OWADIS: Rapid Discovery of OWASP10 Vulnerability based on Hybrid IDS

Leon Wirz, Asipan Ketphet, Nattapol Chiewnawintawat, Rinrada Tanthanathewin, Somchart Fugkeaw

School of Information, Computer, and Communication Technology,

Sirindhorn International Institute of Technology, Thammasat University

leon.wir@dome.tu.ac.th, asipanketphet@gmail.com, 6222781914@g.siit.tu.ac.th, rinrada.tanth.gal@gmail.com, somchart@siit.tu.ac.th

2023 15th International Conference on Knowledge and Smart Technology (KST) | 978-1-6654-7712-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/KST57286.2023.10086878

***Abstract*— Rapid advancements in internet applications introduce new vulnerabilities and threats that malicious actors are keen to exploit. These activities are becoming more versatile and challenging to address. In addition to implementing firewalls to control the inbound and outbound network traffic, an intrusion detection system (IDS) is commonly employed to monitor the network for malicious activities and policy violations. However, most IDSs are generally designed to monitor network traffic. They are incapable to detect the vulnerabilities embedded in the legitimate packets, especially the vulnerabilities targeting web applications. In this paper, we propose a cloud-based IDS with an emphasis on the detection of OWASP Top 10 Injection vulnerabilities, combined with additional common vulnerabilities such as brute-forcing and session hijacking. Furthermore, DDoS attacks, which are commonly seen, can also be detected with our proposed adaptable HTTP flooding detection engine. We also provide the evaluation to show that our proposed scheme provides fewer false positives than SNORT and gives efficient system throughput based on the leverage of Kafka and Spark streaming.**

Keywords*— *IDS, Injection, HTTP Flooding, Session Hijacking

I. INTRODUCTION

An IDS is a security system that primarily focuses on monitoring and analyzing system events and providing alerts to administrators. Protocol-based intrusion detection systems (IDSs) are specialized IDSs used only to monitor and analyze HTTP or HTTPS requests in a stream-like style; they are frequently installed on web servers.

Typically, there are three approaches used for the development of IDS: Signature/Heuristic-based, Anomaly-based, and Hybrid-based. A signature or heuristic-based is an approach where the detection is based on the pre-defined rules or known patterns of malicious activities, which is applied for our injection detection, brute forcing detection, and session hijacking detection, while the anomaly-based approach relies on the detection analysis through the statistical method or machine learning is applied to detect HTTP Flooding. A hybrid method encompasses both approaches to offer faster and more efficient detection in the evolving environment.

As a cloud service security strategy, a cloud-based intrusion detection system (IDS) is deployed in a cloud environment to monitor both network and system run on-premise and cloud. It leverages cloud infrastructure to facilitate better resource operation with more efficient workload management.

Deploying naïve IDS on the cloud requires system customization and integration into the particular cloud configuration. The system needs to be run on a cloud virtualization platform to accommodate a huge volume of network traffic and a variety of activity patterns. Therefore, a cloud-based IDS incorporated such requirements since the system design is preferred.

In this paper, we extend the signature-based IDS system proposed in [1] that applies Apache Kafka and Spark Streaming to efficiently handle the detection of HTTP flooding, SQL injection, and cross-site scripting attack. In this work, we proposed a hybrid-based IDS which uses a method of matching known patterns and rules of malicious actions in the signature-based detection together with the classification machine learning model. In addition, we extend the capabilities of our work to handle more five new attacks for our OWADIS:

i) XML external entity injection (XXE) as described in [2], which is a kind of attack that exploits an application that parses XML input. It allows an attacker to interfere with any back end or external system that the application itself can access as well as examine files on the application server file system. XXE occurs when XML parsers allow the loading of external entities. This attack can be used to steal data, perform DOS attacks, or map out the application and its environment.

ii) Shell/OS command injection, which aims to use a vulnerable application to execute arbitrary commands on the host operating system. When an application sends unsecured user-supplied data to a system shell, command injection attacks are conceivable. This attack enables an attacker to entirely compromise the application and all of its data by allowing arbitrary operating system (OS) commands to be executed on the server that is hosting the application.

iii) Local File Inclusion (LFI) and Remote File Inclusion (RFI), which are the kinds of attacks that occur when a web application allows the user to submit input into files or upload files to the server. To deceive the application into exposing or running files on the server, a local file inclusion attack is performed. It allows attackers to execute arbitrary commands or, if the server is configured improperly and is operating with high privileges, to access sensitive data. The remote file inclusion is similar to the local file inclusion but instead of accessing a file on the local machine, the attacker will remotely execute code hosted on their own machine. Detailed descriptions of this type of attack are further elaborated in by Begum et al. in [3].

iv) Open Redirect, which is an attack that emerges when the application allows attackers to pass information to the app that results in the users being sent to another location.

v) Session Hijacking, which is a kind of attack where the attackers take control of a target's session. To get direct access to the machine, the attacker can deploy malware to infect the target's computer. Additionally, this type of attack can gain the malicious actor unauthorized access to information that should not be accessible to the attacker.

In essence, Spark [4] and Kafka [5] are used as the main platforms to implement our system. Since they rely on functional programming, this allows for functions such as MapReduce to improve performance, which is important for the monitoring of web application requests. Furthermore, CatBoost [6] was used in the development of the machine learning model. It supports the concept of continuous learning which is important when it comes to the concept of everchanging traffic in web applications.

To the best of our knowledge, there are no IDS works dedicated to supporting the automatic discovery of web application vulnerabilities run over Apache Kafka and Spark Streaming.

II. Related Work

The construction of IDS always comes with a false positive problem, which is a test result that incorrectly triggers an alert system when there is an absence of the condition. This weakness can occur due to the quality of the input, [7] proposed two concepts of reducing a false positive in a Signature-Based detection system, which is used for efficiently detecting known attacks. Although simple, and effective, there is a limitation to adapting to an immediate event of an attack that the system is currently unknown. Therefore, the development of an anomaly-based detection system, a combination of extensive machine learning model IDS is in demand.

There are several research works that introduced intrusion detection systems in response to common network attacks such as DDoS. With the elevation of cyber-attack in various forms, the limitation of the intrusion detection system (IDS) has been acknowledged and demanded higher reliability. IDS' vulnerabilities are that it is unable to detect cyber-attacks in the form of Distributed Denial of Service (DDoS) attacks and session hijacking. As reported in [8], several studies investigated the techniques to detect an appropriately classified DDoS attack, but the limitation of input and the complexity of the attack makes the IDS vulnerable. A predefined function may not be an efficient technique to logically address the weakness.

The authors utilize the capability of a technology trend by constructing a machine learning model. Although training a machine learning model can help improve protection against the unknown attack, data training and classification is a challenge in constructing a model that is performed against a highly adaptable attack. [9] proposed a categorical machine model, an integrated machine called an adaptive model, by combining learning history and learning content information; this concept has been applied to our project by using historical statistics, in order to analyze the frequency and adapt the model to be able to react against an attack previously performed.

The major problem of constructing a machine model is the time required to learn and classify the data [10]. The development of a machine learning model can solve the computation power and time, which are necessary when developing an efficient IDS. Therefore, in [11] the authors improve the model by using gradient boosting and sufficiently handling categorical features in terms of available datasets. In addition to the improvement of the training datasets, CPU and GPU implementations were taken into consideration for the programmable code to enhance the scoring algorithm.

In [12], the authors proposed an injection detection IDS, which has the ability to detect SQL injection attacks and Cross-Site Scripting. The system is built on top of SNORT, which allows for a lightweight IDS. However, the system is limited to only the two mentioned attacks, since the creation of a rule for SNORT that yields a low FP rate is difficult.

Session Hijacking, specifically HTTP Session Hijacking has been researched and studied by Cheng, Gao, and Guo [13]. The paper shows different types of attack scenarios in that a malicious actor might take advantage of a system by stealing someone's secret session identifier. Nevertheless, a methodology to detect such activity has not been proposed.

Priya et al [14] proposed a machine-learning model to tackle to problem of DDoS attacks. Data collected from Wireshark is used to train and test three classification algorithms, KNN, RF, and NB. Taking this idea, we found several issues regarding the dynamics of a web server. Traffic doesn't stay constant throughout the web server's lifetime; therefore, a dynamically changing model is required to accommodate this problem.

Nevertheless, all the above works did not simultaneously achieve efficient and computationally light processing of traffic, while having a trustworthy result (Low FP rate, High TP rate), and supporting most modern cyber-attacks that are found in OWASP10.

III. Technical Background

A. Injection Attacks

Injection attacks are attacks that provide malicious input to the web server. In our proposed system, six separate injection attacks that are mentioned in this paper are: SQL injection (SQLi), Cross-Site Scripting (XSS), XML External Entity injection (XXE), Shell/OS command, Local/Remote File Inclusion (RFI/LFI), and Open Redirect. Injection attacks utilize fields in HTTP paths or body parameters to be executed on an environment that is authorized.

B. Continuous Learning

In the field of machine learning, the standard is to fit a model that is used to predict some output from an input, where the model is trained. Traditionally, a model is final and complete after a certain threshold of correctness is achieved, leading to the usage of this model indefinitely. Continuous Learning allows the retraining of an existing model. In older Machine Learning frameworks, if a model needed to be adjusted the training process had to be redone from the beginning, whereas in continuous learning existing weights are preserved, while a new dataset for training is applied to the model. CatBoost is the framework that we applied in this system, which achieves this.

C. *Kafka*

Kafka is a distributed streaming platform that provides a very high scalable, high fault tolerance and also allows a high level of parallelism between data producers and data consumers. This means that Kafka is a main component of the Big Data Platforms. Kafka brokers are Kafka nodes on a cluster and topics which are categories of streams and streaming records that can be partitioned and replicated and also support many writers and many readers. Each broker has its own topics and partition. The producer can push updated data into Kafka corresponding to its topic. The consumer can pull the updated data that the producer pushed earlier from Kafka.

D. *Spark Streaming*

Spark Streaming is a framework used for large-scale stream processing and functional programming. It achieves second-scale latencies, and the delay should be minimized in seconds or milliseconds creating a near real-time. Spark streaming requires a cluster to run spark jobs the same way Spark is running jobs. The major difference between Spark streaming and Spark is that Spark streaming receives data in a stream formula, which in this paper's case is from a Kafka Topic. The Figure 1. Our System Diagram

IV. Our Proposed System

Our proposed scheme consists of two main environments, the System's Back-end server, and the Cloud Environment. The only purpose for the external Back-end server is to forward HTTP requests as a log format to the IDS on the cloud environment. The cloud consists of four major subsystems, Kafka, Spark Streaming, HTTP Flooding Retrain Engine, and Data Export. The Kafka subsystem is responsible for collecting Data from the producer, being the external Back-end Server, and distributing the collected data to consumers which are comprised of Spark Streaming Jobs. The Spark Streaming module is the main detection engine of the system as all detection algorithms are executed in this environment. HTTP Flooding is detected using a Machine learning classification model. To keep up with the constant and unpredictable changes in request rates to each webserver, the model has to be retrained, where we have a specialized and novel framework the HTTP Flooding Retrain Engine. The Data Export module handles the interaction between the detections in the Spark Streaming Module and the administrator. In this module logs are exported to be kept in a Cloud Storage, Alerts are sent out via SMS and Email, and a Grafana Dashboard is linked to the system via this channel. Our proposed scheme's system model is shown in Figure 1. Additionally, each detecting algorithm can be categorized as in Figure 2, where it can be explained as all the injection detection algorithms, brute force, and session hijacking are considered signature-based detection. HTTP flooding on the other hand is Anomaly-Based, resulting in the system as a whole being a hybrid-based approach.

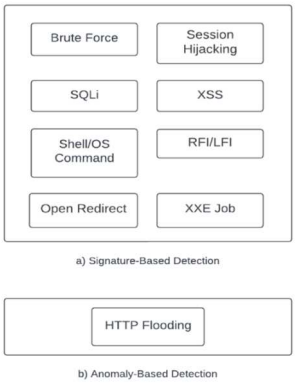

Fig. 2. Detection Type Categorization.

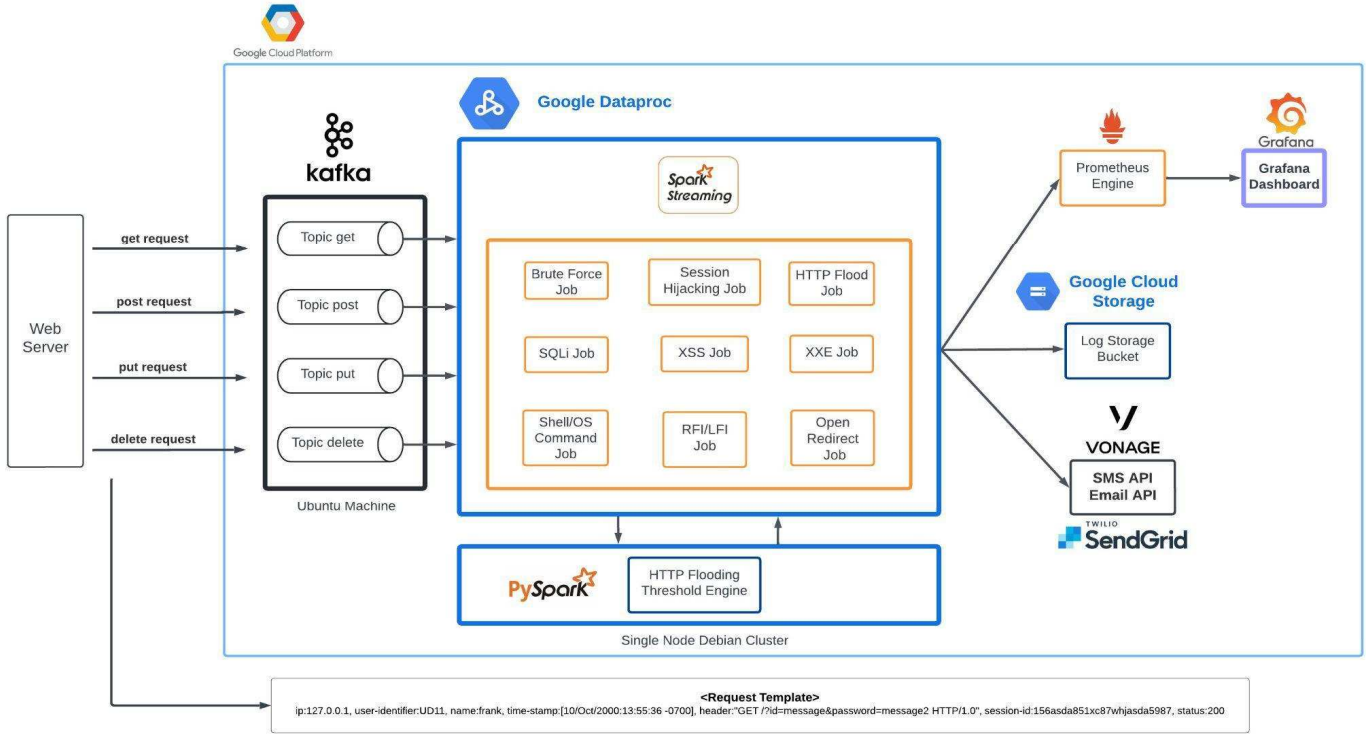

Fig. 1. Our System Diagram.

Our system assumes to receive a fixed format by Kafka which can be seen as follows:

ip:127.0.0.1, user-identifier:UD11, name:frank, time-stamp:[10/Oct/2000:13:55:36 -0700], header:"GET /?id=message&password=message2 HTTP/1.0", session-id:156asda851xc87whjasda5987, status:200

A. Detection Algorithms

Our proposed system consists of four major detection algorithms that can detect various types of attacks, being Injection Detection, Session Hijacking, Brute Forcing, and HTTP Flooding.

Injection Detection Algorithm

As in [1], which consists of two types of attacks that can be detected using signature checking, we additionally included the detection of XML External Entity injection (XXE), Shell/OS command Injection, Local/Remote File Inclusion (RFI/LFI) and Open Redirect. Furthermore, we improved the signature collections of SQL injection (SQLi) and Cross-Site Scripting (XSS) to decrease the rate of False Positives. The algorithm takes in a data stream and a list as inputs and then alerts and logs any injection attacks that have been detected as follows:

```
detectInjection(dataStream, signatureList):
        RDD rdd = dataStream
                .flatMap(_.value().split("\n"))
                .map(x => x.split(", ")(0), x.split(", ")(4) )
        Collection Result =  rdd
                .reduceByKey((x, y) => x + " , " + y)
                .filter(x => signatureList.exists(
                        y => x._2.contains(y)))
        for each Result as r:
                alertAndLog(r)
```

This algorithm is called in each timeframe t, where the inputs dataStream and signatureList are used, and the output consists of the logging of each detected injection attack and alerting the system administrator. The signatureList argument is a List collection of Strings that represent each injection signature, where each Injection type has its own collection. The signature comparison is considered to be the most time-consuming aspect of all detection algorithms as it scales with the size of the collection of signatures (Number of Signatures).

Session Hijacking Detection Algorithm

This algorithm detects any usage of duplicated sessions by using Functional Programming to filter out any unique session IDs that have duplicate usages by different users, who are differentiated by IP addresses. The algorithm takes a stream as input and then alerts and logs any attacks that have been detected as session hijacking as follows:

```
detectSessionHijacking(dataStream):
        RDD rdd = dataStream
                .flatMap(_.value().split("\n"))
                .map(x => x.split(", ")(5), x.split(", ")(0) )
        Collection Result =
                rdd
                .distinct()
                .countByKey()
                .filter(x => { x._2 > = 2});
        for each Result as r:
                        alertAndLog(r)
```

This algorithm is called in each timeframe t, where the input is a dataStream, and the output consists of the logging of each detected Session Hijacking attack and making an alert.

Brute Forcing Detection Algorithm

This algorithm detects abnormal amounts of requests from the same IP addresses. This is a key characteristic of brute force attacks that our system is able to detect. The algorithm takes a data stream as input and then alerts and logs any attacks that have been detected as brute force attacks as follows:

```
detectBruteForcing(dataStream):
        RDD rdd = dataStream
                .flatMap(_.value().split("\n"))
                .map(record => (record.split(", ")(0), 1))
        Collection Result =
                rdd
                .reduceByKey((x, y) => x + y)
                .filter(x => x._2>bruteForceThreshold)
        for each Result as r:
                        alertAndLog(r)
```

This algorithm is called in each timeframe t, where the input is a dataStream is used, and the output consists of the logging of each detected Brute Force attack and alerting the system administrator. Furthermore, an optional parameter can be passed, being the bruteForceThreshold, which will be an integer that determines how many requests per second are considered brute force.

HTTP Flooding Algorithm

The last detection algorithm detects any anomalies or abnormal total requests in the system. This is done by feeding in the number of total requests to a ML model that has been trained to classify between HTTP Flooding attacks and normal traffic. The algorithm takes in a data stream as input and then alerts and logs any attacks that have been detected as HTTP Flooding as follows:

```
detectHttpFlooding(dataStream):
        Model model = LoadModel("Path-To-Model")
        RDD rdd = dataStream
                .flatMap(_.value().split("\n"))
                .map(record => (record.split(", ")(4), 1))
        Collection Result =
                rdd
                .reduceByKey((x, y) => x + y)
                .filter(x => model.predict(x))
        for each Result as r:
                alert(r)
                log(r)
```

B. *HTTP Flooding Retraining Engine*

The HTTP Flooding Retraining Engine in our proposed system is a self-contained environment for updating and retraining the HTTP Flooding detection model, as can be seen in Figure 3.

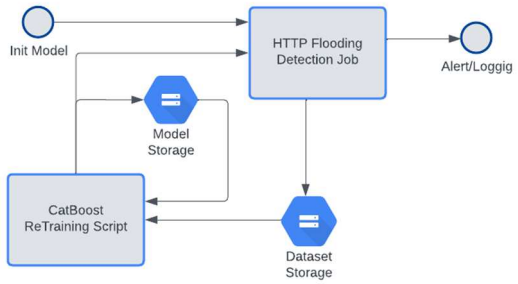

Fig. 3. HTTP Flooding Retraining Engine.

The engine consists of five components that each have their own individual functionality. The first component is an initial trained model. After a specified time, the CatBoost ReTraining Script takes the most recently trained model to retrain the Model to output a new Model, which finally is forwarded to the HTTP Flooding detection Job. Additionally, the trained model is stored on Model Cloud Storage.

The CatBoost ReTraining script retrains an existing model by refitting the model using two input attributes, the number of requests and the timestamp while having an output of either being True or False. To obtain each timeframes training dataset, the HTTP Flooding detection job aggregates data by adding an output column by applying an algorithm as follows:

```
aggregateHttpFloodingData(dataStream):
        Model model = LoadModel("Path-To-Model")
        RDD rdd = dataStream
                .flatMap(_.value().split("\n"))
                .map(record => (record.split(", ")(4), 1))
        Collection Result =
                rdd
                .reduceByKey((x, y) => x + y)
        for each Result as r:
                if (r <= Result.mean()):
                        r.value().set(r.value() + false)
                else:
                        r.value().set(r.value() + true)
```

The algorithm compares the number of requests with the corresponding mean value and groups them into being either an attack (True) or not (False).

V. EVALUATION

This section presents the comparative analysis on account of details of experiments simulated to measure the precision of injection detection and performance of our system and SNORT as implemented in [12]. Each Injection detection algorithm is run using 100 000 simultaneous requests as a batch containing 2500 payloads that are randomly selected from [15]. The experimentation environment is GCP [16], which includes a cluster for Apache Kafka, using a single-core CPU and 3.75GB of memory. The Dataproc cluster for running Spark jobs, consisting of a quad-core CPU and 15GB of memory is used. The ML model is both initialized and retrained on Google Collab which during the experimentation consists of a dual-core CPU and 13GB of memory. The Experimentation results, that capture the Precision can be seen in table 1.

TABLE I Experimentation Results

Scheme	False Positive	False Negative	Precision
[12]	23	0	0.9908
Ours	3	0	0.9988

Our system's Precision Rate is high enough to detect injection attacks such as those used in SNORT in [12]. Even though our system's Precision only leads by 0.008, our system can cover a wider range of injection attacks, as the work in [12] can only detect SQL injection attacks and Cross-Site Scripting. Throughput-wise, OWADIS scales well with high numbers of requests which can be seen in Figure 4.

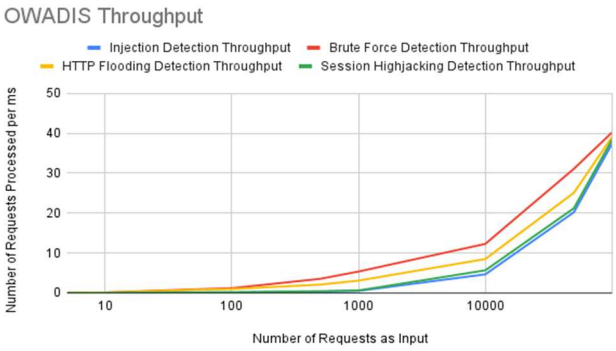

Fig. 4. Throughput of OWADIS.

Figure 4 shows the throughput of our system with respect to the number of requests that are run through per millisecond. This shows that our system is capable when it comes to detecting massive amounts of requests, since our tests of a maximum of 100 000 Requests sent at once yield a throughput of around 40 Requests/Millisecond, which corresponds to approximately 2500ms of total processing time used for the total process of detecting attacks. As the timeframe of each RDD is 10s, this system does not result in any delays. To measure the HTTP Flooding engine, we compute the confusion matrix as in figure 5.

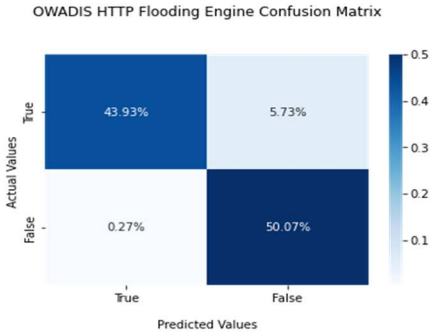

Fig. 5. HTTP Flooding Engine Confusion Matrix.

The initial HTTP Flooding detection Model is trained using 100 000 rows of data. The Validation has the output as can be seen in Figure 5, where True Positive and True Negative rates yield promising results of 43.93% and 50.07% respectively. Hence, we can say that the initial model used in the engine is sufficiently trained. Lastly, we compute the retraining time of the HTTP Flooding Detection Engine with respect to the size of the new training set.

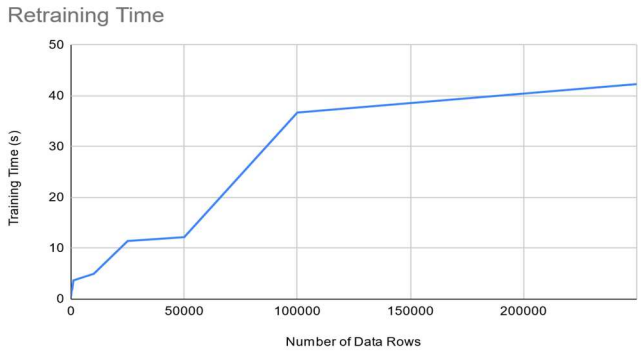

Fig. 6. Retraining Time of HTTP Flooding Detection Engine.

Since our proposed system for detecting HTTP Flooding is retrained in a user-specified timeframe, the time CatBoost uses to retrain the model can be seen in Figure 6. From the figure, it can be derived that the time it takes for retraining does not exceed 1 minute for the maximum testing size of 250000 rows of training data. As a training process in the field of machine learning, this can be considered as efficient, as the frequency of retraining is recommended to be daily to weekly depending on the rate of change in requests.

VI. Conclusion

In this paper, we have proposed and implemented a Hybrid IDS called OWADIS, which can detect a variety of injection attacks, such as SQLi, XSS, XXE, Shell injection, LFI/RFI, and Open Redirect using Signature-based detection. Additionally, a novel way of detecting session hijacking attacks using functional programming is also introduced, while Brute forcing attacks can also be detected. Furthermore, an Anomaly-based detection engine using ML is applied to detect HTTP Flooding attacks, which can also adapt to changing traffic conditions using retraining applied by CatBoost. To evaluate the system, we implement each part on GCP and run various tests using changing payloads and sizes, which shows promising results across the board. Alert systems are placed in each detection algorithm where SMS alerts and Email Reports are available, and additional support APIs can be implemented further on.

References

[1] L. Wirz, R. Tanthanathewin, A. Ketphet and S. Fugkeaw, "Design and Development of A Cloud-Based IDS using Apache Kafka and Spark Streaming," 2022 19th International Joint Conference on Computer Science and Software Engineering (JCSSE), 2022, pp. 1-6, doi: 10.1109/JCSSE54890.2022.9836264.

[2] S. Jan, C. D. Nguyen and L. Briand, "Known XML Vulnerabilities Are Still a Threat to Popular Parsers and Open Source Systems," 2015 IEEE International Conference on Software Quality, Reliability and Security, 2015, pp. 233-241, doi: 10.1109/QRS.2015.42.

[3] A. Begum, M. M. Hassan, T. Bhuiyan and M. H. Sharif, "RFI and SQLi based local file inclusion vulnerabilities in web applications of Bangladesh," 2016 International Workshop on Computational Intelligence (IWCI), 2016, pp. 21-25, doi: 10.1109/IWCI.2016.7860332.

[4] *Spark overview*, [online] Available: https://spark.apache.org/docs/latest/

[5] Kafka *Documentation*, [online] Available: https://kafka.apache.org/documentation/

[6] CatBoost Documentation, [online] Available at: https://catboost.ai/en/docs/

[7] P. Pitre, A. Gandhi, V. Konde, R. Adhao and V. Pachghare, "An Intrusion Detection System for Zero-Day Attacks to Reduce False Positive Rates," 2022 International Conference for Advancement in Technology (ICONAT), 2022, pp. 1-6, doi: 10.1109/ICONAT53423.2022.9726105.

[8] A. Agarwal, R. Singh and M. Khari, "Detection of DDOS Attack Using IDS Mechanism: A Review," 2022 1st International Conference on Informatics (ICI), 2022, pp. 36-46, doi: 10.1109/ICI53355.2022.9786899.

[9] Dol Aher, Sunita & Lobo, Louis. (2013). Combination of machine learning algorithms for recommendation of courses in E-Learning System based on historical data. Knowledge-Based Systems. 51. 1-14. 10.1016/j.knosys.2013.04.015.

[10] S. M. Othman, F. M. Ba-Alwi, N. T. Alsohybe, and A. Y. Al-Hashida, “Intrusion detection model using machine learning algorithm on Big Data environment,” Journal of Big Data, vol. 5, no. 1. Springer Science and Business Media LLC, Sep. 24, 2018. doi: 10.1186/s40537-018-0145-4.

[11] A. V. Dorogush, V. Ershov, and A. Gulin, “CatBoost: gradient boosting with categorical features support.” arXiv, 2018. doi: 10.48550/ARXIV.1810.11363.

[12] H. Alnabulsi and R. Islam, "Protecting Code Injection Attacks in Intelligent Transportation System," 2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), 2019, pp. 799-806, doi: 10.1109/TrustCom/BigDataSE.2019.00116.

[13] K. Cheng, M. Gao and R. Guo, "Analysis and Research on HTTPS Hijacking Attacks," 2010 Second International Conference on Networks Security, Wireless Communications and Trusted Computing, 2010, pp. 223-226, doi: 10.1109/NSWCTC.2010.187

[14] S. S. Priya, M. Sivaram, D. Yuvaraj and A. Jayanthiladevi, "Machine Learning based DDOS Detection," 2020 International Conference on Emerging Smart Computing and Informatics (ESCI), 2020, pp. 234-237, doi: 10.1109/ESCI48226.2020.9167642.

[15] I. Tasdelen, *Payloadbox* 2021, [online] Available: https://github.com/payloadbox.

[16] *Google*, [online] Available: https://cloud.google.com/.