Security of Web APIs: Strategies, Challenges, and Best Practices

1st Divesh Gadhvi
Rutgers Business School
New Brunswick, USA
divesh.gadhvi@rutgers.edu

2nd Jatin Thakkar
Rutgers Business School
New Brunswick, USA
jatin.thakkar@rutgers.edu

3rd Ashish Pardeshi
Rutgers Business School
New Brunswick, USA
ashish.pardeshi@rutgers.edu

***Abstract*—The rise of internet in the past two decades has given birth to a new digital age. An increasing global market for new software solutions has created a demand for robust software entities. The interconnected nature of digital systems, which include banking, medical and political sector has fueled the need for efficient software integration. Web APIs are one of the technologies that have gained importance in this new age.**

However, the rapid popularity of web APIs has presented a brand new, complicated set of problems that necessitate an instant, lasting answer. The necessity of safeguarding APIs from possible attacks initiated by malicious actors lies at the heart of these difficulties, thereby demanding security to be incorporated into the API architecture from the very beginning. An in-depth understanding of the many vulnerabilities that web APIs frequently encounter is needed as well as an all-encompassing approach incorporating several countermeasures to prevent and resist these threats.

There are two main goals for this research. We first aim to gather and thoroughly examine the corpus of existing data concerning the security risks that web APIs encounter. By doing this, we hope to produce a comprehensive solution to tackle risks related to using web APIs. Secondly, it aims to and explore methods and strategies at the design level for detecting, mitigating, responding to, and recovering from security threats focusing on web APIs. This study seeks to strengthen the security posture for internet APIs and, in turn, the larger software systems that are dependent on them by doing so.

***Index Terms*—APIs, safeguarding, security, vulnerabilities**

I. INTRODUCTION AND LITERATURE SURVEY

Web APIs (Application Programming Interface) have become increasingly significant over time. Web protocols like HTTP are used by many software applications to interact nowadays. These include anything from entertainment systems to financial information systems and systems for managing medical information. This is primarily an outcome of the advantages that web platforms offer to their users. As a result, web APIs are vital for the operation of many software systems. More than 83which marks a considerable increase in their use compared to 47of the Internet study from 2018. The software industry's tremendous increase in the use of web APIs likewise increases the significance of studying related to Web APIs, especially in the field of API security.

Web APIs have become increasingly significant over time. Web protocols like HTTPS are used by many software applications to interact nowadays. These include anything from entertainment systems to financial information systems and systems for managing medical information. This is primarily an outcome of the advantages that web platforms offer to their users. As a result, web APIs are vital for the operation of many software systems. More than 83internet is attributed to web APIs, which marks a considerable increase in their use compared to the 472014, according to Akamai's State of the Internet study from 2018. The software industry's tremendous increase in the use of web APIs likewise increases the significance of studying related to Web APIs, especially in the field of API security. When a web API is developed without taking security needs into account, vulnerabilities are introduced that might damage both the infrastructure of the company providing the service and the integrity of the service the API delivers.

Many API developers often prioritize API design and feature implementation before security design, which leaves them with a limited amount of time to consider what security measures are required in their APIs. A 2016 survey found that: 30APIs are specified without the IT security team's input; 27APIs move through the development stage without their input; and 21experts. Additionally, it has been discovered that almost 40the APIs included in the survey engaged security teams in either during testing (21makes sense that the majority of the survey participants (83were still concerned about API security. Not only due to a lack of awareness that security must be considered during every stage of the software development lifecycle (e.g., sensitive data, external APIs), but also because the system is complex and developers are not acquainted with the authentication flow. Additionally, the widely used API management platform could or might not have extensive security measures.

API security must be considered at the planning stage of development. Failing to do so can result in increased cost because of manual API testing. In some scenarios, it can also result on business loss for the company if the API is breached after production. It is very common for developers to often ignore the security aspects while designing an API. This is true for any software development; however, it is even more important for API development as APIs are generally given to external partners and other stakeholders for integration. Many APIs have been discovered having flaws in design security which include improper input validation, inadequate authentication or denial of service attacks, disclosure of API implementation

information, broken authentication or authorization, or in extreme cases, a complete lack of such mechanisms.

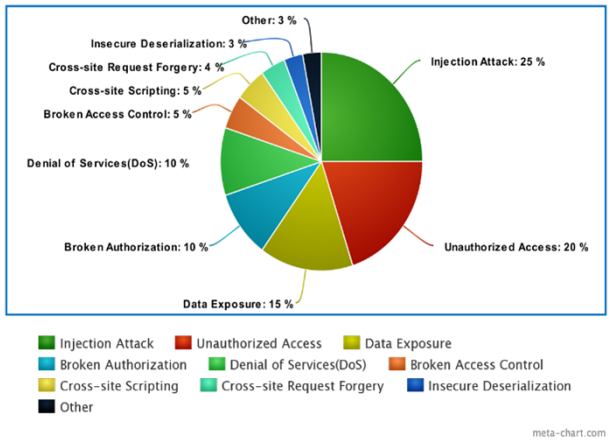

Fig. 1. Distribution of API Attacks

II. Problem Definition

Unauthorized Accesses: Attackers are always trying to target APIs that lack proper authentication, and authorization mechanisms which can lead to unauthorized entry to sensitive data.

The problem of unauthorized access to APIs is of great security concern, as attackers often try to exploit APIs with improper authentication/authorization mechanisms. This vulnerability is often caused due to inadequate authentication practices, such as easily guessable credentials, unencrypted storage of credentials, and techniques that do not properly restrict access due to poor implementation. Unauthorized access can endanger critical data and functionalities which can lead to data breaches, privacy invasion, and business disruption. To prevent this risk, it is very important to implement robust authentication and authorization mechanisms, rotate the credentials at regular intervals, routinely conduct security tests, and provide proper training to personnel involved with API security. Hence, it is of paramount importance to protect the APIs against unauthorized access to preserve system integrity.

In one such incident, MyfitnessPal, a fitness and nutrition tracking app experienced a security breach in February 2018 which allowed a malicious user unauthorized access to their data via public API. This incident was quiet notable because MyFitnessPal has an incredible Technology infrastructure.

What happened: Although the credentials were encrypted, the MyfitnessPal data leak served as a reminder that even the most advance systems can be penetrated if the API access is not properly implemented. Strong API security that includes data encryption, authorization, authentication, and proactive monitoring is critical in light of the incident that revealed users' private information, including email addresses and possibly stolen passwords. The need to use strong, one-of-a-kind passwords and turning on two-factor authentication wherever feasible was also underlined as a means of protecting user accounts.

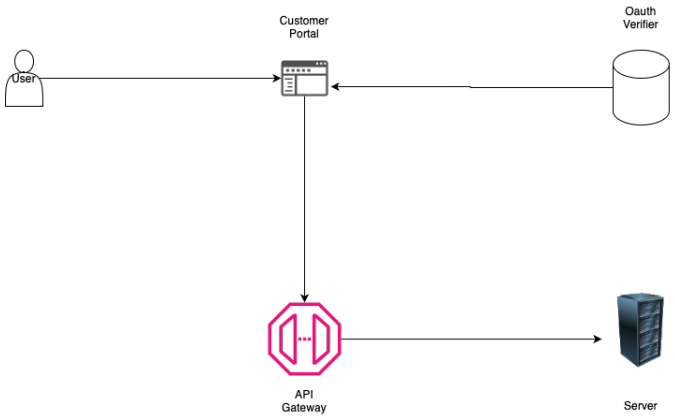

Fig. 2. Authentication Flow using Oauth

Denial Of Services (DoS) Attack: Attackers may try to flood the API with an extensive volume of requests, potentially leading to performance degradation, making it slow or even rendering it unavailable, thereby disrupting to legitimate users.

A Denial of Service (DoS) attack is a type of API attack in which attackers intentionally overload the API by creating a flood of fake requests. This kind of attack can have several negative implications, including slowing down the API or, in the worst-case scenario, making it inaccessible. This overwhelming number of requests maximize the system's capacity and disrupt the entire business flow. The API is completely occupied by malicious requests and the authentic users are unable to use it because the system is overloaded.The attackers' ultimate the objective is to overwhelm the API to the point that it can no longer process incoming requests successfully creating discomfort, service outages, and significant economic losses.

A denial-of-service (DoS) assault on Amazon Web Services (AWS) peaked in 2020 at 2.3 terabits per second (Tbps). One of the biggest DoS assaults ever seen was this one. Connectionless Lightweight Directory Access Protocol (CLDAP) reflection was the method used to initiate the attack. This method increases the volume of data transmitted to the victim's IP address by utilizing unreliable third-party CLDAP servers.

Numerous AWS clients experienced severe outages as a result of the assault, including well-known websites like GitHub, Disney+, and Spotify. Although the attack persisted for several hours, AWS was able to stop it and get its customers' services back up.

Injection Attack:SQL Injection and NoSQL injection attacks are types of attacks which manipulate the API requests, the requests are manipulated by adding malicious code in headers or request body of the API. This malicious code can

be used to access the data and potentially damage the internal servers if goes undetected.

The hackers use these attack techniques to manipulate the API requests in an attempt to undermine and weaken the API's security architecture. NoSQL injection attacks are generally targeted towards non-relational databases such as MongoDb and Cassandra DB, the attackers insert specially targeted query to modify database or interrupt regular operations of a database. The NoSQL injection attacks are generally used to get unauthorized access to data and these attacks are very dangerous which can result in data breach and data loss.

A SQL injection assault against the digital social networking site LinkedIn in 2013 resulted in the exposure of over 117 million members' email information. By taking advantage of a weakness in LinkedIn's login page, the attackers were able to introduce malicious SQL code into the database. With the usage of this code, the intruders were able to take over the database and get millions of user email addresses.

Over 57 million consumers' personal information was compromised by a SQL injection attack that Uber experienced in 2016. Through the use of a website vulnerability, the hackers were able to insert malicious SQL code into the database of Uber. With the usage of this code, the attackers were able to extract the information and take millions of user names, phone numbers, email addresses, and driver's licenses.

Data Exposure: Insufficient safeguarding of data can result in data exposure or leakage, which may involve the unauthorized disclosure of sensitive information to unauthenticated users.

The potential of data exposure is a genuine and all-to-common concern in today's digital environment. It comes forward when firms fail to set strong safeguards for their sensitive data. This vulnerability can appear in a variety of ways, but the bottom line is that it allows unauthorized persons or entities access to data that they should not have. The consequences of data exposure can be disastrous since it frequently entails the disclosure of secret, personal, or exclusive data. This is especially important in the domain of online APIs, where design flaws, poor encryption, or loose access restrictions within the API might allow malicious parties to get inappropriate access to data that is supposed to be limited.

T-Mobile had a data breach in 2021, exposing the personal information of around 76 million subscribers. The attackers acquired access to the company's network by using a web API provided by a third-party vendor. The attackers used this vulnerability to obtain consumers' names, addresses, email addresses, phone numbers, and Social Security numbers.

In 2022, hackers exploited a flaw in Equifax's online application to obtain the personal information of over 147 million individuals. Names, addresses, Social Security numbers, dates of birth, and credit card numbers were among the data.

III. Existing Solutions

In this part, will discuss about the existing solutions for the difficulties faced by the Web APIs . For each attack category, there are various solutions designed to protect the integrity of private information included in the APIs.

A. Solutions for Unauthorized Access

A. Solutions for Unauthorized Access Unauthorized access to an API leads to various problem for the company sensitive data can be breached and other serious consequences can occur. To countermeasure unauthorized access, the company can implement multiple security techniques and best practices.

1) **JSON Web Tokens (JWT):** JSON Web Tokens are a convenient, safe, and self-contained way for exchanging data between two parties, often as part of an authentication procedure. There are three elements to them: a header, a payload, and a signature.
    a) **Authorization:** A JWT's payload may contain information about the user's permissions. This ensures that the user only utilizes resources that they are authorized to access.
    b) **Statelessness:** JWTs do not need a server to maintain session information because they are stateless. This scales effectively and is especially helpful in a microservices design.
    c) **Security:** JWTs are signed to ensure their integrity. Only the server can validate the signature, which is generated using a secret key by the server when signing the token. This prevents manipulation of the token's content.

B. Solutions for DoS Attacks

A Firewall can be used to prevent DoS attacks on APIs. A cyberattack known as a denial of service (DoS) attack occurs when a hacker purposefully overloads a computer system, network, or service with a large volume of traffic, requests, or malicious activity. A DoS attack seeks to prevent the target from operating normally so that its intended users cannot access it. To prevent a DOS attack we can implement the following best practices:

1) **Rate Limiting:** Use rate limiting to regulate the number of requests an individual can submit in a certain period. By implementing this, you can stop a single user from sending too many queries to your API.
2) **Blacklisting and Whitelisting:** A firewall can be used to blacklist or whitelist IP addresses and domains. Blacklisting blocks traffic from known malicious sources, while whitelisting allows traffic from known trusted sources.
3) **Logging and monitoring:** A firewall can log and monitor traffic for suspicious activity. This information can be used to identify and block attacks in progress.

C. Solutions for SQL Injection Attacks

SQL Injection attacks can be mitigated by implementing necessary Security Headers like Content Security Policy. This can also help reduce Cross-Site Scripting attacks.

1) **Content Security Policy (CSP):** We can declare which content sources are permitted to be run on a web page using the security header CSP. Limiting the execution of scripts and other resources to trustworthy sources, aids in the prevention of XSS attacks.
2) **Refer-Policy:** Security headers are useful in preventing data leaks that could accidentally reveal data suitable for SQL injection attacks. To restrict the disclosure of sensitive information in URLs, the "Referrer-Policy" header, for instance, can govern what information is provided in the "Referer" header.
3) **XSS Prevention:** By reducing the risk of XSS attacks through CSP, we can directly reduce the risk of SQL injection. If an attacker can't inject malicious scripts, it is more challenging to execute the sequence of actions necessary for SQL injection.

D. Solutions for Data Exposure

Data Exposure is a serious security threat when API security is involved. We can implement payload encryption to mitigate the data leak risk in the APIs. The following methods for Encryption can be considered for effective API security:

1) **TLS / SSL:** To encrypt data being sent between clients and the API server, we can use TLS/SSL. This guarantees that information sent across the network is private and safe. Utilize safe cipher suites and robust encryption mechanisms.
2) **Symmetric and Asymmetric Encryption:** Symmetric and Asymmetric Encryption: It is best to use both the encryption for efficiency. Symmetric encryption is widely use for data encryption while the asymmetric encryption is use for making the key secure and digital certificate.
3) **Secure Key Rotation:** Secure Key Rotation: encryption keys can be rotated regularly to dial down the risk of key exposure. We should make certain the key is in transition while changing encryption keys.

IV. New Suggestions

Unusual patterns of data access can be recognized by User Analytical Behaviour (UBA). One red signal can be if a person who usually only accesses their own data suddenly tries to access data belonging to other users. UBA has the ability to record the occurrence, set off alarms, and maybe prevent more access to private information. It can be used as combined solution for all the attacks such as Unauthorised User, Dos attack, Injection Attack, Data Exposure.

In the framework of web APIs, UBA can enhance security in the following ways:

1) **Unauthorized Access:** UBA helps us to identify abnormal login patterns and access attempts, failed login attempts done repeatedly, Try to login and access the data from unfamiliar location, or the use of suspicious device. the system can trigger a warning to the user and notify the administrators about this unusual activity, it can also block the user temporarily.
2) **DoS Attack:** UBA can easily monitor network traffic and behavior to analyze abnormal spikes in requests or traffic pattern. It may not prevent the initial flow of traffic, it can recognize there is something suspicious about this traffic and can trigger corrective measures so that it can prevent the a DDos attack from happening.
3) **Injection Attack:** UBA can detect the anomaly pattern of inject attack and help to prevent it before the attacker can access the data and try to alter it, such as sql injection attack and cross-site scripting. If an API detect a known patterns of input, the system can block the response and log the warning and also notify the authorities.
4) **Data Exposure:** UBA can prevent and detect the malicious data access pattern. For example, if a user who is accessing his/her data suddenly wants to access the data which belongs to other user, this can raise a red flag. UBA sends alerts, also log the incident, and block further access to sensitive data.

While applying UBA into your web API security strategy improves your power to detect and respond to unauthorized user, DDos attack, data breaches, and injection attack in real time. It can be developed as its learn from its analysis and can sharpen its accuracy even further.

V. Implementation Details

User Behavior Analysis (UBA) is a cybersecurity approach that emphasizes the understanding and monitoring the user behavior, within a system. It entails a systematic analysis of user activities, interactions, and access patterns to identify any irregularities or possible security risks. UBA surpasses traditional security measures by taking into account the context and historical behavior of users empowering organizations to proactively detect and handle security incidents.

Implementing User Behavior Analysis (UBA) involves several steps, including setting up the environment, monitoring user actions, defining baseline behavior, detecting anomalies, and taking specific actions when suspicious activities are detected.

We have utilized the Flask web framework for User Behavior Analysis (UBA) because of its flexible design, which makes it a great option, for creating API endpoints that track user actions like login, accessing data and making transactions. With its simplicity and ease of use, Flask enables rapid development and easy integration of behavior analytics engines and anomaly detection functions.

Following is the implementation for User Behaviour Analysis using the Flask framework:

1) Data Collection: Collect and store user activity data, including login times, IP addresses, devices used, the frequency of access, and the types of actions performed. In our Flask project, we executed data collection by systematically documenting user activity details at each API endpoint. Specifically, we set up three API endpoints (/login, /data, and /transaction). Access to the /data endpoint was restricted to a specific group of authorized users. We created a log_request_to_excel function and used Flask's request object to gather information, like user ID, timestamp, IP address, endpoint, and method. We need then to store this collected data in an Excel file using Python's Pandas library to ensure tracking of user interactions.

2) Analysis of Data Collected and Define Baseline Behaviour: We used scripting activities to mimic user interactions with API calls. Next, to find trends indicative of normal behavior, we carefully examined the user activity data stored in an Excel file. This analysis considered factors like IP addresses, login times, and specific actions taken. We performed both descriptive and visual analysis of the gathered data using Jupyter Notebook. Finally, using the patterns we observed, we established a baseline for each user that describes the expected behavior.

3) Behaviour Analytics Engine: We have developed an effective behavior analytics engine that can analyze and compare user activity data with the established baseline. We created an analyze_user_activity function which leverages API endpoint details including, time, IP address used, and type of action performed. The function systematically compares this information with the baseline. By examining these factors, the function adeptly identifies instances where the user behavior significantly diverges from the usual behavior. This approach allows for early detection and identification of anomalies with each API call, enabling proactive response to deviations in user activity patterns.

4) Anomaly Detection and Suspicious Activity Triggers: Upon every API call, the relevant details are passed to the aforementioned behavior analytics engine, which meticulously assesses any anomalies against the established baseline. When the analytics engine identifies suspicious activity associated with a specific API, it promptly raises a flag denoting an anomaly. This crucial information is then logged using the Python logging library, and the anomalies detected are recorded in an external file alongside the API logs. These comprehensive logs serve as a valuable resource for administrators, allowing them to examine and review any potentially suspicious activity and subsequently take appropriate actions as needed.

Example Scenario:
Consider a user utilizing the three created APIs in the following manner:

- The user typically logs in through the /login endpoint, utilizing a set of defined IP addresses within a specific organization.
- The user performs routine activities including accessing data through the /data endpoints and conducting transactions via the /transaction endpoints.
- The user typically engages in login and other actions during typical working hours.
- However, on a particular day, their account exhibits activity from a different IP address, irregular login times, attempts to access data through an unauthorized user or performs transactions exceeding the allowable limit.

In this scenario, the behavior analytics engine would identify anomalies in IP addresses, login times, unauthorized data access, and transaction count. It would then flag this activity as suspicious and log the details in a file. Administrators can review these files and implement additional security measures, such as locking the user's access and notifying the security team.

While UBA can greatly enhance security, it's most effective when combined with other security measures like firewalls, authentication and authorization controls, and monitoring solutions to provide a comprehensive defense against a wide range of threats.

VI. Results

We simulated a couple of user's daily activities, capturing and storing user activity data, including login times, IP addresses, the frequency of data access, and the types of actions performed. This data is logged into an Excel file. Utilizing this Excel file, we performed data analysis to identify the regular patterns in the activities of a particular user.

	Timestamp	User ID	IP Address	Endpoint	Method
0	2023-12-10 09:19:09	user1	127.0.0.1	login.login_route	POST
1	2023-12-10 09:20:10	user1	127.0.0.1	access_data.access_data_route	GET
2	2023-12-10 09:30:11	user1	127.0.0.1	make_transaction.make_transaction_route	POST
3	2023-12-10 09:33:34	user1	127.0.0.1	make_transaction.make_transaction_route	POST
4	2023-12-10 13:40:21	user1	127.0.0.1	login.login_route	POST
5	2023-12-10 13:45:09	user1	127.0.0.1	access_data.access_data_route	GET
6	2023-12-10 14:01:35	user1	127.0.0.1	make_transaction.make_transaction_route	POST
7	2023-12-10 14:14:02	user1	127.0.0.1	make_transaction.make_transaction_route	POST
8	2023-12-10 14:28:44	user1	127.0.0.1	make_transaction.make_transaction_route	POST
9	2023-12-10 16:30:11	user1	127.0.0.1	login.login_route	POST

Fig. 3. First 10 rows of the simulated user data

The first ten rows of the user data are shown in the figure 3. The columns "Timestamp," "UserID," "IP Address," "Endpoint," and "Method" are included in the table.

After performing a descriptive analysis of the data, we were able to identify distinctive characteristics for every table column as shown in figure 4.

	Timestamp	User ID	IP Address	Endpoint	Method
count	22	22	22	22	22
unique	20	2	2	3	2
top	2023-12-10 14:14:02	user1	127.0.0.1	make_transaction.make_transaction_route	POST
freq	2	12	12	12	17
first	2023-12-10 09:19:09	NaN	NaN	NaN	NaN
last	2023-12-10 16:51:55	NaN	NaN	NaN	NaN

Fig. 4. Descriptive Analysis

Users initiate calls to various APIs, and subsequently, we determined the frequency of each API call for different users, visually representing the results.

Endpoint	access_data.access_data_route	login.login_route	make_transaction.make_transaction_route
User ID			
user1	3	3	6
user2	2	2	6

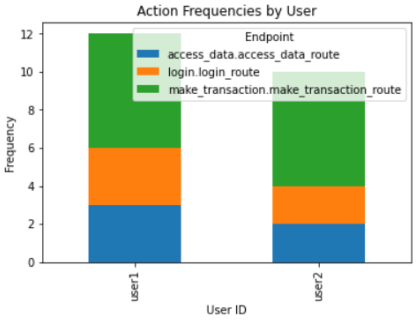

Fig. 5. Count of each Endpoint called by users

Based on the previous results given in figure 5, we established a baseline for the permissible limits of calls to the /data and /transactions endpoints. We defined an allowable limit of 10 transactions and 3 data access endpoint calls.

Following on, we generated a time series plot of the information. We were able to determine through plot analysis as seen in figure 6 that the user regularly logs in and performs actions within a given time frame. This insight contributed to creating another baseline rule that allows users to log in and perform various actions in a particular time range.

Additionally, we established another baseline rule by defining a set of specified IP addresses commonly used by users within the organization. This identification was based on the unique IP addresses observed in the collected data.

From the above descriptive and visual analysis we were able to define the following baseline rules:

- Login Time Range: Defined from 9:00 AM to 4:00 PM, representing the typical working hours for user logins.

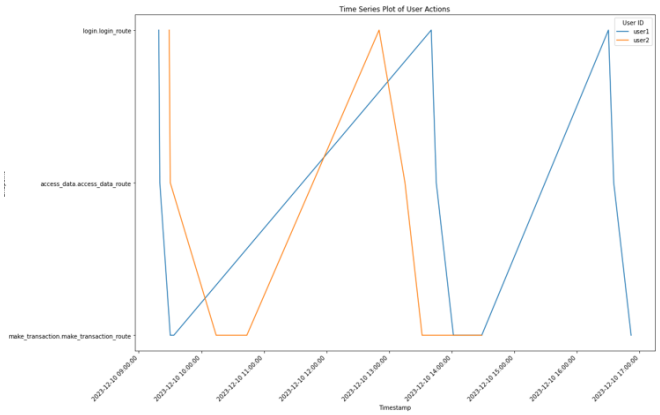

Fig. 6. Count of each Endpoint called by users

- IP Address Range: Specified IP addresses allowed for user activities are within the range '127.0.0.1' to '127.0.0.9'.
- Maximum Data Access per Day: Users are allowed to access a maximum of 3 data endpoints per day.
- Maximum Transactions per Day: Users are allowed a maximum of 10 transactions per day.

In the behavior analytics engine, we integrate and configure these baseline behaviors, particularly in the analyze_user_activity function. Then, each time an API call is performed, the necessary data is sent to this function, and it is compared to the predefined baseline behavior.

We then simulated malicious activity

VII. Conclusion

We conclude that by implementing Use Behavior Analysis we can cumulatively countermeasure all the attacks, such as Unauthorized users, DDoS attacks, Injection attacks, and Data exposure. In this project we were able to analyze the user-specific pattern and form a base rule from there we can monitor the user behavior detect any anomaly and generate a warning in its correspondence. This project helped us to realize the importance of security for web APIs and a great learning opportunity. In the future, we can make many advancements to this for more efficiency and can induce machine learning.

Acknowledgment

Our study was guided by the wise counsel and unwavering support of Dr. Shamik Sural. His wisdom greatly influenced the direction this work took intellectually. A huge thank you to Rutgers University for providing the funding, cutting edge facilities, and instruments necessary to make this research possible.

We are grateful to the amazing people who generously donated their time and knowledge for our research. Their participation changed everything and allowed our initiative to go deeper and farther. And a huge thank you to my family and friends for supporting me during the hard times.

Their confidence in me helped me persevere through the difficulties I encountered while conducting my study. This sincere gratitude attests to the vital part these individuals and organizations played in creating This remarkable journey took place. They have together had a significant influence on the direction of this investigation.

WORKLOAD DISTRIBUTION

Name	Task Completed
Ashish Pardeshi	Introduction Existing Solution Literature Survey
Jatin Thakkar	Abstract Problem Definition Graphs
Divesh Gadhvi	New Suggestions Implementation Details Literature Survey

REFERENCES

[1] D. Fett, P. Hosseyni and R. Kusters, ”An extensive formal security analysis of the OpenID financial-grade API”, 2019 IEEE Symposium on Security and Privacy (SP), pp. 453-471, ISSN 2375-1207.

[2] D. S. Islamiati, D. Agata and A. R. A. Besari, ”Design and implementation of various payment system for product transaction in a mobile application”, 2019 International Electronics Symposium (IES), pp. 287-292.

[3] P. Solapurkar, ”Building secure healthcare services using OAuth 2.0 and JSON web token in IOT cloud scenario”, 2016 2nd International Conference on Contemporary Computing and Informatics (IC3I), pp. 99-104.

[4] ”State of the internet / security — retail attacks and API traffic (volume 7 issue 4)” 2022.

[5] M. Biehl, API Architecture: The Big Picture for Building APIs, vol. 2. 2015

[6] J. Higginbotham, Designing Great Web APIs, O’Reilly Media Inc. 2021

[7] A. Masood and J. Java, ”Static analysis for web service security - tools techniques for a secure development life cycle”, 2015 IEEE International Symposium on Technologies for Homeland Security (HST), pp. 1-6.

[8] R. Barnett and E. Shuster, ”PART 2: THE DARK SIDE OF APIS”

[9] L. Murphy, M. B. Kery, O. Alliyu, A. Macvean and B. A. Myers, ”API designers in the field: Design practices and challenges for creating usable APIs”, 2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pp. 249-258. 2019

[10] Dieste Tubío, A. Grimán, and N. Juristo, ”Developing search strategies for detecting relevant experiments for systematic reviews”, vol. 14, pp. 513-539. 2018

[11] A. Masood and J. Java, ”Static analysis for web service security - tools techniques for a secure development life cycle” 2016