International Journal of Computer Trends and Technology
ISSN: 2231–2803 / https://doi.org/10.14445/22312803/IJCTT-V72I9P125

Volume 72 Issue 9, 157-164, September 2024
© 2024 Seventh Sense Research Group®

Original Article

How to Secure APIs to Defend Against Emerging Cyber Threats to Digital Web Assets

Piyush Dixit

Director - Integrations & API Software Engineering, Cummins Inc., Indiana, USA.

Corresponding Author : piyushdixitwork@gmail.com

Received: 08 August 2024 Revised: 06 September 2024 Accepted: 26 September 2024 Published: 30 September 2024

Abstract *- Application Programming Interfaces, or APIs, enable modern businesses to exchange data and establish connectivity between digital systems. However, increased connectivity needs have spawned more novel security risks as more and more digital assets get deployed on the web. This is why API security has become paramount for any successful business organization, that is exposing its web digital assets on the internet. Common API security risks include things like vulnerability exploitation, where attackers exploit flaws in an API's construction, or zero-day exploits that exist on the infrastructure where APIs are hosted or most famous DDoS attacks, where APIs are made unavailable, all these risks lead to cyber issues like unintended access, data breaches or data unavailability. There are several best practices to ensure API security, like API OWASP (2023 list) top 10, API rate limiting, DDoS mitigation, payload validation, authentication mechanisms like OAuth, logging and monitoring to ensure accountability and traceability. The purpose of this study is to present an accomplished stepwise solution in the form of an API security framework for organizations that are serious about API security and want to know how and where they can start their journey to secure their digital assets exposed over the web, while still benefiting from them as intended originally.*

Keywords - *API Security, OWASP top 10, OAuth, API rate limiting, DDoS, HTTPS.*

1. Introduction

There is a considerable gap in current research about API security. The gap is multi-faceted, starting from the fact that there is no basic defined structure of what entails within the realm of API security to all the way up to what all the threats to API security that are relevant to separate the chaff from the wheat and most importantly how to address those threats. There are a multitude of definitions, practices and processes that are prevalent in the API security space. API security refers to the collection of methods, processes and software that are used to protect Application Programming Interfaces (APIs) from cyber-attacks. APIs enable communication between different software and deal with transferring business data between digital applications, which is why it is important that every organization building or using APIs pays special attention to securing APIs.

What further adds to the gap in the current research is the fact that over the years organizations have become better and more experienced in securing their websites that are exposed over the internet. However, the same cannot be said when it comes to securing their APIs. When it comes to protecting APIs, there is still a long way to go for even some big organizations. Although websites and APIs both operate over OSI layer 7 using https protocol over TCP/IP, with both running on the same client-server architecture at their core, the format of data that gets exchanged between a website and a browser as compared to the data between an API and its consumer application is structurally very different. This is why all security measures applied to protect a website mostly do not fully protect APIs as well, partially may be up to an extent. Hence, API security warrants special attention and measures applied.

It is understandable to get intimidated by API security requirements given their different structure as compared to websites. However, by any means it is not any more complex, though different for sure. For organizations not under targeted attack by a competitor or a state actor from foreign countries, there can be a systematic method that can be applied to ensure API security and protection of digital assets. The unique purpose of the research presented here is to present standard and simple-to-follow guidelines for organizations to protect APIs effectively and only focus on the most relevant measures based on the most prevalent threats.

However, before applying a solution, it is necessary to understand better the problem first, which in the case of API security will mean understanding the nature of threats that exist against the APIs, posing risks to the business data they

This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/)

transport. Moreover, it is only after understanding those threats that it will make more sense to work via a stepwise approach to secure APIs and protect the data they transport. The stepwise approach will include the application of multiple processes, methods and software tools in unison to protect APIs.

This means it requires intention and leadership support within the organization to support the vision and budgetary needs in accordance with the company's risk exposure and spending capabilities.

2. Overview of most Common API Threats

2.1. Misuse of Broken Object-Level Authorization

Typically, an API offers accessibility to different data objects either via the same endpoint using different identifiers or via different endpoints clubbed within the scope of the same API as a common implementation.

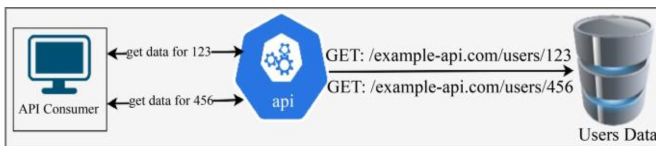

Fig. 1 Different objects, different identifiers, same endpoint

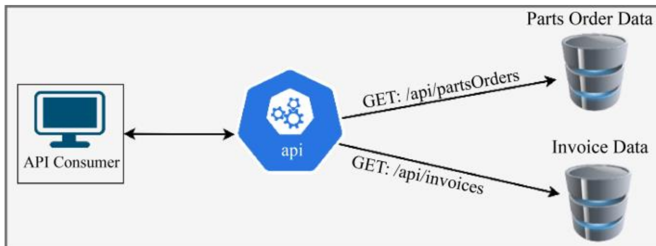

Fig. 2 API delivering different objects via different endpoints

Figures 1 and 2 above explain both scenarios when an API delivers different data objects using the same or different endpoints. In either case, a bad actor can misuse a broken object-level authorization to access the data they are not authorized to access.

For the case shown in Figure 1, a bad actor would be allowed to see only data for user 123. However, since object-level authorization is missing, they can pass identifier 456 and access data related to user 456 as well, using the same access.

For the case shown in Figure 2, a bad actor will get authenticated for accessing order data using the first endpoint but will change the endpoint within the same API to invoices and shall be able to access the invoice data as well, which they may not be authorized to access.

2.1.1. Real-World Example of Breach Misusing BOLA

A real-world incident exploiting Broken Object Level Authorization (BOLA) occurred in 2019 with Uber. As reported in multiple news articles, there was a vulnerability in one of Uber's APIs that allowed attackers to access any of the user's data from the master database because the API implementation did not validate that the client sending the request actually had access to the user information being queried using the user ID parameter, the calling client could simply access data of any other users by simply changing the user ID.

Another real-world case of misuse of BOLA broken object-level authorization happened with the Fleed dating app in 2024. BOLA vulnerabilities in Feeld's API allowed attackers to access private photos and chats of other users by sending different IDs in the URL for other users.

2.2. Misuse of Broken Authentication of an API

It will not be an overstatement that misuse of broken or week authentication is the most common threat that exists against the majority of the APIs due to poor authentication practices implemented in the construction or hosting of an API. What is surprising is that it is the most basic and possibly the easiest to mitigate risk out of all other threats, yet it remains the most exploited one. Broken authentication occurs when the authentication mechanism applied to secure an API either does not exist or is flawed or is week, resulting in unauthorized access, where attackers can assume the identities of legitimate users.

The most common way broken authentication is misused is for the APIs that use static API keys to authenticate the client or consumer of the API. These static keys are often expected to be appended in the API endpoint, which often gets passed in clear text as part of the http URL. It is very easy for a man-in-the-middle type set-up to capture these keys and then misuse them later for accessing data via the API. The challenge with these keys is the fact that there is no other way to differentiate the calling party from a malicious actor. If the key that is being passed by both is the same, the API will authenticate the calling consumer, as shown below in Figure 3.

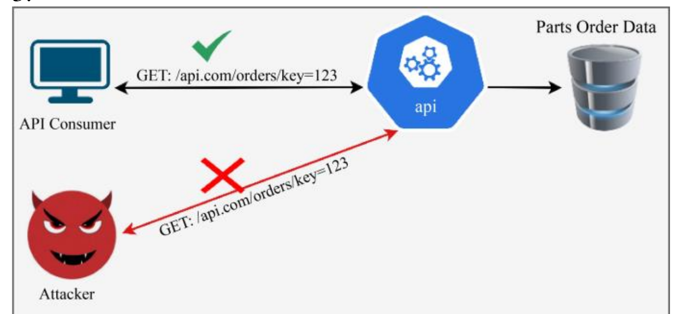

Fig. 3 Attacker using same API key as genuine consumer

These API keys are retrieved by hackers, typically using man-in-the-middle attacks, sometimes by accessing breached data sold over the dark net. Another way a broken authentication attack is manifested is by using brute force

password guessing algorithms for APIs that use simple user ID and password-based authentication, which is different from static key-based authentication. Sometimes, it is an even worse type of API authentication where the user ID and password are passed in http headers as clear text, making it further easier to sniff and misuse to gain access to API data.

Another very common API authentication type is where some kind of client ID and secret are used to issue an access token, and going forward, that access token is used repeatedly for some finite or sometimes infinite amount of time to continue authenticating the consumer to continue getting access to API data. Now, this one is certainly better than the other two discussed so far, as it is not based on any static key or clear text user ID or password. However, it comes with its own set of weaknesses that can be misused to gain unauthorized access, compromising the API security. The main issue with token-based authentication is the token itself, more precisely, how the token lifespan is managed. Many times, tokens do not have a definite amount of time since the generation after which they expire, which means a token of that kind is similar to a static key and can continue to be misused. Another problem with tokens that can lead to misuse is improper revocation of tokens, which leads to tokens staying valid even though the expectation is for them to expire. All these in the hands of trained attackers are means to a misuse of an API.

2.3. Misuse of Unrestricted API Resource Consumption

Unrestricted resource consumption misuse happens when an API does not limit the use of its underlying resources like throughput, payload size, network bandwidth consumed, CPU, memory, and storage without proper controls. Attackers exploit this vulnerability to execute Denial of Service (DoS) attacks by overwhelming the API with a high volume of requests, ultimately leading to resource exhaustion, making the API partially or fully unavailable to legitimate users, resulting in business loss. This requires high operation effort and hence cost to manage to scale accordingly and results in high billing of infrastructure.

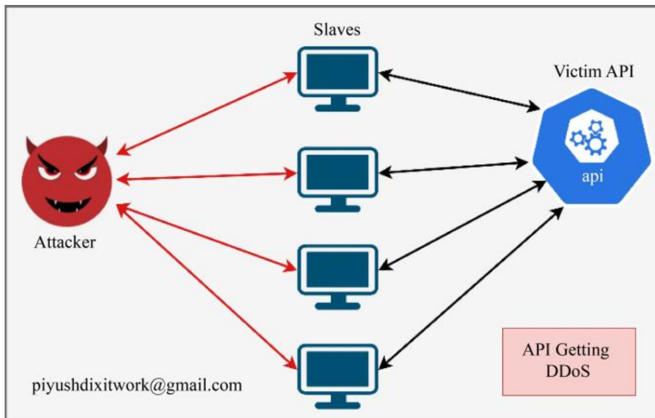

Fig. 4 DDoS due to unrestricted resource consumption

Figure 4 shows how an attacker uses unrestricted resource consumption to launch a DDoS attack on an API that has not implemented checks to avoid and restrict the usage of its resources. As seen either with different slave hosts or sometimes even with the same host, an attacker can launch a bogus flurry of multiple http requests to flood the API server. Since the API server does not have any restrictions to protect against this tsunami of requests, it ends up denying genuine requests sent by legitimate users to do business using the API, which is its original objective. The impact of this is not only restricted to not being able to serve genuine requests, but it impacts complete backend infrastructure where API is hosted or deployed, and that infrastructure offers various resources that are consumed by the API to run the business logic implemented and fetch and deliver the data.

All this backend underlying infrastructure that involves things like CPU, memory, network bandwidth, etc., gets overused. To address it, constant intervention is needed from operations personnel managing the API hosting to beef up the infrastructure to meet the demand. This not only involves the cost to pay for the effort of operations but also requires incurring additional infrastructure cost that comes with beefing it up.

2.4. Misuse of Insufficient Logging and Monitoring

Misuse of insufficient logging and monitoring happens when an API implementation fails to deploy enough logging and monitoring of API activities, in turn making it difficult to detect threats and respond to security incidents like overuse of resources or, broken authentication or unauthorized access, to name a few. Lack of proper mechanisms to record and review activities within an API typically includes issues like missing logs, inadequate log details, or unavailability of continuous monitoring. The impact manifests twofold, where the inner workings of API, like high infrastructure resource consumption, are not highlighted in time, leading to a big failure and two, when a bad actor tries to exploit the API and the data it serves, there are not enough means to detect that.

Sometimes logging is there, but it is not adequate in terms of capturing key details like user IDs or IP addresses or, payload structures or timestamps. Logging may capture error messages as a bare minimum most of the time, but missing these key pieces of information does not leave many options to address the threat in the long term. There is no traceability to establish any sort of accountability for appropriate lawful interventions.

Capturing unique attributes like IP addresses or user IDs or the http headers and payload structures adds more character to the incident. It helps boil down to a more pinpointed root cause, which enables quick and long-term solutions as opposed to hit and trial that are both short-term and sometimes totally ineffective in nature to address the actual root cause of the incident.

Sometimes integrity of logs is questionable because the logs can be easily tampered with or can be inserted with false information, compromising their reliability. This plainly happens due to blatant disregard for safeguarding the logs by ensuring limited and restricted access to them and restricting any updates after first writing.

It is noteworthy that this restriction is needed not only for unauthorized access from external bad actors but is equally necessary to safeguard rogue actors from within the company who may be operating under the feeling of personal vendetta against the company.

Last but certainly not least, by any means, is the lack of monitoring that results in the exploitation of APIs going unchecked and unaddressed. API implementations could have state-of-the-art logging, as discussed above.

However, if there is no monitoring and alerting mechanism, then there is no way to act immediately on an attack and either fully stop it or at least prevent it from happening to mitigate the damage. Monitoring is key to acting in time and avoiding impact due to an attack.

3. The Solution to Common API Threats

3.1. Address Broken Object-Level Authorization

Step 1 is, at the time of design of an API or at least during implementation, thoroughly identify all the different objects that are served via the API. This could be in different forms, like there could be different data objects that an API is delivering utilizing multiple endpoints that are clubbed within the same API implementation. Another variation could be that the API is serving different data records, that are identified by unique and distinct identifiers. The third variation could be multiple http methods like GET, POST, PUT and DELETE that are supported by some API to enable the execution of different CRUD operations.

Step 2 is to first, on paper, create an access control matrix that highlights different variations of different objects that exist within an API implementation and against each of those object variations, it also highlights respective user groups or user roles or, if applicable, individual users and associated respective IDs that are allowed to access the data within corresponding objects.

Step 3 is to utilize an authentication and authorization solution that allows the implementation of this object roll-based access control matrix. A tool that offers implementing authorization capabilities will ensure that only authorized user roles for a particular object within any API are allowed to access the data underneath. Authorization is something that happens after the authentication. If authentication identifies the validity of who the calling party is, then authorization ensures that they can access only the information they are entitled to.

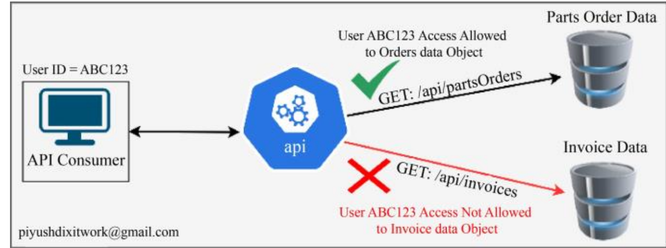

Fig. 5 Object level restricted access control in action

Figure 5 shows how properly implemented object-level access control shall work in action. As depicted in the diagram, when user "ABC123" sends a request to access Orders data, the access is granted, given that this particular user ID is authorized to access Orders data per the access control matrix created in step 1. But the same does not happen when the same user, "ABC123", sends a request to access Invoice data; in that case, the access is not granted, and the request is rejected because, per the matrix, this user ID is not authorized to access Invoice data, and API will not deliver it. The way to implement authorization is to have a tool typically referred to as authorization server software that implements and manages the matrix of access control. When real-time live calls are made to an API, it is the authorization server that does the job of ensuring that access to the data is only granted per the authorization rules established in the matrix. If there are any anomalies, it denies the access request for that particular object.

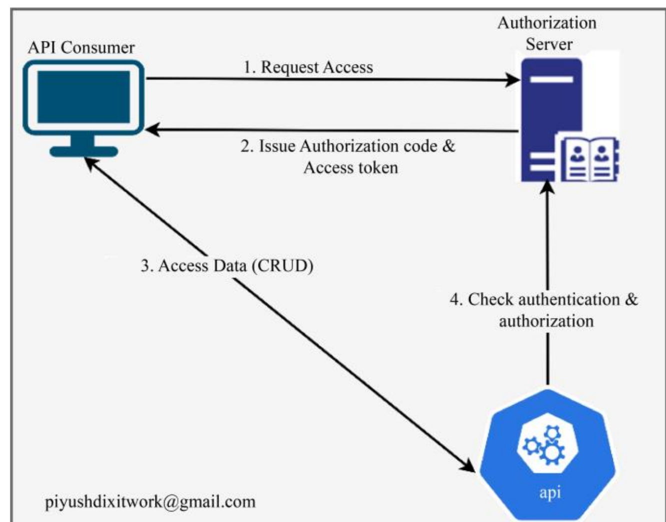

Fig. 6 Workflow showing access control using an authorization server

Figure 6 shows the flow that takes place for a typical implementation of role-based object-level access control while using authorization server software. As seen in the diagram, the API-consuming client makes the first request, and it sends that first request to the authorization server, which first authenticates the caller. Once the caller is identified, the authorization server will look up the authorized accesses that are permitted for this identified user, and it will issue an

authentication token and an authorization ID, which, going forward, will dictate what level of access and what type of objects this particular user can access via the API.

The calling API consumer, in the third step, makes a call to the API using the authentication token and authorization code, which are both embedded into the request that is being sent from the consumer to the API. The API extracts the authentication token and authorization code and, in Step 4 makes a call to the authorization server to validate both. It is only after the API receives the confirmation response from the authorization server, that access is being granted to the consumer and API controls, and the level of access is strictly according to the authorization code. This is how companies can safeguard against the misuse of broken object-level authorization.

3.2. Address Broken Authentication of an API

The first and foremost thing that needs to be done to protect against broken authentication is to eliminate the utilization of static key-based or user ID and password-based authentication mechanisms for APIs. In today's day and age, OAuth 2.0 has emerged as an effective and trustworthy authentication solution for APIs. It offers more dynamic and granular control over managing access to API resources.

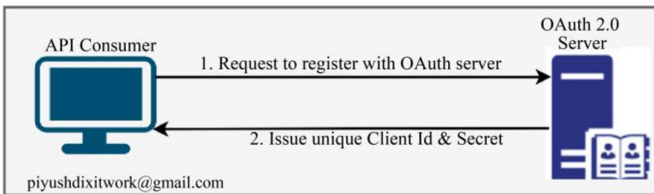

Fig. 7 Client registration with OAuth server

As seen in Figure 7, the very first step that an API calling client or consumer needs to do in a typical OAuth authentication-based API implementation is to register itself with the OAuth server. This registration process creates a unique special entry for that client or consumer in the OAuth database. This registration entry is typically centrally managed within the OAuth server by the administrators of the API. In response to this registration request by the client, the OAuth server responds back with a uniquely generated client ID and client Secret, that identifies this API consumer or client. Since this registration is maintained centrally through the OAuth server, it becomes very easy to revoke the granted access for the clients in a matter of a minute in the event of a breach or misuse.

The expectation for the client or the API consumer is to hold on to this newly issued client ID and client secret and store it somewhere on its side. Every time when the client makes a call in future, post registration with the OAuth server, it will be required to send these client IDs and secret pair along with its API request to authenticate itself and get authorized to access the API. Another important step here is to implement the functionality on the API side of the codebase that enables the API to check with the OAuth server every time when a request comes from a client.

It is important to focus on building processes within the company to control and manage the access within the OAuth tool. A centralized admin operations team typically does this within a company that is always ready to act on a moment's notice if some accesses need to be removed from the OAuth server for a calling consumer or client of an API in case of misuse. What makes this team and their control more effective is the existence of a robust monitoring and alerting system that tips them off about unauthorized access as and when it happens on a live API.

Once the API client is being registered in the OAuth server, after, there is a complete flow of actions that takes place when the client tries to access an API with OAuth being in the middle of the solution. It could be explained by a simple five steps flow that happens between three parties involved, first being the client or the consumer of the API requesting the access, second being the authentication and authorization server and third being the API itself that receives the request from the client and renders the requested data or allows the client to perform requested CRUD operation on the expected data object.

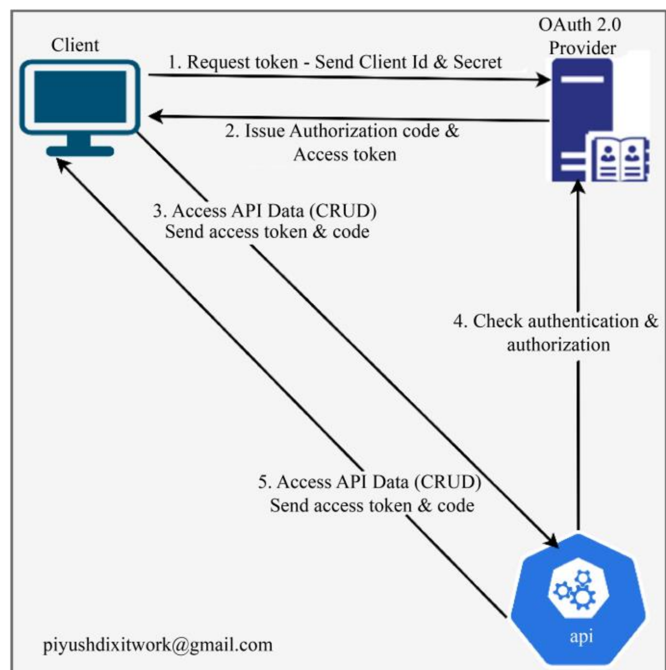

Fig. 8 Client validation done by the API using the OAuth server

As shown in Figure 8 an API Consumer or client makes the very first call to the OAuth server and sends its client ID and secret embedded within that call to identify itself to the OAuth server. The OAuth server uses those credentials to locate the client registration entry in its database. It looks up

specific accesses that the client is authorized to have on resources and objects exposed by the API delivering the data. Once the OAuth server validates the client, then it responds back to the client with an access token an authorization ID. The client extracts them both from the response and appends those in the request it sends to the API to perform CRUD operations. API, while keeping this request on hold, initiates a request to the OAuth server in parallel to validate the token and ID sent by the client and on successful approval from the OAuth server, API allows to perform the requested operation by the client.

One most important steps that need to be done in the implementation of the OAuth to protect APIs is to ensure there is a very mature token management setup both systemically and process-wise. From a systems perspective, all the tokens shall have a limited lifetime from a few minutes to a few days at max and shall expire after that, requiring a new token. From a process perspective, there shall be periodic audits within the company to review accesses that are allowed for different clients per the registration in the OAuth database.

3.3. Address Unrestricted API Resource Consumption

Implementation of rate limiting of different kinds depending upon the implementation of the API is the foremost solution for addressing this threat on APIs. Rate limiting is putting policies on top of the API implementation that monitor for usage statistics of an API. When it goes above a certain limit, as defined in the rate-limiting policies, then the connection is terminated to free up the API resources.

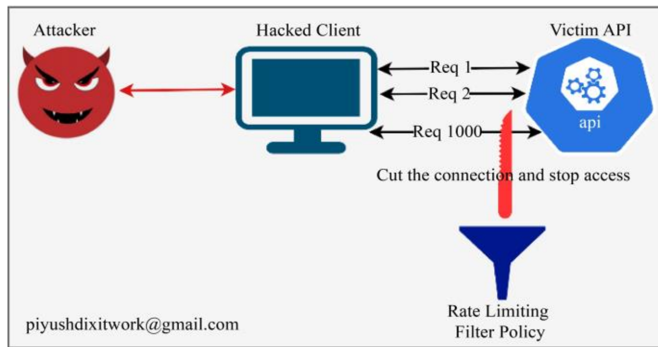

Fig. 9 Rate limiting in action to cut connection when violated

As shown in Figure 9, a very simple rate-limiting policy that checks for the same client sending more than 999 requests in an hour is effective in cutting the connection for that client without impacting any other connection. In the case shown in this figure, a bad actor has gained access to an authorized legitimate client and has modified the client to blast API implementation with more than 999 requests within an hour, which goes above the set rate limiting policy on the API, which ensures that it stays within the limit.

Another common scenario that benefits from rate limiting is the size of the payload request being sent by the client to the API server. This shall be within certain maximum size limit boundaries to keep the I/O utilization by the API within the boundaries of the set infrastructure capacity.

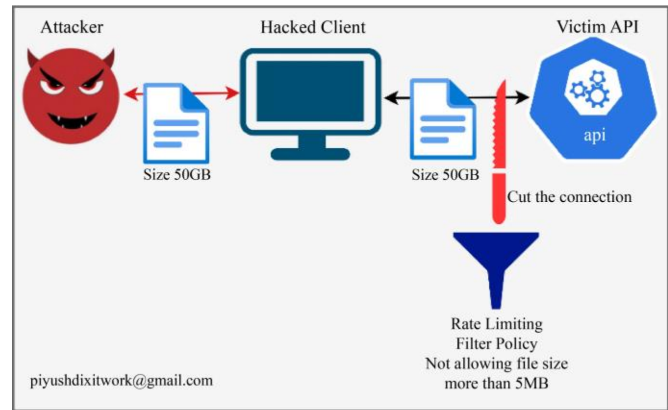

Fig. 10 Rate limiting cutting the connection with large file size

As seen in Figure 10, a hacked client tries to overwhelm the API server by sending a very large sized file, which will require heavy I/O and memory utilization on the API server, resulting in other genuine requests facing a scarcity of server resources. The simple rate limiting policy that watches out for any file being sent in the request that sizes over 5MB will immediately disconnect the thread and will not let the file land on the API server and not let it process this heavy file.

Another key variation of rate limiting policy is implementing a time-out setting on an API. This setting watches out for processing time for a single request or a process thread within the API. If that exceeds the set threshold processing time limit, then the policy kills the thread and stops any further execution for that request to ensure there is no resource exhaustion on the API server. Long-running requests that may require high processing due to a wrong input or stuck thread can consume significant CPU, memory, and other infrastructure resources, potentially affecting the performance of other requests getting served by the same API.

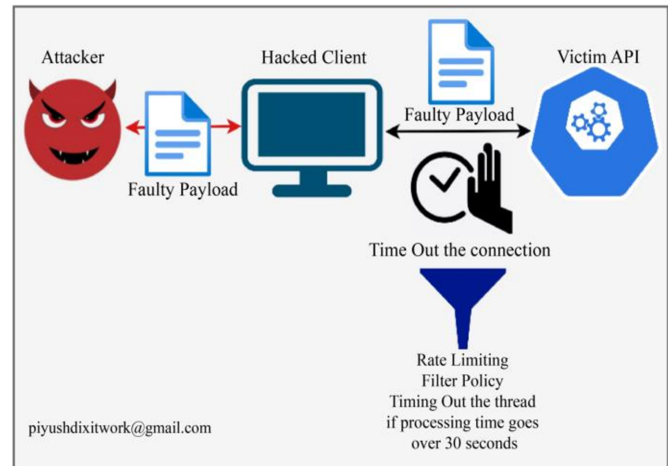

Fig. 11 Time out setting killing the thread above 30 seconds

As seen in Figure 11, it is clear how the time-out policy works to safeguard the burnout of server resources. A bad actor can hack a genuine client and force it to send malicious payload or faulty characters that may result in the API processing times increasing significantly or, even worst, a stuck thread situation on the API server, which mostly does not get resolved on its own unless there is some manual intervention to kill the thread to free up the resources. Setting up time-out limits offers an automated mechanism to detect such long-running requests or processes. It allows the policy to terminate such sessions and free up resources to be accessed by other genuine requests, protecting API from going into self-denial of service-type situations affecting business usage and sometimes company revenue.

3.4. Address the Issue of Insufficient Logging and Monitoring

The first step to do this is to have a centralized logging and monitoring tool implemented in the company. The second step is to make this tool accessible and connected with API implementations across the company. The API shall have the logic to post the logs with appropriate details at periodic checkpoints within the process flow of the API and not only for error logging purposes but also for logging other relevant information like IP, timestamps, payload, etc.

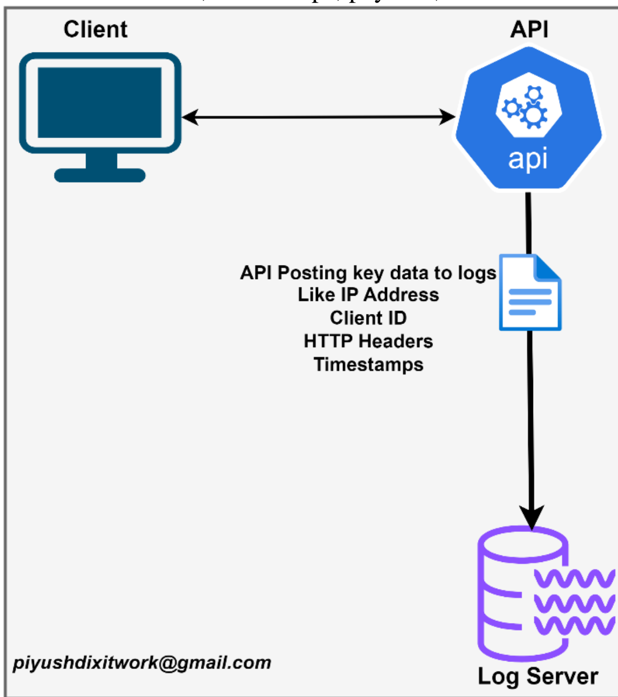

Fig. 12 API logging data into centralized log server

As seen in Figure 12, an API with clearly identified checkpoints within its flow posts key details like IP addresses, Client ID, timestamps, and HTTP headers to the central logging server, which later offers this data for all kinds of real-time mitigation actions that can be taken by a team or software monitoring the logs or it can be used for doing a retrospective analysis in future to make informed decision on infrastructure sizing or other interventions needed to improve the performance and reliability of the API.

No logging can be complete without appropriate monitoring and alerting mechanisms built on top of it. Hence, it is of equal importance to ensure there are alerts implemented and people or automation software are acting on top of those alerts when one is triggered. Ensure that the logs are fully secure and that only authorized personnel can access the logs. For monitoring as well restrict the edit access to limited personnel.

4. Stepwise Approach to Address Common API Threats

Table 1 presents a stepwise approach to plan and execute the strategy for safeguarding APIs against main threats that compromise the integrity and functioning of APIs. Using this approach will ensure that the APIs are protected against the most well-known threats that often are overlooked, resulting in massive data and business loss.

Table 1. Stepwise approach

Step	Description
1	Implement OAuth 2.0
2	Build object role-based access control matrix.
3	Use an authorization server to manage object-level access.
4	Periodically audit access and authorization data.
5	Implement various rate-limiting policies.
6	Implement adequate logging and monitoring.

5. Conclusion

These basic safeguards that are discussed so far are surprisingly very effective in protecting against threats, if not all then still most of the attacks and threats that exist against APIs. The majority of the breaches happen not due to very sophisticated attacks but due to not having even very basic safeguards in place. But API security is an ongoing, never-ending process. These safeguards work as a solid starting point. However, there is always a need for constant evaluation and, review and audit of the risk exposure of a company, and according to changing landscape, there is always a decent margin to implement modifications. Another good practice is to implement a robust and periodic testing mechanism to challenge the existing safeguards in place and determine new ones needed to bolster security further. Something as simple as open box testing with an internal team to something as sophisticated as complete black box testing done by professional penetration testing organizations are all good interventions. What is better for which organization solely depends on their risk exposure and existing threats to the company's existence, but a periodic evaluation of current security measures of APIs within a company is a mandatory step in ensuring API security.

References

[1] OWASP API Security Project, OWASP, 2023. [Online]. Available: https://owasp.org/www-project-api-security/

[2] OWASP Top 10 API Security Risks, OWASP, 2023. [Online]. Available: https://owasp.org/API-Security/editions/2023/en/0x11-t10/

[3] The Ten Most Critical API Security Risks, OWASP, pp. 1-31, 2019. [Online]. Available: https://owasp.org/API-Security/editions/2019/en/dist/owasp-api-security-top-10.pdf

[4] API1:2023 Broken Object Level Authorization, OWASP API Security Top 10, OWASP, 2023. [Online]. Available: https://owasp.org/API-Security/editions/2023/en/0xa1-broken-object-level-authorization

[5] Inon Shkedy, The Uber API Authorization Vulnerability, Traceable, 2021. [Online]. Available: https://www.traceable.ai/blog-post/the-uber-api-authorization-vulnerability

[6] Mark Dolan, Issue 255: Versa Director API Flaw, Feeld BOLA Vulnerabilities, Logic Flaw Risks Aircraft Disaster, API Security News, 2024. [Online]. Available: https://apisecurity.io/issue-255-versa-director-api-flaw-feeld-bola-vulnerabilities-logic-flaw-risks-aircraft-disaster/

[7] API2:2023 Broken Authentication, OWASP, 2023. [Online]. Available: https://owasp.org/API-Security/editions/2023/en/0xa2-broken-authentication

[8] API3:2023 Broken Object Property Level Authorization, OWASP, 2023. [Online]. Available: https://owasp.org/API-Security/editions/2023/en/0xa3-broken-object-property-level-authorization

[9] API4:2023 Unrestricted Resource Consumption, OWASP, 2023. [Online]. Available: https://owasp.org/API-Security/editions/2023/en/0xa4-unrestricted-resource-consumption

[10] API5:2023 Broken Function Level Authorization, OWASP, 2023. [Online]. Available: https://owasp.org/API-Security/editions/2023/en/0xa5-broken-function-level-authorization

[11] API6:2023 Unrestricted Access to Sensitive Business Flows, OWASP, 2023. [Online]. Available: https://owasp.org/API-Security/editions/2023/en/0xa6-unrestricted-access-to-sensitive-business-flows

[12] API7:2023 Server Side Request Forgery, OWASP, 2023. [Online]. Available: https://owasp.org/API-Security/editions/2023/en/0xa7-server-side-request-forgery

[13] API8:2023 Security Misconfiguration, OWASP, 2023. [Online]. Available: https://owasp.org/API-Security/editions/2023/en/0xa8-security-misconfiguration

[14] API9:2023 Improper Inventory Management, OWASP, 2023. [Online]. Available: https://owasp.org/API-Security/editions/2023/en/0xa9-improper-inventory-management

[15] API10:2023 Unsafe Consumption of APIs, OWASP, 2023. [Online]. Available: https://owasp.org/API-Security/editions/2023/en/0xaa-unsafe-consumption-of-apis

[16] Authorization Servers, Okta Developer, 2024. [Online]. Available: https://developer.okta.com/docs/concepts/auth-servers/

[17] OAuth 2.0 and OpenID Connect Overview, Okta Developer, 2024. [Online]. Available: https://developer.okta.com/docs/concepts/oauth-openid/

[18] David Neal, An Illustrated Guide to OAuth and OpenID Connect, Okta Developer, 2019. [Online]. Available: https://developer.okta.com/blog/2019/10/21/illustrated-guide-to-oauth-and-oidc

[19] API2:2023 Broken Authentication, OWASP, 2023. [Online]. Available: https://owasp.org/API-Security/editions/2023/en/0xa2-broken-authentication/

[20] MyF5, K000135849: Unrestricted Resource Consumption | APIs and the OWASP Top 10 guide (2023), My.F5, 2023. [Online]. Available: https://my.f5.com/manage/s/article/K000135849