CLOUD API SECURITY AUDIT

- An Extensive Approach to API Assessment -

Master Thesis

Charity Uche Orji
20210771

Aalborg University
Electronics and IT

AALBORG UNIVERSITY
STUDENT REPORT

Electronics and IT
Aalborg University
http://www.aau.dk

Title:
Cloud API Security Audit

Theme:
Master Thesis

Project Period:
Fall Semester 2023

Project Group:
Charity Uche Orji

Participant(s):
Charity Uche Orji

Supervisor(s):
Marios Anagnostopoulos

Copies: 1

Page Numbers: 54

Date of Completion:
June 2, 2023

Abstract:

The migration of computing services to the cloud and the use of APIs to communicate and interact with several applications over the internet come with its challenges. As these challenges increase by the day, it is very important that attention is paid to auditing the use of APIs, especially in enterprises. This project presents an extensive approach to assessing RESTful APIs. It commences by touching on why this is essential, discussing vulnerabilities to APIs, and outlining security requirements for APIs. A review of current API security audit frameworks and approaches to auditing is highlighted. Next, an extensive approach to assessing Cloud APIs is proposed using the presented security requirements as metrics. Finally, the results of an assessment of a Public API are discussed.

Contents

Chapter 1

Introduction

The use of cloud computing by individuals and organizations is on the increase. This comes as a result of the ease made available by cloud service providers (CSPs), in the form of storage and other useful resources. Cloud APIs make it very easy to access and use these various services. Application Programming Interfaces(APIs) are sets of functions and protocols that enable communication and sharing of resources between services and applications [18]. Cloud APIs therefore, enable communication and transfer of information between applications in the cloud. They can be used to access data or resources from an external server or database. They can equally be used to write or post data to cloud services. Many organizations and enterprises now employ the use of APIs to ease their work, and the services rendered to their clients. Since the use of cloud APIs involve some communication or transfer of data, it is very crucial to pay attention to its security. Also significant to enterprises using APIs, is the need to gain the trust of clients in the handling and protection of data (personal data). It is especially important if the APIs being used are public or external APIs, as the use of insecure APIs can affect an organization's reputation and finances. There have been several breaches in recent times, due to vulnerabilities in APIs. For example, in 2022 alone, there were data breaches in 5 companies/ platforms that exposed an average of 5.58 million accounts due to API security issues. These security issues include the likes of Broken Object Level Authentication, Broken Function level authentication, and lack of encryption[48]. The exploitation of these vulnerabilities could lead to privilege escalation, brute force attacks, data exposure, manipulation, and theft.

1.1 Problem Formulation

The vulnerabilities and data breaches now experienced in the use of APIs make consideration of API security very crucial. If an API is not secure, it affects the confidentiality and integrity of data hosted in its system and transmitted through its endpoint. It can also affect the availability of a ser-

vice to those using it. This research work, thus focuses on answering the question:
How can we better enhance the Security of Cloud APIs in use?
What practical measures are to be taken to ensure the confidentiality, integrity, availability, and appropriate authentication of API and API endpoints? To better approach this subject, we will consider the following research questions:

- What are the security vulnerabilities common to APIs?
- What are the essential security requirements for APIs?
- How best can cloud APIs be assessed/ audited?

This report is divided into chapters and sections to address these research questions. The methods used for this project and the reason for the choices are discussed in Chapter 2. The first and second research questions: *what are the security vulnerabilities common to APIs?* and *What are the essential security requirements for APIs* are answered in Chapter 3. Chapter 3 discusses some background information on APIs, such as API vulnerabilities, breaches, and security requirements. Related works on API security and different approaches to auditing are also presented in Chapter 3. The third research question: *How best can cloud APIs be assessed/audited?* is discussed extensively in chapter 4. The final chapter discusses and provides a summary of the work done, the limitations of the project, answers to the research questions, and suggestions on further areas of research on this subject.

Chapter 2

Methodology

This chapter discusses the approach and methods taken for this project. We take a closer look at methods used for gathering data, analyzing data, and the reason behind such choices. For clarity, the chapter is divided into sections covering: data collection, data analysis, choice of APIs used, and API assessment.

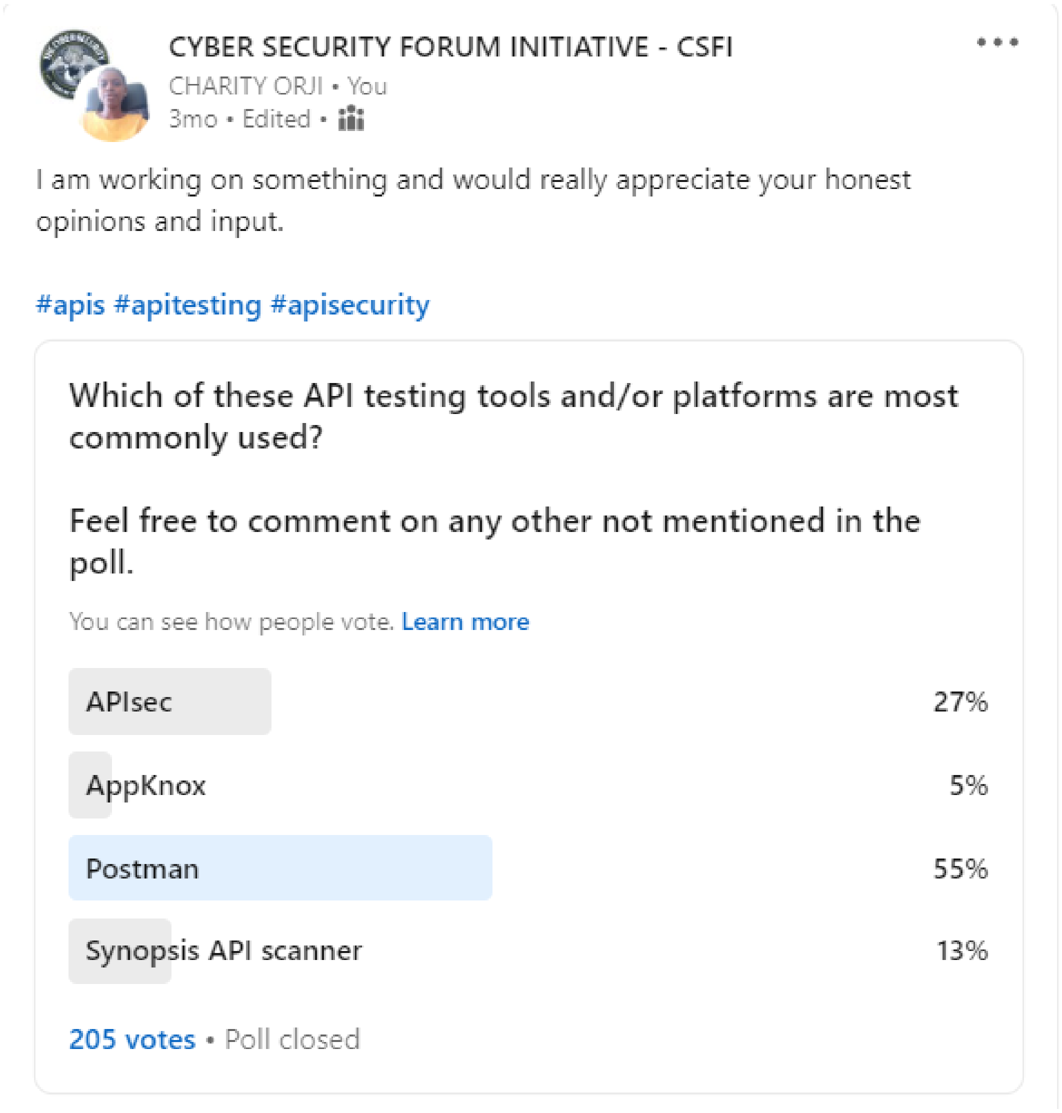
2.1 Data Collection

The data collection phase was focused on gathering relevant, published academic research work on subject matters such as APIs, API vulnerability, security, and auditing. Popular search engines were used, the likes of Google Scholar, IEEE Xplore, Aalborg University Library, and Semantic Scholar. To get suitable research papers, certain keywords were applied to the search. The keywords used were: *Cloud APIs auditing, API auditing*. This did not produce much result. One reason for this is that it is a relatively new field with very few published research works. As a result, the majority of the results from this search were irrelevant. 18 research publications were obtained from the search but only 4 were relevant.
However, the search for related academic work was broadened. More keywords were applied such as: *API testing, API vulnerabilities, API security, Information systems auditing, API security testing, API audit framework.*

2.2 Data Analysis

The choice of API testing tools and platforms used for this project were products of quantitative data analysis. The data collection process revealed a lot of API testing tools available in the market. From this list of API testing tools, 4 open-source tools were chosen randomly. An online poll was made in a forum on LinkedIn to seek the views of professionals in the Industry who make use of or have made use of these API testing tools. The aim of this poll was to find out the most commonly used API testing tools.

The 2 API testing tools with the highest votes as seen in Figure 2.1 were used for this project. However, it was observed that there were limitations on what can be achieved with one of the source tools: APISec. This led to the use of an alternative software testing tool called *Synk* [24].


CYBER SECURITY FORUM INITIATIVE - CSFI
CHARITY ORJI • You
3mo • Edited •

I am working on something and would really appreciate your honest opinions and input.

#apis #apitesting #apisecurity

Which of these API testing tools and/or platforms are most commonly used?

Feel free to comment on any other not mentioned in the poll.

You can see how people vote. Learn more

APIsec	27%
AppKnox	5%
Postman	55%
Synopsis API scanner	13%

205 votes • Poll closed

Figure 2.1: LinkedIn Poll Result

2.3 Choice of APIs Used

The APIs used for these projects are all Public APIs. YouTube Data API [14] and Simple Inventory API [31] are some of the APIs evaluated in this project. The reason for this choice is the difficulty to get organizations who would be willing to collaborate on auditing and assessing their APIs. Due to security reasons, many would be unwilling, to avoid any chance of exposure of data used by these APIs which may be sensitive in nature.

2.4 API Assessment/Auditing

This project is focused on auditing Cloud APIs. In general, auditing a system is done based on selected metrics. The same approach is applied here in this project. However, after careful research, it was observed that there is no known, published, or recognized framework specifically for auditing APIs. Based on this finding, our focus was shifted to security frameworks for the Cloud, such as the Cloud Control Matrix by the Cloud Security Alliance [8]. Further research was made to find out if there are still any security frameworks or standards related to the Cloud and specifically APIs. Further extensive research led to the discovery of a document called *Security Guidelines for Providing and Consuming APIs* by the Israeli chapter of Cloud Security Alliance [4]. The security controls contained in this document along with the CCM were used as metrics for the assessment of APIs in this project. Details on how these controls were used and their various areas of applicability are detailed in Chapter 4.

Chapter 3

Background and Related Work

This chapter works through background information on cloud computing architecture, service models, types of API, and API architecture. It also looks into API threats and vulnerabilities and security requirements. The chapter concludes with a presentation of related works on API security and existing API security frameworks.

3.1 Cloud Computing Architecture

Microsoft defines Cloud computing as "the delivery of computing services-including servers, storage, databases, networking, software, analytics, and intelligence-over the internet("the cloud") to offer faster innovation, flexible resources, and economics of sales" [30]. In order words, computing resources which are usually provisioned and accessed on-premise, are made available on demand over the internet [33]. This computing model makes the management of computing resources very easy and efficient, especially the sharing of resources.

3.1.1 Cloud Computing Service Models

The various services provided and accessed in the cloud are dependent on different service models. What a user is able to do, and how much control he has differs with each service model. The 3 main cloud computing service models are:

- Infrastructure as a Service (IaaS)
- Platform as a Service (PaaS)
- Software as a Service (SaaS)

Infrastructure as a Service(IaaS).
With this service model, a consumer is able to use cloud infrastructure(software and hardware) hosted by a cloud provider [33]. These could be servers, storage, and networking. The consumer only controls and manages components such as operating systems, deployed applications, and data.

Platform as a Service(PaaS)
These cloud computing services allow a consumer to develop, run and manage applications in the cloud [13]. The consumer only has control over his deployed applications.

Software as a Service(SaaS)
Software as a Service allows users to make use of cloud-based software solutions/ applications over the internet. Some examples of SaaS solutions include Gmail, Microsoft Office 365, Google Workspace, Adobe, etc. These applications can be accessed using a web browser or a program interface such as an Application Programming Interface(API) [33].
Cloud APIs enable communication and interaction with other applications and services, such as other SaaS products. With them, one is also able to access/ retrieve data from applications deployed in the cloud.

3.2 Types of APIs

The most common types of APIs we have are:

1. Open APIs
2. Public APIs
3. Partner APIs
4. Private APIs
5. Composite APIs

Open APIs
Open APIs are free and public APIs, with Open API specifications. They generally do not need the authorization to be accessed [5].

Public APIs
Public APIs are in some way similar to Open APIs [5]. They are publicly available for use to developers, but most do not have an Open-API specification. Some may also require a subscription to fully utilize the APIs [5, 16]. They are usually built with little security(authorization and authentication) measures [16]. The Google APIs are an example of Public APIs.

Partner APIs
These APIs are only accessible to certain parties, usually authorized clients with licenses who are partners with the API developers [5, 16]. Due to their limited access and nature, partner APIs have more built-in security measures than public APIs.

Private APIs
Private APIs are also known as Internal APIs. They are only accessible and designed for use in an organization. Thus, they are not available to external or third parties [16].

Composite APIs
A Composite API has multiple APIs embodied in it, such that multiple requests can be made even from multiple servers, with a single API call [5]. These APIs are efficient, as they reduce the server load and system complexity [16].

3.3 API Architecture

API Architecture is the framework of rules or protocols for creating and building APIs. There are several architectural styles adopted in building APIs. These include:

- REST
- SOAP
- GraphQL

3.3.1 REST APIs

REST stands for Representational State Transfer [5]. REST APIs allow communication with web services over the HTTP protocol. All requests are made using HTTP methods. The methods commonly used are POST, GET, PUT, PATCH, and DELETE. These methods are used to create, read, update, modify, and delete resources respectively [17].
To access a web service or resource in the cloud using APIs, a request is made in the form of a web URL using any of the HTTP methods above [17]. In turn, a response is sent back by the server in HTML, XML, Image, or JSON. The most commonly used response is in JSON format [17].

3.3.2 SOAP

SOAP stands for Simple Object Access Protocol. SOAP APIs are more structured than REST APIs. It encodes data sent and received over the API client in XML [50]. SOAP APIs use HTTP as a primary protocol for transport. However, it is language-independent and can use HTML, plain text, and JSON as data formats. It also makes use of other protocols such as SMTP, TCP, and UDP to transport messages [50]. Due to its structure, SOAP APIs are not as flexible and lightweight as REST APIs.

3.3.3 GraphQL

GraphQL is a protocol developed by Facebook. To use GraphQL, an outline is made of all possible queries and their specific type of response. [49]. This makes GraphQL very good for privacy, as it keeps data secure, returning only exactly what the user requests for [5, 49]. It is also well structured and provides detailed error logs [49].

3.4 API Security

A report published in 2019 showed that APIs constitute about 83% of today's internet traffic [1]. As more enterprises adopt the use of APIs in their organizations, their percentage is sure to increase. Different types of APIs are used in different ways and for different purposes by organizations today. APIs no matter the type or purpose, could be vulnerable to certain security risks. OWASP in 2019 published a list of the top 10 vulnerabilities common to APIs [48].

3.4.1 OWASP API Security Top 10

The OWASP Top 10 for APIs are threats and vulnerabilities most exploited and common to APIs. For 2023, its API Security Top 10 release candidate on GitHub has a list of top threats to API. They are as follows in order of priority:

1. Broken object-level authorization
2. Broken Authentication
3. Broken Object Property Level Authorization
4. Unrestricted resource consumption
5. Broken function level authorization

6. Server Side Request Forgery
7. Security misconfiguration.
8. Lack of protection from automated threats
9. Improper Inventory management
10. Unsafe Consumption of APIs

Broken object level authorization
An API is vulnerable to broken object-level authorization if an attacker is able to manipulate the ID of an object. This is usually implemented at the code level. Successful exploitation of this vulnerability can lead to unauthorized access resulting in data exposure, loss, and manipulation [38].

Broken Authentication
The absence of user authentication or improper implementation of it on API endpoints and API management tools makes it vulnerable to malicious actors. An API is vulnerable to broken authentication if it allows the use of weak passwords, sends sensitive details such as tokens and passwords in its URL, or uses weak encryption keys. This vulnerability can be exploited to gain access to users' accounts in a system, reading their data and probably using the data obtained for something malicious [48, 39].

Broken Object Property Level Authorization
With the appropriate function level authorization and user authentication implemented, an API endpoint can still be vulnerable to broken object property level authorization(BOPLA). BOPLA comes with the inability of an API endpoint to validate what specific properties of an object a user has access to. This can lead to excessive data exposure, data loss, and data manipulation [40]

Unrestricted resource consumption
An API is vulnerable if any of the following is lacking or set to be too low or high: execution timeouts, maximum allocable memory, the maximum number of file descriptors, the maximum number of processes, maximum upload file size, number of operations to perform in a single API client request (e.g. GraphQL batching), number of records per page to return in a single request-response and third-party service providers' spending limit [41]

Broken Function Level Authorization
Broken function level authorization occurs as a result of the inability to properly define functions such as the administrative and general user functions [48], and restricting it to authorized users alone [52]. These irregularities could give an average user to resources only accessible to administrators. Attackers could take advantage of this flaw to steal valuable or sensitive data.

Server Side Request Forgery
This applies to API endpoints that take a URL as an input and access it without validation [42]. An attacker could exploit this, making the vulnerable API endpoints redirect its request to an internal network [23]. This exposes the network, resulting in service enumeration and information disclosure [42].

Security misconfiguration
This entails the use of unpatched and out-of-date system components, lack of Transport Layer Security(TLS), enabling features that are not necessary, and lack of Cross-Origin Resource Sharing (CORS) policy on APIs. These can lead to the exposure of sensitive data and possibly full system compromise [43].

Lack of protection from automated threats
This threat exploits sensitive business flows and automates access to it. When this is done by an attacker and in excess, it can lead to a DOS attack: preventing legitimate users from using the service or purchasing a product. It can also lead to the spread of false news since the attackers are able to send a lot of messages [44].

Improper Inventory management
An API is vulnerable to improper inventory management if it has a documentation blindspot and a data blindspot. An API has a documentation blindspot if there is outdated or no documentation of the API and its endpoint, providing inventory and essential details on the purpose and visibility of the APIs. Additionally, if an API shares sensitive data with a third party without any inventory and basis for such, the API is said to have a data blind spot. Improper Inventory management makes it difficult to find and fix vulnerabilities in APIs. Attackers can also gain access to sensitive data through unpatched APIs [45].

Unsafe Consumption of APIs
This happens when developers fail to strengthen the security of their endpoints when using external APIs by well-known companies. The trust they have in these third-party APIs makes them pay little or no attention to important things such as input validation and sanitation of data from other APIs. They fail to also look into the security of the channel of communication used by these APIs. As a result of this, the endpoints are left exposed to injection attacks and exposure of sensitive data by attackers [37].

There are so many APIs and different API management platforms as well. As a result, security functions are implemented in different ways in these tools. This requires a proper study and understanding of these tools to properly affect these functions. Vulnerabilities in API are not limited to the OWASP TOP 10. APIs are written programs and codes of instructions. This means that certain code vulnerabilities would also be applicable to APIs.For example, improper input validation.

3.4.2 Cloud API Data Breaches

A good number of Cloud API data breaches exploiting some of the API vulnerabilities mentioned earlier were experienced in the year 2022.
The Broken Object Level Authorization(BOLA) vulnerability was quite prevalent in 2022. In January 2022 for example, Twitter, a social media platform was made aware of a Broken Object level authorization vulnerability in its system. This enabled the disclosure of certain details of a Twitter account by an API if an email address or phone number associated with that account is entered by an individual [55]. About 5.4 million Twitter accounts were exposed in July 2022 and reportedly offered for sale as a result of this bug in their code [55, 27]. Another case of a data breach experienced as a result of the BOLA vulnerability was that of Flexbooker. Flexbooker offers digital appointment scheduling services. It uses an Amazon S3 bucket for cloud storage. It experienced a DDoS attack on its AWS server, which attackers exploited to gain unauthorized access to sensitive information of over 3.7 million of its customers [57]. The Texas Department of Insurance, on the other hand, had some security issues with its web application that handles worker's compensation information. This was a case of Broken Function Level Authorization. The program code allowed internet access to a protected area of the application. This led to the exposure of certain information of about 1.8 million Texans, such as names, addresses, dates of birth, phone numbers, part or all of Social Security numbers, and information about injuries and workers' compensation claims [19, 27]

3.4.3 Cloud API Security Requirements

The OWASP API Security T0p 10 and the examples of data breaches discussed in the previous sections highlight that are certain risks associated with the use of APIs. These risks could be in [2]

- API accessibility
- the volume of data involved
- the sensitivity of the data
- integration frequency
- data retention
- third-party trust and
- third-party security

The document: *Security Guidelines for Providing and Consuming APIs* [3], of the Cloud Security Alliance(CSA) provides guidelines that can be applied in all phases of the development cycle of APIs as well as during the use. The guidelines are presented based on two use cases based on the associated risks, namely:

1. Ingress Access and
2. Egress Access

Ingress API Connectivity
Ingress API connectivity deals with a third party accessing data from a public or private API [4]. A good example is an organization making use of an external or public API in its application. These security controls apply from the development stage to the use and/or monitoring of APIs. They are organized based on risk areas.

API Accessibility	- Threat modelling and appropriate counter measures. - Service authentication (For example, use of API keys, OAuth, JWT.) - Complete mediation on API management platform (Authorization checks must be done on all requests). - Token Strength (use state of-the-art encryption algorithm). - Protection of testing/staging environments. This includes software update and proper configurations. This might also include restricting them from external parties. - Regular penetration testing and continuous vulnerability scanning by certified professionals. - Security oriented code review. - Application security scanning and secret scanning. Removal of hard coded sensitive string such as secrets, passwords, and keys. - Use of TLS and only strong cipher suites. - Source IP limitation. - Lockdown accounts not in use, unsuccessful authentication attempts, and malicious accounts based on a predefined threshold. - Only mandatory ports (ports in use) should be exposed especially on the admin APIs. This also applies to external network infrastructure related to the API hosting. - Protect API gateways with Web/ API application Firewall. - Session termination on the API gateway. Termination of inactive authentication tokens or API keys after predefined inactivity time. - Protection of API against DOS and brute force attack. For example, implementing rate limiting. - Proper and updated documentation of the API inventory
Required Permissions to all functions	- Threat modeling and countermeasures - Least privilege (authorization should be limited to only permissions and HTTP methods needed to perform a required action - Authorization should be implemented as a separate service. - Request rate limiting - Data sanitation (to ensure removal of sensitive data)

Table 3.1: Security Controls To Risk Areas in API [4]

Volume of Data Involved	- Request rate limiting - Regex Denial of Service (When using Regex for input validation, restrict input length to avoid DOS attack) - Protect API gateways with Web/ API application Firewall - Session termination - Protection of API platform from DOS and brute force attacks. For example, implementing rate limiting - API Security tool for implementing API protection (Detection and response to API security incidents)
Sensitivity of Data	- Threat modeling and counter measures - Service authentication - Granting authorization on principle of least privilege - Authorization should be implemented as a separate service - Input validation decoupled from the application - Complete mediation on API management platform (Authorization checks must be done on all requests). - Input validation and output encoding - Data sanitation (to ensure removal of sensitive data) - Error handling (should not disclose any information that could be exploited by an attacker. E.g internal path, filename, data, etc.) - Protection of testing/ staging environments. - Regular penetration testing and continuous vulnerability scanning - Security-oriented code review - Application Security scanning and secret scanning - Web/API Application Firewall - Continuous monitoring across the entire stack. Enable audit, access, and other relevant types of logs. - Storing HTTP Access Logs for use during an incident or digital forensics. The logs should not have any sensitive data. This practice should be in line with necessary data retention policy and other regulations. - API Security tool for implementing API protection (Detection and response to API security incidents)
Trust of third Party	- Threat modeling and appropriate counter measures. - Request rate limiting - Input validation and output encoding - Regex Denial of Service (When using Regex for input validation, restrict input length to avoid DOS attack) - Data sanitation - Error handling (should not disclose any information that could be exploited by an attacker. E.g internal path, filename, data, etc.) - Regular penetration testing and continuous vulnerability scanning - Application security testing and secret scanning - Web/API Application Firewall - Protection of API platform from DOS and brute force attacks. For example, implementing rate limiting - API Security tool for implementing API protection (Detection and response to API security incidents)

Table 3.2: Security Controls To Risk Areas in API [4]

Existing vs. New API	- Threat modeling and countermeasures - Proper and updated documentation of the API.
Integration Frequency	- Threat modeling and countermeasures - Service authentication - Granting authorization on principle of least privilege - Authorization should be implemented as a separate service - Request rate limiting - Data sanitation - Security-oriented code review - Session termination - Continuous monitoring across the entire API stack. - Storing HTTP Access Logs for use during an incident or digital forensics. The logs should not have any sensitive data. - API Security tool for implementing API protection.
Data Retention	- Threat modelling and countermeasures - Granting authorization on principle of least privilege - Authorization should be implemented as a separate service - Request rate limiting - Data sanitation - Regular penetration testing and continuous vulnerability scanning - Security-oriented code review - Lockdown accounts not in use, unsuccessful authentication attempts and malicious accounts based on a predefined threshold. - Session termination - Continuous monitoring across the entire API stack. - Storing HTTP Access Logs for use during an incident or digital forensics. The logs should not have any sensitive data. - API Security tool for implementing API protection.
Third Party's Security and Security compliance status (SOC 2, penetration test, ISO 27001, etc.)	- Threat modeling and countermeasures - Granting authorization on the principle of least privilege - Cross-Site Request Forgery (CSRF) protection - Request rate limiting - Storage of Application Secrets in a secure location, away from code repository. Encryption of API keys in transit and delivery through a secure channel. - Complete mediation on API management platform (Authorization checks must be done on all requests). - Input validation and output encoding - Regex Denial of Service (When using Regex for input validation, restrict input length to avoid DOS attack) - Protection of testing/ staging environments. - Regular penetration testing and continuous vulnerability scanning - Installation of a valid, up-to-date and signed TLS certificate on the API server/gateway. - Lockdown accounts not in use, unsuccessful authentication attempts , and malicious accounts based on a predefined threshold. - Continuous monitoring across the entire API stack.

Table 3.3: Security Controls To Risk Areas in API [4]

Egress API Connectivity
Egress API connectivity deals with API connectivity that grants read/write access to an external party, i.e. sharing information owned by one's organization to a third party [4]. This is more of an outbound connectivity than an ingress, which is inbound. For this use case, the following security controls apply:

1. **Threat modeling and countermeasures**: Identification of possible threats during the design of the API, and measures to deal with it [4].
2. **Storing Secrets**: Storing of encryption keys and secrets in a safe location [4].
3. **Application Security Scanning and Secrets Scanning**: Scanning of the program code for vulnerabilities before deployment(Static Testing) and continuously after deployment(Dynamic testing) [4].
4. **TLS Valid Certificate**: TLS certification validation is necessary to ensure a secure connection to the third party [4].
5. **Session Termination**: All inactive external connections should be closed based on preset rules [4].
6. **Destination IP and Port limitation**: Connections to external parties should be restricted to fixed IP addresses and ports [4].
7. **Continuous Monitoring**: This should be done to look out for possible security issues [4].
8. **Detection and Response**: Implementation of appropriate tools and procedures for detection of, and response to security issues [4].
9. **Documentation**: Proper and updated documentation of the API inventory [4].

The security controls presented in Tables 3.1, 3.2, and 3.3 serve as security requirements for Cloud APIs. More details on these controls are discussed later on in Chapter 4.

3.5 Related Works

This section presents related work on API vulnerabilities and security. It also takes a look at existing API security Audit Frameworks.

3.5.1 API Vulnerabilities

With its use, the security features provided by API management tools are usually not sufficient to ensure the secure use of APIs [53, 54]. A contributing factor is the lack of security in the design of APIs. Another significant reason is the inability of API management tools and platforms to handle authentication and authorization. Authentication and authorization services usually identify the source of API requests. The failure of API management tools in doing this can leave APIs vulnerable to attacks caused by a lack of authorization. This is reflected in the T-Mobile breach that led to the compromise of 2.3 million customer data. The API failed to check who was making the query. As a result, any request for customer data made with a valid phone number was granted. This led to the exposure of customer data which could be used to gain access to other accounts or services used by a customer to steal sensitive data. DDoS attack is another vulnerability common to APIs. Poor design in APIs such as not putting rate limits on API requests could lead to DDoS attack [53].

3.5.2 API Security

To properly manage vulnerabilities associated with APIs, API security testing and audit are very necessary. Analysis of API assets is one way of testing APIs, as stated by Sun et al [54]. However, identifying these assets through active scanning can be quite challenging. API data flow analysis, on the other hand, can be used to supplement this. Sun et al proposed an API security audit system based on the Internet traffic. The security audit system is divided into three parts based on assets, namely: Asset discovery, Asset portraits, and Vulnerability detection. Asset discovery allows for the identification of API assets including unknown APIs, as well as how they process data. This discovery is enabled through methods such as traffic analysis, docking data, and importing data. The Asset portrait produces a profile list of identified APIs using data analysis. This contains vital information such as the functions and permissions the API has such as user login, registration, data query, and administrator permissions. The API safety detection and protection module which handles vulnerability detection, applies active and passive means to audit APIs. It allows for the detection of possible vulnerabilities such as Remote command/code execution, data leakage, unauthorized access, unauthorized access, and logic defects in APIs. This, in turn, would enable an immediate response to fixing the vulnerabilities where applicable. The figure below shows the structure of Sun et al. Security Audit Framework.

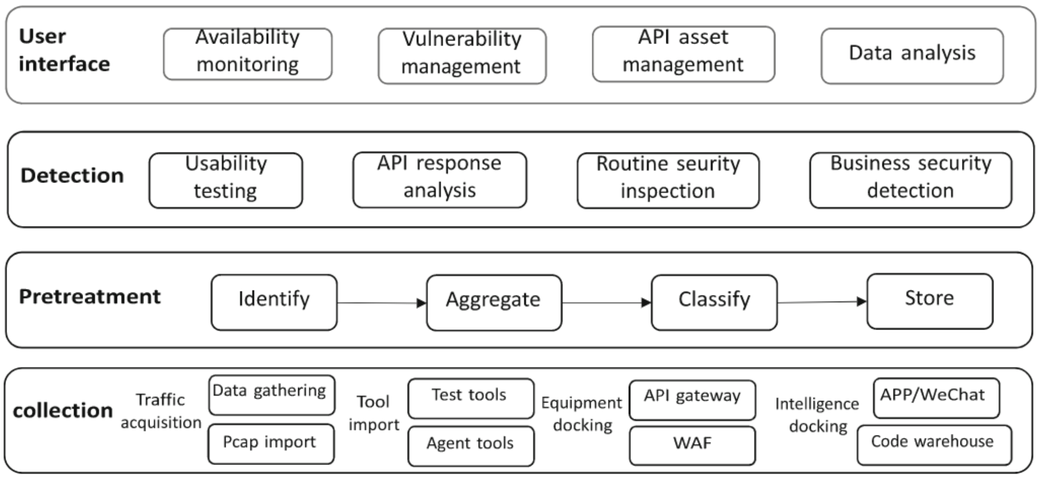


Figure 3.1: Security Audit Framework [54]

For active property monitoring as opposed to the passive monitoring of APIs, Atlidakis et al. came up with 4 security rules as security checkers on REST APIs [6]. These rules are designed to check out for essential security properties REST APIs should have, which are usually not detected by passive monitoring. The rules are the Use-after-free rule, Resource-leak rule, Resource-hierarchy rule, and User-namespace rule. The Use-after-free rule states that any deleted API resource must not be available for further use. As such any more attempts at using it should fail. An attacker could exploit a violation of this rule to bypass resource quotas leading to the elevation of privilege attacks or possible corruption of services. With the Resource-leak rule, a resource that was not successfully created must not be accessible nor "leak" any associated resources in the backend service state. If a resource quota is exhausted by violation of the resource-leak rule, it becomes impossible to create new resources if needed. Violation of this rule can also affect the overall performance of the service. The resource-hierarchy rule prevents a child resource created from a parent resource, to be accessible by another parent resource with no parent-child relationship to the given sub-resource. Violation of this rule gives an attacker unauthorized access to a parent object identifier, which would in turn be used to steal or hijack its child object. The last rule called the User-namespace rule, states that a resource created in a user namespace must not be accessible from another user namespace. In other words, if a resource is created using the token of user A, the same resource must not be accessible using the token of user B. Violation of this rule allows an attacker to perform unauthorized operations on a victim's resource, using unauthorized authentication tokens.
Atlidakis et al. implemented four active checkers to check and detect violations of these 4 rules in APIs.

- Use-after-free checker
- Resource-leak checker
- Resource-hierarchy checker and

- User-namespace checker

Use-after-free checker: the algorithm for the use-after-free checker is called after the main driver of an API executes a DELETE request. It takes as input: a sequence *seq* of requests(the latest executed by the API); global_cache which has the most recent object types and ids for the dynamic objects created until that point and the set of all available API requests. The id of the deleted object type is fetched from the global_cache, and a request is executed on it. It is expected that the request on the deleted object fails and returns a "404 Not Found" HTTP status code in their response. But if the request is successfully executed, then the use-after-free rule is violated.
Resource-leak checker: This checker takes the same three inputs as the use-after-free checker. It operates on request sequences executed by the main driver whose last request led to an invalid HTTP status code in the response. The algorithm iterates over object ids "guessed" for the current object type for which an invalid HTTP status code was received. The function GUESS takes as argument an object type and returns a set of possible object ids matching this type and which were not created successfully. A guessed object-id value is temporarily added to the global cache of properly-created dynamic objects. The algorithm then tries to trigger a resource-leak violation or asserts that no such violation occurs for the given request sequence
Resource-hierarchy checker: This checker takes a sequence seq of requests(the latest executed by the API) and the global_cache as inputs. The algorithm records the object types consumed by the last request and those prior to it. After this, it retrieves the most recent id of each child object consumed only by the last request and attempts to violate its hierarchy. The algorithm executes a request on the old child id using a parent id different from that of the object. Failure of the request is an indication that the resource-leak rule is not violated.
User-namespace checker: This checker tries to re-execute the valid last request of any test case executed by the API's main driver with another authentication token. Successful execution of this request indicates a violation of the use-namespace rule.

3.5.3 Cloud and API Auditing

Auditing refers to the steps and processes an organization takes in evaluating and assessing its IT infrastructure, to ensure it is compliant with certain security requirements [56]. These security requirements could be industry standards such as the CCM, NIST, and ISO standards. It could also be internal requirements set by the organization. However, auditing a cloud system in which cloud APIs are inclusive, and using industry standards can be challenging as these systems are designed in different ways to cater to different needs. This makes it quite impractical to evaluate and assess these different systems in relation to one another using the same standard [56] There are several approaches to auditing. These are the retroactive approach, the intercept-and-check approach, and the proactive approach [26]. The retroactive approach to auditing in the cloud utilizes security patches and updates to detect intrusion and any vulnerabilities [7]. It executes two copies of patched code: one with the patch and the other without, using the same input to determine its outcome. A varying response on the patch shows an indication of a vulnerability [58]. This approach has its limitation as it only detects an incident after it has occurred [34]. The intercept-and-check approach as the name implies, intercepts requests to verify their compliance before accepting or rejecting it [7]. This is an improvement to the retroactive approach. But its method of auditing creates delays in responding to requests [25]. The proactive approach audits cloud systems way ahead of time, before any security incident occurs [34, 26].

The API Security Audit Framework by Sun et.al and the Security Rules and Checkers by Atlidakis et al. all focus on different parts of an API system that are crucial to assessing and maintaining API security. Nevertheless, each approach is not sufficient enough to evaluate an API. In this report, we would present a more general, but specific API assessment method that takes into account various areas of an API system.

Chapter 4

API Assessment

This chapter presents a more extensive approach to auditing APIs. First, we take a look at guidelines and/or principles for auditing APIs. This is followed by how to evaluate APIs with API management tools and API testing tools using certain standards and controls. Finally, we present the outcome of assessing some public APIs using this proposed approach. The words "audit"/"auditing" and "assess"/"assessing" are used interchangeably in this chapter, and both refer to the same thing.

4.1 Audit Principles and Guidelines

Auditing of IT systems makes certain that the systems meet certain security requirements [22]. The standard ISO/IEC 19770-1:2017 states 3 reasons for auditing an IT asset management system, namely: to determine if a system [22]

- meets the organization's own requirements for the system
- fulfills the requirements of the ISO/IEC 19770-1:2017 standard
- is effectively implemented and maintained [22].

These reasons mentioned above also apply to APIs in order to ensure their security.
Audits are usually carried out based on some criteria [21]. These audit criteria are outlined in the ISO 19011:2018 standard on Guidelines for auditing management systems. They include [21]

```
- requirements defined in one or more management system standards;
- policies and requirements specified by relevant interested parties;
- statutory and regulatory requirements;
- one or more management system processes defined by the organization
```

```
  or other parties;
- management system plan(s) relating to the provision of specific outputs
  of a management system(e.g. quality plan, project plan).
```

In general, when auditing any system, it is essential to know the objective, scope, and criteria of the audit [21], The objective and scope are usually defined by the auditee. The criteria is agreed upon by the auditor and auditee.

4.2 API Assessment

As with any IT system, the objectives, scope, and criteria for auditing APIs should be defined from the beginning. Of equal importance is getting to know the number of APIs the organization has and uses. Having this knowledge helps to more clearly define the scope of the assessment. The organization should have necessary documentations that provides valuable information such as

- Type of APIs (Public or private)
- identify assets used by the APIs
- Policies associated with the use

The assets used by these APIs should be classified based on the level of sensitivity, importance, and associated risks to the organization. For organizations that make use of API gateways and management tools, a thorough understanding of how these tools work would make for a very efficient assessment.

Cloud API Audit Criteria

One of the criteria that could be used for assessing IT systems are requirements defined in management system standards [21]. Some widely recognized standards are the ISO standards [20] and the NIST framework [32]. There is also the Cloud Control Matrix (CCM) [8] developed by Cloud Security Alliance(CSA). Although there are no standards specifically dedicated to APIs, the above-mentioned standards contain security controls that apply to Software applications and the Cloud as a whole. As a result, some of the controls can be applied to APIs as well. In the subsections that follow this chapter, we will have an overview of these controls that can prove very useful in assessing APIs.

4.3 Proposed API Assessment Method

The objective here is focused on Information security and assessing how cloud APIs maintain Confidentiality, Integrity, and Availability. The assessment criteria applied are select controls from the Cloud Control Matrix(CCM) [8] and the document: *Security Guidelines for Providing and Consuming APIs* [4], both developed by the Cloud Security Alliance. In using these publications, we aim at highlighting how APIs can be evaluated with the associated controls. The choice of controls from CCM V4.0.7 for assessment is based on the article: *Top Threats to Cloud Computing Pandemic Eleven*. The controls dependent on 5 domains are seen in Table 4.1

Control Domain	**Control Title**	**Control ID**
Application and Interface Security(AIS)	Application and Interface Security Policies and Procedures	AIS-01
AIS	Secure Application Design and Development	AIS-04
AIS	Automated Secure Application Deployment	AIS-06
Change Control and Configuration Management (CCC)	Change Management Policy and Procedure |	CCC-01
CCC	Quality Testing	CCC-02
CCC	Change Agreements	CCC-05
Cryptography, Encryption & Key Management (CEK)	Data Encryption	CEK-03
CEK	Encryption Algorithm	CEK-04
Data Security and Privacy Lifecycle Management (DSP)	Security and Privacy Policy and Procedures	DSP-01
DSP	Data Inventory	DSP-3
DSP	Data Classification	DSP-4
DSP	Data Flow Documentation	DSP-5
Infrastructure & Virtualization Security (IVS)	Network Security	IVS-03
IVS	OS Hardening and Base Controls	IVS-04
IVS	Network Defense	IVS-09

Table 4.1: CSA CCM Controls Version 4.0

These controls are further classified in the area of applicability:

- Documentation
- Code
- API Management Platform

Documentation:
Documentations are very important in any IT system and this includes sys-

tems using APIs. Proper documentation makes monitoring of APIs easier. With these documentation, APIs can be examined to see if they are in compliance with the security requirements outlined in the documentation. Documentations here include policies and logs. The controls DSP-01, DSP-04, and DSP-05 are applied primarily to documentation. Thus, during an audit, an auditor should look out for documented evidence such as data inventory, flow diagrams, and logs. These should provide details on the classification, handling, storage, sharing, and protection of data [8], which could reveal if there is any risk of excessive exposure of data.

Code:
APIs are basically sets of instructions and codes. As such they should follow the same security design principles as softwares, during development. One such principle is input validation. This is what the control AIS-04 looks out for. Other things to pay attention to include:

- **query parameters**: be certain that no sensitive data such as API key, client secret, or personal data that could be traced to an individual, are used as query parameters. These are very likely to be exposed and leaked to authorized persons who could use them for selfish and inferior motives.
- **scope**: the principle of least privilege should be applied in defining scopes to avoid excessive exposure of data
- **response body in JSON format**: The expected response to an API request should be examined to ensure that only the data required by the user is contained in the response body.

Details are provided in the later section of this chapter on what more can be ascertained from analyzing the codes that make an API. Even though analysis of the code can reveal a lot about an API, this is still limited. This makes the use of API management tools very essential.

API Management Platform and Tools:
API management enables developers to incorporate APIs into applications. In an enterprise, API management tools provide a platform to control access to APIs and monitor and analyze the use of the APIs. They also provide useful documentation [35]. As a result, API management platforms/tools can prove very useful in evaluating APIs. The CCM controls in Table 4.1 which can be assessed from API management platform/tools are the AIS-01, CCC-01, CEK-03, and CEK-04. Details of how these controls can be assessed using Microsoft Azure API Management Service and Google Cloud Platform would be discussed in the later parts of this chapter.

Another assessment criteria used here for evaluating APIs is based on the document: *Security Guidelines for Providing and Consuming APIs* (SGPC APIs) also from the Cloud Security Alliance. The controls in the document, are mapped to the OWASP API TOP 1O 2019 [48] and when applied can ensure the security of APIs from its design phase even through monitoring during use [4].

4.3.1 Assessing APIs Using The Security Guidelines for Providing and Consuming APIs

The table below provides a classification of the controls from the Security Guidelines for Providing and Consuming APIs, based on different areas of applicability.

S/N	**Controls**	Area of Applicability
1	Asset and API Documentation	Documentation
2	Input Validation decoupled from the application	Code
3	Output encoding	Code
4	Regex Denial of Service	Code
5	Data sanitation	Code
6	Safe Package Usage	Code
7	Storage of Application Secrets - Rotation of secrets, Mode of transfer or delivery	Code and API Management Platform (API MP)
8	Service Authentication: authentication mechanisms/ protocols - API Key, OAuth, JWT	API MP
9	Least Privilege (Access control to API endpoints and HTTP methods based on need only)	API MP
10	Cross Site Request Forgery (CSRF) Protection	API MP
11	Request Rate Limiting	API MP
12	Complete Mediation - Authorization to object and scope	API MP
13	Token Strength: Encryption Algorithm	API MP
14	Protection of Staging Environment	API MP
15	Valid TLS Certificate	API MP
16	SSL/TLS Cipher Suites	API MP
17	Source IP Limitation	API MP
18	Account Lockdown	API MP
19	Exposed Network Interfaces - Open Ports	API MP
20	Session Termination - inactive authentication tokens or API keys	API MP
21	Denial of Service Mitigation	API MP
22	Storing HTTP Access logs - check contents for sensitive data	API MP
23	Error handling (should not disclose any information)	Pentesting results and logs
24	Pentesting and Vulnerability scanning	Review test reports
25	Security-Oriented Code Review	Review code test results in line with organizational policies
26	WEB/API Application Firewall	Firewalls
27	OSI/Packages updates	OS and Packages/libraries update

Table 4.2: API Security Controls from SGPC APIs

Documentation Dependent Control:
Identification of an organization's assets based on priority and use by APIs is vital when evaluating an API. This is especially important if the use of an API would involve fetching the personal data of clients. Proper documentation of assets and API documentation would make the following parts of the evaluation very easy and efficient. It provides details on the various endpoints and methods of the API, as well as its parameters, if any. These would serve as a guide on what to look out for during Code analysis, as

well as assessment through API management platform.

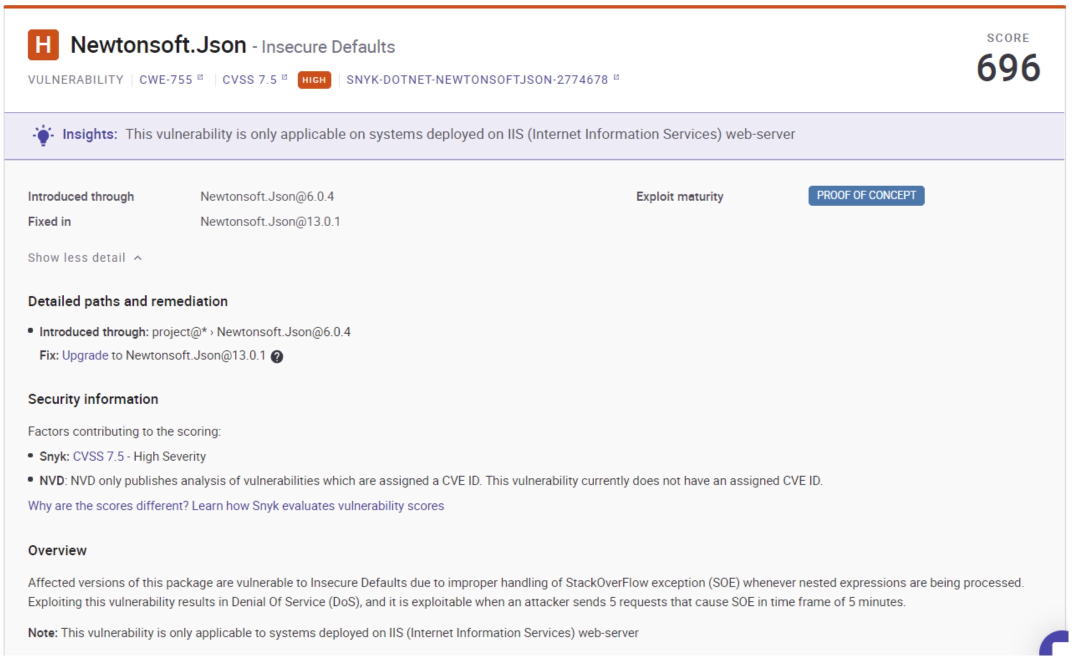
Code Dependent Controls:
Code analysis of APIs is an integral part of API assessment. This is quite so as security is usually not in the picture, during the design phase of APIs. As seen in Table 4.4, some security requirements to check for when analyzing APIs include:

- **Input Validation and Output Encoding**: APIs that take some form of input from a user or database require input validation to guarantee that only the expected data is taken as input. Output encoding, on the other hand, ensures that any input from the user, be it a script from an attacker, does not have any malicious effect [46]. This also helps control vulnerabilities such as injection attacks and Cross site scripting [47].

- **Regex Denial of Service**: The use of regular expression is one way to implement input validation. However, if there is no check on how regular expressions are used, especially when the input text is long, that could lead to a denial of service.

- **Data Sanitation**: This entails an inspection of data returned by APIs, to remove any which may be sensitive [4]. This can be done through code and log analysis. Executing this also requires thorough knowledge of the assets of an organization that the APIs make use of.

- **Safe Package Usage**: There are many public APIs out there used by many. These APIs probably have never been updated since they were published. There is also a possibility that they may have been updated at some point, but that may be a long time ago. As a result, these APIs are running outdated packages. A good example is YouTube Data API [14]. A look at its GitHub repository [15] showed the last commit to it was done in 2018. The repository was tested to check for vulnerabilities. One of the results showed it had a vulnerability called *Insecure Default*. This is the case because the API is using Newtonsoft.Json 6.0.4, a package of the Json.NET framework which was released in August 2014. Figure 4.1 shows the test result.


H Newtonsoft.Json - Insecure Defaults

VULNERABILITY | CWE-755 | CVSS 7.5 | HIGH | SNYK-DOTNET-NEWTONSOFTJSON-2774678

SCORE 696

Insights: This vulnerability is only applicable on systems deployed on IIS (Internet Information Services) web-server

Introduced through Newtonsoft.Json@6.0.4
Fixed in Newtonsoft.Json@13.0.1
Exploit maturity PROOF OF CONCEPT

Show less detail

Detailed paths and remediation

- Introduced through: project@* › Newtonsoft.Json@6.0.4
  Fix: Upgrade to Newtonsoft.Json@13.0.1

Security information

Factors contributing to the scoring:

- Snyk: CVSS 7.5 - High Severity
- NVD: NVD only publishes analysis of vulnerabilities which are assigned a CVE ID. This vulnerability currently does not have an assigned CVE ID.

Why are the scores different? Learn how Snyk evaluates vulnerability scores

Overview

Affected versions of this package are vulnerable to Insecure Defaults due to improper handling of StackOverFlow exception (SOE) whenever nested expressions are being processed. Exploiting this vulnerability results in Denial Of Service (DoS), and it is exploitable when an attacker sends 5 requests that cause SOE in time frame of 5 minutes.

Note: This vulnerability is only applicable to systems deployed on IIS (Internet Information Services) web-server

Figure 4.1: Vulnerability Test on YouTube API detecting an outdated package in use [1]

When assessing APIs, therefore, the APIs should be checked for vulnerabilities associated with outdated packages. These APIs should be examined to ensure they are running the latest patch version available [4].

- **Error Handling**: How APIs respond to, or handle errors should not be overlooked when auditing APIs. Its response to errors should never reveal any information an attacker could exploit [4]. Code Analysis and logs can be used to check for Error handling in APIs.

- **Storage of Application Secrets**: As the name implies, application secrets are *secrets*. Thus, they should be stored securely and not be left exposed. If API keys and client secrets are to be used for any request, their delivery should be over a secure channel [4]. The response body of an API request should not contain any application secret. Assessing this security requirement can be done through code analysis, and from the API management platform. Through analysis, the code can be checked for keywords that would detect if the code contains any sensitive strings. API testing tools can also be used to make API requests and check the request headers and response body for any sensitive strings or secrets. It is recommended that organizations have a rotation of application secrets as part of their security policy. In that regard, API management platforms used by organizations should be checked to see if and how they implement rotation of application secrets [4].

Software testing tools can be used to check for most of the code-dependent controls. Synk was used to test the public APIs used for this project. Some of the results of these tests are in the appendix.

API Management Platform Dependent Controls
API management platforms are used to manage the entire lifecycle of APIs. In effect, they provide a platform for enterprises to more easily develop, design, monitor, test, secure, and analyze APIs [12]. We will now consider an overview assessing these security requirements.

- Service Authentication: The organization should have established its authentication and authorization mechanism. This could be API keys, or protocols such as OAuth, JWT, etc. [4] It should be verified that these mechanisms are properly implemented in the API management Platform.
- Least Privilege: Authorization and access control to API endpoints and HTTP methods should be restricted to just those required to perform a certain operation.
- Cross-Site Request Forgery(CSRF) Protection: For APIs that make use of cookies for authentication and authorization, it is essential to avoid the execution of unwanted actions via the API. Mechanisms that check the value in the cookie to determine if it matches that in the API request, can be used. Additionally, all forms can be checked to verify they are hashed [36].
- Denial of Service and Request Rate Limiting: Lack of control over how an API is used, could result in denial of service. One way to prevent this is by implementing rate limiting on requests. Rate limiting defines the maximum number of requests allowed in a given time interval [9]. The rate limit of an API and its operations should be defined in the organization's security policy [4]. This is then used to check against what is in the API management platform for consistency.
- Complete Mediation: This involves preventing any access to API endpoints without authorization. As such, all API endpoints should have authorization mechanisms. All-access to any resource should be checked for authorization and this should be limited to just the required scope [4].
- Token Strength: Weak encryption algorithms should be avoided. Only strong and state-of-the-art encryption algorithms should be used.
- Protection of Staging Environment: The platforms used to host these APIs in the cloud should be properly set up. If this is not done, this still leaves the APIs vulnerable even when the necessary security measures have been applied to the codes. In this regard, the platforms and any software used should be properly configured, and always patched when necessary, among other things [4].
- Cipher Suites and TLS Certificate: The API management platform should be checked for valid, up-to-date, and signed TLS certificates.

It should be verified TLS is used, as well as strong cipher suites based on the organization's security policy [4].

- Source IP Limitation: To ensure access control and mitigation of denial of service, API calls should be limited to only dedicated IP services if possible [4].
- Account Lockdown and Session Termination: Accounts which are not in use for various reasons, accounts which have tried carrying out malicious actions and authenticating with invalid credentials should be locked, based on a preset timeline [4]. To do this effectively, the API management platform's logs and monitoring system should be checked against the organization's list of current and valid accounts.
- Exposed Network Interfaces: The necessary network ports needed for hosting the API should be defined by the organization. The results of a network scan would reveal if any unnecessary ports are open and exposed.
- Storage of HTTP Access Logs: API calls and requests should be properly logged. It should be verified that these logs do not contain any sensitive data [4].

Some commonly used API management platforms include the *Microsoft Azure API Management Service* and *Apigee* - Google Cloud's API Management platform. The Microsoft Azure API Management platform consists of an API gateway, a management plane, and a developer portal. APIs and other management services are defined and configured in the management plane, while the API gateway enforces the services and policies defined in the management plane [28]. In assessing APIs using the Azure API Management platform, our emphasis would be mainly on the API gateway and management plane.
Apigee is also a platform for building and managing APIs. With Apigee, API proxies can be created to manage and secure APIs and communication between client Applications and the server [11].

In auditing Cloud APIs through Management platforms, we will map some controls of the CCM V4.0.7 to specific controls of Security Guidelines for Providing and Consuming APIs.

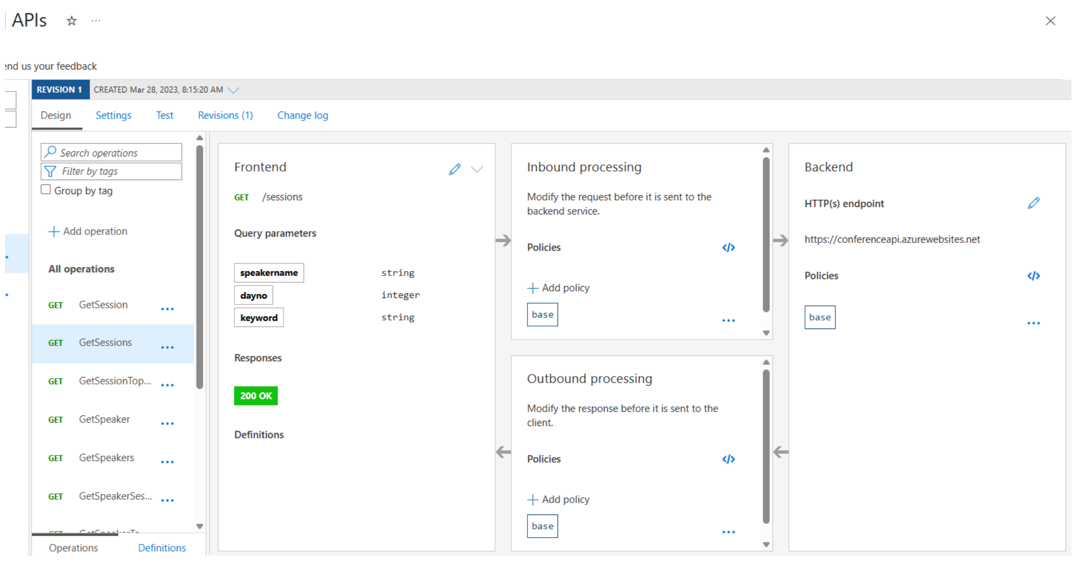
CCM Control	SGPC APIs Control
AIS - 01	Least Privilege (Access Control) Complete Mediation Denial of Service Mitigation Request Rate Limiting
CCC - 01	Storage of Application Secrets (Rotation of Secrets) Account Lockdown OSI/ Packages update
CEK - 03	Storage of Application Secrets
CEK - 04	Token Strength: Encryption Algorithm
DSP - 01	Asset and API Documentation
DSP - 04	Asset Documentation
DSP - 05	Asset and API Documentation
IVS - 03	Exposed Network Interfaces - Open Ports Valid TLS Certificate SSL/TLS Cipher Suites
IVS - 04	Firewalls

Table 4.3: Mapping of CCM V4.0.7 Controls To Security Guidelines For Providing and Consuming APIs

Having mapped these controls, we will now consider in detail how to audit Cloud APIs from management platforms based on these controls. The order of presentation is based on how each control relates to the other

DSP - 01: Security and Privacy Policy and Procedures
Asset and API documentation is a point of reference when evaluating APIs for this control. The auditor should identify and ensure that the organization has policies on classification, protection, and handling of data throughout its lifecycle [8]. These relevant documentation should be examined to establish if they are compliant with local regulations on data protection.

DSP - 04: Data Classification
The data used by the APIs should be checked to ensure that they are properly classified according to the data privacy policies and procedures established by the organization. There are likely to be changes in the data used and its sensitivity level with time. Thus, it should be checked that these classifications are reviewed and updated frequently. The implementation of these classifications can be examined in the Azure API management platform as well.



speakername	string
dayno	integer
keyword	string

Figure 4.2: Azure API Operations and Policies

As seen in Figure 4.2, the operation *GetSessions* of *Demo Conference API* has three parameters of which response value is expected. The classification of these data should be in the data privacy policies. When evaluating these data and their implementation based on data classification, it should be checked that the policies on this operation and those who are allowed access to it reflect its sensitivity level.

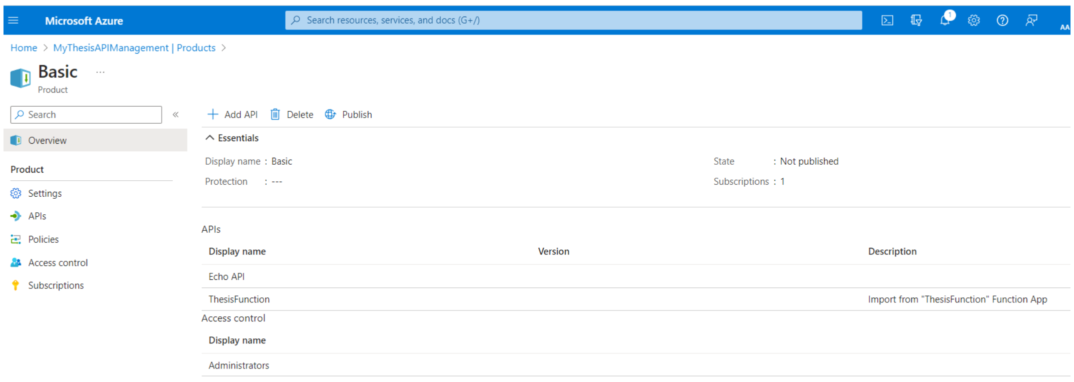
DSP - 05: Data Flow Documentation
This involves a check of the availability of data flow documentation. It should be assessed to ensure it is complete, accurate, and in line with the API's data security and privacy policies as well as the flow of data when using the API [8].

AIS - 01: Application and Interface Security Policies
This entails examining documentation on policies and procedures for maintaining API security, against the policies in the API management service. There should be clear documentation on

1. Users and/or accounts who are allowed access to an API or API service and their roles.
2. authorized access to individual APIs and their respective operations.
3. Allowed or accepted number of requests that can be made to an API endpoint within a specified time interval
4. IP addresses that are allowed to communicate with the API service, make API calls, or request to a certain endpoint.

The Microsoft Azure API management provides policies to help implement this. The policies in MS Azure management could have the following scopes, namely:

1. Global Policies: These policies cover all APIs in the management service.
2. Product Policies: Product policies apply to all APIs in a particular product. Figure 4.3 shows a product with two(2) APIs: *Echo API and ThesisFunction*. The policies applied to it show that the APIs can only be accessed by administrators, and would need a subscription to access.
3. API: These policies cover all operations in a given API.
4. Operation: Operation policies are limited to just specified operations in an API.



Display name : Basic
Protection : ---
State : Not published
Subscriptions : 1

APIs

Display name	Version	Description
Echo API		
ThesisFunction		Import from "ThesisFunction" Function App

Access control

Display name
Administrators

Figure 4.3: Product Policies in Azure API Management [2]

The figures below show an API management service with two APIs: *Demo Conference API* and *Echo API*. Each shows specific areas to check on the implementation of API policies.

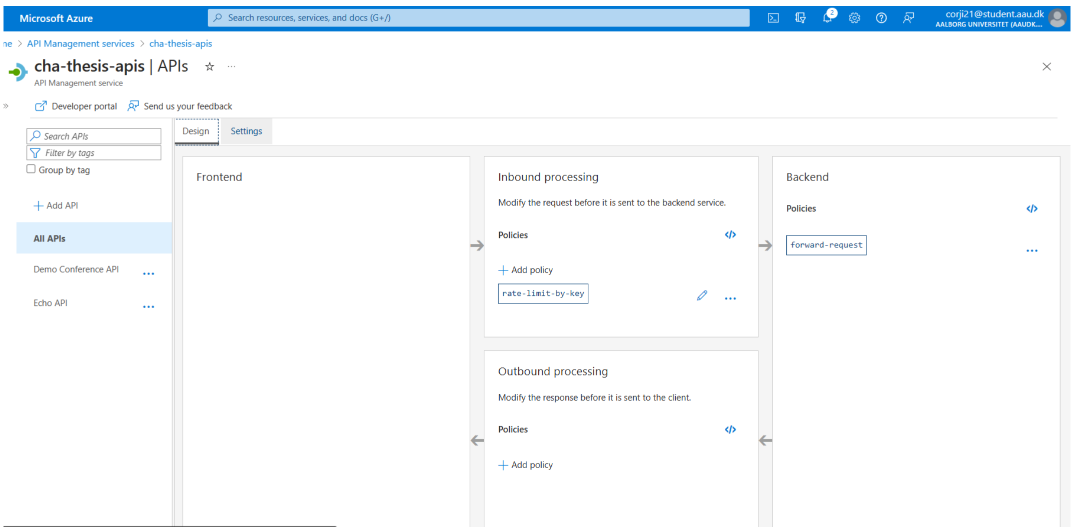


Figure 4.4: Global API Policies

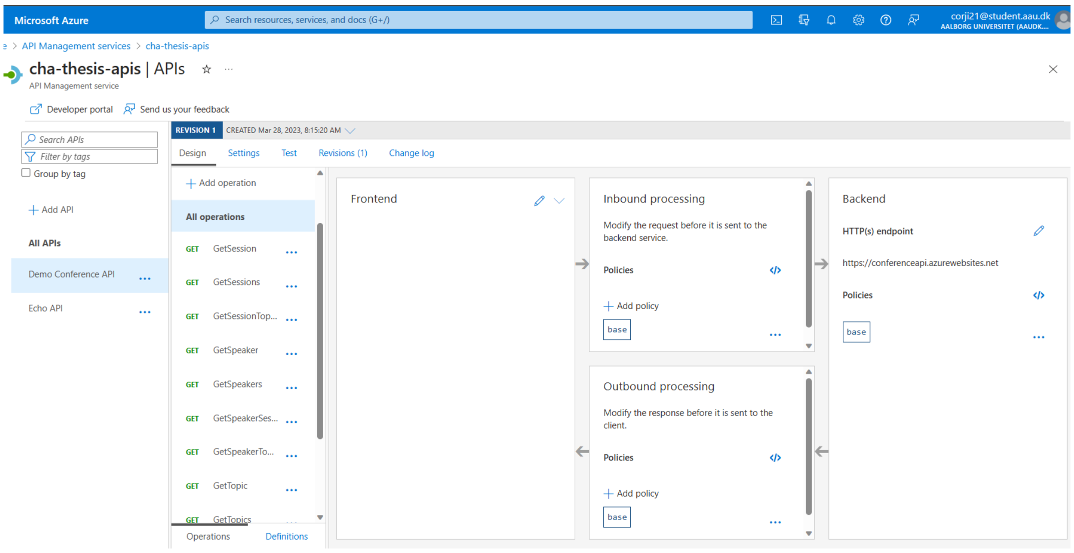
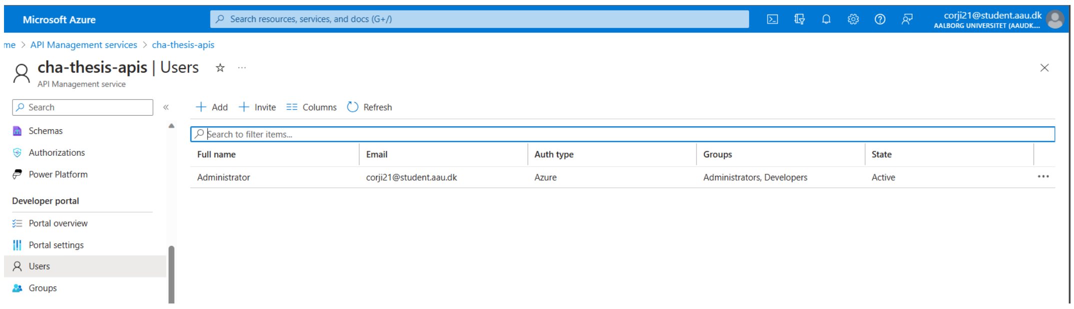


Figure 4.5: API scoped Policies

As shown in the figures above, policies applied to API can vary in scope. The policies in Figure 4.4 apply to all APIs in that Management service. On the other hand, policies in Figure 4.5 can apply to all operations in the Demo Conference API or a specific operation. Therefore, in auditing cloud APIs for Application and Interface Security Policies, the scopes of the policies should be checked to ensure they are in harmony with the documentation.

CCC - 01: Change Management Policy and Procedure

This would involve reviewing changes to an organization's assets, application, systems infrastructure, and configurations [8]. An organization should have policies on the rotation of access keys and other application secrets, as well as how often this should be done. There should be documentation on the set timeline of accounts, especially when the user of such an account quits or no longer uses it for various reasons [4]. These should be compared to what is obtainable from the API management service for uniformity. Figure 4.6 below shows the users in an API management service. Auditing an API management service based on Change Management Policy and Procedures would require that the organization provide a list of all current users of the API service. This should be checked against the active users on the API service, to determine if there is any user or account not in use that is still active.



Full name	Email	Auth type	Groups	State
Administrator	corji21@student.aau.dk	Azure	Administrators, Developers	Active

Figure 4.6: Users In an Azure API Management Service.

As part of the CCC-01 control, the API management service, and all software, packages, and libraries in use should be examined to see if they are updated.

CEK - 03: Data Encryption
Cloud APIs can be examined for data encryption both in transit and at rest. For data in transit, logs and HTTP response to API requests should be inspected for any presence of application secrets or encryption keys. Azure API management platform also allows for the tracing of subscription keys. Enabling the tracing of these keys can make examining the encryption of data in transit much easier. Storage locations of applications and API secrets should be inspected to determine if they are secure.

CEK - 04: Encryption Algorithm
The APIs and their management platform should be examined to ensure that the encryption algorithm used is state-of-the-art and strong [4]. In the Azure API management platform, for example, a good number of these encryption algorithms are pre-activated, both weak and strong alike. It is good practice to check that no weak encryption algorithms are in use.

IVS - 03: Network Security
Communication with cloud services is enabled through network ports and interfaces. Misconfiguration and exposure of these interfaces and ports can leave the network open and vulnerable to attackers. The bad actors could exploit this to gain access to sensitive and unauthorized data, modify or delete certain information and possibly steal them for selfish gains. Therefore, when auditing cloud APIs, the network should be checked for any open ports which may not really be needed. The service should also be examined for the use of valid TLS Certificates and strong cipher suites [4].

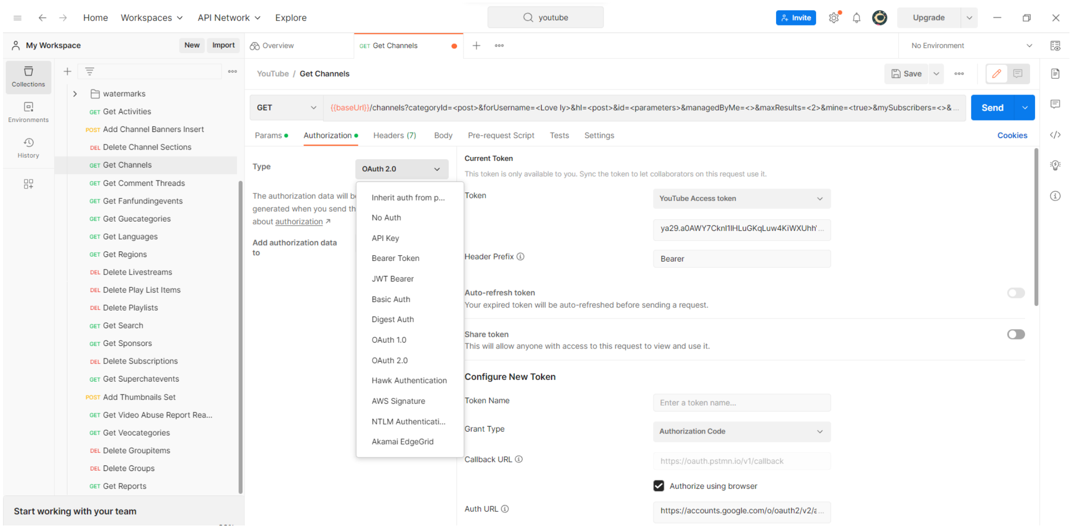
IVS - 09: Network Defense
Certain restrictions are essential in a network to maintain confidentiality, integrity, and availability of data and assets in a network. For Cloud APIs, a Web/API Application Firewall can prove very helpful in achieving this goal. Google has a Cloud Firewall [10] which can be used with Apigee to protect API connectivity. However, not all Web Application firewalls are compatible with APIs [4]. But other products can be used to cater to this need in some way. One such is Microsoft Defender for cloud [29].

4.3.2 API Testing Tools

API testing is an important aspect of the API lifecycle. It provides a means to determine if an API is working as it should. There are quite several API testing tools used by many. From a list of commercial API testing tools and platforms, 4 were picked at random. A poll was made to find out the most preferred by many. Postman and APISec had the most preference, with 55% and 27% respectively. Thus, we will consider how these tools can prove useful in the testing and security assessment of APIs.

Postman
The Postman platform can be used for performance testing of APIs. It tests APIs based on requests and expected responses. This is done in two(2) ways: Unit testing and End-to-End testing. Unit testing allows for the testing of an individual endpoint in an API collection. End-to-End testing enables the testing of all endpoints in an API collection one after the other.

For APIs that require authorization and authentication, these mechanisms can be tested to ensure they are working properly as should.



watermarks
GET Get Activities
POST Add Channel Banners Insert
DEL Delete Channel Sections
GET Get Channels
GET Get Comment Threads
GET Get Fanfundingevents
GET Get Guecategories
GET Get Languages
GET Get Regions
DEL Delete Livestreams
DEL Delete Play List Items
DEL Delete Playlists
GET Get Search
GET Get Sponsors
DEL Delete Subscriptions
GET Get Superchatevents
POST Add Thumbnails Set
GET Get Video Abuse Report Rea...
GET Get Veocategories
DEL Delete Groupitems
DEL Delete Groups
GET Get Reports

YouTube / Get Channels

GET {{baseUrl}}/channels?categoryId=<post>&forUsername=<Love ly>&hl=<post>&id=<parameters>&managedByMe=<>&maxResults=<2>&mine=<true>&mySubscribers=<>&...

Type: OAuth 2.0

The authorization data will be generated when you send th about authorization

Add authorization data to

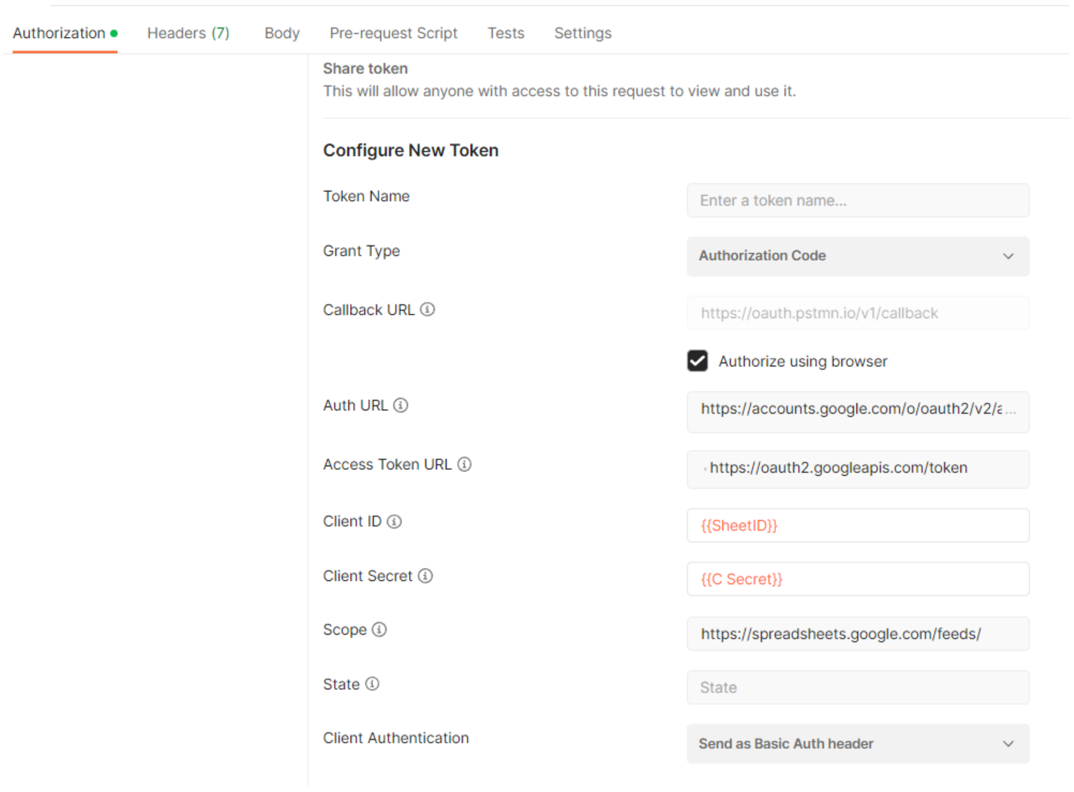
Current Token
This token is only available to you. Sync the token to let collaborators on this request use it.
Token: YouTube Access token
ya29.a0AWY7CknI1lHLuGKqLuw4KiWXUhh'...
Header Prefix: Bearer
Auto-refresh token
Your expired token will be auto-refreshed before sending a request.
Share token
This will allow anyone with access to this request to view and use it.

Configure New Token
Token Name: Enter a token name...
Grant Type: Authorization Code
Callback URL: https://oauth.pstmn.io/v1/callback
☑ Authorize using browser
Auth URL: https://accounts.google.com/o/oauth2/v2/a...

Figure 4.7: Authorization mechanisms and Protocols in Postman [3]

The figure above shows the various authorization mechanisms Postman supports. An enterprise can set up its API collection and test its authorization to ensure it works as it should. This feature can also aid in security testing when it comes to authentication. The results of an attempt at authorization and authentication can be checked against the organization's policies on authorization and authentication to see if it is working as expected.

In obtaining access tokens for APIs, application secrets such as Client ID and Client Secret would need to be supplied. This is seen in the figure below



Share token
This will allow anyone with access to this request to view and use it.

Configure New Token

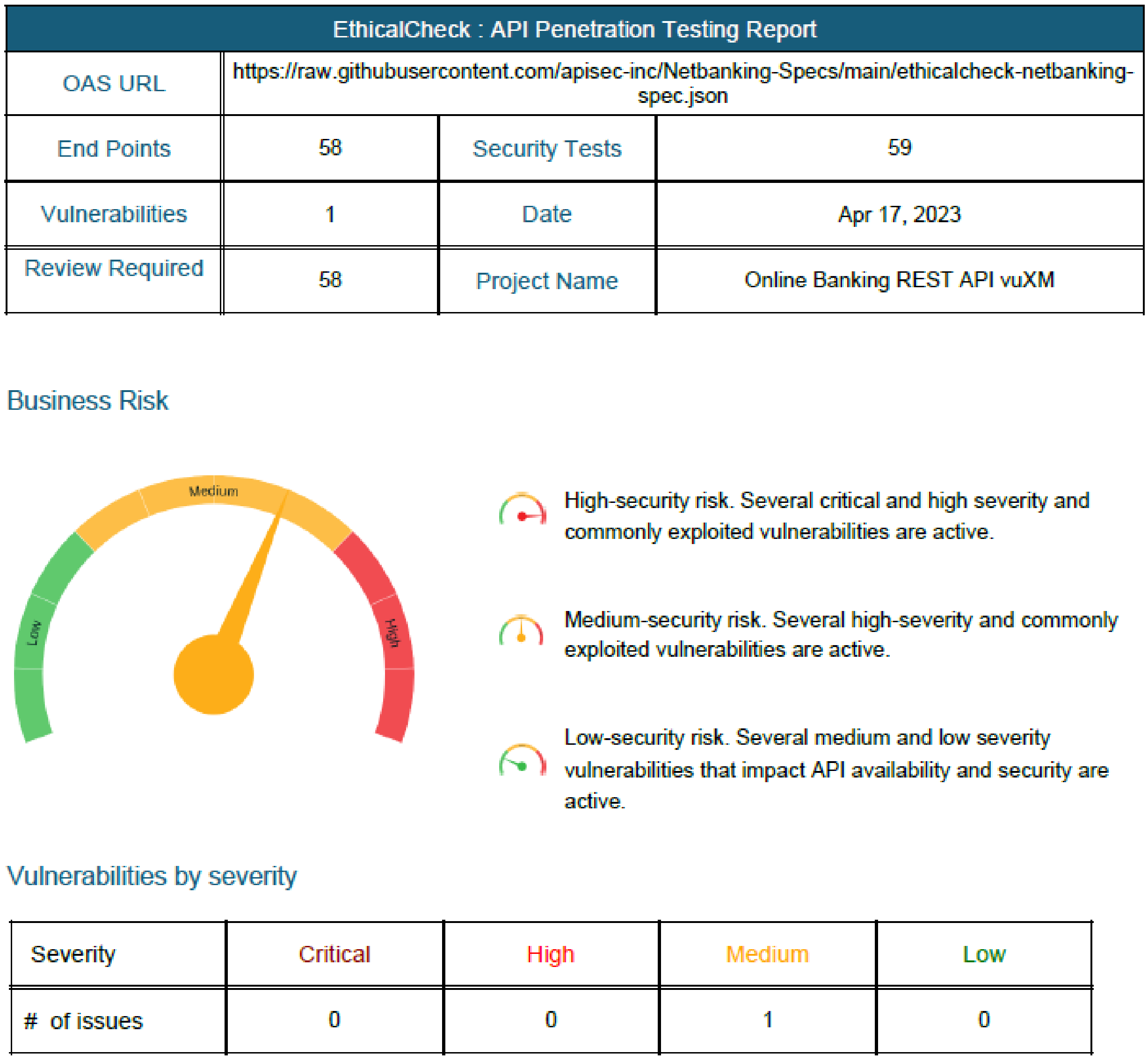
Token Name	Enter a token name...
Grant Type	Authorization Code
Callback URL	https://oauth.pstmn.io/v1/callback
	☑ Authorize using browser
Auth URL	https://accounts.google.com/o/oauth2/v2/a...
Access Token URL	https://oauth2.googleapis.com/token
Client ID	{{SheetID}}
Client Secret	{{C Secret}}
Scope	https://spreadsheets.google.com/feeds/
State	State
Client Authentication	Send as Basic Auth header

Figure 4.8: Use of Application Secrets in Postman

In configuring access tokens for an API in Postman, the client ID and client secret can be supplied directly into the specified box or set as a variable. Application secrets such as these, when input directly to the new token form could likely be exposed, and used for malicious purposes. This can be used to test for another security control, Control No. 7 in Table 4.4: Storage of Application Secrets - Mode of transfer or delivery.
Other nice features of Postman are the Pre-request Scripts and Test scripts. The Pre-request scripts are very useful for analyzing the codes before API requests are sent. This can be used, for example, to search for a variable in the code before the API request is sent. The Test scripts, on the other hand, run tests scripts after the execution of API request [51]. These tests vary from getting variables, the status code of the response to the API request, to searching the response body for a string. These features can be used to check the API requests and responses for sensitive data such as application

secrets, and also see how the APIs handle errors.
The interesting thing about using test scripts is that they can be customized by an organization to suit its needs and security criteria based on the organization's policies.

APISec
APISec is an API solution that provides automated testing for APIs. Its API testing tool EthicalCheck is used in testing API. To test APIs using this tool, the user needs to supply its Open-API Specification URL or a Postman URL. The EthicalCheck Engine invokes the API and runs tests on it, scanning for the OWASP API Top Ten list. On successful completion of the process, a report is generated, detailing the outcome of the scan. The image below shows a summary of the scan result of an Online Banking REST API.



EthicalCheck : API Penetration Testing Report			
OAS URL	https://raw.githubusercontent.com/apisec-inc/Netbanking-Specs/main/ethicalcheck-netbanking-spec.json		
End Points	58	Security Tests	59
Vulnerabilities	1	Date	Apr 17, 2023
Review Required	58	Project Name	Online Banking REST API vuXM

Business Risk

High-security risk. Several critical and high severity and commonly exploited vulnerabilities are active.

Medium-security risk. Several high-severity and commonly exploited vulnerabilities are active.

Low-security risk. Several medium and low severity vulnerabilities that impact API availability and security are active.

Vulnerabilities by severity

Severity	Critical	High	Medium	Low
# of issues	0	0	1	0

Figure 4.9: APISec API Scan Result

As evident in the scan result above, the APISec scanner discovers all endpoints associated with the API to be tested. With this discovery, the methods in each endpoint are all tested for vulnerabilities. The vulnerabilities discovered are classified based on the level of severity and their impact on the business. This classification of discovered vulnerabilities would help an organization determine what areas to pay more attention to and prioritize how to remedy the associated risks.

4.4 Evaluation of A Public API Following This Extensive Approach

An attempt was made in assessing a Public API using the extensive approach outlined in earlier sections of this chapter. The assessed API is a Simple Inventory API. It is a public API with an Open API specification available for use on SwaggerHub. This API was assessed using the Azure API Management Platform and Postman. It is assumed that an organization is

- using this API to keep stock of its physical assets.
- using this API as built by the developer without any added security measures

The table below provides the results of the assessment of this API using the controls from *Security Guidelines For Providing and Consuming APIs*. The remark "Y" means a particular control is implemented correctly, while "N" signifies it is not implemented. On the other hand, when "N/A" is used it means a certain command is not applicable.

S/N	**Controls**	Remark(Y,N,N/A)
1	Asset Documentation API Documentation	N/A Y
2	Input Validation decoupled from the application	N
3	Output encoding	Y
4	Regex Denial of Service	N/A
5	Data sanitation	N
6	Safe Package Usage	N/A
7	Storage of Application Secrets - Rotation of secrets, Mode of transfer or delivery	Y
8	Service Authentication: authentication mechanisms/ protocols - API Key, OAuth, JWT	N
9	Least Privilege (Access control to API endpoints and HTTP methods based on need only)	N
10	Cross-Site Request Forgery (CSRF) Protection	N/A
11	Request Rate Limiting	N/A
12	Complete Mediation - Authorization to object and scope	N
13	Token Strength: Encryption Algorithm	API MP
14	Protection of Staging Environment	N/A
15	Valid TLS Certificate	API MP
16	SSL/TLS Cipher Suites	API MP
17	Source IP Limitation	N
18	Account Lockdown	N
19	Exposed Network Interfaces - Open Ports	N/A
20	Session Termination - inactive authentication tokens or API keys	N
21	Denial of Service Mitigation	N
22	Storing HTTP Access logs - check contents for sensitive data	Y
23	Error handling (should not disclose any information)	N
24	Pentesting and Vulnerability scanning	Review test reports
25	Security-Oriented Code Review	Review code test results in line with organizational Policies
26	WEB/API Application Firewall	N/A
27	OSI/Packages updates	N/A

Table 4.4: API Security Controls from SGPC APIs

Assessment of the Simple Inventory API showed using Azure API Management Service showed

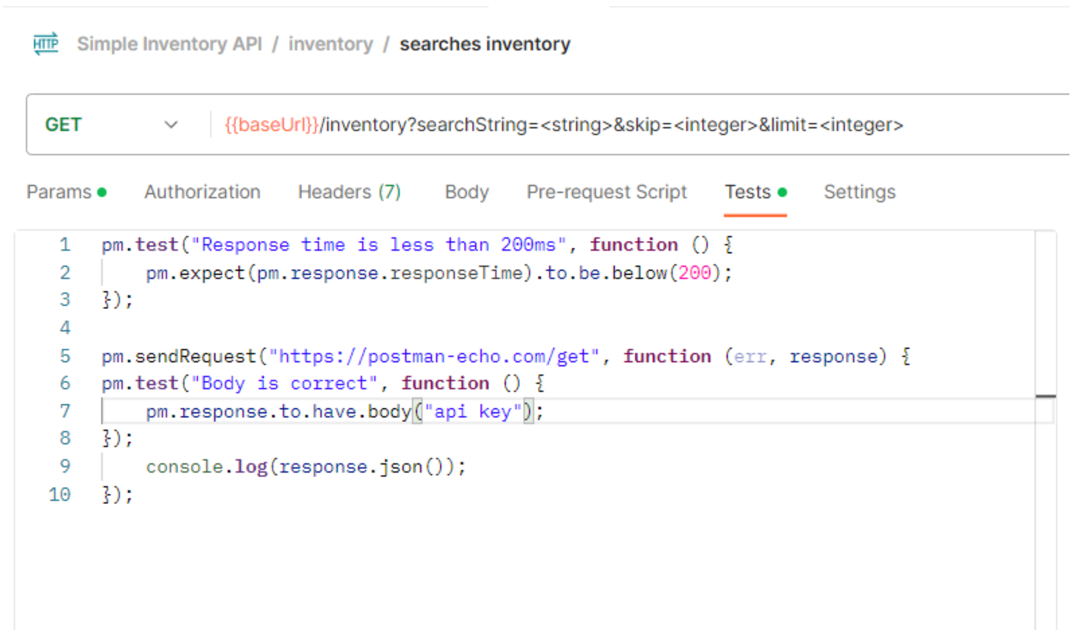
- There were no access control implemented on the API
- The absence of complete mediation - specifying authorization to APIs' object and scope

- There were no policies implementing request rate limiting, source IP limitation, Denial of service mitigation, etc

The Postman tool was also used to test and assess this API. The results of the assessment are as follows

1. Output encoding was incorporated in the API.
2. There were no application secrets in the API request or response body
3. Content headers were also free of any sensitive data.
4. A view of the raw form of the requests and results showed that cookies were not used in sending the server's response to the API Call. Hence, the API is protected from Cross-Site Request Forgery.

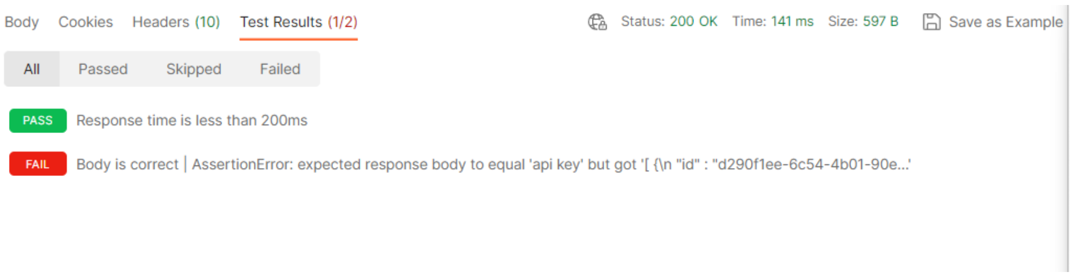
The figures below show some of the screenshots from Postman



{{baseUrl}}/inventory?searchString=<string>&skip=<integer>&limit=<integer>

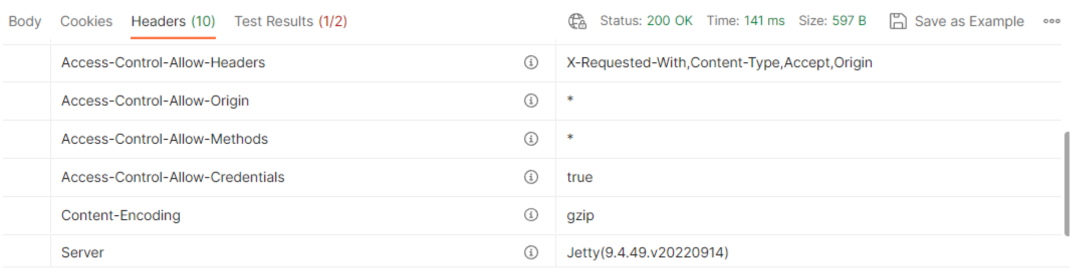
```
pm.test("Response time is less than 200ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(200);
});

pm.sendRequest("https://postman-echo.com/get", function (err, response) {
pm.test("Body is correct", function () {
    pm.response.to.have.body("api key");
});
    console.log(response.json());
});
```

Figure 4.10: Simple Inventory API Test Scripts



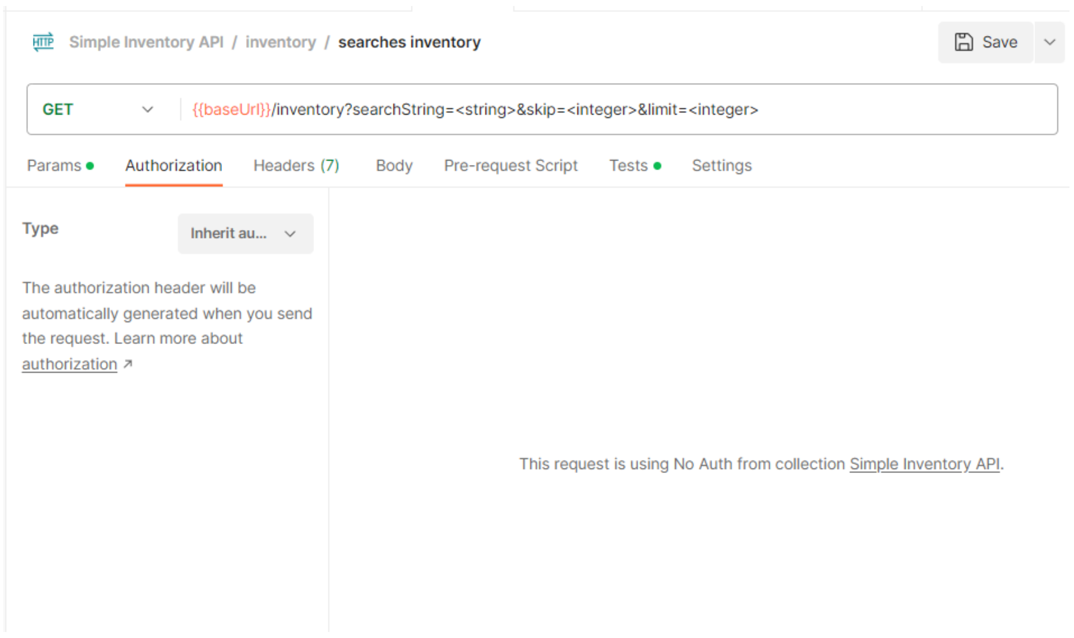
```
PASS  Response time is less than 200ms
FAIL  Body is correct | AssertionError: expected response body to equal 'api key' but got '[ {\n "id" : "d290f1ee-6c54-4b01-90e...'
```

Figure 4.11: Simple Inventory API Test Results



Access-Control-Allow-Headers	X-Requested-With,Content-Type,Accept,Origin
Access-Control-Allow-Origin	*
Access-Control-Allow-Methods	*
Access-Control-Allow-Credentials	true
Content-Encoding	gzip
Server	Jetty(9.4.49.v20220914)

Figure 4.12: Simple Inventory API Headers



```
GET {{baseUrl}}/inventory?searchString=<string>&skip=<integer>&limit=<integer>
```

Type: Inherit au...

The authorization header will be automatically generated when you send the request. Learn more about authorization ↗

This request is using No Auth from collection Simple Inventory API.

Figure 4.13: Simple Inventory API Authorization

The security controls with the remark "N/A", "Review test reports" and "Review code test results in line with organizational Policies" were not applicable in this assessment as the security controls can only be examined in a real API system, not a test system. Also, it was a challenge integrating this API into the software testing tool Snyk. However, vulnerability scanning of the YouTube Data API was achieved with the software testing tool Snyk. A Screenshot of the test result was presented earlier in this chapter. Other screenshots of the test results are in the appendix.

Chapter 5

Discussion and Conclusion

This chapter discusses at length further details on the approach taken to assess Cloud APIs in this report. It also highlights areas that could benefit from more research in the future. The chapter concludes with a summary of the report and how the research questions were answered in this report.

5.1 Discussion

5.1.1 Proposed API Assessment Method.

The API audit system proposed by Sun et.al mentioned earlier in related works, is based on data analysis of captured internet traffic [54]. As effective as that may sound, it can be time-consuming as well. It would take a lot of time to analyze captured network, especially when that is done manually. Additionally, this system of audit is focused more on the identification of APIs and vulnerabilities through the data flow of API assets. Basing security audits only on assets can be limiting. How quickly can an unauthorized attempt by an insider, from a legitimate source IP be detected using this system? A lot can go wrong and may probably not be captured by internet traffic.

The proposed API assessment method applied in this report, therefore focuses more on tools and platforms commonly used with APIs today. The Cloud Control Matrix and the Security Guidelines for Providing and Consuming APIs, all created by the Cloud Security Alliance were used for this project. The specifics of the security controls in the Security Guidelines for providing and Consuming APIs make it a good reference, for audit criteria when accessing APIs. However, the choice of audit criteria is always dependent on the needs of the organization requiring an audit.

API Management Tools
In highlighting the vulnerabilities of APIs, Sun et. al made mention of the inability of API management tools and platforms to handle authentication and authorization [54]. However, assessment of APIs using these tools has shown that APIs managed with these tools are more vulnerable to attacks if the tools are not properly configured. Thus any organization employing the use of these tools should first gain a proper understanding of these tools and their features to properly configure them. The same applies to the auditing party.
As noted by Sun et. al, attackers usually access APIs with legal/authorized credentials and then simulate normal operations [54]. This makes the input validation and access control implemented in code, as well as security mechanisms provided by traditional API security gateways insufficient. Nevertheless, with the use of dedicated IP addresses and the setting of quota limits on APIs and API operations, this can be controlled. This is more efficient for organizations making use of internal APIs at work. Assuming each staff has a dedicated work computer and is only allowed to work from it, and just during official working hours, any access to the API is limited. Thus, if that organization already has policies on this, comparisons of how these are implemented in the API management platform against the policies is a good way to evaluate the APIs. By doing this, an organization can also ensure that Availability, as a security property is maintained by its API.

API Testing and Testing Tools
API testing remains an invaluable aspect of API auditing. Many approaches are applied. Many tools are used as well. Manual testing or code analysis of API is still used by some today. But the choice of API testing tool to be used can greatly affect the quality of tests to be conducted. This is reflected in the API testing tools used in this project: Postman and APISec. Both tools handle testing in different ways. Postman allows for performance testing of APIs. It allows for flexibility in testing to suit the needs of an organization. It can be used to test if an API has authorization and authentication mechanisms or not. Its built-in feature "Test" can be used to perform certain tests on responses to API requests. These tests range from searching the response body for certain variables, searching the content header for any sensitive data, checking the response time for API requests, the status code, and much more. However, it was observed with the use of APISec that testing is automated. It tests APIs based on the OWASP Top 10 for API. The entire testing process is automated. Nevertheless, the results obtained are limited as its free API Scan engine only scans APIs for a few of the Top 10 API Vulnerabilities. The paid version of APISec offers full access to the tool's scanning capabilities. The only limitation is its cost as it is really expensive. Thus due to a lack of funding, we opted to use another software

testing tool.

Another delimitation of this project is that the assessment and testing of APIs were restricted to a test environment only. As such, some of the security controls were not fully tested and assessed. This is because they can only be assessed in a real working system.

5.2 Conclusion

It has been established that security is not taken into consideration in the design of APIs. With the rise in data breaches experienced in the use of API, an API audit is necessary to review how organizations implement security measures in API. It has also been seen that security vulnerabilities such as Broken Object Level Authentication, Broken function-level authentication, and Security misconfiguration are some of the vulnerabilities common to APIs. When adequate measures are not put in place, exploitation of these vulnerabilities can come in the way of confidentiality, integrity, and availability of an organization's assets and its API system. The Security Guidelines for Providing and Consuming APIs [4] provides security controls that when implemented can help ensure the confidentiality, integrity, and availability of APIs. An extensive approach to how best these controls can be used to assess cloud APIs has also been considered.
Assessment of the Business Logic of an API would be a great topic to consider for further research on this subject

Bibliography

[1] Akamai. *Akamai State of the Internet Security Report: Retailers Most Common Credential Stuffing Attack Victim; Points to Dramatic Rise in API Traffic as Key Trend*. Accessed: 2023-03-02. URL: https://www.akamai.com/us/en/about/news/press/2019-press/state-of-the-internet-security-retail-attacks-and-api-traffic.jsp.

[2] Cloud Security Alliance. *A New Resource for API Security Best Practices*. Accessed: 2023-03-10. URL: https://cloudsecurityalliance.org/blog/2021/04/30/a-new-resource-for-api-security-best-practices/.

[3] Cloud Security Alliance. *Security Guidelines for Providing and Consuming APIs*. Accessed: 2023-03-10. URL: https://cloudsecurityalliance.org/artifacts/security-guidelines-for-providing-and-consuming-apis/.

[4] Cloud Security Alliance. *Security Guidelines for Providing and Consuming APIs*. 2021. URL: https://cloudsecurityalliance.org/artifacts/security-guidelines-for-providing-and-consuming-apis/.

[5] Nordic APIs. *6 Types of APIs: Open, Public, Partner, Private, Composite, Unified*. Accessed: 2023-05-26. URL: https://nordicapis.com/6-types-of-apis-open-public-partner-private-composite-unified/.

[6] Vaggelis Atlidakis, Patrice Godefroid, and Marina Polishchuk. “Checking Security Properties of Cloud Service REST APIs”. In: *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. 2020, pp. 387–397. DOI: 10.1109/ICST46399.2020.00046.

[7] Livia Maria Brumă. “Cloud security audit – issues and challenges”. In: *2021 16th International Conference on Computer Science & Education (ICCSE)*. 2021, pp. 263–266. DOI: 10.1109/ICCSE51940.2021.9569654.

[8] Cloud Security Alliance. *Cloud Controls Matrix (CCM)*. en. fFramework. 2021. URL: https://cloudsecurityalliance.org/artifacts/cloud-controls-matrix-v4/.

[9] IBM Corporation. *Understanding rate limits for APIs and Plans*. Accessed: 2023-05-15. URL: https://www.ibm.com/docs/en/api-connect/10_reserved_instance?topic=connect-understanding-rate-limits-apis-plans.

[10] Google. *Cloud Firewall*. Accessed: 2023-05-22. URL: https://cloud.google.com/firewall#section-1.

[11] Google. *Understanding APIs and API proxy*. Accessed: 2023-05-16. URL: https://cloud.google.com/apigee/docs/api-platform/fundamentals/understanding-apis-and-api-proxies.

[12] Google. *What is API Management?* Accessed: 2023-05-12. URL: https://cloud.google.com/learn/what-is-api-management?.

[13] Google. *What is cloud architecture?* Accessed: 2023-05-22. URL: https://cloud.google.com/learn/what-is-cloud-architecture#section-6.

[14] Google. *YouTube Data API v3*. URL: https://console.cloud.google.com/apis/library/youtube.googleapis.com?project=psyched-choir-377513.

[15] Google. *YouTube Data API v3*. URL: https://github.com/youtube/api-samples.

[16] HubSpot. *4 Types of APIs All Marketers Should Know*. Accessed: 2023-05-26. URL: https://blog.hubspot.com/website/types-of-apis.

[17] HubSpot. *REST API (Introduction)*. Accessed: 2023-05-26. URL: https://www.geeksforgeeks.org/rest-api-introduction/.

[18] Fatima Hussain et al. "Enterprise API Security and GDPR Compliance: Design and Implementation Perspective". In: *IT Professional* 22.5 (2020), pp. 81–89. DOI: 10.1109/MITP.2020.2973852.

[19] Texas Department of Insurance. *Notice of Data Security event*. Accessed: 2023-03-25. URL: https://www.tdi.texas.gov/data-security-event/index.html.

[20] ISO. *Standards*. Accessed: 2023-05-02. URL: https://www.iso.org/standards.html.

[21] ISO Central Secretary. *Guidelines for auditing management systems*. en. Standard. Geneva, CH, 2018. URL: https://www.iso.org/standard/70017.html.

[22] ISO Central Secretary. *Information technology – Part 1: IT asset management systems – Requirements*. en. Standard. Geneva, CH, 2017. URL: https://www.iso.org/standard/68531.html.

[23] Bahruz Jabiyev et al. "Preventing Server-Side Request Forgery Attacks". In: *Proceedings of the 36th Annual ACM Symposium on Applied Computing*. SAC '21. Virtual Event, Republic of Korea: Association for Computing Machinery, 2021, 1626–1635. ISBN: 9781450381048. DOI: 10.1145/3412841.3442036. URL: https://doi.org/10.1145/3412841.3442036.

[24] Snyk Limited. *Snyk*. URL: https://www.nist.gov/cyberframework/framework.

[25] Suryadipta Majumdar et al. "Cloud security auditing: Major approaches and existing challenges". In: *Foundations and Practice of Security: 11th International Symposium, FPS 2018, Montreal, QC, Canada, November 13–15, 2018, Revised Selected Papers 11*. Springer. 2019, pp. 61–77.

[26] Suryadipta Majumdar et al. “User-Level Runtime Security Auditing for the Cloud”. In: *IEEE Transactions on Information Forensics and Security* 13.5 (2018), pp. 1185–1199. DOI: 10.1109/TIFS.2017.2779444.

[27] Anne McCormick. *The Top 5 API Security Breaches in 2022, and How to Avoid Them in 2023*. Accessed: 2023-03-11. URL: https://techblog.cisco.com/blog/top-5-api-security-breaches-in-2022.

[28] Microsoft. *Azure API Management*. Accessed: 2023-05-16. URL: https://learn.microsoft.com/en-us/azure/api-management/api-management-key-concepts.

[29] Microsoft. *Microsoft Defender for Cloud documentation*. Accessed: 2023-05-22. URL: https://learn.microsoft.com/en-us/azure/defender-for-cloud/.

[30] Microsoft. *What Is Cloud Computing?* Accessed: 2023-05-22. URL: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-cloud-computing.

[31] SUNANDHINI MURALIDHAR. *Simple Inventory API*. URL: https://app.swaggerhub.com/apis/SUNANDHINIMURALIDHAR/InventoryAPI/1.0.0.

[32] NIST. *Framework Documents*. Accessed: 2023-05-02. URL: https://www.nist.gov/cyberframework/framework.

[33] NIST. *NIST SP 800-145, The NIST Definition of Cloud Computing*. Accessed: 2023-05-22. URL: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf.

[34] Minjie Ou, Liming Wang, and Hao Xun. “Deaps: Deep learning-based user-level proactive security auditing for clouds”. In: *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE. 2019, pp. 1–6.

[35] Michiel Overeem, Max Mathijssen, and Slinger Jansen. “API-m-FAMM: A focus area maturity model for API Management”. In: *Information and Software Technology* 147 (2022), p. 106890. ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2022.106890. URL: https://www.sciencedirect.com/science/article/pii/S0950584922000532.

[36] OWASP. *Cross Site Request Forgery (CSRF)*. Accessed: 2023-05-15. URL: https://owasp.org/www-community/attacks/csrf.

[37] OWSAP. *API10:2023 Unsafe Consumption of APIs*. Accessed: 2023-03-24. URL: https://github.com/OWASP/API-Security/blob/master/2023/en/src/0xaa-unsafe-consumption-of-apis.md.

[38] OWSAP. *API1:2023 Broken Object Level Authorization*. Accessed: 2023-03-25. URL: https://github.com/OWASP/API-Security/blob/master/2023/en/src/0xa1-broken-object-level-authorization.md.

[39] OWSAP. *API2:2023 Broken Authentication*. Accessed: 2023-03-11. URL: https://github.com/OWASP/API-Security/blob/master/2023/en/src/0xa2-broken-authentication.md.

[40] OWSAP. *API3:2023 Broken Object Property Level Authentication*. Accessed: 2023-03-11. URL: https://github.com/OWASP/API-Security/blob/master/2023/en/src/0xa3-broken-object-property-level-authorization.md.

[41] OWSAP. *API4:2023 Unrestricted Resource Consumption*. Accessed: 2023-03-25. URL: https://github.com/OWASP/API-Security/blob/master/2023/en/src/0xa4-unrestricted-resource-consumption.md.

[42] OWSAP. *API6:2023 Server Side Request Forgery*. Accessed: 2023-03-24. URL: https://github.com/OWASP/API-Security/blob/master/2023/en/src/0xa6-server-side-request-forgery.md.

[43] OWSAP. *API7:2023 Security Misconfiguration*. Accessed: 2023-03-24. URL: https://github.com/OWASP/API-Security/blob/master/2023/en/src/0xa7-security-misconfiguration.md.

[44] OWSAP. *API8:2023 Lack of Protection from Automated Threats*. Accessed: 2023-03-24. URL: https://github.com/OWASP/API-Security/blob/master/2023/en/src/0xa8-lack-of-protection-from-automated-threats.md.

[45] OWSAP. *API9:2023 Improper Inventory Management*. Accessed: 2023-03-24. URL: https://github.com/OWASP/API-Security/blob/master/2023/en/src/0xa9-improper-assets-management.md.

[46] OWSAP. *Encode and Escape Data*. Accessed: 2023-05-10. URL: https://owasp.org/www-project-proactive-controls/v3/en/c4-encode-escape-data#:~:text=Encoding%20(commonly%20called%20%E2%80%9COutput%20Encoding,writing%20to%20an%20HTML%20page..

[47] OWSAP. *Input Validation Cheat Sheet*. Accessed: 2023-05-10. URL: https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html.

[48] OWSAP. *OWASP API Security Project*. Accessed: 2023-03-02. URL: https://owasp.org/www-project-api-security/#.

[49] Postman. *A guide to the different types of APIs*. Accessed: 2023-05-29. URL: https://blog.postman.com/different-types-of-apis/.

[50] Postman. *API 101: What Is a SOAP API?* Accessed: 2023-05-29. URL: https://blog.postman.com/soap-api-definition/.

[51] Postman. *Scripting in Postman*. Accessed: 2023-05-15. URL: https://learning.postman.com/docs/writing-scripts/intro-to-scripts/.

[52] Rapid. *API Security - Broken Function Level Authorization Vulnerability*. Accessed: 2023-03-02. URL: https://rapidapi.com/guides/broken-function-level-authorization.

[53] Salah Sharieh and Alexander Ferworn. “Securing APIs and Chaos Engineering”. In: *2021 IEEE Conference on Communications and Network Security (CNS)*. 2021, pp. 290–294. DOI: 10.1109/CNS53000.2021.9705049.

[54] Ronghua Sun, Qianxun Wang, and Liang Guo. “Research Towards Key Issues of API Security”. eng. In: *Communications in Computer and Information Science*. Vol. 1506. 2022, pp. 179–192. ISBN: 9811692289.

[55] Twitter. *An incident impacting some accounts and private information on Twitter*. Accessed: 2023-03-11. URL: https://privacy.twitter.com/en/blog/2022/an-issue-affecting-some-anonymous-accounts.

[56] Kazi Wali Ullah, Abu Shohel Ahmed, and Jukka Ylitalo. “Towards Building an Automated Security Compliance Tool for the Cloud”. In: *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*. 2013, pp. 1587–1593. DOI: 10.1109/TrustCom.2013.195.

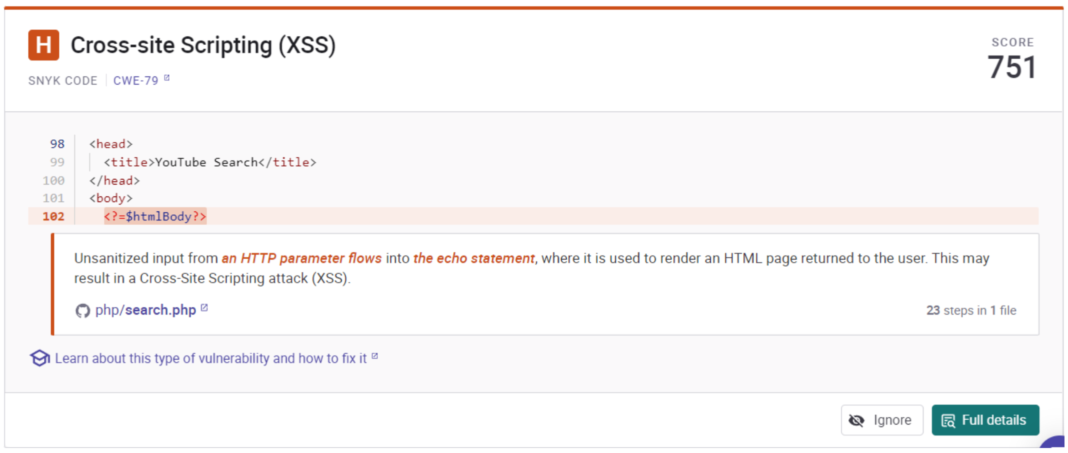
[57] WhiteBlueOcean. *Retroactive Auditing*. Accessed: 2023-03-24. URL: https://www.whiteblueocean.com/newsroom/5-key-data-breaches-in-2022/.

[58] M. Frans Kaashoek Xi Wang Nickolai Zeldovic. *Retroactive Auditing*. Accessed: 2023-03-23. URL: https://people.csail.mit.edu/nickolai/papers/wang-rad.pdf.

Appendix A

Appendix

Below are Screenshots of vulnerability scanning of the YouTube Data API using Snyk.


H
Cross-site Scripting (XSS)
SNYK CODE | CWE-79
SCORE
751

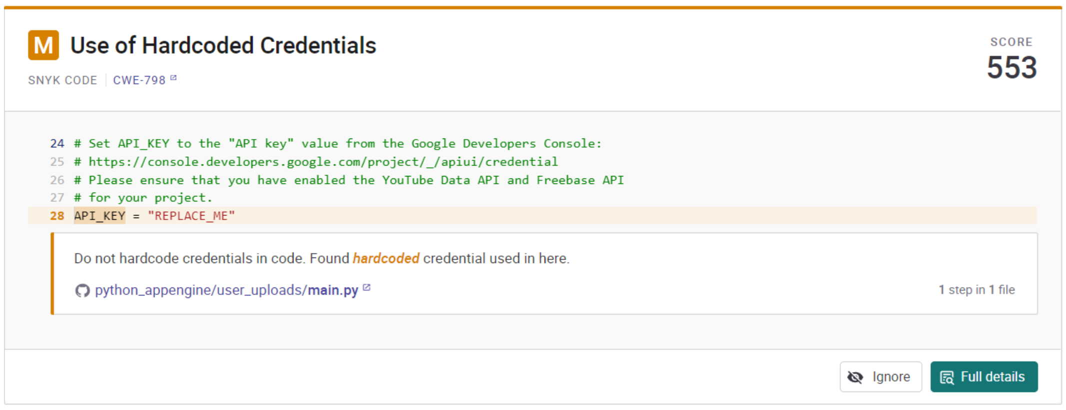
```
<head>
  <title>YouTube Search</title>
</head>
<body>
  <?=$htmlBody?>
```

Unsanitized input from *an HTTP parameter flows* into *the echo statement*, where it is used to render an HTML page returned to the user. This may result in a Cross-Site Scripting attack (XSS).

php/search.php — 23 steps in 1 file

Learn about this type of vulnerability and how to fix it
Ignore
Full details

Figure A.1: Youtube Data API Vulnerability Scan Results - 1a

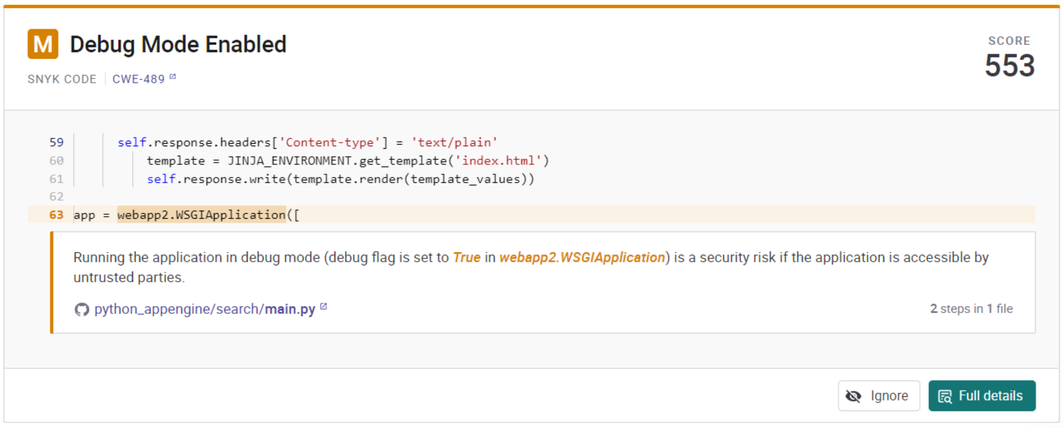


```
# Set API_KEY to the "API key" value from the Google Developers Console:
# https://console.developers.google.com/project/_/apiui/credential
# Please ensure that you have enabled the YouTube Data API and Freebase API
# for your project.
API_KEY = "REPLACE_ME"
```

Do not hardcode credentials in code. Found *hardcoded* credential used in here.

python_appengine/user_uploads/main.py — 1 step in 1 file

Figure A.2: Youtube Data API Vulnerability Scan Results - 1b



M Debug Mode Enabled

SNYK CODE | CWE-489

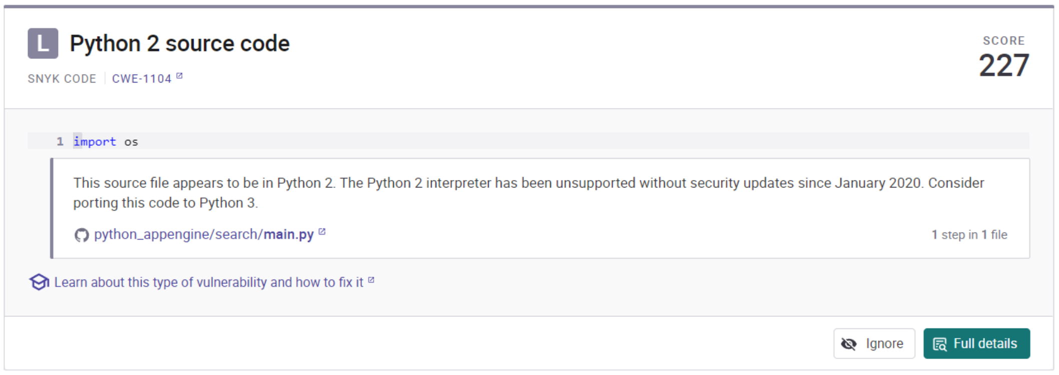
SCORE 553

```
        self.response.headers['Content-type'] = 'text/plain'
            template = JINJA_ENVIRONMENT.get_template('index.html')
            self.response.write(template.render(template_values))

app = webapp2.WSGIApplication([
```

Running the application in debug mode (debug flag is set to *True* in *webapp2.WSGIApplication*) is a security risk if the application is accessible by untrusted parties.

python_appengine/search/**main.py** — **2** steps in **1** file

Figure A.3: Youtube Data API Vulnerability Scan Results - 1c



L Python 2 source code

SNYK CODE | CWE-1104

SCORE 227

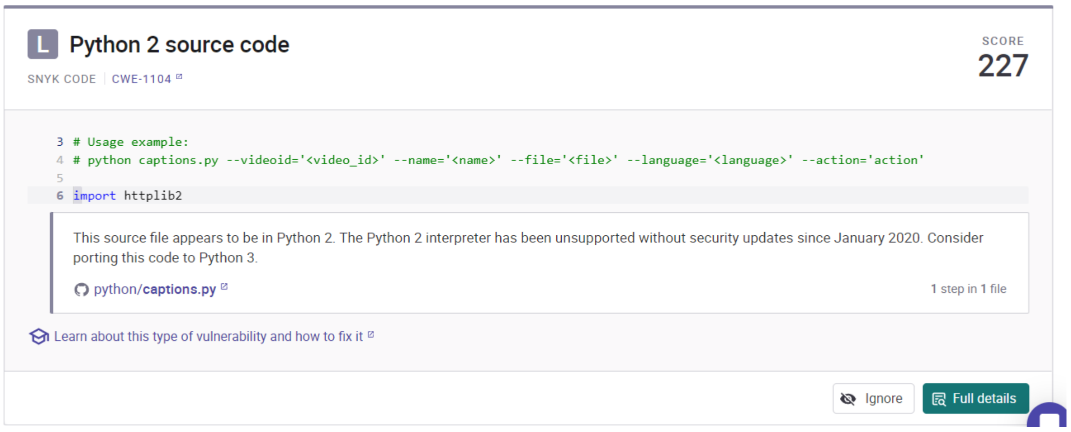
```
import os
```

This source file appears to be in Python 2. The Python 2 interpreter has been unsupported without security updates since January 2020. Consider porting this code to Python 3.

python_appengine/search/**main.py** — **1** step in **1** file

Learn about this type of vulnerability and how to fix it

Figure A.4: Youtube Data API Vulnerability Scan Results - 1d



L Python 2 source code

SNYK CODE | CWE-1104

SCORE 227

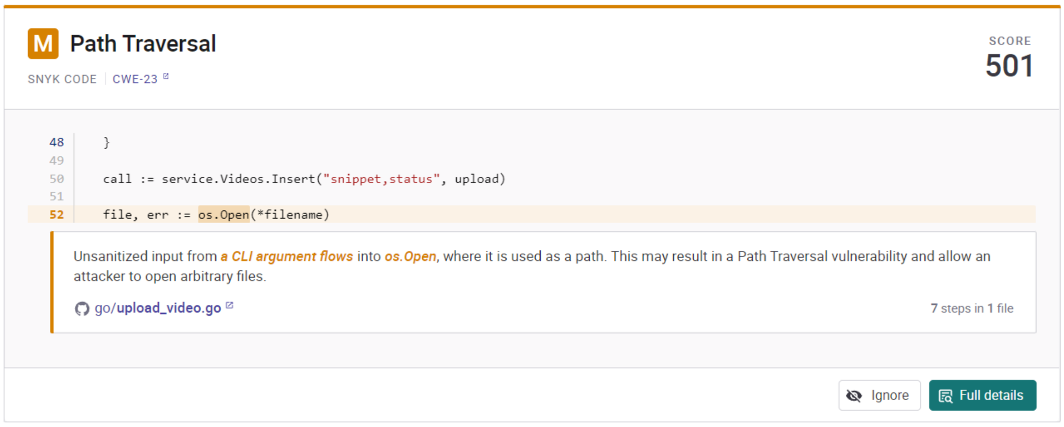
```
# Usage example:
# python captions.py --videoid='<video_id>' --name='<name>' --file='<file>' --language='<language>' --action='action'

import httplib2
```

This source file appears to be in Python 2. The Python 2 interpreter has been unsupported without security updates since January 2020. Consider porting this code to Python 3.

python/**captions.py** — **1** step in **1** file

Learn about this type of vulnerability and how to fix it

Figure A.5: Youtube Data API Vulnerability Scan Results - 1e


M Path Traversal

SNYK CODE | CWE-23

SCORE
501

```
    }

    call := service.Videos.Insert("snippet,status", upload)

    file, err := os.Open(*filename)
```

Unsanitized input from *a CLI argument flows* into *os.Open*, where it is used as a path. This may result in a Path Traversal vulnerability and allow an attacker to open arbitrary files.

go/upload_video.go

7 steps in 1 file

Figure A.6: Youtube Data API Vulnerability Scan Results - 1f