is retained by the software. Examples of software security controls include writing to a secure memory location or using cryptography to render the data unreadable.
	3.3.b Where sensitive data is stored outside of temporary variables within the code itself, the assessor shall test the software to confirm that sensitive data is protected using either strong cryptography or other methods that provide an equivalent level of security.	
	3.3.c Where protection methods use cryptography, the assessor shall examine evidence and test the software to confirm that the cryptographic implementation complies with Control Objective 7 of this standard.	
	3.3.d Where sensitive data is protected using methods other than strong cryptography, the assessor shall examine evidence and test the software to confirm that the protections are present in all environments where the software is designed to be executed and are implemented correctly.	
	3.3.e Where user input or interaction is required to configure protection methods, the assessor shall examine evidence to confirm that guidance on configuring these options is provided to stakeholders in accordance with Control Objective 12.1.	

Control Objectives	Test Requirements	Guidance
3.4 The software securely deletes persistent sensitive data when it is no longer required.	**3.4.a** The assessor shall examine evidence to identify the methods implemented to render persistent sensitive data irretrievable and to confirm that sensitive data is rendered unrecoverable after the process is complete.	Secure deletion is the process of rendering data irretrievable to other people, processes, or systems. Secure deletion may be required at the end of a software-specific operation or upon completion of user-specified retention requirements. In the latter case, the software should be able to securely delete the sensitive data after expiry of the user-specified retention period. Only in circumstances where the retention of sensitive data is explicitly permitted should the data be retained after transaction processing is complete.
	3.4.b The assessor shall examine evidence and test the software to identify any platform or implementation level issues that complicate the secure deletion of non-transient sensitive data and to confirm that any non-transient sensitive data is securely deleted using a method that ensures that the data is rendered unrecoverable. Methods may include (but are not necessarily limited to) overwriting the data, deletion of cryptographic keys (of sufficient strength) that have been used to encrypt the data, or platform-specific functions that provide for secure deletion. Methods must accommodate for platform-specific issues, such as flash wear-levelling algorithms or SSD over-provisioning, which may complicate simple over-writing methods.	
	3.4.c The assessor shall test the software using forensic tools to identify any non-transient sensitive data residue in the execution environment, and to confirm that the methods attested by the software vendor are correctly implemented and applied to all sensitive data. This analysis should accommodate for the data structures and methods used to store the sensitive data (for example, by examining file systems at the allocation level and translating data formats to identify sensitive data elements) and cover all non-transient sensitive data types. ***Note:*** *Where forensic testing of some or all aspects of the platform is not possible, the assessor should examine additional evidence to confirm secure deletion of sensitive data. Such evidence may include (but is not necessarily limited to) memory and storage dumps from development systems, evidence from memory traces from emulated systems, or evidence from physical extraction of data performed on-site by the software vendor.*	

Control Objectives	Test Requirements	Guidance
3.5 Transient sensitive data is securely deleted from temporary storage facilities automatically by the software once the purpose for which it is retained is satisfied.	**3.5.a** The assessor shall examine evidence to identify the methods implemented to render transient sensitive data irretrievable and to confirm that sensitive data is unrecoverable after the process is complete. ***Note:*** *This includes data which may be stored only temporarily in program memory / variables during operation of the software.*	Where sensitive data is only retained temporarily to perform a specific function (such as a payment transaction), mechanisms are required to securely delete the sensitive data once the specific function has completed. Transient sensitive data is often erased from temporary storage locations after processing is complete. However, that data may remain resident in volatile memory (RAM) or in other storage locations for longer periods than anticipated (such as in swap files/partitions or log files). Software vendors should account for all locations where sensitive data is stored, regardless of the intended duration of storage, and ensure that such data is securely deleted once the purpose for which the software collected the data has been satisfied.
	3.5.b The assessor shall examine evidence and test the software to identify any platform or implementation level issues that complicate the erasure of such transient sensitive data, such as abstraction layers between the code and the hardware execution environment, and to confirm that methods have been implemented to minimize the risk posed by these complications.	
	3.5.c The assessor shall test the software to identify any sensitive data residue in the execution environment and to confirm that the methods implemented are implemented correctly and enforced for all transient sensitive data. This analysis should accommodate for the data structures and methods used to store the sensitive data (for example, by examining file systems at the allocation level and translating data formats to identify sensitive data elements) and cover all non-transient sensitive data types. ***Note:*** *Where forensic testing of some or all aspects of the platform is not possible, the assessor should examine additional evidence to confirm secure deletion of sensitive data. Such evidence may include (but is not necessarily limited to) memory and storage dumps from development systems, evidence from memory traces from emulated systems, or evidence from physical extraction of data performed on-site by the software vendor.*	

Control Objectives	Test Requirements	Guidance
3.6 The software does not disclose sensitive data through unintended channels.	**3.6.a** The assessor shall examine evidence to confirm the software vendor has performed a thorough analysis to account for all sensitive data disclosure attack vectors including, but not limited to: • Error messages, error logs, or memory dumps. • Execution environments that may be vulnerable to remote side-channel attacks to expose sensitive data, such as attacks that exploit cache timing or branch prediction within the platform processor. • Automatic storage or exposure of sensitive data by the underlying execution environment, such as through swap-files, system error logging, keyboard spelling, and auto-correct features. • Sensors or services provided by the execution environment that may be used to extract or leak sensitive data, such as through use of an accelerometer to capture input of a passphrase to be used as a seed for a cryptographic key, or through capture of sensitive data through use of cameras or near-field communication (NFC) interfaces.	Proactive measures to ensure that sensitive data is not inadvertently "leaked" should be implemented by the software vendor or within the software. Disclosure of sensitive data to unauthorized parties often occurs through unknown or unintended outputs or channels. For example: sensitive data could be unintentionally disclosed through error- or exception-handling routines, logging or debugging channels, third-party services and/or components, or through the use of shared resources such as memory, disk, files, keyboards, displays, and functions. Protective mechanisms, whether process or programmatic in nature, should be implemented to ensure that sensitive data is not accidentally disclosed through such means.
	3.6.b The assessor shall examine evidence, including the results of the analysis described in Test Requirement 3.6.a, and test the software to confirm that methods are implemented to protect against unintended disclosure of sensitive data. Such methods may include usage of cryptography to protect the data, or the use of blinding or masking of cryptographic operations (where supported by the execution environment).	
	3.6.c Where protection methods require user input or interaction, the assessor shall examine evidence to confirm that guidance on the proper configuration and use of such methods is provided to stakeholders in accordance with Control Objective 12.1.	

Control Objectives	Test Requirements	Guidance
	3.6.d The assessor shall test the software to identify any sensitive data residue in the execution environment, and to confirm that protection methods are implemented correctly and the software does not expose or otherwise reveal sensitive data to unauthorized users.	

Software Protection Mechanisms

Software security controls are implemented to protect the integrity and confidentiality of critical assets.

Control Objectives	Test Requirements	Guidance
Control Objective 4: Critical Asset Protection Critical assets are protected from attack scenarios.		
4.1 Attack scenarios applicable to the software are identified. ***Note:*** *This control objective is an extension of Control Objective 10.1. Validation of both control objectives should be performed at the same time.*	**4.1.a** The assessor shall examine evidence to confirm that the software vendor has identified and documented relevant attack scenarios for the software.	Software vendors should evaluate the design of their payment software to identify attack scenarios applicable to the software and should document the results of that analysis. Documentation should describe the various aspects of the code that could be attacked (including tasks or actions that frameworks and libraries do on the software's behalf), the difficulty in mounting a successful attack, the mitigation techniques used to protect against such attacks, and the methodology used for measuring the likelihood and impact of each potential attack method. When the software relies on execution environment security controls, the software vendor should review and reference the implementation documentation for the platform (such as Security Policies for PCI-approved POI devices or FIPS140-2 or 140-3 approved cryptographic modules) and should confirm that the software and its associated documentation correctly and completely accommodate the guidance in these documents.
	4.1.b The assessor shall examine evidence to determine whether any specific industry-standard methods or guidelines were used to identify relevant attack scenarios. Where such industry standards are not used, the assessor shall confirm that the methodology used provides equivalent coverage for the attack scenarios applicable to the software under evaluation.	

Control Objectives	Test Requirements	Guidance
	4.1.c The assessor shall examine evidence to confirm the following: • A formal owner of the software is assigned. This may be a role for a specific individual or a specific name, but evidence must clearly show an individual who is accountable for the security of the software. • A methodology is defined for measuring the likelihood and impact for any exploit of the system. • Generic threat methods and types that may be applicable to the software are documented. • All critical assets managed, and all sensitive resources used by the system are documented. • All entry and egress points for sensitive data, as well as the authentication and trust model applied to each of these entry/egress points, are documented. • All data flows, network segments, and authentication/privilege boundaries are documented. • All static IPs, domains, URLs, or ports required by the software for operation are documented. • Considerations for cryptography elements like cipher modes, and protecting against relevant attacks such as timing attacks, padded oracles, brute force, “rainbow table” attacks, and dictionary attacks against the input domain are documented. • Execution environment implementation specifics or assumptions, such as network configurations and operating system security configurations, are documented. • Considerations for the software execution environment, the size of the install base, and the attack surfaces that must be mitigated are documented. Examples of such attack surfaces may include insecure user prompts or protocol stacks, or the storage of sensitive data post authorization or using insecure methods.	

Control Objectives	Test Requirements	Guidance
4.2 Software security controls are implemented to mitigate software attacks.	**4.2.a** The assessor shall examine evidence to confirm that one or more mitigation methods are defined for each of the threats identified in Test Requirement 4.1.a or that justification for the lack of mitigations is provided.	Once attack scenarios are identified, the risk of their occurrence should be mitigated. Software vendors should define and implement mechanisms to protect the software from attacks and reduce the likelihood and impact of successful execution. Any attack scenarios left unmitigated or insufficiently mitigated should be reasonably justified. The exact nature of the protection mechanism(s) will depend on the attack scenarios, the development platform, and the software-development languages, frameworks, libraries, and APIs used by the software, as well as the execution environment where the software is intended to be deployed. To minimize the software attack surface, the software should be developed using secure design principles such as layered defense, application segmentation and isolation (logical), and adaptive response. Examples of software security controls include input and output validation, authentication, parameterization, escaping, segmentation, logging, etc. For guidance on implementing cyber resiliency techniques and approaches, refer to industry standards and guidance such as the current version of *NIST Special Publication 800-160*.
	4.2.b Where any mitigations rely on settings within the software, the assessor shall test the software to confirm that such settings are applied by default upon installation, initialization, or first use of the software.	
	4.2.c Where user input or interaction can disable, remove, or bypass any such mitigations, the assessor shall examine evidence and test the software to confirm that such action requires authentication and authorization and that guidance on the risk of such actions is provided to stakeholders in accordance with Control Objective 12.1.	
	4.2.d When any mitigations rely on features of the execution environment, the assessor shall examine evidence to confirm that guidance is provided to stakeholders on how to enable such settings in accordance with Control Objective 12.1.	
	4.2.e Where the execution environment provides APIs to query the status of mitigation controls, the assessor shall test the software to confirm that software checks for these mitigations are in place and active prior to being launched and periodically throughout execution.	

Control Objectives	Test Requirements	Guidance
Control Objective 5: Authentication and Access Control The software implements robust authentication and access control methods to protect the confidentiality, integrity, and resiliency of critical assets.		
5.1 Access to critical assets is authenticated.	**5.1.a** The assessor shall examine evidence to confirm that authentication requirements are defined (i.e., type and number of factors) for all roles based on critical asset classification, the type of access (e.g., local, non-console, remote) and level of privilege. ***Note:*** *The assessor should refer to evidence obtained in the testing of Control Objective 1.3 to determine the classifications for all critical assets.*	Secure authentication ensures individual responsibility for actions and allows the software to maintain an effective audit trail of user activity. This expedites issue resolution and containment when the software is misused for malicious purposes. Authentication mechanisms should cover all non-public resources managed by or accessible through the software, as well as sensitive functions that can alter the software operation or impact the security of sensitive data and sensitive resources. Examples of authentication methods include: • Something you know, such as a password or passphrase. • Something you have, such as a token device or smart card. • Something you are, such as a biometric. To ensure that the implemented authentication mechanisms are adequate to address the risk of unauthorized access to sensitive data or sensitive resources, or misuse of a sensitive function, the vendor should analyze threats and identify the required level of authentication for all types of users and roles. For example, a user with limited access to sensitive data and sensitive resources could be required to perform authentication using a single authentication factor (for example, a password or a passphrase) while a user that is able to export the entire database might be required to perform multi-factor authentication. *(continued on next page)*
	5.1.b The assessor shall examine evidence and test the software to confirm that access to critical assets is authenticated and authentication mechanisms are implemented correctly.	
	5.1.c Where the software recommends, suggests, relies on, or otherwise supports the use of external mechanisms (such as third-party VPNs, remote desktop features, etc.) to provide secure non-console access to the system on which the software is executed or directly to the software itself, the assessor shall examine evidence to confirm that guidance on how to configure authentication mechanisms correctly is provided to stakeholders in accordance with Control Objective 12.1.	

Control Objectives	Test Requirements	Guidance
	5.1.d The assessor shall examine evidence to confirm that sensitive data associated with authentication credentials, including public keys, is identified as a critical asset.	Other factors such as the type of access (for example, local, non-console, or remote access) and the level of privilege (for example, the ability to invoke sensitive functions such as pause logging or change access privileges) may influence the level of authentication that should be required.
5.2 Access to critical assets requires unique identification.	**5.2.a** The assessor shall examine evidence and test the software to confirm that all implemented authentication methods require unique identification.	The software should not require the use of any group, shared, or generic accounts. The use of group or shared accounts makes it more difficult to determine which individuals execute specific actions since a given action could have been performed by anyone that has knowledge of the group or shared accounts' authentication credentials.
	5.2.b Where interfaces, such as APIs, allow for automated access to critical assets, the assessor shall examine evidence and test the software to confirm that unique identification of different programs or systems accessing the critical assets is required (for example, through use of multiple public keys) and that guidance on configuring a unique credential for each program or system is provided to stakeholders in accordance with Control Objective 12.1.	
	5.2.c Where identification is supplied across a non-console interface, the assessor shall test the software to confirm that authentication credentials are protected from attacks that attempt to intercept them in transit.	
	5.2.d The assessor shall examine evidence to confirm that the guidance provided to stakeholders per Control Objective 12.1 specifically notes that identification and authentication parameters must not be shared between individuals, programs, or in any way that prevents the unique identification of each access to a critical asset.	

Control Objectives	Test Requirements	Guidance
5.3 Authentication methods (including session credentials) are sufficiently strong and robust to protect authentication credentials from being forged, spoofed, leaked, guessed, or circumvented.	**5.3.a** Using information obtained in Test Requirement 4.1.a, the assessor shall examine evidence to confirm that authentication methods implemented by the software are evaluated to identify known vulnerabilities or attack methods involving the authentication method and how the implementation of these methods mitigates against such attacks. The assessor shall also confirm that the evidence examined demonstrates the implementation used in the software was considered. For example, a fingerprint may be uniquely identifiable to an individual, but the ability to spoof or otherwise bypass such technology can be highly dependent on the way the solution is implemented.	The software vendor must evaluate, document, and justify the usage of implemented authentication methods to demonstrate that they are sufficiently strong to protect authentication credentials in the software's intended specific use case or deployment scenario. For example, if the software uses biometric authentication, the vendor may want to identify all points at which a malicious user may attack the authenticator and implement mitigations to address those risks. The authentication mechanism implemented in the software could rely on additional sensors to ensure the provided biometric sample is from a living human and not a forged or spoofed sample. In some use cases or deployment scenarios, an authentication mechanism that relies on a single authentication method may not be sufficient. In such circumstances, the software vendor may want to implement additional mitigation strategies (for example, multi-factor authentication mechanism). To support a claim that the implemented authentication mechanism is sufficiently strong and robust, a vendor should adopt an industry-accepted methodology for assigning assurance levels (for example, *NIST SP800-63-3* and *NIST SP800-63B*).
	5.3.b The assessor shall examine evidence to confirm that the implemented authentication methods are robust and the robustness of the authentication methods was evaluated using industry-accepted methods. ***Note:*** *The vendor assessment and robustness justification include consideration of the full path of the user credentials, from any input source (such as a Human Machine Interface or other program), through transition to the execution environment of the software (including any switched/network transmissions and traversal through the execution environment's software stack before being processed by the software itself).*	
	5.3.c The assessor shall test the software to confirm that the authentication methods are implemented correctly and do not expose vulnerabilities.	

Control Objectives	Test Requirements	Guidance
5.4 By default, all access to critical assets is restricted to only those accounts and services that require such access.	**5.4.a** The assessor shall examine evidence to confirm that information is maintained that identifies and justifies the required access for all critical assets.	To ensure the software protects the confidentiality and integrity of critical assets, access privileges to those critical assets should be restricted based on vendor-defined access requirements. There are various approaches to implementing privilege restriction, such as trust-based privilege management, attribute-based usage restriction, and dynamic privileges. To reduce the attack surface of the software, the software authorization mechanisms might limit access to critical assets to only those accounts that need such access (the principle of "least privilege"). Other techniques include implementation of Role-Based Access Control (RBAC), Attribute-Based Access Control (ABAC), time-based adjustment to privilege, and dynamic revocation of access authorization.
	5.4.b The assessor shall examine evidence and test the software to identify the level of access that is provided to critical assets and to confirm that such access correlates with the evidence examined in Test Requirement 5.4.a. Testing to confirm access to critical assets is properly restricted should include attempts to access critical assets through user accounts, roles, or services which should not have the required privileges.	

Control Objectives	Test Requirements	Guidance
Control Objective 6: Sensitive Data Protection Sensitive data is protected at rest and in transit.		
6.1 Sensitive data is secured anywhere it is stored.	**6.1.a** The assessor shall examine evidence to confirm that protection requirements for all sensitive data are defined, including requirements for rendering sensitive data with confidentiality considerations unreadable anywhere it is stored persistently.	Sensitive data must be protected wherever it is stored. In some cases, the integrity may be the primary concern. In other cases, it may be the confidentiality of the sensitive data that must be protected. Sometimes, both the integrity and confidentiality must be secured. The type of data and the purpose for which it is generated will often determine the need for integrity or confidentiality protection. In all cases, those protection requirements must be clearly defined. In cases where the confidentiality of sensitive data is a concern, it is imperative to know where and for how long the data is retained. The vendor must have details of all locations where the software may store sensitive data, including in any underlying software or systems, and documentation detailing the security controls used to protect the data. Sensitive data requiring confidentiality protection, when stored persistently, must be protected to prevent malicious or accidental access. Examples of methods to render sensitive data unreadable include usage of a one-way hash or the use of strong cryptography with associated key-management processes. *(continued on next page)*
	6.1.b The assessor shall examine evidence and test the software to confirm that security controls are implemented to protect sensitive data during storage and that they address all defined protection requirements and identified attack scenarios. ***Note:*** *The assessor should refer to evidence obtained in the testing of Control Objective 1.1 to determine all sensitive data retained by the software, and Control Objective 4.1 to identify all attack scenarios applicable to the software.*	
	6.1.c Where cryptography is used for securing sensitive data, the assessor shall examine evidence and test the software to confirm that methods implementing cryptography for securing sensitive data comply with Control Objective 7.	
	6.1.d Where index tokens are used for securing sensitive data, the assessor shall examine evidence and test the software to confirm that these are generated in a way that ensures there is no correlation between the value and the sensitive data that is being referenced (without access to the software to perform the correlation as part of a formally defined and assessed feature of that software, such as “de-tokenization”).	

Control Objectives	Test Requirements	Guidance
	6.1.e Where protection methods rely on the security properties of the execution environment, the assessor shall examine evidence and test the software to confirm that these security properties are valid for all platforms where the software is intended to be deployed.	Where the integrity of sensitive data is a concern, strong cryptography with appropriate key-management practices is one method that could be used to satisfy integrity protection requirements during storage.
	6.1.f Where protection methods rely on the security properties of third-party software, the assessor shall examine evidence and test the software to confirm that there are no unmitigated vulnerabilities or issues with the software providing the security properties.	
6.2 Sensitive data is secured during transmission.	**6.2.a** The assessor shall examine evidence to identify the locations within the software where sensitive data is transmitted outside of the physical execution environment and to confirm protection requirements for the transmission of all sensitive data are defined.	To prevent malicious individuals from intercepting or diverting sensitive data while in transit, it must be protected during transmission. One method to protect sensitive data in transit is to encrypt it using strong cryptography prior to transmission. Alternatively, the software could establish an authenticated and encrypted channel using only trusted keys and certificates (for authentication) and appropriate encryption strength for the selected protocols.
	6.2.b The assessor shall examine evidence and test the software to confirm that for each of the ingress and egress methods that allow for transmission of sensitive data outside of the physical execution environment, the data is encrypted with strong cryptography prior to transmission or is transmitted over an encrypted channel using strong cryptography. ***Note:*** *The assessor should refer to evidence obtained in the testing of Control Objective 1.1 to determine the sensitive data stored, processed, or transmitted by the software.*	
	6.2.c Where third-party or execution-environment features are relied upon for the security of the transmitted data, the assessor shall examine evidence to confirm that guidance on how to configure such features is provided to stakeholders in accordance with Control Objective 12.1.	

Control Objectives	Test Requirements	Guidance
	6.2.d Where transport layer encryption is used to secure the transmission of sensitive data, the assessor shall examine evidence and test the software to confirm that all ingress and egress methods enforce a secure version of the protocol with end-point authentication prior to transmission.	
	6.2.e Where the methods implemented for encrypting sensitive data allow for the use of different types of cryptography or different levels of security, the assessor shall examine evidence and test the software, including capturing software transmissions, to confirm the software enforces the use of strong cryptography at all times during transmission.	
6.3 Use of cryptography meets all applicable cryptography requirements within this standard.	**6.3.a** Where cryptography is relied upon (in whole or in part) for the security of critical assets, the assessor shall examine evidence and test the software to confirm that the use of cryptography is compliant to Control Objective 7. ***Note:*** *The assessor should refer to Control Objective 7 to identify all requirements for appropriate and correct implementation of cryptography.*	Wherever cryptography is used to meet software security requirements in this standard, it must be done in accordance with the specific security requirements related to the use of cryptography (including those in Control Objective 7). For example, storing a cryptographic key used for protecting sensitive data in a plaintext file would not be considered sufficient security unless additional controls are implemented to prevent the file containing the cryptographic key from being accessed or modified by, or exposed to unauthorized parties. Further guidance on appropriate uses of cryptographic algorithms can be found in current versions of *NIST SP 800-175* or in other related industry guidance from ISO or ANSI.
	6.3.b Where cryptographic methods provided by third-party software or aspects of the execution environment or platform on which the application is run are relied upon for the protection of sensitive data, the assessor shall examine evidence and test the software to confirm that guidance on configuring these methods during the installation, initialization, or first use of the software is provided to stakeholders in accordance with Control Objective 12.1.	
	6.3.c Where asymmetric cryptography such as RSA or ECC is used for protecting the confidentiality of sensitive data, the assessor shall examine evidence and test the software to confirm that private keys are not used for providing confidentiality protection to the data.	

Control Objectives	Test Requirements	Guidance
Control Objective 7: Use of Cryptography Cryptography is used appropriately and correctly.		
7.1 Industry-standard cryptographic algorithms and methods are used for securing critical assets. Industry-standard cryptographic algorithms and methods are those recognized by industry-accepted standards bodies such as NIST, ANSI, ISO, and EMVCo. Cryptographic algorithms and parameters that are known to be vulnerable are not used.	**7.1.a** The assessor shall examine evidence to determine how cryptography is used for the protection of critical assets and to confirm that: • Industry-standard cryptographic algorithms and modes of operation are used. • The use of any other algorithms is in conjunction with industry-standard algorithms. • The implementation of non-standard algorithms does not reduce the equivalent cryptographic key strength provided by the industry-standard algorithms.	Not all cryptographic algorithms are sufficient to protect sensitive data. It is a well-established principle in software security to utilize only recognized cryptographic implementations based on current, industry-accepted standards such as those from industry bodies like NIST, ANSI, ISO, and EMVCo. The use of proprietary cryptographic implementations may increase the risk of data compromise as proprietary implementations are often not subjected to the same level of testing that industry-accepted implementations have undergone. Only those implementations that have been subjected to sufficient testing (for example, by NIST, ANSI, or other recognized industry bodies) should be used.
	7.1.b The assessor shall examine evidence, including the vendor threat information obtained in Test Requirement 4.1.a, and test the software to confirm that: • Only documented cryptographic algorithms and modes of operation are used in the software, and • Protection methods are implemented to mitigate common attacks on cryptographic implementations (for example, the use of the software as a decryption oracle, brute-force or dictionary attacks against the input domain of the sensitive data, the re-use of security parameters such as IVs, or the re-encryption of multiple datasets using linearly applied key values, such as XOR'd key values in stream ciphers or one-time pads).	
	7.1.c Where cryptographic implementations require a unique value per encryption operation or session, the assessor shall examine evidence and test the software to confirm that the cryptographic implementations do not expose vulnerabilities. For example, this may include the use of a unique IV for a stream cipher mode of operation or a unique and random "k" value for a digital signature.	

Control Objectives	Test Requirements	Guidance
	7.1.d Where padding is used prior to or during encryption, the assessor shall examine evidence and test the software to confirm that the encryption operation always incorporates an industry-accepted standard padding method.	
	7.1.e Where hash functions are used to protect sensitive data, the assessor shall examine evidence and test the software to confirm that: • Only approved, collision-resistant hash algorithms and methods are used for this purpose, and • A salt value of appropriate strength that is generated using a secure random number generator is used to ensure the resultant hash has sufficient entropy. ***Note:*** *The assessor should refer to Control Objective 7.3 for more information on secure random number generators.*	
7.2 The software supports industry-standard key management processes and procedures. Industry-standard key management processes and procedures are those recognized by industry standards bodies, such as NIST, ANSI, and ISO.	**7.2.a** The assessor shall examine evidence to confirm that information is maintained that describes the following for each key specified in the inventory: • Key label or name • Key location • Effective date • Expiration date • Key purpose/type • Key generation method/algorithm used • Key length	Whether implemented within or outside the software, the manner in which cryptographic keys are managed is a critical part of the continued security of payment software and the sensitive data it handles. While cryptographic key management processes are often implemented as operational procedures, the software should support secure key-management practices based on industry standards or best practices. *(continued on next page)*

Control Objectives	Test Requirements	Guidance
	7.2.b The assessor shall examine evidence and test the software to validate the evidence examined in Test Requirement 7.2.a and to confirm that: • All cryptographic keys that are used for providing security to critical assets (confidentiality, integrity, and authenticity) and other security services to the software have a unique purpose, and that no key is used for both encryption and authentication operations. • All keys have defined generation methods, and no secret or private cryptographic keys relied upon for security of critical assets are shared between software instances, except when a common secret or private key is used for securing the storage of other cryptographic keys that are generated during the installation, initialization, or first use of the software (for example, white-box cryptography). • All cryptographic keys have an equivalent bit strength of at least 128 bits in accordance with industry standards. • All keys have a defined cryptoperiod aligned with industry standards, and methods are implemented to retire and/or update each key at the end of the defined cryptoperiod. • The integrity and confidentiality of all secret and private cryptographic keys managed by the software are protected when stored (for example, encrypted with a key-encrypting key that is at least as strong as the data-encrypting key and is stored separately from the data-encrypting key, or as at least two full-length key components or key shares, in accordance with an industry-accepted method). • All keys have a defined generation or injection process, and this process ensures sufficient entropy for the key. • All key-generation functions must implement one-way functions or other irreversible key-generation processes, and no reversible key calculation modes (such as key variants) are used to directly create new keys from an existing key.	Industry-standard key management practices should be applied to the following: • The generation of strong cryptographic keys. • Secure cryptographic key distribution. • Secure cryptographic key storage. • Cryptographic key changes for keys that have reached the end of their cryptoperiod. • The retirement or replacement of keys. • The enforcement of split knowledge and dual control (when the software supports manual clear-text cryptographic key-management operations). • The prevention of unauthorized substitution of cryptographic keys. • The implementation of a mechanism to render irretrievable any cryptographic key material or cryptogram stored by the payment software. This requirement applies to keys used to encrypt sensitive data and any respective key-encrypting keys.

Control Objectives	Test Requirements	Guidance
	7.2.c Where cryptography is used to protect a key, the assessor shall examine evidence and test the software to confirm that security is not provided to any key by a key of lesser strength (for example, by encrypting a 256-bit AES key with a 128-bit AES key).	
	7.2.d Where public keys are used by the system, the assessor shall examine evidence and test the software to confirm that the authenticity of all public keys is preserved.	
	7.2.e Where public or white-box keys are not unique per software instantiation the assessor shall examine evidence to confirm that methods and procedures to revoke and/or replace such keys (or key pairs) exist.	
	7.2.f Where the software relies upon external files or other data elements for key material, such as for public TLS certificates, the assessor shall examine evidence to confirm that guidance on how to install such key material, including details noting any security requirements for such key material, is provided to stakeholders in accordance with Control Objective 12.1.	
	7.2.g Where public keys are manually loaded or used as root keys, the assessor shall examine evidence and test the software to confirm that the keys are installed and stored in a way that provides dual control (to a level that is feasible for the execution environment), preventing a single user from replacing a key to enable a man-in-the-middle attack or the allow for unauthorized decryption of stored data. Where complete dual control is not feasible (for example, due to a limitation of the execution environment), the assessor shall confirm that the methods implemented are appropriate to protect the public keys.	

Control Objectives	Test Requirements	Guidance
	7.2.h The assessor shall examine evidence to confirm that secret and/or private keys are managed in a way that ensures split knowledge over the key to a level that is feasible given the platform on which the software is executed. Where absolute split knowledge is not feasible, the assessor shall confirm that the methods implemented are reasonable to protect secrets and/or private keys.	
7.3 All random numbers used by the software are generated using only industry-standard random number generation (RNG) algorithms or libraries. Industry-standard RNG algorithms or libraries are those that meet industry standards for sufficient unpredictability (for example, *NIST Special Publication 800-22*).	**7.3.a** The assessor shall examine evidence and test the software to identify all random number generators used by the software and to confirm that all random number generation methods: • Use at least 128 bits of entropy prior to the output of any random numbers. • Ensure it is not possible for the system to provide or produce reduced entropy upon start-up or entry of other predictable states of the system.	Random numbers are often used with cryptography to protect sensitive information. Encryption keys and initialization values (seeds) are examples of implementations in which random numbers are required. It is not a trivial endeavor to design and implement a secure random number generator. Software vendors are required to use only approved random number generation algorithms and libraries or provide evidence to illustrate how the random number generation algorithms and libraries were tested to confirm that random numbers generated are sufficiently unpredictable. The implementation may rely on either a validated cryptographic library or module. The software vendor should have a good understanding of the installation, initialization, configuration, and usage (for example, initial seeding of the random function) of the RNG mechanisms to ensure that the implementation can meet the effective security strength required for the intended use.
	7.3.b Where third-party software, platforms, or libraries are used for all or part of the random number generation process, the assessor shall examine evidence (such as current publicly available literature) to confirm that the third-party software does not expose any vulnerabilities that may compromise its use for generating random values.	
	7.3.c Where the software vendor relies on a previous assessment of the random number generator or source of initial entropy, the assessor shall examine evidence (such as the approval records of the previous assessment) to confirm that this scheme and specific approval include the correct areas of the software in the scope of its assessment, and that the vendor claims do not exceed the scope of the evaluation or approval of that software. For example, some cryptographic implementations approved under FIPS 140-2 or 140-3 require seeding from an external entropy source to correctly output random data.	

Control Objectives	Test Requirements	Guidance
	7.3.d Where the software vendor does not rely on a previous assessment of the random number generator or source of initial entropy, the assessor shall test the software to obtain 128MB of data output from each random number generator implemented in the system to confirm the lack of statistical correlation in the output. This data may be generated by the assessor directly, or supplied by the vendor, but the assessor must confirm that the generation method implemented ensures that the data is produced as it would be produced by the software during normal operation. ***Note:*** *The assessor can use the NIST Statistical Test Suite to identify statistical correlation in the random number generation implementation.*	
7.4 Random values have entropy that meets the minimum effective strength requirements of the cryptographic primitives and keys that rely on them.	**7.4.a** The assessor shall examine evidence and test the software to confirm that the methods used for the generation of all cryptographic keys and other material (such as IVs, "k" values for digital signatures, and so on) have entropy that meets the minimum effective strength requirements of the cryptographic primitives and keys.	Entropy is the degree of randomness of a random value generator. The higher the entropy, the less predictable the next value in a random number generator is likely to be. *(continued on next page)*

Control Objectives	Test Requirements	Guidance
	7.4.b Where cryptographic keys are generated through processes which require direct user interaction, such as through the entry of a passphrase or the use of "random" user interaction with the software, the assessor shall examine evidence and test the software to confirm that these processes are implemented in such a way that they provide sufficient entropy. Specifically, the assessor shall confirm that: • Methods used for generating keys directly from a password/passphrase enforce an input domain that is able to provide sufficient entropy, such that the total possible inputs are at least equal to that of the equivalent bit strength of the key being generated (for example, a 32-hex-digit input field for an AES128 key). • Passphrases are passed through an industry-standard key-derivation function, such as PBKDF2 or bcrypt, which extends the work factor for any attempt to brute-force a passphrase value. The assessor shall confirm that a work factor of at least 10,000 is applied to any such implementation. • Guidance is provided to stakeholders in accordance with Control Objective 12.1 that includes instructions that any passphrase used must: – Be randomly generated itself using a valid and secure random process, and that an online random number generator must not be used for this purpose. – Not be implemented by a single person, such that one person has an advantage in recovering the clear key value, violating the requirements for split knowledge.	Note that a non-deterministic random number generator (NDRG) may produce an output string that contains less entropy than implied by the length of the output. A deterministic random number generator (DRNG) is dependent on the entropy of its seed value.

Secure Software Operations

The software provides mechanisms to detect and alert on anomalous activity and to ensure user accountability.

Control Objectives	Test Requirements	Guidance
Control Objective 8: Activity Tracking All software activities involving critical assets are tracked.		
8.1 All access attempts and usage of critical assets are tracked and traceable to a unique user. ***Note:*** *This Secure Software Standard recognizes that some execution environments cannot support the detailed logging requirements in other PCI standards. Therefore, the term "activity tracking" is used here to differentiate the expectations of this standard with regards to logging from similar requirements in other PCI standards.*	**8.1** The assessor shall examine evidence and test the software to confirm that all access attempts and usage of critical assets are tracked and traceable to a unique individual, system, or entity.	To ensure user accountability and to support post-incident forensic investigation, payment software should capture and maintain historical records of all software activities involving critical assets and ensure that all such activities are traceable to a unique user (for example, a person, system, or other entity). Examples of activities that the software should record include: • All individual user attempts to access sensitive data or resources. • Usage of or changes to sensitive functions, such as the software's identification and authentication mechanisms or activity tracking mechanisms. • Initialization, stopping, or pausing of sensitive functions. This control objective does not mandate the logging of each encryption operation or function processing sensitive data, but it does require that access is tracked and any methods that may expose sensitive data are also tracked.

Control Objectives	Test Requirements	Guidance
8.2 All activity is captured in sufficient and necessary detail to accurately describe the specific activities that were performed, who performed them, the time they were performed, and the critical assets that were affected.	**8.2.a** The assessor shall examine evidence and test the software to confirm that the tracking method(s) implemented capture specific activity performed, including: • Enablement of any privileged modes of operation. • Disabling of encryption of sensitive data. • Decryption of sensitive data. • Exporting of sensitive data to other systems or processes. • Failed authentication attempts. • Disabling or deleting a security control or altering security functions.	By recording the details in this requirement for all attempts to access or use critical assets, malicious activity or potential software or data compromise can be quickly identified and with sufficient detail to know who performed the activity, whether the attempt was successful, when the activity occurred, what critical assets were affected, and the origination of the event.
	8.2.b The assessor shall examine evidence and test the software to confirm that the tracking method(s) implemented provide the following: • A unique identification for the individual, system, or entity accessing or using critical assets. • A timestamp for each tracked event. • Details on what critical asset has been accessed.	
	8.2.c The assessor shall test the software to confirm that confidential data is not directly recorded in the tracking data.	
8.3 The software supports secure retention of detailed activity records.	**8.3.a** Where the activity records are managed by the software, including only temporarily before being passed to other systems, the assessor shall examine evidence and test the software to confirm that the protection methods are implemented to protect completeness, accuracy, and integrity of the activity records.	In order to identify anomalous behavior and to enable forensic investigation upon suspicion of potential software or data compromise, the software must provide for the retention of detailed activity records either through native means (within the software itself) or support integration with other solutions such as centralized log servers, cloud-based logging solutions, or back-end monitoring solutions. *(continued on next page)*

Control Objectives	Test Requirements	Guidance
	8.3.b Where the software utilizes external or third-party systems for the maintenance of tracking data, such as a log server, the assessor shall examine evidence to confirm that guidance on the correct and complete setup and/or integration of the software with the external or third-party system(s) is provided to stakeholders in accordance with Control Objective 12.1.	Without adequate protection of activity records, their completeness, accuracy, and integrity cannot be guaranteed and any reliance that would otherwise be placed on them (such as during a forensic investigation) would be negated. When activity records are managed by the software, the records must be protected in accordance with all applicable requirements for the protection of sensitive data.
	8.3.c The assessor shall test the software to confirm methods are implemented to secure the authenticity of the tracking data during transmission to the log storage system, and to confirm that this protection meets the requirements of this standard (for example, authenticity parameters must be applied using strong cryptography) and any account or authentication parameters used for access to an external logging system are protected.	
8.4 The software handles failures in activity-tracking mechanisms such that the integrity of existing activity records is preserved.	**8.4.a** The assessor shall examine evidence and test the software to confirm that the failure of the activity-tracking mechanism(s) does not violate the integrity of existing records by confirming that: • The software does not overwrite existing tracking data upon a restart of the software. Each new start shall only append to existing datasets or shall create a new tracking dataset. • Where unique dataset names are relied upon for maintaining integrity between execution instances, the implementation ensures that other software (including another instance of the same software) cannot overwrite or render invalid existing datasets. *(continued on next page)*	Software security controls should be implemented to ensure that when activity-tracking mechanism(s) fail, those failures are handled in a way that maintains the integrity of the records. Otherwise, attackers may intentionally target activity-tracking mechanisms and cause failures that would allow the attackers to conceal or overwrite evidence of their activities.

Control Objectives	Test Requirements	Guidance
	8.4.a • The software applies, where possible, suitable file privileges to assist with maintaining the integrity of the tracking dataset (such as applying an append-only access control to a dataset once created). Where the software does not apply such controls, the assessor shall confirm reasonable justification exists describing why this is the case, why the behavior is sufficient, and what additional mitigations are applied to maintain the integrity of the tracking data.	
	8.4.b The assessor shall examine evidence and test the software to confirm that the integrity of activity tracking records is maintained by: • Performing actions that should be tracked, force-closing and then restarting the software, and performing other tracked actions. • Performing actions that should be tracked, power-cycling the platform on which the software is executing, and then restarting the software and performing other tracked actions. • Locking access to the tracking dataset and confirming that the software handles the inability to access this dataset in a secure way, such as by creating a new dataset or preventing further use of the software. • Preventing the creation of new dataset entries by preventing further writing to the media on which the dataset is located (for example, by using media that has insufficient available space). Where any of the tests above are not possible, the assessor shall interview personnel to confirm reasonable justification exists to describe why this is the case and shall confirm protections are in place to prevent such scenarios from affecting the integrity of the tracking records.	

Control Objectives	Test Requirements	Guidance
Control Objective 9: Attack Detection Attacks are detected, and the impacts/effects of attacks are minimized.		
9.1 The software detects and alerts upon detection of anomalous behavior, such as changes in post-deployment configurations or obvious attack behavior.	**9.1.a** The assessor shall examine evidence and test the software to confirm that methods are implemented to validate the integrity of software executables and any configuration options, files, and datasets that the software relies upon for operation such that unauthorized post-deployment changes are detected. Where the execution environment prevents this, the assessor shall examine evidence (including publicly available literature on the platform and associated technologies) to confirm that there are indeed no methods for validating authenticity, and that additional security controls are implemented to minimize the associated risk.	Software should possess basic functionality to differentiate between normal and anomalous user behavior. Examples of anomalous behavior that should be automatically detected by the software include changes in post-deployment (or post-initialization) configurations or obvious automated-attack behaviors, such as repeated authentication attempts at a frequency that is infeasible for a human user. In some cases, it may be impractical to implement these capabilities directly into payment software, and third-party tools or services may be required. Where such tools or services are relied upon, the software vendor must provide guidance (or direction on where appropriate guidance may be obtained) that describes how and to what extent third-party tools and services should be configured to satisfy the control objective and associated test requirements.
	9.1.b The assessor shall examine evidence and test the software to confirm that integrity values used by the software and dataset(s) upon which it relies for secure operation are checked upon software execution, and at least every 36 hours thereafter (if the software continues execution during that time period).	
	9.1.c Where cryptographic primitives are used by any anomaly-detection methods, the assessor shall examine evidence and test the software to confirm that the cryptographic primitives are protected.	
	9.1.d Where stored values are used by any anomaly detection methods, the assessor shall examine evidence and test the software to confirm that these values are considered sensitive data and are protected accordingly.	
	9.1.e Where configuration or other dataset values can be modified by the software during execution, the assessor shall examine evidence and test the software to confirm that integrity protections are implemented to allow for this update while still ensuring dataset integrity can be validated after the update.	

Control Objectives	Test Requirements	Guidance
	9.1.f The assessor shall examine evidence and test the software to confirm that the software implements controls to prevent brute-force attacks on account, password, or cryptographic-key input fields (for example, input rate limiting).	
	9.1.g Where third-party tools or services are relied upon by the software to provide attack detection capabilities, the assessor shall examine evidence to confirm that guidance on how to configure such tools and services to support this control objective is provided to stakeholders in accordance with Control Objective 12.1.	

Secure Software Lifecycle Management

The software is maintained using secure software lifecycle management practices.

Control Objectives	Test Requirements	Guidance
Control Objective 10: Threat and Vulnerability Management Payment software threats and vulnerabilities are identified, assessed, and managed appropriately.		
10.1 Software threats and vulnerabilities are identified, assessed, and addressed.	**10.1.a** Using information obtained in Test Requirement 4.1.a, the assessor shall examine evidence to confirm that common attack methods against the software are identified. This may include platform-level, protocol-level, and/or language-level attacks.	Determining how to effectively secure and defend the software against attacks requires a thorough understanding of the specific threats and potential vulnerabilities applicable to the vendor's software. This typically involves understanding the following: • The types of information collected, stored, processed, or transmitted by the software. • The motivations an attacker may have for attacking the software. • The methods an attacker might utilize or the vulnerabilities an attacker might try to exploit during an attack. • The exploitability of any identified vulnerabilities. • The impact a successful attack. The identified threats and vulnerabilities should be tracked, assigned to responsible personnel, and fixed or mitigated prior to payment software release. For guidance on threat analysis and cyber-resiliency design principles, refer to industry standards and guidance, such as the current version of *NIST Special Publication 800-160*.
	10.1.b The assessor shall examine evidence to confirm that the identified attacks are valid for the software and shall note where this does not include common attack methods detailed in industry-standard references such as OWASP and CWE lists.	
	10.1.c The assessor shall examine evidence to confirm that mitigations against each identified attack are implemented, and that the software release process includes ongoing validation of the existence of these mitigations.	

Control Objectives	Test Requirements	Guidance
10.2 Vulnerabilities in the software and third-party components are tested for and fixed prior to release.	**10.2.a** The assessor shall examine evidence to confirm that robust testing processes are used throughout the software lifecycle to manage vulnerabilities in software and to verify that the mitigations used to secure the software against attacks remain in place and are effective.	Most software vulnerabilities are introduced as a result of coding errors, bad design, improper implementation, or the use of vulnerable components. Software should be developed and tested in a manner that minimizes the existence of any vulnerabilities and detects those that emerge over time, such that the vulnerabilities may be addressed before the software is released or updated. Techniques to avoid the introduction of vulnerabilities during development include the use of security coding practices, testing code during each phase of the development lifecycle using automated tools (such as static/dynamic analysis tools and interactive security testing tools), and the use of known secure components (for example, common code that has already undergone significant security vetting). To minimize the introduction of software vulnerabilities from third-party components, those components must also be evaluated. Ideally, they should be subject to the same secure development and testing processes as the software created by the vendor. Security testing should be performed by appropriately skilled vendor personnel or third parties. In addition, security testing personnel should be able to conduct tests in an objective manner and be authorized to escalate any identified vulnerabilities to appropriate management or development personnel, so they can be properly addressed.
	10.2.b The assessor shall examine evidence including documented testing processes and output of several instances of the testing to confirm that the testing process: • Includes, at a minimum, the use of automated tools capable of detecting vulnerabilities both in software code and during software execution. • Includes the use of security testing tools that are suitable for the software architecture, development languages, and frameworks used in the development of the software. • Accounts for the entire code base and detects vulnerabilities in third-party, open-source, or shared components and libraries. • Accounts for common vulnerabilities and attack methods. • Demonstrates a history of finding software vulnerabilities and remediating them prior to software release.	
	10.2.c Where evidence examined in Test Requirement 10.2.b shows the release of software with known vulnerabilities, the assessor shall examine further evidence to confirm that: • An industry-standard vulnerability-ranking system (such as CVSS) is used to classify/categorize vulnerabilities. • A remediation plan is maintained for all detected vulnerabilities that ensures vulnerabilities do not remain unmitigated for an indefinite period.	

Control Objectives	Test Requirements	Guidance
Control Objective 11: Secure Software Updates Software releases and updates to address vulnerabilities are provided in a secure and timely manner.		
11.1 Software updates to fix known vulnerabilities are made available to stakeholders in a timely manner.	**11.1.a** The assessor shall examine evidence to confirm that: • Reasonable criteria are defined for releasing software updates to fix security vulnerabilities. • Security updates are made available to stakeholders in accordance with the defined criteria.	Vulnerabilities in software should be fixed as soon as possible to enable software users and other stakeholders to address any risks before vulnerabilities in their payment systems and software can be exploited by attackers. Vulnerabilities should be addressed in a manner that is commensurate with the risk they pose to software users or other stakeholders. The most critical or severe vulnerabilities (those with the highest exploitability and the greatest potential impact to stakeholders) should be patched immediately, followed by those with moderate-to-low exploitability or potential impact. The criteria for determining how and when to make patches available to stakeholders should be defined and followed.
	11.1.b The assessor shall examine evidence, including update-specific security-testing results and details, to confirm that security updates are made available to stakeholders in accordance with the defined criteria. Where updates are not provided in accordance with the defined criteria, the assessor shall confirm that such instances are justified and reasonable.	
11.2 Software releases and updates are delivered in a secure manner that ensures the integrity of the software code.	**11.2.a** The assessor shall examine evidence to confirm that the method(s) by which the vendor releases software updates maintain the integrity of the software code during transmission and installation.	Security updates should include a mechanism within the update process to verify the update code has not been replaced or tampered with. Examples of integrity checks include, but are not limited to, checksums and digitally signed certificates (where implemented correctly). Verification could be implemented within the software itself or through guidance that is provided to stakeholders to direct them on the manual verification of software updates. In addition, the process of distributing updates and patches should prevent malicious individuals from intercepting the updates in transit, modifying them, and then redistributing them to unsuspecting stakeholders.
	11.2.b Where user input or interaction is required to validate the integrity of the software code, the assessor shall examine evidence to confirm that guidance on this process is provided to stakeholders in accordance with Control Objective 12.1.	
	11.2.c Where the integrity method implemented is not cryptographically secure, the assessor shall examine evidence to confirm that the software distribution method provides a chain of trust, such as through use of a TLS connection that provides compliant cipher−suite implementations.	

Control Objectives	Test Requirements	Guidance
	11.2.d The assessor shall examine vendor evidence to confirm that stakeholders are notified of software updates, and that guidance on how they may be obtained and installed is provided to stakeholders in accordance with Control Objective 12.1.	
	11.2.e The assessor shall examine evidence to confirm that stakeholders are notified when known vulnerabilities are detected in software that has not yet been updated with a fix. This includes vulnerabilities that may exist in third-party software and libraries used by the software. The assessor shall confirm that this process includes providing the stakeholders with suggested mitigations for any such vulnerabilities.	
	11.2.f The assessor shall examine evidence to confirm that the software update mechanisms cover all software, configuration files, and other metadata that may be used by the software for security purposes or which may in some way affect the security of the software.	

Control Objectives	Test Requirements	Guidance
Control Objective 12: Software Vendor Implementation Guidance The software vendor provides stakeholders with clear and thorough guidance on the secure implementation, configuration, and operation of the software.		
12.1 The software vendor provides stakeholders with clear and thorough guidance on the secure implementation, configuration, and operation of its payment software.	**12.1.a** The assessor shall examine evidence to confirm that the vendor creates and provides stakeholders, clear and sufficient guidance to allow for the secure installation, configuration, and use of the software.	When followed, the software vendor's implementation guidance provides assurance that the software and patches can be securely installed, configured, and maintained in a customer environment, and that all desired security functions are active and working as intended. The guidance should cover all options and functions available to software users that could affect the security of the software or the data it interacts with. The guidance should also include secure configuration options for any components provided with or supported by the software, such as external software and underlying platforms. Examples of configurable options include: • Changing default credentials and passwords. • Enabling and disabling software accounts, services, and features. • Changes in resource access permissions. • Integration with third-party cryptographic libraries, random number generators, and so on. The provided guidance should result in a secure configuration across all supported platforms and all configurable options.
	12.1.b The assessor shall examine evidence to confirm that the guidance: • Includes details on how to securely and correctly install any third-party software that is required for the operation of the vendor software. • Provides instructions on the correct configuration of the platform(s) on which the software is to be executed, including setting security parameters and installation of any data elements (such as certificates). • Includes instructions for key management (for example, the use of keys and how they are distributed, loaded, removed, changed, and destroyed.) • Does not instruct the user to disable security settings or parameters within the installed environment, such as anti-malware software or firewall or other network-level protection systems. • Does not instruct the user to execute the software in a privileged mode higher than what is required by the software. • Provides details on how to validate the version of the software and clearly indicates for which version(s) of the software the guidance is written. *(continued on next page)*	

Control Objectives	Test Requirements	Guidance
	12.1.b • Provides justification for any requirements in this standard that are to be assessed as not applicable. For each of these, the assessor shall confirm justification exists for why this is the case and shall confirm that it agrees with their understanding and the results of their software testing.	

Module A – Account Data Protection Requirements

Module Name	Overview	Control Objectives
Module A – Account Data Protection Requirements	Security requirements for software that stores, processes, or transmits account data.	**A.1**: Sensitive Authentication Data **A.2**: Cardholder Data Protection

Purpose and Scope

This section (hereinafter referred to as the "Account Data Protection Module") defines security requirements and assessment procedures for software that stores, processes, or transmits Account Data. For the purposes of this module, account data is defined as follows:

Account Data	
Cardholder Data includes:	**Sensitive Authentication Data includes:**
▪ Primary Account Number (PAN) ▪ Cardholder Name ▪ Expiration Date ▪ Service Code	▪ Full track data (magnetic-stripe data or equivalent on a chip) ▪ CAV2/CVC2/CVV2/CID ▪ PINs/PIN blocks

The primary account number (PAN) is the defining factor for cardholder data. If PAN is stored, processed, or transmitted or is otherwise present, the requirements in this module apply in addition to the Secure Software Core Requirements.

The table on the following page illustrates commonly used elements of cardholder data and sensitive authentication data, whether storage of that data is permitted or prohibited, and whether this data needs to be protected. This table is not meant to be exhaustive, but it is presented to illustrate the different types of requirements that apply to each data element.

		Data Element	Storage Permitted	Render Stored Data Unreadable per Control Objective A.2.3
Account Data	**Cardholder Data**	Primary Account Number (PAN)	Yes	Yes
		Cardholder Name	Yes	No
		Service Code	Yes	No
		Expiration Date	Yes	No
	Sensitive Authentication Data[2]	Full Track Data [3]	No	Cannot store per Control Objective A.1.1
		CAV2/CVC2/CVV2/CID[4]	No	Cannot store per Control Objective A.1.1
		PIN/PIN Block[5]	No	Cannot store per Control Objective A.1.1

Control Objectives A.2.2 and A.2.3 apply only to PAN. If PAN is stored with other elements of cardholder data, only the PAN must be rendered unreadable according to Control Objective A.2.3. Sensitive authentication data must not be stored after authorization, even if encrypted, unless the software is intended only for use by issuers or organizations that support issuing services. Only in those cases can sensitive authentication data be stored post-authorization.

2 *Sensitive authentication data must not be stored after authorization (even if encrypted).*

3 *Full track data from the magnetic stripe, equivalent data on the chip, or elsewhere.*

4 *The three- or four-digit value printed on the front or back of a payment card.*

5 *Personal identification number entered by cardholder during a card-present transaction, and/or encrypted PIN block present within the transaction message.*

Security Requirements

Control Objectives	Test Requirements	Guidance
Control Objective A.1: Sensitive Authentication Data Sensitive Authentication Data (SAD) is not retained after authorization.		
A.1.1 The software does not store sensitive authentication data after authorization (even if encrypted) unless the software is intended only for use by issuers or organizations that support issuing services.	**A.1.1** Using information obtained in Test Requirement 1.1.a in the Core Requirements section, the assessor shall examine evidence and test the software to identify all potential storage locations for Sensitive Authentication Data, and to confirm that the software does not store such data after transaction authorization is complete. This includes storage of SAD in temporary storage (such as volatile memory), semi-permanent storage (such as RAM disks), and non-volatile storage (such as magnetic and flash storage media). Where Sensitive Authentication Data is stored after authorization, the assessor shall examine evidence to confirm that the software is designed explicitly for issuing purposes or for use by issuers or organizations that support issuing services.	Sensitive authentication data consists of full track data, card validation code or value, and PIN data. Storage of sensitive authentication data after authorization is prohibited. This data is valuable to malicious individuals as it allows them to generate counterfeit payment cards and create fraudulent transactions. Testing should include at least the following types of files, as well as any other output generated by the payment software: • Incoming transaction data • All logs (for example, transaction, history, debugging, error) • History files • Trace files • Audio and image files (for example, digital voice and biometrics) • Non-volatile memory (including non-volatile cache) • Database schemas • Database contents

Control Objectives	Test Requirements	Guidance
Control Objective A.2: Cardholder Data Protection Stored cardholder data is protected.		
A.2.1 The software vendor provides guidance to stakeholders regarding secure deletion of cardholder data after expiration of defined retention period(s).	**A.2.1** The assessor shall examine evidence to confirm that guidance is provided to stakeholders in accordance with Control Objective 12.1 that details: • All locations where the software stores cardholder data. • How to securely delete cardholder data stored by the payment software, including cardholder data stored on underlying software or systems (such as in OS files or in databases). • How to configure the underlying software or systems to prevent the inadvertent capture or retention of cardholder data (for example, by system backup or restore points).	The software vendor must provide details of all locations where the software may store cardholder data, including in any underlying software or systems (such as OS, databases, and so on), as well as instructions for securely deleting the data from these locations once the data has exceeded any defined retention period(s). Stakeholders must also be provided with configuration details for the underlying systems that the software runs on to ensure these underlying systems are not capturing cardholder data without the stakeholder's knowledge. Stakeholders need to know how underlying systems could be capturing data from the software so they can either prevent it from being captured or ensure the data is properly protected.
A.2.2 The software provides features to restrict or otherwise mask all displays of PAN to the minimum number of digits required.	**A.2.2.a** The assessor shall examine evidence to confirm that the software provides features that enable responsible parties to restrict or otherwise mask the display of PAN to the minimum number of digits required to meet a defined business need. **A.2.2.b** The assessor shall examine evidence to confirm that all displays of PAN are completely masked by default, and that explicit authorization is required to display any digits of the PAN.	The display of full PAN on items such as computer screens, payment card receipts, logs, faxes, or paper reports can result in this data being obtained by unauthorized individuals and used fraudulently. The masking approach should always ensure that only the minimum number of digits is displayed as necessary to perform a specific business function. For example, if only the last four digits are needed to perform a business function, the software should provide features to mask the PAN so that individuals performing that function can view only the last four digits.

Control Objectives	Test Requirements	Guidance
	A.2.2.c Where user input or interaction is required to configure PAN masking features and options, the assessor shall examine evidence to confirm that guidance on how to configure these features/options is provided to stakeholders in accordance with Control Objective 12.1.	
	A.2.2.d The assessor shall test the software to confirm that all displays of PAN are completely masked by default, and that explicit authorization is required to display any element of the PAN.	
A.2.3 PAN is rendered unreadable anywhere it is stored (including data on portable digital media, backup media, and in logs) by using any of the following approaches: • Truncation (hashing cannot be used to replace the truncated segment of PAN). • Index tokens and pads (pads must be securely stored). • Strong cryptography with associated key-management processes and procedures.	**A.2.3.a** The assessor shall examine evidence and test the software to confirm that methods are implemented to render PAN unreadable anywhere it is stored using the following methods: • Truncation. • Index tokens and pads, with the pads being securely stored. • Strong cryptography, with associated key-management processes and procedures. ***Note**: The assessor should examine several tables, files, log files, and any other resources created or generated by the software to verify the PAN is rendered unreadable.*	Lack of protection of PANs can allow malicious individuals to view or download this data. The intent of truncation is that only a portion (not to exceed the first six and last four digits) of the PAN is stored. The intent of strong cryptography is that the encryption be based on an industry-tested and accepted algorithm (not a proprietary or "home-grown" algorithm) with strong cryptographic keys.
	A.2.3.b Where user input or interaction is required to configure methods to render PAN unreadable when stored, the assessor shall examine evidence to confirm that guidance on configuring these options is provided to stakeholders in accordance with Control Objective 12.1 and that the guidance includes the following: • Details of any configurable options for each method used to render cardholder data unreadable, and instructions on how to configure each method for all locations where cardholder data is stored. *(continued on next page)*	

Control Objectives	Test Requirements	Guidance
	A.2.3.b • A list of all instances where cardholder data may be output for storage outside of the payment application, and instructions that the implementing entity is responsible for rendering the PAN unreadable in all such instances. • Instruction that if debugging logs are ever enabled (for troubleshooting purposes) and they contain PAN, they must be protected, that debugging must be disabled as soon as troubleshooting is complete, and that debugging logs must be securely deleted when no longer needed.	
	A.2.3.c Where software creates both tokenized and truncated versions of the same PAN, the assessor shall examine evidence and test the software to confirm that the tokenized and truncated versions cannot be correlated to reconstruct the original PAN.	
	A.2.3.d Where software creates or generates files for use outside the software (for example, files generated for export or backup), including for storage on removable media, the assessor shall examine evidence and test the software to confirm that PAN is rendered unreadable.	
	A.2.3.e If the software vendor stores PAN for any reason (for example, because log files, debugging files, and other data sources are received from customers for debugging or troubleshooting purposes), the assessor shall examine evidence and test the software to confirm that PAN is rendered unreadable in accordance with this control objective.	

Module B – Terminal Software Requirements

Module Name	Overview	Control Objectives
Module B: Terminal Software Requirements	Security requirements for software intended for deployment and execution on PCI-approved POI devices.	**B.1**: Terminal Software Documentation **B.2**: Terminal Software Design **B.3**: Terminal Software Attack Mitigation **B.4**: Terminal Software Security Testing **B.5**: Terminal Software Implementation Guidance

Purpose and Scope

This section (hereinafter referred to as the "Terminal Software Module" or "this module") defines security requirements and assessment procedures for payment software and applications that rely on the security features of PCI-approved POI devices to protect payment data. Software applications that are developed explicitly for deployment and execution on PCI-approved POI devices that do not meet the definition of Firmware as defined in the *PCI PIN Transaction Security (PTS) Point-of-Interaction (POI) Modular Security Requirements* (hereinafter referred to as the "PCI PTS POI Standard") are in scope for the requirements in this module.

Background

PCI-approved POI devices provide a high degree of confidentiality and integrity protection for payment data and payment transactions through the implementation of strict physical and logical protection mechanisms. Software that is deployed and executed on PCI-approved POI devices must not degrade or adversely affect the protection mechanisms provided by the device. In addition, the software must not provide features or functions that could allow those protection mechanisms to be circumvented or rendered ineffective.

The requirements and assessment procedures defined in the Terminal Software Module have been developed to help ensure that terminal software protects payment data and does not introduce features, functions, or weaknesses that could enable an attacker to circumvent or render ineffective the protection mechanisms provided by the underlying PCI-approved POI devices upon which the software is intended to be deployed.

Considerations

Some assessment procedures in this module require examination of documentation describing the security features and functions of the underlying payment terminal. The terminal software vendor should work with their assessor(s), as well as the respective payment terminal vendors for each of the devices to be included as part of the terminal software evaluation, to identify and compile all device documentation needed for the terminal software evaluation. For more information about Secure Software assessment preparation and activities, refer to the *Secure Software Program Guide*.

Security Requirements

Control Objectives	Test Requirements	Guidance
Control Objective B.1: Terminal Software Documentation The software architecture is documented and includes diagrams that describe all software components and services in use and how they interact.		
B.1.1 The software vendor maintains documentation that describes all software components, interfaces, and services provided or used by the software.	**B.1.1** The assessor shall examine evidence to confirm that documentation is maintained that describes the software's overall design and function including, but not limited to, the following: • All third-party and open-source components, external services, and Application Programming Interfaces (APIs) used by the software. • All User Interfaces (UI) and APIs provided or made accessible by the software.	Software vendors should also maintain detailed documentation that clearly and effectively describes the overall design and function of its software, including all services (internal and external), components, and functions used and provided by the software, and how those services, components, and functions interact.
B.1.2 The software vendor maintains documentation that describes all data flows and functions that involve sensitive data. ***Note:*** *This control objective is an extension of Control Objectives 1.1 and 1.2. Validation of these control objectives should be performed at the same time.*	**B.1.2.a** The assessor shall examine evidence to confirm that documentation is maintained that describes all sensitive data flows including, but not limited to, the following: • All sensitive data stored, processed, or transmitted by the software. • All locations where sensitive data is stored, including both temporary and persistent storage locations. • How sensitive data is securely deleted from storage (both temporary and persistent) when no longer needed.	In addition to identifying the components, interfaces, and services exposed by the software, the software vendor should also maintain documentation that clearly identifies and describes the types of data stored, processed, and transmitted by the software; whether and how that data is shared between components and functions; and the protection mechanisms implemented or relied upon by the software to protect that data. This type of documentation clarifies how data is stored, processed, or transmitted by the software, with whom the data is shared, and how the software may be attacked to gain access to the software's critical assets.

Control Objectives	Test Requirements	Guidance
	B.1.2.b The assessor shall examine evidence to confirm that documentation is maintained that describes all functions that handle sensitive data including, but not limited to, the following: • All inputs, outputs, and possible error conditions for each function that handles sensitive data. • All cryptographic algorithms, modes of operation, and associated key management practices for all functions that employ cryptography for the protection of sensitive data.	
B.1.3 The software vendor maintains documentation that describes all configurable options that can affect the security of sensitive data.	**B.1.3** The assessor shall examine evidence to confirm that documentation is maintained that describes all configurable options provided or made available by the software that can impact the security of sensitive data including, but not limited to, the following: • All configurable options that could allow access to sensitive data. • All configurable options that could enable modification of any mechanisms used to protect sensitive data. • All remote access features, functions, and parameters provided or made available by the software. • All remote update features, functions, and parameters provided or made available by the software. • The default settings for each configurable option.	Software vendors should identify all configurable options available within their software, particularly those that control security features and functions. Configurable features must be considered as potential avenues for attacking the software. Where configurable options enable control over security features and functions, robust security controls should be implemented to protect the configurable security features from being misused. Additionally, all configurable options should be configured to their most secure settings by default in accordance with Control Objective 2.

Control Objectives	Test Requirements	Guidance
Control Objective B.2: Terminal Software Design The software does not implement any feature that enables the security features, functions, and characteristics of the payment terminal to be circumvented or rendered ineffective.		
B.2.1 The software is intended for deployment and operation on payment terminals (PCI-approved POI devices).	**B.2.1** The assessor shall examine evidence to determine the payment terminals upon which the software is to be deployed. For each of the payment terminals identified and included in the software assessment, the assessor shall examine the payment terminal's device characteristics and compare them with the following characteristics specified in the *PCI SSC's List of Approved PTS Devices* to confirm they match: • Model name/number • PTS approval number • Hardware version number • Firmware version number(s)	Payment terminals provide a high degree of confidentiality and integrity protection for payment data and payment transactions through the implementation of strict physical and logical protection mechanisms. Software that is deployed and executed on these payment terminals should use the approved features and functions provided by the payment terminal rather than implementing its own equivalent features or functions, to avoid exposing vulnerabilities or other weaknesses that could allow an attacker to circumvent or render ineffective the security features of the payment terminal.
B.2.2 The software uses only the external communication methods included in the payment terminal's PTS device evaluation. ***Note:*** *The payment terminal may provide an IP stack approved per the PTS Open Protocols module, or the device may provide serial ports or modems approved by the PTS evaluation to communicate transaction data encrypted by its PCI PTS SRED functions. Using any external communication methods not included in the PCI-approved POI device evaluation invalidates the PTS approval, and such use is prohibited for terminal software.*	**B.2.2.a** The assessor shall examine evidence (including source code) to determine whether the software supports external communications. **B.2.2.b** Where the software supports external communications, the assessor shall examine all relevant payment terminal documentation (including the payment terminal vendor's security guidance/policy) to determine which external communication methods were included in the payment terminal's PTS device evaluation. **B.2.2.c** The assessor shall examine evidence (including source code) to confirm that the software uses only the external communication methods included in the payment terminal's PTS device evaluation and does not implement its own external communication methods or IP stack.	To ensure software does not degrade or defeat the security mechanisms provided by the underlying payment terminal, the software must use the device-provided security features and functions in accordance with the payment terminal vendor's security guidance/policy. This is particularly true for external communication methods. Under no circumstances should the software provide its own communication methods (for example, VMs, IP stack, scripting languages, and so on) to control device-level interfaces. The introduction of any such function by the software could introduce new vulnerabilities or weaknesses that would allow malicious entities to circumvent the security protections provided by the payment terminal and degrade the overall security characteristics of both the software and the underlying device.

Control Objectives	Test Requirements	Guidance
B.2.2.1 Where the software relies on the Open Protocols feature of the payment terminal, the software is developed in accordance with the payment terminal vendor's security guidance/policy.	**B.2.2.1** The assessor shall examine all relevant payment terminal documentation (including the payment terminal vendor's security guidance/policy) and all relevant software vendor process documentation and software design documentation to confirm that the software is developed in accordance with the payment terminal vendor's security guidance/policy.	Payment terminal vendor security guidance/policy is intended for application developers, system integrators, and end-users of the platform to meet the PCI PTS POI Open Protocol (as well as other PTS) requirements as part of a PCI-approved POI device evaluation.
B.2.2.2 The software does not circumvent, bypass, or add additional services or protocols to the Open Protocols of the payment terminal as approved and documented in the payment terminal vendor's security guidance/policy. This includes the use of: • Link Layer protocols • IP protocols • Security protocols • IP Services	**B.2.2.2** The assessor shall examine evidence (including source code) to confirm that the software does not circumvent, bypass, or add additional services or protocols to the Open Protocols of the payment terminal as approved and documented in the payment terminal vendor's security guidance/policy. This includes the use of: • Link Layer protocols • IP protocols • Security protocols • IP Services	The Open Protocol requirements in the *PCI PTS POI Standard* ensure that open protocols and services in payment terminals do not have vulnerabilities that can be remotely exploited and yield access to sensitive data or sensitive resources in the payment terminal. The payment terminal vendor defines what protocols and services are supported by the payment terminal and provides guidance to their use. Adding or enabling additional services or protocols or failing to follow the issued payment terminal vendor's security guidance/policy invalidates the approval status of that device for that implementation.
B.2.3 The software does not bypass or render ineffective any encryption methods or account data security methods implemented by the payment terminal in accordance with the payment terminal vendor's security guidance/policy.	**B.2.3.a** The assessor shall examine evidence (including source code) to determine whether the software provides encryption of sensitive data. Where the software does provide such a function, the assessor shall confirm the software does not bypass or render ineffective any encryption methods or account data security methods implemented by the payment terminal as follows:	Payment terminals are designed to provide robust cryptographic and key management functions. For example, PCI PTS POI-approved devices are verified to meet stringent requirements for loading, managing, and protecting cryptographic keys. Software that provides its own data encryption methods must not include methods that would enable an attacker to bypass or render ineffective the encryption methods implemented by the payment terminal and required by the payment terminal vendor's security guidance/policy.

Control Objectives	Test Requirements	Guidance
	B.2.3.b The assessor shall examine all relevant payment terminal documentation (including payment terminal vendor security guidance/policy)to determine which encryption methods are provided by the payment terminal.	
	B.2.3.c The assessor shall examine evidence (including source code) to confirm that the software does not bypass or render ineffective any encryption methods provided by the payment terminal in accordance with the payment terminal vendor's security guidance/policy.	
	B.2.3.d Where the software provides encryption of sensitive data, but the payment terminal is not required to provide approved encryption methods (per the *PCI PTS POI Standard*), the assessor shall examine evidence (including source code) to confirm that the encryption methods used or implemented by the software for encrypting sensitive data provide "strong cryptography" and are implemented in accordance with Control Objectives 7.1 and 7.2.	
B.2.4 The software uses only the random number generation function(s) included in the payment terminal's PTS device evaluation for all cryptographic operations involving sensitive data or sensitive functions where random values are required and does not implement its own random number generation function(s).	**B.2.4.a** The assessor shall examine evidence (including source code) to determine whether the software requires random values to be generated for any cryptographic operations involving sensitive data or sensitive functions.	The unpredictability of random numbers is of critical importance to ensure the effectiveness of cryptographic operations. It is not a trivial endeavor to design and implement a secure random number generator. For this reason, the terminal software should only use the random number generation function(s) implemented by the payment terminal for all cryptographic operations involving sensitive data or sensitive functions where random values are required.
	B.2.4.b Where the software requires random values for cryptographic operations involving sensitive data or sensitive functions, the assessor shall examine all relevant payment terminal documentation (including payment terminal vendor security guidance/policy) to determine all of the random number generation functions included in the payment terminal's PTS device evaluation.	

Control Objectives	Test Requirements	Guidance
	B.2.4.c The assessor shall examine evidence (including source code) to confirm that the software uses only the random number generation function(s) included in the payment terminal's PTS device evaluation for all cryptographic functions involving sensitive data or sensitive functions where random values are required and does not implement its own random number generation function(s).	
B.2.5 The software does not provide, through its own logical interface(s), the sharing of clear-text account data directly with other software. ***Note:*** *The software is allowed to share clear-text account data directly with the payment terminal's firmware.*	**B.2.5.a** The assessor shall examine evidence (including source code) to determine all logical interfaces of the software, including: • All logical interfaces and the purpose and function of each. • The logical interfaces intended for sharing clear-text account data, such as those used to pass clear-text account data back to the approved firmware of the payment terminal. • The logical interfaces not intended for sharing of clear-text account data, such as those for communication with other software.	Many payment terminals provide mechanisms for the secure reading and exchange of data (SRED). These mechanisms are rigorously tested as part of the payment terminal's PTS device evaluation to confirm that the confidentially and integrity of clear-text account data is maintained during information exchange with the payment terminal's firmware. Software that provides its own mechanisms for sharing clear-text account data directly with other software is more likely to be prone to attacks and the unintended or unauthorized disclosure of clear-text account data than software that uses the payment terminal-provided SRED (or similar) functions.
	B.2.5.b The assessor shall examine evidence (including source code) to confirm that the software does not allow sharing of clear-text account data directly with other software through its own logical interfaces.	
	B.2.5.c The assessor shall install and configure the software in accordance with the guidance required in Control Objectives 12.1 and B.5.1. Using an appropriate "test platform" and suitable forensic tools and/or methods, the assessor shall test the software using all software functions that handle account data to confirm that the software does not allow the sharing of clear-text account data directly with other software through its own logical interfaces.	

Control Objectives	Test Requirements	Guidance
B.2.6 The software uses and/or integrates all shared resources securely and in accordance with the payment terminal vendor's security guidance/policy.	**B.2.6.a** The assessor shall examine evidence (including source code) to determine whether and how the software connects to and/or uses any shared resources provided by the payment terminal, and to confirm that: • The guidance required in Control Objectives 12.1 and B.5.1 includes detailed instructions for how to configure the software to ensure secure integration with shared resources. • The required guidance for secure integration with shared resources is in accordance with the payment terminal vendor's security guidance/policy.	Where the software uses or integrates shared resources provided by the payment terminal, the software must use or integrate resources in accordance with the payment terminal vendor's guidance/policy. Failure to use such shared resources in accordance with payment terminal guidance puts any sensitive data shared with such resources at greater risk of unauthorized disclosure.
	B.2.6.b The assessor shall install and configure the software in accordance with the guidance required in Control Objectives 12.1 and B.5.1. Using an appropriate "test platform" and suitable forensic tools and/or methods, the assessor shall test the software using all software functions that use or integrate shared resources to confirm that any connections to or use of shared resources are handled securely.	

Control Objectives	Test Requirements	Guidance
B.2.7 The software does not bypass or render ineffective any application segregation enforced by the payment terminal.	**B.2.7.a** The assessor shall examine all relevant payment terminal documentation (including the payment terminal vendor's security guidance/policy) to determine whether and how application segregation is enforced by the payment terminal.	Many payment terminals enforce logical separation between software applications. In the context of this module, software applications are logical entities that do not meet the PTS definition of "firmware." Logical application segmentation controls are intended to prevent one application on the payment terminal from interfering or tampering with other applications. However, these logical segregation controls are not intended to prevent applications from sharing data. They are mainly intended to prevent applications from modifying the structure or function of other applications or the payment terminal's firmware. To preserve the integrity of payment terminal application-segregation controls, all terminal software should adhere to those segregation controls and not include or introduce any function(s) that would allow the software to be used (intentionally or unintentionally) to bypass or defeat device-level application segregation.
	B.2.7.b The assessor shall examine evidence (including source code) to confirm that the software does not introduce any function(s) that would allow it to bypass or defeat any device-level application segregation controls.	
B.2.8 All software files are cryptographically signed to enable cryptographic authentication of the software files by the payment terminal firmware.	**B.2.8.a** The assessor shall examine the guidance required in Control Objectives 12.1 and B.5.1 to confirm that it includes detailed instructions for how to cryptographically sign the software files in a manner that enables the cryptographic authentication of all such files by the payment terminal.	To support cryptographic authentication of software files by the payment terminal, software vendors must cryptographically "sign" all of the software files (including all binaries, libraries, and configuration files) using digital certificates where the payment terminal vendor is included in the certificate chain. Additionally, the cryptographic signing process should incorporate the use of a secure cryptographic device (SCD), typically provided by the payment terminal vendor. Cryptographic signing should also be performed under dual control to protect the integrity of all cryptographic keys, software files, and the cryptographic signing process in general.
	B.2.8.b The assessor shall install and configure the software in accordance with the guidance required in Control Objectives 12.1 and B.5.1. Using an appropriate "test platform" and suitable forensic tools and/or methods, the assessor shall confirm that all software files are cryptographically signed in a manner that enables the cryptographic authentication of all software files.	

Control Objectives	Test Requirements	Guidance
	B.2.8.c Where the software supports the loading of files outside of the base software package(s), the assessor shall examine evidence and test the software to determine whether each of those files is cryptographically signed in a manner that enables the cryptographic authentication of those files by the payment terminal. For any files that cannot be cryptographically signed, the assessor shall justify why the inability to cryptographically sign such files does not adversely affect the security of the software or the underlying payment terminal.	
	B.2.8.d The assessor shall examine evidence (including source code) to determine whether and how the software supports EMV® payment transactions. Where EMV payment transactions are supported by the software, the assessor shall install and configure the software in accordance with the guidance required in Control Objectives 12.1 and B.5.1. Using an appropriate "test platform" and suitable forensic tools and/or methods, the assessor shall confirm that all EMV Certification Authority Public Keys are cryptographically signed in a manner that enables the cryptographic authentication of those files by the payment terminal.	Where terminal software supports EMV payment transactions, the EMV Certificate Authority public keys should also be signed and cryptographically authenticated using the same methods and procedures as the terminal software files.
B.2.9 The integrity of software prompt files is protected in accordance with Control Objective B.2.8.	**B.2.9.a** The assessor shall examine evidence (including source code) to determine whether the software supports the use of data entry prompts and/or prompt files. Where the software supports such features, the assessor shall confirm the software protects the integrity of those prompts as defined in Test Requirements B.2.9.b through B.2.9.c.	Sensitive data (including PIN and other account data) captured and handled by the software and underlying payment terminal is often controlled using prompt files. Prompt files are configuration files that control software display prompts. To preserve the integrity of the prompts, prompt files should be stored and managed securely. Anywhere clear-text data entry is allowed by the software, prompt controls should be implemented. *(continued on next page)*

Control Objectives	Test Requirements	Guidance
	B.2.9.b The assessor shall examine the guidance required in Control Objectives 12.1 and B.5.1 to confirm that it includes detailed instructions for stakeholders to cryptographically sign all prompt files in a manner that enables the cryptographic authentication of all such files in accordance with B.2.8.	Many prompt files are stored within a secure boundary of the device, such as a Secure Chip or Secure Element or within a Trusted Execution Environment. When prompt files are to be maintained in shared storage locations, the files should be cryptographically signed and authenticated by the payment terminal prior to installation or execution.
	B.2.9.c The assessor shall install and configure the software in accordance with the guidance required in Control Objectives 12.1 and B.5.1. Using an appropriate "test platform" and suitable forensic tools and/or methods, the assessor shall confirm that all prompt files are cryptographically signed in a manner that enables the cryptographic authentication of those files by the payment terminal in accordance with B.2.8.	

Control Objectives	Test Requirements	Guidance
Control Objective B.3: Terminal Software Attack Mitigation Software security controls are implemented to mitigate software attacks.		
B.3.1 The software validates all user and other external inputs. ***Note:*** *Control Objectives B.3.1 through B.3.3 are extensions of Control Objective 4.2. Validation of these control objectives should be performed at the same time.*	**B.3.1.a** The assessor shall examine evidence (including source code) to identify all locations where the software accepts input data from untrusted sources. For each instance, the assessor shall confirm that input data is required to conform to a list of expected characteristics and that all input that does not conform to the list of expected characteristics is rejected by the software or otherwise handled securely.	Any terminal software functions that accept externally supplied data (directly or indirectly) is a potential attack vector, particularly when the data is processed by an interpreter. Injection attacks are common for almost all types of software and are intended to manipulate input data in a way that causes software to behave unexpectedly or unintentionally. For example, software that accepts externally supplied information, such as a file name or file path to construct a search command, can be easily manipulated to disclose information about sensitive files and resources that were never intended to be accessed through the software interface. To protect against this and other types of injection attacks, all input data should be validated, filtered, and/or sanitized before the information is sent to any interpreter. Inputs for terminal software tend to involve simple commands and data. Therefore, all terminal software input data should be validated against a defined and restricted set of acceptable values before passing the data to any command interpreter. Any data that is not explicitly identified as an acceptable value or an acceptable range of values should be rejected.
	B.3.1.b The assessor shall install and configure the software in accordance with the guidance required in Control Objectives 12.1 and B.5.1. Using an appropriate "test platform" and suitable forensic tools and/or methods, the assessor shall test the software by attempting to supply each user or other external input with invalid or unexpected characteristics to confirm that the software validates all inputs and either rejects or securely handles all unexpected characteristics.	
B.3.1.1 All string values are validated by the software.	**B.3.1.1.a** The assessor shall examine evidence (including source code) to identify all terminal software functions where string values are passed as inputs, and to confirm that all strings are checked for text or data that can be erroneously or maliciously interpreted as a command.	Externally supplied inputs that can be interpreted as commands are particularly susceptible to injection attacks. Even if externally supplied inputs are processed or transformed in some way (for example, augmented with additional data or sanitized), they may still be susceptible. *(continued on next page)*

Control Objectives	Test Requirements	Guidance
	B.3.1.1.b The assessor shall install and configure the software in accordance with the guidance required in Control Objectives 12.1 and B.5.1. Using an appropriate "test platform" and suitable forensic tools and/or methods, the assessor shall test the software by attempting to supply each of the identified functions with data that includes commands to confirm that the software either rejects such inputs or otherwise handles such inputs securely.	Therefore, all inputs that can be interpreted as commands must be handled securely so that the execution of any constructed commands is controlled, as opposed to blindly executing whatever commands are included in the string.
B.3.1.2 The software checks inputs and rejects or otherwise securely handles any inputs that violate buffer size or other memory allocation thresholds.	**B.3.1.2.a** The assessor shall examine evidence (including source code) to identify all software functions that handle buffers and process data supplied from untrusted sources. For each of the noted functions, the assessor shall confirm that each of the identified functions: • Uses only unsigned variables to define buffer sizes. • Conducts checks to confirm that buffers are sized appropriately for the data they are intended to handle, including consideration for underflows and overflows. • Rejects or otherwise securely handles any inputs that violate buffer size or other memory allocation thresholds.	Payment terminals and terminal software often leverage low-level programming languages, such as C and C++. These languages allow the software to directly manipulate OS-level or hardware-level features and functions. Using low-level programming languages offers many benefits but also has several drawbacks. Low-level programming languages are susceptible to attacks that use low-level characteristics to manipulate the software or the underlying hardware. Buffer overflows and underflows are examples of these types of attacks. To protect against buffer overflow attacks, all terminal software functions that define or control buffer sizes should compare the amount of data intended for those buffers with the buffer size. Data that violates buffer size thresholds (overflows and underflows) should be rejected or otherwise handled securely.
	B.3.1.2.b The assessor shall install and configure the software in accordance with the guidance required in Control Objectives 12.1 and B.5.1. Using an appropriate "test platform" and suitable forensic tools and/or methods, the assessor shall test the software by attempting to supply each noted function with inputs that violate buffer size thresholds to confirm that the software either rejects or securely handles all such attempts.	

Control Objectives	Test Requirements	Guidance
B.3.2 Return values are checked, and error conditions are handled securely.	**B.3.2.a** Using information obtained in Test Requirement 1.2.a, the assessor shall examine evidence (including source code) to identify all software functions that handle sensitive data. For each of the noted software functions, the assessor shall confirm that each function: • Checks return values for the presence of sensitive data. • Processes the return values in a way that does not inadvertently "leak" sensitive data.	Another common technique used by attackers to compromise sensitive data that is stored, processed, or transmitted by software is to manipulate the software in a way that generates unhandled exceptions. Unhandled exceptions are error conditions that the software vendor has not anticipated and, therefore, has not factored into the software design. If an attacker can manipulate a software function that is known to handle sensitive data in a way that generates a condition that the software does not handle properly, it is possible that the software may output an error that includes sensitive data. To protect against attacks involving unhandled exceptions, all terminal software functions handling sensitive data should include processes or routines that instruct the software how to treat unknown exceptions. These processes should determine what information to include in any error codes or values. The disclosure of sensitive data through error conditions or error reporting, whether intentional or accidental, should be avoided.
	B.3.2.b The assessor shall install and configure the software in accordance with the guidance required in Control Objectives 12.1 and B.5.1. Using an appropriate "test platform" and suitable forensic tools and/or methods, the assessor shall test each software function that handles sensitive data by attempting to manipulate the software in a manner that generates an unhandled exception to confirm that error conditions do not expose sensitive data.	
B.3.3 Race conditions are avoided.	**B.3.3.a** The assessor shall examine evidence (including source code) to identify all software functions that rely on synchronous processing. For each of the noted functions, the assessor shall confirm that protection mechanisms have been implemented in the software to mitigate race conditions.	Race conditions can arise when the software requires sequential processing of data to perform some software function. For example, a "time-of-use, time-of-check race condition" exists when a file is checked at one point and used immediately after, with the assumption that the previous check is still valid. This assumption may not be correct if the system allows the file to be modified in between. If an attacker can identify and manipulate the software to take advantage of a race condition, they may be able to execute arbitrary code or generate other conditions that the attacker could exploit further.

Control Objectives	Test Requirements	Guidance
	B.3.3.b The assessor shall install and configure the software in accordance with the guidance required in Control Objectives 12.1 and B.5.1. Using an appropriate “test platform” and suitable forensic tools and/or methods, the assessor shall test each software function that relies on synchronous processing by attempting to generate a race condition (such as through specially crafted attacks intended to exploit the timing of synchronous events) to confirm that the software is resistant to such attacks.	To protect against race conditions, protection mechanisms should be implemented by the terminal software to control sequential processing more tightly. Using the example described above, a “locking” mechanism could be used to prevent updates to the file until the file can be processed completely. Regardless of the methods used, any terminal software that requires sequential processing of data for its operation should implement protections to avoid race conditions.

Control Objectives	Test Requirements	Guidance
Control Objective B.4: Terminal Software Security Testing The software is tested rigorously for vulnerabilities prior to each release.		
B.4.1 A documented process is maintained and followed for testing software for vulnerabilities prior to each update or release. ***Note:*** *This control objective is an extension of Control Objective 10.2. Validation of these control objectives should be performed at the same time.*	**B.4.1.a** The assessor shall examine evidence to confirm that the software vendor maintains a documented process in accordance with Control Objective 10.2 for testing the software for vulnerabilities prior to each update or release, and that the documented process includes detailed descriptions of how the vendor tests for the following: • The presence or use of any unnecessary ports and protocols. • The unintended storage, transmission, or output of any clear-text account data. • The presence of any default user accounts with default or static access credentials. • The presence of any hard-coded authentication credentials in code or in configuration files. • The presence of any test data or test accounts. • The presence of any faulty or ineffective software security controls.	Many software vulnerabilities are the result of the software vendor's failure to remove test functions or data. These lingering functions and data can provide an attacker with a path to compromise the software. Before software is released to the public, it must be tested to confirm that test functions and data are not included in the release version. Examples of such functions and data that must be explicitly removed prior to release include: • Any communication ports or protocols that are not absolutely required for software operation. • Any functions that allow the unintended storage, transmission, or output of any clear-text account data. • Any hard-coded authentication credentials in code or configuration files. • Any test data or test user accounts. • Any faulty or ineffective software security controls and protection mechanisms.
	B.4.1.b The assessor shall examine evidence to confirm that the software is tested for vulnerabilities prior to each release and that the testing covers the following: • The presence or use of any unnecessary ports and protocols. • The unintended storage, transmission, or output of any clear-text account data. *(continued on next page)*	

Control Objectives	Test Requirements	Guidance
	B.4.1.b • The presence of any default user accounts with static access credentials. • The presence of any hard-coded authentication credentials in code or in configuration files. • The presence of any test data or test accounts. • The presence of any faulty or ineffective software security controls.	

Control Objectives	Test Requirements	Guidance
Control Objective B.5: Terminal Software Implementation Guidance The software vendor provides stakeholders with clear and thorough guidance on the secure implementation, configuration, and operation of the software on applicable payment terminals.		
B.5.1 The software vendor provides implementation guidance on how to implement and operate the software securely for the payment terminals on which it is to be deployed. ***Note**: This control objective is an extension of Control Objective 12.1. Validation of these control objectives should be performed at the same time.*	**B.5.1** The assessor shall examine evidence to confirm that guidance on how to securely implement and operate the software for all applicable payment terminals is provided to stakeholders in accordance with Control Objective 12.1.	Because many security features used by terminal software are provided by the underlying payment terminal, the terminal software vendor should include instructions in its implementation guidance on how to configure all the available security features of both the terminal software and underlying payment terminal where applicable.
B.5.1.1 Implementation guidance includes detailed instructions for how to configure all available security options and parameters of the software.	**B.5.1.1** The assessor shall examine evidence to confirm that the required guidance includes detailed instructions on how to configure all available security options and parameters of the software in accordance with Control Objective B.1.3.	
B.5.1.2 Implementation guidance includes detailed instructions for how to securely configure the software to use the security features and functions of the payment terminal where applicable.	**B.5.1.2** The assessor shall examine evidence to confirm that the required guidance includes detailed instructions on how to securely configure the software to use the security features and functions of the payment terminal where applicable.	
B.5.1.3 Implementation guidance includes detailed instructions for how to configure the software to securely integrate or use any shared resources provided by the payment terminal.	**B.5.1.3** The assessor shall examine evidence to confirm that the required guidance includes detailed instructions on how to configure the software to securely integrate or use any shared resources provided by the payment terminal in accordance with Control Objective B.2.6.	

Control Objectives	Test Requirements	Guidance
B.5.1.4 Implementation guidance includes detailed instructions on how to cryptographically sign the software files in a manner that enables the cryptographic authentication of all such files by the payment terminal.	**B.5.1.4** The assessor shall examine evidence to confirm that the required guidance includes detailed instructions on how to cryptographically sign the software files in a manner that enables the cryptographic authentication of all such files by the payment terminal in accordance with Control Objective B.2.8.	
B.5.1.5 Implementation guidance includes instructions for stakeholders to cryptographically sign all prompt files.	**B.5.1.5** The assessor shall examine evidence to confirm that the required guidance includes detailed instructions for stakeholders to cryptographically sign all prompt files in accordance with Control Objective B.2.9.	
B.5.2 Implementation guidance adheres to payment terminal vendor guidance on the secure configuration of the payment terminal.	**B.5.2** The assessor shall examine evidence (including the payment terminal vendor's security guidance/policy and the guidance required in Control Objective B.5.1) to confirm that the guidance aligns with the payment terminal vendor's security guidance/policy.	Software implementation guidance must exclude instructions that conflict with the guidance and recommendations of the payment terminal vendor. Software implementation guidance must align with the payment terminal vendor's security guidance/policy. Otherwise, software users who rely on the software vendor for instructions may unknowingly configure the software and/or the underlying payment terminal improperly.

Module C – Web Software Requirements

Module Name	Overview	Control Objectives
Module C: Web Software Requirements	Additional security requirements for payment software that uses Internet technologies, protocols, and languages to initiate or support electronic payment transactions.	**C.1**: Web Software Components & Services **C.2**: Web Software Access Controls **C.3:** Web Software Attack Mitigation **C.4**: Web Software Communications

Purpose and Scope

This section (hereinafter referred to as the "Web Software Module" or "this module") defines security requirements and assessment procedures for payment software and applications that use Internet technologies, protocols, and languages for the purposes of initiating or supporting electronic payment transactions. This includes both traditional (monolithic) and cloud-native payment applications, APIs, web services, microservices, serverless functions, GRPC, and any other methods used to make payment functions accessible or to conduct electronic payment transactions over the Internet. Any software-based features or functions that handle requests from Internet "clients" and generate responses to initiate or support an electronic payment transaction are in scope for the requirements in this module.

Considerations

Web software architectures can be extremely complex and involve features and functions that are provided by different entities and may be distributed across different geographic locations. The security issues affecting web software can vary significantly. The security requirements defined within this web module do not address all risks affecting web-based payment software. They are intended as a minimum set of security characteristics, controls, features, and capabilities that web-based payment software must possess to defend itself from the most common attacks on web software.

While many of the control objectives defined in this standard protect against new and/or novel attacks beyond the most common techniques, attacks will inevitably evolve or emerge that will require new methods or approaches to mitigate them. It is ultimately the responsibility of payment software vendors, providers, developers, and suppliers to keep abreast of evolving attacks techniques and to implement appropriate security controls to enable their software to defend against such attacks.

Security Requirements

Control Objectives	Test Requirements	Guidance
Control Objective C.1: Web Software Components & Services All components and services used by the software are identified and maintained in a manner that minimizes the exposure of vulnerabilities.		
C.1.1 All software components and services are documented or otherwise cataloged in a software bill of materials (SBOM).	**C.1.1** The assessor shall examine evidence to confirm that information is maintained that describes all software components and services comprising the software solution, including: • All proprietary software libraries, packages, modules, and/or code packaged in a manner that enables them to be tracked as a freestanding unit of software. • All third-party and open-source frameworks, libraries, and code embedded in or used by the software during operation. • All third-party software dependencies, APIs, and services called by the software during operation.	Modern software is rarely created entirely in-house and is typically composed of various bespoke code segments integrated with numerous components such as commercial and/or open-source frameworks, libraries, APIs, and services. Any part of this code may have or develop vulnerabilities over time that will require patching or mitigation. Knowing all of the components that comprise a software application or service, where they come from, and how they are updated and maintained is critical to minimizing and managing vulnerabilities in software applications. Without this information, it would be extremely difficult to identify and track vulnerabilities in software components that could expose the embedding software application to attacks. A Software Bill of Materials or “SBOM” services this purpose by documenting information about the software components and versions used to create a software product, their suppliers, and any third-party code that may also be embedded in these components. NIST refers to this information as “provenance data” and there are numerous standards and frameworks available, such as CycloneDX, SPDX and SWID, that describe how this information should be structured. For more information, refer to those standards and frameworks.

Control Objectives	Test Requirements	Guidance
C.1.2 The SBOM describes each of the primary components and services in use, as well as their secondary transitive component relationships and dependencies to the greatest extent feasible.	**C.1.2.a** The assessor shall examine evidence to confirm that the SBOM describes all primary (top-level) components and services in use and all of their secondary transitive relationships and dependencies.	Software components and services may have many nested relationships and dependencies with other software components and services that are owned or maintained by multiple different entities. Identifying all of these different relationships can be challenging where there are many different third-party components nested in the software code. Fortunately, many software development frameworks and compilers provide the capabilities to identify and map nested and transient dependencies. For the purposes of this standard, the SBOM is expected to identify, at a minimum, the code obtained from third parties as well as their secondary transitive component relationships and dependencies (i.e., code embedded in third-party code). If circumstances exist that complicate or prevent the identification of secondary transitive component relationships and dependencies, then such circumstances should be documented, and reasonable justification should be maintained to explain why these dependencies are not accurately reflected in the SBOM. Examples of such circumstances may include third-party APIs, where transparency into nested third-party components called by or embedded into those APIs is not provided by the API provider.
	C.1.2.b The assessor shall test the software to confirm that the information provided in the SBOM accurately reflects the software components and services in use during software operation, including both primary components and services as well as their secondary transitive component relationships and dependencies. Where such dependencies and relationships are not identified and described in the SBOM, the assessor shall confirm that the absence of such information is justified and reasonable.	
C.1.3 Where the software is provided "as a service," the SBOM includes information describing the software dependencies present in the production software execution environment to the greatest extent feasible.	**C.1.3.a** The assessor shall examine evidence to confirm that the SBOM describes all dependencies present in the production software execution environment that the software relies upon for operation or to satisfy security requirements in this standard.	Software that is provided "as a service" often involves the use of components and services resident in the production environment that are unique to that environment. To ensure that these dependencies and relationships are identified and tracked and vulnerabilities in these components and services identified and mitigated, these components and services must be also included in the SBOM. *(continued on next page)*

Control Objectives	Test Requirements	Guidance
	C.1.3.b The assessor shall examine evidence and test the software (to the extent possible) to confirm that the information provided in the SBOM accurately reflects the software dependencies present in the production software execution environment. Where such dependencies are not identified and described in the SBOM, the assessor shall confirm that the absence of such information is justified and reasonable.	Examples of these types of components include, but are not limited to, database servers, web servers, application servers, runtime platform(s), authentication servers/services, "plugins", and any other components or services present in the production environment.
C.1.4 The SBOM includes sufficient information about each component or service to enable tracking each component or service across the software supply chain.	**C.1.4.a** The assessor shall examine evidence to confirm that information is maintained in the SBOM that describes the following for each component and service in use, including secondary component relationships and dependencies: • The original source/supplier of the component or service. • The name of the component or service as defined by the original supplier. • A description of the relationship(s) between the component and service and other components/services embedded in or used by the software. • The version of the component or service as defined by the original supplier to differentiate it from previous or other versions. • The name of the author who designed/developed the component or service. • Any other identifiers provided by the original supplier to uniquely identify the component or service.	The primary purpose of an SBOM is to enable the tracking of software components across the software supply chain and to map them to repositories containing vulnerability information about these components. To facilitate tracking components for these purposes, information must be included in the SBOM that enables software stakeholders to: • Uniquely identify each of the components and services used by the software. • Uniquely identify different versions of the same software components and services that may be used by the software, and to differentiate them from other versions of the same software components and services made available by the supplier(s). • Locate the sources of these components and services such that updated versions may be downloaded, installed, and/or referenced where applicable. Without this basic information, tracking vulnerabilities and available patches in these components and services may be extremely difficult, if not impossible.
	C.1.4.b The assessor shall examine evidence and test the software to confirm that the information provided in the SBOM is an accurate representation of the software components and services present in and/or in use by the software.	

Control Objectives	Test Requirements	Guidance
C.1.5 A new SBOM is created or generated each time the software is updated.	**C.1.5** The assessor shall examine evidence to confirm that a new SBOM is created or otherwise generated for each new release of the software.	To enable tracking of vulnerabilities across different versions of a payment software, it is imperative that each version of the software has a SBOM generated that accurately reflects the components and services in use by that version. Since many different versions of a payment software may be available (or active) at any given time and may include multiple versions of numerous third-party components and services, each version of the payment software must be tracked independently of other versions. Failure to identify and describe the components and services unique to a given version of payment software could enable vulnerabilities to be introduced without the software provider's knowledge if they are unaware that a vulnerable version of a software component or service is in use.
C.1.6 Vulnerabilities in third-party components and services are monitored and managed in accordance with Control Objective 10.	**C.1.6.a** The assessor shall examine evidence to confirm that third-party components and services present in and/or in use by the software are regularly monitored for vulnerabilities in accordance with Control Objective 10.1. **C.1.6.b** The assessor shall examine evidence to confirm that vulnerabilities in third-party components and services are identified and are patched or otherwise mitigated in a timely manner in accordance with Control Objective 10.2.	Vulnerabilities in third-party components and services must be handled in the same manner as vulnerabilities in vendor-controlled code. They must be monitored for vulnerabilities through testing and/or the monitoring of publicly available vulnerability disclosure repositories and managed such that any known vulnerabilities in those components and services are patched, or otherwise mitigated, as quickly as possible. Failure to patch or mitigate a vulnerability in third-party components or services can have the same ramifications as a failure to patch or mitigate a vulnerability in the payment software vendor's own code.

Control Objectives	Test Requirements	Guidance
C.1.7 Where software components and/or resources are hosted or maintained on third-party systems, such as content delivery networks (CDN), the authenticity of those components and resources is verified each time they are fetched.	**C.1.7.a** Where software components and resources are fetched from external and/or third-party repositories, the assessor shall examine evidence to confirm that the authenticity of the software component is verified each time the component is fetched. **C.1.7.b** The assessor shall test the software to confirm that the authenticity of all software components and resources fetched from third-party systems or repositories is verified each time they are fetched by the software.	It is a common architectural design technique in modern web applications to download or "fetch" third-party components and resources (for example, files, scripts, stylesheets, packages, and libraries) that are housed on publicly available code repositories (such as public content delivery networks) at the time they are needed rather than embedding and maintaining those components and resources in local code repositories. This technique provides many benefits including the ability to automatically deploy updates to third-party components and resources without necessarily having to recompile code. Unfortunately, there are some significant drawbacks to this approach. Third-party code repositories are heavily targeted by attackers because it enables them to potentially compromise numerous applications and entities by compromising a single package, library, script, or function. For example, if a malicious individual were able to compromise these repositories or otherwise replace a widely used JavaScript library with a modified version, then that malicious library could be automatically propagated to every user of that library without their knowledge. To mitigate the risk of fetching malicious versions of code from third-party repositories, payment software vendors must validate the authenticity of such components before they are fetched (and/or loaded) by the calling application. There are numerous ways in which this can be accomplished, but the most common method of verifying a component's authenticity is through strong cryptography and digital signatures.

Control Objectives	Test Requirements	Guidance
Control Objective C.2: Web Software Access Controls Software security controls are implemented to restrict access to Internet-accessible interfaces, functions, and resources to explicitly authorized users only.		
C.2.1 User access to sensitive functions and sensitive resources exposed through Internet-accessible interfaces is authenticated.	**C.2.1** Using information obtained in Test Requirements 1.2.a and 2.1.a in the Core Requirements, the assessor shall examine evidence to identify all sensitive functions and sensitive resources exposed through Internet-accessible interfaces.	Writing custom authentication functions is not a trivial matter. There are numerous issues and considerations that must be factored into the design and implementation of such functions including, but not limited to, the fact that they are a significant target for attackers. Authentication functions must be free from weaknesses in design and must be resistant to targeted attacks. Given the importance of and the heavy reliance on such functions for security purposes (and those of this standard), it is recommended that entities use third-party authentication functions, modules, libraries, services, and so on, that are already widely used within the industry and have been subjected to thorough security testing and scrutiny. Where the use of such mechanisms is not feasible, then custom methods may be used. However, custom methods must be designed and implemented in strict accordance with applicable industry standards or best practices for authentication. Failure to do so could expose vulnerabilities or design weaknesses in custom authentication methods to malicious entities who may exploit those vulnerabilities to manipulate or otherwise bypass custom authentication mechanisms.

Control Objectives	Test Requirements	Guidance
C.2.1.1 The methods implemented to authenticate user access to sensitive functions and sensitive resources use industry-standard mechanisms.	**C.2.1.1.a** The assessor shall examine evidence to identify all methods implemented by the software to authenticate access to sensitive functions and sensitive resources.	Similar to developing one's own authorization mechanisms, developing custom authentication mechanisms can be quite a complex undertaking. Much of an application's security is dependent on the strength and robustness of its authentication and authorization mechanisms. There are numerous issues and considerations that must be factored into the design and implementation of such functions including but not limited to, the fact that they are a significant target for attackers. Authentication functions must be free from weaknesses and must be able to withstand targeted attacks. For this reason, only well-designed and well-tested mechanisms should be used. Authentication mechanisms that are provided by industry-accepted suppliers and widely adopted within the industry are generally understood to have been subjected to substantial testing and validation throughout the course of that adoption. Therefore, it is strongly recommended that these mechanisms be used by other entities instead of writing and implementing their own mechanisms. Where the use of third-party mechanisms is not feasible, then custom methods may be used. However, custom methods must be designed and implemented in strict accordance with applicable industry standards or best practices for authentication. Failure to do so could expose vulnerabilities or design weaknesses in custom authentication methods to malicious entities who may exploit those vulnerabilities to manipulate, or otherwise bypass the custom authentication mechanisms, rendering all such functions effectively useless.
	C.2.1.1.b The assessor shall examine evidence to confirm that the implemented methods use industry-standard mechanisms that are: • Provided by well-known and industry-accepted third-party suppliers; or • Designed and implemented in accordance with applicable industry standards or best practices.	
	C.2.1.1.c Where sessions are used to authenticate user access to sensitive functions and sensitive resources, the assessor shall examine evidence to confirm that the sessions are handled in accordance with industry-recognized standards and best practices for secure session management.	
	C.2.1.1.d Where tokens (for example, access tokens and refresh tokens) are used to authenticate user access to sensitive functions and sensitive resources, the assessor shall examine evidence to confirm that the tokens are handled in accordance with industry-recognized standards and best practices for secure token management.	

Control Objectives	Test Requirements	Guidance
C.2.1.2 The methods implemented to authenticate user access to sensitive functions and sensitive resources through Internet-accessible interfaces are sufficiently strong and robust to protect authentication credentials in accordance with Control Objective 5.3.	**C.2.1.2** Using information obtained in Test Requirement C.2.1.1.a, the assessor shall examine evidence to confirm that the authentication methods implemented are sufficiently strong and robust to protect authentication credentials in accordance with Control Objective 5.3 in the Core Requirements section.	Strong and robust authentication methods are those that are resistant to common attacks. Examples of such methods include, but are not limited to, multi-factor authentication and/or authentication methods that employ strong cryptography (such as digital signatures or certificates).
C.2.1.3 Authentication decisions are enforced within a secure area of the software.	**C.2.1.3.a** The assessor shall examine evidence to identify where within the software architecture authentication decisions are enforced.	Like authorization decisions, authentication decisions must be enforced within a secure area of the software. Authentication methods should never rely solely on scripts or data obtained from the client or browser. With that said, it is permissible to use client-side scripts and data when combined with server-side methods to enhance authentication capabilities.
	C.2.1.3.b The assessor shall examine evidence to confirm that all authentication decisions are enforced within a secure area of the software architecture.	
	C.2.1.3.c The assessor shall examine evidence and test the software to confirm that client-side or browser-based functions, scripts, and data are never solely relied upon for authentication purposes.	
C.2.2 Access to all Internet-accessible interfaces is restricted to explicitly authorized users only.	**C.2.2.a** Using information obtained in Test Requirement 2.1.a in the Core Requirements section, the assessor shall examine evidence to identify all software interfaces that are exposed to the Internet or that can be configured in a way that exposes them to the Internet.	Modern web applications, particularly those that rely heavily on APIs, microservices and serverless environments, require fine-grained access control capabilities to handle the increasingly complex relationships between software users, interfaces, functions, and resources. *(continued on next page)*
	C.2.2.b The assessor shall examine evidence to identify all methods used to authorize access to Internet-accessible interfaces.	

Control Objectives	Test Requirements	Guidance
	C.2.2.c The assessor shall examine evidence and test the software to confirm that each of these methods is: • implemented correctly; • appropriate for the types of users expected to use the interface; and • does not expose known vulnerabilities.	One key difference between traditional "monolithic" web applications and modern web applications is the degree to which an application is exposed (or potentially exposed) to the Internet. Where monolithic web applications tend to keep interactions between software components confined to a single security context (such as an internal or isolated system or network), modern web applications are typically segmented into many distinct and/or independent software functions that are then exposed to the Internet through APIs so that they may be accessible to other application or users, regardless of where they may reside. Unfortunately, each Internet accessible interface (and the functions and resources it provides) is a potential attack vector. To mitigate the risks associated with exposing so many software functions to the Internet, each interface must implement access control mechanisms to ensure that only authorized systems and users are able to access the interface, and the functions and resources exposed through those interfaces.
	C.2.2.d Where the methods used to authorize access to Internet-accessible interfaces is user configurable, or otherwise requires user input or interaction, the assessor shall examine evidence to confirm that appropriate guidance is made available to stakeholders in accordance with Control Objective 12.1 that describes the configurable options available and how to configure each method securely.	
	C.2.2.e Where the methods used to authorize access to Internet-accessible interfaces are configured and controlled by the assessed entity, the assessor shall examine evidence to confirm that access to Internet-accessible interfaces is restricted to an appropriate set of explicitly authorized users (or entities).	
	C.2.2.f The assessor shall examine evidence and test the software to confirm that access to all Internet-accessible interfaces is restricted to explicitly authorized users only.	
C.2.3 Access to all software functions and resources exposed through Internet-accessible interfaces is restricted to explicitly authorized users only.	**C.2.3** Using information obtained in Test Requirement C.2.2.a, the assessor shall examine evidence to identify all software functions and resources that are exposed, or that can be configured in a way that exposes them, through Internet-accessible interfaces.	In addition to controlling access at the interface-level, access to individual functions and resources provided through each Internet accessible interface must also be controlled. Access needs to the different functions and resources within a given interface can be quite complex depending upon the types of users and systems that need to use a given interface and the different capabilities and data accessible through those functions and resources.

Control Objectives	Test Requirements	Guidance
C.2.3.1 The software ensures the enforcement of access control rules at both the function level and resource level with fine-grained access control capabilities.	**C.2.3.1.a** Using information obtained in Test Requirement C.2.3, the assessor shall examine evidence to determine how the software controls access to individual functions and resources exposed (or potentially exposed) through Internet-accessible interfaces.	To support the fine-grained access control needs of modern web application architectures and to ensure that users are only able to access the software functions and resources that they are authorized to use, the software must support the ability to define and enforce access control rules at varying "levels" within the interface's hierarchy, including at the individual function and resource level(s). Depending upon the types of functions and resources exposed in a given software interface, the methods used to authorize access at the interface-level may not be appropriate to provide access to individual functions and resources exposed through such interfaces. For example, API keys are often used to authorize access to an API for a particular entity (also called project-level or entity-level authorization). While API keys may be suitable for authorizing this level of access to an API, it may not be suitable for authorizing individual user access to specific functions or resources exposed (or potentially exposed) through the API. *(continued on next page)*
	C.2.3.1.b The assessor shall then examine evidence to identify the methods used to restrict access to the functions and resources exposed (or potentially exposed) through Internet-accessible interfaces and to confirm that each of these methods is: • implemented correctly; • appropriate for the type of function(s) and resource(s) provided; and • does not expose known vulnerabilities.	
	C.2.3.1.c Where the methods used to authorize access to the functions and resources exposed (or potentially exposed) through Internet-accessible interfaces are user configurable or otherwise requires user input or interaction, the assessor shall examine evidence to confirm that guidance is made available to stakeholders in accordance with Control Objective 12.1 that describes the mechanisms and configurable options available to restrict access to the functions and resources exposed through these interfaces, and how to configure such mechanisms.	

Control Objectives	Test Requirements	Guidance
	C.2.3.1.d Where the methods used to authorize access to the functions and resources exposed (or potentially exposed) through Internet-accessible interfaces is configured and controlled by the assessed entity, the assessor shall examine evidence to confirm that access to the functions and resources is restricted to an appropriate set of explicitly authorized users.	Where fined-grained access control is necessary, the methods implemented to control access to all software functions and resources exposed through Internet accessible interfaces must be appropriate for the types of authorization(s) required (for example, user versus entity) and the functions and resources involved (sensitive versus non-sensitive functions and resources). Wherever end users are required to configure the access control authorizations and permissions for individual functions and resources exposed through Internet accessible interfaces, the software vendor must provide guidance (or otherwise make guidance accessible) to users and other stakeholders to explain how to configure such permissions and to alert them to important security considerations when configuring available options and parameters.
	C.2.3.1.e The assessor shall examine evidence and test the software to confirm that the methods used to restrict access to the functions and resources exposed (or potentially exposed) through Internet-accessible interfaces require users to be explicitly authorized prior to being granted such access.	
C.2.3.2 Authorization rules are enforced upon each user request to access software functions and resources through Internet-accessible interfaces.	**C.2.3.2.a** Using information obtained in Test Requirement C.2.3.1.a, the assessor shall examine evidence to confirm that authorization checks are performed each time users request access to a function or resource exposed (or potentially exposed) through Internet-accessible interfaces to verify they are authorized for the function, resource, and type of access requested.	Most modern web applications, particularly those that use APIs, microservices and serverless architectures, operate on a request/response basis. Each time a user wants to perform a function or access application data, they submit a request to the application (usually through an API or similar) to access a particular function or resource. The software then processes that request and, if authorized, executes the requested function and/or returns the requested data. It is often trivial for attackers to obtain login credentials of authorized users. A defense-in-depth strategy is essential to ensuring that only authorized users can access protected functions and resources. When combined with other security controls, such as expiring sessions or tokens after a relatively short period of time, authorization checks can significantly limit what an attacker can do if they are able to compromise the credentials of an authorized user.
	C.2.3.2.b The assessor shall examine evidence and test the software to confirm that access control rules are enforced each time a user attempts to access a function or resource exposed (or potentially exposed) through Internet-accessible interfaces.	

Control Objectives	Test Requirements	Guidance
C.2.3.3 Access control decisions are enforced within a secure area of the software architecture.	**C.2.3.3.a** The assessor shall examine evidence to identify where in the software architecture authorization and access control decisions are enforced.	Payment software should never rely on unknown or insecure services and features for security-related purposes. Secure areas or systems are those within the software architecture where the integrity of available services and data is ensured, and therefore can be relied upon by the software. Historically, web application architectures consisted of "client-side" components and "server-side" components. Client-side functions are those typically performed by a common web browser. Server-side functions are those typically performed by web, application, and/or database servers. Given the open nature and design of most common web browsers and the fact that they are maintained by end users, server-side functions are typically considered to be more secure given a software/service provider's ability to control and secure those aspects of the software architecture. Modern web software architectures, however, have become increasingly complex with software components often deployed across multiple geographic locations and managed by multiple entities. In these circumstances, the distinction between "client-side" or "server-side" functions can be increasingly ambiguous. The term "secure area" is a reference to traditional "server-side" functions without getting into architectural specifics. Examples of a secure area include a secured server environment, a microservice, or a serverless API.
	C.2.3.3.b The assessor shall examine evidence to confirm that all access control decisions are enforced within a secure area of the software architecture.	
	C.2.3.3.c The assessor shall examine evidence and test the software to confirm that client-side or browser-based functions, scripts, and data are never solely relied upon for access control purposes.	

Control Objectives	Test Requirements	Guidance
Control Objective C.3: Web Software Attack Mitigation Software security controls are implemented to mitigate common attacks on web applications.		
C.3.1 The software enforces or otherwise supports the use of the latest HTTP Security Headers to protect Internet accessible interfaces from attacks.	**C.3.1.a** The assessor shall examine evidence to confirm the software supports the use of the latest HTTP Security Headers, and to determine the options available and how such settings are configured.	HTTP Security Headers are a set of security-related configuration options available on most common web servers. Examples include the X-Frame-Options header, the HTTP-Strict-Transport-Security header, and the Content-Security-Policy header. The use of these options can protect against a variety of different types of attacks including cross-site scripting, clickjacking, and cross-site request forgery attacks. While support for specific HTTP Security Headers may differ depending on the underlying platform or software technology, these options are widely available and should be enabled and configured to the most secure configuration feasible for a given implementation.
	C.3.1.b Where HTTP Security Headers are configured and controlled by the software provider, the assessor shall examine evidence to confirm that the software is configured to use the latest available HTTP Security Headers and that the configuration settings are reasonable and justified.	
	C.3.1.c Where user input or interaction is required to configure HTTP Security Headers, the assessor shall examine evidence to confirm that guidance is made available to stakeholders in accordance with Control Objective 12.1 that describes the HTTP Security Headers supported by the software and how to configure such settings.	
C.3.2 Input data from untrusted sources is never trusted and software security controls are implemented to mitigate the exploitation of vulnerabilities through the manipulation of input data.	**C.3.2.a** Using information obtained in Test Requirement C.2.1.a, the assessor shall examine evidence to identify all interfaces that accept data input from untrusted sources.	Many vulnerabilities in software and systems are exposed when input data supplied by an untrusted source is inherently trusted by the software and is processed without first ensuring the data is safe. Untrusted sources are those that reside in a different security context than the API or system receiving and processing the input data. Examples of an untrusted source could include a system, API, or microservice residing in a different environment, an internal system that resides on the same network but is maintained under a lower security classification, or a user's browser. *(continued on next page)*
	C.3.2.b Where the software accepts input from untrusted sources, the assessor shall examine evidence to identify the data format(s) expected by the software for each input field and the parsers and interpreters involved in processing the input data.	

Control Objectives	Test Requirements	Guidance
	C.3.2.c Using information obtained in Test Requirement 4.1.a in the Core Requirements section, the assessor shall examine evidence to determine whether attacks that target all such parsers and interpreters are acknowledged in the threat model.	Two of the most common types of attacks, Injection (SQL, XML, Code, String, and so on) and Cross-Site Scripting (XSS), exploit the software's trust in input data provided by untrusted sources to execute malicious code or to force the software to behave in unintended ways. To protect against these and other types of related attacks, input data must never be trusted, and software security controls must be implemented to ensure input data is validated, rendered safe, or otherwise handled in a manner that mitigates the likelihood and/or impacts of executing malicious input data.
	C.3.2.d Where such attacks are acknowledged and using information obtained in Test Requirement 4.2.a in the Core Requirements section, the assessor shall examine evidence to confirm that software security controls are defined and implemented to mitigate attacks that attempt to exploit vulnerabilities through the manipulation of input data.	
	C.3.2.e Where the implementation of software security controls is configurable or otherwise requires user input or interaction, the assessor shall examine evidence to confirm that guidance is made available to stakeholders in accordance with Control Objective 12.1 that describes how to properly configure such security controls.	
C.3.2.1 Industry-standard methods are used to protect software inputs from attacks that attempt to exploit vulnerabilities through the manipulation of input data.	**C.3.2.1.a** Using information obtained in Test Requirement 4.2.a in the Core Requirements section, the assessor shall examine evidence to identify all software security controls implemented to mitigate attacks that attempt to exploit vulnerabilities through the manipulation of input data.	There are a variety of methods and techniques that may be used to protect software inputs from injection and other similar types of attacks. The method most often associated with such protections is "input validation." Input validation, however, is difficult to implement correctly, particularly where complex input data, such as URLs, XML, JSON, serialized objects, and so on, are involved. Therefore, input validation is not appropriate as a primary defense against input manipulation attacks. *(continued on next page)*

Control Objectives	Test Requirements	Guidance
	C.3.2.1.b The assessor shall examine evidence to confirm that the methods implemented to protect against such attacks use industry-standard mechanisms and/or techniques that are: • Provided by well-known and industry-accepted third-party suppliers; or • Designed and implemented in accordance with applicable industry standards or best practices.	Other methods, such as parameterization and output escaping, are better suited as primary defense mechanisms. While the type and complexity of the input data and how the data is expected to be used often dictate the methods that are most appropriate for a given input, parameterization should be used where possible. Output escaping can be used as an alternative if parameterization is not feasible. The use of input validation may be used as a secondary defense, where appropriate, to provide defense-in-depth. As is the case with other critical functions such as authentication and authorization, input protection methods should leverage industry-accepted third-party mechanisms where possible. If the use of such mechanisms is not feasible, then custom methods may be used if they are designed and implemented in accordance with applicable industry standards or best practices.
	C.3.2.1.c The assessor shall examine evidence and test the software to confirm that the implemented methods: • Are implemented correctly in accordance with available guidance, and • Do not expose any vulnerabilities.	
C.3.2.2 Parsers and interpreters are configured with the most restrictive configuration feasible.	**C.3.2.2.a** Using information obtained in Test Requirement C.3.2.b, the assessor shall examine evidence to identify the configurations for each parser or interpreter used to process untrusted input data.	In some cases, it may not be feasible to isolate (parameterization) or modify (escaping, encoding, etc.) input data prior to processing the data. In such cases, the only viable method to protect against input manipulation attacks is to use a parser or interpreter that has been hardened to prevent such attacks. For example, at the time of this publication the only viable way to protect against an XML External Entity attack is to configure the XML parser to disable the Document Type Definition (DTD) feature, otherwise known as External Entities feature. *(continued on next page)*

Control Objectives	Test Requirements	Guidance
	C.3.2.2.b For each of the parsers/interpreters and the configurations identified, the assessor shall examine evidence to confirm that parsers and interpreters are configured with the most restrictive set of capabilities feasible and that the settings are justified and reasonable. Where certain parser/interpreter features cannot be configured securely, the assessor shall examine evidence to confirm that other methods are implemented to mitigate the lack of secure settings and to further protect against the execution of malicious commands.	The specific settings that must be disabled/enabled to protect against certain attacks depends on the parsers and interpreters. For more information, refer to available security guidance on the specific parsers/interpreters in use. Where certain features of the parsers or interpreters cannot be configured with the most secure settings possible, then the processing of untrusted input data should use techniques such as sandboxing to prevent (or otherwise mitigate the impacts of) malicious code execution.
C.3.3 Software security controls are implemented to protect software interfaces from resource starvation attacks.	**C.3.3.a** Using information obtained in Test Requirements C.2.1.a and C.2.2, the assessor shall examine evidence to identify all Internet accessible interfaces and the functions and resources exposed (or potentially exposed) through those interfaces to identify where such interfaces, functions, and resources may be susceptible to resource starvation attacks.	While the goal of many attacks is to expose sensitive data and sensitive functions (directly or indirectly) to unauthorized users, other attacks are intended to prevent an application's use of or access to important computing resources. Such attacks aim to overwhelm the software/system with requests or fill all available system resources such as processing time or memory, therefore starving the software/system of the resources it requires for normal operation and rendering it unusable to other users. In other cases, these attacks are intended to force the software to behave in unintended ways which could, in turn, enable an attacker to execute arbitrary code on the targeted system or expose sensitive data through error messages. *(continued on next page)*
	C.3.3.b Where such interfaces, functions, and resources are potentially susceptible to resource starvation attacks, the assessor shall examine evidence to confirm that: • The threat of such attacks is documented in accordance with Control Objective 4.1, and • Software security controls to mitigate such attacks are documented and implemented in accordance with Control Objective 4.2.	
	C.3.3.c The assessor shall examine evidence to confirm that the software security controls implemented to mitigate resource starvation and other similar attacks on Internet accessible interfaces are designed and implemented in accordance with applicable industry standards and best practices.	

Control Objectives	Test Requirements	Guidance
	C.3.3.d Where the implementation of software security controls is user configurable or otherwise requires user input or interaction, the assessor shall examine evidence to confirm that guidance is made available to stakeholders in accordance with Control Objective 12.1 that describes how to configure such mechanisms.	Examples of methods used to mitigate the risk of such attacks include limiting the rate on the number of requests that can be submitted within a given time period (rate limiting). Additional methods to prevent such attacks may include defining other limits such as the number of users and/or systems that can submit requests, mutually authenticating those users and systems, or using CAPTCHA or other anti-automation techniques that can prevent large volumes of requests being submitted to software interfaces within a short time period. For SaaS or other similar environments, appropriate network-based controls may also be used to address these types of attacks.
C.3.4 Software security controls are implemented to protect Internet accessible interfaces from malicious file content.	**C.3.4.a** Using information obtained in Test Requirement C.2.1.a, the assessor shall examine evidence to identify all Internet accessible interfaces that accept file uploads and the file types permitted.	File uploads can be used to provide larger datasets to a piece of software. However, such uploads must be managed securely to prevent the misuse of this function. Files not correctly managed may end up being executable on the host system or be used as a vector to infect or subvert the software or other systems (for example, by creating or overwriting malicious configuration files). File upload interfaces may also provide unintended access to the underlying host system or software. Different file types may be provided with different permissions or functions within a host system, and any file upload system should ensure that only expected file types are accepted for upload. However, care must be taken to ensure that this added process does not itself expose vulnerabilities that could be exploited. *(continued on next page)*
	C.3.4.b Where the software accepts file uploads over Internet accessible interfaces, the assessor shall examine evidence to confirm that: • The threat of attacks on file upload mechanisms is documented in accordance with Control Objective 4.1, and • Software security controls to mitigate such attacks are documented and implemented in accordance with Control Objective 4.2.	
	C.3.4.c The assessor shall examine evidence to confirm that the software security controls implemented to mitigate attacks on file upload mechanisms are implemented in accordance with applicable industry standards and best practices.	

Control Objectives	Test Requirements	Guidance
	C.3.4.d The assessor shall examine evidence to confirm that the software security controls implemented to mitigate attacks on file upload mechanisms include methods to restrict the file types permitted by the file upload mechanisms.	Many file formats allow for the embedding of other files or data which can be 'expanded' out when parsing the source file. In some scenarios this can be used to gain privileges or exploit vulnerabilities on the host platform which would not otherwise be possible. Uploaded files need to be managed in ways that prevent the exploitation of file parsing or expansion attacks. To prevent the exploitation of file upload systems, any files that are uploaded cannot be assigned writable or executable privileges. Files which are required to be writable need to be copied across to a separate file managed only by the software to prevent a malicious user from exploiting the file between upload and use. For defense-in-depth, some software development languages and frameworks include the ability to make calls to anti-malware systems to scan these files upon upload. For more information, refer to relevant third-party documentation. File and file type parsers are notorious sources of exploits. Such parsers must not make security decisions based on file names or file extensions. Acceptable file types should have a basic structure that enables the software to determine the file type without using file names or file extensions.
	C.3.4.e The assessor shall examine evidence to confirm that the software security controls implemented to mitigate attacks on file upload mechanisms include methods to restrict the maximum number and size of files permitted for upload.	
	C.3.4.f The assessor shall examine evidence to confirm that the software security controls implemented to mitigate attacks on file upload mechanisms account for the use of complex or compressed file formats that are often used to overwhelm or otherwise exploit file-parsing mechanisms.	
	C.3.4.g The assessor shall examine evidence to confirm that the software security controls implemented to mitigate attacks on file upload mechanisms include methods that store uploaded files outside of the webroot and assign those files read-only permissions.	
	C.3.4.h The assessor shall examine evidence to confirm that the use of file-parsing mechanisms does not rely on file names or file extensions for security purposes.	
	C.3.4.i Where the implementation of software security controls is user configurable or otherwise requires user input or interaction, the assessor shall examine evidence to confirm that guidance is made available to stakeholders in accordance with Control Objective 12.1 that describes how to configure such mechanisms.	

Control Objectives	Test Requirements	Guidance
C.3.5 Software security controls are implemented to protect Internet accessible interfaces from hostile object creation and data tampering.	**C.3.5.a** Using information obtained in Test Requirements C.2.1.a and C.2.2, the assessor shall examine evidence to identify all software functions exposed through Internet accessible interfaces that accept and process data objects as inputs.	Some software APIs accept serialized data objects (for example, arrays, cookies, tokens, and so on) to be passed from other systems. However, without appropriate methods in place to restrict object deserialization and creation, malicious individuals may be able to use these APIs to launch denial-of-service attacks, compromise access control mechanisms, or inject and execute malicious code on underlying systems. There are numerous methods to protect against serialization (and deserialization) attacks. Some programming languages, libraries, and APIs provide features and functions that are resistant to serialization attacks. Other methods include using deserialization mechanisms that only support pure data formats like JSON or XML, limiting data types allowed during object creation, encrypting communications, and authenticating API clients. Appropriate methods to protect against serialization attacks depend on the API implementation. Refer to industry sources, such as the Open Web Application Security Project (OWASP) for more information. For the same reasons explained in the last paragraph of guidance for Control Objective C.3.4, file-parsing mechanisms must not make security decisions based on file names or file extensions. Acceptable file types should have a basic structure that enables the software to determine the file type without using file names or file extensions. *(continued on next page)*
	C.3.5.b Where the software accepts and processes data objects as inputs, the assessor shall examine evidence to confirm that: • The threat of hostile object creation and data tampering attacks is documented in accordance with Control Objective 4.1, and • Software security controls to mitigate such attacks are documented and implemented in accordance with Control Objective 4.2.	
	C.3.5.c The assessor shall examine evidence to confirm that the software security controls implemented to mitigate hostile object creation and data tampering attacks are implemented in accordance with applicable industry standards and best practices.	
	C.3.5.d The assessor shall examine evidence to confirm that the software security controls implemented to mitigate hostile object creation and data tampering attacks include methods that restrict the file formats permitted by file-parsing mechanisms.	
	C.3.5.e The assessor shall examine evidence to confirm that the use of file-parsing mechanisms does not rely on file names or file extensions for security purposes.	
	C.3.5.f The assessor shall examine evidence to confirm that the use of file-parsing mechanisms does not expose other vulnerabilities.	

Control Objectives	Test Requirements	Guidance
	C.3.5.g Where the software accepts serialized objects as inputs, the assessor shall examine evidence to confirm that software security controls are implemented to protect against deserialization attacks and that such security controls adhere to applicable industry standards and best practices.	Some file-parsing mechanisms are inherently susceptible to certain vulnerabilities. For example, XML parsers are often vulnerable to External Entity attacks. Similarly, JSON parsers are vulnerable to attacks where insecure commands, such as eval(), can enable the execution of malicious code. To mitigate attacks that attempt to exploit vulnerabilities in file-parsing mechanisms, it may be necessary for entities to implement additional software security controls. Examples of such controls include, but are not limited to, configuring file-parsing mechanisms to use the most restrictive configuration feasible, avoiding or escaping certain commands that are known issues for file-parsing mechanisms, or isolating and executing file-parsing commands in a sandbox. The methods used to further mitigate such attacks should consider the specific parsers and interpreters in use and be implemented in a manner appropriate for each parser and interpreter.
	C.3.5.h Where the software security controls implemented to protect against hostile object creation and data tampering are user configurable or otherwise require user input or interaction, the assessor shall examine evidence to confirm that guidance is made available to stakeholders in accordance with Control Objective 12.1 that describes how to configure such mechanisms.	
C.3.6 Software security controls are implemented to protect Internet accessible interfaces from attacks that exploit multi-origin resource sharing.	**C.3.6.a** The assessor shall examine evidence to determine if and/or how the software supports cross-origin access to Internet accessible interfaces, and to confirm that access to software APIs and resources from browser-based scripts is disabled by default.	Software may be required to allow access to resources or API interfaces from other domains or Internet origins. This practice may lead to vulnerabilities that expose sensitive data or sensitive functions to attacks. Where not required, cross-origin resource sharing should be disabled. Where cross-origin resource sharing is necessary due to a legitimate business purpose, such access must be enabled only for the domains and origins required for the software to perform its intended function(s). Use of permission lists or other configurations may be suitable for identifying permitted origins, but such configurations must also be protected against modification by malicious parties.
	C.3.6.b Where cross-origin access is enabled, the assessor shall examine evidence to confirm that the reasons for enabling cross-origin access are reasonable and justified, and that access is restricted to the minimum number of origins feasible.	
	C.3.6.c The assessor shall test the software to confirm that the claims made by the assessed entity regarding cross-origin access are valid. At a minimum, testing is expected to include functional testing using forensic tools/techniques.	

Control Objectives	Test Requirements	Guidance
	C.3.6.d Where disabling or restricting cross-origin access to software APIs requires user input or interaction, the assessor shall examine evidence to confirm that appropriate guidance on this process is provided to stakeholders in accordance with Control Objective 12.1.	

Control Objectives	Test Requirements	Guidance
Control Objective C.4: Web Software Communications Sensitive data transmissions are secured in accordance with Control Objective 6.		
C.4.1 Sensitive data transmissions are encrypted in accordance with Control Objectives 6.2 and 6.3.	**C.4.1.a** Using information obtained in Test Requirement 6.2.a, the assessor shall examine evidence to determine how communications are handled by the software, including those between separate systems in the overall software architecture.	The types of data which may be considered sensitive may vary across different implementations. See Control Objective 1.1 for more information on identifying sensitive data. It is therefore important that any connection that transmits sensitive data is encrypted using strong cryptography. Common methods for achieving this will include the use of TLS using appropriate cipher suites. Although connections that do not transmit sensitive data do not explicitly require the use of encryption, it is noted that the use of strong cryptography to secure all connections is considered a best practice and should be implemented for all communications unless legitimate business or technological constraints exist that render such an approach infeasible. In most cases, however, communications between web application components include the transmission of authentication information (user credentials or session information) which is considered sensitive data by definition and should therefore be encrypted using strong cryptography.
	C.4.1.b Where the software allows or otherwise supports the transmission of sensitive data between users and systems in different security contexts, the assessor shall examine evidence to confirm that all such communications are encrypted using strong cryptography in accordance with Control Objectives 6.2 and 6.3.	

Control Objectives	Test Requirements	Guidance
	C.4.1.c Where sensitive data is transmitted using server-to-server communications (for example, using APIs), the assessor shall examine evidence to confirm that the software enforces or otherwise supports mutual authentication between systems.	Where sensitive data is transmitted between systems operating within different security contexts and/or different environments, it is important that such communications be restricted to an explicitly approved list of systems, and that the systems involved be mutually authenticated such that attempts to intercept or compromise such communications are appropriately mitigated. Where the software provider controls the configuration of such communications, mutual authentication must be enforced. Otherwise, the software provider must provide features to support the mutual authentication of disparate systems so that an implementing entity may configure such features accordingly.
	C.4.1.d Where internally generated or self-signed certificates are used for securing sensitive data transmissions, the assessor shall examine evidence to confirm that: • The use of internally generated or self-signed certificates is reasonable and justified. • The software is configured to accept the minimum feasible number of internally generated or self-signed certificates.	Many organizations that choose to use internally generated and/or self-signed certificates do so for the benefits they offer without considering the additional overhead needed to manage them securely. As a result, critical security processes such as certificate revocation and key management are not implemented or maintained properly. For this reason, the use of internally generated and/or self-signed certificates should be kept to an absolute minimum. Where their use is required, such instances should be documented and justified.