CIS API Security Guide

v1.0.0

MARCH 2025

Table of Contents

Overview

There are some key factors and changes in the web application industry. Both the trend of migrating existing services to the cloud (and newer companies being fully cloud-native), as well as the trends toward microservice architectures, have resulted in a radical increase in the number of Application Programming Interfaces (APIs) being used over the internet.

Imperva

- https://www.imperva.com/resources/resource-library/reports/the-state-of-api-security-in-2024
- API calls make up a massive 71% of all web traffic

CloudFlare

- https://radar.cloudflare.com/year-in-review/2024
- More than half of the dynamic traffic seen by CloudFlare is API related

Akamai

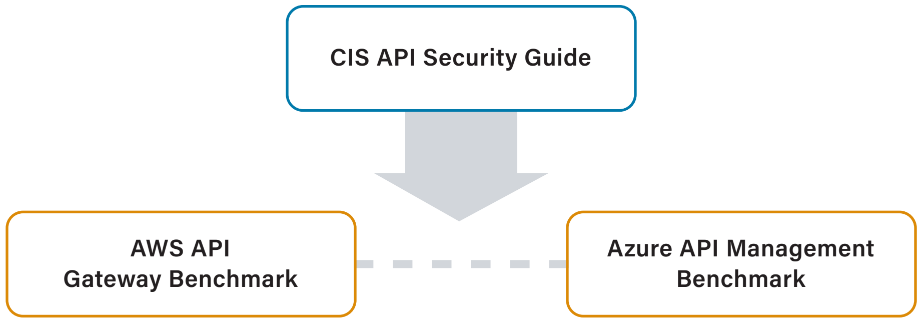
- https://www.akamai.com/blog/security/8-most-common-causes-of-data-breaches
- APIs account for 83% of all web traffic, making them a prime target for attackers.

The outsourcing to third parties of service functionalities such as payment processing, identity management, shipment and delivery, etc. has resulted in highly valuable and confidential data being sent over the network. Security and design flaws in the APIs handling this data are leading to data breaches, which are exacerbated by the automatic nature in which APIs can be used to exfiltrate data once a weakness has been identified.

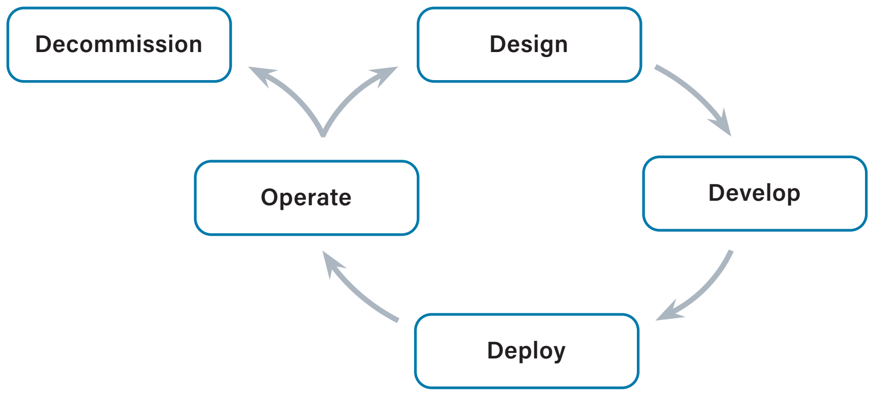
While the above situation is well understood, comprehensive resources for API security are still lacking. Many existing resources are general guidance or give only vague overviews of various threats and risks around APIs, but cannot be used by an organization wishing to assess its own security posture around APIs, API development, monitoring or other related activities. With this in mind, FireTail approached the Center for Internet Security (CIS) with the idea of developing a CIS Benchmark for API Security.

CIS has developed and published secure configuration guidance (i.e., CIS Benchmarks) covering a wide variety of technologies for many years. However, where Benchmarks target specific technologies or providers, APIs are implemented across a diverse range of platforms and technologies. How can we ensure consistent security recommendations across this diverse landscape?

The same challenges were previously encountered with the topic of supply chain security, which faces very similar considerations. It was decided to apply the same solution and instead of initially creating a specific Benchmark, a more general API security guidance document would be developed first to serve as the foundation for future platform-specific Benchmarks.



This guide is divided into sections that follow the API development lifecycle.



1 **Design:** recommendations for best practices during API design that will enable many of the recommendations in the later sections. This section outlines key considerations for designing secure and efficient APIs. It begins with API Documentation, which emphasizes the importance of clear, standardized documentation to ensure consistency and understanding among developers, consumers, and stakeholders. Next, Authentication and Authorization guidelines offer recommendations for implementing secure access control mechanisms, detailing various technologies used to protect API interactions. Lastly, Data Security and Privacy provides recommendations for conducting data assessments, enabling teams to identify the types of data handled by the API and implement appropriate protection measures to ensure privacy and security.

2 **Develop:** recommendations for security specific aspects and mechanisms of an API that need to be addressed during the development process. This section consists of comprehensive guidelines for developing and securing APIs. It covers key areas like Identity and Access Management (IAM), API Endpoint Security, Error Handling, Session Management, Third-Party Integrations, and System Testing and QA. IAM protocols emphasize defining user roles, implementing multi-factor authentication, and securing access with Role-Based Access Control (RBAC). API endpoint security focuses on encryption, input validation, and rate limiting, while ensuring logging and monitoring to prevent attacks. Error handling recommends standardized procedures, logging, and preventing data leaks. Session management covers secure creation, storage, and monitoring of sessions, and third-party integrations stress risk mitigation, compliance, and continuous monitoring. Lastly, System Testing and QA highlight defining testing methodologies and automating tests to ensure consistency and security throughout development.

3 **Deploy:** recommendations for best practices related to deploying an API. This section covers key practices for deploying secure and efficient API systems, including asset management, versioning, and incident response planning. Asset management focuses on securely configuring and monitoring API systems to prevent data leaks, with SSL/TLS usage and regular audits ensuring system security. Implementing versioning helps track changes, maintain compatibility, and phase out outdated versions to reduce security risks. A strong incident response plan ensures resilience by setting clear recovery goals, automating notifications, and regularly testing processes to minimize downtime and data loss. Clear documentation and communication with stakeholders are essential for successful deployment and accountability.

4 **Operate:** recommendations for processes and practices during the operation of an API. This section highlights essential practices for effectively operating API systems, focusing

on monitoring, availability, compliance, and security. Monitoring and logging are crucial for detecting anomalies and malicious activities by tracking API requests, responses, and internal calls. Robust logging mechanisms and regular audits help prevent unauthorized access and ensure compliance. Maintaining system availability involves monitoring key performance metrics to proactively identify issues and enable scalability during high traffic periods, supported by automated scaling processes and load testing. Lastly, ensuring compliance and testing through systematic audits and regular Vulnerability Assessment and Penetration Testing (VAPT) strengthens security and regulatory adherence, fostering transparency and trust within the organization.

5 **Decommission:** recommendations for the secure removal of partial or full API functionality. This section outlines the essential steps for effectively decommissioning an API, focusing on establishing a decommissioning plan, communicating with stakeholders, and applying necessary actions. The plan emphasizes identifying items for phasing out, developing a secure data migration strategy, and creating a comprehensive timeline to enhance coordination and minimize miscommunication. Clear communication ensures that users, third-party companies, and developers receive timely updates about timelines, alternative solutions, and required actions, facilitating smoother transitions and preventing operational errors. Finally, executing critical actions involves carefully terminating services, retiring associated infrastructure, and reviewing network configurations to eliminate vulnerabilities.

This Guide covers security aspects in the key stages of API development, from design and development to deployment, operation, and eventually decommissioning. The overall vision is to support emerging API security standards and best practices with recommendations that can be used to set and audit configurations on a wide range of API platforms and technologies.

By publishing the CIS API Security Guide, CIS and FireTail hope to catalyze a vibrant community of subject matter experts (SMEs) interested in developing the platform-specific Benchmark guidance to come. We are calling on SMEs who develop or work with APIs to contribute to this project, for the benefit of the entire API security community. In addition, this Guide will be used to scope the creation of technology specific API security CIS Benchmarks.

Intended Audience

This CIS Guide is intended for API developers, architects, security specialists, auditors, and operations personnel who are responsible for designing, building, deploying, assessing, or securing API-based solutions.

Consensus Guidance

This guide was created using a consensus review process comprised of a global community of subject matter experts. The process combines real world experience with data-based information to create technology specific guidance to assist users to secure their environments. Consensus participants provide perspective from a diverse set of backgrounds including consulting, software development, audit and compliance, security research, operations, government, and legal.

Acknowledgments

This Guide exemplifies the great things a community of users, vendors, and subject matter experts can accomplish through consensus collaboration. The CIS community thanks the entire consensus team with special recognition to the following individuals who contributed greatly to the creation of this Guide:

Authors
Timo Rüppell, FireTail
Viktor Markopoulos, FireTail
Deborah Broomfield, FireTail

Editor
Phil White, Center for Internet Security

Contributor
Alana James-Aikins, CISSP
Rajashekar Yasani, Credit Acceptance Corp.
Valko Yotov

1 Design

This section outlines key considerations for designing secure and efficient APIs. It begins with API Documentation, which emphasizes the importance of clear, standardized documentation to ensure consistency and understanding among developers, consumers, and stakeholders. Next, Authentication and Authorization guidelines offer recommendations for implementing secure access control mechanisms, detailing various technologies used to protect API interactions. Lastly, Data Security and Privacy provides recommendations for conducting data assessments, enabling teams to identify the types of data handled by the API and implement appropriate protection measures to ensure privacy and security.

1.1 Documentation

This section consists of API documentation recommendations and the role it plays in API security. Establishing a standardized framework for documenting all aspects of the API ensures clarity and consistency across teams (developers, consumers, stakeholders).

1.1.1 Define a Format (Manual)

Description
A comprehensive standard framework for documenting all aspects of the APIs should be established.

Rationale
Clarity and consistency are offered by a standardized documentation framework for APIs, which serves as a single source of truth for all stakeholders involved. Accuracy is ensured, and collaboration within and between teams is promoted by having this resource.

Audit
- The documentation requirements should be reviewed.
- The comprehensiveness of the documentation should be assessed.
- All API changes should be monitored, and the documentation updated accordingly.

Remediation
- A gap analysis between the existing documentation and the desired standards or requirements should be completed. Areas that are incomplete, outdated, inaccurate, or lacking in coverage will be identified and highlighted for improvement.
- Feedback from impacted teams should be reviewed, and any necessary changes should be implemented.
- Versioning and document control should be used.
- Missing information should be incorporated into the documentation.
- An ongoing monitoring procedure should be implemented for the documentation, with mechanisms established to ensure continuous updates.

1.1.2 Specify Endpoints (Manual)

Description
Detailed definitions and descriptions of the API endpoints should be included in the documentation.

Rationale
Development efficiency and security are enhanced by clear endpoint specifications in API documentation, as they outline the API's capabilities, including endpoints, request formats, methods, and responses. This ensures that APIs are interacted with accurately, reducing errors and vulnerabilities. A concise summary of the endpoints is provided to aid end-users in understanding the API's functionality and locating relevant endpoints quickly. Use cases and examples are used to enhance comprehension and facilitate API integration into workflows

Audit
- The documentation requirements should be reviewed.
- A review of the specification files, including all existing endpoints utilized by the API, should be performed.
- Feedback from consumers, stakeholders, and developers should be gathered.

Remediation
- A gap analysis on the current state of the specification files and the desired standards or requirements should be conducted. Areas that are incomplete, outdated, inaccurate, or lacking in coverage will be identified and highlighted for improvement.
- Documentation requirements are to be updated, if needed.
- Missing endpoints from already-existing or updated requirements should be added.
- A continuous monitoring and evaluation procedure is to be implemented.

1.1.3 Specify Methods (Manual)

Description
The resources offered by the API and the methods used for each endpoint are to be defined in the documentation.

Rationale
Clarity for the development team is offered by specifying methods in API documentation, which details supported HTTP methods and helps developers understand the intended actions for each endpoint, such as retrieving data or modifying resources. This reduces trial and error, ensuring correct API implementation and minimizing security risks associated with inappropriate method usage. For end-users, a concise overview of supported methods is provided in the documentation, enabling them to quickly grasp available actions.

Audit
- The documentation requirements should be reviewed.
- The specification files are to be assessed to determine the methods used in the existing endpoints utilized by the API.
- It should be confirmed that methods follow a specific pattern based on their design type (e.g., CRUD for REST).
- Feedback from consumers, stakeholders, and developers should be gathered.

Remediation
- A gap analysis on the current state of the specification files and the desired standards or requirements should be performed to identify any discrepancies. Areas that are incomplete, outdated, inaccurate, or lacking in coverage will be pinpointed and highlighted for improvement.
- Feedback from all interested parties, including consumers, stakeholders, and developers, should be gathered to understand their needs and experiences.
- Documentation requirements are to be updated, if needed.
- Missing methods from already-existing or updated requirements should be added or updated.
- A continuous monitoring and evaluation procedure should be implemented.

1.1.4 Specify Data Models (Manual)

Description
The data models and objects that are used in the API are to be specified in the documentation.

Rationale
Clarity regarding the data structure and format for requests and responses is achieved for developers through the detailing of data models in API documentation, resulting in more efficient and secure development. Accurate request formatting and response handling are facilitated by clear data models, which minimize errors and security vulnerabilities, such as Insecure Direct Object Reference (IDOR). For end-users, understanding the types of data that can be provided or expected is improved by the specification of data models, thereby enhancing usability and adoption.

Audit
- The documentation requirements should be reviewed.
- The specification files are to be evaluated to ensure they include all models and variables utilized by the API.
- Feedback from consumers, stakeholders, and developers should be gathered.

Remediation
- A gap analysis on the current state of the specification files and the desired standard is to be performed to identify any discrepancies.
- Feedback from consumers, stakeholders, and developers should be gathered to understand their needs and perspectives.
- Documentation requirements are to be updated, if needed.
- Any missing data models or variables that are required based on existing or updated requirements should be added.
- A continuous monitoring and evaluation procedure is to be implemented.

1.1.5 Select an Automation-Friendly Format (Manual)

Description
API documentation should be in an automation-friendly format for easy use and integration with automation tools, supporting efficient management and automation of API interactions to enhance development processes and productivity.

Rationale
By adopting an automation-friendly format, updates can be automatically pulled directly from development sources like Git commits, ensuring that documentation stays current with minimal manual effort. This automation saves time and streamlines the documentation process, allowing developers to focus more on coding and less on manual documentation tasks.

Audit
- Ensure that APIs are being documented in a format that is structured and automation friendly such as OpenAPI, RAML, or Blueprint.
- Verify that API documents are up to date and reflect the operation and configuration of the API in production.
- Ensure that, where possible, automation tooling is in place to maintain the validity and accuracy of the API documentation.
- Ensure that, where possible, the documentation is being used to validate and enforce the operation and configuration of the API.

Remediation
- A suitable format for automation programs, including RAML (RESTful API Modeling Language), OpenAPI, and API Blueprint, should be identified.
- Any required manual changes are to be made first.
- Automation scripts are to be implemented within the development pipeline to automatically update the documentation.
- Syntactic validity checks are to be implemented to ensure that updates to the documentation adhere to specified standards and formatting requirements.

1.1.6 Specify Endpoint Parameters (Manual)

Description
Specifying endpoints in API documentation involves detailing the parameters utilized by each endpoint, including the type of data expected in requests, whether parameters are required or optional, and the content type used for parameter values.

Rationale
Clarity regarding API endpoints is provided to the development team through their specification in documentation, leading to more efficient and secure development practices. By specifying endpoints, developers are enabled to correctly understand how to interact with the API, identify potential security risks related to parameters, and implement appropriate validation to mitigate risks, such as injection attacks. This documentation ensures consistent usage of endpoints and reduces the likelihood of errors or vulnerabilities during the development and implementation of the API.

Audit
- Each endpoint is to be reviewed to identify the specified parameters used in API requests.
- It is to be determined whether parameters are required or optional for each endpoint, including special cases such as sorting parameters.
- The documentation requirements are to be reviewed.
- The specification files, including all required parameters, their data type, and their content type, are to be assessed.
- Feedback from interested parties should be gathered.

Remediation
- A gap analysis on the current state of the specification files and the desired standard is to be performed.
- Documentation requirements are to be updated, if needed.
- The data types expected for each parameter passed in requests are to be documented.
- The content type (e.g., application/json) for parameters used in API requests is to be specified.
- Missing parameters from already existing or updated requirements are to be added.
- A continuous monitoring and evaluation procedure is to be implemented.

1.1.7 Make Documentation the Single Source of Truth (Manual)

Description
The API documentation should be a single point of truth, providing a comprehensive reference document for all teams to consult.

Rationale
Multiple sources of documentation can lead to inconsistencies and miscommunication among teams, resulting in poor coordination. A unified document ensures that all teams have access to the same information, promoting alignment and clarity across the organization.

Audit
- Verification is required to ensure that only one document exists for each endpoint or location accessible to all teams.

Remediation
- A dedicated endpoint should be created to allow all teams access for uploading the documentation.

1.1.8 Establish Clear and Comprehensive Documentation Standards (Manual)

Description
The documentation should align with industry-standard frameworks that can be easily understood and followed by all teams. It must encompass all essential information needed by each team.

Rationale
Minimization of miscommunication is achieved when the documentation is comprehensible to all teams.

Audit
- Feedback from all teams should be gathered to determine if the current framework being used is understandable.

Remediation
- A format that is understood and familiar to all teams is to be identified.
- Any identified gaps or areas of confusion in the documentation, based on the feedback received, are to be addressed.
- The documentation is to be updated accordingly to ensure it remains comprehensive and clear.
- Any revisions or updates are to be communicated to the teams.
- Ongoing feedback and collaboration are to be encouraged to continuously improve the documentation and support team productivity.

1.1.9 Communicate Documentation Updates Across all Teams (Manual)

Description
Any changes to the documentation are to be communicated to all involved teams.

Rationale
All teams should be made aware of changes to ensure that they are working from the same version of the documentation.

Audit
- The effectiveness of the communication protocols for documentation changes is to be checked to ensure that all teams are notified properly.
- Feedback should be gathered from the teams to confirm that the documentation provides sufficient information for effective work.

Remediation
- An automated system should be set up to notify all involved teams whenever changes are made to the documentation.
- The content of notifications should be tailored to ensure proper context and sufficient information is communicated.

1.2 Authentication and Authorization

This section consists of recommendations on authentication and authorization mechanisms as well as defining the various technologies used.

1.2.1 Define Authentication and Authorization Requirements (Manual)

Description
The necessity for authentication and authorization mechanisms for the API is to be determined, along with the tools to be utilized. For RBAC, for example, controls are to be implemented with user roles defined as user, admin, and superadmin, etc., each possessing specific permissions and access levels to various data and functions within the API.

Rationale
A structured plan for defining API authentication and authorization requirements is provided from the outset, outlining how user identity and access will be managed. This approach aids in risk mitigation by implementing appropriate security controls and ensures compliance with regulatory laws and standards governing data protection and privacy, safeguarding against potential legal and compliance risks associated with API usage.

Audit
- The documentation and requirements provided by the development team and compliance standards are to be reviewed to ensure alignment and completeness.
- Compliance with requirements is to be evaluated using VAPT and/or automated solutions across various use cases, including normal scenarios and situations where low-privilege users attempt to access high-privilege data or data belonging to other users.
- The current state should be verified for compliance with the standards.
- To safeguard sensitive information, the proper and secure management of secrets and tokens is to be ensured through the use of solutions like encrypted vaults.

Remediation
- Current authentication and authorization mechanisms should be assessed.
- A gap analysis is to be conducted between the existing mechanisms and the compliance standards' mechanisms.
- Authentication methods are to be defined (for example, SSO, OAuth, API key, JWT).
- User and entity-specific authorization controls are to be established (for example, user, admin).
- Requirements should be documented.

1.2.2 Determine API Authorization Approach (Manual)

Description
A decision regarding whether the API will be private or public, internal or external, should be made.

Rationale
This decision will determine the type of authorization and visibility that the API will have.

Audit
- Documentation should be reviewed to ensure it includes the relevant authorization information for each API.

Remediation
- API user access requirements should be evaluated.
- API authorization requirements for storing user-sensitive data should be determined.
- All changes and decisions made should be documented.

1.2.3 Define User Roles (Manual)

Description
The specific user roles that should exist are to be determined, and the hierarchy of these roles is to be defined.

Rationale
Clearly defined user roles establish a structured plan for how users will interact with the API and outline the user base's organization.

Audit
- It should be confirmed that the documentation clearly states the roles assigned to users.

Remediation
- The intended users of the API should be identified.
- Roles should be selected based on the nature and utility of the API.
- The hierarchy and the roles should be documented.

1.2.4 Decide on role-based access controls (Manual)

Description
The access and authorization levels associated with each role are to be determined.

Rationale
Specific data access and functional permissions are to be assigned to each role to prevent disruptions, privilege escalation, and unauthorized access.

Audit
- It should be verified that the documentation specifies the access levels assigned to each role.

Remediation
- The hierarchy of the roles is to be established.
- The data and functions associated with each role are to be defined.
- The decisions made should be documented.

1.2.5 Choose Authentication Mechanisms (Manual)

Description
Appropriate authentication mechanisms are to be established to ensure secure access control and user authentication.

Rationale
Authentication mechanisms should be selected to ensure secure access control and protect sensitive data from unauthorized access or malicious activities. Strong authentication mechanisms are implemented to verify the identity of users or systems interacting with the API, thereby mitigating risks associated with unauthorized access, data breaches, and potential security vulnerabilities.

Audit
- The documentation on current authentication mechanisms should be consulted.

Remediation
- At least one authentication mechanism appropriate for API usage is to be selected.
- The decisions made are to be documented.

1.3 Data Security and Privacy

This section consists of recommendations to conduct data assessments for understanding the types of data utilized within the API, aiding in proactive planning for implementing appropriate data protection mechanisms.

1.3.1 Assess Data Protection Needs (Manual)

Description
Data protection needs for APIs are assessed by evaluating security requirements, identifying data types and vulnerabilities, and implementing measures such as encryption and access controls to safeguard sensitive information.

Rationale
The assessment of data protection needs for APIs is crucial for several reasons. Sensitive data is identified, and security precautions such as encryption are implemented to protect against unauthorized access. By understanding and addressing these data protection needs, risks of data leaks and unauthorized access can be mitigated, enhancing trust and reputation with users and stakeholders. Additionally, this process enables informed, risk-based decision-making by aligning security measures with identified risks and compliance requirements.

Audit
- A thorough assessment of the data being handled and stored by the API is to be conducted, focusing on its sensitivity.
- Encryption standards for data both in transit (using SSL/TLS) and at rest (with encryption and hashing algorithms) are to be ensured.
- A risk analysis on the data should be performed to identify potential threats and vulnerabilities associated with the data.
- The effectiveness of the data protection measures is to be validated.

Remediation
- A review of the data handled by the API is to be conducted.
- A thorough threat model should be established.
- Encryption mechanisms for data both at rest and in transit are to be implemented.

1.3.2 Determine Applicable Compliance Standards (Manual)

Description
All legal and regulatory standards that apply to the handling of specific data are to be identified. This requires an understanding of the rules and compliance standards that must be followed for the API to handle sensitive information.

Rationale
Ensuring legal obligations are met by adhering to regulations like GDPR or HIPAA, reduces legal risks related to data handling. Meeting compliance standards prioritizes customer data protection, outlining measures such as encryption and access controls to safeguard sensitive information. These standards contribute to effective risk management by implementing protections that mitigate potential data breaches or unauthorized access, enhancing overall data security.

Impact
Certain standards are required to mandate regular security-related activities, including penetration tests and compliance tests.

Audit
- The relevant regulations and compliance standards applicable to the API should be identified.
- The compliance standards are to be reviewed.
- An assessment of how data is handled, stored, and disposed of in accordance with the relevant regulations is to be conducted.

Remediation
- Data is to be classified, and the API's data types are to be identified.
- Research should be conducted on regulatory laws pertaining to the specific data.
- A gap analysis is to be performed to compare the current state of the API with the compliance standards.
- All steps taken are to be documented.
- Continuous monitoring and review processes are to be implemented.

1.3.3 Conduct Threat Modeling (Manual)

Description
Conduct threat modeling to identify and mitigate context specific risks

Rationale
Threat modeling enhances API security by identifying risks such as spoofing, tampering, information disclosure, and denial of service that may otherwise go unnoticed. Threat modeling should also account for application specific risks which may differ from case to case. For example, a messaging API, a data storage API, and a payment processing API can have different risks or different tolerances for similar risks. Defining and addressing these threats in the design phase reduces the likelihood of exploitation and leads to a more secure API architecture and easier alignment with regulatory standards for risk management.

Audit
- Verify that threat modeling documentation is up to date.
- Ensure that new features and releases have threat modeling conducted.
- Ensure all relevant assets and components (e.g., endpoints, data flows) are identified and evaluated for security risks.
- Ensure that a suitable threat model framework is applied consistently.
- Review the prioritization and mitigation strategy documented for each identified threat.

Remediation
- Choose a suitable structured threat modeling framework to maintain uniform threat identification.
- Conduct a threat modeling exercise for the API, documenting threats and their mitigation plans.
- Conduct a threat modeling session for each API feature, documenting threats and their mitigation plans.
- Prioritize high-risk threats and implement recommended controls in the design.
- Regularly review and update the threat model as part of the API lifecycle management.

2 Develop

This section consists of comprehensive guidelines for developing and securing APIs. It covers key areas like Identity and Access Management (IAM), API Endpoint Security, Error Handling, Session Management, Third-Party Integrations, and System Testing and QA. IAM protocols emphasize defining user roles, implementing multi-factor authentication, and securing access with Role-Based Access Control (RBAC). API endpoint security focuses on encryption, input validation, and rate limiting, while ensuring logging and monitoring to prevent attacks. Error handling recommends standardized procedures, logging, and preventing data leaks. Session management covers secure creation, storage, and monitoring of sessions, and third-party integrations stress risk mitigation, compliance, and continuous monitoring. Lastly, System Testing and QA highlight defining testing methodologies and automating tests to ensure consistency and security throughout development.

2.1 Identity and Access Management

This section consists of recommendations for implementing Identity and Access Management (IAM) protocols. It outlines the need to define user roles, access levels, and authentication methods, including multi-factor authentication (MFA) and secure transmission protocols. Authorization models such as Role-Based Access Control (RBAC) are recommended to enforce access restrictions, while cryptographic techniques should be applied to protect user credentials and tokens. Regular audits are advised to ensure compliance with regulations and to maintain security standards across the system.

2.1.1 Define User Identity Management Protocols within the System (Manual)

Description
Clear user identity management protocols should be established.

Rationale
User identity protocols are defined within the system to ensure proper management of authentication and authorization. Strong access controls are enforced to mitigate risks associated with unauthorized access. Efficient identity management protocols ensure operational efficiency, scalability, and adherence to security compliance.

Audit
- Authentication policies should be reviewed for accuracy and completeness.
- Access control policies must be carefully examined.
- The user registration and information management process should be evaluated.
- The systems used for storing user information need to be assessed.
- Logs should be reviewed.

Remediation
- Authentication mechanisms and management are to be enhanced.
- All changes made are to be documented.
- Communication regarding changes should be made with all teams.
- Affected systems are to be monitored.

2.1.2 Define User Types and Required Information (Manual)

Description
User types (e.g., admin, read-only user, superadmin) and their registration details (e.g., username, email address, password, MFA requirements) should be clearly defined for system setup.

Rationale
Defining user types and required information establishes clear authorization levels and ensures a structured hierarchy within the system. Assigning users specific roles such as administrators, regular users, or service accounts allows for appropriate permissions based on their responsibilities. Specifying the necessary user registration details ensures a standardized, secure process for account creation and management.

Audit
- Documentation regarding existing user types and required registration details should be reviewed.
- All systems and code should be reviewed to confirm that specific information is appropriately required, ensuring users cannot access accounts without meeting all necessary requirements.

Remediation
- Required changes should be implemented.
- Teams should be informed of the changes.
- Feedback should be gathered from relevant stakeholders.
- All changes should be documented properly.

2.1.3 Define Authentication Methods and Protocols (Manual)

Description
Authentication methods such as credentials, tokens, or third-party services, along with their associated protocols and processes, must be clearly defined.

Rationale
Implementing authentication methods like password logins and multi-factor authentication (MFA) enhances security by requiring users to authenticate their identity through multiple factors, reducing unauthorized access risks.

Audit
- The existing authentication policy and documentation should be reviewed.
- Compliance with regulatory requirements should be assessed.
- Current authentication methods in use should be evaluated.
- Current authentication processes (account recovery, token expiration times, throttling) should be evaluated.
- Protocols should be reviewed to ensure secure transmission.
- Encryption practices for data at rest should be evaluated.
- Findings and recommendations should be documented.

Remediation
- Update the authentication policy and documentation as needed.
- Ensure that regulatory compliance measures are being met.
- Strengthen authentication methods where necessary.
- Improve authentication processes where necessary.
- Implement secure transmission protocols.
- Apply encryption to data at rest.
- Ensure all updates are reflected in the documentation.

2.1.4 Define Access Levels for Each Role (Manual)

Description
Access levels and authorization requirements for each user type must be clearly defined to ensure proper system functionality and security.

Rationale
Clearly defining access levels and authorization requirements for different user roles is critical for maintaining security, ensuring regulatory compliance, and promoting operational efficiency. This helps prevent unauthorized access, supports adherence to legal requirements, and ensures that users can effectively perform their tasks with minimal risk.

Audit
- The existing access control policies should be reviewed.
- Compliance with regulatory requirements should be assessed.
- Permissions assigned to current user roles should be evaluated.

Remediation
- Roles are to be defined based on the responsibilities associated with each role.
- Permissions for user roles are to be configured according to the defined responsibilities.
- All changes made to access levels and role configurations are to be documented.

2.1.5 Define Authorization Models to Enforce Access Restrictions (Manual)

Description
Frameworks and systems that control and limit user access to resources based on their roles and permissions are to be defined. Examples include Role-Based Access Control (RBAC) and Attribute-Based Access Control (ABAC).

Rationale
Authorization models and access control lists are essential for maintaining security and compliance within a system. By clearly defining who can access sensitive information and perform specific actions, unauthorized access and potential breaches are prevented.

Audit
- Existing authorization policy documentation is to be reviewed.
- Compliance of current procedures with regulations is to be assessed.
- The implementation process is to be evaluated.
- Existing authorization models are to be examined.
- Permissions assigned to individual roles are to be reviewed.
- Findings and recommendations are to be documented.

Remediation
- Authorization policy documentation is to be reviewed.
- Regulatory compliance should be evaluated.
- The authorization model (RBAC or ABAC) to be implemented is to be assessed.
- Permissions for individual roles are to be implemented.
- All changes made should be documented.

2.1.6 Implement Cryptographic Techniques to Secure User Credentials and Authentication Tokens (Manual)

Description

Protocols to secure user data transmission and storage with cryptographic techniques should be specified.

Rationale

Cryptographic techniques are implemented to secure user credentials and authentication tokens, helping to mitigate risks and prevent unauthorized access. Passwords and tokens are encrypted to keep sensitive data secure and ensure compliance with regulations such as GDPR. This process renders data unreadable to unauthorized parties, protecting user credentials and authentication tokens from potential breaches and ensuring compliance with data protection laws and industry standards.

Audit

- Existing relevant policies should be reviewed.
- Compliance of existing procedures with regulations is to be assessed.
- Password hashing methods are to be evaluated.
- Token storage and generation methods should be assessed.
- TLS/SSL versions are to be reviewed.
- Key management practices and policies are to be evaluated.
- Monitoring practices should be assessed.

Remediation

- Password hashing methods are to be selected.
- Encrypted token storage and secure generation methods are to be chosen.
- TLS/SSL is to be implemented.
- Key management practices and policies are to be enhanced.
- Monitoring practices are to be applied.
- All changes made are to be documented.

2.2 API Endpoint Security

This section consists of recommendations for securing API endpoints by identifying specific security requirements, enforcing encryption protocols, and implementing input validation, rate limiting, and logging mechanisms. Each endpoint's security needs are addressed, including authentication methods, TLS/SSL enforcement, and rate-limiting measures to mitigate risks such as DDoS and brute-force attacks. Input validation and sanitization are emphasized to prevent vulnerabilities like SQL injection and cross-site scripting. Additionally, the section highlights the importance of monitoring, logging, and adhering to industry standards and regulations to ensure compliance and secure API operations.

2.2.1 Identify the Security Requirements for Each API Endpoint (Manual)

Description

Security requirements for each API endpoint are to be specified, including authentication methods (e.g., OAuth 2.0, API keys), encryption standards (e.g., TLS/SSL), authorization controls (e.g., role-based access), input validation, rate limiting, and logging to mitigate risks.

Rationale

The identification of security requirements for each API endpoint facilitates a risk assessment that evaluates potential threats and vulnerabilities specific to each endpoint. This process ensures compliance with industry and regulatory standards, preventing legal issues. Furthermore, it enables the implementation of tailored security controls that address the unique needs and risks associated with each endpoint.

Audit

- All existing documentation is to be reviewed.
- Compliance requirements are to be assessed to identify any existing gaps.
- A risk assessment and Vulnerability Assessment and Penetration Testing (VAPT) are to be performed.
- All requirements and procedures are to be documented.

Remediation

- Documentation is to be updated with all changes made.
- Security controls are to be implemented based on the endpoint requirements.
- Security requirements should be reviewed and updated as necessary.

2.2.2 Enforce TLS/SSL Protocols (Manual)

Description
The enforcement of TLS/SSL protocols is required to secure data transmissions.

Rationale
The enforcement of TLS/SSL protocols guarantees encryption of data during transfer, mitigating the risk of interception and eavesdropping by malicious actors. This measure prevents Man-in-the-Middle (MITM) attacks, maintaining confidentiality and integrity throughout transmission. Additionally, it secures the authentication process from unauthorized access and ensures compliance with security standards, promoting secure communication.

Audit
- The configuration settings should undergo a review.
- Certificates are to be examined for currency, correct configuration, and security to avoid vulnerabilities.
- HTTPS implementation and versions should be assessed.
- Non-compliant connections are to be reviewed.
- Policies and compliance requirements should be evaluated.

Remediation
- Configuration settings should be updated.
- Certificates are to be renewed as needed.
- HTTPS redirection from port 80 to port 443 (HTTPS) should be implemented, along with - - HSTS enforcement to mandate HTTPS.
- Vulnerabilities introduced by outdated HTTPS versions are to be addressed.
- Monitoring and alerting mechanisms should be enforced.
- Documentation is to be updated as required.

2.2.3 Enforce Request Validation, Sanitization and Authorization (Manual)

Description
Request validation, sanitization and authorization should be enforced.

Rationale
Request validation and sanitization should be enforced to shield against common attacks like SQL injection (SQLi) and cross-site scripting (XSS), which exploit vulnerable request parameters. Suspicious or malicious requests should be dropped or sanitized, ensuring data integrity and preventing unauthorized access to internal systems.

Request authorization should be enforced to protect against common attacks like enumeration or broken object level authorization, which exploit the manipulation of request parameters of authenticated requests. Request authorization should always ensure that access to a resource or function is intentional and in line with AIM policy.

Audit
- The codebase should be reviewed for existing request authorization practices.
- Request sources must be identified and validated.
- Validation rules are to be reviewed for completeness and security.
- Sanitization measures should be checked for effectiveness.
- Error handling procedures must be assessed.
- Logging and monitoring processes should be examined.

Remediation
- Request authorization practices should be enforced in line with IAM policies.
- Request validation rules should be updated where necessary.
- Sanitization procedures should be enforced.
- Request validation is to be implemented server-side.
- Error handling processes should be improved.
- Documentation of security controls must be completed and updated.

2.2.4 Enforce Response Validation, Sanitization and Authorization (Manual)

Description
Response validation, sanitization and authorization should be enforced.

Rationale
Response validation and sanitization should be enforced to protect against accidental data leakage. Response data models should be defined and responses should be validated and sanitized against those models, ensuring only the intended data is returned.

Response authorization should be enforced to protect against business logic flaws and adversarial data exfiltration. Response data should be checked to ensure that it conforms to the original request authorization and is in line with IAM policies.

Audit
- The codebase should be reviewed for existing response authorization practices.
- Response data sources must be identified and validated.
- Validation rules are to be reviewed for completeness and security.
- Sanitization measures should be checked for effectiveness.
- Error handling procedures must be assessed.
- Logging and monitoring processes should be examined.

Remediation
- Response authorization practices should be enforced in line with IAM policies.
- Response validation rules should be updated where necessary.
- Sanitization procedures should be enforced.
- Response validation is to be implemented server-side.
- Error handling processes should be improved.
- Documentation of security controls must be completed and updated.

2.2.5 Enforce Rate Limiting Mechanisms (Manual)

Description
Rate limiting mechanisms should be enforced to provide protection from repeated requests targeting an endpoint.

Rationale
Rate limiting mechanisms should be enforced to prevent Distributed Denial of Service (DDoS) and brute-force attacks by limiting the number of requests that can be made within a given timeframe. This ensures the availability of services for legitimate users and protects server resources from being overwhelmed. Compliance requirements often mandate the implementation of such measures to enhance security. By maintaining Quality of Service (QoS), a consistent and reliable experience for users should be ensured while mitigating the risk of abuse or exploitation of system resources.

Audit
- The application architecture should be reviewed.
- Existing rate-limiting configurations should be analyzed.
- Logs should be examined.
- Regulation compliance should be reviewed to ensure alignment with regulations.
- Denial of Service (DoS) resilience should be evaluated.
- Stress tests on existing rate limiting mechanisms should be performed.

Remediation
- Rate-limiting configurations should be updated.
- Behavioral rate limiting based on session and/or IP behavior should be implemented.
- Logging and monitoring procedures should be enforced.
- Distributed rate limiting (load balancers), CDNs, WAFs, and IDS/IPS should be deployed.
- All actions taken should be documented.

2.2.6 Define Logging and Monitoring Procedures (Manual)

Description
The locations and methods for logging should be specified, detailing what will be monitored and the procedures for monitoring them.

Rationale
Logging and monitoring practices should be defined to enhance system visibility, allowing for the identification of potential threats. Detailed logs should support future analysis, whether for incident response or service improvement. Comprehensive logging should ensure compliance with regulations and aid in effective risk management by identifying and mitigating risks promptly.

Audit
- Existing logging and monitoring policies should be reviewed.
- Current methods used should be assessed.
- Logging confidentiality and access management should be reviewed to determine who is allowed to view the logs.
- Log retention should be evaluated.

Remediation
- Policies and procedures should be updated.
- The appropriate logging detail required for endpoints should be used.
- Logging controls should be implemented to protect log files against third-party access.
- The logging procedure (collecting, analysis, retention, archiving) should be automated.
- Regular reviews and testing should be conducted.

2.2.7 Establish Practices Based on Regulations and Industry Standards (Manual)

Description
Guidelines and procedures should be put in place in accordance with regulatory requirements and industry standards to ensure compliance and best practices.

Rationale
Practices dictated by regulations and industry standards should be defined to ensure legal assurance, protect data and privacy, avoid financial losses from fines, and build trust with customers. Compliance with these practices should safeguard the organization against legal repercussions and enhance its reputation for reliability and integrity.

Audit
- All standards impacting specific APIs should be reviewed, considering different standards based on the nature of the data the API handles.
- A gap analysis should be performed to determine if existing procedures are in line with regulations.
- All existing documentation, policies, and procedures should be reviewed.
- Compliance assessments should be completed.
- Missing or faulty controls and mechanisms should be identified.
- External (compliance) assessments should be completed.
- Risk management should be conducted based on the above-mentioned assessments.

Remediation
- Regulatory standards affecting the API should be identified.
- Missing or faulty controls and mechanisms should be implemented.
- Changes should be assessed both internally and externally.
- All changes and updates should be documented.

2.3 Error Handling

This section consists of recommendations for proper error handling in APIs, focusing on identifying potential error occurrences and enforcing standardized error-handling procedures. It emphasizes the importance of preventing sensitive information disclosure through consistent error messaging, as well as enforcing error logging and monitoring across all endpoints for timely detection and analysis of issues. The guidelines also include establishing a robust error recovery strategy to minimize system downtime, preserve data integrity, and facilitate efficient troubleshooting. These practices ensure both security and operational continuity while complying with regulatory standards.

2.3.1 Identify Possible Error Occurrences (Manual)

Description
All endpoints where errors may occur (e.g., based on user input) should be identified.

Rationale
Possible error occurrences in APIs should be identified to prevent unpredictable system behavior and avoid risks such as inadvertently disclosing internal paths and technologies used. By proactively addressing these errors, security should be enhanced, and a more reliable user experience should be provided, ultimately contributing to a more robust and user-friendly application.

Audit
- The API design documentation should be reviewed.
- User input endpoints and data sources should be analyzed.
- Business logic should be analyzed.
- External dependencies and their integration with the API should be reviewed.
- Environmental factors like network latency and system load should be considered.

Remediation
- A standardized way to present errors should be established.
- Input validation and sanitization should be implemented.
- Errors at the code level should be caught.
- Monitoring (error logs) should be enforced.

2.3.2 Establish Standardized Error Handling Procedures (Manual)

Description
A consistent procedure to handle errors should be put in place.

Rationale
Standardized error-handling procedures should be established to ensure consistency across the API, providing a uniform approach to managing and communicating errors. This enhances the clarity and usefulness of error messages for developers and users, ultimately improving the overall user experience. Sensitive information that could assist attackers should be prevented from being revealed.

Audit
- The documentation should be reviewed by inspecting the existing error format in place.
- Potential disclosure of internal system information in error messages should be identified.
- The quality of all existing error messages should be assessed—whether they are comprehensive and correctly communicate the error.
- Compliance with HTTP standards should be verified, ensuring that, for example, a 404 error is used for non-existing resources, and 401/403 error codes are used for unauthorized access, and so on.

Remediation
- The clarity of error messages should be improved.
- Guidelines for handling errors should be established.
- Compliance with HTTP return status should be enforced.
- The documentation should be updated accordingly.

2.3.3 Enforce Error Logging (Manual)

Description
Error logging in API security should be enforced to systematically record and store encountered errors, allowing for future analysis.

Rationale
Enforcement of error logging in API security provides visibility into the system's response and state, aiding in the identification of errors or issues and helping to pinpoint potentially malicious activity. Comprehensive future analysis is enabled by these logs, contributing to improved Quality of Service (QoS) through proactive measures to address and prevent recurring issues.

Audit
- The existing logging policy should be reviewed.
- The implementation and methods used for logging should be reviewed.
- Storage methods and access level requirements should be reviewed.
- The log retention policy should be reviewed.
- The logging functionality should be tested.

Remediation
- Logging should be implemented with a robust method, ideally avoiding third-party services.
- Retention policies should be configured.
- The granularity of logging should be adjusted.
- Logs should be stored in a secure place.
- Access controls for stored logs should be implemented.

2.3.4 Enforce Error Monitoring (Manual)

Description
Logging should be made a mandatory practice for all endpoints.

Rationale
Enforcement of error monitoring in APIs is essential for ensuring timely detection of issues and threats, facilitating swift system recoverability, and complying with regulatory standards. Quick identification and remediation of issues are enabled by comprehensive error monitoring, promoting system resilience and integrity. Additionally, robust error monitoring practices are necessitated by adherence to regulation standards to mitigate legal risk.

Audit
- Logging configurations and implementation should be reviewed.
- Configuration compliance and coverage for all endpoints should be reviewed.

Remediation
- API and system logging should be enabled.
- A robust method for logging should be implemented.
- Configuration and logging settings should be enforced across all endpoints to achieve 100% coverage.
- Regular configuration reviews and updates should be conducted.

2.3.5 Establish a Strategy for Error Recovery (Manual)

Description
A standard procedure should be established to recover from errors and system failures.

Rationale
An error recovery strategy should be established to minimize downtime, preserve data integrity, pinpoint root causes, and mitigate business risks. Swift issue resolution is enabled, preventing disruptions to operations while safeguarding data from loss or corruption. Furthermore, identification of the underlying causes of errors is aided, facilitating more efficient troubleshooting.

Audit
- Existing procedure documentation should be reviewed.
- Error handling when errors occur should be assessed.
- The existence of a system recoverability procedure should be checked.
- Logs should be ensured to be kept for post-recovery analysis.

Remediation
- Error recovery procedures should be documented.
- Automated recovery mechanisms should be implemented.
- Operational mechanisms for involved teams should be established.
- Monitoring and alerting mechanisms should be established.
- Procedures should be regularly tested and validated.

2.4 Session Management

This section consists of recommendations for managing the session lifecycle in APIs, covering stages such as session creation, authentication, duration, and termination. It emphasizes defining session initiation processes, session duration and timeout settings, and establishing mechanisms for secure session maintenance. Proper session storage procedures are outlined to ensure the safe handling of authentication tokens, while session activity monitoring and logging are required to detect unauthorized access or unusual behavior. Regular auditing of session logs is advised to enhance security and support performance optimization through user behavior analysis. These practices help safeguard sensitive data and ensure system reliability.

2.4.1 Define the Session Lifecycle Stages (Manual)

Description
The various stages a user undergoes during a session should be identified, including creation, authentication, termination, and others.

Rationale
The session lifecycle stages should be defined to provide a clear plan for dividing a session into distinct phases, allowing for focused development of each stage. This clarity aids in troubleshooting by making it easier to identify and address issues within specific parts of the session lifecycle.

Audit
- Documentation regarding the path of a user from user creation to logging out should be reviewed.
- It should be ensured that the codebase handles each stage separately.
- The logic behind the code should be verified to follow a logical path; for example, active usage functions should be permitted only after authentication.

Remediation
- Every stage should be separated based on specific actions; for example, registration is designated as the creation stage, while logout is recognized as the termination stage.
- Each function should be implemented for every stage with separate logic.

2.4.2 Define How a Session is Initialized (Manual)

Description
The process for initiating an active user session should be established.

Rationale
The method for starting an active session should be established to ensure consistency in the activation process for all users, which is crucial for maintaining uniform security measures.

Audit
- Existing documentation surrounding user initialization should be reviewed.
- Implementation consistency should be assessed.
- Authentication mechanisms should be evaluated.
- Session creation should be validated by checking the response (e.g., whether a JWT is generated).

Remediation
- The logical beginning of a session (e.g., login) should be identified.
- A secure method for initializing a new session should be developed.
- A validation mechanism for the initialized session should be created.

2.4.3 Define Session Duration and Validity (Manual)

Description
A typical session duration should be established to ensure that sessions can be invalidated in a timely manner.

Rationale
Establishing a typical session duration is crucial for effective system resource management, as it allows for the efficient handling and invalidation of tokens, thereby freeing up storage. It also helps prevent exposures, such as leaked tokens, and reduces the risk of unauthorized access by ensuring that hijacked sessions are invalidated promptly.

Audit
- Session timeout duration should be evaluated.
- Renewal mechanisms should be assessed.

Remediation
- A session timeout setting should be decided based on security requirements (e.g., risk exposure, session sensitivity).
- A session timeout mechanism should be implemented.
- A session renewal mechanism should be established.
- The policy should be communicated to users.

2.4.4 Set Up Mechanisms to Maintain Sessions (Manual)

Description
A method to manage and sustain active user sessions securely and efficiently based on user interaction is to be established. This involves the implementation of token-based authentication (like JWTs), session expiry and renewal rules, secure token storage, and procedures for invalidating and logging out sessions.

Rationale
User experience is enhanced by ensuring uninterrupted access to resources. Session hijacking is also prevented through the implementation of robust authentication and authorization protocols, thereby safeguarding user accounts and sensitive data.

Audit
- Implementation consistency is to be assessed.
- The session update frequency is to be evaluated to ensure it is reasonable, as excessively long intervals can pose security risks.

Remediation
- Session maintenance practices are to be standardized.
- Secure session update mechanisms are to be implemented.
- Active sessions are to be monitored.

2.4.5 Define Session Storage Procedures (Manual)

Description
A method for securely storing session-related data, such as user authentication tokens or session identifiers, is to be established. Session-related data, including authentication tokens or session identifiers, is to be encrypted before storage to enhance security and protect sensitive information from unauthorized access or tampering.

Rationale
Session storage is established as vital for the secure and persistent storage of session data. Secure storage is essential for preventing leaks and unauthorized access to sensitive information, such as authentication tokens. Session persistence ensures seamless user interactions across requests. Additionally, scalable storage solutions, like distributed databases, enhance API scalability and performance.

Audit
- Storing implementation is to be assessed.
- The efficiency and suitability of storage technology are to be evaluated.
- The effectiveness of security measures, including encryption and access controls, is to be evaluated and validated.

Remediation
- A session storage implementation is to be chosen.
- Consistency across all components, services, and authentication methods of the API is to be ensured.
- Security controls around storage, including encryption and access controls, are to be implemented.
- Storage infrastructure is to be monitored.

2.4.6 Monitor and Log Session Activity (Manual)

Description
A procedure to monitor and log session activity is to be implemented.

Rationale
Monitoring and logging of session activity are essential for tracking session activity to detect abnormal behavior, which may indicate security threats such as unauthorized access or misuse. Logged activity provides valuable data for later use in analytics and incident response, assisting in identifying patterns, understanding user interactions, and improving security measures.

Audit
- The existing monitoring and logging policy is to be reviewed.
- The logging implementation procedure is to be evaluated.
- Granularity is to be evaluated to ensure it meets threat modeling and security requirements.
- The security controls in place on log storage are to be validated.

Remediation
- The logging and monitoring policy is to be created or updated.
- A logging infrastructure to manage and store data is to be implemented.
- Security controls around the infrastructure are to be enforced, including encryption and access control measures.

2.4.7 Audit Session Activity Logs (Manual)

Description
A policy or procedure for reviewing the logged data is to be established.

Rationale
A policy for reviewing logged data is vital for providing insights into API usage, aiding in the understanding of user behavior and system interactions. This information assists analytics aimed at enhancing performance. Regular log reviews help detect patterns of malicious activity, which can be utilized for future threat modeling.

Audit
- Existing logging configurations are to be reviewed, including the policy, granularity, and format.
- Logging retention and rotation settings are to be examined.
- Security controls in place are to be assessed.
- The existing inspection procedure is to be evaluated.

Remediation
- Logging configurations are to be improved, which may include updating the policy format and granularity.
- Retention and rotation settings are to be optimized to ensure efficient archiving, deletion, and replacement of log data.
- Security controls are to be updated.
- The inspection procedure is to be enhanced.

2.5 Third-Party Integrations

This section outlines best practices for managing third-party integrations in APIs, focusing on identifying dependencies, integration points, and understanding their data flow. It emphasizes the importance of assessing security and privacy risks associated with third-party components, as they may introduce vulnerabilities into the system. Compliance with regulatory requirements should be ensured by evaluating how third-party dependencies handle data. Additionally, third-party providers should be reviewed for security practices and reliability, while mitigation strategies should be developed to address identified risks and vulnerabilities. Continuous monitoring of these integrations is recommended to maintain security and regulatory compliance.

2.5.1 Identify Third-Party Dependencies (Manual)

Description

Third-party dependencies should be identified by determining which dependencies are being used, the developers behind them, and the code or framework on which they are built.

Rationale

The identification of third-party dependencies is crucial because they can pose a liability to system security. By assessing their security, the overall security integrity of the system is ensured, mitigating potential risks associated with these dependencies.

Audit

- The existing documentation should be reviewed.
- The source code should be reviewed to identify libraries, frameworks, and so on.
- Configuration lists and dependency management tools should be reviewed for identifying libraries and frameworks

Remediation

- An inventory of third-party software should be created.
- The inventory should be regularly reviewed and updated with dependencies found in management tools, source code, and configuration lists.
- A solution for automating the task should be implemented.

2.5.2 Identify Where Third-Party Integration Takes Place (Manual)

Description

The locations of third-party integrations should be identified in the code.

Rationale

The identification of third-party integration points is essential, as these points mark potential entryways for vulnerabilities into the system. Knowledge of these integration points allows for thorough security assessments to ensure the integration is implemented correctly and securely, which helps mitigate potential risks while ensuring optimal functionality. Additionally, the documentation of third-party integrations aids in maintaining an accurate inventory of system components and facilitates efficient troubleshooting and maintenance processes.

Audit

- The documentation should be reviewed to ensure it details the integration points.
- The source code should be reviewed to identify where third-party libraries are being imported.
- Network traffic analysis should be conducted to identify external connections.

Remediation

- All findings and any changes made should be documented.

2.5.3 Understand the Integration's Data Flow (Manual)

Description
Awareness of how data is transferred to and from third-party systems and how data is handled by third-party libraries through the code should be established.

Rationale
An understanding of the data flow of integration is crucial for identifying vulnerabilities within the system, enabling proactive measures to address them. Verification of adherence to best practices ensures secure and efficient data transfer processes. Understanding data flow is essential for compliance with regulations such as GDPR, particularly concerning data transfers involving US-based systems

Audit
- Data flows for each third-party library should be listed.
- Transmission protocols should be analyzed for encryption and security issues.
- Network traffic should be analyzed for security controls, transmission protocols, and regulatory compliance.
- Found practices should be reviewed for potential exposures and data leaks.

Remediation
- The flow of each third-party library should be documented.
- Regular reviews and audits should be conducted.

2.5.4 Identify Security and Privacy Risks of Used Dependencies (Manual)

Description
Known and past vulnerabilities and issues associated with any external dependencies should be identified.

Rationale
The identification of security and privacy risks of used dependencies is crucial for several reasons. Third-party software poses a liability as new vulnerabilities can be introduced into the system. An assessment of known vulnerabilities provides an overview of past and possibly present security posture, aiding in proactive risk mitigation and ensuring the integrity of the system.

Audit
- Documentation and the third-party dependency inventory should be reviewed.
- Documented vulnerabilities should be identified and cross-referenced with known databases.
- Automated or manual vulnerability scans and code reviews should be conducted to identify vulnerabilities.
- Dependency versions should be checked and cross-referenced with public databases (NVD) for known vulnerabilities (CVEs).

Remediation
- Findings should be documented.

2.5.5 Identify Security and Privacy Risks Within the Integration (Manual)

Description
Security and privacy risks within the integration should be identified by assessing whether the integration points are properly implemented. Potential vulnerabilities and data protection issues that could arise from integrating third-party services or components should be evaluated.

Rationale
The identification of security and privacy risks within integrations is vital because integrations serve as crucial parts of systems. Incorrectly integrated dependencies or misconfigurations can introduce data leaks.

Audit
- The existing documentation should be reviewed.
- Integration points and their respective findings should be identified.
- It should be checked whether findings have been remediated.

Remediation
- Integration points from documentation should be reviewed.
- Static code analysis and source code reviews should be conducted to identify misconfigurations.
- This process should be integrated into the development lifecycle.
- Findings and remediation should be documented.

2.5.6 Identify Compliance and Regulatory Risks of the Third-Party Dependencies (Manual)

Description
Compliance and regulatory risks of the third-party dependencies should be identified, as regulatory risks can arise depending on how data is handled by these dependencies.

Rationale
The identification of compliance and regulatory risks associated with third-party dependencies provides an understanding of how data is handled by these dependencies, ensuring alignment with data protection regulations and standards. This process offers a legal overview, helping organizations avoid penalties and legal repercussions resulting from non-compliance with regulatory requirements.

Audit
- All regulations relevant to each API should be identified.
- The compliance of the third-party dependencies used should be assessed.
- The impact in the case of non-compliance should be evaluated.
- Documentation for third-party dependency lists should be reviewed.

Remediation
- Documentation for third-party dependency lists should be reviewed.
- Compliance of third-party dependencies with relevant regulations should be ensured.
- Contractual agreements with vendors should be established to ensure compliance obligations and responsibilities.
- Network and data flow should be monitored to verify ongoing compliance.

2.5.7 Review Third-Party Providers (Manual)

Description
Due diligence should be performed by assessing the security practices, reliability, and reputation of third-party providers.

Rationale
The review of third-party providers helps assess their reliability, trustworthiness, and ability to meet security and performance requirements, ensuring alignment with the organization's standards.

Audit
- Security assessments should be conducted before engaging with third-party providers.
- Providers should be evaluated through customer reviews, industry reports, and other relevant sources.
- Contractual agreements and SLAs should be reviewed.
- The provider's security posture should be assessed through vulnerability assessments conducted by internal teams or third parties.

Remediation
- Clear communication channels and procedures should be established to address issues or concerns.
- Dependencies should be monitored for regulatory compliance and non-compliance.

2.5.8 Develop Mitigation Strategies for Identified Risks and Vulnerabilities (Manual)

Description
Mitigation strategies should be developed for identified risks and vulnerabilities.

Rationale
The development of mitigation strategies for identified risks and vulnerabilities enables the patching of vulnerabilities, reducing the risk of exploitation and potential data leaks. Additionally, this process aids in avoiding non-compliance with regulations.

Audit
- Risks and vulnerabilities discovered during the audit of third-party integrations, dependencies, and providers should be identified.
- The potential impact and likelihood of exploitation for each identified vulnerability should be evaluated.
- Mitigation strategies should be developed, which may include risk acceptance, risk avoidance, risk transfer, or risk mitigation measures based on the severity and criticality of identified risks.

Remediation
- Remediation measures should be implemented to address identified risks and vulnerabilities, including the application of patches, updates, and the implementation of new security controls.
- Incident response procedures should be established to mitigate the impact of known or unknown security incidents.
- Mitigation strategies should be monitored continuously.

2.6 System Testing and QA

This section outlines the key steps for implementing a robust system testing and quality assurance (QA) process. It emphasizes defining the scope, objectives, and methodologies for testing to ensure focused and efficient efforts, aligned with API functions and threat models. Establishing a standardized system testing lifecycle is crucial for maintaining consistency and reliability across tests. The setup of automated testing upon code push is also highlighted, promoting early detection of issues and smoother integration into the codebase. Continuous monitoring, documentation, and stakeholder communication are essential for improving testing processes.

2.6.1 Define the Scope and the Objectives of the Testing (Manual)

Description
The scope and objectives of testing are to be defined to ensure targeted and purposeful testing efforts.

Rationale
The scope and objectives of testing are essential for achieving goals by ensuring focused and efficient testing that aligns with specific API functions and objectives.

Audit
- The scope of the test is to be reviewed.
- Alignment of the scope with the goals of the test is to be ensured.

Remediation
- The scope of testing should be clarified and documented.
- Feedback on the scope is to be obtained from stakeholders.

2.6.2 Define the Methodologies Used in Testing (Manual)

Description
The testing approach (black, gray, or white box) should be defined based on the threat model of the API.

Rationale
The definition of the methodologies used in testing is essential, as each methodology provides a different approach and perspective for evaluating the API. Specific kinds of tests, such as gray or white box testing for insider threats, are required by different threat models, ensuring thorough coverage and effective mitigation of potential risks.

Audit
- The scope of work and rules of engagement documents should be reviewed.
- Alignment with stakeholders regarding decisions made should be ensured.
- The testing approach should be verified to align with the threat model.

Remediation
- The threat model of the API should be reviewed.
- Approaches should be discussed with stakeholders.
- Decisions should be documented and regularly reviewed.

2.6.3 Define the Process of the System Testing Lifecycle (Manual)

Description
A standardized testing process should be established to be consistently used for every test conducted.

Rationale
The definition of the process of the system testing lifecycle is important, as it helps maintain organization by outlining the steps beforehand, making it easier to identify issues and troubleshoot problems when they arise. Additionally, structured, systematic, and repeatable testing activities should be ensured from development to deployment, promoting consistency and reliability in the testing process.

Audit
- The documentation should be reviewed.
- The clarity of the lifecycle stages should be assessed.
- Alignment with best practices should be verified.
- Established tests should be verified to undergo the entire lifecycle.

Remediation
- The process should be communicated to all stakeholders.
- An automation process for the lifecycle should be created (moving from development to test environment, pulling test data, etc.).
- Mechanisms for continuous lifecycle enhancement should be established.
- Actions should be documented.

2.6.4 Set Up Automated Testing Upon Code Push (Manual)

Description
An automated testing process should be established to take place upon code push.

Rationale
The setup of automated testing upon code push is essential, as immediate feedback on security, integration, and quality aspects is provided, ensuring the early detection of potential issues. Additionally, smooth integration into the existing codebase should be allowed once tests have passed, streamlining the development process and promoting efficiency.

Audit
- Any existing documentation should be reviewed.
- Coverage should be verified, and it should be ensured that tests are running smoothly without errors.
- Results of the testing should be reviewed.

Remediation
- Automated testing frameworks and scripts should be implemented.
- Tests should be integrated into the CI/CD pipeline.
- Effectiveness should be verified, and processes should be monitored.
- All actions taken should be documented.

3 Deploy

This section covers key practices for deploying secure and efficient API systems, including asset management, versioning, and incident response planning. Asset management focuses on securely configuring and monitoring API systems to prevent data leaks, with SSL/TLS usage and regular audits ensuring system security. Implementing versioning helps track changes, maintain compatibility, and phase out outdated versions to reduce security risks. A strong incident response plan ensures resilience by setting clear recovery goals, automating notifications, and regularly testing processes to minimize downtime and data loss. Clear documentation and communication with stakeholders are essential for successful deployment and accountability.

3.1 Asset Management

This section details best practices for effective asset management and secure configuration within API systems. It emphasizes the importance of proper management and secure configuration to prevent information leaks and protect sensitive data. The implementation of an effective release process is highlighted to ensure consistency and predictability, clarifying roles and responsibilities among teams. Additionally, the necessity of using SSL/TLS for all API endpoints is stressed to safeguard data transmissions against interception and ensure regulatory compliance. Monitoring and logging are crucial for incident response, providing essential insights into security incidents. All processes and configurations should be regularly reviewed and documented to maintain a secure and organized system.

3.1.1 Manage Assets and Configuration Data (Manual)

Description
Proper management and secure configuration of asset management and inventory should be ensured.

Rationale
The security of configuration and asset management is important, as it helps prevent leaks that could aid in malicious reconnaissance, protecting sensitive information from potential attackers. Proper management also leads to a more organized and structured system, facilitating easier maintenance and enhanced security.

Audit
- Configuration settings for API hosts, databases, and network devices should be reviewed.
- Storage and access to credentials needed at run-time should be reviewed.
- Management practices should be evaluated.
- The validity of the asset inventory should be verified.
- Access controls and permissions should be checked.

Remediation
- A management practice and configuration for API hosts, databases, and network devices should be implemented.
- A secrets manager or similar system should be implemented to control access to credentials needed at run-time.
- An asset inventory that provides a single point of truth about the organization's APIs, hosts, and third-party services should be maintained.
- Access controls and permissions should be enforced to prevent leaks, adhering to the principle of least privilege.
- Settings and inventory should be periodically reviewed and updated.

3.1.2 Use Separated Environments (Manual)

Description
The use of separate, isolated environments for production loads vs. development work should be ensured.

Rationale
Separating development work from production services and data minimizes security risks from unauthorized access or accidental misconfiguration which can lead to data exposure or service interruptions. Separated and isolated environments provide a structured, secure setup where changes can be tested without impacting live services and compliance requirements for data handling and privacy can be met.

Audit
- Verify the setup of separate environments for development, staging, production, etc.
- Verify environment isolation for data and external integrations.
- Review access controls, ensuring appropriate restrictions for each environment.
- Ensure that environment-specific configurations are properly managed.
- Confirm that production data is only used in the production environment.

Remediation
- Establish separate environments for testing, staging, production, etc. according to the development workflow's needs.
- Ensure that data and external integrations are isolated to each environment separately.
- Implement appropriate access controls for each environment.
- Set up a system for managing environment specific configuration data.
- Ensure that production data is only used in the production environment.

3.1.3 Implement an Effective Release Process (Manual)

Description
The release of the API and any new versions should be automated and follow a stable process.

Rationale
An effective release process ensures control and stability, promoting consistency and predictability. It also clarifies roles and responsibilities among the involved teams, enhancing coordination and efficiency.

Audit
- The existing release process documentation should be reviewed.
- The efficiency of the release process should be assessed, including actions taken from development to release.
- Roles and responsibilities of each team should be checked.
- It should be ensured that the process includes all essential steps (testing, approval, and deployment stages).

Remediation
- Documentation standardizing the release process should be created.
- Roles should be defined and shared among the involved teams.
- Feedback mechanisms should be implemented.
- All actions should be documented.

3.1.4 Ensure the Use of SSL/TLS (Manual)

Description
SSL/TLS usage should be verified for all API endpoints and versions.

Rationale
Ensuring the use of SSL/TLS is crucial as it encrypts data transfer and communication, protecting sensitive information from interception. This mitigates the risk of Man-in-the-Middle (MITM) attacks, enhancing security. Furthermore, SSL/TLS usage helps ensure compliance with regulations requiring secure data transmission.

Audit
- SSL/TLS enforcement for all endpoints should be reviewed.
- SSL/TLS versions should be evaluated.
- Certification expiration dates should be reviewed, and any potential compromises should be assessed.

Remediation
- SSL/TLS encryption for all API communications should be enforced.
- Configurations and certificates should be regularly reviewed for expiration dates, and potential compromises should be assessed.
- Versions should be updated when necessary.
- Configurations should be monitored.

3.1.5 Monitor and Keep Logs (Manual)

Description
Assets should be monitored, and logs should be securely stored.

Rationale
Monitoring and keeping logs are important for incident response, as they help identify what happened during a security event. Logs provide valuable insights to prevent similar issues in the future.

Audit
- The existing monitoring settings should be reviewed.
- It should be ensured that all assets are monitored.
- Logging processes should be verified, and it should be ensured that logs are stored securely.
- Authorization access to the log storage should be confirmed.

Remediation
- Any assets requiring monitoring should be identified.
- Monitoring for all identified assets should be enabled with appropriate granularity.
- Logs should be securely stored.
- Access to log storage environments should be restricted to authorized personnel only.
- All actions taken should be documented.

3.2 Versioning

This section outlines best practices for implementing versioning in API development to enhance stability and usability. It emphasizes the importance of structured versioning during new feature releases, allowing for easier tracking of changes and minimizing compatibility issues. Ensuring backward compatibility is crucial for seamless integration of new versions with existing systems, preserving API availability. The process of phasing out old versions is recommended to mitigate security risks and compatibility challenges associated with outdated software. Additionally, effective communication with stakeholders regarding changes, new capabilities, and timelines for support is essential to facilitate successful transitions. Finally, all processes related to versioning should be well-documented for clarity and accountability.

3.2.1 Use Versioning and Lifecycle Management (Manual)

Description
Versioning should be used where necessary when releasing new features.

Rationale
Versioning when releasing new features provides a structured and stable way of releasing code updates. Additionally, it allows for easier tracking and monitoring of changes over time, facilitating better management of the development lifecycle. Versioning reduces compatibility issues, users and developers can clearly see which version of the software they are using, reducing the likelihood of compatibility issues between different versions or updates.

Audit
- The version control documentation should be reviewed.
- The existence of a versioning scheme and its application should be verified.
- Backward compatibility should be checked.
- The process for removing old versions should be verified.

Remediation
- A versioning scheme (e.g., v1, v2) should be implemented.
- Backward compatibility should be ensured.
- Outdated versions should be removed.
- A process for releasing code under a specific version should be documented.
- Both new and old versions should be tracked, monitored, and logged.

3.2.2 Ensure Backward Compatibility (Manual)

Description
Ensure new versions are seamlessly integrated with the workflow of the previous version.

Rationale
Ensuring backwards compatibility is important because new versions may not always be released in large updates. By ensuring that a new version of an API function is backward compatible, it can integrate with existing systems and applications when it's ready, maintaining the availability and usability of the API.

Audit
- The process for integrating new versions should be reviewed to ensure flawless operation.
- Integration with existing systems should be verified.

Remediation
- QA and VAPT tests should be conducted on new versions.
- Integration should be verified to ensure it functions without disruptions or errors and remains secure against vulnerabilities.
- The process should be documented.

3.2.3 Remove Old Versions (Manual)

Description
Old versions should be removed after a period, once the majority of users have transitioned to the new version.

Rationale
Phasing out old versions is important because they may no longer be maintained, leaving vulnerabilities unpatched and creating unnecessary attack surfaces. Additionally, outdated versions can lead to compatibility issues and pose risks to system security.

Audit
- The process for removing old API versions should be reviewed.
- The automation process, if available, should be assessed.
- It should be verified that no traces, such as forgotten endpoints, remain after removal.
- The removal of integrations from old versions that connect to existing databases and other systems should be ensured.

Remediation
- An automation process should be implemented to remove old versions from public and private access.
- Integrations from old versions connecting to existing databases and systems should be removed.
- The process should be documented.

3.2.4 Notify Stakeholders (Manual)

Description
All impacted parties should be notified of changes.

Rationale
Stakeholders must be informed about new versions, including their capabilities and endpoints. Clear communication regarding the support timeline for older versions ensures users can plan transitions or updates accordingly.

Audit
- The procedure for removing or adding API versions should be reviewed.
- It should be confirmed that changes are communicated to users and stakeholders.
- The clarity and comprehensibility of the communication should be ensured.

Remediation
- A process for notifying stakeholders about API changes through appropriate channels should be implemented.
- Clear and comprehensible communication should be ensured.
- The changes and timeline for each step should be communicated.
- All changes should be documented.

3.3 Incident Response Plan

This section outlines practices for developing a robust incident response plan focused on resilience and effective communication during downtimes. It emphasizes defining achievable Recovery Time Objectives (RTO) and Recovery Point Objectives (RPO) to minimize downtime and data loss, supporting business continuity. Regular evaluations of resilience measures against these objectives enhance recovery mechanisms. Additionally, an automated notification process is essential for alerting relevant teams promptly during incidents. Testing these processes for effectiveness and documenting actions taken is crucial for continuous improvement and operational readiness.

3.3.1 Ensure Resilience in Line with RTO/RPO (Manual)

Description

Recovery Time Objective (RTO) and Recovery Point Objective (RPO) goals should be ensured to be realistically achievable through established mechanisms and procedures.

Rationale

Resilience in line with RTO and RPO should be ensured to minimize downtime and data loss, supporting business continuity and operational efficiency. Achieving these goals enhances recovery mechanisms, plans, and incident response actions.

Audit

- Resilience and disaster recovery documentation and procedures should be reviewed.
- It should be evaluated whether RTO and RPO have been defined.
- The alignment of resilience measures, such as recovery mechanisms, with RTO and RPO goals should be assessed.
- The effectiveness of these measures during disasters or downtime should be evaluated.

Remediation

- RTO and RPO should be defined and documented.
- Resilience measures, including recovery mechanisms, should be implemented to meet RTO and RPO goals.
- Regular resilience testing should be conducted to verify effectiveness.
- Resilience plans and procedures should be updated based on lessons learned from testing and real-world incidents.

3.3.2 Create an Automated Notification Process (Manual)

Description

An automated process should be implemented to notify the relevant team when the system experiences downtime.

Rationale

The creation of an automated notification process is essential for reducing Recovery Time Objective (RTO), enabling a quicker response and system restoration in the event of an issue. This helps ensure availability and enhances the user experience.

Audit

- Incident response policies should be reviewed.
- Verification that a process for managing system downtime is in place should be conducted.
- The automation of the process should be checked, ensuring that teams are notified through secure channels.

Remediation

- A process for managing system downtime should be created, identifying the necessary teams and automating notifications through secure channels.
- The process should be tested for effectiveness.
- All actions taken should be documented.

4 Operate

This section highlights essential practices for effectively operating API systems, focusing on monitoring, availability, compliance, and security. Monitoring and logging are crucial for detecting anomalies and malicious activities by tracking API requests, responses, and internal calls. Robust logging mechanisms and regular audits help prevent unauthorized access and ensure compliance. Maintaining system availability involves monitoring key performance metrics to proactively identify issues and enable scalability during high traffic periods, supported by automated scaling processes and load testing. Lastly, ensuring compliance and testing through systematic audits and regular Vulnerability Assessment and Penetration Testing (VAPT) strengthens security and regulatory adherence, fostering transparency and trust within the organization.

4.1 Monitoring and Logging

This section outlines the critical practices for enabling comprehensive monitoring and logging of API activities. It emphasizes the need to monitor API requests, responses, and internal API calls to detect anomalies and malicious activities, thereby enhancing security. Real-time monitoring aids in identifying potential vulnerabilities, coding errors, and insider threats, providing valuable insights for performance optimization. The establishment of robust logging mechanisms, along with the secure storage of logs and redaction of sensitive information, is essential to prevent unauthorized access. Regular audits of logging configurations and adherence to retention policies are crucial for maintaining operational integrity and compliance.

4.1.1 Monitor API Requests (Manual)

Description
API requests should be monitored to observe real-time traffic.

Rationale
Monitoring API requests in real-time is crucial for detecting anomalies and malicious activities through both static and dynamic analysis. The identification of potential attacks and unknown vulnerabilities (zero-day exploits) is enabled, enhancing security. Additionally, the storage and analysis of API requests provide valuable insights into usage trends and performance, assisting in the optimization and security of the API infrastructure.

Audit
- The logging and monitoring configuration should be reviewed.
- Verification should be conducted to ensure that the information stored is relevant and detailed.
- The current log retention policy should be assessed.
- Logs should be ensured to be stored securely, with proper security controls implemented to prevent unauthorized access and leaks.

Remediation
- A logging and monitoring mechanism should be set up based on the network load and available hardware (servers, load balancers, etc.).
- Log aggregation mechanisms should be implemented, and the storage location should be decided.
- The configuration should be reviewed regularly.
- All actions taken should be documented.

4.1.2 Monitor API Responses (Manual)

Description
API responses should be monitored to observe real-time traffic.

Rationale
Monitoring API responses is essential for detecting anomalies and vulnerabilities, such as instances where responses mistakenly include data from multiple users, indicating potential security breaches or coding errors. The storage and analysis of this traffic provide valuable insights into potential issues and usage patterns.

Audit
- The logging and monitoring configurations should be reviewed.
- Verification should be conducted to ensure that the information stored is relevant and detailed, with checks in place to confirm that any sensitive information (PII, JWTs) is redacted.
- The log retention policy should be assessed.
- Logs should be ensured to be stored securely, with proper security controls implemented to prevent unauthorized access and leaks.

Remediation
- A logging and monitoring mechanism should be set up based on the network load and available hardware (servers, load balancers, etc.).
- A procedure should be implemented to redact sensitive information such as PII from the logs.
- Log aggregation mechanisms should be implemented, and the storage location should be decided.
- The configuration should be reviewed regularly.
- All actions taken should be documented.

4.1.3 Monitor Internal API Calls (Manual)

Description
Calls from internal APIs and from load balancers/proxies to internal systems should be monitored.

Rationale
Monitoring internal API calls helps in detecting malicious insiders and Advanced Persistent Threats (APTs) operating within the network. This monitoring also facilitates the identification of second-order attacks, such as HTTP smuggling, which can exploit vulnerabilities in the interactions between different systems and services.

Audit
- The current logging and monitoring configuration should be reviewed.
- Verification should be conducted to ensure that internal APIs and systems are included in the monitoring process.
- The log retention policy should be assessed.
- Logs should be ensured to be stored securely, with proper security controls implemented to prevent unauthorized access and leaks.

Remediation
- Internal APIs and their exit/entry points to the Internet (proxy, load balancer) should be identified.
- A logging and monitoring mechanism should be set up based on the network load and available hardware (servers, load balancers, etc.).
- Log aggregation mechanisms should be implemented, and the storage location should be decided.
- The configuration should be reviewed regularly.
- All actions taken should be documented.

4.1.4 Set up Alerting and Notification Channels (Manual)

Description
Alerts from Monitoring and Logging should go out to the appropriate stakeholders

Rationale
Alerting the correct stakeholders ensures that timely action can be taken to remediate the underlying issue that caused the alert. Enabling quick remediation can protect an API form downtime and speed up recovery. Well configured alerts have a low rate of false positives and are limited to those stakeholders responsible for the continued operation of the API.

Audit
- The configured alerts and corresponding notification channels should be reviewed.
- The responsible stakeholders for each alert and notification channel should be reviewed.
- The notification contents should be reviewed to contain sufficient actionable information.
- The rate of false positives should be reviewed.
- Any incidents that were missed by the alerting and notification system should be reviewed.

Remediation
- Alerting and notification channels should be established and adjusted to ensure that the correct stakeholders receive alerts.
- The contents of the notifications should be adjusted to enable and help with remediation.
- The alerts should be adjusted to minimize false positives and to cover any gaps in visibility that have been found.

4.2 System Availability

This section emphasizes the importance of maintaining system availability and performance through comprehensive monitoring and scalability measures. Key metrics, such as CPU usage, RAM, and network latency, should be tracked to identify potential issues and optimize system health proactively. Regular audits of monitoring tools and performance thresholds are essential to ensure timely detection of downtime or performance degradation. Additionally, ensuring the API's ability to scale with user traffic demands is crucial for maintaining seamless operation during high usage periods. Implementing automated scaling processes and conducting load and stress testing helps ensure reliable performance, customer satisfaction, and long-term operational success.

4.2.1 Monitor Availability and Performance (Manual)

Description
Availability and performance should be monitored by tracking system metrics such as CPU usage, RAM, network latency, and workload, while system uptime is monitored to detect and address downtime issues.

Rationale
Monitoring availability and performance is crucial as it provides an overview of the system workload and performance, offering insights into areas for enhancement and identifying issues that require fixing or may result in downtime. This ensures proactive management of system health and efficiency.

Audit
- The existing monitoring tools and systems should be evaluated.
- Performance thresholds and alerting mechanisms should be reviewed for timely detection.
- It should be validated that monitoring covers key aspects (resource utilization, network latency).
- The frequency and granularity of monitoring checks should be assessed.

Remediation
- Monitoring mechanisms should be configured to cover the system aspects of the API systems.
- Performance baselines should be developed to serve as a "bare minimum".
- Alerting mechanisms should be configured.
- Regular reviews of monitoring mechanisms should be conducted.
- All actions taken should be documented.

4.2.2 Ensure Scalability Under Load (Manual)

Description
The ability of the API and its components to scale as user traffic demands should be ensured.

Rationale
Ensuring scalability under load is crucial for maintaining API availability and performance, even during high usage spikes. This prevents performance issues and ensures a smooth user experience, maintaining business continuity. By avoiding downtimes and lag, customer satisfaction remains high, which is essential for the business's reputation and reliability. Scalability supports the seamless handling of increased demand, securing long-term operational stability and success.

Audit
- The existing scalability testing procedures should be evaluated.
- Infrastructure and resource allocation when scaling should be reviewed.
- It should be ensured that scaling is an automated process.

Remediation
- Load and stress testing should be performed to assess the API's traffic-handling capacity.
- Infrastructure should be modified, and resources allocated accordingly to accommodate scaling requirements.
- Scaling processes should be automated to streamline efficiency and responsiveness.
- The scaling process should be continuously monitored for ongoing optimization.

4.3 Compliance and Testing

This section focuses on ensuring compliance and enhancing security through systematic auditing and testing measures. Regular audits are vital for verifying the adequacy of the security posture and compliance with regulatory standards, enabling timely detection of incidents and mitigating risks. Aligning data retention practices with compliance standards is crucial for legal adherence and enhancing transparency in data management. Additionally, conducting regular Vulnerability Assessment and Penetration Testing (VAPT) helps identify and remediate potential threats before they can be exploited. By implementing automated audit reports and ensuring comprehensive coverage in testing, organizations can maintain a proactive approach to security and compliance, ultimately fostering trust and accountability in their operations.

4.3.1 Audit Requirements and Reporting (Manual)

Description
Audits are essential to ensure that the security posture and compliance measures are adequate.

Rationale
Considering audits is essential because it demonstrates adherence to regulatory requirements, industry standards, and internal policies. It also enables timely detection and response to security incidents, breaches, and compliance violations, helping to mitigate risks and prevent future incidents.

Audit
- The audit logs and reports should be viewed to verify compliance.
- Audit events should be validated for accurate logging, including user actions, system events, and security incidents.
- The completeness and accuracy of audit trails should be assessed.

Remediation
- Audit logging mechanisms should be enhanced for complete coverage of the system.
- The generation of audit reports should be automated.
- Regular reviews of audit logs should be conducted to identify anomalies and trends and for optimization.

4.3.2 Align Data Retention with Compliance Standards (Manual)

Description
Data retention and storage must comply with regulations and industry standards for best performance and security.

Rationale
Aligning data retention with compliance standards is crucial as it ensures that data is retained according to regulatory requirements, thus avoiding legal risks. Security and privacy standards are maintained while enhancing transparency, accountability, and trustworthiness in data handling practices.

Audit
- Data retention policies and procedures should be reviewed to ensure compliance.
- Data retention periods should be verified and defined based on legal requirements.
- Data storage and practices should be assessed to ensure secure data retention.
- A deletion process should be validated for data that exceeds retention periods.

Remediation
- Data retention policies and procedures should be revised to align with current regulatory requirements and industry best practices.
- Data that requires retention should be classified.
- An automated process for data retention should be implemented.
- Periodic audits should be conducted for continuous compliance and optimization.

4.3.3 Schedule Regular VAPT (Manual)

Description
Regular Vulnerability Assessment and Penetration Testing (VAPT) is conducted to enhance security by identifying and addressing potential vulnerabilities and threats.

Rationale
Regular VAPT is crucial as it mitigates risks by catching vulnerabilities early, assesses an attacker's ability to enumerate the API in black box scenarios, uncovers system weaknesses beyond vulnerabilities, and fulfills regulatory requirements.

Audit
- Vulnerability assessment and penetration testing procedures and schedules should be reviewed, incorporating legal documents and establishing clear communication channels.
- It should be validated that VAPT activities cover all aspects of the API infrastructure.
- Documentation of VAPT findings should be ensured, with prioritization and prompt addressing of issues.

Remediation
- Regular VAPT activities should be scheduled based on risk assessments, compliance requirements, and industry best practices.
- Tests should encompass a wide range of scenarios, include various vulnerability tests, and incorporate both automated and manual testing methods.
- A procedure for prioritizing and remediating vulnerabilities should be established.

5 Decommission

This section outlines the essential steps for effectively decommissioning an API, focusing on establishing a decommissioning plan, communicating with stakeholders, and applying necessary actions. The plan emphasizes identifying items for phasing out, developing a secure data migration strategy, and creating a comprehensive timeline to enhance coordination and minimize miscommunication. Clear communication ensures that users, third-party companies, and developers receive timely updates about timelines, alternative solutions, and required actions, facilitating smoother transitions and preventing operational errors. Finally, executing critical actions involves carefully terminating services, retiring associated infrastructure, and reviewing network configurations to eliminate vulnerabilities.

5.1 Planning

This section outlines key steps for creating a decommissioning plan, focusing on identifying items for decommissioning, establishing a data migration strategy, and determining timelines. Identifying required actions ensures clarity in project planning and facilitates communication with stakeholders regarding phased-out items. A well-defined data migration strategy is crucial for securely managing data, deciding whether it should be migrated, erased, or archived to mitigate risks of leaks. Finally, a comprehensive timeline for decommissioning tasks enhances coordination among teams, minimizing miscommunication and promoting efficiency. Regular audits and documentation ensure accountability and adherence to plans.

5.1.1 Determine the Required Actions (Manual)

Description
Identification of what should be decommissioned should be carried out.

Rationale
Identification of decommissioned items is essential as it provides a comprehensive overview of the project's timeline and tasks, aiding in better organization. Clarity in planning is ensured through this process, facilitating effective communication with stakeholders by articulating the specific elements that will be phased out. By delineating decommissioning objectives early on, efforts can be streamlined, resources can be allocated efficiently, and potential disruptions to ongoing operations can be mitigated.

Audit
- The decommissioning plan should be examined to assess the outlined timelines and tasks.
- It should be ensured that the plan is shared and communicated to all teams involved.
- Communication with stakeholders regarding the decommissioning process should be established.

Remediation
- Tasks that have to be completed by the end of the project should be organized.
- Suitable timelines for each task should be determined in collaboration with the responsible team.
- Verification of documentation of actions should be conducted.

5.1.2 Establish a Data Migration Strategy (Manual)

Description
A migration plan should be created that determines whether data should be migrated, erased, or archived.

Rationale
The migration planning process is important for ensuring that data, which typically needs to be migrated and stored, is managed securely. Risks of leaks or data loss are mitigated through the implementation of organized and controlled transfer procedures.

Audit
- The data migration plan should be examined to assess the timelines and tasks outlined.
- It should be ensured that the plan is shared and communicated to all teams involved.
- The process should be verified to ensure it is automated

Remediation
- Tasks that need to be completed by the end of the migration should be organized.
- Tasks should be assigned to each team involved.
- The process should be automated, and its performance tested.
- A suitable timeline should be determined.
- Documentation of actions should be verified.

5.1.3 Decide on a Timeline (Manual)

Description
A timeline for decommissioning should be created, outlining the tasks and responsibilities for each team involved.

Rationale
A timeline for decommissioning, detailing tasks and responsibilities for each team involved, should be created to offer a clear plan of action, with miscommunications between interdependent teams minimized. This ensures that coordination and execution of decommissioning activities are achieved, promoting efficiency and avoiding delays or disruptions in the process.

Audit
- The existing action plan should be reviewed.
- Suitable timelines should be determined.
- Feedback from the teams regarding the feasibility of the proposed timeline should be collected.

Remediation
- The action plan should be put in place or updated.
- Communication with the teams on the timeline needed for actions should be established.
- Teams should be informed about the timeline required for each action.
- The timeline and actions should be documented.

5.2 Communication

This section emphasizes the importance of clear communication during the decommissioning process with users, third-party companies, and developers. Stakeholders must receive timely information about the decommissioning timeline, proposed alternative solutions, and required actions to prevent errors and enhance user experience. Notifying third-party companies is critical for transparency and ensuring they can take necessary actions, such as terminating contracts or removing integrations. Developers should also be informed of the action plan to minimize misunderstandings and streamline the transition. Establishing reliable communication channels and documenting all communications ensures that all parties are aligned and prepared for the changes ahead.

5.2.1 Notify Users (Manual)

Description
Clear timelines should be provided to all stakeholders, along with proposed alternative solutions and outlined actions to be taken.

Rationale
Errors, poor user experience, and security issues can be avoided through clear communication.

Audit
- Communication channels should be ensured.
- Communication with stakeholders should encompass the action plan, future steps, and any necessary user actions.
- Clarity and comprehensibility of communication with stakeholders should be ensured.

Remediation
- A reliable communication channel should be chosen.
- Stakeholders should be notified about the plan and informed of any actions they should take.
- They should also be notified of what actions may or may not need to be taken.
- All actions taken should be documented.

5.2.2 Notify Third-Party Companies (Manual)

Description
Third-party companies utilizing your API or whose APIs, libraries, or systems are integrated with your API should be notified regarding the decommissioning process, with clear timelines provided, alternative solutions outlined, and actions that will be taken specified.

Rationale
Clear communication is considered essential, as it provides third parties with insight into actions being taken, ensuring transparency throughout the decommissioning process. Notification allows necessary actions to be taken, such as terminating contracts or removing systems that support integration, thereby facilitating a smooth transition and minimizing disruptions for all parties involved.

Audit
- Communication channels should be ensured to be in place.
- Communication should include the action plan, future actions, and actions that may need to be taken.
- Clarity and comprehensibility of the message should be ensured.

Remediation
- A reliable communication channel should be chosen if one is not in place.
- Stakeholders should be notified about the plan and informed of any actions that should be taken.
- Communication should be conducted in a clear and comprehensible way.
- Actions taken should be documented.

5.2.3 Notify Developers (Manual)

Description
Developers should be provided with transparent timelines, alternative solutions proposed, and actions to be executed detailed.

Rationale
Clear communication is recognized as helpful in mitigating organizational errors and misunderstandings while also providing developers with a better understanding of the systems.

Audit
- Communication channels should be ensured to be in place.
- Communication should include the action plan, future actions, and actions that developers may need to take.
- Clarity and comprehensibility of the message should be ensured.

Remediation
- A reliable communication channel should be chosen if one is not in place.
- Developers should be notified about the plan and informed of any actions that should be taken.
- Communication should be conducted in a clear and comprehensible way.
- Actions taken should be documented.

5.3 Execution

This section outlines the critical actions required for successfully decommissioning an API, focusing on the termination of services, retirement of infrastructure, and review of network configurations. Terminating services must be carefully managed to ensure a smooth transition for stakeholders, allowing time to adapt while minimizing disruptions. The retirement of any physical or cloud infrastructure associated with the terminated services is essential to reduce costs and security risks, particularly concerning compliance with regulations regarding personal data. Additionally, reviewing existing network configurations is crucial to eliminate vulnerabilities that could be exploited, such as outdated DNS records and subdomains.

5.3.1 Terminate Services (Manual)

Description
Services should be ceased as part of the API decommissioning process.

Rationale
Ceasing services as part of the API decommissioning process is ensured to facilitate a smooth transition. This allows stakeholders time to adjust to the changes, minimizing interruptions and errors. By effectively coordinating the cessation of services, a seamless decommissioning process is achieved, maintaining operational continuity and reducing the risk of disruptions.

Audit
- The existing service termination plan should be reviewed.
- The plan should be ensured to be realistic and that all teams have tasks that can be completed on time.
- Monitoring logs should be reviewed for the best time to terminate.
- Proper handling of termination with stakeholders should be ensured, and users should be helped to identify alternatives to the terminated service.
- The termination should be verified.

Remediation
- A termination plan should be put in place, including steps needed for complete API service termination.
- A schedule should be planned out, and tasks should be assigned to each team.
- Strategies should be optimized to minimize the impact on the service, including fine-tuned error handling and directing users to the correct API.
- All actions taken should be documented.

5.3.2 Retire Infrastructure (Manual)

Description
Any physical or cloud infrastructure that supported the terminated service should be retired.

Rationale
The retirement of any physical or cloud infrastructure that supported the terminated service is imperative for several reasons. Timely retirement is ensured to minimize unnecessary costs associated with maintaining unused infrastructure. Additionally, the attack surface is reduced, as lingering infrastructure poses potential security risks. Compliance with regulations is also ensured, particularly concerning non-retired infrastructure containing personally identifiable information (PII), thereby mitigating legal and reputational risks.

Audit
- The existing infrastructure retirement plan should be reviewed.
- The plan and schedule should be checked to ensure realism and that all teams have tasks that can be delivered on time.
- Data should be validated to ensure it is stored securely or erased.
- Hardware should be ensured to be securely decommissioned and compliant with industry standards.

Remediation
- A plan should be scheduled for retiring hardware.
- Tasks should be shared with teams that are involved in decommissioning infrastructure.
- Data sanitization should be performed, and data should be securely erased or stored according to regulations.
- The smooth execution of the plan should be verified.
- The process should be automated.
- All actions taken should be documented.

5.3.3 Review Network Configurations (Manual)

Description
Existing network configurations should be reviewed before the termination of services and infrastructure.

Rationale
Forgotten network configurations could introduce vulnerabilities that may be exploited by malicious actors, posing security risks to the network. Configurations such as referenced subdomains that are no longer tied to infrastructure or forgotten CNAME records could lead to subdomain takeovers, potentially compromising the integrity of the network and its assets.

Audit
- The infrastructure documentation should be reviewed.
- The configuration and network associated with its hardware should be examined.
- It should be ensured that there are no dangling settings or network configurations, such as:
    - DNS records are pulled down.
    - Any referenced (sub-)domains are changed or deleted.
    - CNAME and DNS records that point to expired CMS domains are pulled.

Remediation
- Put correct configurations in place:
    - Pull down any unnecessary DNS records.
    - Any referenced (sub-)domains to other sites and/or APIs should be changed/deleted.
    - CNAME and DNS records that point to expired CMS domains should be deleted to avoid subdomain takeovers.

6 Glossary

ABAC (Attribute-Based Access Control)
An access control model that grants access based on attributes (e.g., a user's department or a document's metadata).

API (Application Programming Interface)
API (Application Programming Interface) is a set of definitions and protocols that enable different software applications to communicate and interact with each other.

API Key
A unique code passed in requests to authenticate an API call.

API Methods
Operations that can be performed on resources in an API, typically including CRUD actions.

Authentication
Authentication, sometimes abbreviated as AuthN, is the process of verifying the identity of the user who initiated an API request. Some standard authentication methods include API keys, OAuth, JWT Authentication and basic authentication.

Authorization
Authorization, sometimes abbreviated as AuthZ, is the process of granting or denying access to different operations, resources, or data in the API for an authenticated user.

CDNs (Content Delivery Networks)
Content Delivery Networks: Systems of servers that distribute web content across multiple locations to reduce load time and increase availability.

CNAME (Canonical Name Record)
A type of DNS record that maps an alias name to a true or canonical domain name.

Cross-Site Scripting (XSS)
A vulnerability that allows attackers to inject malicious scripts into web pages viewed by other users.

CRUD
Create, Read, Update, Delete: The basic operations that can be performed on data.

DDoS (Distributed Denial of Service)
An attack where multiple systems flood a targeted server with requests, overwhelming it and causing a service disruption.

DELETE request
An HTTP method used to remove a specified resource from a server.

DNS (Domain Name System)
The system that translates domain names (like example.com) into IP addresses for locating websites.

Endpoints
The specific address (URL) where an API can be accessed. Each endpoint typically represents a distinct function or resource provided by the API.

GET Request
An HTTP method used to request data from a specified resource.

GDPR (General Data Protection Regulation)
A European Union regulation that governs the privacy and security of personal data.

HTTPS (Hypertext Transfer Protocol Secure)
A protocol that encrypts HTTP traffic to secure communication over the internet.

Identity and Access Management (IAM)
A framework for managing user identities, authentication, and access rights within an organization.

Insecure Direct Object Reference (IDOR)
A vulnerability that occurs when an application provides direct access to objects based on user-supplied input without proper authorization.

JWT (JSON Web Token)
A compact token format used for securely transmitting information between parties as a JSON object.

Man-in-the-Middle (MITM) Attack
A cyberattack where a third party intercepts communication between two parties to eavesdrop or alter data.

Multi-Factor Authentication (MFA)
A security process that requires two or more methods of verification to access a system.

NVD (National Vulnerability Database)
A U.S. government repository of publicly known security vulnerabilities.

OAuth
An open standard for token-based authentication and authorization that allows third-party applications to access user data without exposing passwords.

PII (Personally Identifiable Information)
Data that can uniquely identify an individual, such as name, address, or social security number.

POST Request
An HTTP method used to send data to a server to create or update a resource.

PUT Request
An HTTP method used to update a current resource with new data.

RAML (RESTful API Modeling Language)
A language for defining APIs that is readable by both humans and machines.

RBAC (Role-Based access Control)
An access control model that grants access based on user roles.

REST
Representational State Transfer: An architectural style for designing networked applications, primarily using HTTP for requests.

SQL
Structured Query Language: A programming language for managing and querying relational databases.

SQL Injection
A type of cyberattack that manipulates SQL queries by injecting malicious code into an application's database query.

SSL (Secure Sockets Layer)
A protocol for encrypting data between a web server and a client to secure communications.

TLS (Transport Layer Security)
An improved, more secure version of SSL for encrypting data in transit.

VAPT (Vulnerability Assessment and Penetration Testing)
A security practice to identify and address security vulnerabilities in a system.

Web Application Firewall (WAF)
A security tool that filters and monitors HTTP traffic between a web application and the internet.

Versioning
A method to manage changes in API functions and features over time, often using numbers (e.g., v1, v2) to indicate major updates.

WAF (Web Application Firewall)
A security tool designed to protect web applications by filtering and monitoring HTTP traffic between a web application and the internet.

The Center for Internet Security, Inc. (CIS®) makes the connected world a safer place for people, businesses, and governments through our core competencies of collaboration and innovation. We are a community-driven nonprofit, responsible for the CIS Critical Security Controls® and CIS Benchmarks™, globally recognized best practices for securing IT systems and data. We lead a global community of IT professionals to continuously evolve these standards and provide products and services to proactively safeguard against emerging threats. Our CIS Hardened Images® provide secure, on-demand, scalable computing environments in the cloud.

CIS is home to the Multi-State Information Sharing and Analysis Center® (MS-ISAC®), the trusted resource for cyber threat prevention, protection, response, and recovery for U.S. State, Local, Tribal, and Territorial government entities. To learn more, visit CISecurity.org or follow us on X: @CISecurity.