Detecting DoS Attacks in Microservice Applications: Approach and Case Study

Jessica Castro
University of Coimbra, CISUC, DEI
Coimbra, Portugal
jesscmaciel@dei.uc.pt

Nuno Laranjeiro
University of Coimbra, CISUC, DEI
Coimbra, Portugal
cnl@dei.uc.pt

Marco Vieira
University of Coimbra, CISUC, DEI
Coimbra, Portugal
mvieira@dei.uc.pt

ABSTRACT

A microservices-based architecture decreases the complexity of developing new systems, making them highly scalable and manageable. However, its distributed nature, the high granularity of services, and the large attack surface increase the need to secure those systems at runtime. This paper investigates the challenges of detecting low- and high-volume DoS attacks against microservices using application-level metrics. We conducted an exploratory study to evaluate how different services influence attack detection, the use of *Machine Learning* (ML) techniques to detect DoS attacks, and the application-level metrics that can be used to detect DoS attacks. The results show that, analysing the services in parallel improves the detection rate, ML models are promising in detecting DoS attacks, and the numbers of sockets and threads used by containers are valuable metrics to indicate high-volume DoS attacks.

CCS CONCEPTS

• **Security and privacy → Software and application security**.

KEYWORDS

microservices, security, attack detection, machine learning, denial of service, container

ACM Reference Format:
Jessica Castro, Nuno Laranjeiro, and Marco Vieira. 2022. Detecting DoS Attacks in Microservice Applications: Approach and Case Study. In *11th Latin-American Symposium on Dependable Computing (LADC 2022), November 21–24, 2022, Fortaleza/CE, Brazil.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3569902.3569916

1 INTRODUCTION

Nowadays, many applications are built based on microservices-based architectures to decrease the complexity of developing modern distributed computer systems. The microservices paradigm empowers developers to construct highly scalable, flexible, and manageable systems [22]. A single application can be composed of several small independent services running their own tasks and communicating through a lightweight mechanism, such as REST APIs [1]. The services can be implemented using different technologies, and they can be deployed, updated and deleted individually [18].

Microservice architectures raise several security challenges. Their distributed nature and fine granularity enlarge the attack surface because instead of a single point of the attack (i.e., the application), it has multiple points (i.e., all the services). So, together with lightweight communication mechanisms with network-exposed APIs for communication with other services, it turns microservices systems into a potential target for cyber-attack, such as *Denial of Service* (DoS) [18].

DoS attacks happen when a malicious user sends requests to an application or device to turn it unavailable and incapable of delivering its services to legitimate users [16]. Hence, if the attack is successful, it may lead the target to resource exhaustion [11]. Three factors increase the challenge of detecting DoS attacks: the attackers use legitimate requests, making it hard to differentiate from non-attack ones; the attacker only needs a few resources to initiate an attack; the attacks can overwhelm the application quickly [13].

There are two main types of DoS attacks at the application layer: High-volume (or Request Flooding) Attacks and Low-volume (or Slow Request/Response) Attacks. A *High-volume* uses regular POST and GET requests to look like legitimate users. However, the number of requests is enormous because the attacker tries to overwhelm the system by consuming the maximum available resources. A *Low-volume* is when the attacker sends several HTTP requests slowly. The attacker keeps sending new requests leaving the previous ones open and keeping the resources busy. For example, the server may not have any more sockets available to start new connections, making it impossible for the application to accept requests from legitimate clients [10].

Attack detection at the application layer is necessary to secure applications at runtime. So, having a security monitor is essential since it is the last defence of a system [20]. Therefore, monitoring the system's properties and collecting security-related data at runtime is indispensable to assess the security status. However, a single microservice-based application can be composed of tens to hundreds of microservices which generate a large amount of data, making it impossible to analyse all the collected metrics [9]. Besides that, each service is impacted differently depending on the focus of the request/attack, which increases the challenge of monitoring the system and detecting vulnerabilities, threats, and performance bottlenecks [18]. Thus, how to monitor microservice applications and deal with the diversity of services is still an open issue.

LADC '22, November 21–24, 2022, Fortaleza-CE, Brazil

ACM ISBN 978-1-4503-9737-7/22/11…$15.00
https://doi.org/10.1145/3569902.3569916

This paper investigates and discusses the challenges behind detecting low- and high-volume application-layer DoS attacks targeting microservice applications. For this, we propose an approach to generating the data by monitoring the microservice application, collecting application-layer metrics provided by a container orchestrator, and analysing the data, including the composition of datasets and the detector responsible for identifying when an attack is happening. The research questions guiding this work are:

RQ1. Which application-level metrics should be monitored to detect DoS attacks?

RQ2. How should monitored data be arranged in the dataset to increase the success of detecting DoS attacks in a microservice application?

RQ3. Can we use machine learning to detect low- and high-volume DoS attacks effectively in microservice applications?

To answer the research questions, (i) we built the testbed; (ii) we defined four attack profiles that describe the duration and moment of the attacks; (iii) we collected and investigated which metrics can impact the success of the attack detection; (iv) we constructed three datasets organising the collected data in sequential, parallel, and individually; and (v) we assessed the use of machine learning classification algorithms to detect low- and high-volume application-layer DoS attacks and evaluate how the individuality of each service may increase the challenge of detecting the attacks.

The main contributions of this paper are the following:

- The definition of an approach to analyse the detection of attacks against microservice applications allowing the proposal of novel security solutions.
- Analysis of the differences regarding monitoring services sequential, parallel and individually.
- Analysis of a set of metrics to monitor and identify DoS attacks against microservice applications.
- The exploratory use of ML classification algorithms to identify DoS attacks against microservice applications.

The paper is organised as follows. Section 2 presents the related work. Section 3 describes the proposed approach, Section 4 presents the experiments setup and Section 5 discusses the results. Finally, Section 6 concludes this paper.

2 RELATED WORK

Works have been conducted to monitor a microservice application to collect metrics indicating an anomaly in the system. Du et al. [7] try to answer what metrics should be monitored and how to evaluate whether the application behaviours are anomalous by collecting CPU and memory usage metrics from the containers and analysing them using ML algorithms. Samir and Pahl [15] also monitored the CPU and memory usage to detect and localise container abnormalities using Spearman's rank correlation coefficient and hierarchical hidden Markov model (HHMM). In [9], the authors try to predict the performance degradation of the system, also collecting several performance metrics (e.g., CPU, memory usage, the throughput of I/O-devices, etc.) and use different ML techniques to assess the proposed approach.

Detecting DoS attacks on microservice or containerised applications is a recent topic and is still in progress. Baarzi et al. proposed an unsupervised, non-intrusive, and application-agnostic approach to protecting microservice systems from DoS attacks at the application level [3]. The authors analysed historical data of collected resource utilisation (i.e., CPU and memory) of different applications under various load levels only with legitimate users and calculated the average and variance of both CPU and Memory utilisation at different load levels to identify abnormal use of the resources. To the best of our knowledge, most of the works focused on microservices deal with attacks against other layers (e.g., Network layer [14] [19]) or with detection in a previous security layer (e.g., in intrusion detection methods [21]). However, several works address the detection of DoS using ML for other system architectures showing that it is possible to use these techniques [4]. For instance, Filho et al. [11] proposed Smart Detection, an approach to detect low- and high-volume DoS attacks using Random Forest Tree algorithm to analyse random traffic samples.

In summary, the works focusing on monitoring and detection in microservice applications mainly deal with anomaly detection and usually monitor CPU and memory metrics. Furthermore, the existing works on DoS detection mainly focus on protecting other layers of microservices applications, lacking solutions to protect the systems at the application layer. Therefore, this work investigates other application-level metrics, beyond CPU and memory, that can be used to indicate DoS attacks and how the DoS attacks and microservices characteristics can influence the detection.

3 APPROACH

This section presents our approach for exploring and evaluating the detection of DoS attacks against container-based microservice applications. Figure 1 shows the approach composed of two phases: Data Generation and Data Analysis. The first phase is suported by (i) the Workload Generator and Attack Injector responsible for generating normal workload and performing the attacks, respectively; (ii) the microservice application deployed in a container orchestrator; and (iii) a monitor module that collects the metrics. The second phase has (iv) the generated datasets; (v) the attack detector; and (vi) the detection results.

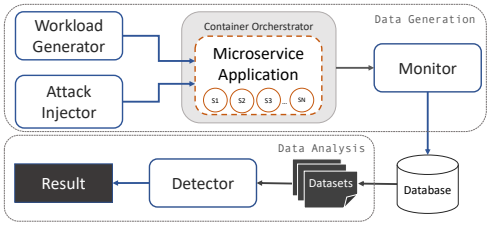

Figure 1: DoS Detection Approach

3.1 Data Generation

The availability of DoS application-layer datasets is one of the significant challenges, and it is increased since microservices are an emergent technology [12]. Because of this, it is necessary to generate the data to create the datasets to be used to develop detection methods.

3.1.1 Workload Generation. The workload is used to emulate requests submitted by real users to the system. It is necessary to

define the system's behaviour in normal circumstances so it is possible to evaluate the impact of the system under attack. Therefore, when defining the workload, it is essential to consider the system's utilisation, the available resources, and the experimental context. With this, the system can be exercised in a representative manner, and it will be possible to identify any change in the system's normal behavior.

We must define the following points to perform the experiments: **Paths**: indicate the application's services/pages the users will send requests to; **Duration**: the running duration must be sufficient to warm up the system and have time to perform pre-attack, attacks, and pos-attack phases; **Requests**: it is the definition of the number of simultaneous users making requests, the number of requests per time unit, and whether there will be a ramp-up or a specified number of users; **Load testing tool**: a specialised application that will execute the script defined in the previous points; **Attack Tools/Attack emulation**: tools to perform low- and high-volume DoS attacks that should be representative and realistic as an actual attack.

3.1.2 Monitor and Data Collector. With a defined workload generation and attack emulation, it is necessary to determine how to monitor the microservice application. More specifically, (a) *which monitoring tool to use*, and (b) *which system properties to monitor*. We must carefully define this step because the success in detecting the attacks lies in monitoring and collecting the accurate system's properties. So, if it is not accurate, it can trigger faulty decisions.

A microservice application comprises several services that can be distributed all over the internet. To manage the services, the application is usually deployed in a container orchestrator, such as Kubernetes. Thus, it is necessary to research tools that can monitor containerised microservices at runtime. After that, it is required to verify all the metrics collected by the tools to identify which ones are affected by the experiments. These metrics will be used to construct the dataset.

Data collectors are responsible for gathering the data shared by the monitor and storing them in a database that the analysis component can access. This tool must perform its task with a high frequency. The data is stored in a time-series database because this information is highly dependent on time.

3.2 Data Analysis

Once the data is collected and stored, it is necessary to analyse them to detect when an attack is happening. First, we need to construct the dataset. Since a microservice application comprises multiple independent services that implement different functions, it is important to investigate how to deal with their heterogeneity and unique characteristics. Therefore, to analyse it, we organise the data in three ways: **i) Sequential**, where each sample is the information of only one service following the order by which it is made available by the monitor; **2) Parallel**, where each sample is composed of the metrics of all the services; **iii) Individual**, where each service has its dataset that is used to identify the attack individually.

Regarding the DoS attack detector, this work focuses on application-layer attacks, specifically slow and flood attacks. The technique used to detect DoS attacks in microservice applications must be aware of the diversity of the services. The detector must be capable of analysing all the dataset's features from all services simultaneously. This is the challenge when dealing with microservices because a single application is composed of several services that can be impacted differently by the attack.

4 EXPERIMENTAL EVALUATION

This section describes the steps executed to perform the proposed approach. Figure 2 shows the TeaStore application as the target system, Hulk and Torshammer as attack tools, Browser workload profile using JMeter, cAdvisor as the monitor and Telegraf and InfluxDB as data collector and storage, respectively.

4.1 Setup

Our setup consists of a machine equipped with an Intel Core i5-10400 processor with 12 logical cores at 2.90GHz and 64GB RAM. We created four *Virtual Machine*s (VMs), named Client, Master, Worker1, and Worker2, with the following configuration: *Master* has 2 CPU cores and 8GB RAM; *Worker1* has 6 CPU cores and 32GB RAM; *Worker2* has 2 CPU cores and 4GB RAM; *Client* has 1 CPU core and 14GB RAM. Figure 2 shows how the technologies were deployed in the VMs.

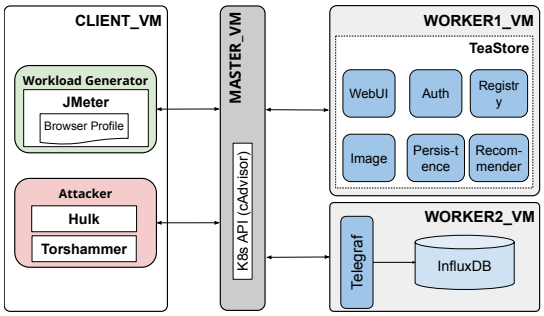

Figure 2: Experiment Setup

Our target system is the TeaStore application. It is a Microservice-based application for benchmarking, evaluating performance modelling, and resource management research [17]. It contains five services: WebUI, Auth, Image-Provider, Persistence, and Recommender. Also, it has a Registry service and a database. Each of the five services has unique functionalities and can be replicated, added, or removed at runtime. TeaStore runs in Kubernetes, a container orchestration system to deploy, manage, and scale microservice applications [6]. For this study, we need to define each service's memory and CPU requests and limits. Based on the available resources, we defined the following: the memory request is 1GB; the memory limit is 3Gb for all services; and the CPU request and limit per service are: Auth and Image - 400m and 800m; Persistence - 500m and 1000m; Recommender - 210m and 420m; WebUI - 750m and 1500m.

We use the cAdvisor tool [8] to monitor the TeaStore application providing resource usage and performance information of the containers (also known as pods) at runtime. Table 1 describes the selected monitoring metrics. Once the cAdvisor provides the monitored data, the Telegraf is the agent responsible for collecting and storing the data in the InfluxDB database, a time-series database.

Table 1: Monitoring Metrics

Name	Description
cpu_cfs_periods_total	Elapsed enforcement period intervals
cpu_cfs_throttled_seconds	Duration container has been throttled
cpu_system_seconds_total	System CPU time consumed
cpu_user_seconds_total	User CPU time consumed
cpu_usage_seconds_total	CPU time consumed
cpu_cfs_throttled_periods_total	Throttled period intervals
memory_usage_bytes	Memory usage
memory_working_set_bytes	Working set
memory_RSS	Size of RSS
memory_max_usage_bytes	Maximum memory usage recorded
memory_failures_total	Memory allocation failures
spec_CPU_period	CPU period of the container
spec_CPU_quota	CPU quota of the container
spec_memory_limit_bytes	Memory limit for the container
sockets	Open sockets for the container
threads	Threads running inside the container

For performing the attacks, we researched existing DoS tools to perform low- and high-volume attacks. After a preliminary analysis, we selected four tools (GoldenEye, Hulk, Slowloris, and Torshammer) and verified their impact on resource consumption and response time (RT). The tools that most impacted the system were **Torshammer** (RT of 462880ms) and **Hulk** (RT of 498ms). For comparison, the RT for the Golden Run (no attacks) is about 23ms. The tools GoldenEye and Slowloris got RT of 251ms and 49ms, respectively. Hulk is a tool created for performing a high-volume DoS attack, while Thorshammer is designed to perform low-volume HTTP Post attacks.

A single run is divided into five stages: warm-up, pre-attack, attack window, pos-attack, and end totalizing in 35-minutes of running. The JMeter performs normal requests (i.e., non-malicious) during the entire run. First, there are 10 minutes of **warm-up**, which is the time necessary to warm up the TeaStore application after being deployed. In these 10 minutes, the system performance has a high variance and can not be used in the analysis. The **pre-attack** is a five-minute stage only with regular requests. Then, there are the 10 minutes **attack windows** where one or multiple attacks can happen in this interval. Then, the **pos-attack** is a 5 minutes stage with only regular requests. Finishing the run, there is the **end**, a stage to let the requests finish, which will be discarded in the analysis phase.

4.2 Data Generation

We use the Browsing profile, a workload script provided by the TeaStore authors, for generating the workload. This profile emulates users browsing the store, logging on, adding items to the cart, and discarding those items. These paths impact all five services in distinct moments with different intensities. To execute this script, we use JMeter, an open-source application developed in Java, to perform load test functional behaviour and measure performance [2]. The JMeter script is executed with 300 users, a ramp-up of 1 second, a duration of 2100 seconds, and a JMeter plugin, called Constant Throughput Timer, is used to maintain a constant of 10000 requests per minute.

We defined four attack profiles that intend to cover several attack situations and increase the diversity of the generated data. Figure 3[a] presents the Standard profile in which the attack takes place during the whole 10-minutes available of the attack window. Figure 3[b] represents the 2M attack with a short duration of 2 minutes. It can happen between 6 to 7min, 10 to 11min or 14 to 15min. Then, Figure 3[c] shows the INT3 attack, which has four attacks of 1 minute in variable intervals. The first attack starts in the first minute of the stage, and then there is a 1-minute pause. After that, another attack starts, so there is a 3 minutes pause, the third attack starts, the last pause of 2 minutes, and the last attack starts at the last minute of the attack window. Finally, Figure 3[d] presents the ALT3 attack with fixed intervals but variable attack duration. It is defined as follows: 2 minutes of attacks, 2 minutes of interval, 3 minutes of attacks, 2 minutes of interval, and 1 minute of attacks.

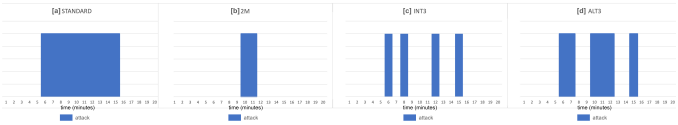

Figure 3: Attack Profiles

We use the Hulk and Torshammer tools to perform the attacks. There are some differences between both tools; in Torshammer, we pass the IP and the port as separate parameters, while in Hulk, we only need to pass one parameter with the URL and port. This allows us to define different pages in different executions using the Hulk tool. For the Standard profile, the attack tools perform the attacks for the entire 10-minutes against the home page of the TeaStore application. For the Hulk tool, we have two subcategories of the Standard profile that have the same duration (10 minutes) but attack the Category and Product page. This difference is because those pages use different services of the TeaStore application. For example, the product page presents product recommendations that use the recommender service. In the INT3 profile, all the attacks executions are against the home page as well as in the 2-minutes profile. The ALT3 is only conducted using the Hulk tool, and a different page is focused on each of the three attacks: the category page first, then the home page, and lastly, the product page. We execute each experience ten times, counting 110 runs executed and collected for this work.

4.3 Datasets and ML Algorithms

We perform a set of steps to construct the dataset from the collected monitoring data. First, we collect the data stored in the database for the duration of the running of all five TeaStore services and save it in a CSV file. Then, the **Pre-processing** phase, implemented in the Python programming language, starts. In this, the features are merged by timestamp. Thus, a sample contains all the collected metrics. Sometimes (rarely), samples have missing data in one of the features, but when this occurs, the **Missing data** step deals with it. Metrics with fixed values are filled with the average (e.g., spec CPU quota); other metrics are filled with the *bfill* method that backward fill the missing values in the dataset. After that, the percentage of CPU usage, CPU Throttled, and memory usage are calculated following the consumption of available resources. Finally, the three versions of the dataset are created.

The **Individual** dataset does not need more intervention after the previous steps. Data for each service for each run are saved in a separate CSV file. The **Sequential** dataset concatenates all the services for the same run together. After the samples are concatenated, they are sorted by time, and the CSV file is saved. To create the **Parallel** dataset, each service file is merged based on the timestamp and has a tolerance time of 15 seconds. Still, it can have some missing data because the metrics are not shared in synchrony. It can take 9 to 25 seconds for the cAdvisor to post the next collected metric. We deal with the missing data using the backward fill method. Thus, in this one, all samples have the features from all five services (WebUI, Auth, Image, Persistence and Recommender).

For this work, we selected classification algorithms used by related works [7][9]. The ML algorithms are K-Nearest Neighbors (n_neighbors = 5), Logistic Regression, Naïve Bayes, Random Forest (max _depth = 5, n_estimators = 50), Support Vector Machines (kernel = linear), and XGBoost. The models are trained using the collected data from regular use of the system and attacks and used to identify the attacks at runtime.

Attack Detection is a classification and time series problem [5]. So, to assess the model performance, splitting the data into training and testing is necessary. In problems with such characteristics, i.e. being dependent on various experiments, it is recommended to use the *Experiment-wise Leave-one-out Cross-Validation (ELOOCV)* [5]. In that case, all the experiences are used for training except for one, that is used as the test set to test the model's performance. All the Golden Run experiments are used for training; only the attacks experiments go into the ELOOCV. This process is repeated for all the experiments, thus providing cross-validation.

5 RESULTS

After implementing and executing the proposed approach and generating the data, it is time to classify the datasets using the ML algorithms. Therefore, this section presents and discusses the classification results.

Based on the results, we can state that ML can be used to detect low- and high-volume DoS attacks (**RQ3**). We analysed the results of all six algorithms (*Random Forest* (RF), *Support Vector Machine* (SVM), *K-Nearest Neighbors* (KNN), *Naive Bayes* (NB), Logistic Regression (LR), and XGBoost (XGB)) for the Sequential and Parallel datasets for both high- and low-volume attacks. When analysing the parallel dataset, the XGB has the best results with an *F1-score* mean of 97.6%, it is followed by the RF with a mean of 94.8%. When dealing with the sequential dataset, the XGB is overcome by the KNN in some attack profiles achieving a *F1-Score* mean of 87.7% against 87.2% of XGB. It can be explained by the parallel dataset having about five times more features than the sequential since we have all services in a single sample. XGBoost is known to be good at dealing with several features, indicating that it is a good algorithm for dealing with microservices scenarios.

5.1 Metrics for DoS Attacks Detection

To analyse which metrics to collect (**RQ1**), we used the RF algorithm to investigate the feature importance for the predictions. The results showed that the number of sockets and the CPU throttled percentage are highly important. Most of the works that monitor and assess the performance of the systems use resource consumption metrics (i.e., CPU and memory usage). Based on these two pieces of information, we defined four metrics groups to investigate the detection success for each group. The groups are: **[Group A]** 'sockets', 'threads'; **[Group B]** 'throttleed%', 'CPU CFS throttled seconds', 'CPU system', 'CPU user', 'CPU usage'; **[Group C]** 'memory usage', 'memory working', 'memory max', 'memory rss', 'container memory failures total'; and **[Group D]** 'CPU usage', 'memory usage'.

Table 2: Metrics Groups RF Classification Results

	GROUP A		GROUP B		GROUP C		GROUP D	
Metrics	High	Low	High	Low	High	Low	High	Low
Precision	0.96 ± 0.04	0.91 ± 0.07	0.83 ± 0.07	0.99 ± 0.03	0.1 ± 0.3	0.12 ± 0.26	0.8 ± 0.13	0.98 ± 0.03
Recall	0.94 ± 0.03	0.8 ± 0.16	0.78 ± 0.08	0.85 ± 0.05	0.01 ± 0.02	0.02 ± 0.06	0.57 ± 0.24	0.83 ± 0.04
F1	0.95 ± 0.03	0.84 ± 0.12	0.8 ± 0.05	0.91 ± 0.04	0.01 ± 0.03	0.04 ± 0.09	0.64 ± 0.22	0.9 ± 0.03

We classified the groups using the ML algorithms for both low- and high-volume DoS attacks. Table 2 presents the RF classification results for the INT3 experiences. For the high-volume attacks, *Group A* has better results. This is very interesting because, to the best of our knowledge, container threads and sockets features are not used to detect anomalies and attacks in containerised applications in existing works or even to assess the system's performance. For low-volume attacks, *Group B*, composed of the CPU-related features, has the best result, closely followed by the classical *Group D*. Still, *A Group* has significant results. In this scenario, memory-related metrics are not very influential, especially in high-volume attacks. It is possible to see how the attacks affect different properties of the system. The CPU-related metrics (Group B) have a major impact on detecting low-volume attacks, while Group A has a major impact on high-volume attacks. This interesting result indicates that future works should include the number of socks and threads.

5.2 Datasets Analysis

In order to answer **RQ2**, we compare the results of parallel and sequential datasets and analyse the individual dataset. Regarding the parallel and sequential datasets, the results show that the parallel increases the success of the results considerably. For instance, in the 2M experiments with the high-volume attack, the XGBoost results increased on average by more than 10% (e.g., *F1-Score* went from **0.79%** to **0.98%**). It may not be as perceived in the low-volume as it is in the high-volume attack (e.g., in the low-volume attack, the XGBoost *F1-Score* for 2M went from **0.89%** to **0.95%**). So, organising the data in parallel is more significant for high-volume attacks. Hence the low-volume attack prevents the system from answering any regular request, and all the services stop spending resources, being more accessible to be detected even if the service is not used with high frequency.

We can verify that in the experiences with 10-minute continuous attacks, the results are incredibly high, and this is due to the fact that attacks are long and easy to detect. Therefore, the INT3, 2M, and ALT3 results are more significant to analyse because they increase the challenge of the model to detect the attack. Still, the XGBoost got excellent results with the parallel dataset. The results show that evaluating all the service features simultaneously increases the chance of correctly classifying the samples.

Table 3: Individual Dataset - Random Forest Results

	Standard				
	WebUI	Auth	Image	Persistence	Recom.
Precision	0.99 ± 0.02	0.89 ± 0.3	0.88 ± 0.3	0.99 ± 0.3	0.34 ± 0.48
Recall	0.99 ± 0.01	0.83 ± 0.28	0.85 ± 0.28	0.90 ± 0.07	0.23 ± 0.29
F1	0.99 ± 0.01	0.86 ± 0.29	0.87 ± 0.29	0.94 ± 0.04	0.27 ± 0.34
	Prod				
Precision	0.99 ± 0.01	0.78 ± 0.39	0.68 ± 0.45	0.98 ± 0.2	0.88 ± 0.29
Recall	0.99 ± 0.01	0.76 ± 0.39	0.65 ± 0.43	0.96 ± 0.02	0.84 ± 0.30
F1	0.99 ± 0.01	0.79 ± 0.39	0.67 ± 0.44	0.97 ± 0.01	0.86 ± 0.29

Next, we discuss the individual dataset results. When investigating the prediction for each TeaStore service, we can see that different services are affected depending on the attacker's focus. Table 3 presents the RF algorithm results per service for Standard and Product experiments. We can verify that WebUI results are the same in both experiments. Nevertheless, the others have slight to elevated differences. The greatest difference is related to the Recommender service. When the home page is attacked, it almost does not impact the Recommender service, so the results are close to 30% success against more than 80% when the product page is attacked because this page explicitly uses this service.

It is important to point out that even though the results of WebUI are close to perfection, we can not consider that only monitoring the User Interface service will be enough to detect high-volume attacks. It may be true in this strict scenario. However, the attacker may find a vulnerability that will impact another service that is not the WebUI. So, if it is not being monitored, it can lead to ignoring an attack. Therefore, this result highlights how tricky it can be to construct a model to detect attacks on the microservice application not to be overfitting.

6 CONCLUSION

In this work, we proposed an approach to assess the detection of low- and high-volume application-layer DoS attacks. We implemented a test bed using ML models as the attack detector and performed experiments to answer the research questions. The results show that the ideal way is by combining all metrics from all application services in the same sample. Regarding the ML classification algorithms, the XGBoost achieves better performance, is the most suitable and presents the more accurate results. Low- and high-volume attacks impact the system in different ways. The low-volume attack can be easier to detect because it affects all the services. However, high-volume attacks can impact different combinations of services depending on which is the focus. Thus, the detection can be more challenging in microservice applications than in monolithic systems.

ACKNOWLEDGMENTS

This work is supported by the project AIDA – Adaptive, Intelligent and Distributed Assurance Platform (POCI-01-0247-FEDER-045907), co-financed by the European Regional Development Fund (ERDF) through the Operational Program for Competitiveness and Internationalisation (COMPETE2020), the CENTRO 2020 Portugal Regional Operational Program and by the Portuguese FCT under CMU Portugal. This work is funded by the Operational Program for Competitiveness and Internationalization (COMPETE 2020) and FCT - Foundation for Science and Technology, I.P./MCTES through national funds (PIDDAC), within the scope of CISUC R&D Unit - UIDB/00326/2020 or project code UIDP/00326/2020.

REFERENCES

[1] Carlos M. Aderaldo, Nabor C. Mendonça, Claus Pahl, and Pooyan Jamshidi. 2017. Benchmark Requirements for Microservices Architecture Research. *Proceedings - 2017 IEEE/ACM 1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering, ECASE 2017* (2017), 8–13. https://doi.org/10.1109/ECASE.2017.4

[2] Apache Software Foundation. 2022. *Apache JMeter*. The Apache Software Foundation. https://jmeter.apache.org/

[3] Ataollah Fatahi Baarzi, George Kesidis, Dan Fleck, and Angelos Stavrou. 2020. Microservices made attack-resilient using unsupervised service fissioning. In *Proceedings of the 13th European workshop on Systems Security*. 31–36.

[4] Samuel Black and Yoohwan Kim. 2022. An Overview on Detection and Prevention of Application Layer DDoS Attacks. In *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 0791–0800.

[5] João Rodrigues de Campos. 2022. *Advanced Online Failure Prediction Through Machine Learning*. Ph. D. Dissertation. 00500:: Universidade de Coimbra.

[6] Cloud Native Computing Foundation. 2022. *Kubernetes*. Kubernetes. https://kubernetes.io/

[7] Qingfeng Du, Tiandi Xie, and Yu He. 2018. Anomaly detection and diagnosis for container-based microservices with performance monitoring. In *International Conference on Algorithms and Architectures for Parallel Processing*. Springer, 560–572.

[8] Google. 2022. *cAdvisor*. Google. https://github.com/google/cadvisor

[9] Johannes Grohmann, Patrick K Nicholson, Jesus Omana Iglesias, Samuel Kounev, and Diego Lugones. 2019. Monitorless: Predicting performance degradation in cloud applications with machine learning. In *Proceedings of the 20th international middleware conference*. 149–162.

[10] Ghafar A Jaafar, Shahidan M Abdullah, and Saifuladli Ismail. 2019. Review of recent detection methods for HTTP DDoS attack. *Journal of Computer Networks and Communications* 2019 (2019).

[11] Francisco Sales de Lima Filho, Frederico AF Silveira, Agostinho de Medeiros Brito Junior, Genoveva Vargas-Solar, and Luiz F Silveira. 2019. Smart detection: an online approach for DoS/DDoS attack detection using machine learning. *Security and Communication Networks* 2019 (2019).

[12] Modupe Odusami, Sanjay Misra, Olusola Abayomi-Alli, Adebayo Abayomi-Alli, and Luis Fernandez-Sanz. 2020. A survey and meta-analysis of application-layer distributed denial-of-service attack. *International Journal of Communication Systems* 33, 18 (2020), e4603.

[13] Amit Praseed and P Santhi Thilagam. 2018. DDoS attacks at the application layer: Challenges and research perspectives for safeguarding web applications. *IEEE Communications Surveys & Tutorials* 21, 1 (2018), 661–685.

[14] Alireza Ranjbar, Miika Komu, Patrik Salmela, and Tuomas Aura. 2017. Synaptic: Secure and persistent connectivity for containers. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 262–267.

[15] Areeg Samir and Claus Pahl. 2020. Detecting and localizing anomalies in container clusters using Markov models. *Electronics* 9, 1 (2020), 64.

[16] Karanpreet Singh, Paramvir Singh, and Krishan Kumar. 2017. Application layer HTTP-GET flood DDoS attacks: Research landscape and challenges. *Computers & security* 65 (2017), 344–372.

[17] Joakim Von Kistowski, Simon Eismann, Norbert Schmitt, Andre Bauer, Johannes Grohmann, and Samuel Kounev. 2018. TeaStore: A micro-service reference application for benchmarking, modeling and resource management research. *Proceedings - 26th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, MASCOTS 2018* (2018), 223–236. https://doi.org/10.1109/MASCOTS.2018.00030

[18] Lingzhi Wang, Nengwen Zhao, Junjie Chen, Pinnong Li, Wenchi Zhang, and Kaixin Sui. 2020. Root-Cause Metric Location for Microservice Systems via Log Anomaly Detection. *Proceedings - 2020 IEEE 13th International Conference on Web Services, ICWS 2020* (2020), 142–150. https://doi.org/10.1109/ICWS49710.2020.00026

[19] Tao Wang, Jiwei Xu, Wenbo Zhang, Zeyu Gu, and Hua Zhong. 2018. Self-adaptive cloud monitoring with online anomaly detection. *Future Generation Computer Systems* 80 (2018), 89–101.

[20] Gregory B White, Eric A Fisch, and Udo W Pooch. 2017. *Computer system and network security*. CRC press.

[21] Tetiana Yarygina and Christian Otterstad. 2018. A game of microservices: Automated intrusion response. In *IFIP International Conference on Distributed Applications and Interoperable Systems*. Springer, 169–177.

[22] Dongjin Yu, Yike Jin, Yuqun Zhang, and Xi Zheng. 2019. A survey on security issues in services communication of Microservices-enabled fog applications. *Concurrency Computation* 31, 22 (2019), 1–19. https://doi.org/10.1002/cpe.4436