2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing Workshops (CCGridW) | 979-8-3503-0208-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/CCGridW59191.2023.00044

API Traffic Anomaly Detection in Microservice Architecture

Sowmya M
Dept. of CSE
PES University
Bengaluru, India
sowmya7501@gmail.com

Ankith J Rai
Dept. of CSE
PES University
Bengaluru, India
ankithrai2001@gmail.com

Spoorthi V
Dept. of CSE
PES University
Bengaluru, India
spoorthi.vasantharaju@gmail.com

MD Irfan
Dept. of CSE
PES University
Bengaluru, India
mohammedmusfar786@gmail.com

Prasad B Honnavalli
Dept. of CSE
PES University
Bengaluru, India
prasad.honnavalli@gmail.com

Nagasundari S
Dept. of CSE
PES University
Bengaluru, India
snagasundari5@gmail.com

***Abstract*—In the current Digital Age, data is an important asset that is constantly targeted in cyberattacks. Attackers make use of vulnerabilities in the application design to perform data theft. Therefore, there is a need to implement an intrusion detection mechanism that is specific to the application architecture. The Microservices Architecture is predominantly used by organizations to develop their software applications. This application design architecture is a group of individual services that interact through Application Programming Interfaces (APIs). As the number of API endpoints increases, there is an increase in the attack surface for hackers to exploit the application. The activity at these endpoints and API calls can be monitored to check for anomalies, which indicates abnormal behaviour. An API call refers to a request made to an API endpoint. Multiple API calls among the services generate API traffic in the application. This traffic can be analyzed for detecting unusual behaviour. In this paper, a machine-learning based technique, API Traffic Anomaly Detection (API-TAD), that detects anomalies in API traffic at two levels – a generalized level, and an application-specific level is proposed. This makes it a more efficient and accurate anomaly detection, not only in the network layer of the OSI model, but also in the application layer.**

Keywords—API traffic, Anomaly detection, Microservices, Denial of Service (DoS) detection, Machine Learning models, Microservice architecture, Data breach, bagging, RandomForest.

I. Introduction

The monolithic architecture is the most basic type of architecture where all the functionalities are clubbed into a single unit. However, this possesses a disadvantage of slow development. As opposed to the monolithic architecture, the Microservices architecture combines the functionalities of an application into highly cohesive, loosely coupled, individual units called microservices. There has been a major shift from the monolithic architecture to the microservices architecture in applications, owing to increased scalability, continuous integration and optimized business functionality. Some of the popular applications that have adopted this architecture are Netflix, Amazon and Spotify.

The Microservices architecture has a larger number of APIs relative to the monolithic architecture. A single mismanaged or forgotten API proves to be a huge loophole and acts as an insecure doorway for attackers to sneak into the services and steal classified data from the application databases leading to major cyber incidents like data breaches [1]. A data breach is a situation in which information is taken from a system without the owner's knowledge or consent. According to the Identity Theft Resource Center, data breaches increased by 14% in the first quarter compared to the same period last year (ITRC). The most recent surge follows a 68 percent jump in breaches from 2020 to 2021, which broke the previous record set in 2017 by 23 percent [2]. There is a need to step up the security measures implemented in microservices to detect and prevent such attacks.

APIs can prove to be an important source of information to track unusual behavior. Microservices generate an enormous amount of API traffic, which can be analyzed for abnormal behaviors. Therefore, it is essential to monitor API traffic continuously, detect abnormal intrusion and prevent the attack from escalating any further. Rule based methods cannot keep up with the changes in the application that takes place regularly in a development environment in an organization. Automation through machine learning is needed to solve these problems.

Most of the Anomaly detection method that have been researched until now are based on network traffic to detect abnormal behaviors in the network layer in the OSI model, but these methods fall short in detecting the same in application layer. The traditional methods may be effective for general network traffic but not API traffic since API traffic is specific and has distinctive features and behaviours that can differ from other network traffic [3]. API abuses can be more accurately detected by monitoring traffic in the Application layer which gives more insight about the traffic to detect potential attack. Existing Intrusion Detection System (IDS) and Intrusion Prevention System (IPS) are one of the legacy

979-8-3503-0208-0/23/$31.00 ©2023 IEEE
DOI 10.1109/CCGridW59191.2023.00044

methods that use rule-based and signature-based methodologies. These techniques, however, do not take into consideration sneaky attacks and malevolent behaviours. In other words, these methods don't keep an eye on nefarious activities within API traffic. Hence there exists dire need to detect the malicious behaviours of API traffic.

Thus, the key contributions focused are summarized as follows:

- The API anomalies in focus are data breach and DoS attacks. The data breach is detected with machine learning models while the DoS attack is detected with Pruned Exact Linear Time (PELT) algorithm.
- A sample Banking application with basic functionalities is built. API logs from this application is collected for further analysis and development.
- A recursive compressor is implemented which extracts the necessary features from the API logs from each service, to a set of parameter values. These values are extracted every minute and used to monitor the traffic.
- The API traffic is monitored at two levels. The initial layer (Layer 1) is specific to large scale attacks when the application has large resources. The next layer (Layer 2) is specific to the application itself.
- The Layer 1 detects the abnormal API traffic with the aid of RandomForest model. This model is trained with a real-time dataset consisting of 34,400 records.
- The Layer 2 works on a synthetic dataset which is generated by introducing new parameters. The Bagging method of machine learning is trained with this dataset to detect the attack in case it bypasses Layer 1.
- Comparison of different ML models like RandomForest Classifier, Decision tree, Naive Bayes, Support Vector Machine (SVM), Logistic Regression Classifier, ensemble methods is done to analyze the performance of the models.
- The PELT search method, a change detection point algorithm is used to detect DoS attack on the application.

This paper is organized as follows: Section II summarizes the related work. Section III gives an insight into the Microservices Architecture and the Banking Application from which the API logs are collected. Section IV elaborates on the API logs features, its collection and parameter generation. Section V gives details about the dataset used. The anomaly detection mechanism is explained in Section VI. Sections VII and VIII elaborate on the comparison of different machine learning models and the performance of the algorithm. The proposed method is compared against the existing methods mentioned in Section II. The possible future work is discussed and the work is concluded in Section IX.

II. Related Work

Several works related to security in microservices architecture, including the details on security frameworks, security auditing is discussed in the literature. Matt McLarty [4] discusses all the Access control mechanism techniques in Web APIs in microservice architecture. Few methods that are discussed are Network-Level Controls using SSL/TLS Communication, application-level controls using JSON web tokens for authentication and authorization bypassing requests through gateway. New emerging approaches such as service mesh and Domain Hierarchy Access Regulation for Microservice Architecture (DHARMA) is also discussed to incorporate access control. A study [5] talks about utilising Service Mesh Framework for API Security and Management, however the author proposes a security framework using service mesh, but not an intrusion detection mechanism in an attack scenario.

The study [6] mentions about the significance in learning about API security with respect to OWASP Top 10 API Vulnerabilities [7], proposes an API security learn- ing/testing environment called Vulnerable Academic Informa- tion System (VAIS) based on the OWASP API Security Risks. Tang, Longji, Liubo Ouyang, and Wei-Tek Tsai [8] proposes three factor API security architecture in cloud architecture using OAuth and OpenID authentication methods. A literature review paper by Mathijssen, Max, Michiel Overeem, and Slinger Jansen [9] discusses about various capabilities and practices of API management like Authentication, Security, Monitoring, Load Balancing, Access Control, OAuth Authentication. Study based on DoS prevention [10] proposes a defense mechanism that prevents network-based DoS traffic in cloud REST APIs.

Study by Ronghua Sun, Qianxun Wang, and Liang Guo [11] discusses that machine learning methods are the way to detect anomalies in API traffic and states that the some reasons for API security concerns are out-of-date APIs, unauthorized users abusing APIs. A study [12], focusses on preventing XSS attack on the API gateway to gain control of a microservice and use that information to attack other microservices with similar image layers using moving target defense mechanism.

There has been very few works on anomaly detection based on API traffic. One of the studies[3] that proposes an SVM-based technique for anomaly detection in API traffic, which recorded 83.5% accuracy and 7.3% false positive. Since real-time applications need to adapt to the new cyber security attacks it needs be faster in detection of these attacks, Machine learning is a great option for API managers and developers as it can perform security analysis, make crucial judgments, and launch corrective responses. However, this method does not provide more details on the microservice architecture on which the proposed model has been demonstrated on. It does not detect which API endpoints are under attack. A study by Deshani Geethika [13], generates a dataset based on CPU consumption in high performance API Gateway that is close to production environment using WSO2 open- source API Gateway. Ifthikar, Arshardh, et al [14] follows a layered architecture to detect anomalous API Traffic. A hybrid approach using both supervised and unsupervised learning techniques was proposed. The layered architecture includes three layers. The first layer checks the API trends and detects if there is any behavior that is unusual from normal. The second layer analyses the neighbouring anomalous requests through a granular approach using hybrid machine learning method using K-Means and Local outlier Factor algorithm. The third layer does a per-request analysis, it is similar to in-depth classification technique that is running on background.

III. Microservices Application

The microservices of an application communicates with the application users through a reverse proxy, which may be an API Gateway. All the client requests are forwarded to this gateway, which then forwards the requests to specific services with the help of the API Discovery Service. The services fetch the appropriate response from the database and forward the data to the respective clients through the gateway. These requests and responses are directed to and from the APIs at the gateway and the services respectively.

The environment developed involves a basic Banking Microservice Application that is built using Spring Boot Technology. The application is hosted in a virtual machine instance of 8GB RAM using Azure Cloud Services. This application has the following services and the HTTP REST API endpoints exposed:

- *Account Service*: This service is used to create and get details of the account.
  - POST api/user/
  - GET api/user/id
- *Transaction Service*: This service performs transactions and is used get transaction details of each account.
  - POST api/transaction
  - GET api/transaction/account-number
- *Banking Service*: This service checks if accounts exist and updates the final balance in respective accounts.
  - POST api/account/trans
- *API Gateway*: This service is a reverse proxy that routes request to respective services.
- *Discovery Service*: This service is used to discover all the available services.
- *MySQL Database*: It stores account and transaction details of the application.
- *FireStore Database*: It stores the features extracted by account, transaction and banking services. The features are further used to populate parameters for Anomaly detection.
- *Monitoring Service*: This service is used to analyze API traffic and detect anomalous API traffic.

The above-mentioned services are integrated into Docker Compose for final deployment.

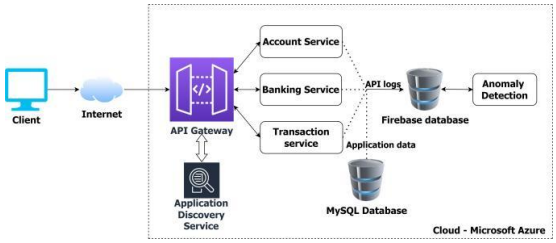

Fig. 1. API Traffic Anomaly Detection.

IV. API Traffic Analysis

A. Feature Engineering

Certain characteristics of API traffic are used as features to calculate mathematical parameters. These parameters are used as data points in the API dataset that classifies the traffic as normal or anomalous.

- *Client IP address*: The IP address of the client/browser that is making the API call.
- *API Endpoint*: The endpoint of each API that is accessed in the request.
- *Response Data Size in bytes*: The amount of data (bytes) that is sent back to the client with respect to the API call.
- *Status Code*: The resultant HTTP status Code of the API call response.
- *Timestamp*: The timestamp of the request.
- *Request Type*: The type of HTTP request.

B. Parameter Engineering

The features listed above are collected for every minute and the following parameters are calculated for the classification.

- *inter-api-access*: Median of all differences between the timestamps of two consecutive requests.

$$inter\text{-}api\text{-}access = Median\ (td) \quad (1)$$

where *td* is time difference between two consecutive timestamps. This parameter helps check how fast a sequence of calls are made. The lower the value indicates a greater possibility of a bot attack. The median is chosen out of all the measures of central tendency, because it is least likely to be affected by outliers.

- *api-access-unique*: Ratio of distinct number of APIs accessed to total number of calls made in that session.

$$api - access - unique = \frac{No\ of\ Distinct\ APIs}{Total\ No\ of\ APIs}$$

The value decreases when total number of calls increases and number of distinct API calls decreases. Lower value indicates that fewer APIs are targeted with larger abnormal number of requests. This can be used to detect possible attack with respect to APIs.

- *total-calls*: Total number of API calls made. If the value of this parameter increases drastically, it indicates to a potential Denial-of-Service or a request flooding attack.
- *max-api-call*: Maximum of total number of calls made to each API endpoint.

$$max\text{-}api\text{-}call = max\ (n[i]) \quad (3)$$

where *n[i]* - no of calls made to an API *i*. A drastic increase in the number of calls made to one specific API endpoint may signal the possibility of a vulnerable API that is being targeted by attackers for a potential data breach.

- *data-out-rate*: The amount of data sent out in responses in bytes.

$$data\text{-}out\text{-}rate = \sum_{i=0}^{total-calls} response\_size_i \quad (4)$$

where *response_size* is response data size in bytes. If the total amount of data sent out to that IP address is suspiciously high, it is an indication of a possible data breach.

- *num-failed*: The number of failed calls with status codes *4xx* (client error).
  If the number of responses with status codes *4xx* are unusually large, it is safe to assume that the API requests are being tampered and tried on the application's API endpoints to find vulnerabilities within, which is what most attackers do, to gain access into the application and its databases.

C. Logging and Collection of parameters

The services discussed in the previous section log the parameters discussed in Feature Engineering section to a Firebase firestore database hosted on cloud. The logs are collected from the database for time interval of one minute.

These logs are compressed into parameters listed in Parameter Engineering. The compression is done and sent to the trained model for the prediction.

V. Dataset

The dataset used in the Level 1 is a real-time dataset. The dataset consists 34,400 records [15]. The table below represents the type of traffic and the respective number of records in the dataset. This dataset consists of data records where attackers abuse API access in an effort to take advantage of the business logic of these APIs. Business Logic vulnerabilities have been a major cause for data breaches. The features considered are inter-api-access, api-access-unique, max-api- call.

Table I: Realtime dataset.

Behavior Type	No. of Records
Attack	1,310
Outlier(abnormal)	24144
Normal	8,946

The dataset is imbalanced with respect to the number of records it contains for each category of traffic. Moreover, the microservice application is large-scale application that has more than 100 API endpoints. The resources consumed differ with applications of each organization. The number is too huge when compared to the implemented banking application. Therefore, for Level 2 detection, a dataset is generated that is specific to the application. Considering the above reasons, an anomaly detection method is proposed that can be used to detect anomalies through machine learning algorithm to detect and classify traffic through extracted features from the application as normal, data breach or Denial-of-Service attack. Threshold assumed for the generation of synthetic dataset: The parameters for normal condition are total number of calls less than 40 for one API endpoint. To make sure that there are no data anomalies, these parameters are chosen after a number of performance tests and consideration of the application's resources. For data breach attack, a custom python script to fetch the information from the API is used to generate the dataset. The dataset consists of 898 records. The features considered are inter-api-access, api-access-unique, max-api-call,data-out-rate, num-failed-responses. The table below represents the type of traffic and the respective number of records in the dataset.

Table II : Synthetic dataset.

Behavior Type	No. of Records
Attack	501
Normal	397

VI. Anomaly Detection In API Traffic

The Layer-1 and Layer-2 introduced in the paper are used to detect a data breach attack. Layer-1 consists of a Random Forest model trained with a real time dataset that is relevant for detecting data breach attacks caused due to business vulnerabilities. This dataset is derived from an application that consists of about 100 APIs whereas the application used in the paper consists of five API endpoints. The scale of the application used for creating real time dataset is large when compared to the application used in this paper. The attacks performed on the microservice application were not detected by Layer-1. Hence synthetic dataset was generated to train a model that is application specific and more relevant considering the scale and resources of the application. The dataset generated included new parameters like data-out-rate and number of failed requests. Although layer-1 did not detect any attacks performed, it is still relevant to be prepared for unknown zero-day attacks and advanced attacks conducted with higher efficiency (high compute resources) at a faster rate. Hence it has been sustained in the detection mechanism. All the parameters mentioned in parameter engineering are extracted by the recursive compressor. The API traffic is represented by these parameters using which the detection is made. These traffic parameters are then sent to Layer 1 i.e generalized layer detection which comprises of RandomForest model trained with real-time dataset. Upon detection as attack, API endpoint under attack is displayed. If the traffic is detected as normal or outlier, it is further passed down to Layer 2 detection. Layer 2 i.e application-specific detection comprises of Bagging method-based model trained with synthetic dataset. If the traffic is further detected as an attack, the same output as Layer 1 detection is followed. No action is taken in case detected as normal traffic. The DoS detection module runs parallelly to the main detection module. Variable index has the starting point of the traffic that needs to be analyzed is passed
as input to the module.

A DoS attack occurs when the application slows down and starts showing errors of resource exhaustion. Apache JMeter [16] is used for simulating the attack by flooding the application with requests with 25-30 concurrent threads. As the resource exhaustion occurs there is drastic decrease in parameters like data-out-rate and total-calls and increase in parameters like inter-api-access. These values are combined into one single value using equation (5).

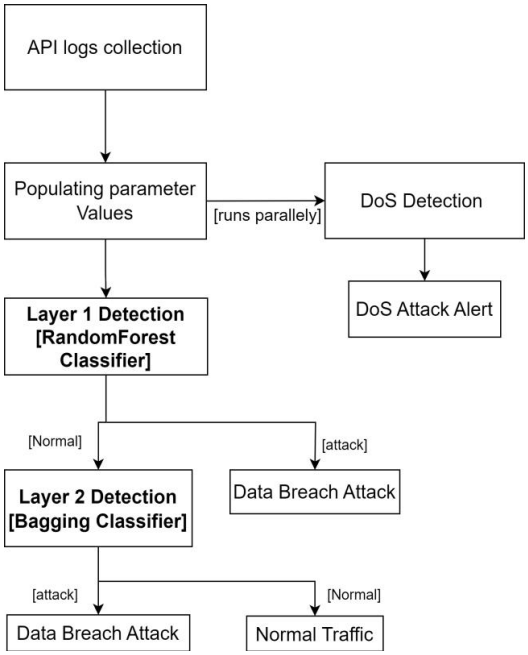

Fig. 2. Work Flow Diagram.

Algorithm 1 API Traffic Anomaly Detector

```
procedure AnomalyDetector(parameter[5], api)
    result1 = LOneMLClassifier(parameter[3], api)
    if result1 ≠ attack then
        result2 = LTwoMLClassifier(parameter[5], api)
        if result2 = attack then
            return "{ApiEndpoint} is under attack"
        else
            return "Normal Traffic"
        end if
        if parameter[totalCalls] ≥ 1000 then
            dosDetect = DosDetection(requestRecords)
            if dosDetect = True then
                return "DoS Alert!"
            else
                pass
            end if
        else
            pass
        end if
    else
        return "{ApiEndpoint} is under attack"
    end if
procedure LowFreqReqDetector
    features = ExtractFeatures(30mins)
    ip_records = Compress_IPaddress(features)
    for i in ip_records do
        if i.total_calls ≥ 15 then
            return "Possible Attack from i.ip_address"
```

$$\text{DoS factor} = \frac{\text{data-out-rate x total-calls}}{\text{inter-api-access}} \quad (5)$$

The DoS factor decreases drastically when there is a DoS attack due to combined effect of data-out-rate, total-calls and inter-api-access. In this implementation, 1000 or more number of requests per minute triggers the DoS detection module. The DOS detection module waits for 20 minutes to collect the compressed records.

PELT search method [17], a Point Change Detection algorithm is used to find anomalous states or sequences in a time series. This algorithm checks for a possible DoS attack and displays the same. This method checks for a change in the pattern of DoS factor with respect to time.

Algorithm 2 API Traffic Anomaly DoS Detector

```
procedure DoSDetector(index)
    Sleep(20 minutes)
    records = fetchLastTwentyRecords(index)
    result = PeltSearchMethod(records)
    return result
```

The number of requests made by a specific IP address in last thirty minutes is kept in check. If the number of requests for one IP address is more than or equal to 15, a possible attack from that address is detected. This helps in detecting unusual request traffic in low frequencies.

VII. Classification And Validation

In Layer 1, the training dataset contains data classification for normal and anomalous behavior of API traffic. The parameters mentioned in the previous section are the features mentioned in the dataset.

Machine learning models like RandomForest, DecisionTree, KNN, Naive Bayes, LogisticRegressionClassifier, SupportVectorMachine are used to train and test the data against the dataset. While these models have their advantages, ensemble methods are used to bring about a higher rate of accuracy into the classification. The ensemble methods used include Bagging, Boosting and Voting.

In Layer 1 detection, RandomForest model is chosen since it gave the highest accuracy i.e 91.63%. In Layer 2 detection, bagging model is trained on synthetic dataset, which is an ensemble method. Bagging is chosen due to its high accuracy i.e 97.77%. The synthetic dataset is balanced with respect to two classes. A two-layer mechanism is used to detect the attack. The values captured in recursive compressor is sent to Layer 1. The respective API endpoint is identified and detected as vulnerable if the API traffic is detected as anomalous following the algorithm 1. If it is predicted as normal, no action is taken.

Table III: Realtime dataset results.

Model	Accuracy	Response Time	F1-score
RandomForest	91.63%	1.226s	0.916
DecisionTree	89.72%	0.002s	0.897
KNN	85.63%	0.171s	0.856
NaiveBayes	64.11%	0.002s	0.641
LRModel	91.63%	1.17s	0.916
SVM	82.70%	1.49s	0.827
Voting	89.05%	1.397s	0.891
Boosting	86.13%	0.049s	0.861
Bagging	64.11%	0.0609s	0.641

Table IV: Synthetic dataset results.

Model	Accuracy	Response Time	F1-score
RandomForest	97.22%	0.009s	0.969
DecisionTree	96.11%	0.002s	0.958
KNN	96.11%	0.004s	0.956
NaiveBayes	96.66%	0.00016s	0.962
LRModel	97.22%	0.001s	0.969
SVM	97.22%	0.003s	0.972
Voting	97.22%	0.011s	0.969
Boosting	97.22%	0.006s	0.969
Bagging	97.77%	0.005s	0.975

VIII. Experiments And Results

In this section, findings of research and project are discussed. A real-time dataset that detects attacks against applications with a very large number of APIs is used for Layer- 1 anomaly detection. Numerous instances of requests to the application were used to create a synthetic dataset, specific to our application. Simulation of many instances of data breach attack has been done. During this attack, a flood of requests to the vulnerable API is generated. This dataset is used for Layer-2 anomaly detection, that is specific to the application. Multiple models have been used to train and test the datasets. It was observed that the RandomForest Model registers the highest accuracy in Layer-1 anomaly detection. Bagging, an ensemble of multiple DecisionTree Model Classifiers has registered the highest accuracy in Layer-2 anomaly detection. Random Forest model is observed giving a high response time when compared to other models. This is because it is a collection of various decision trees. The bulk of the results from each decision tree classifier are used to determine the outcome in random forest. As a result, it takes much longer than other models to produce a result. Naive Bayes is observed to record the least response time in both the layers because it generally has low time complexity.

The confusion matrices of the respective models are depicted in Fig. 3 and 7. A confusion matrix is a presentation of the number of correct (true positives and true negatives) and incorrect (false positives and false negatives) predictions with respect to each class.

Fig. 3 shows the confusion matrix of the RandomForest model in Layer1 detection. 7615 records out of 8311 records has been correctly classified. Here 171 records out of 269 records has been correctly classified as attack, 5495 records out of 5790 records are correctly classified as outlier and 1950 records out of 2252 records has been correctly classified as normal. The recall value of the model is 0.916.

Fig. 7 shows the confusion matrix of the Bagging model in Layer2 detection. 176 records out of 180 records have been correctly classified. Here 78 records out of 82 records have been correctly classified as normal and 98 records out of 98 records are correctly classified as attack. The recall value of the model is 0.951.

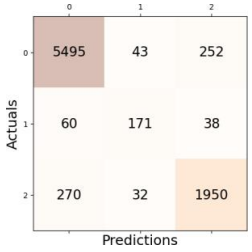

Fig. 3. Confusion Matrix of RandomForest model (realtime dataset)

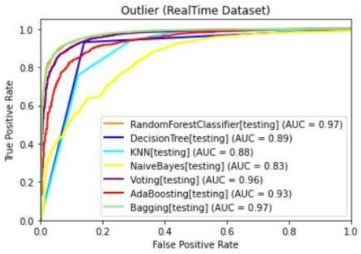

Fig. 4. RoC curve for the class 'outlier' (realtime dataset)

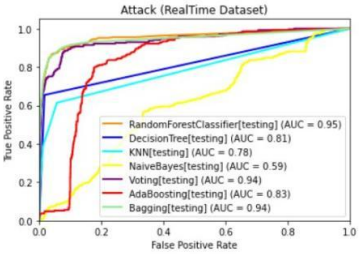

Fig. 5. RoC curve for the class 'attack' (realtime dataset)

The ROC-AUC curves differentiate between the various ML models. ROC (Receiver Operating Characteristic) is a probability curve. AUC (Area Under Curve) represents the degree of separability. A higher AUC represents better performance. In Layer 1, the RandomForest classifier shows the highest AUC curve for the class 'outlier' with a value of 0.97, a value of 0.95 for the class 'attack' and a value of 0.97 for the class 'normal'. This is represented in Fig. 4, 5 and 6. In Layer 2, the Bagging model gives the highest score for both 'normal' and 'attack' classes. The same is represented in Fig. 8 and 9.

The combined accuracy for both the layers is 96.54%. As shown in Fig. 10, the proposed model represented in blue gives an F1-score of 0.97, the SVM-based approach is represented in green [3], the Hybrid-approach is represented in red [1 4] mentioned in the related work have F1-scores of 0.964 and 0.89 repectively.

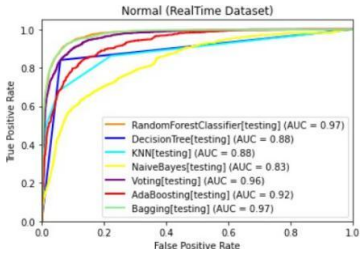

Fig. 6. RoC curve for the class 'normal' (realtime dataset)

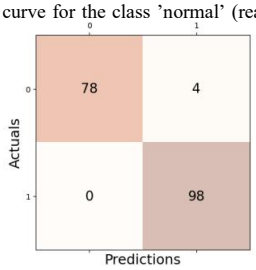

Fig. 7. Confusion Matrix of Bagging model (synthetic dataset)

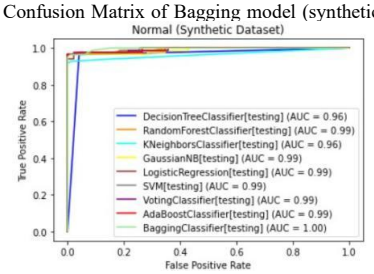

Fig. 8. RoC curve for the class 'normal' (synthetic dataset)

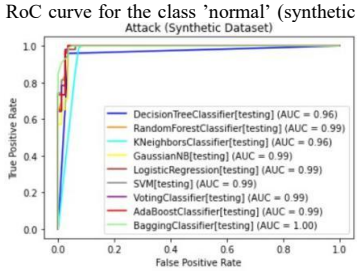

Fig. 9. RoC curve for the class 'attack' (synthetic dataset)

IX. Conclusion

In this paper, a two-layer API-TAD is proposed. This can be used by organizations to counter against generic and application-specific data breach and DoS attacks.

Currently, API-TAD is focused on detecting anomalies and possible attacks i.e. it acts as an IDS. The work can be extended to prevent the potential attacks, thus acting as an IPS. This could help the applications prevent zero-day attacks, one of the most dangerous forms of attacks.

The API-TAD is also focused only on HTTP GET requests and responses. The work can be further extended to HTTP

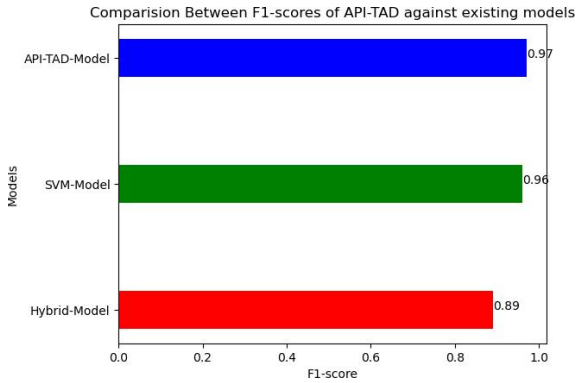

Fig. 10. Comparison of F1-scores of different methods

POST requests and responses. It can also include other types of HTTP calls and API calls of other protocols. Suspicious behavior with fewer HTTP requests might pass undetected. The synthetic data generated is application specific and is different for different applications. This dataset needs to be generated by the organization and they will have to set their own thresholds. Furthermore unsupervised learning methods could be used to enhance the accuracy.

API-TAD uses machine learning models along with rule based approach. It proves to be an efficient approach that identifies attack patterns and trends and can handle large applications and continuous improvement.

References

[1] Ramaswamy Chandramouli. Microservices-based application systems. *NIST Special Publication*, 800(204):800–204, 2019.

[2] Dr. Rey LeClerc Sveinsson. Top 10 data breaches so far in 2022 [online]. available: https://ermprotect.com/blog/top-10-data-breaches-so-far-in-2022.

[3] Gaspard Baye, Fatima Hussain, Alma Oracevic, Rasheed Hussain, and SM Ahsan Kazmi. Api security in large enterprises: Leveraging machine learning for anomaly detection. In *2021 International Symposium on Networks, Computers and Communications (ISNCC)*, pages 1–6. IEEE, 2021.

[4] Matt McLarty. Securing microservice apis [online]. available: https://www.oreilly.com/library/view/securing-microservice-apis/.

[5] Fatima Hussain, Rasheed Hussain, Brett Noye, and Salah Sharieh. Enterprise api security and gdpr compliance: Design and implementation perspective. *IT Professional*, 22(5):81–89, 2020.

[6] Muhammad Idris, Iwan Syarif, and Idris Winarno. Development of vulnerable web application based on owasp api security risks. In *2021 International Electronics Symposium (IES)*, pages 190–194. IEEE, 2021.

[7] OWASP. Owasp api security project [online]. available: https://owasp.org/www-project-api-security/.

[8] Longji Tang, Liubo Ouyang, and Wei-Tek Tsai. Multi-factor web api security for securing mobile cloud. In *2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, pages 2163–2168. IEEE, 2015.

[9] Max Mathijssen, Michiel Overeem, and Slinger Jansen. Identification of practices and capabilities in api management: a systematic literature review. *arXiv preprint arXiv:2006.10481*, 2020.

[10] Re´gio A Michelin, Avelino F Zorzo, and Cesar A De Rose. Mitigating dos to authenticated cloud rest apis. In *The 9th International Conference for Internet Technology and Secured Transactions (ICITST-2014)*, pages 106–111. IEEE, 2014.

[11] Ronghua Sun, Qianxun Wang, and Liang Guo. Research towards key issues of api security. In *China Cyber Security Annual Conference*, pages 179–192. Springer, 2021.

[12] Kennedy A Torkura, Muhammad IH Sukmana, Anne VDM Kayem, Feng Cheng, and Christoph Meinel. A cyber risk based moving target defense mechanism for microservice architectures. In *2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, pages 932–939. IEEE, 2018.

[13] Deshani Geethika, Malith Jayasinghe, Yasas Gunarathne, Thilina Ashen Gamage, Sudaraka Jayathilaka, Surangika Ranathunga, and Srinath Perera. Anomaly detection in high-performance api gateways. In *2019 International Conference on High Performance Computing & Simulation (HPCS)*, pages 995–1001. IEEE, 2019.

[14] Arshardh Ifthikar, Nipun Thennakoon, Sanjeewa Malalgoda, Harsha Kumara Moraliyage, Thamindu Jayawickrama, Tiroshan Madushanka, and Saman Hettiarachchi. A novel anomaly detection approach to secure apis from cyberattacks. 2022.

[15] Ravi Guntur. Api security: Access behavior anomaly dataset [online] available: https://www.kaggle.com/datasets/tangodelta/api-access-behaviour-anomaly-dataset.

[16] The Apache Software Foundation. Apache jmeter [online] available: https://jmeter.apache.org/.

[17] Charles Truong, Laurent Oudre, and Nicolas Vayatis. Selective review of offline change point detection methods. *Signal Processing*, 167:107299, 2020.