AKS IT SERVICES

Web Service Security Audit Report:

Pan Demat Link Service

Report Release Date	20-06-2024
Type of Audit	Application Security
Type of Audit Report	Initial Audit Report
Period	14-06-2024 to 20-06-2024

Report Prepared By:

AKS Information Technology Services Pvt. Ltd.

www.aksitservices.co.in

E-Mail: info@aksitservices.co.in

NON-DISCLOSURE STATEMENT

This report is the sole property of NSDL. All information obtained during the assessment is deemed privileged information and not for public dissemination. **AKS Information Technology Services Pvt. Ltd**. pledges its commitment that this information will remain strictly confidential. It will not be discussed or disclosed to any third party without the express written consent of NSDL. except required by the government regulator (Cert-In) or by the order of the Court.

Document Control

Document Preparation	
Document Title	Web Service Security Audit Report Ver 1.0 Pan Demat Link Service
Document ID	-
Document Version	1.0
Prepared by	Ms. Vibhuti Bhatt
Reviewed by	Mr. Vinayak Kshirasagar
Approved by	Ms. Pallavi Roy
Released by	Ms. Vibhuti Bhatt
Release date	20-06-2024

Document Change History		
Version	**Date**	**Remarks / Reason of change**
1.0	20-06-2024	Initial release

Document Distribution List			
Name	**Organization**	**Designation**	**Email Id**
Durgesh Varma	NSDL	Assistant Manager	durgeshv@nsdl.com
Tejashree Thakare	NSDL	Assistant Manager-IT Security	tejashreet@nsdl.com

Table of Contents

Introduction

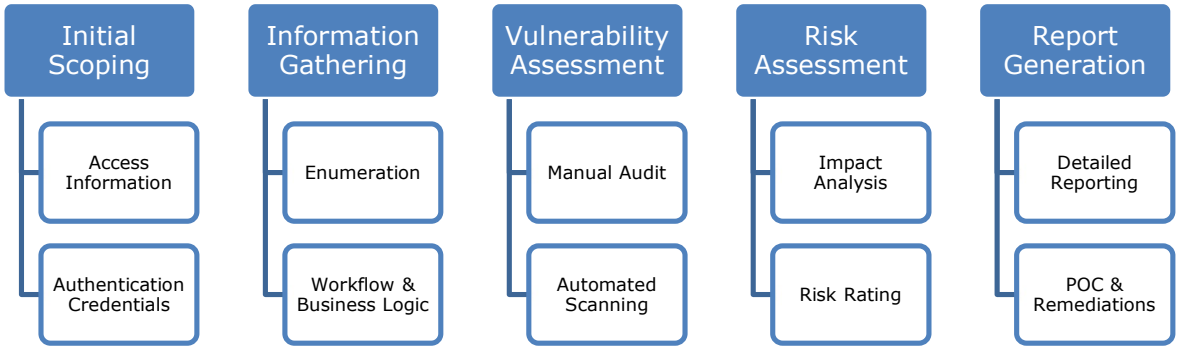
Objectives:

The key objective of this Web Application Security Audit was to identify whether any vulnerabilities exist in the Web Application and to exploit those that can be seen and compromised by malicious users. Additionally, the objective of this activity was to ensure the security of the network and web server from external threats through the Web Application.

Methodology & Standard:

Security Consultants at AKS IT Services Pvt. Ltd. Used the OWASP Web Application Security Testing Methodology for conducting the security audit of the in-scope Web Application.

The OWASP Web Application Methodology is based on the 'grey box' approach. The testing model consists of the following phases:



Standard:

The Open Worldwide Application Security Project (OWASP) standard was used for conducting the final level security audit of the Pan Demat Link Service web application. The assessment was aimed at identifying the vulnerabilities that are defined in the OWASP, SANS, Common Weakness Enumeration, and other common global best practices.

Appendix A: Details of the OWASP Top 10:2023 Standard

Appendix B: SANS TOP 25 Most Dangerous Software Errors

Engagement Scope

S. No	Asset Description	Criticality of Asset	Internal IP Address	URL	Public IP Address	Location	Hash Value of final audit report	Version
1	NA	NA	NA	https:// eservices-test.nsdl.com/pandematlinkservice/PanDematLinkService	NA	Remote	NA	NA

Details of the Auditing team

S. No	Name	Designation	Email Id	Professional Qualifications/ Certifications	Whether the resource has been listed in the Snapshot information published on CERT-In's website (Yes/No)
1	Vibhuti Bhatt	Infosec Consultant-L1	vibhuti.bhatt@aksitservices.co.in	CEH	Yes
2	Vinayak Kshirasagar	Team Lead – Application Security	vinayak.kshirasagar@aksitservices.co.in	CEH , CAP , eWPTX	Yes
3	Pallavi Roy	Assistant Manager – Application Security	pallavi.roy@aksitservices.co.in	CEH, ISO 27001 LA, ISO 27701 PIMS LI	Yes

Audit Activities and Timelines

Audit Activity	**Timelines**
Phase I	
Auditor Assigned	16-01-2024
Audit Initiated	14-06-2024
Audit Report Preparation	20-06-2024
Initial Audit Report Published	20-06-2024

Tools/ Software Used

S. No	Name of Tool/Software used	Version of the tool/Software used	Open Source/Licensed
1	Burp Suite Professional	2023.10	Commercial
2	Soap UI 5.2.1	11.1.0	Commercial

Appendix C: Description of the tools

Executive Summary

Total **High** **Medium** **Low**

S. No	Affected Asset i.e. IP/URL/Application etc.	Observation/ Vulnerability title	CVE/ CWE	Severity	Recommendation	Reference	New or Repeat or closed observation
1	https://eservices-test.nsdl.com/pandematlinkservice/PanDematLinkService	Lack of security headers	CWE-644	Low	Implement security headers such as X-XSS-Protection, Content-Security-Policy, Referrer Policy, X-Content-Type-Options, Permission Policy, Strict-Transport-Security Header (HSTS) and X-Frame-Options.	Case I	New
2	https://eservices-test.nsdl.com/pandematlinkservice/PanDematLinkService	Improper Input Validation	CWE-20	Low	All only the required characters for all the parameters as per requirement by whitelisting of those fields with not allowing all special and meta characters. Data type validators available natively in web application frameworks (such as Django Validators, Apache Commons Validators etc.). Minimum and maximum value range check for numerical parameters and dates, minimum and maximum length check for strings. Array of allowed values for small sets of string parameters (e.g., days of week).	Case II	New

Detailed Findings

Case I

i. **Affected Asset i.e. IP/URL/Application etc.**

https://eservices-test.nsdl.com/pandematlinkservice/PanDematLinkService

ii. **Observation/ Vulnerability title**

Lack of security Headers

iii. **Detailed observation / Vulnerable point**

Applications can unintentionally leak information about their configuration, internal workings, or violate privacy through a variety of application problems. Attackers use this weakness to steal sensitive data or conduct more serious attacks.

iv. **CVE/CWE**

644

v. **Control Objective #**

NA

vi. **Control Name #**

NA

vii. **Audit Requirement #**

NA

viii. **Severity**

Low

ix. **Recommendation**

Implement security headers such as X-XSS-Protection, Content-Security-Policy, Referrer Policy, X-Content-Type-Options, Permission Policy and Strict-transport-layer-protection.

x. Reference

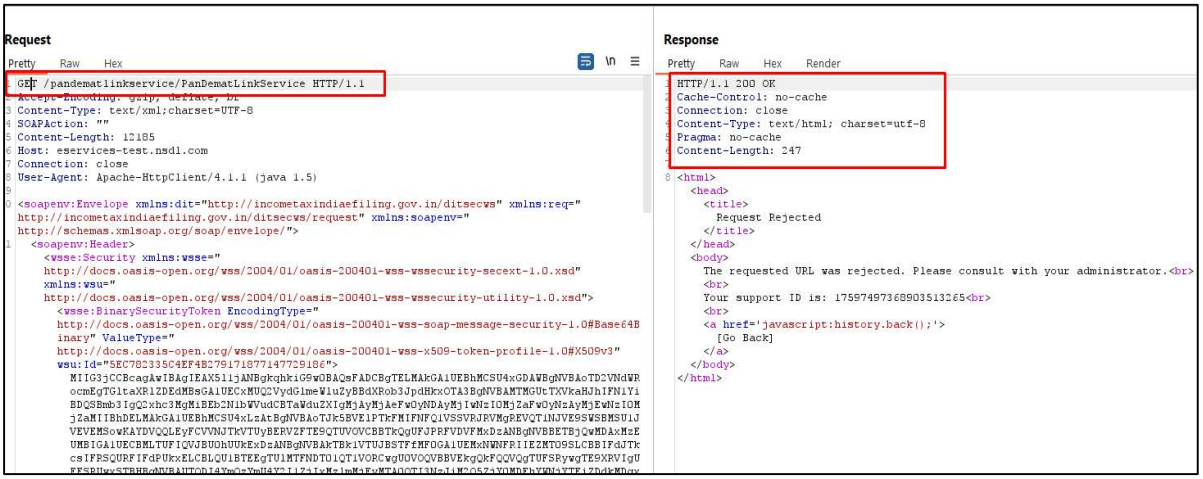
Case I

xi. New or Repeat observation

New

xii. References to evidence / Proof of Concept

Step I: Run the api and capture it on proxy tool. Change the method to "GET" and see the 200 OK response. Observe that security headers are not implemented here.



Case II

i. Affected Asset i.e. IP/URL/Application etc.

https:// eservices-test.nsdl.com/pandematlinkservice/PanDematLinkService

ii. Observation/ Vulnerability title

Improper Input Validation

iii. Detailed observation / Vulnerable point

It is observed that the application does not validates user inputs, this can affect the control flow or data flow of a program.

iv. CVE/CWE

CWE-20

v. Control Objective #

NA

vi. Control Name #

NA

vii. **Audit Requirement #**

NA

viii. Severity

Low

ix. Recommendation

All only the required characters for all the parameters as per requirement by whitelisting of those fields with not allowing all special and meta characters. Data type validators available natively in web application frameworks (such as Django Validators, Apache Commons Validators etc.). Minimum and maximum value range check for numerical parameters and dates, minimum and maximum length check for strings. Array of allowed values for small sets of string parameters (e.g., days of week).

x. Reference

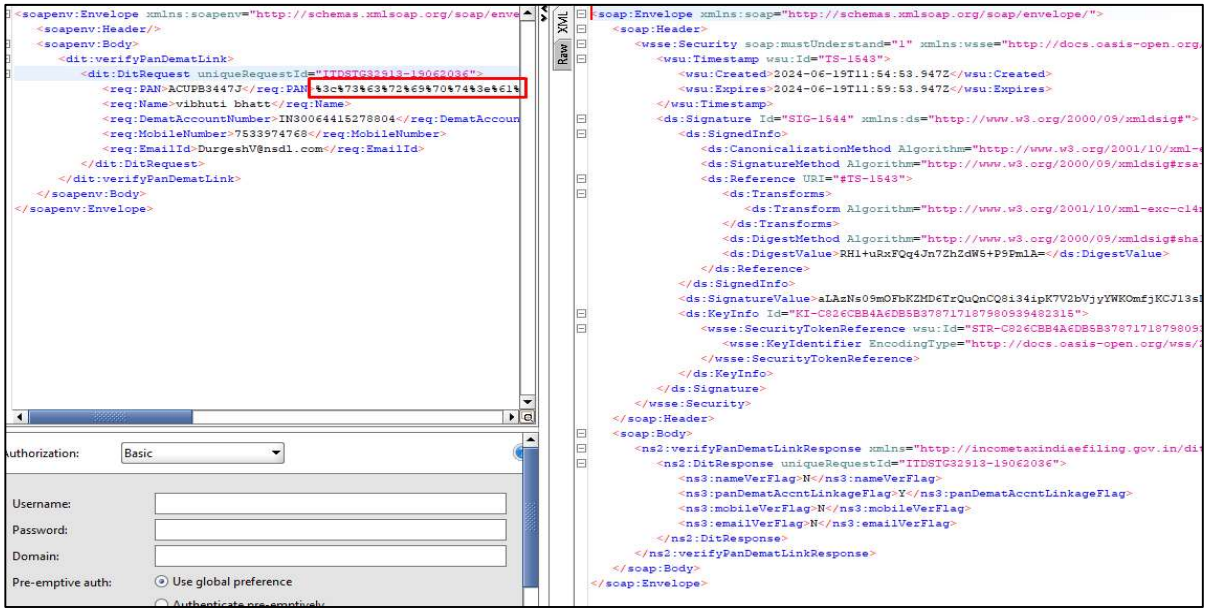
Case II

xi. New or Repeat observation

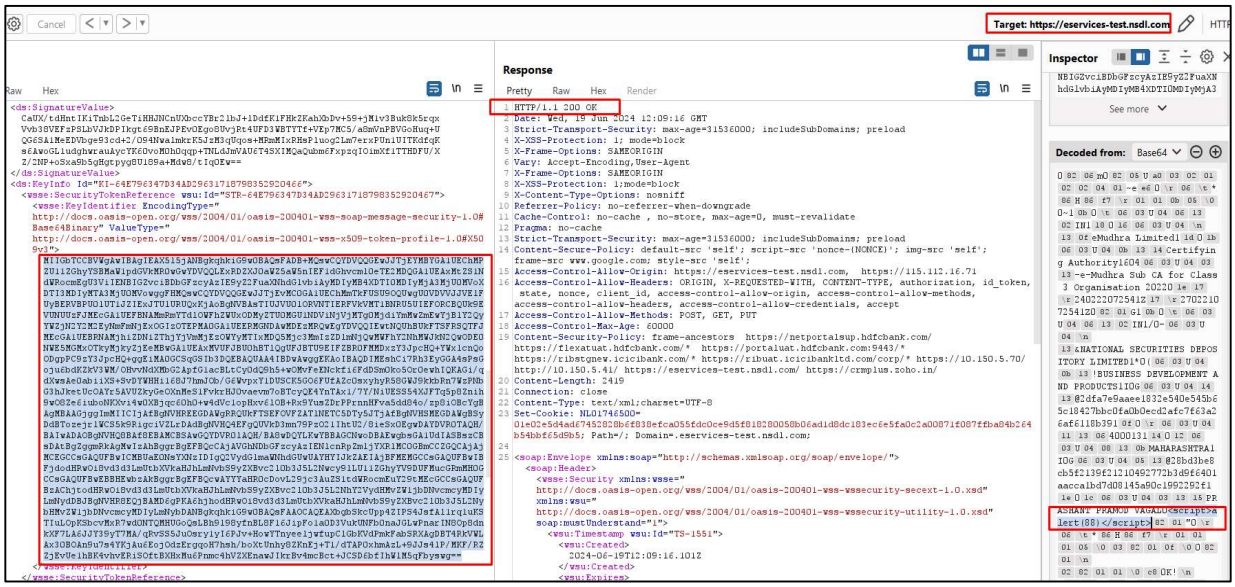
New

xii. References to evidence / Proof of concept

Step I: Enter malicious javascript code in url encoded format and run the api.Observe that the application accepts as given screenshot below.



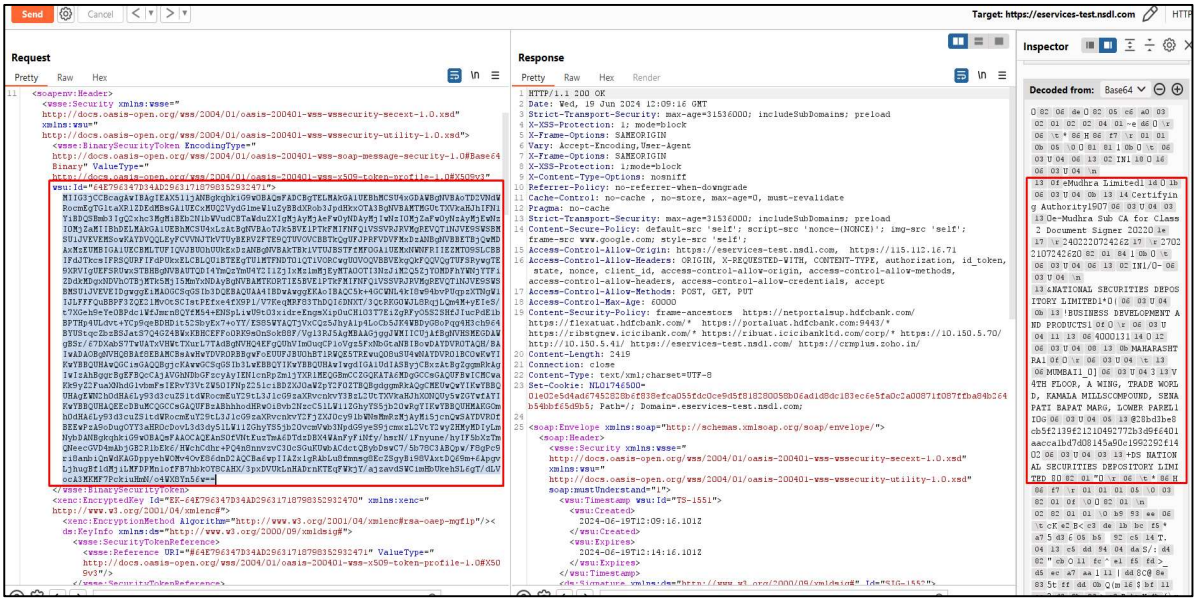
Case II: Observe that the data is Base64 encoded .Enter malicious javascript code in and send the request. Observe that application accepts the data and we receive a 200 OK response.

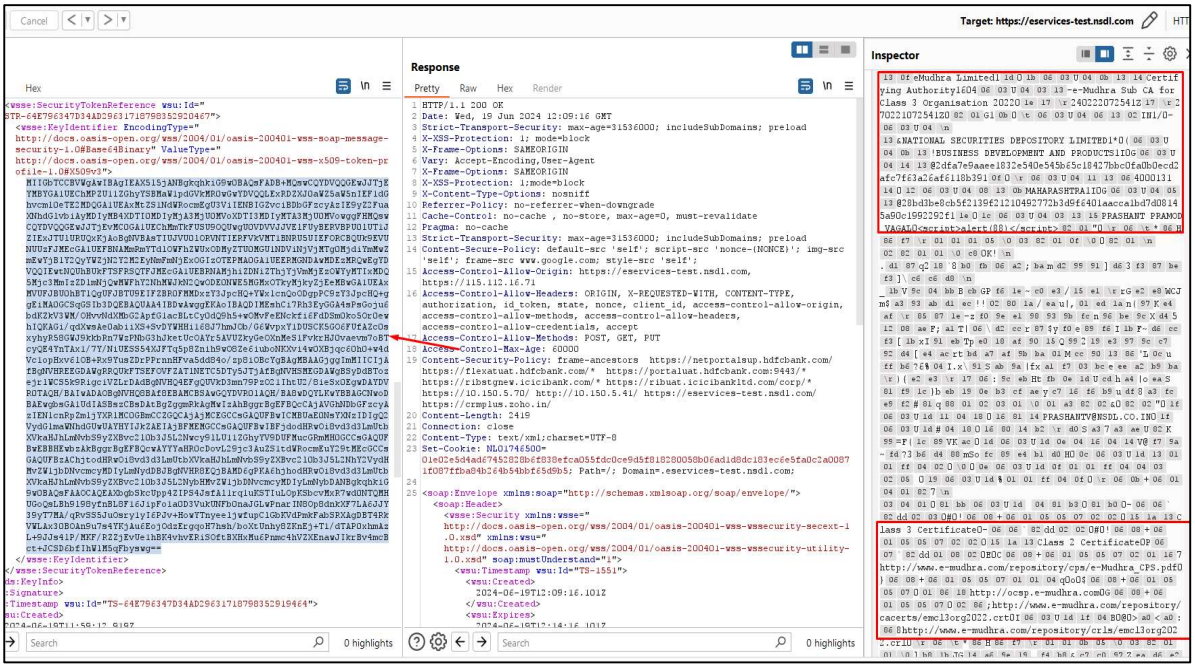


Observations

Case I: Base 64 is used for encoding

Step I: It was observed that password is encoded with base 64 which can be easily decoded using online tools as shown in the given screenshots below.





Mitigations

Implement SHA-256 algorithm to transfer sensitive data.

Appendix 'A'

OWASP TOP 10 API Security: 2023

The Open Web Application Security Project (OWASP) is a non-profit foundation dedicated to improving the security of software. OWASP API Security Top 10 is an online document on OWASP's website that provides ranking of and remediation guidance for the top 10 most critical web service / API security risks. It represents a broad consensus about what are the most critical API security flaws. The risks are ranked and based on the frequency of discovered security defects, the severity of the vulnerabilities, and the magnitude of their potential impacts. The purpose of the report is to offer developers and security professionals insight into the most prevalent security risks so that they may incorporate the report's findings and recommendations into their security practices, thereby minimizing the presence of these known risks in their web service / API.

The following table summarizes the OWASP API Security Top 10 2023 Most Critical Web Service / API Security Vulnerabilities:

S. No.	Vulnerability & Description	Impact
API1: 2023	***Broken Object Level Authorization*** APIs tend to expose endpoints that handle object identifiers, creating a wide attack surface Level Access Control issue. Object level authorization checks should be considered in every function that accesses a data source using an input from the user.	Unauthorized access can result in data disclosure to unauthorized parties, data loss, or data manipulation. Unauthorized access to objects can also lead to full account takeover.
API2: 2023	***Broken Authentication*** Authentication mechanisms are often implemented incorrectly, allowing attackers to compromise authentication tokens or to exploit implementation flaws to assume other user's identities temporarily or permanently. Compromising a system's ability to identify the	Attackers can gain control to other users' accounts in the system, read their personal data, and perform sensitive actions on their behalf, like money transactions and sending personal messages.

	client/user, compromises API security overall.	
API3: 2023	***Broken Object Property Level Authorization*** Looking forward to generic implementations, developers tend to expose all object properties without considering their individual sensitivity, relying on clients to perform the data filtering before displaying it to the user. And binding client provided data (e.g., JSON) to data models, without proper properties filtering based on an allow list, usually leads to Mass Assignment. Either guessing objects properties, exploring other API endpoints, reading the documentation, or providing additional object properties in request payloads, allows attackers to modify object properties they are not supposed to.	Excessive Data Exposure commonly leads to exposure of sensitive data. Exploitation may lead to privilege escalation, data tampering, bypass of security mechanisms, and more.
API4: 2023	***Unrestricted Resource Consumption*** It's common to find APIs that do not limit client interactions or resource consumption. Crafted API requests, such as those including parameters that control the number of resources to be returned and performing response status/time/length analysis should allow identification of the issue. The same is valid for batched operations. Although threat agents don't have visibility over costs impact, this can be inferred based on service providers' (e.g. cloud provider) business/pricing mode	Exploitation can lead to DoS due to resource starvation, but it can also lead to operational costs increase such as those related to the infrastructure due to higher CPU demand, increasing cloud storage needs, etc.
API5: 2023	***Broken Function Level Authorization***	Such flaws allow attackers to access

	Complex access control policies with different hierarchies, groups, and roles, and an unclear separation between administrative and regular functions, tend to lead to authorization flaws. By exploiting these issues, attackers gain access to other users' resources and/or administrative functions.	unauthorized functionality. Administrative functions are key targets for this type of attack.
API6: 2023	***Unrestricted Access to Sensitive Business Flows*** Lack of a holistic view of the API in order to fully support business requirements tends to contribute to the prevalence of this issue. Attackers manually identify what resources (e.g. endpoints) are involved in the target workflow and how they work together. If mitigation mechanisms are already in place, attackers need to find a way to bypass them.	It might prevent legitimate users from purchasing a product, or lead to inflation in the internal economy of a game.
API7: 2023	***Server-Side Request Forgery (SSRF)*** Lack of or improper validation of such URIs are common issues. Regular API requests and response analysis will be required to detect the issue. When the response is not returned (Blind SSRF) detecting the vulnerability requires more effort and creativity.	Successful exploitation might lead to internal services enumeration (e.g. port scanning), information disclosure, bypassing firewalls, or other security mechanisms. In some cases, it can lead to DoS or the server being used as a proxy to hide malicious activities.
API8: 2023	***Security Misconfiguration*** Security misconfiguration is commonly a result of unsecure default configurations, incomplete or ad-hoc configurations, open cloud storage, misconfigured HTTP headers, unnecessary HTTP methods, permissive Cross-Origin resource sharing (CORS), and verbose error	Security misconfigurations can not only expose sensitive user data, but also system details that may lead to full server compromise.

	messages containing sensitive information.	
API9: 2023	***Improper Inventory Management*** APIs tend to expose more endpoints than traditional web applications, making proper and updated documentation highly important. Proper hosts and deployed API versions inventory also play an important role to mitigate issues such as deprecated API versions and exposed debug endpoints.	Attackers may gain access to sensitive data, or even take over the server through old, unpatched API versions connected to the same database.
API10 :2023	***Unsafe Consumption of API's*** Developers tend to trust and not verify the endpoints that interact with external or third-party APIs, relying on weaker security requirements such as those regarding transport security, authentication/authorization, and input validation and sanitization. Attackers need to identify services the target API integrates with (data sources) and, eventually, compromise them.	The impact varies according to what the target API does with pulled data. Successful exploitation may lead to sensitive information exposure to unauthorized actors, many kinds of injections, or denial of service.

Table 1: OWASP API Security Top 10 - 2023

Reference: https://owasp.org/www-project-api-security/

Appendix 'B'

SANS TOP 25 Most Dangerous Software Errors

The SANS Institute is a cooperative research and education organization. The SANS Top 25 Most Dangerous Software Errors is a list of the most widespread and critical errors that can lead to serious vulnerabilities in software (please note: not all vulnerability types apply to all programming languages). The vulnerabilities include insecure interaction between components, risky resource management, and porous defenses.

The following table summarizes the CWE/SANS TOP 25 Most Dangerous Software Errors:

Rank	CWE	Description
1	***CWE-119*** Improper Restriction of Operations within the Bounds of a Memory Buffer.	The application performs operations on a memory buffer, but it can read from or write to a memory location that is outside of the intended boundary of the buffer.
2	***CWE-79*** Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	The application does not neutralize or incorrectly neutralizes user-controllable input before it is placed in output that is used as a web page that is served to other users.
3	***CWE-20*** Improper Input Validation	The application receives input or data, but it does not validate or incorrectly validates that the input has the properties that are required to process the data safely and correctly.
4	***CWE-200*** Information Exposure	The application exposes sensitive information to an actor that is not explicitly authorized to have access to that information.
5	***CWE-125*** Out-of-bounds Read	The application reads data past the end, or before the beginning, of the intended buffer.

6	***CWE-89*** Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	The application constructs all or part of an SQL command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended SQL command when it is sent to a downstream component.
7	***CWE-416*** Use After Free	Referencing memory after it has been freed can cause a program to crash, use unexpected values, or execute code.
8	***CWE-190*** Integer Overflow or Wraparound	The application performs a calculation that can produce an integer overflow or wraparound, when the logic assumes that the resulting value will always be larger than the original value. This can introduce other weaknesses when the calculation is used for resource management or execution control.
9	***CWE-352*** Cross-Site Request Forgery (CSRF)	The Web Application does not, or cannot, sufficiently verify whether a well-formed, valid, consistent request was intentionally provided by the user who submitted the request.
10	***CWE-22*** Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	The application uses external input to construct a pathname that is intended to identify a file or directory that is located underneath a restricted parent directory, but the software does not properly neutralize special elements within the pathname that can cause the pathname to resolve to a location that is outside of the restricted directory.
11	***CWE-78*** Improper Neutralization of Special Elements used in an OS Command	The application constructs all or part of an OS command using externally-influenced input from an upstream component, but it does not neutralize

	('OS Command Injection')	or incorrectly neutralizes special elements that could modify the intended OS command when it is sent to a downstream component.
12	***CWE-787*** Out-of-bounds Write	The application writes data past the end, or before the beginning, of the intended buffer.
13	***CWE-287*** Improper Authentication	When an actor claims to have a given identity, the software does not prove or insufficiently proves that the claim is correct.
14	***CWE-476*** NULL Pointer Dereference	A NULL pointer dereference occurs when the application dereferences a pointer that it expects to be valid, but is NULL, typically causing a crash or exit.
15	***CWE-732*** Incorrect Permission Assignment for Critical Resource	The application specifies permissions for a security-critical resource in a way that allows that resource to be read or modified by unintended actors.
16	***CWE-434*** Unrestricted Upload of File with Dangerous Type	The application allows the attacker to upload or transfer files of dangerous types that can be automatically processed within the product's environment.
17	***CWE-611*** Improper Restriction of XML External Entity Reference	The application processes an XML document that can contain XML entities with URIs that resolve to documents outside of the intended sphere of control, causing the product to embed incorrect documents into its output.
18	***CWE-94*** Improper Control of Generation of Code ('Code Injection')	The application constructs all or part of a code segment using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special

		elements that could modify the syntax or behaviour of the intended code segment.
19	***CWE-798*** Use of Hard-coded Credentials	The application contains hard-coded credentials, such as a password or cryptographic key, which it uses for its own inbound authentication, outbound communication to external components, or encryption of internal data.
20	***CWE-400*** Uncontrolled Resource Consumption	The application does not properly control the allocation and maintenance of a limited resource, thereby enabling an actor to influence the amount of resources consumed, eventually leading to the exhaustion of available resources.
21	***CWE-772*** Missing Release of Resource after Effective Lifetime	The application does not release a resource after its effective lifetime has ended, i.e., after the resource is no longer needed.
22	***CWE-426*** Untrusted Search Path	The application searches for critical resources using an externally-supplied search path that can point to resources that are not under the application's direct control.
23	***CWE-502*** Deserialization of Untrusted Data	The application deserializes untrusted data without sufficiently verifying that the resulting data will be valid.
24	***CWE-269*** Improper Privilege Management	The application does not properly assign, modify, track, or check privileges for an actor, creating an unintended sphere of control for that actor.
25	***CWE-295*** Improper Certificate Validation	The application does not validate, or incorrectly validates, a certificate.

Table 2: CWE/SANS TOP 25 Most Dangerous Software Errors

Reference: https://www.sans.org/top25-software-errors/

Appendix 'C'

Tools Description

Burp Suite Professional

Portswigger's Burp Suite Professional is an advanced set of tools for testing web security. Burp Suite offers the features for both manual and automated scans. Through Burp Suite, a user can intercept HTTP traffic, find hidden attack surface, assess strength of tokens, perform brute-forcing and fuzzing, construct CSRF exploits, modify HTTP messages, scan for common vulnerabilities including the OWASP Top 10.

SoapUI

SoapUI is a tool for testing Web Services; these can be the SOAP Web Services as well RESTful Web Services or HTTP based services. SoapUI is an Open Source and completely free tool with a commercial companion -ReadyAPI- that has extra functionality for companies with mission critical Web Services.