Artificial Intelligence and Cyber Security (AI-CyberSec)

A Novel Anomaly Detection Approach to Secure APIs from Cyberattacks

Arshardh Ifthikar[1], Nipun Thennakoon[2], Sanjeewa Malalgoda[3], Harsha Moraliyage [4], Thamindu Jayawickrama[3], Tiroshan Madushanka[2] and Saman Hettiarachchi[1]

[1]*Informatics Institute of Technology, Sri Lanka*
[2]*University of Kelaniya, Sri Lanka*
[3]*WSO2, Sri Lanka*
[4]*Centre for Data Analytics and Cognition, LaTrobe University, Australia*

Abstract

Digital transformation driven rapid movement towards the digital economy resulted in an elevated amount of cyber risks and cyber threats for nations. In this feat, RESTful Application Programming Interfaces (APIs) are identified as the core building block of the backbone network which connects billions of people, businesses, devices, data, and processes. With almost every nation moving towards digital transformation, organizations adopt strategies such as anything as a service (XaaS) model and APIs are becoming a vital aspect for any business functionality. The rising popularity of APIs makes them an attractive target for hackers and malicious users. The risk and complexity of cyberattacks are increasing exponentially as attackers continuously evolve with the technology and use novel Artificial Intelligence(AI) techniques to launch more sophisticated attacks. Cyberattacks on APIs are often subtle by nature, and the uniqueness of API access patterns makes it even harder to effectively detect them using traditional methods. Organizations are moving towards adopting AI techniques to defend against cyberattacks which enable the possibility to create systems that could learn the unique access patterns of APIs, by monitoring its API traffic and using the gained knowledge to identify the cyberattacks on APIs. Those systems could even dynamically adapt to the changes in API access patterns due to external factors like time. In this study, we propose a novel hybrid approach of anomaly detection and classification ensuring that the API services can be consumed as needed, but passively analyzing the flow to identify anomalies and potential cyberattacks to the APIs and alerting the network administrators of any anomalies detected and ensuring cybersecurity. Experiments have shown that our system is effective in detecting attacks on APIs with a high level of accuracy and reliability.

Keywords

API security, anomaly detection, machine learning, K-Means, random forests, Artificial immune systems, Autoencoders, AI driven cybersecurity,

AI-CyberSec 2021: Workshop on Artificial Intelligence and Cyber Security, December 14, 2021, Cambridge, UK
arshardhifthikar@gmail.com (A. Ifthikar); nipunsampath@gmail.com (N. Thennakoon); sanjeewa190@gmail.com (S. Malalgoda); h.moraliyage@latrobe.edu.au (H. M.); thamindudj.16@cse.mrt.ac.lk (T. Jayawickrama); tiroshanm@kln.ac.lk (T. Madushanka); saman.h@iit.ac.lk (S. Hettiarachchi)
https://arshardh.com/ (A. Ifthikar); https://www.linkedin.com/in/sanjeewa-malalgoda/ (S. Malalgoda); https://medium.com/@harsz89/ (H. M.); http://www.iit.ac.lk/mr-saman-hettiarachchi (S. Hettiarachchi)
0000-0002-1764-8079 (A. Ifthikar); 0000-0002-9568-3101 (N. Thennakoon); 0000–0001–6627–1477 (T. Madushanka)

1. Introduction

An Application Programming Interface (API) is a software-to-software interface that defines the contract for applications to communicate with each other over a network without human interaction. It simplifies programming by exposing only the objects or actions and abstracting the underlying implementation.Web API or RESTful service is an API over the web which can be accessed for message transmission and URIs for resource identification and interaction . They have grown rapidly due to the ubiquity of digital devices and the network infrastructure.The growing popularity of APIs makes it an attractive target for hackers and malicious users which draws our attention to ensure cybersecurity for these APIs. Injection attacks, Application-level Denial of Service (DOS) and Distributed Denial of Service (DDOS) attacks, broken authentication, and sensitive data exposure are prime examples of the threats directed at APIs. Even the big corporations like Venmo[1], T-Mobile [2], and Facebook [3] have recently experienced API security breaches. In general, every API has an access pattern unique to it. These access patterns may sometimes be unique to the API or sometimes to the user or sometimes both. This uniqueness makes it difficult to detect threats to API using traditional methods like static policies in an accurate and efficient manner. However, by using machine learning techniques, it would be possible to create systems that could learn the unique access patterns of an API by monitoring the API traffic and use that knowledge to identify the malicious API calls. Furthermore, these systems will be able to dynamically adapt to each API, as well as to the changes in its access patterns with external factors like time.

2. Related Work

Anomaly-based intrusion detection systems operate by defining the normal behavior of the system and detecting intrusions by observing patterns that deviate from established norms. The main benefit of this approach is the ability to detect zero-day attacks [4]. Anomaly-based IDS generally use unsupervised learning techniques for modeling normal behavior. Several prior studies have been conducted using different unsupervised learning techniques to implement anomaly-based intrusion detection systems.

Sipola et al. [5] proposed using diffusion maps and spectral clustering to detect anomalies on network logs. Their best accuracy and precision figures were 99.9% and 99.8% respectively. Fan [6] introduced a method based on hidden Markov models for anomaly detection of web-based attacks. They claimed that the detection of SQL injection attacks as anomalies is very effective, with a detection rate as high as 98%. Jin Wang et al. [7] proposed a novel anomaly-based HTTP-flooding detection scheme (HTTPsCAN), which can eliminate the influence of the web-crawling traces with the clustering algorithm. They had an average false positive rate of 2.5% and an average true positive rate of 100%.

Zolotukhin et al. [4] has considered the analysis of HTTP logs for intrusion detection. According to their experimental results, the Support Vector Data Description (SVDD) model had the highest detection rate (99.2%) for detecting anomalies using web resource while K-Means had the highest detection rate (100%) for detecting anomalies using query attributes and the DBSCAN had the highest detection rate (97.5%) for detecting anomalies using user agent. Juvonen

et al. [8] introduced an anomaly detection approach that uses dimensionality reduction techniques. They concluded that random projection works well when analyzing bigger datasets, and diffusion maps can be used for more accurate analysis on smaller sets of data. Peng et al. [9] proposed a clustering approach based on Mini Batch K-Means and PCA (PMBKM) for intrusion detection. From the experimental results, they have concluded that PMBKM is effective, efficient, and can be used for intrusion detection in big data environments.

The anomaly-based IDS also has some limitations when compared with signature-based IDS. The false alarm is generally higher in anomaly-based IDS [4] since it is possible that the detected anomaly may have been a legitimate outlier instead of an intrusion. Apart from that, they also cannot classify the intrusion types [10] like in the signature-based IDS.

Next, lets shift focus to the paper authored by Tong Lanxuan, Cao Jian, Qi Qing and Qian Shiyou bearing the title AIMS: A predictive web API invocation behavior monitoring system[11]. The problem they are looking into solving is the need to analyze personalized web API invocation behaviors. The main hurdle they are trying to overcome is to build a unified framework that is able to predict web API invocations by a user and using this model, to detect any anomalies in the web API invocation pattern. They have attempted to tackle this problem by using the K-Nearest-Neighbor classifier. They first convert the user's invocation data to a time series based on invocations and then uses permutation entropy to determine the randomness of this series and fed to the next steps. This information is first used to build an adaptive prediction model according to the active period of the users time series. They have tested this model with both fabricated and actual user invocation data. In conclusion, the authors state that their system; AIMS functioned efficiently, but they mention they wish to improve AIMS using non-linear models like neural networks.

An anomaly detection method to detect web attacks using Stacked Auto-Encoder [12] was published in the 2018 6th Iranian Joint Congress on Fuzzy and Intelligent Systems (CFIS). Here the researchers use the n-gram model to extract data from the HTTP requests and then use a Stacked Auto Encoder for dimensionality reduction. Then once this is done, they use the Isolation forest algorithm to detect anomalies. They have made it a point to test accuracies measured in the number of true positives and false negatives with different n-gram numbers (unigram/bigram) and different activation functions for neurons. This is a good step, as this will allow getting the best possible accuracy possible for the model. Subsequent to this analysis, these researchers have reached an accuracy of 88.32%, which is an adequate number for this use case.

The paper titled Streamlined anomaly detection in web requests using recurrent neural networks [13] publisher in the 2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS) takes a rather different approach to tackle the anomaly detection problem, but using Recurrent Neural Networks (RNN). Here, they use the LSTM (Long Short Term Memory) algorithm to analyze each character of the set of headers of a given HTTP request which in turn gives a set of ratings for the characters. Then they use a binary classifier to look for anomalies in this data. The CSIC 2010 HTTP dataset has been used to evaluate the algorithm and they have come up with a 62% detection rate, which is a decent accuracy figure. They point out the importance of low false-positive rates over the detection rate which does make sense as the high number of false positives can break a system's reliability and can be an inconvenience for administrators which makes the system impractical.

According to the related work in this domain, it can be observed that even though anomaly-based intrusion detection systems were introduced to address the limitations in the signature-based systems, they also have their limitations. A hybrid approach will be beneficial for intrusion detection if it could retain the advantages and address the limitations in anomaly-based IDS. Furthermore, most researches have been focused on detecting intrusions at the transport and network layers in the OSI model and not at the application layer where the APIs reside. Detecting intrusions at lower layers in the OSI models may not be able to detect application layer attacks directed towards APIs. Therefore, we will be focusing on detecting intrusions at the application layer.

3. Methodology

3.1. Design

When designing a system for this use case, we need to make sure that we can meet the non-functional requirements such as having a low latency and hence, not having an adverse effect on the performance, while still maintaining a good accuracy. In order to ensure this, the design of the proposed solution is based on three different layers. These layers are shown in the diagram given below.

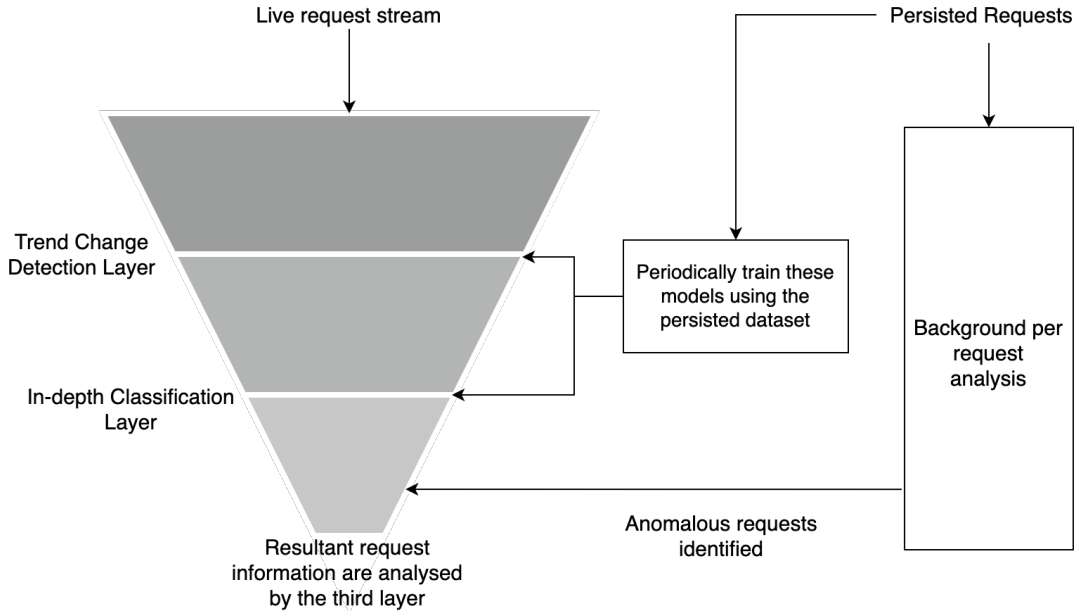

Figure 1: Layered architecture diagram

3.1.1. Trend Change Detection Layer

The responsibility of the trend change detection layer is to analyze the trend of API invocations and determine if there is any behavior that deviates from the normal. A deep learning model is proposed to achieve this task. The algorithm used for the deep learning model is a Stacked Autoencoder. The reason to go with a stacked autoencoder is because it is well recognized as a tool for outlier detection in recent papers. These stacked autoencoders work as a good medium to reduce the dimensionality of an input and represent it in a latent space which can then be used to determine outliers. In the process of training this deep learning model, we normally see use cases where we input raw information and expect the deep learning model to identify all patterns. This might work sometimes if we can input a very large dataset. But our focus should be to pre-process the data so that it is easier for the deep learning model to learn. In order to learn the trend of the API invocations, the dataset is arranged into buckets based on their timestamp. The ideal time that matched our dataset was ten seconds per bucket. These buckets are then processed to extract summary information that will be inserted to the deep learning model training process. The request count field contains the number of requests within the number of seconds of the bucket. The unique IP address count field contains the number of unique IP addresses within the request bucket. The unique access token count includes the unique number of access tokens used during the request bucket. The summary data also has broken down the HTTP method column to separate columns: GET request count within the bucket, the POST request count within the bucket, the DELETE request count within the bucket and the PUT request count within the bucket, in a one-hot encoded[14] manner. The summarized dataset also includes the unique invocation path count, the number of unique invocation paths that were invoked within the time duration, the longest invocation path that a resource was accessed within the time duration, the number of unique user agent values within the time duration, mean value of all response codes within the time duration and the total number of bytes communicated within the bucket.

Once the model is trained and a latent space is obtained, this latent space is stored as a dimensionally reduced representation of the normal behavior of the API invocations in the gateway. The new requests that the gateway receives are aggregated into buckets of ten seconds and then compared with the latent representation, to obtain a score. This score represents the deviation of the bucket in concern, from the normal behavior. We know the maximum deviation of normal requests from the latent space, which will help us to determine a threshold where we can set to consider incoming buckets as anomalies. In our effort, we set this threshold subsequent to adding a ten-point buffer to the normal maximum deviation score which was twenty-two, resulting in a threshold of thirty-two. This means that any request bucket that surpasses a deviation score of thirty-two, will be escalated to the in-depth classification layer for processing. These values such as the buffer value were obtained subsequent to many testing iterations involving attack datasets.

3.1.2. In-depth Classification Layer

In the in-depth classification layer, the buckets containing the anomalous requests and their neighbouring buckets are analyzed at a more granular level to narrow down the actual anoma-

Table 1
Summary - Anomaly detection models

Model	Description	Features
Geolocation	Anomaly detection based on the geolocation of the API request origin. The geolocation is obtained using request IP address	- Geolocation - Geo velocity
User agent-based access frequency	Anomaly detection based on the behavior of the user agent on the API request.	No.of user agent encounters in the past 24 hours, past hour and the past 5 minutes
HTTP method-based access frequency	Anomaly detection based on the behavior of the HTTP method on the API request	No.of HTTP method encounters within the past 24 hours, past hour and the past 5 minutes
Request path-based access frequency	Anomaly detection based on the behavior of the request path on the API request.	No.of request path encounters within the past 24 hours, past hour and the past 5 minutes.

lous requests. This request-by-request analysis is performed via a hybrid approach of both supervised and unsupervised learning techniques. The process consists of two phases. The unsupervised phase addresses the absence of labeled data in a production environment, and the supervised phase addresses the classification problem.

The anomaly detection phase learns the normal behavior of a user by using the HTTP header data from the API requests as input parameters. Instead of training a single model from all data, a collection of machine learning models is trained, where each model focuses on a specific area of the API request. This approach reduces the dimensionality of the training data and allows the models to learn the nuance details since they are more focused. There are four such anomaly detection models per user in the proposed approach.

K-Means and Local Outlier Factor algorithms were evaluated as the possible candidate algorithms for anomaly detection. K-Means was selected as the best performing algorithm due to its lower false-positive rate. Both algorithms were roughly equal in other aspects. It should also be worth noting that the anomaly detection models should be trained using the entire dataset rather than using the requests filtered out from the trend change detection layer to model the normal behavior as closely as possible.

The prediction results from the anomaly detection models are used as inputs for a random forest classifier in the classification phase. Since this model only learns how the predictions of each anomaly detection model affect the final request classification, it does not depend on the behavior of a particular API. In other words, the behavior of this classification model is not unique to a certain API. Therefore, the classification model can be pre-trained using the outputs of the anomaly detection models to “normal”, “token hijacking” and “abnormal token usage” API requests and embedded in a production-ready system.

The system will still train on the API specific data for anomaly detection and as a result, the whole system will still be able to adapt to the unique access patterns in the APIs for detecting attacks. This approach is able to address both the lack of labelled data in a production environ-

ment and the classification problem while being able to learn from the data instead of relying on the domain knowledge and the experience of the humans.

Features of the supervised classifier,

- Prediction of the geolocation model.
- Prediction of the user-agent based access frequency model.
- Prediction of the HTTP method based access frequency model.
- Prediction of the request path based access frequency model.
- Whether the user IP is previously encountered for the particular user.
- Whether the user agent is previously encountered for the particular user.

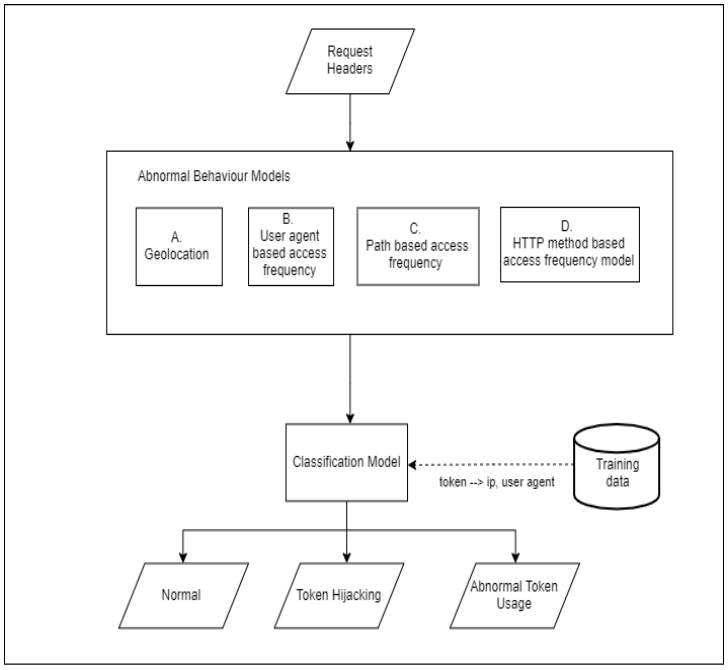

Figure 2: Decision making process with classification model

3.1.3. The Background Layer - Per Request Analysis

In the main three-layer mechanism we have designed, the main focus is given to the trends of the requests received by the gateway. And if the trend seems to deviate from the identified normal, the trend change detection layer identifies this and sends the set of suspicious requests to the in-depth classification layer to classify them. However, during our experiment, we identified that there could be some anomalies that do not make a significant impact on the overall trend of the gateway. The background layer is used to identify these anomalies that get through the trend change detection layer.

The background layer is a replica of the in-depth classification layer. It uses the same models trained from the same data using the same algorithms. However, while the in-depth classification layer only receives the filtered requests from the trend detection layer, the background layer receives and analyzes every request that is received by the gateway. This layer operates as a background process and yields more accurate results. But it can take some time since each request that the gateway receives needs to be analyzed individually. The anomalies detected in the background layer are sent to the human verification layer along with the anomalies detected in the trend change detection layer for further analysis.

3.1.4. Human Verification Layer

This layer is important until the system is fully tuned. Once the system gains a considerable amount of data, we can move towards setting a criteria to enforce policies based on the results of the previous layers.

3.2. The Dataset

3.2.1. Normal Dataset

The datasets for the experiment were collected using a traffic simulator which can be configured against the WSO2 API Manager. The simulator will first set up the data generation environment by creating a set of APIs and applications with necessary subscriptions according to the provided configurations. Also a configured number of users will be created in the system to consume these applications. These users will be randomly distributed among applications assuming a normal distribution in the user count vs number of applications and the user count vs request frequency for each application. The datasets will be collected while consuming these APIs using the created user accounts according to a normalized real world scenario.

3.2.2. Attack Dataset

The above-mentioned traffic simulator is capable of simulating DOS, DDOS, token hijacking (stolen token attack), abnormal token behavior, and extreme delete API attack scenarios by altering the appropriate request parameters. The simulated attack dataset has been used to test and evaluate the experiment.

Token Hijacking/ Stolen Token We define token hijacking as a malicious activity performed by a bad actor where they would steal the API access token from a legitimate API user and use it for malicious purposes. The device from which the API request originates will be different than the usual, and the access pattern of the attacker will also deviate from the usual user behavior.

Abnormal Token Usage We define abnormal token usage as an instance where even though the device from which the API request originates will match the usual pattern, the access pattern still deviates from the usual user behavior. A person accessing an API from a stolen device

could be a good example of this. But it should be noted that these type of anomalies are not always an attack, but could merely be an abnormal behaviour that might be worth investigating.

Extreme Delete Attack These attacks focus on deleting large amounts of API data in place, instead of extracting them. These can be detected by a sudden surge in the amount of DELETE and PUT requests coming into the system. Therefore it is paramount that the API call HTTP method information to be fed into our classification system as a feature.

3.2.3. Dataset Fields

Field	Description
Timestamp	Timestamp in which the request was sent. Includes all time values until the number of milliseconds
IP Address	The origin IP address from where the request originated
Access Token	The access token that the request was sent with. This token is used by the gateway to verify if the request was sent by a verified user who has enough permissions to access the resource.
HTTP Method	The HTTP method relevant to the request.
Invoke Path	The invoke path field gives the path which the request is trying to invoke. This value also includes the path parameters.
Cookie	The cookie value relevant to the request
Accept	The mime type preferred by the client
Content-Type	The mime type of the request payload
X-Forwarded-For	The x-forwarded-for header which normally includes the path in which a request follows to reach the gateway (Ex: proxies and load balancers)
User Agent	The user agent value for the request
Response Code	The final response code of the response pertaining to the request
Body Size	Total amount of bytes that was associated with the request payload
Label	The classification label for the request.

Table 2: Dataset description

4. Results

4.1. The Trend Change Detection Layer

The trend change detection layer was tested by orchestrating a set of attacks on the gateway. We selected a mixture DDoS attacks, stolen token attacks, data deletion attacks and data extraction attacks for this task. This experiment gave us an accuracy of 97%. The attacks that were misclassified here were low impact isolated attacks that we orchestrated to experiment with the thresholds. The following diagram represents a confusion matrix of such an attack dataset. Very few normal requests were used in this effort as we are testing the attack detection capabilities of this layer.

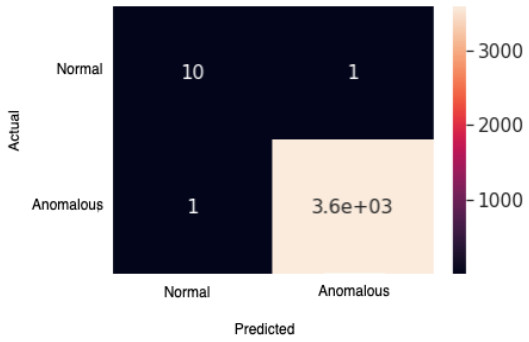

Figure 3: Classification results for the trend change detection layer

4.2. The In-depth Classification Layer

The performance of the in-depth classification layer in conjunction of the trend change detection layer has been evaluated next. The table 3 depicts the performance of each anomaly detection model against the filtered request buckets sent from the trend change detection layer.

Table 3
Anomaly detection results - In-depth classification layer

Model Category	TP		FN		FP		TN	
	N	%	N	%	N	%	N	%
Geolocation	1495	50	1495	50	0	0	138	100
User Agent	2180	72.91	810	27.09	43	31.16	95	68.84
HTTP Method	2665	89.13	335	10.87	91	73.19	37	26.81
Request Path	2499	83.58	491	16.42	102	73.91	36	26.09

In this context, the true positives (TP) are the actual anomalies that the models were able to identify correctly. The true negatives (TN) are the actual normal requests that the models were able to identify correctly as normal requests. The false negatives (FN) are the anomalies that the models have misclassified as normal requests, and the false positives(FP) are the normal requests that the models have misclassified as anomalies. The TP and FN percentages are calculated with respect to the total anomalies in the filtered dataset while the FP and TN percentages are calculated with respect to the total number of normal requests in the filtered dataset.

One interesting observation is that the geolocation model has only identified 50% of the total anomalies. A deeper analysis reveals that the geolocation model has not been able to identify any of the abnormal token usage requests. This is to be expected since by definition abnormal token usage anomalies have the same IP address as the normal users and therefore roughly the same geolocation.

Also, one can observe that the false positive rate is rather high, and the true negative rate is rather low for the HTTP method and request path models. This happens due to two reasons. The first reason is that the request buckets received by the in-depth classification layer contain

only a small number of normal requests (138 compared to the 2990 anomalies) since most of the normal requests are filtered in the trend detection layer. Therefore, misclassifying a single normal request could significantly raise the false positive rate. The next reason is because of the nature of the normal requests in the requests buckets. If a normal request passes through the filtration process of the trend change detection layer, it means the request is closer to abnormal behavior than the normal behavior. Therefore, we could expect more false positives in the filtered request buckets.

It is important to understand that the values represented in the table 3 are only an intermediate result. Since each model focuses on a particular area of the request these models have a narrower point of view than the final classification model. This means that while one model may consider a particular request an anomaly, another may consider the same request as a normal request based on its viewpoint. The classification model which predicts the final result considers the big picture and more accurate since it considered the outputs from every model for its output.

The confusion matrix and the classification reports for the request buckets after the requests are classified is as follows.

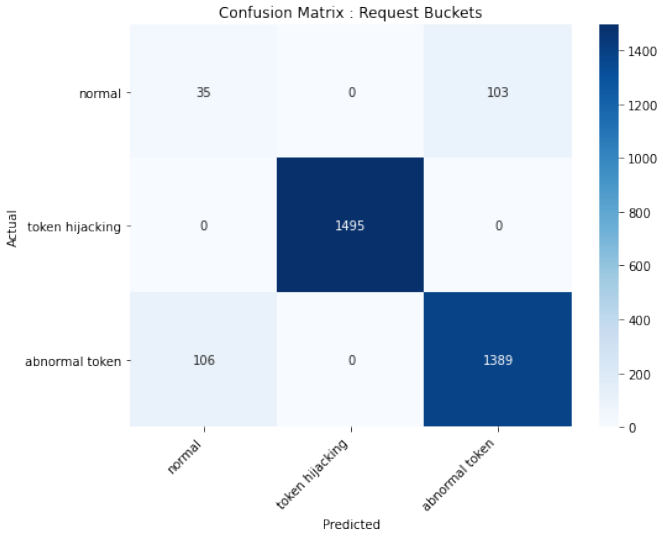

Figure 4: Classification results for the request buckets

Table 4
Classification report for the request buckets

	Precision	Recall	F1-score	Support
Normal	0.25	0.25	0.25	138
Stolen token	1.00	1.00	1.0	1495
Abnormal token	0.93	0.93	0.93	1495
Accuracy			0.93	3128
Macro average	0.73	0.73	0.73	3128
Weighted average	0.93	0.93	0.93	3128

4.3. The Background Layer

In this section we have evaluated the performance of the background layer. The table 5 depicts the performance of each anomaly detection model against the every request received by the gateway at the testing phase.

Table 5
Anomaly detection results - Background layer

Model Category	TP		FN		FP		TN	
	N	%	N	%	N	%	N	%
Geolocation	2500	50	2500	50	0	0	4613	100
User Agent	3921	78.42	1079	21.58	385	8.35	4228	91.65
HTTP Method	4156	83.12	844	16.88	683	14.81	3930	85.19
Request Path	3955	79.10	1045	20.90	717	15.54	3896	84.46

Here, the true positive(TP) and false negative (FN) percentages are calculated with respect to the total anomalies in the requests received by the gateway while the false positive (FP) and true negative (TN) percentages are calculated with respect to the total number of normal requests in the same dataset.

When comparing table 5 with table 3, it is evident that the false positive rate and the true negative rate of the intermediate result has been improved significantly due to the availability of the total dataset for the background layer. Furthermore, it should be noted that the true positive and false negative rates of the user agent model have been slightly improved while deteriorating slightly in the HTTP method and request path models.

The confusion matrix and the classification report for the total test dataset is shown in figure **??** and table 6.

Precision, recall and F1 score values for normal requests in the background layer denoted in table 6 are considerably higher than the values for in-depth classification layer denoted in table 4. As explained earlier, this is because of the request filtering done by the trend change detection layer. As the trend change detection layer filters out most of the normal requests, most of the normal requests that reach the in-depth classification layer borders on the abnormal behavior. As a result, there is a higher chance of those normal requests be classified as abnormal requests.

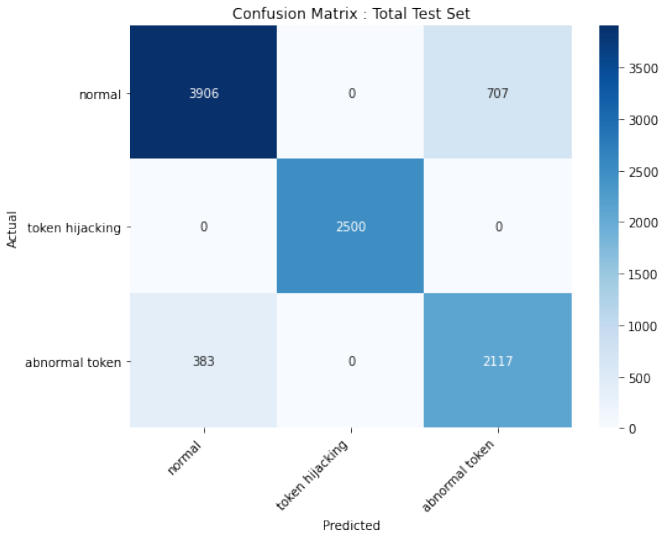

Figure 5: Classification results for the total dataset

Table 6
Classification report for the total dataset

	Precision	Recall	F1-score	Support
Normal	0.91	0.85	0.88	4613
Stolen token	1.00	1.00	1.00	2500
Abnormal token	0.75	0.85	0.80	2500
Accuracy			0.89	9613
Macro average	0.89	0.90	0.89	9613
Weighted average	0.89	0.89	0.89	9613

5. Conclusion and Future Work

The layered approach proposed in this paper was able to successfully overcome the problems and shortcomings of our previous approaches. According to the results of the experiments conducted throughout this effort, a good performance is shown in the department of identifying anomalies in the request stream with great efficiency. Further development is needed for the classification part of things so that this solution can move towards a completely automated defence system that requires negligible amounts of human intervention. The in-depth classification layer needs to be expanded further to identify more attacks so that a criteria can be written to automate the response to an attack and also help administrators to get a clear idea without too much analysis. Another future development eyed is to get rid of the third layer completely. Utilizing the classification done in the third layer with human input, to retrain the model will help with this task. Hope this research will be a major step taken towards a secure API ecosystem that is a testament to pure creativity without the worry of cyberattacks.

References

[1] Z. Whittaker, Millions of venmo transactions scraped in warning over privacy settings, 2019. URL: https://techcrunch.com/2019/06/16/millions-venmo-transactions-scraped/.

[2] M. Kumar, T-mobile hacked — 2 million customers' personal data stolen, 2018. URL: https://thehackernews.com/2018/08/t-mobile-hack-breach.html.

[3] M. Isaac, S. Frenkel, Facebook security breach exposes accounts of 50 million users, 2018. URL: https://www.nytimes.com/2018/09/28/technology/facebook-hack-data-breach.html.

[4] M. Zolotukhin, T. Hamalainen, T. Kokkonen, J. Siltanen, Analysis of http requests for anomaly detection of web attacks, 2014, pp. 406–411. doi:10.1109/DASC.2014.79.

[5] T. Sipola, A. Juvonen, J. Lehtonen, Anomaly detection from network logs using diffusion maps, volume 363, 2011, pp. 172–181. doi:10.1007/978-3-642-23957-1_20.

[6] W. Fan, An adaptive anomaly detection of web-based attacks, 2012, pp. 690–694. doi:10.1109/ICCSE.2012.6295168.

[7] J. Wang, M. Zhang, X. Yang, K. Long, Z. Chimin, Http-scan: Detecting http-flooding attack by modeling multi-features of web browsing behavior from noisy dataset, volume 12, 2013, pp. 677–682. doi:10.1109/APCC.2013.6766035.

[8] A. Juvonen, T. Sipola, T. Hämäläinen, Online anomaly detection using dimensionality reduction techniques for http log analysis, Computer Networks 91 (2015) 46–56. doi:10.1016/j.comnet.2015.07.019.

[9] K. Peng, Q. Huang, Clustering approach based on mini batch kmeans for intrusion detection system over big data, IEEE Access PP (2018) 1–1. doi:10.1109/ACCESS.2018.2810267.

[10] Z. Li, A. Das, J. Zhou, Usaid: Unifying signature-based and anomaly-based intrusion detection, 2005, pp. 702–712. doi:10.1007/11430919_81.

[11] L. Tong, J. Cao, Q. Qi, S. Qian, Aims: A predictive web api invocation behavior monitoring system, 2019, pp. 373–382. doi:10.1109/ICWS.2019.00067.

[12] A. Moradi Vartouni, S. Sedighian Kashi, M. Teshnehlab, An anomaly detection method to detect web attacks using stacked auto-encoder, 2018, pp. 131–134. doi:10.1109/CFIS.2018.8336654.

[13] A. Bochem, H. Zhang, D. Hogrefe, Poster abstract: Streamlined anomaly detection in web requests using recurrent neural networks, 2017, pp. 1016–1017. doi:10.1109/INFCOMW.2017.8116538.

[14] J. Brownlee, Why one-hot encode data in machine learning?, 2020. URL: https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning.html.