Middle East Technical University
Informatics Institute

API SECURITY GUIDELINE FOR DEVELOPERS AND PRODUCT OWNERS

Advisor Name:
Cihangir TEZCAN
(METU)

Student Name:
Okan AKGÜNDÜZ
Department of Cyber Security

January 2025

TECHNICAL REPORT
METU/II-TR-2025-

Orta Doğu Teknik Üniversitesi
Enformatik Enstitüsü

YAZILIM GELİŞTİRİCİLERİ VE ÜRÜN YÖNETİCİLERİ İÇİN API GÜVENLİĞİ KILAVUZU

Danışman Adı:
Cihangir TEZCAN
(ODTÜ)

Öğrenci Adı:
Okan AKGÜNDÜZ
Siber Güvenlik Bölümü

Ocak 2025

TEKNİK RAPOR
ODTÜ/II-TR-2025-

REPORT DOCUMENTATION PAGE

1. AGENCY USE ONLY (Internal Use)	**2. REPORT DATE** 10.01.2025

3. TITLE AND SUBTITLE

API SECURITY GUIDELINE FOR DEVELOPERS AND PRODUCT OWNERS

4. AUTHOR (S)	**5. REPORT NUMBER (Internal Use)**
Okan Akgündüz	METU/II-TR-2025-

6. SPONSORING/ MONITORING AGENCY NAME(S) AND SIGNATURE(S)

Cyber Security Master's Programme, Department of Cyber Security, Informatics Institute, METU

Advisor: Cihangir Tezcan Signature:

7. SUPPLEMENTARY NOTES

8. ABSTRACT (MAXIMUM 200 WORDS)

This report provides a survey for security topics, practices, and recommendations to consider when designing, developing, and maintaining an API service. It is primarily intended for developers, product owners, and other stakeholders involved in API development. Additionally, it is relevant to those who consume API services.

The report draws from widely recognized security standards, guidelines, and surveys. It is created by examining the most common API security vulnerabilities identified in existing studies. Its scope of work is then expanded to include potential vulnerabilities that can arise during an API service's development process/lifecycle. For each identified vulnerability/risk/issue, the report outlines recommended mitigations based on standards and guidelines.

The result is a categorized list that can be used as an API Security Guideline.

9. SUBJECT TERMS	**10. NUMBER OF PAGES** **43**

TABLE OF CONTENTS

TABLE OF CONTENTS IV
1.INTRODUCTION 1
2.PROJECT INITIATION AND DESIGN 3
2.1. PD01 - Neglecting Security in Early Steps 3
2.2. PD02 – No “Secure by Default” Approach 4
2.3. PD03 - Not Documenting Service Internals 5
2.4. PD04 - Not Documenting Role and Privilege Definitions 5
2.5. PD05 - Lack of Fail-Safe/Fail-Secure Approach 6
2.6. PD06 - Lack of Documentation for Users 6
2.7. PD07 - Lack of Understanding of Performance Requirements 7
2.8. PD08 - Not Identifying Data and API Service Components 7
2.9. PD09 - Lack of Security Awareness 8
3.SOFTWARE DEVELOPMENT 9
3.1. SD01 - Sharing Too Much Data in Responses and Error Messages 9
3.2. SD02 - Lack of Protection Against Race Conditions 10
3.3. SD03 - Trusting Input Data Directly Without Validation 11
3.4. SD04 - Lack of SSRF Protection 11
3.5. SD05 - Not Using the "no-cache" Attribute for Sensitive Data 12
3.6. SD06 - Not Implementing Proper HTTP Methods for Requests 13
3.7. SD07 - Including Sensitive Data in URLs 13
3.8. SD08 - Predictable Object/Endpoint IDs 13
3.9. SD09 - Debug Features Being Available in Production 14
3.10. SD10 - Lack of Content Security Policy 15
3.11. SD11 - Lack of Rate Limiting and Timeouts 15
3.12. SD12 - Not Filtering Response Data 16
3.13. SD13 - Lack of Compliance with Software Licenses 16
4.AUTHENTICATION AND AUTHORIZATION 19
4.1. AA01 - Weak Password Policy 19
4.2. AA02 - Lack of Brute Force Protection 20
4.3. AA03 - Non-Secure Credential Storage 20
4.4. AA04 - Non-Secure Session Data Creation 21
4.5. AA05 - Lack of Endpoint/Object-Level Authorization 22
4.6. AA06 - Mass Assignment 22
4.7. AA07 - Excessive Privileges 23
4.8. AA08 – Openly Accessible Administrative Resources 23
4.9. AA09 – Not Requiring Extra Proofs for Critical Actions 24
5.IMPLEMENTATION AND MAINTENANCE 25
5.1. IM01 - Using Non-Secure Communication Channels 25
5.2. IM02 - Not Filtering Access Based on Source 26
5.3. IM03 - Not Storing Secrets in Secure Storage(s) 26
5.4. IM04 - Lack of Active Protection Tools 27
5.5. IM05 - Lack of Reliable Backups 27
5.6. IM06 - Not Retiring Old Data 28
5.7. IM07 - Misconfigurations 28
5.8. IM08 - Lack of Vulnerability and Patch Management Processes 28
5.9. IM09 - Not Retiring Unused API Endpoints 29

5.10. IM10 - Directory Listings 29
6.TESTING AND VALIDATION 31
6.1. TV01 - Lack of Testing for Authentication and Authorization 31
6.2. TV02 - Not Conducting Penetration Test 32
6.3. TV03 - Lack of SAST and DAST Usage 33
6.4. TV04 - Lack of Negative Testing 33
7.MONITORING AND RESPONSE 35
7.1. MR01 - Not Monitoring Performance 35
7.2. MR02 - Not Logging Business Critical Events 36
7.3. MR03 - Not Logging Actions and Data Accesses 36
7.4. MR04 - Not Logging Errors 36
7.5. MR05 - Excessive Logging 37
7.6. MR06 - Not Monitoring Logs and Alerts 37
7.7. MR07 - Not Having an Incident Response Process 38
8.CONCLUSION 39
REFERENCES 41

1. INTRODUCTION

Application Programming Interfaces (APIs) have become a vastly used technology in the last decade, enabling services across both internet and internal networks. According to an October 2018 Akamai traffic review, 83% of internet traffic[1] now comprises of API calls. Also, Cloudflare's inspection on requests/calls of the first week of December 2021, API calls represented 54% of their total requests[2]. Both reports show a substantial use of APIs. When we look at the development side of the APIs, surveys show 89% of developers[3] use APIs in their projects. As APIs continue to grow in use, the importance of security also grows.

When we look at the current security landscape of APIs from 2022 to 2024, the number of vulnerabilities being registered to CVE database increased significantly. In 2022, there were 612 vulnerability registrations to the CVE database. This increased to 805 in 2023 and then to 977 in 2024. Which shows the increasing trend of security issues regarding to APIs.

In the academic world, the increasing importance of security regarding to APIs was also showing its results. Several academic papers/works regarding to security issues of APIs were already shared with the world. (Munsch & Munsch, 2021) shows some of the real-world security related implications of APIs. (Hussain, Hussain, Noye, & Sharieh, 2020) shares privacy related issues and recommendations. (Zhao, 2024), (Díaz-Rojas, Alejandro, Ocharán-Hernández, Pérez-Arriaga, & Limón, 2021), (Bhuiyan, Begum, Rahman, & Hadid, 2018) present valuable information regarding to current vulnerabilities that affect the APIs, and they also provide recommendations and strategies to mitigate these vulnerabilities. In this report, API security related information will be presented in a way for developers and product owners to understand the possible security implications of their actions while developing an API service and see recommendations to mitigate these implications.

This report provides a survey of security considerations, practices, and recommendations for designing, developing, and maintaining API services. It is intended primarily for developers, product owners, and other stakeholders involved in API development but is equally relevant for API consumers. Drawing from widely recognized security standards, guidelines, and surveys, the report identifies the most common API vulnerabilities, examines potential risks that may emerge during the API development process/lifecycle, and proposes mitigations and recommendations. To ensure practical applicability, the findings are

[1] State of The Internet Security Retail Attacks and API Traffic Report 2019 https://www.akamai.com/site/it/documents/state-of-the-internet/state-of-the-internet-security-retail-attacks-and-api-traffic-report-2019.pdf February 2019

[2] Landscape of API Traffic https://blog.cloudflare.com/landscape-of-api-traffic/#api-use-in-2021 January 2022

[3] State of The Developer Nation - 24th Edition https://research.slashdata.co/reports/65b8ff12343d024fbe664ae6 May 2023

categorized roughly based on DevOps phases, highlighting when specific countermeasures should be implemented or when certain risks are likely to arise.

This report is not an exhaustive list for API security. Because each API service project has unique objectives, requirements, resources, and risk tolerances, the report focuses on providing general best practices and guidelines that can be adapted to suit the specific needs and constraints of individual projects. This report does not include specific API technologies or frameworks such as SOAP, REST, GraphQL, and gRPC, to maintain a broader focus on general security principles and practices that are applicable across all web-based APIs.

The result is a structured API Security Guideline, offering actionable insights for building robust, secure APIs. While not universally prescriptive, this guideline aims to provide a strong foundation for understanding and addressing API security risks for developers and product owners.

2. PROJECT INITIATION AND DESIGN

Like any other general software development project, API service's project starts with an initiation phase and continues with design. In the initiation step, business needs and objectives are determined with the stakeholders. Required resources are gathered and a team is created. Then, the project continues with the design step. In this step, a design is created (or prior design is updated) based on the business needs, objectives and current resources. Both steps define the success and the security of an API service and should be done carefully.

In this chapter, following security issues will be addressed regarding to these development process/lifecycle steps.

1 Neglecting Security in Early Steps

2 No "Secure by Default" Approach

3 Not Documenting Service Internals

4 Not Documenting Role and Privilege Definitions

5 Lack of Fail-Safe/Fail-Secure Approach

6 Lack of Documentation for Users

7 Lack of Understanding of Performance Requirements

8 Not Identifying Data and API Service Components

9 Lack of Security Awareness

2.1. PD01 - Neglecting Security in Early Steps

Issue: Security should not be treated as an afterthought or confined solely to technological implementations. It must be integrated into every phase of a project, from requirement analysis to end-of-life planning[4].

Overlooking security in the early stages can lead to significant challenges, such as the need for a complete redesign of the API service. This may be infeasible or prohibitively expensive during the later stages of the development.

[4] OWASP ASVS Version 4.0.3 – Verification 1.1.1 https://github.com/OWASP/ASVS/raw/v4.0.3/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.3-en.pdf October 2021

Also, any partial fix implemented in a later stage as a workaround can create a false sense of security, and potentially lead stakeholders to make decisions based on inaccurate information.

Recommendation: Discuss security considerations early to identify potential risks and create a comprehensive risk assessment plan[5,6].

Perform threat modelling[7,8,9] and consider potential abuse cases[10]. Use these insights to adjust the API service's design as needed.

Even machine learning can be utilized to enhance the processes of threat modelling and risk assessment. (Dahiya & Ranjan, 2021) work shows potential benefits of using Machine Learning techniques for threat modelling and risk assessment.

By addressing security proactively and thoroughly at each stage, costly redesigns in future may be avoided. Enhanced system reliability, and informed decision-making throughout the project can be achieved.

2.2. PD02 – No "Secure by Default" Approach

Issue: API users may begin using the service without fully understanding its details, which may lead to incorrect assumptions. This can result in unexpected usage patterns that pose risks to both the users and the API service.

Recommendation: Anticipate and account for unexpected user behaviour. Implement strict security controls as default settings[11,12,13,14,15] to minimize potential risks. Provide users with the flexibility to modify or disable certain security controls, if necessary, but ensure that any associated risks are limited to the individual user and do not impact the API service.

---

[5] NIST SP 800-30 Rev. 1 - Guide for Conducting Risk Assessments https://doi.org/10.6028/NIST.SP.800-30r1 September 2012
[6] OWASP Risk Rating Methodology https://owasp.org/www-community/OWASP_Risk_Rating_Methodology June 2023
[7] OWASP Threat Modeling https://owasp.org/www-community/Threat_Modeling March 2024
[8] OWASP Threat Modeling Process https://owasp.org/www-community/Threat_Modeling_Process July 2024
[9] OWASP Threat Modeling Cheat Sheet https://cheatsheetseries.owasp.org/cheatsheets/Threat_Modeling_Cheat_Sheet.html December 2024
[10] OWASP Abuse Case Cheat Sheet https://cheatsheetseries.owasp.org/cheatsheets/Abuse_Case_Cheat_Sheet.html June 2024
[11] OWASP SAMM – Architecture Design https://owaspsamm.org/resources/pdf/ February 2020
[12] OWASP Top 10 API Security Risks – 2019 - API7:2019 https://owasp.org/API-Security/editions/2019/en/0x11-t10/ December 2019
[13] NIST SP 800-160v1r1 - Engineering Trustworthy Secure Systems – E.22. Protective Defaults https://doi.org/10.6028/NIST.SP.800-160v1r1 November 2022
[14] NIST SP 800-53 Rev. 5 - SA-8(23) SECURE DEFAULTS https://doi.org/10.6028/NIST.SP.800-53r5 September 2020
[15] NIST SP 800-218 - PW.9 https://doi.org/10.6028/NIST.SP.800-218 February 2022

Strive to balance strong security with a user-friendly experience. If security measures are overly restrictive or inconvenient, users may bypass or disable them entirely, undermining the API service's expected usage.

2.3. PD03 - Not Documenting Service Internals

Issue: Documentation is often overlooked, because it's not directly tied to business solutions. However, as API services evolve, change, or fail, the absence of proper documentation can lead to poor decisions, missed actions, and operational inefficiencies.

Recommendation: Document[16,17] the API service's business flow to ensure a shared understanding of its functionality.

Create a detailed list of used components as an asset inventory. Asset inventory can be used for uptime tracking, software version management, supplier and dependency management, maintenance planning.

List external dependencies and the data exchanged with them to manage integrations effectively.

Document infrastructure specifics to support maintenance processes and streamline problem management.

Comprehensive documentation ensures clarity, improves decision-making, and supports long-term maintenance and scalability of the API service.

2.4. PD04 - Not Documenting Role and Privilege Definitions

Issue: When roles and their associated privileges are not well-documented, it can be difficult to determine which role can do what. As API services and business needs evolve, a lack of proper documentation can lead to; poor decision-making during updates or changes, users lacking sufficient rights to perform their tasks, users being granted excessive rights, increasing the risk of both benign errors and malicious actions.

Recommendation: Comprehensive role documentation[18] reduces confusion, mitigates risks, and facilitates efficient and secure management of the API service over time.

Clearly document each role's privileges. Include the rationale behind assigning specific privileges to provide context for future decision-making.

Regularly review and update role documentation to ensure it reflects the current state of the API service and its requirements.

---

[16] OWASP SAMM – Architecture Assessment https://owaspsamm.org/resources/pdf/ February 2020
[17] OWASP ASVS Version 4.0.3 – Verification 1.1.4 https://github.com/OWASP/ASVS/raw/v4.0.3/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.3-en.pdf October 2021
[18] NIST SP 800-53 Rev. 5 - AC-2 ACCOUNT MANAGEMENT https://doi.org/10.6028/NIST.SP.800-53r5 September 2020

2.5. PD05 - Lack of Fail-Safe/Fail-Secure Approach

Issue: Unexpected events or failures are inevitable for any system. Without a fail-safe or fail-secure approach, an API service may enter a corrupted state without a clear path to recovery. This creates significant risks, such as prolonged instability or downtime. It also provides an attack surface for attackers, who often exploit unexpected conditions to trigger unintended behaviours. Any emergency measures such as disabling features to improve responsiveness can create vulnerabilities.

Recommendation: By designing for fail-safe and fail-secure[19,20,21] operations, stability and security of the API service can be ensured, even in adverse or unexpected conditions.

Build mechanisms to detect and classify failure conditions within the API service. Business flows, asset inventory and other important documents can be utilized to inspect possible failures and their impacts.

For each fail-state, specify clear and appropriate responses that help the API service to recover or limit the further possible damage.

Limit the use or access of compromised or unsalvageable features to prevent further impact.

Avoid disabling security features unless necessary, and always document the reasoning behind such decisions.

2.6. PD06 - Lack of Documentation for Users

Issue: Even well-intentioned users can unintentionally misuse an API service if they lack clear guidance on its proper usage[22]. This benign misuse can inadvertently cause harm, such as introducing errors, overloading the service, or exposing vulnerabilities.

Recommendation: Comprehensive documentation empowers users to interact with the API service effectively and safely, reducing risks and improving the overall user experience.

Provide clear and accessible documentation to help users understand the API's functionality, expected usage, and limitations.

Include a detailed Terms of Service to specify what actions are allowed and what behaviours should be avoided.

---

[19] OWASP SAMM – Architecture Design https://owaspsamm.org/resources/pdf/ February 2020
[20] OWASP ASVS Version 4.0.3 – Verification 7 Error Handling and Logging https://github.com/OWASP/ASVS/raw/v4.0.3/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.3-en.pdf October 2021
[21] NIST SP 800-160v1r1 - Engineering Trustworthy Secure Systems – E.23. Protective Failure https://doi.org/10.6028/NIST.SP.800-160v1r1 November 2022

Create programmatically usable documentation for API consumers (applications). Include details like data formats, expected values, error codes, and response structures to ensure seamless integration and reduce potential misuse.

2.7. PD07 - Lack of Understanding of Performance Requirements

Issue: Availability is a critical metric for both business operations and security. Without clear performance requirements[23,24], it becomes challenging to distinguish between legitimate high-traffic periods (e.g., busy work hours) and malicious activity like denial-of-service attacks. This can delay or misdirect response efforts, negatively impacting both functionality and security.

Recommendation: By understanding and documenting performance requirements, it becomes possible to proactively manage availability, identify anomalies, and optimize both business outcomes and security measures.

Maintain ongoing communication with stakeholders on the business side to understand their expectations and performance needs.

Establish clear performance benchmarks, such as expected traffic patterns, peak usage times, and acceptable response times.

Provide service quality metrics (e.g., uptime, response time, throughput) to stakeholders and gather their feedback to ensure alignment with business goals.

2.8. PD08 - Not Identifying Data and API Service Components

Issue: Every API service and the data it processes is unique. Without proper identification and classification, it becomes difficult to define appropriate protection requirements.

This can lead to overuse of excessive or costly protection mechanisms or insufficient protection for critical components or data.

---

[22] NIST SP 800-160v1r1 - Engineering Trustworthy Secure Systems – 3.1. The Concept of Security https://doi.org/10.6028/NIST.SP.800-160v1r1 November 2022
[23] NIST SP 800-53 Rev. 5 – (28) ACCEPTABLE SECURITY https://doi.org/10.6028/NIST.SP.800-53r5 September 2020
[24] NIST SP 800-160v1r1 - Engineering Trustworthy Secure Systems – 2.2. Systems Engineering Foundations https://doi.org/10.6028/NIST.SP.800-160v1r1 November 2022

Recommendation: By identifying and classifying data and service components and analysing potential impacts; informed, balanced decisions can be made about protection measures[25,26,27]. Therefore, optimizing both security and cost-efficiency can be achieved easier.

Identify and list all data processed by the API service. Also, evaluate the API service's components and their roles.

Assess the impact of breaches in Confidentiality, Integrity, and Availability for each type of data in the event of a security incident. Similarly, determine how breaches in Confidentiality, Integrity, and Availability could affect each component within the API service.

Assign appropriate labels (e.g., "Secret Data", "Availability Important") to both data types and API service components based on their criticality.

Use these labels to guide decisions about protection mechanisms, ensuring that resources are allocated effectively and appropriately.

2.9. PD09 - Lack of Security Awareness

Issue: Every team member involved in an API service project has specific responsibilities and roles. Therefore, their contributions directly affect the project's security posture. Sometimes even well-meaning mistakes from them can introduce significant risks, making security awareness essential at every level.

Recommendation: By fostering a culture of security awareness through targeted training[28,29,30,31], risks can be reduced, decision-making can be improved. Ensure that every team member contributes to the overall security of the API service by implementing proper security training.Provide security training that aligns with each team member's role and responsibilities.

Offer training on both personal and technical security topics, from password hygiene and phishing prevention to secure API design and data protection. Regularly update training to reflect emerging threats and evolving security best practices.

[25] OWASP SAMM – Threat Assessment - Application Risk Profile https://owaspsamm.org/resources/pdf/ February 2020
[26] OWASP ASVS Version 4.0.3 – Verification 1.8.1 https://github.com/OWASP/ASVS/raw/v4.0.3/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.3-en.pdf October 2021
[27] NIST SP 800-160v1r1 - Engineering Trustworthy Secure Systems – 3.4. The Concept of Assets https://doi.org/10.6028/NIST.SP.800-160v1r1 November 2022
[28] OWASP SAMM – Education & Guidance https://owaspsamm.org/resources/pdf/ February 2020
[29] NIST SP 800-160v1r1 - Engineering Trustworthy Secure Systems – H.10.3. Security Activities and Tasks https://doi.org/10.6028/NIST.SP.800-160v1r1 November 2022
[30] NIST SP 800-53 Rev. 5 – 3.2 AWARENESS AND TRAINING https://doi.org/10.6028/NIST.SP.800-53r5 September 2020

3. SOFTWARE DEVELOPMENT

This step is the real coding step of the API service. Developers use designs and coding tasks that were created from business needs and objectives to code the API service. Main deliverable from this step is the source code that provides the required business logic to achieve what API service is needed for.

In this chapter, following security issues will be addressed regarding to this development process/lifecycle step.

1 Sharing Too Much Data in Responses and Error Messages

2 Lack of Protection Against Race Conditions

3 Trusting Input Data Directly Without Validation

4 Lack of SSRF Protection

5 Not Using the "no-cache" Attribute for Sensitive Data

6 Not Implementing Proper HTTP Methods for Requests

7 Including Sensitive Data in URLs

8 Predictable Object/Endpoint IDs

9 Debug Features Being Available in Production

10 Lack of Content Security Policy

11 Lack of Rate Limiting and Timeouts

12 Not Filtering Response Data

13 Lack of Compliance with Software Licenses

3.1. SD01 - Sharing Too Much Data in Responses and Error Messages

Issue: Attackers often begin by gathering information about a system (reconnaissance[32]) to understand its architecture, technologies, and potential vulnerabilities. This reconnaissance phase is made easier if the API

[31] NIST SP 800-218 - PO.2.2 https://doi.org/10.6028/NIST.SP.800-218 February 2022
[32] MITRE ATT&CK – Reconnaissance https://attack.mitre.org/tactics/TA0043/ October 2020

service provides overly detailed responses or error messages[33]. Information gathered in the reconnaissance phase is used to identify weak points within the API service and exploit them effectively. Although detailed information in API responses and error messages are useful for developers during debugging, it can also inadvertently benefit malicious actors in their reconnaissance actions.

Recommendation: Keep API responses concise and relevant. Avoid including sensitive or unnecessary data such as system configurations, user roles, or technical details unless explicitly required for functionality.

Implement error responses as generic as possible. Do not include stack traces, database queries, or debugging data in the response and similar internal information.

Providing simple error IDs to users can be helpful for debugging the issues. These error IDs can be used to match error logs with user's error feedback. But these IDs should be random and shouldn't provide any information relevant to the state of the API service.

3.2. SD02 - Lack of Protection Against Race Conditions

Issue: Race conditions occur when multiple processes or threads execute concurrently and attempt to access or modify the same resource in an uncoordinated manner. In such cases, attackers can exploit these vulnerabilities to perform unauthorized actions, bypass logical protections, or manipulate data inconsistently. This can create too many consequences such as double spending, privilege escalation, or bypassing transaction limits and so on.

Recommendation: Analyse business flows to identify processes where simultaneous actions on shared resources might cause conflicts[34].

Use atomic operations to make sure that critical actions are completed fully or not at all. This can be used for eliminating intermediate states that attackers can exploit.

Use locking mechanisms (e.g., mutexes, semaphores) to prevent concurrent access to shared resources.

Use idempotency keys for critical API endpoints. Idempotency ensures that repeated requests produce the same outcome against duplicate actions.

[33] OWASP ASVS Version 4.0.3 – Verification 7 Error Handling and Logging - Control Objective https://github.com/OWASP/ASVS/raw/v4.0.3/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.3-en.pdf October 2021
[34] OWASP ASVS Version 4.0.3 – Verification 1.11.3 https://github.com/OWASP/ASVS/raw/v4.0.3/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.3-en.pdf October 2021

3.3. SD03 - Trusting Input Data Directly Without Validation

Issue: One of the most common techniques that attackers use is injection attacks, where malicious input is crafted to exploit an API service's trust in the data it receives. If the API service directly processes the input which is expected to be only read/written without validation, attackers can manipulate API service to perform unintended operations, such as unauthorized database queries (SQL injection[35]), remote code execution, or malicious command execution.

Recommendation: Implement input validation techniques[36] before parsing or using any data provided by users or external parties[37].

Define strict maximum and minimum length constraints for all inputs based on expected data. Inspect the input based on expected data type (integer, string, etc.). Use regular expressions (regex[38]) to enforce specific input.

If possible, do semantic validation. Verify that the input makes sense in the given context (like; there are only 12 specific months).

Strip out or encode characters that might have special significance in certain contexts.

Reject any unexpected or malformed data gracefully.

Only rely on server-side validation. Client-side validation can be easily bypassed by attackers. Client-side validation should be only used for better user experience.

Based on used technologies, there can be frameworks or libraries that can provide input validation. Try to implement them into the API service.

Also, for any files provided by users, the provided file should be checked against the "content-type" header sent by the user.

3.4. SD04 - Lack of SSRF Protection

Issue: Server-Side Request Forgery (SSRF)[39,40] is a vulnerability where an attacker manipulates an API service to send unauthorized requests to internal or external resources. Because the API service may have

[35] Common Weakness Enumeration - CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') https://cwe.mitre.org/data/definitions/89.html November 2024
[36] OWASP ASVS Version 4.0.3 – Verification 5.1 Input Validation https://github.com/OWASP/ASVS/raw/v4.0.3/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.3-en.pdf October 2021
[37] OWASP Top 10 API Security Risks – 2023 - API10:2023 https://owasp.org/API-Security/editions/2023/en/0x11-t10/ December 2019
[38] OWASP - OWASP Validation Regex Repository https://owasp.org/www-community/OWASP_Validation_Regex_Repository September 2023

privileges and access to internal systems, an SSRF attack can be used by attackers to bypass protections implemented against them. Also, SSRF attack can be used to exfiltrate data from API service to an attacker controlled external service.

Recommendation: Start by implementing data validation to block any unexpected data.

Filter communication requests triggered by API service. Create a whitelist and only allow what communications should be done by the API service[41].

Disable unused or unexpected URL schemes (like file://) which can be used to trigger communication or execution of commands.

Do not allow open redirects. Open redirects can be used for API service to follow or redirect to arbitrary URLs provided. Also, open redirects can be utilized to bypass the SSRF related protection whitelists.

Validate and encode any URL that needs to be used in order to prevent attackers from injecting special characters and bypassing filters.

3.5. SD05 - Not Using the "no-cache" Attribute for Sensitive Data

Issue: In web-based transactions, there can be multiple web gateways between the users and the API service. These gateways can be used for many things like temporarily storing data to lower the bandwidth requirements for frequently requested data. This technique is called caching. An attacker might trick one of those web gateways within the path to get access to the cached data which can have sensitive data.

Recommendation: Classify sensitive data that API service can serve to users.

Add "Cache-Control: no-cache" header to responses which can include sensitive data. This way any legitimate web gateway will honour the header and would not cache the data[42].

If user side caching is wanted, "Cache-Control: private" header can be used to cache the data only at the user.

---

[39] Common Weakness Enumeration - CWE-918: Server-Side Request Forgery (SSRF) https://cwe.mitre.org/data/definitions/918.html November 2024
[40] OWASP Top 10 API Security Risks – 2023 – API7:2023 https://owasp.org/API-Security/editions/2023/en/0x11-t10/ December 2019
[41] OWASP ASVS Version 4.0.3 – Verification 12.6.1 https://github.com/OWASP/ASVS/raw/v4.0.3/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.3-en.pdf October 2021
[42] OWASP ASVS Version 4.0.3 – Verification 8.1.1 https://github.com/OWASP/ASVS/raw/v4.0.3/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.3-en.pdf October 2021

3.6. SD06 - Not Implementing Proper HTTP Methods for Requests

Issue: Most of the actions in an API service can be categorized within CRUD (create, read, update, delete). HTTP protocol and its implementations inherit protections for specific actions regarding to CRUD. Ambiguity of any actions in case of CRUD can lead to unexpected behaviour which can be used by an attacker to bypass controls that work for specific actions.

Recommendation: Ensure that actions in the API strictly adhere to the appropriate HTTP methods[43].

POST method should only be used for “create” actions.

GET method should only be used for “read” actions.

PUT method should only be used for “update” actions.

DELETE method should only be used for “delete” actions.

3.7. SD07 - Including Sensitive Data in URLs

Issue: URLs are often logged and analysed. Even if all communication is expected to be encrypted, a web gateway may be present between the API service and the user. Also, because URLs are expected to not include any sensitive data, user's browsers and/or client applications may not protect URL information strongly. In the case of a URL including sensitive data, the sensitive data within the URL may be leaked benignly or maliciously.

Recommendation: Design the URL schema of API service to not include any sensitive data in both requests and responses[44]. Any sensitive data should be only transmitted in the headers or in the body.

3.8. SD08 - Predictable Object/Endpoint IDs

Issue: API services often rely on unique IDs to identify objects or endpoints. If the IDs follow a predictable pattern, attackers with limited privileges can exploit this by guessing IDs outside their authorized scope. If there are weaknesses in authorization controls, attackers can do actions against objects with guessed IDs without needing proper privilege. Predictable IDs can also be used for information gathering, reconnaissance by exposing the structure and relationships of the data within the system.

[43] OWASP ASVS Version 4.0.3 – Verification 13.2.1 https://github.com/OWASP/ASVS/raw/v4.0.3/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.3-en.pdf October 2021
[44] OWASP ASVS Version 4.0.3 – Verification 13.1.3 https://github.com/OWASP/ASVS/raw/v4.0.3/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.3-en.pdf October 2021

Recommendation: UUID[45] (also known as GUID) Version 4 can be implemented. UUID v4 can generate IDs with random data when random IDs are needed[46]. Cryptographically Secure Pseudorandom Number Generators (CSPRNG) can be used for random data input which reduces the predictability of the IDs. Also, because UUIDs can have a big range of possible numbers, using brute forcing techniques to find another possible value would be extremely hard.

3.9. SD09 - Debug Features Being Available in Production

Issue: Debugging features are essential during development to find problems. But it can be highly dangerous if left enabled in production environments[47,48,49]. These features are designed to expose detailed information about the API service's internal state, configuration, and operation to aid developers in troubleshooting. If debug features are left enabled in the production version of the application, attackers can find them out and exploit them to get more information about the API service, exfiltrate data and do unauthorized actions.

Recommendation: Have different levels of environments based on their use within the development process/lifecycle. Some of the development environments widely used in development projects are DEV, TEST, STAGE, PROD. DEV environment is used by developers to test their code roughly, mostly for testing whether code can be compiled or not. There is a low level of expectancy against business logic being tested in this environment. This environment is highly disposable in its nature and does not use any real data. The TEST environment is used for testing whether business logic is implemented as expected or not. This environment also should not include any real data. The TEST environment should also be used for security related tests, because it may be the last step before using real data. STAGE or sometimes called as PrePROD environment is the environment that uses real data to test its business logic. Because this environment uses real data, any security related vulnerability can have real consequences. PROD environment is the production environment that gives service to users and therefore uses real data and is expected to have security, accuracy in the execution and high availability.

Clearly label all debug-related code in the development process, ensuring it is easily identifiable for removal or disabling before deployment to the production environment.

---

[45] IETF – RFC 9562 Universally Unique IDentifiers (UUIDs) https://datatracker.ietf.org/doc/html/rfc9562 May 2024
[46] OWASP ASVS Version 4.0.3 – Verification 6.3.2 https://github.com/OWASP/ASVS/raw/v4.0.3/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.3-en.pdf October 2021
[47] Common Weakness Enumeration - CWE-489: Active Debug Code https://cwe.mitre.org/data/definitions/489.html November 2024
[48] OWASP Top 10 API Security Risks – 2019 – API9:2019 https://owasp.org/API-Security/editions/2019/en/0x11-t10/ December 2019
[49] OWASP Top 10 API Security Risks – 2023 – API9:2023 https://owasp.org/API-Security/editions/2023/en/0x11-t10/ December 2019

Most of the development environment aware version control and CI/CD tools can be used to easily filter out debugging code when doing deployment to production environments.

3.10. SD10 - Lack of Content Security Policy

Issue: Modern web services often rely on client-side code execution for functionality. While web browsers provide built-in protections against arbitrary code execution, these protections assume that the web service fully controls the content it delivers to clients. However, if the web service depends on third-party resources or fails to implement proper safeguards, attackers can exploit this reliance to inject malicious code into client applications, web browsers. Because API services can also serve to end users who use web browsers, lack of communication regarding what data should be executed and what should not be, becomes important.

Recommendation: API services use underlying web protections and one of the protection mechanisms is Content Security Policy (CSP)[50]. CSP can be used to communicate what should be executed and what should not be. CSP can be used as strict as only allowing API service sourced content via "default-src 'self'" attribute. But this may not be enough for expected functionality. To have control, any external sources should be identified and added to the CSP. CSP also provides means to allow specific actions like loading fonts, rendering images, executing JavaScript codes, etc. A strict CSP can sometimes be hard to implement. But CSP also provides report-only mode additional to block mode. Via report-only mode, a stricter CSP can be tested and block mode CSP can be updated accordingly.

3.11. SD11 - Lack of Rate Limiting and Timeouts

Issue: API services have finite resources, including bandwidth, CPU, memory, and storage. Attackers can exploit this by flooding the service with requests to consume these resources[51,52], effectively launching a Denial of Service[53] (DoS) attack. Even legitimate requests can occasionally overload the system, reducing the quality of service for all users. Also, certain resource-intensive operations, such as complex computations or accessing large datasets are particularly appealing to attackers, because they can be exploited with little resources to create a big amount of impact. These kinds of operations also create data corruption risks, because cancelling the current operation and rolling back to initial state could be hard to achieve.

[50] OWASP ASVS Version 4.0.3 – Verification 14.4.3 https://github.com/OWASP/ASVS/raw/v4.0.3/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.3-en.pdf October 2021
[51] Common Weakness Enumeration - CWE-770: Allocation of Resources Without Limits or Throttling https://cwe.mitre.org/data/definitions/770.html November 2024
[52] OWASP Top 10 API Security Risks – 2019 – API4:2019 https://owasp.org/API-Security/editions/2019/en/0x11-t10/ December 2019
[53] Common Weakness Enumeration - CWE-400: Uncontrolled Resource Consumption https://cwe.mitre.org/data/definitions/400.html November 2024

Recommendation: Limits can be defined for the number of requests for each user or IP within a specific time frame. This prevents abuse by individual sources.

Service wide caps can be implemented for resource consumption, such as total bandwidth or memory usage, to prevent excessive strain on the API service, even if multiple users collectively cause overload.

Endpoint and object-based rate limits should also be implemented to protect against high resource consumption.

Pagination can be implemented to restrict the amount of data returned in a single request.

Maximum execution times should be set for all requests. If a request exceeds the timeout, it should be terminated gracefully and roll back actions should be done.

3.12. SD12 - Not Filtering Response Data

Issue: API services often respond to user requests with data, which must be strictly controlled. However, unexpected business flows, misconfigurations, or bugs can result in the API exposing unintended data, such as error logs, internal information, or sensitive details. This possible exposure increases the risk of attackers gaining system information or sensitive data without exploits.

Recommendation: Try to create and enforce strict data schemas for all responses.

Only include fields explicitly required by the business logic to lower risk of excessive data exposure. Implement data minimization principle when processing and sharing data.

Implement filtering[54] via whitelists based on predefined data schemas. Also, blacklists can be used to block or sanitize responses with unwanted data. Enhance the filtering feature for nested data types in order to not miss important fields.

3.13. SD13 - Lack of Compliance with Software Licenses

Issue: API services can rely on external libraries, frameworks, and components as dependencies. However, these dependencies come with specific licensing requirements[55,56] and ignoring them can lead to legal problems.

[54] NIST SP 800-53 Rev. 5 - SI-15 INFORMATION OUTPUT FILTERING https://doi.org/10.6028/NIST.SP.800-53r5 September 2020
[55] Common Weakness Enumeration - CWE-1177: Use of Prohibited Code https://cwe.mitre.org/data/definitions/1177.html November 2024
[56] OWASP SAMM – Secure Build - Software Dependencies https://owaspsamm.org/resources/pdf/ February 2020

Recommendation: Define acceptable license requirements and use dependency management tools to automatically scan and identify the licenses associated with all dependencies. Dependency management tools can go through all dependencies and compare their licenses with defined acceptable license attributes.

4. AUTHENTICATION AND AUTHORIZATION

Authentication and Authorization is not a real step within the development of an API service. Despite that, Authentication and Authorization is so crucial that they should be mentioned in their own chapter. Authentication related business logics, processes and activities manage who should access the API service and who should not. Also, any validation of the user being legitimate falls into this category. Similarly, Authorization related activities manage which users or groups of users can do what and cannot do what. Read, write, execute related activities falls into Authorization's interest.

In this chapter, following security issues will be addressed regarding to Authentication and Authorization.

1 Weak Password Policy

2 Lack of Brute Force Protection

3 Non-Secure Credential Storage

4 Non-Secure Session Data Creation

5 Lack of Endpoint/Object-Level Authorization

6 Mass Assignment

7 Excessive Privileges

8 Openly Accessible Administrative Resources

9 Not Requiring Extra Proofs for Critical Actions

4.1. AA01 - Weak Password Policy

Issue: API service may need to allow users to create their own passwords. In such cases, users with limited security awareness may create security risks by choosing non-complex, easy to guess passwords. These weak passwords can be exploited by attackers through methods such as brute force or credential stuffing, leading to unauthorized accesses and potential data breaches.

Recommendation: Require passwords to be at least 12 characters long to ensure basic complexity[57]. Allow passwords up to 64 characters long to accommodate stronger and more complex passwords. Allow users to include any valid characters including whitespace and Unicode characters.

Provide users ability to change their passwords in case of a leak or compromise. Warn users about the risk of using weak passwords and encourage them to use stronger passwords.

4.2. AA02 - Lack of Brute Force Protection

Issue: Even when strong password policies are enforced, a lack of brute force protection can expose API services to account compromise related attacks. There are several kinds of brute force attacks that can be used for compromise. Simple brute force attacks target a single account from a single attacking source. Distributed brute force attacks target a single account from multiple sources. Distributed brute force attacks can be further complicated by targeting multiple accounts.

Recommendation: Several protections need to be implemented at the same time to achieve a proper level of protection against brute force attacks[58]. Rate Limiting can be implemented to lower the number of authentications tried within a period. Account Lockout can be implemented to protect the targeted account from being compromised. Rate limiting and especially account lockouts can sometimes create a self-imposed denial of service attack. So, they need to be implemented with care.

Source based access filtering can be implemented for limiting the number of possible sources that can be used for a distributed brute force attack. Some adaptive controls like checking the last successful logged IP address or some fingerprint from the device that was used for last successful login can also be used to enhance the protection.

4.3. AA03 - Non-Secure Credential Storage

Issue: Storing user credentials insecurely creates highly impactful security risks for an API Service. After a data breach, an attacker can steal the users' credentials to further continue their attack against them. Also, accidental data leaks can happen which can expose users' credentials.

[57] OWASP ASVS Version 4.0.3 – Verification 2.1 Password Security https://github.com/OWASP/ASVS/raw/v4.0.3/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.3-en.pdf October 2021
[58] OWASP ASVS Version 4.0.3 – Verification 2.2.1 https://github.com/OWASP/ASVS/raw/v4.0.3/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.3-en.pdf October 2021

Recommendation: Use a specialized cryptographic algorithm just designed for password storage like pbkdf2_sha256 or argon2id. These algorithms provide one-way hashes with salts and resistance against offline cracking attacks[59].

4.4. AA04 - Non-Secure Session Data Creation

Issue: Using session data is important for maintaining user authentication between requests without requiring repeated logins. If attackers can guess or forge session data, they can impersonate legitimate users, or maybe even administrators.

Recommendation: There are several session management techniques, but all of them share some basic security requirements[60]. From these session management techniques, Session ID and JSON Web Tokens (JWT) are the two of the most used methods for session data creation. (KUMAR & TL, 2024) also provides good information regarding to the security implications of JWT.

In the Session ID method, the server generates a non-guessable random number as the Session ID and shares it with the user. This Session ID is then stored in the server's session database and validated each time a user sends a request.

To make secure Session IDs, Cryptographically Secure Pseudorandom Number Generators (CSPRNG) need to be used. Created Session ID needs to have at least 64 bits of entropy[61]. Any predictable user information (e.g., usernames, IP addresses) or other easily obtainable data should not be used as input for CSPRNG, only random entropy should be used. To provide easier session invalidation, expiration needs to be defined for Session ID.

JWTs are created through taking user information (e.g., username, user role, session expiration), hashing and signing it. User information and signature is then sent to the user as JWT. Unlike Session IDs, JWTs do not require the server to maintain session state in a storage, as validation is performed through checking the signature.

To provide a secure JWT implementation, strong keys (e.g., symmetric keys or asymmetric private keys) need to be used for signing and the key being used needs to be kept secret. Also, secure hashing algorithms like HMAC-SHA256 or RS256 need to be used for creation of the hash. Any sensitive user data should not

[59] OWASP ASVS Version 4.0.3 – Verification 2.4 Credential Storage https://github.com/OWASP/ASVS/raw/v4.0.3/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.3-en.pdf October 2021
[60] NIST SP 800-53 Rev. 5 – SC-23 SESSION AUTHENTICITY https://doi.org/10.6028/NIST.SP.800-53r5 September 2020
[61] OWASP ASVS Version 4.0.3 – Verification 3.2.2 https://github.com/OWASP/ASVS/raw/v4.0.3/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.3-en.pdf October 2021

be in JWT's payload unless that sensitive data is encrypted. Revocation for JWT can be challenging, using short lived keys and/or blacklist databases can be helpful.

For the general security of the session data, generated session data needs to be only shared through encrypted communication channels and new session data needs to be created after each login.

4.5. AA05 - Lack of Endpoint/Object-Level Authorization

Issue: Relying on generic read-only or read-write privilege levels is a simplistic approach that can fail to align with complex business objectives and requirements. If such a generic approach is implemented, users may mistakenly assume that it is alright for them to do some actions or data access, even though it is not[62,63]. Each endpoint or object in an API service may involve sensitive actions or critical data. This simplistic approach can lead to benign damages such as accidental data corruption or data leaks. Also, this situation makes the job of an insider attacker easier.

Recommendation: Incorporating granular, well-defined controls make API services maintain a robust security posture and prevent abuse. Provide means of control against individual actions and group data based on business requirements to implement access control against each data group. Using these controls is crucial for the one of the most important security principles, least privilege to be implemented.

4.6. AA06 - Mass Assignment

Issue: When an API service allows users to perform multiple actions and/or access to multiple pieces of data within a single request, because of some business need, implementing granular controls can be hard[64]. If proper controls are not implemented, an attacker with low privileges can trick the API service via sending a request including both valid and invalid actions and get successful results for invalid ones. This is a Mass Assignment attack, and the result of this attack can be impactful.

Recommendation: Try to avoid enabling features that can lead to mass assignments. If it is necessary to enable such features, try to restrict its usage based on possible actions and groups of data with very specific properties. Ensure that even when multiple objects are accessed or modified in a request, API service checks permissions for each individual object[65].

---

[62] OWASP Top 10 API Security Risks – 2019 – API1:2019 https://owasp.org/API-Security/editions/2019/en/0x11-t10/ December 2019
[63] OWASP Top 10 API Security Risks – 2023 – API1:2023 https://owasp.org/API-Security/editions/2023/en/0x11-t10/ December 2019
[64] OWASP Top 10 API Security Risks – 2019 – API6:2019 https://owasp.org/API-Security/editions/2019/en/0x11-t10/ December 2019
[65] OWASP - Mass Assignment Cheat Sheet https://cheatsheetseries.owasp.org/cheatsheets/Mass_Assignment_Cheat_Sheet.html March 2021

Implementing custom procedures for multiple actions can be also useful. Via custom procedures, expected multiple actions can be merged into a single request action and therefore any other single request multiple action business flows may be disabled.

4.7. AA07 - Excessive Privileges

Issue: API services are designed to meet user needs. However, when users encounter privilege-related limitations that prevent them from performing necessary tasks, they may request additional access rights. Granting such requests without careful consideration can lead to excessive privileges, which not only increase the risk of accidental misuse but also make it easier for insider threats to exploit the API service. Additionally, as users' roles or business needs evolve over time, outdated privileges can remain unchecked, making the problem bigger.

Recommendation: Implementing the principle of least privilege[66] can be challenging due to the complexity of API services and resource constraints. However, it is still possible to define privileges that align with a foundational level of least privilege.

Role-Based Access Control (RBAC) can simplify privilege management by grouping users with similar roles and assigning privileges to these roles, reducing the need to manage individual user permissions.

Attribute-Enhanced Role-Based Access Control (ARBAC) helps this approach by incorporating specific attributes (e.g., user IP address, last login time, account creation date, user or endpoint risk scores) to refine the access controls further. (Kuhn, Coyne, & Weil, 2010) share possible trade-offs between RBAC and Attribute-Based Access Control and their combinations.

To address changes in users' roles or business needs, a periodic re-evaluation process should be established to ensure privileges remain appropriate and up to date.

4.8. AA08 – Openly Accessible Administrative Resources

Issue: Managing the attack surface is a critical aspect of API security. Allowing generic users to access administrative interfaces, endpoints and similar resources greatly increases the risk of exploitation. If attackers can connect to these resources, they could exploit unresolved security vulnerabilities or possible zero-days, potentially compromising the API service’s functionality, data and security.

Recommendation: Avoid granting generic users access to administrative resources unless it is absolutely required for specific business purposes[67].

[66] NIST SP 800-53 Rev. 5 – AC-6 LEAST PRIVILEGE https://doi.org/10.6028/NIST.SP.800-53r5 September 2020

If possible, implement source-based access filtering, such as IP whitelisting or network segmentation to block unauthorized access to administrative endpoints.

4.9. AA09 – Not Requiring Extra Proofs for Critical Actions

Issue: Relying solely on session-based authentication and authorization, without requiring additional proof of validity for the request or triggering re-authentication, can expose users to significant security risks for critical actions. Attackers who gain access or partial access to an active session can perform malicious actions without needing the user's credentials. Session hijacking, session reuse and cross site request forgery (CSRF)[68] attacks utilize on this vulnerability. This vulnerability can lead to unauthorized changes in sensitive data, compromised accounts or breaches of confidential information.

Recommendation: Define critical actions that may require additional proof or re-authentication, like; password resets, authentication method changes, user role and privilege changes, high value business actions.

Against CRSF attacks, implement CSRF tokens[69]. These tokens are single-use random values that are provided by API service after each response and expected to be provided in the following important request. Security of the CSRF tokens rely on randomness of the token, so cryptographically secure pseudo-random number generators (CSPRNG) should be used when creating them. Also, CSRF tokens should be transmitted through HTTP Headers for additional security provided by HTTP protocol.

When critical actions are requested, re-authentication can be triggered[70] or even another factor (MFA) of authentication can be requested.

Configure session timeouts to create balance between security and usability. Having short lived sessions reduces the available attack window of time.

---

[67] NIST SP 800-53 Rev. 5 – SC-2 SEPARATION OF SYSTEM AND USER FUNCTIONALITY https://doi.org/10.6028/NIST.SP.800-53r5 September 2020
[68] Common Weakness Enumeration - CWE-352: Cross-Site Request Forgery (CSRF) https://cwe.mitre.org/data/definitions/352.html November 2024
[69] OWASP ASVS Version 4.0.3 – Verification 13.2.3 https://github.com/OWASP/ASVS/raw/v4.0.3/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.3-en.pdf October 2021
[70] OWASP ASVS Version 4.0.3 – Verification 2.1.6 https://github.com/OWASP/ASVS/raw/v4.0.3/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.3-en.pdf October 2021

5. IMPLEMENTATION AND MAINTENANCE

These steps include activities related to management of the infrastructure that would run the source code. The infrastructure would provide the environment for the API service to work. Initial setup of the environment, configurations of the environment, deployment of the code to the environment, ensuring that environment works as expected, management of any supportive feature required by API service and similar activities falls into these steps.

In this chapter, following security issues will be addressed regarding to these development process/lifecycle steps.

1 Using Non-Secure Communication Channels

2 Not Filtering Access Based on Source

3 Not Storing Secrets in Secure Storage(s)

4 Lack of Active Protection Tools

5 Lack of Reliable Backups

6 Not Retiring Old Data

7 Misconfigurations

8 Lack of Vulnerability and Patch Management Processes

9 Not Retiring Unused API Endpoints

10 Directory Listings

5.1. IM01 - Using Non-Secure Communication Channels

Issue: Network traffic, especially over the internet, is exposed to numerous attack vectors that can compromise data confidentiality and integrity.

Recommendation: Only offer encrypted communication interfaces to API consumers to enforce secure practices across all interactions. Always use encrypted communication channels

Use well-established encryption protocols and strong ciphers (e.g., TLS 1.3)[71] to ensure robust security. Avoid outdated or insecure technologies like TLS 1.0/1.1 or weak ciphers (e.g., MD5, SHA1, DES, 3DES).

5.2. IM02 - Not Filtering Access Based on Source

Issue: A large attack surface increases the likelihood of unauthorized access or malicious activity targeting the API service. Overexposing the API to unnecessary external sources can allow malicious actors, such as botnets or scanners, to probe the service for vulnerabilities.

Recommendation: API service's exposure to threats can be significantly reduced by filtering access[72] based on sources.

If API consumers' IP addresses are known, restrict access to only those addresses using an access control list. This minimizes exposure and reduces the attack surface.

Leverage threat intelligence feeds to identify and block IP addresses associated with known malicious activities, such as botnets, malware distributors, scanners, and anonymous proxies.

5.3. IM03 - Not Storing Secrets in Secure Storage(s)

Issue: Secrets, such as passwords, credentials, and encryption keys, are fundamental to ensuring data security, verifying identities, and enabling secure communication. If these secrets are improperly stored and accessed by an attacker, the API service relying on their protection can easily get compromised.

Recommendation: Secure storage of secrets require precise access controls[73]. Vaults are special tools that are designed to provide these controls. Vaults protect secrets by encrypting them when stored. Restrict and log access to secrets, ensuring only authorized components or users can retrieve them.

Also, for certain types of secrets, vaults can periodically rotate them to reduce the risk of long-term exposure.

Vaults provide strong security but are not foolproof. For example, if an attacker gains access to an application's memory, they can still extract secrets in use. So, proper risk assessment should be done for these cases.

[71] OWASP ASVS Version 4.0.3 – Verification 9.1 Client Communication Security https://github.com/OWASP/ASVS/raw/v4.0.3/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.3-en.pdf October 2021
[72] NIST SP 800-53 Rev. 5 – SC-7 BOUNDARY PROTECTION https://doi.org/10.6028/NIST.SP.800-53r5 September 2020
[73] OWASP SAMM – Secure Deployment - Secret Management https://owaspsamm.org/resources/pdf/ February 2020

5.4. IM04 - Lack of Active Protection Tools

Issue: Addressing vulnerabilities in software is not always immediate or straightforward. There could be several factors like; lack of available mitigations for newly discovered vulnerabilities, time needed to release or apply patches, testing requirements before implementing fixes. These delays provide attackers an opportunity to exploit vulnerabilities, especially as attackers actively monitor and target newly reported vulnerabilities.

Recommendation: Active protection tools act as a defensive layer[74] to mitigate risks when timely patching or mitigation isn't feasible. Web Application Firewalls (WAF) and Intrusion Prevention Systems (IPS) are such tools.

Using active protection tools can create a critical safety net to defend against emerging threats while working to implement long-term mitigations, enhancing the security of the API service. They provide protection against known attack patterns, even when a patch is not available/applied. They also provide defence against common attack vectors, such as brute force, without needing any exploit signature updates. However, these tools are not foolproof. Skilled attackers may find ways to bypass these tools.

5.5. IM05 - Lack of Reliable Backups

Issue: Achieving 100% uptime is virtually impossible, and unexpected downtime can lead to significant issues. Loss of application data, application code, or infrastructure configurations can happen in such downtimes. These can lead to transitioning into an unstable state when the API service is restored. Also, permanent loss of business-critical information, negatively impacting operations and reputation.

Recommendation: Create a backup policy via defining business-specific Key Performance Indicators (KPIs) for downtime and data loss[75]. These KPIs should include Maximum Tolerable Downtime (MTD), maximum acceptable data loss (also known as Recovery Point Objective or RPO) and Recovery Time Objective (RTO)[76].

Backups should at least be taken for Application Data, Logs, Application Source Code and Infrastructure Components. Determine how often backups should be taken based on business needs and have a retention policy to delete old backups. Test backups at least annually to ensure they work as expected and that data can be restored successfully.

---

[74] NIST SP 800-53 Rev. 5 – PL-8(1) DEFENSE IN DEPTH https://doi.org/10.6028/NIST.SP.800-53r5 September 2020
[75] NIST SP 800-53 Rev. 5 – CP-9 SYSTEM BACKUP https://doi.org/10.6028/NIST.SP.800-53r5 September 2020
[76] NIST SP 800-34 Rev. 1 - 3.2.1 Determine Business Processes and Recovery Criticality https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-34r1.pdf May 2010

5.6. IM06 - Not Retiring Old Data

Issue: All data, even if unused, carries legal, regulatory, and operational responsibilities. Storing old, unused data not only creates storage problems but data breaches involving outdated information can still have severe consequences.

Recommendation: Establish clear data retention policies[77] based on legal, regulatory, business, and operational needs. Take inventory of all stored data and analyse its purpose and relevance. Classify data based on usage, sensitivity, and retention requirements (e.g., business-critical, compliance-related, archival).

Identify data that is no longer required for business, legal, or compliance reasons. Use secure deletion methods, such as crypto shredding to get rid of the old data.

5.7. IM07 - Misconfigurations

Issue: Managing infrastructures is not always easy. Both complexity of the infrastructure and changes within the infrastructure can bring security risks. Taking account of time pressure and human error, the security risks get bigger. Infrastructure affects an API service's behaviour deeply and any unwanted change can create new vulnerabilities on the API service.

Recommendation: Before applying configuration changes to the production environment, test them in a non-production environment to identify and address issues. Testing changes can minimize the risks associated with misconfiguration while maintaining a secure and reliable infrastructure for the API service.

If it is possible, try to implement infrastructure as code (IaC) practices. IaC brings tracking for changes, easier rollbacks and consistency while updating configurations. Use hardening guidelines provided by each tool's vendor and/or use security best practices provided by institute like CIS[78].

5.8. IM08 - Lack of Vulnerability and Patch Management Processes

Issue: Most of the software development projects rely on external dependencies such as libraries, packages, operating systems, and firmwares. Any security flaw in a dependency can directly impact the API service's security.

Recommendation: Implement vulnerability and patch management processes[79]. Create a detailed inventory of all components/dependencies (libraries, frameworks, packages, operating systems, firmwares, etc.) in the API service. Scan all dependencies for known vulnerabilities.

[77] NIST SP 800-53 Rev. 5 – SI-12 INFORMATION MANAGEMENT AND RETENTION https://doi.org/10.6028/NIST.SP.800-53r5 September 2020
[78] CIS Benchmarks List https://www.cisecurity.org/cis-benchmarks December 2024

Prioritize vulnerabilities based on their severity, potential impact, and likelihood of exploitation and deploy patches accordingly.

5.9. IM09 - Not Retiring Unused API Endpoints

Issue: Every API service, version, or endpoint requires maintenance, which translates to ongoing costs and liabilities. Failure to properly maintain them introduces security risks. Unused or forgotten endpoints can become entry points for attackers due to outdated security measures or unpatched issues[80,81]. Attackers often exploit such "low-hanging fruits", as these are more likely to have unattended vulnerabilities. Each unretired API endpoint or service increases the system's overall attack surface, introducing unnecessary security risks.

Recommendation: By retiring unused APIs, attack surface can be minimized, liabilities can be reduced, and use of resources can be optimized for maintaining active and critical components.

Periodically review all API endpoints and versions to identify those no longer in use. Safely decommission unused APIs and ensure that they are no longer accessible. Reflect these changes in your documentation and inform users to prevent confusion or unintended usage.

5.10. IM10 - Directory Listings

Issue: API services often interact with files and directories on the operating system to fulfil functional or data-related requirements. An attacker can trick an API service to list the files within directories that the API service has access to and gather important information in their reconnaissance.

Recommendation: Limit API service's access to only necessary directories and files. Remove or relocate files that are not required for the API service's operation[82].

Web framework that API service is using may have the ability to disable directory listing which can be used to protect against this attack.

---

[79] OWASP SAMM – Environment Management - Patching & Updating https://owaspsamm.org/resources/pdf/ February 2020
[80] OWASP Top 10 API Security Risks – 2019 – API9:2019 https://owasp.org/API-Security/editions/2019/en/0x11-t10/ December 2019
[81] OWASP Top 10 API Security Risks – 2023 – API9:2023 https://owasp.org/API-Security/editions/2023/en/0x11-t10/ December 2019
[82] OWASP ASVS Version 4.0.3 – Verification 4.3.2 https://github.com/OWASP/ASVS/raw/v4.0.3/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.3-en.pdf October 2021

6. TESTING AND VALIDATION

These steps cover the activities that are required for validating the API service works as expected. Activities like checking business logic of the API service works as expectedly, infrastructure provides what it should be and making sure that there are no unexpected negative qualities present, are tested and validated in these steps.

In this chapter, following security issues will be addressed regarding to these development process/lifecycle steps.

1 Lack of Testing for Authentication and Authorization

2 Not Conducting Penetration Test

3 Lack of SAST and DAST Usage

4 Lack of Negative Testing

6.1. TV01 - Lack of Testing for Authentication and Authorization

Issue: Authentication and authorization mechanisms are fundamental for API service's security. However, if these features are not thoroughly tested, vulnerabilities such as missing authentication checks or misconfigured access controls can arise. These give opportunity to attackers to access restricted resources or perform unauthorized actions, compromising the API service's security.

Recommendation: Integrate authentication and authorization checks into testing[83,84] steps.

Verify that all API endpoints require proper authentication.

Ensure that expired or invalid session data cannot be used to access resources.

Tests actions that require CSRF tokens or re-authentication.

Create test users for each role and verify that their access aligns with their assigned privileges. Do positive testing via verifying that legitimate test users can access resources that they are authorized for. Also implement negative testing via checking that users cannot access what they should not.

To avoid human errors in tests, implement automation for authentication and authorization tests.

[83] OWASP WSTG - 4.4 Authentication Testing https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Security_Testing/04-Authentication_Testing/README.html December 2024

[84] OWASP WSTG - 4.5 Authorization Testing https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Security_Testing/04-Authentication_Testing/README.html December 2024

6.2. TV02 - Not Conducting Penetration Test

Issue: Security is a complex topic. Even if most advanced security scanning tools are being used, some security issues still can be missed. Because these tools cannot comprehend the context of each API service, it is impossible for them to find every business logic related vulnerability. Also, sometimes developers take shortcuts to implement some functionality, but these may result in weaknesses within the API service. Attackers can find these weaknesses and exploit them. Relying solely on security tools would be a mistake for such situations.

Recommendation: Add periodic penetration testing[85,86] to the API service's development process/lifecycle. Because penetration testing requires manual effort, it may not be possible to do it in each development step/iteration. Based on API service's security risks, threats choose the penetration testing frequency. Also, before going live with big changes, arrange penetration tests.

Quality of penetration testers who will be doing the penetration testing is important. With inexperienced and low skilled people, output of a penetration test would be almost no different than a security scanning tool. Skilled and experienced people create a huge difference to finding weaknesses within the API service.

Also, penetration tests can be categorized into two; black-box testing and white-box testing. In white-box testing, penetration testers would have access to source code of the API service and have information about the infrastructure that it runs. This helps penetration testers to see problems easier, but this approach has the disadvantage of focusing on what they see and having a bias on vulnerabilities. This can lead to missing other vulnerabilities within the API service. Additionally, having access to large amounts of data makes penetration testing effort bigger. In black-box testing, penetration testers would not have any access to source code of the API service and information about the infrastructure that runs it. This would make penetration testers try many things to understand the API service and find weaknesses with it. Black-box testing has the advantage of taking less effort. But it has a con of potentially missing basic vulnerabilities that can be found easily by going through the source code.

Both white-box and black-box approaches bring value, and doing both at the same time can be possible by using different people or doing black-box first and doing white-box next.

After a penetration test, a report is provided by the penetration testers about the vulnerabilities within the API service. This creates the opportunity of fixing the vulnerabilities before they are exploited by an

[85] The PTES Team - The Penetration Testing Execution Standard Documentation Release 1.1 https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Security_Testing/04-Authentication_Testing/README.html April 2022
[86] OWASP WSTG - Penetration Testing Methodologies https://owasp.org/www-project-web-security-testing-guide/latest/3-The_OWASP_Testing_Framework/1-Penetration_Testing_Methodologies December 2024

attacker. But implemented fixes may not be accurate and should be checked. Therefore, another smaller penetration test should be triggered to verify the fixes.

6.3. TV03 - Lack of SAST and DAST Usage

Issue: Developers may make mistakes. A wrong type definition, a misused function and many more minor mistakes create their own impact on the security of the API service. Even if developers have taken security training and are trying to align with the security best practices, some things may still be overlooked. Including the possible complexity and sheer amount of code within the codebase, this possibility gets bigger. Not using any tools to find these potential mistakes would be like allowing a ticking bomb to be in your backyard. Attackers with the right tools can find these mistakes easily and exploit them.

Recommendation: There are many tools that can be used to check source code or running instances of an application that has security issues. These tools are divided into two classes: Static Application Security Testing (SAST) tools and Dynamic Application Security Testing (DAST) tools[87].

SAST tools work on source code without requiring running it. They search for known unwanted software development practices like not using prepared statements to call database connections, using unsafe functions and so on. They provide alerts about these unwanted practices and share information to avoid them. Also, there are similar tools that work on the source code but provide different outputs regarding to security. Such tools are code quality tools, secret scanners and software composition analysis (SCA) tools. Code quality tools investigate the code and provide recommendations based on best practices for coding. Secret scanners look for any secrets/credentials that should not be in the source code and provide alerts if there are some. SCA tools inform about vulnerabilities in the used libraries and other similar dependencies. Additionally, there are SAST tools that can be integrated into developer's integrated development environments (IDE) which can scan the code while the developer is writing it. Main disadvantages of SAST tools are false positive alerts and missing logic-based issues.

DAST tools work on the running code and checks it against known attacks and attack techniques. Because DAST tools are working on the running code, they can find the security issues with more accuracy and with less false positives. But it is almost impossible for DAST tools to go through every function or logic, which creates the disadvantage of missing vulnerabilities in not tested functions and logics.

6.4. TV04 - Lack of Negative Testing

Issue: Conducting acceptance tests in the software development processes/lifecycles is a general best practice. Acceptance tests are done against the defined requirements, and software is expected to provide

[87] OWASP SAMM – Secure Build - Security Practice Overview https://owaspsamm.org/resources/pdf/ February 2020

what it should. But this test doesn't cover unexpected cases, despite attackers forcing applications into unexpected business flows to exploit them.

Recommendation: Implement negative testing[88] in the testing phase. Use test cases defined for user acceptance tests and change them to include unwanted or unexpected inputs. Look into the API service's behaviour against these negative tests and check if API service provides right behaviour like blocking the input, giving errors, abandoning the flow and not transitioning into unstable states. If not, fix the issues. Also, outputs of the negative test can be used for updating abuse cases.

There are tools called fuzzers that can be used for automated negative testing. With fuzzers, random inputs are fed to the application and the application's behaviour is tracked. If unexpected behaviour is found, developers can investigate it to fix it. Because walking the whole input space is almost an impossible task, some example inputs can be fed to the fuzzer. The fuzzer can change/mutate this input based on most common fuzzing practices or through application's behavioural changes and make the fuzzing step more efficient. Some fuzzers can even analyse the source code to try to go through all the functions by mutating the inputs.

[88] OWASP SAMM – Requirements-driven Testing - Security Practice Overview
https://owaspsamm.org/resources/pdf/ February 2020

7. MONITORING AND RESPONSE

This step covers all the activities that are used for tracking the status of the API service and its details. Logs, performance metrics, errors and similar outputs are analysed in this step to be aware of the API service's current and previous status. Knowing this information and not doing anything accordingly would be meaningless. So, responding to any important event also falls into this step.

In this chapter, following security issues will be addressed regarding to these development process/lifecycle steps.

1 Not Monitoring Performance

2 Not Logging Business Critical Events

3 Not Logging Actions and Data Accesses

4 Not Logging Errors

5 Excessive Logging

6 Not Monitoring Logs and Alerts

7 Not Having an Incident Response Process

7.1. MR01 - Not Monitoring Performance

Issue: Not monitoring performance metrics of an API service makes it difficult to detect issues in a timely manner, whether caused by natural usage growth, bugs, or deliberate attacks. Performance metrics are essential for understanding the "normal" operational profile of an API service and creating baselines. Without such baselines, detecting deviations from normal operation becomes hard and mostly late. Additionally, an attack's unexpected effects on the API service may not be sensed in a timely manner.

Recommendation: Define important performance metrics. By using these metrics, gather data about the API service. Analyse historical data to define what is "normal" and create a baseline[89]. Define thresholds for the metrics and set up alerts for anomalies. Periodically go through the baselines and update them in case of possible changes within the API service.

[89] MITRE D3FEND - Resource Access Pattern Analysis
https://d3fend.mitre.org/technique/d3f:ResourceAccessPatternAnalysis/ December 2024

7.2. MR02 - Not Logging Business Critical Events

Issue: API services are expected to deliver a reliable service. When deviations from the intended behaviour occur, it would be difficult to identify and address the issue without sufficient logging. Such unintended behaviour can happen because of errors, misconfigurations, or malicious actions. Lack of logging of business-critical events make it hard to troubleshoot, investigate incidents, and detect attacks or unauthorized activities.

Recommendation: Map out the business flow of the API service and mark steps where important actions, decisions, or transactions occur[90]. Enrich the logs to make troubleshooting easier, but do not include any confidential data into logs. Create alert mechanisms to know about unwanted or unexpected behaviour.

7.3. MR03 - Not Logging Actions and Data Accesses

Issue: Authentication and authorization mechanisms are fundamental to securing API services. Misconfigurations, bypasses, or vulnerabilities in these controls can result in unauthorized actions and/or data breaches. Without comprehensive logging of actions and data accesses, it becomes difficult to detect, investigate or respond to malicious activities or unwanted issues effectively.

Recommendation: Log all critical actions and data accesses[91]. Current authorization rules can be used to choose which actions and data accesses to be logged. Like other logging actions, put important details into logs without any confidential data. Creating alerts for unexpected actions and data accesses is also crucial.

7.4. MR04 - Not Logging Errors

Issue: Encountering errors is an expected part of providing an API service. But their origins can vary significantly, from benign issues to malicious activities. Failing to log errors properly can result in missed opportunities to detect, diagnose, and respond to security incidents or operational problems. Furthermore, unlogged errors make debugging and identifying root causes of issues more difficult.

Recommendation: Log the errors and try to define unexpected states of the API service to be further informed. Putting assertions into business flows can give the benefit of defining data/object based expected situations. With assertions, any unexpected situation would trigger an error and can be more easily tracked.

Error logs and traces may include confidential data because of the unexpected state of the API service when the error gets caught. Make sure that only appropriate people have access to error logs and traces.

---

[90] OWASP ASVS Version 4.0.3 – Verification 11.1.7
https://github.com/OWASP/ASVS/raw/v4.0.3/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.3-en.pdf October 2021

[91] NIST SP 800-53 Rev. 5 – AU-3 CONTENT OF AUDIT RECORDS
https://doi.org/10.6028/NIST.SP.800-53r5 September 2020

7.5. MR05 - Excessive Logging

Issue: Logs are crucial for investigating security incidents. But sometimes bad habits like trying to log everything can arise. This may look better when lacking important information to do an investigation and not having that information[92]. But, logging excessive amounts of events requires excessive amounts of resources, which most of the time is not feasible. Also, logging everything does not proportionally bring the same amount of visibility. Because of the vast amounts of logs, focusing on the important events would be harder and losing time on looking into unimportant events would be more probable. Additionally, logged events may not need to be relevant about the important business and security risks. In such cases, malicious events done by an attacker may be missed and a probable response may not be done in the right time.

Recommendation: Start implementing logging rules based on regulations and laws that apply to the API service.

Categorize events based on their importance and not log unimportant events. When inspecting logs, focus on important logs and then continue with less important ones. While inspecting logs, mark unimportant ones that were missed in the first place and prune them from being logged in the logging source.

Using abuse cases and known intrusion techniques create alerts. Such alerts can be used for focusing on the logging right events.

Define a retention policy for logs and delete old logs based on this policy.

7.6. MR06 - Not Monitoring Logs and Alerts

Issue: Logs and alerts are essential for detecting, investigating, and responding to security incidents. But if no one investigates these logs and alerts, having them would be meaningless. In such cases, responding to security incidents would be almost impossible. Even some attackers exploit this issue by disabling logging or overwhelming alert systems, making it difficult to detect and respond to intrusions[93].

Recommendation: Define responsibilities within the API service team for monitoring logs. Real time log tracking may be best for security, but it may not be possible because of limited human resources. At least periodically go through logs based on possible risks and their impact.

Create alerts to be informed when unwanted things happen. Most security incident and event management (SIEM) tools provide predefined alerts for security incidents.

[92] NIST SP 800-92 Guide to Computer Security Log Management – 5.1.1 - Log Generation https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-92.pdf September 2006
[93] OWASP Top 10 API Security Risks – 2019 – API10:2019 https://owasp.org/API-Security/editions/2019/en/0x11-t10/ December 2019

Forward created logs into central logging systems. This way an attacker needs to also seize control of the central logging system to delete logs regarding the prior attacks. Creating alerts regarding the lack of logs from a log source is also a best practice against an attacker that stops logging services.

7.7. MR07 - Not Having an Incident Response Process

Issue: Security incidents are almost inevitable. Thinking that, no one would bother to attack the API service, or no one would get past the security controls within the API service would be dangerous. If such a security incident happens and no one knows how to respond to the incident, the incident can escalate quickly into something worse.

Recommendation: Define an incident response process and implement it[94]. This process should include possible cases of being informed about an incident, validation of the incident, informing relevant people about the incident, measuring the initial possible impact of the incident, containment of the attack, stopping the attack and fixing the vulnerability, gathering forensic evidence of what has attacker done, recovering from the effects of the attack and a lesson learned phase. Establish the roles and responsibilities of the people for the incident response and train them. Also, document the incident response process to have the capability to re-evaluate and improve it periodically.

[94] NIST SP 800-53 Rev. 5 – 3.8 INCIDENT RESPONSE https://doi.org/10.6028/NIST.SP.800-53r5 September 2020

8. CONCLUSION

API security requires a proactive approach against potential risks. Reactive measures are akin to conceding defeat and merely working to limit further losses. Therefore, adopting a proactive mindset is essential for developing secure APIs and mitigating security risks. However, it is not always possible to mitigate every risk and expect insignificant impact after these risks are realised, even with proactive approaches. This is because not every risk has a straightforward mitigation. Trade-offs between security, business goals and resources may be required.

Right decision making is key to managing such situations. With right decision making, business goals, resources and risks can be assessed to find a plausible solution. However, right decision making requires a well-defined problem statement, accurate information of the current state, an acceptable target state, available resources and applicable methods that can be utilized to solve the problem. Excluding the resources, realizing the right decision mostly about information and strategic thinking. But we cannot really exclude resources, because we are living in the real world. Therefore, another complexity comes up, is the risk significant enough to justify the resources needed to mitigate it. That creates another decision point.

Since every API service project is unique, it is not possible to create a one-size-fits-all solution for every API project's security needs. As a result, this guideline avoids detailing specific risks and their corresponding mitigations. Instead, it focuses on widely seen and generalized security issues and recommendations. Where possible, this guideline provides multiple alternate solutions to implement as recommendations, helping developers and product owners make informed choices in case of limited resources. This guideline also points out possible security problems and how stuff should be in order to help for better decision making. Additionally, this guideline focuses on lack of visibility and information gathering as an issue and provides recommendations. Throughout this guideline proactive solutions are prioritized and shared.

This report organizes the guideline items based on the specific steps of the development process/lifecycle in which they arise. The goal is to provide developers and product owners with a structured "mind model" that facilitates the seamless integration of security practices into their workflows.

REFERENCES

Bhuiyan, T., Begum, A., Rahman, s., & Hadid, I. (2018, March). API vulnerabilities: Current status and dependencies. *International Journal of Engineering and Technology(UAE), 7*, 9–13. doi:10.14419/ijet.v7i2.3.9957

Dahiya, S., & Ranjan, P. (2021, December). Optimizing API Security in FinTech Through Genetic Algorithm based Machine Learning Model. *International Journal of Computer Network and Information Security, 13*, 24.

Díaz-Rojas, J., Alejandro, Ocharán-Hernández, J. O., Pérez-Arriaga, J. C., & Limón, X. (2021). Web API Security Vulnerabilities and Mitigation Mechanisms: A Systematic Mapping Study. *2021 9th International Conference in Software Engineering Research and Innovation (CONISOFT)*, (pp. 207-218). doi:10.1109/CONISOFT52520.2021.00036

Hussain, F., Hussain, R., Noye, B., & Sharieh, S. (2020). Enterprise API Security and GDPR Compliance: Design and Implementation Perspective. *IT Professional, 22*, 81–89. doi:10.1109/MITP.2020.2973852

Kuhn, D., Coyne, E., & Weil, T. (2010, June). Adding Attributes to Role-Based Access Control. *Computer, 43*, 79–81. doi:10.1109/MC.2010.155

KUMAR, A., & TL, D. (2024, September). Security measures implemented in RESTful API Development. *Open Access Research Journal of Engineering and Technology, 7*, 105–112. doi:10.53022/oarjet.2024.7.1.0042

Munsch, A. P., & Munsch, P. M. (2021). The Future of API (Application Programming Interface) Security: The Adoption of APIs for Digital Communications and the Implications for Cyber Security Vulnerabilities. *Journal of International Technology and Information Management: Vol. 29: Iss. 3, Article 2.*

Zhao, C. (2024, November). API Common Security Threats and Security Protection Strategies. *Frontiers in Computing and Intelligent Systems, 10*, 29–33. doi:10.54097/k5djs164