Seezo

Security Assessment Report

2025-08-14

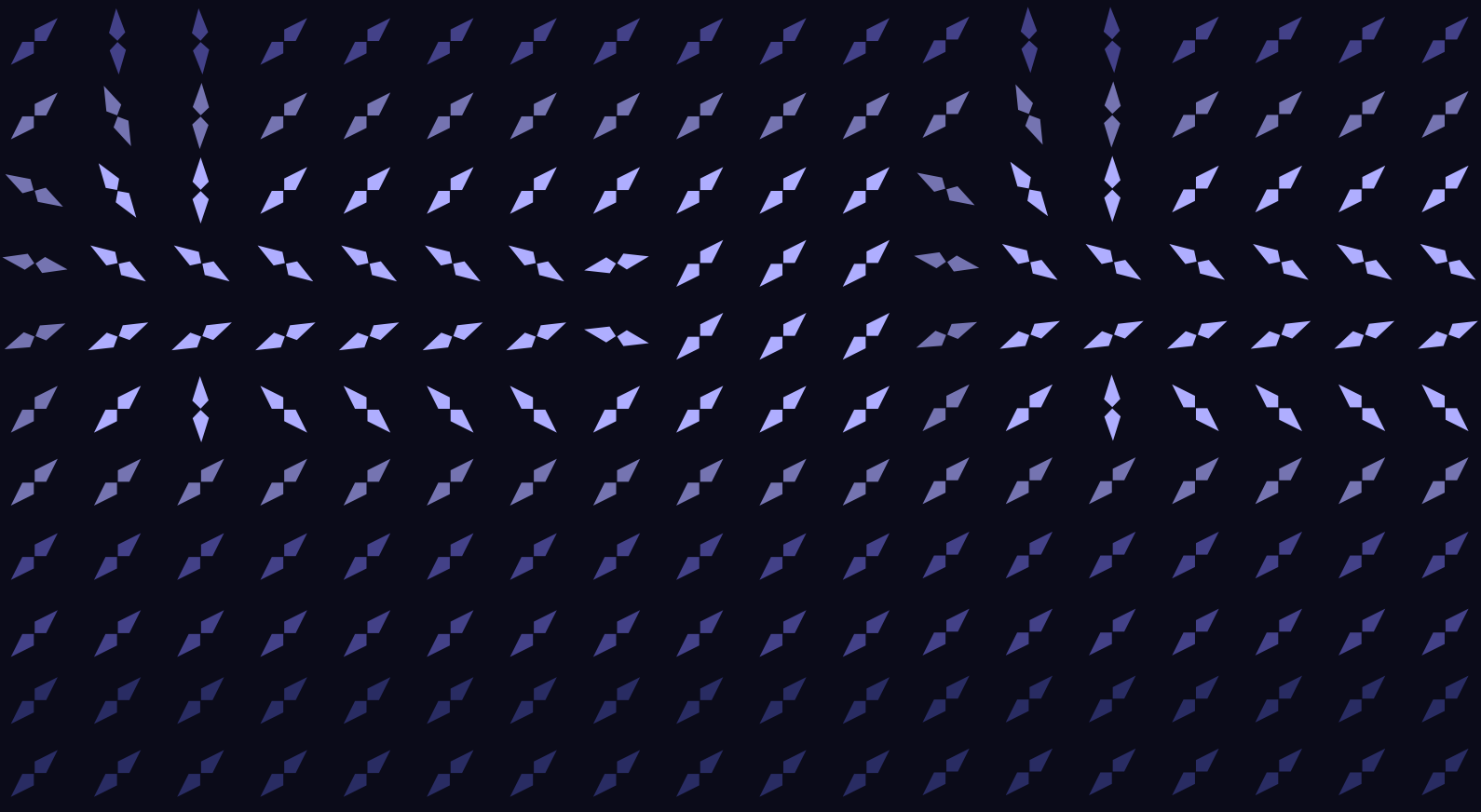

Table of Contents

Overview

Category wise Security Requirements and Open Questions

Security Controls

Identified Assets

Threats

Compliance

Appendix: Security Mapping

DVPATS

Overview of results of the DVPATS Assessment

Executive Summary

The document details the technical architecture of strideelectronics.com, an e-commerce platform using AWS, Django, and MongoDB with a microservices architecture. The technology stack includes React.js for the frontend, Django REST Framework for the backend, and MongoDB managed via Atlas or self-hosted on EC2. The infrastructure is updated to use AWS EKS for Kubernetes management, with components like Amazon RDS for MongoDB, S3 for storage, and ElastiCache for caching. Security and communication are handled through HTTPS and JWT-based authentication.

Assessment Risk: High

The factors contributing to the risk are:

- **Weak Authentication Mechanisms**: Use of SHA1 hashing for passwords and JWT tokens for authentication, with modifications and new implementations in authentication processes.
- **Sensitive Data Handling**: Management of PII (full name, email, phone number, addresses) and financial information (payment methods, StrideCredit balance) with secure storage practices like AWS Secrets Manager and TLS encryption.
- **PCI-DSS Compliance**: Direct handling of credit/debit card information with tokenization via a payment gateway, indicating processing of cardholder data.
- **Microservices Architecture**: Introduction of new APIs for services like Auth, Payment, Order, Inventory, and Notification, managing functionalities such as login, signup, and payment methods.
- **Public-Facing Features**: Internet-facing e-commerce platform with a React.js frontend, managed by AWS Load Balancer, supporting user registration, login, and product browsing.
- **Authenticated CRUD Operations**: Authenticated users can perform CRUD operations on products, inventory, orders, profile settings, addresses, and payment methods.

High Assessment Risk	**22** Open Questions
35 Security Requirements	**51** Identified Assets

Resources

Techincal Specification - Stride Electronics.docx

Product Specification - Stride Electronics.docx

Category wise Security Requirements and Open Questions

1. Authentication

Security Requirements	Open Questions
10	5

Security Requirements

	State	Ranking
1.1 Use cryptographically secure JWT signing secret	Open	High

Description

Context: The application uses JWTs for authentication, with tokens signed using a secret key stored in AWS Secrets Manager. Access tokens expire in 2 hours, while refresh tokens are valid for 30 days.

Gap: The JWT signing secret is described as simple, which compromises the cryptographic strength needed to prevent unauthorized token generation.

Threat: An attacker can predict or brute-force the simple secret key, allowing them to generate unauthorized tokens and gain access to protected resources.

Remediation

Use a cryptographically secure random key for JWT signing. The key should be at least 256 bits long and generated using a secure random number generator. Store the key securely in AWS Secrets Manager and rotate it periodically to enhance security.

	State	Ranking
1.2 Implement IAM roles for AWS services to avoid hardcoded credentials	Open	Medium

Description

Context: The application utilizes various AWS services, including Amazon EKS, Amazon RDS, Amazon S3, and AWS Secrets Manager, integrated with AWS Identity and Access Management (IAM) for managing access and permissions.

Gap: While IAM is mentioned for access management, there is no explicit reference to the use of IAM roles for EC2 instances or other AWS services, leading to potential hardcoding of credentials.

Threat: An attacker could exploit hardcoded credentials to gain unauthorized access to AWS resources, potentially compromising sensitive data and application integrity.

Remediation

Implement IAM roles for EC2 instances and other AWS services to manage permissions securely. Ensure that roles are configured with the least privilege principle and that credentials are not hardcoded in the application code.

Standard Mapping

ASVS: V2.10.1 V1.6.4 V6.4.1 V2.10.4

1.3 Use secure hashing algorithms for password storage

Open High

Description

Context: The application uses password-based authentication, where users log in using an email and password combination.

Gap: Passwords are stored using SHA1 hashing, which is cryptographically weak and not suitable for secure password storage.

Threat: An attacker can exploit the weak hashing algorithm to crack stored passwords, potentially gaining unauthorized access to user accounts.

Remediation

Switch to a secure hashing algorithm such as bcrypt or Argon2 for password storage to ensure cryptographic security and resistance to brute force attacks.

Standard Mapping

ASVS: V2.10.3 V2.4.1 V2.10.4 V2.4.2 V6.4.1 V2.9.1

1.4 Invalidate JWTs upon user logout and password changes

Open High

Description

Context: The application uses JWTs for authentication, with tokens generated at login/signup and signed using a secret key stored in AWS Secrets Manager. The logout feature is intended to invalidate the current JWT, and tokens are removed from the browser upon logout.

Gap: While tokens are removed from the browser during logout, they remain valid on the server, indicating a lack of server-side invalidation. Additionally, there is no mention of token invalidation upon password changes.

Threat: An attacker could use old JWTs to gain unauthorized access even after a user logs out or changes their password, potentially compromising user accounts and sensitive data.

Remediation

Implement a token revocation mechanism that ensures JWTs are invalidated immediately upon user logout or password change. This can be achieved by maintaining a token blacklist or using short-lived tokens with refresh tokens that are invalidated upon logout or password change. Additionally, consider implementing token versioning, where a token is invalidated if the user's password is changed, and a new token version is issued.

1.5 Implement robust password policy for user authentication

Open High

Description

Context: The application uses password-based authentication, requiring users to log in with an email and password combination.

Gap: The password policy allows passwords between 4—12 characters without restrictions on special characters, lacks robust length checks, and does not maintain password history, enabling reuse of old passwords.

Threat: An attacker can exploit weak passwords to gain unauthorized access, potentially leading to account compromise and data breaches.

Remediation

Implement a strong password policy requiring a minimum length of 12 characters, including a mix of uppercase, lowercase, numbers, and special characters. Enforce password history to prevent reuse of old passwords.

Standard Mapping

ASVS: V2.1.7 V2.1.1 V2.1.10 V2.1.12 V2.1.5 V2.1.3 V2.1.2 V2.1.11 V2.1.8 V2.1.9 V2.1.6 V2.1.4

1.6 Implement MFA for user account authentication

Open High

Description

Context: The application uses password-based authentication, requiring users to log in with an email and password combination.

Gap: There is no mention of implementing multi-factor authentication (MFA) or any additional security measures to enhance the security of user accounts.

Threat: An attacker could compromise user accounts through credential stuffing or brute force attacks, gaining unauthorized access to sensitive information.

Remediation

Implement MFA or 2FA for user authentication, using methods such as SMS OTP, email OTP, or authenticator apps to enhance account security.

Standard Mapping

ASVS: V2.3.2 V2.2.1 V2.8.2 V2.8.7 V2.2.4 V2.2.7 V2.7.1 V2.2.2 V2.8.5 V2.5.5 V2.5.7 V2.2.6 V2.8.3 V2.2.3 V2.5.6

1.7 Implement server-side revocation for JWTs in authentication system

Open High

Description

Context: The application uses JWTs for authentication, with tokens generated at login/signup and signed using a secret key stored in AWS Secrets Manager. Access tokens expire in 2 hours, while refresh tokens are valid for 30 days.

Gap: There is no server-side mechanism to revoke or invalidate JWTs immediately, such as blacklisting or maintaining a token revocation list, which means tokens remain valid on the server even after logout.

Threat: An attacker could exploit the lack of token revocation to continue accessing resources with a compromised JWT until it naturally expires, potentially leading to unauthorized access and data breaches.

Remediation

Implement a server-side mechanism to revoke JWTs, such as maintaining a blacklist of invalidated tokens. This can be achieved by storing invalidated tokens in a database or cache and checking against this list during token validation. Additionally, consider reducing the token expiration time to minimize the window of opportunity for misuse.

1.8 Implement generic error messages for login attempts

Open High

Description

Context: The application uses password-based authentication, where users log in using an email and password combination. Error messages specify whether the email or password is incorrect.

Gap: There is no mention of security measures such as rate limiting or CAPTCHA to prevent enumeration attacks.

Threat: An attacker can enumerate valid user login IDs by repeatedly attempting logins and analyzing the error messages, potentially leading to unauthorized access or further targeted attacks.

Remediation

Implement generic error messages that do not specify whether the email or password is incorrect. Introduce rate limiting and CAPTCHA to prevent automated enumeration attempts.

Standard Mapping

ASVS: V2.1.7 V2.2.1 V2.2.3

1.9 Implement regular rotation for JWT secrets, API tokens, and MongoDB credentials

Open High

Description

Context: The application uses AWS Secrets Manager to store sensitive credentials, including JWT secrets, API tokens, and MongoDB access credentials, which are crucial for authentication and secure communication between microservices.

Gap: There is no mention of a policy or mechanism for the regular rotation of secrets and keys, nor any integration with secret management tools for automated or manual rotation schedules.

Threat: An attacker could exploit static secrets and keys to gain unauthorized access to sensitive data and services, potentially leading to data breaches, unauthorized transactions, or service disruptions.

Remediation

Implement a regular rotation policy for secrets and keys using AWS Secrets Manager's automatic rotation feature. Ensure that all secrets and keys are rotated at least every 90 days, and integrate this process into the CI/CD pipeline to automate and enforce the rotation schedule.

Standard Mapping

ASVS: V2.10.1 V2.10.4 V6.4.1 V1.6.2 V1.6.3 V1.6.1

1.10 Validate JWT signatures for authentication tokens

Open High

Description

Context: The application uses JWTs for authentication, with tokens generated at login/signup and signed using a secret key stored in AWS Secrets Manager. Access tokens expire in 2 hours, and refresh tokens are valid for 30 days.

Gap: While the middleware checks the Authorization header for token validity, there is no explicit mention of JWT signature validation or the use of cryptographic methods for verifying the JWT signature. Signature verification is also skipped during the refresh token process.

Threat: An attacker could forge or manipulate JWTs to gain unauthorized access, bypassing authentication mechanisms and potentially accessing sensitive user data or system functionalities.

Remediation

Implement JWT signature validation using a secure cryptographic method such as HMAC, RSA, or ECDSA. Ensure that the server verifies the signature of every JWT before processing its payload. In Django, this can be achieved using libraries like PyJWT or djangorestframework-jwt, which provide built-in methods for signature verification.

Open Questions

Q1.1 Amazon EKS, RDS, S3, and IAM are used for managing access to resources, with user registration via email and social logins like Google and Facebook. 1. How is IAM federation configured to integrate external identity providers securely?

Q1.2 Password-based authentication is used in the application, allowing users to log in with their email and password, and providing error messages for invalid credentials. 1. Are passwords masked during input to prevent exposure to shoulder surfing and unauthorized access?

Q1.3 The application uses a password reset feature that sends reset links via email, relying on password-based authentication for user accounts. 1. Are SPF, DKIM, and DMARC protocols implemented to secure email communications and prevent spoofing?

Q1.4 Mobile apps handle sensitive data, including user information and payment methods. 1. Is the copy-paste functionality disabled for sensitive input fields like passwords or credit card numbers to prevent data leakage or unauthorized access?

Q1.5 JWTs are used for authentication, with tokens generated at login/signup and signed using a secret key stored in AWS Secrets Manager. 1. What signing algorithm is used for JWTs, and is the 'none' algorithm disabled to prevent token forgery?

---

2. Communication

Security Requirements	Open Questions
6	3

Security Requirements

	State	Ranking
2.1 Track and manage x.509 certificate validity for MongoDB authentication	Open	Medium

Description

Context: MongoDB access and security involve authentication via x.509 certificates, indicating the use of TLS certificates for authentication purposes.

Gap: While x.509 certificates are used for authentication, the document lacks information on the validity period of these digital certificates, including details such as start and end dates or certificate lifecycle management.

Threat: An attacker could exploit expired or invalid certificates to bypass authentication mechanisms, potentially gaining unauthorized access to the database.

Remediation

Implement a certificate management policy that includes tracking the validity period of digital certificates. Set up automated alerts to notify the development and operations teams at least 30 days before a certificate is due to expire. Ensure that the renewal process is documented and tested to prevent service disruptions.

Standard Mapping

ASVS: V9.2.4 V1.6.1

2.2 Use POST requests for transmitting payment details

Open High

Description

Context: The application handles sensitive data, including payment details like card number, expiry date, and CVV, which are transmitted via the 'POST /account/payment-methods' endpoint.

Gap: The use of query parameters to submit sensitive payment details suggests that data is being exposed in the URL, which is typically associated with GET requests, leading to potential exposure of sensitive information.

Threat: An attacker can intercept URLs containing sensitive data, leading to unauthorized access to payment information, potential financial fraud, and non-compliance with data protection standards.

Remediation

To mitigate this risk, ensure that sensitive data such as card numbers, expiry dates, and CVV codes are transmitted using POST requests with the data included in the request body rather than as URL parameters. Additionally, implement encryption for data in transit using HTTPS to protect the data from being intercepted. Ensure that sensitive data is not logged in plaintext on the server side. Consider using tokenization or encryption for sensitive data before transmission.

Standard Mapping

ASVS: V8.2.1 V8.3.1 V8.1.1 V8.3.4

PCI-DSS: 4.2.1 6.2.4 4.2.2

2.3 Implement certificate revocation for MongoDB x.509 authentication

Open High

Description

Context: MongoDB access and security involve authentication via x.509 certificates, indicating the use of TLS certificates for authentication purposes.

Gap: While x.509 certificate-based authentication is implemented, there is no mention of a certificate revocation process, including CRLs or OCSP, to manage the lifecycle of these certificates.

Threat: An attacker could exploit the absence of a certificate revocation process to use compromised or expired certificates for unauthorized access, potentially leading to data breaches or unauthorized actions within the system.

Remediation

Implement a certificate revocation process using Certificate Revocation Lists (CRLs) or Online Certificate Status Protocol (OCSP) to ensure that compromised or invalid certificates are promptly revoked. Regularly update and monitor the revocation lists to maintain the integrity and security of the certificate-based authentication system.

Standard Mapping

ASVS: V9.2.4

2.4 Implement error handling for unrecognized MongoDB x.509 certificates

Open Medium

Description

Context: MongoDB access and security involve authentication via x.509 certificates, indicating the use of TLS certificates for authentication purposes.

Gap: While x.509 certificate-based authentication is implemented, there is no information on handling unrecognized or invalid certificates, nor any error handling procedures for certificate authentication failures.

Threat: An attacker could exploit the lack of error handling for unrecognized certificates to bypass authentication, potentially gaining unauthorized access to the database.

Remediation

Implement error handling mechanisms to reject unrecognized or invalid certificates. Log such events for auditing and alert administrators to investigate potential security incidents. Ensure that the server provides informative error messages without revealing sensitive information.

Standard Mapping

ASVS: V9.2.1 V1.9.2 V9.2.3

2.5 Use trusted Certificate Authorities for MongoDB x.509 certificates

Open High

Description

Context: The application uses x.509 certificates for MongoDB authentication, indicating the use of TLS certificates for secure access.

Gap: The document lacks explicit references to a trusted issuer or Certificate Authority (CA) for the digital certificates, leaving the trustworthiness of the certificates unclear.

Threat: An attacker could exploit untrusted or self-signed certificates to perform man-in-the-middle attacks, intercepting or altering data between the application and the database.

Remediation

Ensure that all digital certificates used for authentication are issued by a trusted Certificate Authority (CA). Maintain an updated list of trusted CAs in the application's trust store to verify the authenticity of certificates. Regularly review and update the list to include any new trusted CAs and remove any that are no longer trusted.

Standard Mapping

ASVS: V9.2.3 V9.2.1 V9.2.4

2.6 Validate x.509 certificates for MongoDB authentication

Open High

Description

Context: MongoDB access and security involve authentication via x.509 certificates, indicating the use of TLS certificates for authentication purposes.

Gap: While x.509 certificates are used for authentication, the document lacks explicit references to the validation of certificate attributes such as domain, common name, or organization.

Threat: An attacker could exploit the absence of certificate validation to perform man-in-the-middle attacks, intercepting or altering data between the client and server.

Remediation

Implement strict certificate validation by checking key attributes such as the common name (CN), organization, and issuer. Ensure that the certificate is signed by a trusted Certificate Authority (CA) and is not expired or revoked. Use libraries or frameworks that support robust certificate validation.

Standard Mapping

ASVS: V9.2.1 V9.2.3 V1.9.2

Open Questions

Q2.1 Amazon EC2 instances are used to host services in the application. 1. Are the EC2 security groups configured to deny all inbound traffic by default?

Q2.2 EC2 instances are used in the application with IP whitelisting enabled for EC2/VPC endpoints, indicating some instances are non-internet facing. 1. Are the EC2 instances placed in private subnets to prevent direct internet access?

Q2.3 iOS applications are being launched, but there is no mention of the NSAllowsArbitraryLoads setting or Application Transport Security (ATS) configuration in the Info.plist file, and no minimum OS version requirement is specified. 1. How is the application ensuring secure data transmission without the ATS configuration in place? 2. What is the minimum OS version requirement for the iOS apps to ensure they include essential security features and updates?

---

3. Validation, Sanitization and Encoding

Security Requirements	Open Questions
2	0

Security Requirements

	State	Ranking
3.1 Sanitize log entries to prevent injection and forging risks	Open	High

Description

Context: The application uses logging functionality through tools like 'CloudWatch Logs', 'ELK Stack', and 'Prometheus + Grafana' to capture and store logs for various events or actions.

Gap: There is no mention of data sanitization methods or practices being implemented before logging, such as escaping special characters or input validation.

Threat: An attacker could exploit the lack of data sanitization to perform log injection or forging, potentially leading to unauthorized access, data manipulation, or obfuscation of malicious activities.

Remediation

Implement data sanitization practices for log entries to prevent log injection and log forging attacks. This can include escaping special characters, validating input data, and using logging libraries with built-in sanitization features. Ensure that no user input is directly logged without proper sanitization. Additionally, refer to security standards and best practices for log data handling to ensure secure implementations.

Standard Mapping

ASVS: V7.3.1

3.2 Implement context-specific output encoding for user input in API endpoints

Open High

Description

Context: The application accepts user input through various sources, including user registration, login, password reset requests, and product listings, via API endpoints such as POST /auth/signup and POST /auth/login.

Gap: There is no mention of context-specific output encoding or sanitization for these untrusted user inputs, allowing them to flow directly into downstream processes.

Threat: An attacker can exploit this lack of sanitization to perform injection attacks, potentially compromising the application's integrity and accessing sensitive data.

Remediation

Implement context-specific output encoding for all untrusted user inputs. Use libraries or frameworks that provide built-in protection against injection attacks, such as OWASP's Java Encoder or similar tools for other languages.

Standard Mapping

ASVS: V5.3.4 V5.5.3 V5.1.3 V5.2.4 V5.3.8 V5.3.3 V5.3.1 V5.2.5 V5.2.8 V5.3.5 V5.2.6 V5.3.9 V5.3.10 V5.2.7

4. Data Protection

1 1

Security Requirements

	State	Ranking
4.1 Implement background splash screen to protect user financial data	Open	High

Description

Context: The mobile application handles sensitive data, including user information and financial details, as part of user registration, profile settings, and checkout processes. It supports secure payment methods and complies with KYC verification and PCI-DSS standards.

Gap: The document does not mention the implementation of a background splash screen or similar mechanism to protect sensitive information when the app is backgrounded.

Threat: An attacker could exploit the absence of a background splash screen to capture or view sensitive information when the app is minimized, leading to unauthorized data access and potential data breaches.

Remediation

Implement a background splash screen that activates when the app is minimized or switched to the background. This screen should obscure any sensitive information displayed on the app. For Android, use the `onPause()` method to trigger the splash screen, and for iOS, use the `applicationDidEnterBackground()` method.

Standard Mapping

ASVS: V8.1.1 V8.2.1 V8.3.1 V8.2.2

PCI-DSS: 6.2.1 6.2.3

Open Questions

Q4.1 Amazon RDS is used for managing MongoDB instances in the application. 1. Is there a deletion protection mechanism in place for the MongoDB instances managed by Amazon RDS?

5. Malicious Code

Security Requirements	Open Questions
6	1

Security Requirements

	State	Ranking
5.1 Implement root detection for Android application	Open	High

Description

Context: The application is designed to support mobile platforms, including Android and iOS, as part of its rollout plan.

Gap: The document lacks references to root detection mechanisms for the Android application, such as SafetyNet, RootBeer, or custom root detection logic.

Threat: An attacker can exploit the absence of root detection to run the application on rooted devices, potentially bypassing security controls and gaining unauthorized access to sensitive data or application functionalities.

Remediation

Implement root detection mechanisms using tools like SafetyNet or RootBeer to prevent the application from running on rooted devices. Ensure that the implementation is robust and regularly updated to handle new rooting methods. Additionally, conduct thorough testing to verify the effectiveness of the root detection logic.

	State	Ranking
5.2 Implement code signing for mobile app binaries	Open	High

Description

Context: The application includes mobile versions for Android and iOS, as indicated by the rollout plan for 'Phase 4: Mobile App Launch (Month 6)'.

Gap: There is no mention of code signing for the mobile app binaries, leaving the integrity and authenticity of the apps unverified.

Threat: An attacker could distribute tampered or malicious versions of the app, leading to unauthorized access, data theft, or reputational damage.

Remediation

Implement code signing for the mobile application using a secure cryptographic method such as X.509 certificates. Ensure that the signing process is integrated into the CI/CD pipeline to automatically sign the application binaries before deployment. This will help verify the integrity and authenticity of the application, preventing unauthorized modifications.

Standard Mapping

ASVS: V10.3.2

5.3 Implement jailbreak detection in the iOS application

Open High

Description

Context: The iOS application is part of the mobile app launch in 'Phase 4' and is intended for use on iOS devices.

Gap: The document lacks references to jailbreak detection or related security measures for the iOS application.

Threat: An attacker can exploit the absence of jailbreak detection to run the application on compromised devices, potentially bypassing security controls and accessing sensitive data or functionalities.

Remediation

Implement jailbreak detection mechanisms in the iOS application. Use libraries like JailMonkey or custom checks to detect signs of a jailbroken device, such as the presence of Cydia or writable system directories. Ensure that the application does not run on devices that fail these checks.

Standard Mapping

ASVS: V10.2.1 V10.2.2 V10.2.3

5.4 Restrict custom keyboards for entering user credentials and payment information

Open Medium

Description

Context: The mobile application handles sensitive data, including user credentials and payment information, during interactions such as user sign-up, login, and checkout processes.

Gap: While security measures like SSL encryption, PCI-DSS compliance, and GDPR are mentioned, there is no specific restriction on the use of custom keyboards during sensitive data entry.

Threat: An attacker could exploit the lack of keyboard restrictions to capture sensitive information, such as passwords and credit card details, leading to data leakage and potential identity theft.

Remediation

Implement a restriction on the use of custom keyboards when entering sensitive information in the application. This can be achieved by configuring the input fields to use the default system keyboard only. For example, in iOS, use the 'secureTextEntry' property for UITextField to ensure secure input.

Standard Mapping

PCI-DSS: 8.3.1 8.3.2 8.3.11

ASVS: V10.2.2 V2.1.11 V2.1.12

5.5 Implement binary obfuscation for Android and iOS application code

Open High

Description

Context: The application is set to launch mobile versions for Android and iOS platforms, as outlined in the rollout plan.

Gap: There is no mention of binary obfuscation techniques or tools to protect the application code from reverse engineering.

Threat: An attacker can reverse engineer the mobile application to extract sensitive information, understand the application's logic, or inject malicious code, potentially compromising user data and application integrity.

Remediation

Implement robust binary obfuscation techniques using tools such as ProGuard, DexGuard, or R8 to protect the application code from reverse engineering. Ensure that the obfuscation process is part of the build pipeline and regularly updated to counteract new reverse engineering techniques. Additionally, consider implementing runtime application self-protection (RASP) to detect and respond to tampering attempts in real-time.

Standard Mapping

ASVS: V10.2.3

5.6 Implement certificate pinning in mobile applications

Open

High

Description

Context: The mobile application is designed to support Android and iOS platforms, with SSL encryption and HTTPS communication mentioned for secure data transmission.

Gap: While SSL encryption and HTTPS communication are implemented, there is no mention of certificate pinning, which is crucial for preventing unauthorized interception of data.

Threat: An attacker could perform a man-in-the-middle (MITM) attack to intercept and manipulate data transmitted between the mobile application and the server, compromising user privacy and data integrity.

Remediation

Implement certificate pinning in the mobile application to ensure that only trusted certificates are accepted during SSL/TLS handshake. For Android, use the `OkHttp` library to pin certificates, and for iOS, use `NSURLSession` with pinned certificates. Ensure that the pinned certificates are updated regularly and securely.

Standard Mapping

ASVS: V1.9.2

PCI-DSS: 4.2.1.1 4.2.2 4.2.1

Open Questions

Q5.1 The Android and iOS apps are being launched, but the permissions these mobile applications request, such as access to location, contacts, camera, microphone, or storage, are not specified. 1. What permissions do the mobile applications request, and how are they managed to ensure user privacy and security?

---

6. Business Logic

Security Requirements	Open Questions
2	1

Security Requirements

State

Ranking

6.1 Implement rate limiting for POST /auth/login API endpoint

Open High

Description

Context: The application uses REST APIs, including the POST /auth/login endpoint, with service-level API tokens stored in Secrets Manager for microservices authentication.

Gap: There is no rate limiting or CAPTCHA implemented on the POST /auth/login endpoint, and request bursts are absorbed without per-caller pacing rules.

Threat: An attacker can perform brute force or denial-of-service attacks on the login endpoint, potentially leading to unauthorized access or service disruption.

Remediation

Implement rate limiting on API endpoints to restrict the number of requests a user can make in a given time frame. For example, limit login attempts to 5 per minute per IP address. Additionally, consider using CAPTCHA to prevent automated abuse.

Standard Mapping

ASVS: V2.2.1 V11.1.8 V11.1.4 V8.1.4 V11.1.7

6.2 Implement controls to prevent race conditions in financial transactions

Open High

Description

Context: The application facilitates financial transactions through features like payment method selection and secure payment processing, including credit cards, gift cards, and StrideCredit. It integrates with payment gateways and manages transaction ledgers and commission deductions.

Gap: There is no mention of controls to prevent race conditions or timing attacks, such as synchronization mechanisms, concurrency control, or transaction isolation levels.

Threat: An attacker could exploit the absence of these controls to manipulate transaction timing, potentially leading to unauthorized financial gains or data corruption.

Remediation

Implement transaction isolation levels and locking mechanisms to ensure atomicity and integrity of financial transactions. Use timestamps or versioning to manage concurrency and prevent race conditions. Additionally, conduct thorough testing to identify and mitigate

potential timing attack vectors.

Standard Mapping

PCI-DSS: 7.2.5.1 10.2.1.1 10.2.1.2 10.7.3 10.2.1.4 6.2.4 10.2.1.5 10.6.3 10.2.1.6

ASVS: V11.1.6 V1.11.2 V1.11.3

Open Questions

Q6.1 The application handles sensitive user data, including PII and financial information, and supports various payment methods and transactions. 1. What fraud detection mechanisms, such as anomaly detection or transaction monitoring, are in place to identify and prevent fraudulent activities?

---

7. Configuration

Security Requirements	Open Questions
1	5

Security Requirements

	State	Ranking
7.1 Configure EKS pods to run as non-root users	Open	High

Description

Context: The application uses an Amazon EKS Cluster to manage the Kubernetes control plane, which hosts various pods for application deployment.

Gap: The document lacks specific references to pod security configurations, such as 'runAsUser' or 'securityContext', which are necessary to ensure pods do not run as root users.

Threat: An attacker could exploit pods running as root to gain elevated privileges, potentially compromising the entire Kubernetes cluster and accessing sensitive data or services.

Remediation

Ensure that all pods in the EKS cluster are configured to run as non-root users by setting the 'runAsUser' field in the pod's security context to a non-zero value. This can be done by updating the Kubernetes deployment YAML files to include a security context with a specific user ID, such as 'runAsUser: 1000'.

Standard Mapping

ASVS: V1.2.1 V1.14.5

Open Questions

Q7.1 Amazon EKS, RDS, S3, and ElastiCache are used within a VPC for application deployment and data management. 1. What specific security group rules are in place to control access to these AWS services within the VPC?

Q7.2 Amazon ECS is used for container orchestration in the application. 1. What are the ECS task definitions and resource limits set for CPU and memory to prevent resource exhaustion? 2. Are ECS containers configured to run as non-root users to prevent unauthorized access to the host system?

Q7.3 Android apps are being launched without specifying the minimum SDK version in the AndroidManifest.xml or build.gradle files. 1. What is the minimum SDK version set for the Android app to ensure compatibility with modern security features?

Q7.4 The application uses third-party open-source libraries like React.js, Django REST Framework, and MongoDB, along with AWS services such as EC2 and S3. 1. How do you ensure that the versions of these libraries and services are up-to-date and free from known vulnerabilities?

Q7.5 iOS applications are being launched, but there is no mention of the NSAllowsArbitraryLoads setting or Application Transport Security (ATS) configuration in the Info.plist file, and no minimum OS version requirement is specified. 1. How is the application ensuring secure data transmission without the ATS configuration in place? 2. What is the minimum OS version requirement for the iOS apps to ensure they include essential security features and updates?

---

8. Error Handling and Logging

Security Requirements	Open Questions
1	0

Security Requirements

	State	Ranking
8.1 Exclude passwords from application logs	Open	High

Description

Context: The application logs plaintext passwords temporarily for behavioral analytics in the POST /auth/login endpoint.

Gap: There is no mention of encryption or masking of sensitive data, specifically passwords, in server-side logs.

Threat: An attacker can access logs to retrieve plaintext passwords, leading to unauthorized access, data breaches, and potential compromise of user accounts.

Remediation

Sensitive data, especially passwords, should never be logged. Implement a logging policy that excludes sensitive information from being recorded. If logging is necessary for debugging, ensure that sensitive data is masked or redacted before being logged.

Standard Mapping

ASVS: V7.3.3 V7.3.1 V7.1.1 V7.1.2 V7.1.3

---

9. Stored Cryptography

Security Requirements	Open Questions
6	1

Security Requirements

	State	Ranking
9.1 Replace SHA-1 for password storage in signup API	Open	High

Description

Context: The application uses SHA-1 hashing for storing passwords in the 'POST /auth/signup' API.

Gap: While SHA-1 hashing is implemented for password storage, it is deprecated and lacks the necessary cryptographic strength to protect against modern attacks.

Threat: An attacker can exploit the weaknesses in SHA-1 to perform collision attacks, potentially compromising user passwords and gaining unauthorized access to user accounts.

Remediation

Replace SHA-1 with a more secure hashing algorithm such as bcrypt, Argon2, or PBKDF2 with SHA-256 for password storage. This will enhance the security of stored passwords by making them resistant to collision and brute-force attacks.

Standard Mapping

ASVS: V6.2.5

9.2 Use cryptographically secure methods for generating JWT and API tokens

Open Medium

Description

Context: The application uses JWT tokens for authentication, generated at login/signup with user ID and role, and signed using a secret key stored in AWS Secrets Manager. Service-level API tokens for microservices authentication are also stored in Secrets Manager.

Gap: The document lacks explicit references to cryptographically secure methods for unique ID generation, such as UUIDv4 or SecureRandom.

Threat: An attacker could exploit predictable or insecure unique ID generation to impersonate users or services, leading to unauthorized access and potential data breaches.

Remediation

Implement cryptographically secure methods for unique ID generation, such as using UUIDv4 or SecureRandom in Java, or the secrets module in Python. Ensure that all unique IDs are generated using a method that is resistant to prediction and collision.

Standard Mapping

ASVS: V6.3.2 V6.3.1

9.3 Specify encryption algorithms for MongoDB TLS communications

Open Medium

Description

Context: MongoDB is hosted with TLS encryption enabled, and SSL encryption is noted for secure communications within the application.

Gap: While encryption is mentioned, the document does not specify the encryption algorithms used, leaving uncertainty about their strength and effectiveness.

Threat: An attacker could exploit weak encryption algorithms to intercept and decrypt sensitive data, compromising data integrity and confidentiality.

Remediation

Ensure that all encryption implementations use strong, industry-standard algorithms such as AES-256 for symmetric encryption and RSA-2048 or higher for asymmetric encryption. Update documentation to specify the algorithms used.

Standard Mapping

ASVS: V6.2.2 V2.9.3 V6.2.5 V6.2.3 V6.2.4

9.4 Enable encryption at rest for Amazon RDS MongoDB instances

Open High

Description

Context: Amazon RDS is used for managing MongoDB instances within the application, with TLS encryption mentioned for data in transit.

Gap: While TLS encryption is implemented for data in transit, there is no explicit mention of encryption at rest for Amazon RDS instances, leaving stored data potentially unprotected.

Threat: An attacker with access to the underlying storage could extract sensitive data from the database in clear text, leading to data exposure and potential compliance violations.

Remediation

Ensure that encryption at rest is enabled for all Amazon RDS instances by using AWS KMS to manage encryption keys. This can be configured during the creation of the RDS instance or by modifying existing instances to enable encryption.

Standard Mapping

ASVS: V8.1.6 V6.1.2 V6.1.3 V6.1.1

9.5 Encrypt PII and financial data at rest

Open High

Description

Context: The application processes sensitive data, including PII and financial information, and uses JWT secrets stored in AWS Secrets Manager. It mentions PCI-DSS compliance for payments and KYC verification for sellers.

Gap: The document lacks explicit mention of encryption for sensitive data at rest using strong cryptographic algorithms.

Threat: An attacker could access unencrypted sensitive data at rest, leading to data breaches, identity theft, or financial fraud.

Remediation

Implement encryption for all sensitive data at rest using strong cryptographic algorithms such as AES-256. Ensure that encryption keys are managed securely, possibly using a service like AWS Key Management Service (KMS). Regularly audit and update encryption practices to align with current security standards.

Standard Mapping

PCI-DSS: 3.6.1.4 3.6.1.2 3.7.3 3.7.1 3.6.1

ASVS: V8.3.7 V6.1.2 V6.2.2 V6.2.7 V6.1.1 V6.1.3

9.6 Use secure algorithms for generating password reset tokens

Open High

Description

Context: The application uses a 'Forgot Password' feature that allows users to request a password reset via email, indicating the presence of a password reset token mechanism.

Gap: The document does not specify the algorithm used for generating the password reset token, leaving the security of the token generation process unclear.

Threat: An attacker could potentially predict or reuse password reset tokens, leading to unauthorized access to user accounts.

Remediation

Implement a secure token generation algorithm such as HMAC with SHA-256 or UUIDs to ensure that password reset tokens are cryptographically secure and resistant to prediction or reuse attacks.

Standard Mapping

ASVS: V6.3.1 V6.2.2 V2.9.3

Open Questions

Q9.1 Encryption is used in the application, including TLS for MongoDB and SSL for secure communications. 1. What key sizes are used for the encryption protocols to ensure they are strong enough to prevent brute-force attacks? 2. How is sufficient entropy ensured in the generation of encryption keys?

---

10. Files and Resources

Security Requirements	Open Questions
1	0

Security Requirements

	State	Ranking
10.1 Implement security checks for image uploads in seller experience	Open	High

Description

Context: The application allows users to upload images as part of the product listing process, which is described under the 'Seller Experience' section.

Gap: The document lacks explicit references to security checks for file uploads, including file name validation, content type and file type checks, MIME type verification, file size checks, and access control on files.

Threat: An attacker could exploit the absence of these security checks to upload malicious files, potentially leading to unauthorized access, data corruption, or execution of harmful scripts.

Remediation

Implement comprehensive file upload security measures. Validate file names to prevent special characters that could lead to injection attacks. Enforce strict content type and file type checks to ensure only allowed file types are uploaded. Verify MIME types to prevent bypassing security controls. Implement file size restrictions to prevent DoS attacks. Ensure proper access control mechanisms are in place to protect uploaded files from unauthorized access, updates, or deletions. If the application is using a specific programming language, such as Python, consider using libraries like 'python-magic' for MIME type checking and 'os.path' for file path validation.

Standard Mapping

ASVS: V12.5.1 V12.4.1 V12.2.1 V1.12.2 V11.1.4 V12.4.2 V12.3.4 V12.3.3 V5.3.9 V12.1.3 V12.3.2 V12.5.2 V12.3.1 V12.1.1 V5.2.6

---

11. Access Control

Security Requirements	Open Questions
1	4

Security Requirements

	State	Ranking
11.1 Implement role-based access control for financial and PII data	Open	High

Description

Context: The application uses JWT authentication with user roles to manage access, and JWT secrets are stored in AWS Secrets Manager. MongoDB access is limited to read-only for analytics and write/delete for respective microservices.

Gap: While JWT authentication with user roles is present, there is no explicit mention of role-based access control (RBAC) specifically for sensitive data such as financial information and PII.

Threat: An attacker could exploit the lack of specific RBAC for sensitive data to gain unauthorized access to financial information and PII, leading to data breaches and potential misuse of sensitive information.

Remediation

Implement role-based access control (RBAC) specifically for sensitive data. Ensure that access to sensitive information is restricted based on user roles and permissions. For example, use Django's built-in permissions system to enforce access controls on sensitive data endpoints.

Additionally, review and update the access control policies regularly to ensure they align with security best practices.

Standard Mapping

ASVS: V1.1.3 V4.1.1 V1.4.5 V1.4.4 V4.1.2 V4.1.3

Open Questions

Q11.1 Amazon S3 is used for storing static assets and user-uploaded content, including business-critical or sensitive data. 1. Is MFA Delete implemented to protect against unauthorized deletions of critical data in the S3 buckets?

Q11.2 Amazon S3 is used for storing static assets and user-uploaded content. 1. What access control measures are in place to secure the data stored in the S3 buckets?

Q11.3 JWT tokens are used for role-based access control, containing user IDs and roles like 'buyer' and 'seller.' 1. How do you ensure server-side validation and authorization checks to prevent parameter tampering with JWT tokens?

Q11.4 JWT-based token verification and authentication mechanisms are used, with role-based access controls for roles like 'buyer' and 'seller' set at registration. 1. Are there any authorization frameworks like OAuth or OpenID Connect implemented to prevent forced browsing of APIs and URLs?

---

12. Validation

Security Requirements	Open Questions
0	2

Open Questions

Q12.1 Amazon EKS, RDS, S3, and CloudFormation are used for infrastructure provisioning in the application. CloudFormation templates are utilized to manage and deploy AWS resources. 1. How is input validation and sanitization implemented for CloudFormation templates to prevent injection attacks?

Q12.2 JSON is used in API requests and responses within a Django REST Framework-based backend for endpoints like 'POST /auth/login' and 'POST /auth/signup'. 1. How is input validation and escaping implemented to prevent JSON injection attacks in these API endpoints?

---

13. Sanitization and Encoding

Security Requirements	Open Questions
0	2

Open Questions

Q13.1 Amazon EKS, RDS, S3, and CloudFormation are used for infrastructure provisioning in the application. CloudFormation templates are utilized to manage and deploy AWS resources. 1. How is input validation and sanitization implemented for CloudFormation templates to prevent injection attacks?

Q13.2 JSON is used in API requests and responses within a Django REST Framework-based backend for endpoints like 'POST /auth/login' and 'POST /auth/signup'. 1. How is input validation and escaping implemented to prevent JSON injection attacks in these API endpoints?

Security Controls

Access Control

The application employs numerical and role-based identifiers, such as 'user_id' and 'role' in JWT tokens, generated during login/signup to define user roles and access levels. These identifiers are crucial for access control and are included in the response of the GET /account/settings API to support dynamic UI permissions.

JWT-based token verification is used for authentication, with tokens included in headers rather than cookies, as indicated by 'Authorization: Bearer <token>' in middleware checks. This approach, along with JSON body requests for API endpoints like POST /auth/login, mitigates CSRF risks through secure design practices.

Role-Based Access Control (RBAC) is implemented using JWT tokens that include a 'role' field to manage access to state-changing operations. The token verification process checks for role mismatches, ensuring permissions are aligned with user records from the database, supporting the RBAC framework.

Authentication

MongoDB access and security involve authentication via x.509 certificates and SCRAM-SHA, indicating the use of TLS certificates for authentication. Password-based authentication is implemented using an 'email' and 'password' combination, with error messages for invalid credentials confirming this method. JWT (JSON Web Token) authentication is used, with tokens generated at login/signup, signed with a secret key stored in AWS Secrets Manager. Access tokens expire in 2 hours, and refresh tokens are valid for 30 days, with token verification using the 'Authorization: Bearer <token>' header.

The application employs a microservices architecture, including services like Auth Service, Payment Service, and Order Service. Authentication between these services uses service-level API tokens stored in AWS Secrets Manager. Each service connects to MongoDB with unique credentials stored in AWS Secrets Manager. JWTs are used extensively, with token generation, expiry, and refresh details outlined, and middleware in Django verifying tokens via the Authorization header.

The 'exp' claim in JWT tokens indicates a default expiry of 2 hours, aligning with security best practices. A change password feature verifies the old password before accepting changes, and a 'Forgot Password' feature allows password reset requests via email. Password reset links expire after 15 minutes, indicating an expiry policy for reset tokens, though their single-use nature is not specified.

Communication

Communication within the application utilizes HTTPS via AWS ACM, ensuring secure, authenticated exchanges, supported by JWT-based token verification. Internal API calls can be made using either HTTP or HTTPS, with HTTPS providing encryption for data transmission, while HTTP does not. The technology stack confirms the use of HTTPS for secure data transport. MongoDB is hosted with TLS encryption, further securing communications.

Data Protection

Payment method details are returned with the card type, masked number, and expiry date, with only the last four digits visible, ensuring sensitive data is properly masked. Shared user functions also display only the last four digits of credit cards, confirming the consistent masking of sensitive information.

The JWT payload includes 'user_id' and 'role', without storing sensitive information like emails or passwords. JWTs are signed with a secret key stored in AWS Secrets Manager, ensuring that only non-sensitive data such as user ID and role are included in the payload.

AWS Secrets Manager is used to secure sensitive information, including JWT secrets and database credentials. Microservices authenticate using service-level API tokens stored in Secrets Manager, indicating that Kubernetes secrets in the EKS cluster are securely managed with AWS Secrets Manager, a recognized external secrets management tool.

Error Handling and Logging

The document explicitly mentions the use of AWS CloudWatch Logs for ECS service health, which indicates that logging is enabled and centralized for ECS services. Additionally, the use of the ELK Stack for full-text search on logs further supports the presence of a centralized logging system. These references suggest that the logging setup follows best practices for security and monitoring, as it involves log aggregation and centralized management.

Malicious Code

The document provides explicit references to the secure management of secrets, keys, or credentials. It mentions that each service connects via its own credential stored in AWS Secrets Manager, and that microservices authenticate using service-level API tokens stored in Secrets Manager. Additionally, it is stated that the JWT signing secret is stored in AWS Secrets Manager. These references indicate that the application follows best practices for secret management by using a secure storage solution.

Session Management

The document mentions that JWT tokens are stored and rotated securely, indicating that the application facilitates token rotation. This is supported by the statement in the Technology Stack section: 'Auth & Tokens | JWT (stored & rotated securely)'.

Stored Cryptography

SHA1 hashing is used for storing passwords in the 'POST /auth/signup' API, indicating the application employs hashing techniques for password security. JWT tokens are used for authentication, generated at login/signup with user ID and role, and signed with a secret key

stored in AWS Secrets Manager. Service-level API tokens for microservices authentication are also stored in Secrets Manager, highlighting secure token management practices.

Encryption is implemented through TLS for MongoDB hosting, ensuring secure data transmission, and SSL for secure communications, demonstrating the application's commitment to data security. AWS Secrets Manager is utilized for storing sensitive information like JWT secrets and database credentials, providing encrypted storage and secure retrieval, which aligns with best practices for secret management. MongoDB access is secured with TLS encryption, and each service uses its own credentials stored in AWS Secrets Manager, further ensuring secure secret management.

Validation, Sanitization and Encoding

The document mentions several sources of user input, including user registration via email or Google/Facebook login, login with email/password or social login, and password reset requests. Additionally, users can upload images, titles, categories, prices, and quantities when listing a product. These inputs are user-generated and can be considered untrusted. The document also describes API endpoints that accept user input, such as POST /auth/signup and POST /auth/login, which process user credentials.

Identified Assets

REST endpoint

1. /auth/logout

Invalidates current JWT. (POST)

2. /account/payment-methods?card_number=4242424242424242&expiry=12/25&cvv=234

Adds payment method. (POST)

3. /auth/login

Authenticates users and returns access/refresh tokens. (POST)

4. /products/{id}

Get product details (GET), update product details (PUT), or delete a product (DELETE, seller-only) at the /products/{id} endpoint.

5. /stridecredit/balance

Get current balance. (GET)

6. /orders

Place order from cart and list past orders. (POST, GET)

7. /account/payment-methods/{id}

Deletes a payment method. (DELETE)

8. /products/{id}/comment

Comment on a product. (POST)

9. /orders/{id}

Get specific order details. (GET)

10. /auth/signup

Registers a new user. (POST)

11. /payments/confirm

Confirm payment (called by gateway webhook). (POST)

12. /auth/change-password

Allows users to change passwords. (POST)

13. /stridecredit/redeem

Use credits during checkout. (POST)

14. /cart

Returns the user's cart and adds items to it. (GET, POST)

15. /account/payment-methods

Lists payment methods. (GET)

16. /products/{id}/follow

Follow product for updates. (POST)

17. /products/{id}/like

Like a product. (POST)

18. /payments/initiate

Start payment session. (POST)

19. /account/settings

Retrieves and updates user profile. (GET, PUT)

20. /account/addresses/{id}

Updates (PUT) or deletes (DELETE) a user's address by address ID, as part of user account management.

21. /payments/history

List previous payments. (GET)

22. /auth/refresh

Endpoint for refreshing JWT tokens using refresh token. (POST)

23. /cart/{item_id}

Update cart item quantity (PUT) or remove items from cart (DELETE) as described in the technical specification for cart management.

24. /products

Returns products with filters and allows sellers to list new products. (GET, POST)

25. /account/addresses

Lists saved addresses and adds a new address. (GET, POST)

Cloud

1. AWS CodeBuild

AWS CodeBuild is part of the CI/CD pipeline, used for automated testing and deployment, playing a key role in automating the build and test phases.

2. AWS ElastiCache

AWS ElastiCache (Redis) handles session storage and caching, improving application performance and scalability.

3. AWS ALB

AWS ALB (Application Load Balancer) manages ingress resources and provisions load balancers, ensuring efficient traffic distribution and application availability.

4. AWS S3

AWS S3 is used for storing static assets and user-uploaded content, offering scalable storage solutions for various data types.

5. AWS IAM

AWS IAM manages identities and access, ensuring secure and controlled access to AWS resources.

6. AWS ACM

AWS ACM manages HTTPS communication, ensuring secure connections and data integrity.

7. AWS CodePipeline

AWS CodePipeline is utilized for implementing the CI/CD pipeline, facilitating automated testing and deployment to streamline the development process.

8. AWS VPC

AWS VPC provides network isolation and security as part of the infrastructure setup, enhancing network management and protection.

9. AWS Secrets Manager

AWS Secrets Manager secures sensitive information like JWT secrets and database credentials, ensuring secure key management and access control for services.

10. AWS EC2

AWS EC2 is used for hosting the self-managed MongoDB database, providing flexibility in database management and deployment options.

11. AWS ECR

AWS ECR is used for storing Docker images as part of the CI/CD process, integrating with ECS for efficient container management and deployment.

12. AWS EKS

AWS EKS manages the Kubernetes control plane, facilitating efficient orchestration and management of containerized applications.

13. AWS CloudWatch

AWS CloudWatch is used for monitoring and logging, providing visibility into application logs, API Gateway metrics, and ECS service health as part of the platform's operational monitoring.

Personal Identifiable Information (PII)

1. Phone number

User data includes phone number for account management and notifications. Provides full order details including shipping address and phone number for operational use.

2. Full name

User data includes full name for registration and account management.

3. Personal email addresses

User data includes email for signup/login and notifications. Users can update their email in profile settings.

4. Personal information such as first name

The response from the GET /account/settings endpoint includes the user's name, such as 'John'.

5. Address

User data includes addresses for payment methods and order fulfillment. Users can add, edit, delete addresses, and set a default address.

Authentication and Access

1. JWT tokens

JWT tokens are generated at login/signup with user ID and role, signed using a secret key stored in AWS Secrets Manager. They authenticate users and return access/refresh tokens. A new JWT can be generated using a refresh token while the original remains valid.

2. API keys

Service-level API tokens used by microservices for authentication, stored securely in AWS Secrets Manager.

3. Bearer tokens

Bearer tokens are used in the Authorization header for authentication. Middleware in Django checks Authorization: Bearer <token> for token verification.

4. Session tokens

Session tokens support multiple active sessions per user without restriction and support calls from any valid session token.

5. Password

Passwords are used for user authentication and credential changes. Old and new password hashes are logged for diagnostics, and plaintext passwords are temporarily logged server-side for behavioral analytics.

Financial Information

1. StrideCredit balance

User data includes StrideCredit balance for transactions and rewards.

2. Payment methods

User data includes payment methods for secure transactions.

3. Credit card numbers

Card details, including full credit card numbers, are submitted as query parameters and logged before tokenization. Users can add credit cards, but only the last 4 digits are shown for identification in the UI.

Compliance

1. PCI-DSS (Payment Card Industry Data Security Standard)

Answer: Yes

Explanation: The project involves handling payment card information, as indicated by the mention of secure payment methods including credit cards and gift cards. The documents specify PCI-DSS compliance for payments, which directly relates to the storage, processing, and transmission of cardholder data. Additionally, the payment service handles gateway integration and transaction ledger, further necessitating PCI-DSS compliance.

Appendix: Security Mapping

ASVS

Section	Description
V9.2.4	Verify that proper certification revocation, such as Online Certificate Status Protocol (OCSP) Stapling, is enabled and configured.
V1.6.1	Verify that there is an explicit policy for management of cryptographic keys and that a cryptographic key lifecycle follows a key management standard such as NIST SP 800-57.
V7.3.1	Verify that all logging components appropriately encode data to prevent log injection.
V8.1.1	Verify the application protects sensitive data from being cached in server components such as load balancers and application caches.
V8.2.1	Verify the application sets sufficient anti-caching headers so that sensitive data is not cached in modern browsers.
V8.3.1	Verify that sensitive data is sent to the server in the HTTP message body or headers, and that query string parameters from any HTTP verb do not contain sensitive data.
V8.2.2	Verify that data stored in browser storage (such as localStorage, sessionStorage, IndexedDB, or cookies) does not contain sensitive data.
V2.2.1	Verify that anti-automation controls are effective at mitigating breached credential testing, brute force, and account lockout attacks. Such controls include blocking the most common breached passwords, soft lockouts, rate limiting, CAPTCHA, ever increasing delays between attempts, IP address restrictions, or risk-based restrictions such as location, first login on a device,

Section	Description
	recent attempts to unlock the account, or similar. Verify that no more than 100 failed attempts per hour is possible on a single account.
V11.1.8	Verify that the application has configurable alerting when automated attacks or unusual activity is detected.
V11.1.4	Verify that the application has anti-automation controls to protect against excessive calls such as mass data exfiltration, business logic requests, file uploads or denial of service attacks.
V8.1.4	Verify the application can detect and alert on abnormal numbers of requests, such as by IP, user, total per hour or day, or whatever makes sense for the application.
V11.1.7	Verify that the application monitors for unusual events or activity from a business logic perspective. For example, attempts to perform actions out of order or actions which a normal user would never attempt.
V1.2.1	Verify the use of unique or special low-privilege operating system accounts for all application components, services, and servers.
V1.14.5	Verify that application deployments adequately sandbox, containerize and/or isolate at the network level to delay and deter attackers from attacking other applications, especially when they are performing sensitive or dangerous actions such as deserialization.
V10.3.2	Verify that the application employs integrity protections, such as code signing or subresource integrity. The application must not load or execute code from untrusted sources, such as loading includes, modules, plugins, code, or libraries from untrusted sources or the Internet.
V7.3.3	Verify that security logs are protected from unauthorized access and modification.
V7.1.1	Verify that the application does not log credentials or payment details. Session tokens should only be stored in logs in an irreversible, hashed form.

Section	Description
V7.1.2	Verify that the application does not log other sensitive data as defined under local privacy laws or relevant security policy.
V7.1.3	Verify that the application logs security relevant events including successful and failed authentication events, access control failures, deserialization failures and input validation failures.
V11.1.6	Verify that the application does not suffer from "Time Of Check to Time Of Use" (TOCTOU) issues or other race conditions for sensitive operations.
V1.11.2	Verify that all high-value business logic flows, including authentication, session management and access control, do not share unsynchronized state.
V1.11.3	Verify that all high-value business logic flows, including authentication, session management and access control are thread safe and resistant to time-of-check and time-of-use race conditions.
V8.3.4	Verify that all sensitive data created and processed by the application has been identified, and ensure that a policy is in place on how to deal with sensitive data.
V6.2.5	Verify that known insecure block modes (i.e. ECB, etc.), padding modes (i.e. PKCS#1 v1.5, etc.), ciphers with small block sizes (i.e. Triple-DES, Blowfish, etc.), and weak hashing algorithms (i.e. MD5, SHA1, etc.) are not used unless required for backwards compatibility.
V12.5.1	Verify that the web tier is configured to serve only files with specific file extensions to prevent unintentional information and source code leakage. For example, backup files (e.g. .bak), temporary working files (e.g. .swp), compressed files (.zip, .tar.gz, etc) and other extensions commonly used by editors should be blocked unless required.
V12.4.1	Verify that files obtained from untrusted sources are stored outside the web root, with limited permissions.

Section	Description
V12.2.1	Verify that files obtained from untrusted sources are validated to be of expected type based on the file's content.
V1.12.2	Verify that user-uploaded files - if required to be displayed or downloaded from the application - are served by either octet stream downloads, or from an unrelated domain, such as a cloud file storage bucket. Implement a suitable Content Security Policy (CSP) to reduce the risk from XSS vectors or other attacks from the uploaded file.
V12.4.2	Verify that files obtained from untrusted sources are scanned by antivirus scanners to prevent upload and serving of known malicious content.
V12.3.4	Verify that the application protects against Reflective File Download (RFD) by validating or ignoring user-submitted filenames in a JSON, JSONP, or URL parameter, the response Content-Type header should be set to text/plain, and the Content-Disposition header should have a fixed filename.
V12.3.3	Verify that user-submitted filename metadata is validated or ignored to prevent the disclosure or execution of remote files via Remote File Inclusion (RFI) or Server-side Request Forgery (SSRF) attacks.
V5.3.9	Verify that the application protects against Local File Inclusion (LFI) or Remote File Inclusion (RFI) attacks.
V12.1.3	Verify that a file size quota and maximum number of files per user is enforced to ensure that a single user cannot fill up the storage with too many files, or excessively large files.
V12.3.2	Verify that user-submitted filename metadata is validated or ignored to prevent the disclosure, creation, updating or removal of local files (LFI).
V12.5.2	Verify that direct requests to uploaded files will never be executed as HTML/JavaScript content.
V12.3.1	Verify that user-submitted filename metadata is not used directly by system or framework filesystems and that a URL API is used to protect against path

Section	Description
	traversal.
V12.1.1	Verify that the application will not accept large files that could fill up storage or cause a denial of service.
V5.2.6	Verify that the application protects against SSRF attacks, by validating or sanitizing untrusted data or HTTP file metadata, such as filenames and URL input fields, and uses allow lists of protocols, domains, paths and ports.
V1.1.3	Verify that all user stories and features contain functional security constraints, such as "As a user, I should be able to view and edit my profile. I should not be able to view or edit anyone else's profile"
V4.1.1	Verify that the application enforces access control rules on a trusted service layer, especially if client-side access control is present and could be bypassed.
V1.4.5	Verify that attribute or feature-based access control is used whereby the code checks the user's authorization for a feature/data item rather than just their role. Permissions should still be allocated using roles.
V1.4.4	Verify the application uses a single and well-vetted access control mechanism for accessing protected data and resources. All requests must pass through this single mechanism to avoid copy and paste or insecure alternative paths.
V4.1.2	Verify that all user and data attributes and policy information used by access controls cannot be manipulated by end users unless specifically authorized.
V4.1.3	Verify that the principle of least privilege exists - users should only be able to access functions, data files, URLs, controllers, services, and other resources, for which they possess specific authorization. This implies protection against spoofing and elevation of privilege.

Section	Description
V6.3.2	Verify that random GUIDs are created using the GUID v4 algorithm, and a Cryptographically-secure Pseudo-random Number Generator (CSPRNG). GUIDs created using other pseudo-random number generators may be predictable.
V6.3.1	Verify that all random numbers, random file names, random GUIDs, and random strings are generated using the cryptographic module's approved cryptographically secure random number generator when these random values are intended to be not guessable by an attacker.
V9.2.1	Verify that connections to and from the server use trusted TLS certificates. Where internally generated or self-signed certificates are used, the server must be configured to only trust specific internal CAs and specific self-signed certificates. All others should be rejected.
V1.9.2	Verify that application components verify the authenticity of each side in a communication link to prevent person-in-the-middle attacks. For example, application components should validate TLS certificates and chains.
V9.2.3	Verify that all encrypted connections to external systems that involve sensitive information or functions are authenticated.
V6.2.2	Verify that industry proven or government approved cryptographic algorithms, modes, and libraries are used, instead of custom coded cryptography.
V2.9.3	Verify that approved cryptographic algorithms are used in the generation, seeding, and verification.
V6.2.3	Verify that encryption initialization vector, cipher configuration, and block modes are configured securely using the latest advice.
V6.2.4	Verify that random number, encryption or hashing algorithms, key lengths, rounds, ciphers or modes, can be reconfigured, upgraded, or swapped at any time, to protect against cryptographic breaks.

Section	Description
V2.10.1	Verify that intra-service secrets do not rely on unchanging credentials such as passwords, API keys or shared accounts with privileged access.
V1.6.4	Verify that the architecture treats client-side secrets--such as symmetric keys, passwords, or API tokens--as insecure and never uses them to protect or access sensitive data.
V6.4.1	Verify that a secrets management solution such as a key vault is used to securely create, store, control access to and destroy secrets.
V2.10.4	Verify passwords, integrations with databases and third-party systems, seeds and internal secrets, and API keys are managed securely and not included in the source code or stored within source code repositories. Such storage SHOULD resist offline attacks. The use of a secure software key store (L1), hardware TPM, or an HSM (L3) is recommended for password storage.
V5.3.4	Verify that data selection or database queries (e.g. SQL, HQL, ORM, NoSQL) use parameterized queries, ORMs, entity frameworks, or are otherwise protected from database injection attacks.
V5.5.3	Verify that deserialization of untrusted data is avoided or is protected in both custom code and third-party libraries (such as JSON, XML and YAML parsers).
V5.1.3	Verify that all input (HTML form fields, REST requests, URL parameters, HTTP headers, cookies, batch files, RSS feeds, etc) is validated using positive validation (allow lists).
V5.2.4	Verify that the application avoids the use of eval() or other dynamic code execution features. Where there is no alternative, any user input being included must be sanitized or sandboxed before being executed.
V5.3.8	Verify that the application protects against OS command injection and that operating system calls use parameterized OS queries or use contextual command line output encoding.

Section	Description
V5.3.3	Verify that context-aware, preferably automated - or at worst, manual - output escaping protects against reflected, stored, and DOM based XSS.
V5.3.1	Verify that output encoding is relevant for the interpreter and context required. For example, use encoders specifically for HTML values, HTML attributes, JavaScript, URL parameters, HTTP headers, SMTP, and others as the context requires, especially from untrusted inputs (e.g. names with Unicode or apostrophes, such as ねこ or O'Hara).
V5.2.5	Verify that the application protects against template injection attacks by ensuring that any user input being included is sanitized or sandboxed.
V5.2.8	Verify that the application sanitizes, disables, or sandboxes user-supplied scriptable or expression template language content, such as Markdown, CSS or XSL stylesheets, BBCode, or similar.
V5.3.5	Verify that where parameterized or safer mechanisms are not present, context-specific output encoding is used to protect against injection attacks, such as the use of SQL escaping to protect against SQL injection.
V5.3.10	Verify that the application protects against XPath injection or XML injection attacks.
V5.2.7	Verify that the application sanitizes, disables, or sandboxes user-supplied Scalable Vector Graphics (SVG) scriptable content, especially as they relate to XSS resulting from inline scripts, and foreignObject.
V2.10.3	Verify that passwords are stored with sufficient protection to prevent offline recovery attacks, including local system access.
V2.4.1	Verify that passwords are stored in a form that is resistant to offline attacks. Passwords SHALL be salted and hashed using an approved one-way key derivation or password hashing function. Key derivation and password hashing functions take a password, a salt, and a cost factor as inputs when generating a password hash.

Section	Description
V2.4.2	Verify that the salt is at least 32 bits in length and be chosen arbitrarily to minimize salt value collisions among stored hashes. For each credential, a unique salt value and the resulting hash SHALL be stored.
V2.9.1	Verify that cryptographic keys used in verification are stored securely and protected against disclosure, such as using a Trusted Platform Module (TPM) or Hardware Security Module (HSM), or an OS service that can use this secure storage.
V10.2.1	Verify that the application source code and third party libraries do not contain unauthorized phone home or data collection capabilities. Where such functionality exists, obtain the user's permission for it to operate before collecting any data.
V10.2.2	Verify that the application does not ask for unnecessary or excessive permissions to privacy related features or sensors, such as contacts, cameras, microphones, or location.
V10.2.3	Verify that the application source code and third party libraries do not contain back doors, such as hard-coded or additional undocumented accounts or keys, code obfuscation, undocumented binary blobs, rootkits, or anti-debugging, insecure debugging features, or otherwise out of date, insecure, or hidden functionality that could be used maliciously if discovered.
V8.1.6	Verify that backups are stored securely to prevent data from being stolen or corrupted.
V6.1.2	Verify that regulated health data is stored encrypted while at rest, such as medical records, medical device details, or de-anonymized research records.
V6.1.3	Verify that regulated financial data is stored encrypted while at rest, such as financial accounts, defaults or credit history, tax records, pay history, beneficiaries, or de-anonymized market or research records.

Section	Description
V6.1.1	Verify that regulated private data is stored encrypted while at rest, such as Personally Identifiable Information (PII), sensitive personal information, or data assessed likely to be subject to EU's GDPR.
V2.1.11	Verify that "paste" functionality, browser password helpers, and external password managers are permitted.
V2.1.12	Verify that the user can choose to either temporarily view the entire masked password, or temporarily view the last typed character of the password on platforms that do not have this as built-in functionality.
V8.3.7	Verify that sensitive or private information that is required to be encrypted, is encrypted using approved algorithms that provide both confidentiality and integrity.
V6.2.7	Verify that encrypted data is authenticated via signatures, authenticated cipher modes, or HMAC to ensure that ciphertext is not altered by an unauthorized party.
V2.1.7	Verify that passwords submitted during account registration, login, and password change are checked against a set of breached passwords either locally (such as the top 1,000 or 10,000 most common passwords which match the system's password policy) or using an external API. If using an API a zero knowledge proof or other mechanism should be used to ensure that the plain text password is not sent or used in verifying the breach status of the password. If the password is breached, the application must require the user to set a new non-breached password.
V2.1.1	Verify that user set passwords are at least 12 characters in length (after multiple spaces are combined).
V2.1.10	Verify that there are no periodic credential rotation or password history requirements.
V2.1.5	Verify users can change their password.

Section	Description
V2.1.3	Verify that password truncation is not performed. However, consecutive multiple spaces may be replaced by a single space.
V2.1.2	Verify that passwords of at least 64 characters are permitted, and that passwords of more than 128 characters are denied.
V2.1.8	Verify that a password strength meter is provided to help users set a stronger password.
V2.1.9	Verify that there are no password composition rules limiting the type of characters permitted. There should be no requirement for upper or lower case or numbers or special characters.
V2.1.6	Verify that password change functionality requires the user's current and new password.
V2.1.4	Verify that any printable Unicode character, including language neutral characters such as spaces and Emojis are permitted in passwords.
V2.3.2	Verify that enrollment and use of user-provided authentication devices are supported, such as a U2F or FIDO tokens.
V2.8.2	Verify that symmetric keys used to verify submitted OTPs are highly protected, such as by using a hardware security module or secure operating system based key storage.
V2.8.7	Verify that biometric authenticators are limited to use only as secondary factors in conjunction with either something you have and something you know.
V2.2.4	Verify impersonation resistance against phishing, such as the use of multi-factor authentication, cryptographic devices with intent (such as connected keys with a push to authenticate), or at higher AAL levels, client-side certificates.

Section	Description
V2.2.7	Verify intent to authenticate by requiring the entry of an OTP token or user-initiated action such as a button press on a FIDO hardware key.
V2.7.1	Verify that clear text out of band (NIST "restricted") authenticators, such as SMS or PSTN, are not offered by default, and stronger alternatives such as push notifications are offered first.
V2.2.2	Verify that the use of weak authenticators (such as SMS and email) is limited to secondary verification and transaction approval and not as a replacement for more secure authentication methods. Verify that stronger methods are offered before weak methods, users are aware of the risks, or that proper measures are in place to limit the risks of account compromise.
V2.8.5	Verify that if a time-based multi-factor OTP token is re-used during the validity period, it is logged and rejected with secure notifications being sent to the holder of the device.
V2.5.5	Verify that if an authentication factor is changed or replaced, that the user is notified of this event.
V2.5.7	Verify that if OTP or multi-factor authentication factors are lost, that evidence of identity proofing is performed at the same level as during enrollment.
V2.2.6	Verify replay resistance through the mandated use of One-time Passwords (OTP) devices, cryptographic authenticators, or lookup codes.
V2.8.3	Verify that approved cryptographic algorithms are used in the generation, seeding, and verification of OTPs.
V2.2.3	Verify that secure notifications are sent to users after updates to authentication details, such as credential resets, email or address changes, logging in from unknown or risky locations. The use of push notifications - rather than SMS or email - is preferred, but in the absence of push notifications, SMS or email is acceptable as long as no sensitive information is disclosed in the notification.

Section	Description
V2.5.6	Verify forgotten password, and other recovery paths use a secure recovery mechanism, such as time-based OTP (TOTP) or other soft token, mobile push, or another offline recovery mechanism.
V1.6.2	Verify that consumers of cryptographic services protect key material and other secrets by using key vaults or API based alternatives.
V1.6.3	Verify that all keys and passwords are replaceable and are part of a well-defined process to re-encrypt sensitive data.

PCI-DSS

Section	Description
6.2.1	Bespoke and custom software are developed securely, as follows: • Based on industry standards and/or best practices for secure development. • In accordance with PCI DSS (for example, secure authentication and logging). • Incorporating consideration of information security issues during each stage of the software development lifecycle. 6.2.1 Examine documented software development procedures to verify that processes are defined that include all elements specified in this requirement. Customized Approach Objective Bespoke and custom software is developed in accordance with PCI DSS and secure development processes throughout the software lifecycle. Applicability Notes This applies to all software developed for or by the entity for the entity's own use. This includes both bespoke and custom software. This does not apply to third-party software.
6.2.3	Bespoke and custom software is reviewed prior to being released into production or to customers, to identify and correct potential coding vulnerabilities, as follows: • Code reviews ensure code is developed according to secure coding guidelines. • Code reviews look for both existing and emerging software vulnerabilities. • Appropriate corrections are implemented prior to release. 6.2.3.a Examine documented software development procedures and interview responsible personnel to verify that processes are defined that require all bespoke and custom software to be reviewed in accordance with all elements specified in this requirement. 6.2.3.b Examine evidence of changes to bespoke and custom software to verify that the code changes were reviewed in accordance with all elements specified in this requirement. Customized Approach Objective Bespoke and custom software cannot be exploited via coding vulnerabilities. Applicability Notes

Section	Description
	This requirement for code reviews applies to all bespoke and custom software (both internal and public-facing), as part of the system development lifecycle. Public-facing web applications are also subject to additional controls, to address ongoing threats and vulnerabilities after implementation, as defined at PCI DSS Requirement 6.4. Code reviews may be performed using either manual or automated processes, or a combination of both.
7.2.5.1	All access by application and system accounts and related access privileges are reviewed as follows: • Periodically (at the frequency defined in the entity's targeted risk analysis, which is performed according to all elements specified in Requirement 12.3.1). • The application/system access remains appropriate for the function being performed. • Any inappropriate access is addressed. • Management acknowledges that access remains appropriate. 7.2.5.1.a Examine policies and procedures to verify they define processes to review all application and system accounts and related access privileges in accordance with all elements specified in this requirement. 7.2.5.1.b Examine the entity's targeted risk analysis for the frequency of periodic reviews of application and system accounts and related access privileges to verify the risk analysis was performed in accordance with all elements specified in Requirement 12.3.1. 7.2.5.1.c Interview responsible personnel and examine documented results of periodic reviews of system and application accounts and related privileges to verify that the reviews occur in accordance with all elements specified in this requirement. Customized Approach Objective Application and system account privilege assignments are verified periodically by management as correct, and nonconformities are remediated. Applicability Notes This requirement is a best practice until 31 March 2025, after which it will be required and must be fully considered during a PCI DSS assessment.
10.2.1.1	Audit logs capture all individual user access to cardholder data. 10.2.1.1 Examine audit log configurations and log data to verify that all individual user access to cardholder data is logged. Customized Approach Objective Records of all individual user access to cardholder data are captured.
10.2.1.2	Audit logs capture all actions taken by any individual with administrative access, including any interactive use of application or system accounts. 10.2.1.2 Examine audit log configurations and log data to verify that all actions taken by any individual with administrative access, including any interactive use of application or system accounts, are logged. Customized Approach Objective Records of all actions performed by individuals with elevated privileges are captured.

Section	Description
10.7.3	3 Failures of any critical security controls systems are responded to promptly, including but not limited to: • Restoring security functions. • Identifying and documenting the duration (date and time from start to end) of the security failure. • Identifying and documenting the cause(s) of failure and documenting required remediation. • Identifying and addressing any security issues that arose during the failure. • Determining whether further actions are required as a result of the security failure. • Implementing controls to prevent the cause of failure from reoccurring. • Resuming monitoring of security controls. 10.7.3.a Examine documentation and interview personnel to verify that processes are defined and implemented to respond to a failure of any critical security control system and include at least all elements specified in this requirement. 10.7.3.b Examine records to verify that failures of critical security control systems are documented to include: • Identification of cause(s) of the failure. • Duration (date and time start and end) of the security failure. • Details of the remediation required to address the root cause. Customized Approach Objective Failures of critical security control systems are analyzed, contained, and resolved, and security controls restored to minimize impact. Resulting security issues are addressed, and measures taken to prevent reoccurrence. Applicability Notes This requirement applies only when the entity being assessed is a service provider until 31 March 2025, after which this requirement will apply to all entities. This is a current v3.2.1 requirement that applies to service providers only. However, this requirement is a best practice for all other entities until 31 March 2025, after which it will be required and must be fully considered during a PCI DSS assessment.
10.2.1.4	Audit logs capture all invalid logical access attempts. 10.2.1.4 Examine audit log configurations and log data to verify that invalid logical access attempts are captured. Customized Approach Objective Records of all invalid access attempts are captured
6.2.4	Software engineering techniques or other methods are defined and in use by software development personnel to prevent or mitigate common software attacks and related vulnerabilities in bespoke and custom software, including but not limited to the following: • Injection attacks, including SQL, LDAP, XPath, or other command, parameter, object, fault, or injection-type flaws. • Attacks on data and data structures, including attempts to manipulate buffers, pointers, input data, or shared data. • Attacks on cryptography usage, including attempts to exploit weak, insecure, or inappropriate cryptographic implementations, algorithms, cipher suites, or modes of operation. • Attacks on business logic, including attempts to abuse or bypass application features and functionalities through the manipulation of APIs, communication protocols and channels, clientside functionality, or other system/application functions and resources. This includes cross-site scripting

Section	Description
	(XSS) and cross-site request forgery (CSRF). • Attacks on access control mechanisms, including attempts to bypass or abuse identification, authentication, or authorization mechanisms, or attempts to exploit weaknesses in the implementation of such mechanisms. • Attacks via any “high-risk” vulnerabilities identified in the vulnerability identification process, as defined in Requirement 6.3.1. 6.2.4 Examine documented procedures and interview responsible software development personnel to verify that software engineering techniques or other methods are defined and in use by developers of bespoke and custom software to prevent or mitigate all common software attacks as specified in this requirement
10.2.1.5	Audit logs capture all changes to identification and authentication credentials including, but not limited to: • Creation of new accounts. • Elevation of privileges. • All changes, additions, or deletions to accounts with administrative access. 10.2.1.5 Examine audit log configurations and log data to verify that changes to identification and authentication credentials are captured in accordance with all elements specified in this requirement. Customized Approach Objective Records of all changes to identification and authentication credentials are captured.
10.6.3	Time synchronization settings and data are protected as follows: • Access to time data is restricted to only personnel with a business need. • Any changes to time settings on critical systems are logged, monitored, and reviewed. 10.6.3.a Examine system configurations and timesynchronization settings to verify that access to time data is restricted to only personnel with a business need. 10.6.3.b Examine system configurations and time synchronization settings and logs and observe processes to verify that any changes to time settings on critical systems are logged, monitored, Customized Approach Objective and reviewed. System time settings cannot be modified by unauthorized personnel.
10.2.1.6	Audit logs capture the following: • All initialization of new audit logs, and • All starting, stopping, or pausing of the existing audit logs. 10.2.1.6 Examine audit log configurations and log data to verify that all elements specified in this requirement are captured. Customized Approach Objective Records of all changes to audit log activity status
4.2.1	Strong cryptography and security protocols are implemented as follows to safeguard PAN during transmission over open, public networks: • Only trusted keys and certificates are accepted. • Certificates used to safeguard PAN during transmission over open, public networks are confirmed as valid and are not expired or revoked. This bullet is a best practice until its effective date;

Section	Description
	refer to applicability notes below for details. • The protocol in use supports only secure versions or configurations and does not support fallback to, or use of insecure versions, algorithms, key sizes, or implementations. • The encryption strength is appropriate for the encryption methodology in use. 4.2.1.a Examine documented policies and procedures and interview personnel to verify processes are defined to include all elements specified in this requirement. 4.2.1.b Examine system configurations to verify that strong cryptography and security protocols are implemented in accordance with all elements specified in this requirement. 4.2.1.c Examine cardholder data transmissions to verify that all PAN is encrypted with strong cryptography when it is transmitted over open, public networks. 4.2.1.d Examine system configurations to verify that keys and/or certificates that cannot be verified as trusted are rejected. Customized Approach Objective Cleartext PAN cannot be read or intercepted from any transmissions over open, public networks.
4.2.2	PAN is secured with strong cryptography whenever it is sent via end-user messaging technologies. 4.2.2.a Examine documented policies and procedures to verify that processes are defined to secure PAN with strong cryptography whenever sent over end-user messaging technologies. 4.2.2.b Examine system configurations and vendor documentation to verify that PAN is secured with strong cryptography whenever it is sent via enduser messaging technologies. Customized Approach Objective Cleartext PAN cannot be read or intercepted from transmissions using end-user messaging technologies. Applicability Notes This requirement also applies if a customer, or other third-party, requests that PAN is sent to them via end-user messaging technologies. There could be occurrences where an entity receives unsolicited cardholder data via an insecure communication channel that was not intended for transmissions of sensitive data. In this situation, the entity can choose to either include the channel in the scope of their CDE and secure it according to PCI DSS or delete the cardholder data and implement measures to prevent the channel from being used for cardholder data.
8.3.1	All user access to system components for users and administrators is authenticated via at least one of the following authentication factors: • 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 element. 8.3.1.a Examine documentation describing the authentication factor(s) used to verify that user access to system components is authenticated via at least one authentication factor specified in this requirement. 8.3.1.b For each type of authentication factor used with each type of system component, observe an authentication to verify that authentication is functioning consistently with documented authentication factor(s). Customized Approach Objective An account cannot be accessed except with a combination of user identity and an authentication factor.

Section	Description
	Applicability Notes This requirement is not intended to apply to user accounts on point-of-sale terminals that have access to only one card number at a time to facilitate a single transaction (such as IDs used by cashiers on point-of-sale terminals). This requirement does not supersede multi-factor authentication (MFA) requirements but applies to those in-scope systems not otherwise subject to MFA requirements. A digital certificate is a valid option for “something you have” if it is unique for a particular user.
8.3.2	Strong cryptography is used to render all authentication factors unreadable during transmission and storage on all system components. 8.3.2.a Examine vendor documentation and system configuration settings to verify that authentication factors are rendered unreadable with strong cryptography during transmission and storage. 8.3.2.b Examine repositories of authentication factors to verify that they are unreadable during storage. 8.3.2.c Examine data transmissions to verify that authentication factors are unreadable during Customized Approach Objective transmission. Cleartext authentication factors cannot be obtained, derived, or reused from the interception of communications or from stored data.
8.3.11	Where authentication factors such as physical or logical security tokens, smart cards, or certificates are used: • Factors are assigned to an individual user and not shared among multiple users. • Physical and/or logical controls ensure only the intended user can use that factor to gain access. 8.3.11.a Examine authentication policies and procedures to verify that procedures for using authentication factors such as physical security tokens, smart cards, and certificates are defined and include all elements specified in this requirement. 8.3.11.b Interview security personnel to verify authentication factors are assigned to an individual user and not shared among multiple users. 8.3.11.c Examine system configuration settings and/or observe physical controls, as applicable, to verify that controls are implemented to ensure only the intended user can use that factor to gain access. Customized Approach Objective An authentication factor cannot be used by anyone other than the user to which it is assigned.
3.6.1.4	Cryptographic keys are stored in the fewest possible locations. 3.6.1.4 Examine key storage locations and observe processes to verify that keys are stored in the fewest possible locations. Customized Approach Objective Cryptographic keys are retained only
3.6.1.2	2 Secret and private keys used to encrypt/decrypt stored account data are stored in one (or more) of the following forms at all times: • Encrypted with a key-encrypting key that is at least as strong as the data-encrypting key, and

Section	Description
	that is stored separately from the dataencrypting key. • Within a secure cryptographic device (SCD), such as a hardware security module (HSM) or PTS-approved point-of-interaction device. • As at least two full-length key components or key shares, in accordance with an industry-accepted method. 3.6.1.2.a Examine documented procedures to verify it is defined that cryptographic keys used to encrypt/decrypt stored account data must exist only in one (or more) of the forms specified in this requirement. 3.6.1.2.b Examine system configurations and key storage locations to verify that cryptographic keys used to encrypt/decrypt stored account data exist in one (or more) of the forms specified in this requirement. 3.6.1.2.c Wherever key-encrypting keys are used, examine system configurations and key storage locations to verify: • Key-encrypting keys are at least as strong as the data-encrypting keys they protect. • Key-encrypting keys are stored separately from data-encrypting keys. Customized Approach Objective Secret and private keys are stored in a secure form that prevents unauthorized retrieval or access. Applicability Notes It is not required that public keys be stored in one of these forms. Cryptographic keys stored as part of a key management system (KMS) that employs SCDs are acceptable. A cryptographic key that is split into two parts does not meet this requirement. Secret or private keys stored as key components or key shares must be generated via one of the following: • Using an approved random number generator and within an SCD, OR • According to ISO 19592 or equivalent industry standard for generation of secret key shares..
3.7.3	Key-management policies and procedures are implemented to include secure storage of cryptographic keys used to Protect Stored Account Data 3.7.3.a Examine the documented key-management policies and procedures for keys used for protection of stored account data to verify that they define secure storage of cryptographic keys. 3.7.3.b Observe the method for storing keys to verify that keys are stored securely. Customized Approach Objective Cryptographic keys are secured when stored
3.7.1	Key-management policies and procedures are implemented to include generation of strong cryptographic keys used to Protect Stored Account Data 3.7.1.a Examine the documented key-management policies and procedures for keys used for protection of stored account data to verify that they define generation of strong cryptographic keys. 3.7.1.b Observe the method for generating keys to verify that strong keys are generated. Customized Approach Objective Strong cryptographic keys are generated.
3.6.1	Procedures are defined and implemented to protect cryptographic keys used to protect stored account data against disclosure and misuse that include: • Access to keys is restricted to the fewest number of custodians necessary. •

Section	Description
	Key-encrypting keys are at least as strong as the data-encrypting keys they protect. • Key-encrypting keys are stored separately from data-encrypting keys. • Keys are stored securely in the fewest possible locations and forms. 3.6.1 Examine documented key-management policies and procedures to verify that processes to protect cryptographic keys used to protect stored account data against disclosure and misuse are defined to include all elements specified in this requirement. Customized Approach Objective Processes that protect cryptographic keys used to protect stored account data against disclosure and misuse are defined and implemented. Applicability Notes This requirement applies to keys used to encrypt stored account data and to key-encrypting keys used to protect data-encrypting keys. The requirement to protect keys used to protect stored account data from disclosure and misuse applies to both data-encrypting keys and keyencrypting keys. Because one key-encrypting key may grant access to many data-encrypting keys, the key-encrypting keys require strong protection measures.
4.2.1.1	An inventory of the entity's trusted keys and certificates used to protect PAN during transmission is maintained. 4.2.1.1.a Examine documented policies and procedures to verify processes are defined for the entity to maintain an inventory of its trusted keys and certificates. 4.2.1.1.b Examine the inventory of trusted keys and certificates to verify it is kept up to date. Customized Approach Objective All keys and certificates used to protect PAN during transmission are identified and confirmed as trusted. Applicability Notes This requirement is a best practice until 31 March 2025, after which it will be required and must be fully considered during a PCI DSS assessment.