MIDPS: A Multi-agent host Intrusion Detection and Prevention System

Saad El Jaouhari, Andrei Vavilov, Julia Soloveva, Alexandru Archip, Nour El Madhoun

MIDPS: A Multi-agent host Intrusion Detection and Prevention System

Saad EL JAOUHARI, Andrei VAVILOV, Julia SOLOVEVA, Alexandru ARCHIP and Nour EL MADHOUN

Abstract Web servers are the backbone of the modern Internet, handling user requests, managing data, and hosting applications, making them frequent targets for cyberattacks. In this context, HTTP logs are valuable source of information for analyzing and identifying potential threats. They have been used in Host-Based Intrusion Detection Systems (HIDS) to identify cyberattacks, but evolving threats demand ongoing research and new solutions. In this paper, a Multi-agent host-based Intrusion Detection and Prevention System (MIDPS) is proposed. It analyzes HTTP logs to detect various categories of cyberattack, such as file disclosure, SQL injection, and (D)DoS attacks. The proposed strategy is adaptive and independent of the analysed web application, as MIDPS automatically crawls the input application for new contents, recomputes and updates its threat detection heuristics. This is due to its adoption of a multi-agent approach, enhanced by the integration of Machine Learning. Moreover, when deployed on servers hosting web applications, MIDPS neutralised 100% of incoming File Disclosure, SQL injection, and (D)DoS attacks.

1 Introduction

In the modern digital landscape, web servers face escalating threats. Traditional security measures often fall short in detecting and preventing these threats effectively, particularly in static web applications and servers handling static content. To address this, a Multi-Agent Host Intrusion Detection and Prevention System (HIDPS)

Saad EL-JAOUHARI & Nour EL MADHOUN
Isep: École d'ingénieurs du numérique, France, e-mail: firstname.lastname@isep.fr

Andrei VAVILOV & Alexandru ARCHIP
Technical University of Iași, Romania, e-mail: first.lastname@academic.tuiasi.ro

Julia SOLOVEVA
University of Applied Sciences Campus Vienna, Austria

is proposed, leveraging HTTP logs as a rich source of data for identifying and mitigating these attacks. This paper proposes a system called **MIDPS** (Multi-Agent Host Intrusion Detection and Prevention System), which operates on HTTP logs. HTTP is the backbone of the web, providing crucial information that can be extracted from the corresponding logs. MIDPS is primarily designed for static web applications and servers that mainly handle static content. The proposed solution comprises a *Backbone, Logs Analyser, Attack Detection Modules*, and *Prevention Module* to counter multiple attacks namely: network-based attacks such as (D)DoS, web-based vulnerabilities like SQL Injection (SQLi), and privacy breaches exemplified by File Disclosure. This list of identified threats serves as a Proof of Concept (PoC) for ongoing research and development, with plans to expand the list in our future works. The efficacy of MIDPS in detecting and blocking these attacks has been thoroughly examined.

The central research question of this paper is: *How to dynamically and effectively protect web application and hosting servers in **real-time** against various types of cyberattacks using HTTP logs?*

The following contributions were achieved during this work: 1) An adaptive and application independent solution that analyses real-time traffic logs to detect cyberattacks. The solution combines rule-based (where the rules are dynamically generated) and Machine Learning (ML) approaches. 2) A solution that applies bans on the fly in order to block attacks in real-time.

The proposed strategy is adaptive and independent of the web application, as MIDPS automatically and continuously crawls the input application for new contents (web pages and links), then recomputes and updates its threat detection heuristics. It combines two techniques to cover various categories of cyberattacks: rule-based agents to detect File Disclosure and DDoS attack, while it uses Machine Learning models to detect SQLi.

We opted for a rule-based approach to detect file disclosure attack since we can efficiently crawl the web application on the server to recover its file structure. The latter will be used to generate access patterns, allowing us to reliably and accurately derive the expected requests (normal behaviour) to the server. The objective is to reduce the numbers of false positive since the actual analysis comes down to comparing a list of known items. Moreover, giving the fact that we work with web logs, we also opted for a rule-based approach to detect and prevent (D)DoS attacks. For this purpose, heuristics that combines the frequency of HTTP requests, the source IP address(es) and a sliding window approach were used. Finally, to detect SQLi attacks we used ML models due to its dynamic nature and its suitability to analyse dynamic payloads (i.e., SQLi payloads in the HTTP requests).

This paper is structured as follows. Section 2 presents a state of the art analysis together with a comparison. Section 3 presents the proposed solution architecture. Section 4 discusses the experiments and tests conducted to validate our PoC. Finally, Section 5 provides a set of conclusions and future works.

2 State of the Art

The use of statistical techniques such as Machine Learning (ML) and Deep Learning (DL) in intrusion detection and prevention has intrigued numerous researchers over the last decade. Moreover, classical rule-based approaches have also been widely used [5], [1]. This section surveys studies proposing IDS tools leveraging ML and rule-based technologies on server HTTP logs. Moreover, a comparison of the mentioned papers is provided in Table 1.

In [3], the authors tackled anomaly detection on web servers. They gathered logs from three distinct Apache web servers and developed multiple detection models focusing on attribute length, character distribution, presence or absence, and order. In [2], supervised ML techniques were employed to identify prevalent web attacks like SQLi, Cross-site scripting (XSS), Command Injection (CMDi), and path traversal. The researchers compared various algorithms and determined that Random Forest detected these attack types effectively. In [6], the researchers employed the Apriori algorithm for Data Mining (DM) purposes, extracting association rules. These association rules were then applied to data generated by attack-simulating tools. The study yielded promising results, demonstrating that the tool could effectively identify attacks at an early stage. The work in [5] revealed the effectiveness of reducing false positive rates by applying generalization. The researcher emphasised the significance of pinpointing specific areas where generalization can be effectively implemented. The study encompassed a wide range of web attacks and employed network packet data as the data source. The issue of high false positive rates was extensively discussed in [9]. The authors highlighted that the cost of a mistake in detecting attacks is often high, underscoring the importance of prioritizing efforts to lower the false positive rate. Efforts have been made to leverage Natural Language Processing (NLP) for feature extraction from server logs in [10]. The authors proposed a solution that involves intelligent extraction of vectorised features using neural networks, specifically employing the Word2Vec method and the Convolutional Neural Network for Text (TextCNN) model. Once the vectorised features are extracted, boosting classification techniques are applied to detect anomalous logs. Boosting algorithms combine multiple weak classifiers to create a robust classifier. In [11], the authors employed n-gram models for feature extraction in weblogs. This approach enables the identification of patterns in accumulated logs. This tool adopts a similar technique by utilizing a sliding time window to analyse a log sequence, proving effective in detecting attacks with identifiable log patterns. The authors concluded that models based on Support Vector Data Description (SVDD), K-means, and Density-Based Spatial Clustering of Applications with Noise (DBSCAN) exhibited the highest accuracy in detecting intrusive HTTP logs. In [4], the authors presented AMiner, an open-source tool that enables fast log parsing, analysis, and alerting. The solution is a pipeline for log data analysis and intrusion detection. They ingests logs from diverse sources and uses custom and automatically generated parser trees to process the data.

On the other side, researchers have raised concerns regarding using ML in cybersecurity, as highlighted in [9][1]. The authors in [1] analysed 30 papers and

summarised the pitfalls of employing ML for intrusion detection. They draw attention to issues such as *sampling bias* and *label inaccuracy* during the data collection and labeling stages, which can lead to a lack of representation of the data distribution. While these problems cannot be completely eliminated, they can be mitigated if addressed carefully. Both studies also criticize the lack of publicly available datasets that can be effectively utilised to develop field models. Therefore, creating new datasets can help in addressing those issues.

Table 1: Comparison of our work with relevant state-of-the-art methods

	[5]	[6]	[2]	[3]	[10]	[11]	[4]	**MIDPS**
Adaptablity	✗	✗	✗	✗	✗	✗	✗	✔
Methods	Generalization, heuristic	Apriori algorithm	Random Forest	Selected features	NLP	Unsupervised ML	Parser trees	Dynamic Rules, ML
Multiple attacks detected?	✓	✓	✓	✓	✓	✓	✗	✔
Datasets	HTTP logs, Pugillis	Apache logs	HTTP Param Dataset	Apache logs	CSIC 2010	HTTP logs	Audit, Apache, Syslog, Suricata logs	HTTP Logs
Detection / Prevention	✗	✗	✗	✗	✓	✗	✗	✔
Different types of logs	✗	✗	✗	✗	✗	✗	✓	✗
Reproducibility	✓	✓	✗	✗	✗	✗	✓	✔

Table 1 compares different works based on multi-criteria. 1) Adaptability: is the solution flexible enough to detect attacks in evolving web servers and their applications? Our solution excels in this context as, for instance, it dynamically crawl the web application for new content and then adapts its modules accordingly. 2) Detection method(s) used. 3) Detection of multiple type/categories of attacks. 4) Dataset(s) used. 5) Does the solution propose both detection and prevention techniques? Also a strong point in our solution since MIDPS applies bans on the fly to stop the ongoing attacks. 6) Ingests different types of logs. Our solution handles only standardised HTTP logs, but future works involve dealing with different types of logs. 7) Reproducibility of the work.

3 Proposed solution

Architecture

The proposed MIDPS, actively monitors HTTP logs for normal behavior and potential anomalies or suspicious activities and can detect three types of attacks: *File Disclosure*, *(D)DoS*, and *SQLi*. If there are any deviations detected, the data gets sent to *Attack Detection Modules*. Designed for modularity, the framework integrates new attack types via a central hub, enabling comprehensive and responsive

threat detection. More details about the main components of MIDPS are presented in Fig. 1. Once the determination has been made whether or not an attack has occurred, the MIDPS processes this verdict and reacts based on the findings. The source code for the proposed MIDPS can be found at [1].

HTTP Logs: A Valuable Data Source for MIDPS

Web servers store files, including the content of web pages, and provide them to clients based on their requests. The connection between a web server and a client occurs mainly over HTTP. Billions of HTTP logs are generated daily, providing valuable information about all activities happening on the server. However, attackers often target web servers seeking to obtain user data, disrupt services, or even gain full control over the server. This damage can be avoided using a robust HIDS combined with other tools, such as firewalls.

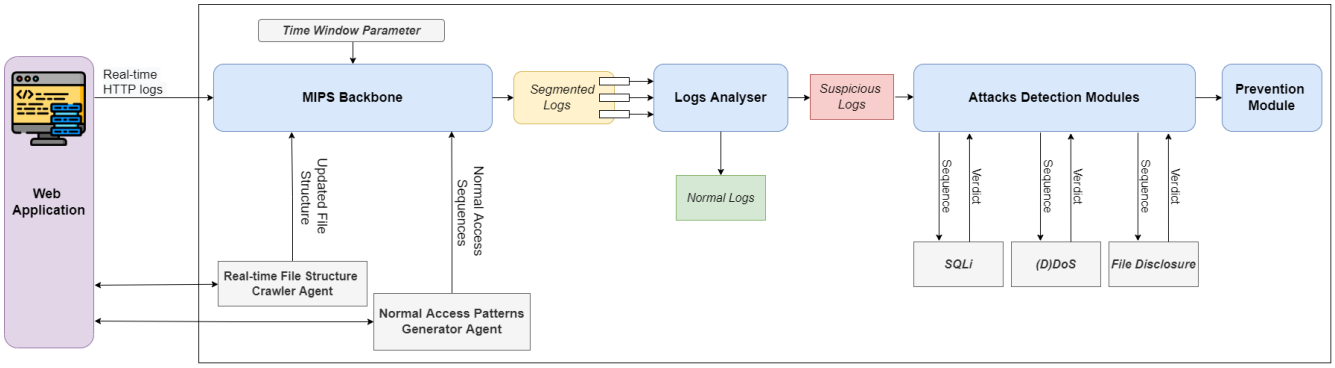


Fig. 1: Architecture of MIDPS

MIDPS Backbone

This module initializes by launching agents, processing HTTP logs, and configuring the Fail2Ban jail *(named "MIDPS-jail")*. It achieves this by segmenting the logs into **time windows** and forwarding them to the Logs Analyser module. The outputs of this module are segmented logs.

Normal Access Patterns Generator Agent

This agent continuously crawls each web page of the web application at specific intervals, typically quantified in seconds. The primary purpose of this agent is to construct the *standard access sequences* of the web application (or the normal behavior of a web app). The *"normal" access sequence* is defined as the specific series of requests made when a user accesses a particular page. Once all web pages have been crawled, the agent extracts all logs produced within a designated time frame. The logs are then partitioned by time windows, with the time window inter-

[1] github.com/andutzu7/MIPS

val being the same as the aforementioned “x” seconds interval. The newly generated sequences are compared with the pre-existing ones, and if any differences are found, the outdated sequences are replaced with the new ones. This allows to update the patterns easily when new features are added to the web application.

Real-time File Structure Crawler Agent

This agent periodically crawls the web application to rebuild its file structure as a graph. Its main objective is to monitor the file structure of the web application, enabling it to discern any changes and to initiate automatic updates to MIDPS modules if alterations are detected as shown in Fig. 2. This agent performs the crawling operation employing “wget” and subsequently extracts the logs it generates post a designated timestamp. The agent then recalculates the file structure graph and compares it to the previous version. If the two graphs do not match, the previous graph is replaced with the newly generated one.

Logs Analyser

MIDPS collects all requests occurring within a pre-established time frame, quantified in seconds. This time frame is then subjected to a sanitization process, eliminating requests originating from the local machine, specifically those produced by crawling agents. Once cleared, this refined data is forwarded to the *Logs Analyser* module for subsequent scrutiny. A comprehensive logs analysis necessitates prior categorizing requests from the given time frame based on originating IP addresses. These requests are then segmented into chronological sequences for comparative analysis. This assembled data, segmented according to time and source, is cross-referenced against the output of the *Normal Access Patterns Generator Agent*. This process facilitates the identification of any irregularities or deviations. In cases where a sequence fails to align with established patterns, it is tagged as **“Suspicious”** and is redirected to the *Attack Detection Modules*. These specialised modules are designed to detect and counter various forms of threats. Parallel processing improved the efficiency and robustness of the module through a multi-threaded framework, with each thread analyzing requests from a specific IP address. This approach ensures systematic and efficient data processing, thereby improving overall system performance and reliability. If the request pattern matches one of the already established “normal access patterns,” then the user request is considered **“Normal”**. To resume, the log analyser provides the inputs for the detection and prevention modules. This module is not an active part in "decision making", that’s why it is separated from the Attacks Detection Modules, as can be seen on Fig. 1.

Attack Detection Modules

This block facilitates the seamless integration of various attack detection modules in plug-and-play mode to enhance the extensibility of our tool.

- File Disclosure refers to a security vulnerability where an attacker gains unauthorised access to sensitive files on a web server. Implementing this attack detection module involved thoroughly examining the web application. A web crawling process that recursively accesses all the web resources was executed, and the generated logs were then employed to reconstruct the structure of the web application. This reconstruction was a tree/graph in which each resource (such as images, JavaScript files, CSS files, etc.) is directly linked to its referring entity. Fig. 2

illustrates a segment of this reconstructed resource tree. This architectural layout facilitates the identification of File Disclosure attacks. The incoming requests are parsed and cross-referenced with the resources they demand. Any request soliciting non-existent resources or those not mapped within the tree is flagged as suspicious.

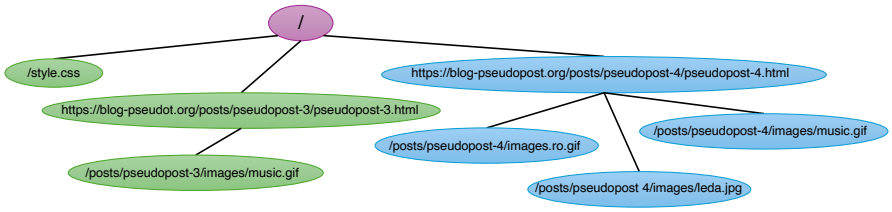


Fig. 2: A segment of the constructed resource tree showcasing the structure of the web application. The green leaves represent an extract of the old file structure, while the blue one represents a new update caught by the crawling agent.

- The (D)DoS module is based on quantitatively analysing incoming logs. The approach is rule-based and checks whether a sequence of logs exceeds a given threshold. The IDPS generates the threshold in an adaptive manner as it computes by multiplying the length of the longest normal access sequence by a given ratio. If the number of requests exceeds the threshold, depending on the number of source IPs (a single or multiple), the requests are flagged as suspicious, and ban / blacklisting rules are generated.
- The SQL injection detection module conducts a line-by-line analysis of the incoming logs. This module utilizes an ML algorithm, specifically a Gradient Boosting Classifier model, to ascertain if the resource field of a log line incorporates elements of an SQLi query. The feature extraction methods proposed by [7] were adopted in the training phase. The data features used for training the model are Entropy and G-Score. The decision to employ a Gradient Boosting Classifier resulted from a series of comparative tests between different algorithms, namely Gradient Boost Classifier, Naive Bayes, Support Vector Machines and Random Forest. The test results are illustrated in Table 2.

Table 2: Assessment of various ML algorithms for SQLi detection from logs

	Accuracy	Sensitivity	Precision	Recall	F1 Score
Gradient Boosting Classifier	**0.95**	**0.96**	**0.94**	**0.96**	**0.95**
Naive Bayes	0.72	0.72	0.84	0.60	0.70
SVM	0.78	0.62	0.70	0.62	0.76
Random Forest	0.77	0.69	0.86	0.67	0.75

Prevention Module

After the *Attack Detection Modules* finished analysing the logs, this module applies the verdicts. MIDPS is designed to save all detected logs as malign to a fixed location. The *Prevention Module* iterates through the log files placed in that folder, extracts the IP addresses of the malicious actors, and then blocks all their network traffic temporarily or permanently.

4 Experiments

MIDPS utilises the standard nginx/apache web server log format. Each module relies on different fields to perform an analysis. The tests were conducted against a Linux platform hosting a static web application using nginx as a web server. The platform runs on Debian 11, with kernel version Linux 5.10.0-21-amd64.

4.1 Dataset Description

The lack of standardized HTTP log datasets for model training and testing, especially with diverse cyberattacks, is a known challenge [1]. To address this, we created our own HTTP server log dataset for experiments. We separated the dataset in two categories: **normal usage** and **attacks**. First, we crawled the website to generate all possible legitimate traffic sequences. The *normal traffic* is defined as the requests that the server is expected to receive and respond to. Security experts analyzed these requests to ensure they are free from malicious traffic and cover nearly all expected normal server interactions. Those sequences of normal traffic were then used for filtering the dataset that we acquired by exposing to the Internet for three months a static website built with Hugo[2]. The dataset was enhanced by repeatedly crawling the application using tools such as wget and Torify. To simulate typical traffic, we randomly accessed links on the website using wget. We dispatched the requests at random intervals ranging between one to three minutes, with each request made through a proxy to vary the IP. The resulting *normal traffic dataset* comprises **21,384** log lines, **16,106** of which are manually generated.

The *attack dataset* captured traffic from real malicious actors and botnets targeting our server over three months. We also manually generated some malicious traffic using tools such as: Nikto, OpenVAS and a GET flood (D)DoS[3]. We generated abnormal traffic by instigating several attacks on our web application from various virtual machines. The malicious traffic dataset contains **577,905** log lines, with **563,405** of these being manually generated. The data is available in[4].

[2] https://gohugo.io/

[3] https://github.com/akashblackhat/DDos_Attack.py

[4] https://github.com/andutzu7/MIPS

The dataset used for training and testing of the ML model used to detect SQLi comprises several publicly available datasets from kaggle[5], in [8].

To summarize, the main objective is to be able to detect these attacks on the fly, using real-time streaming logs. As explained before, for both DDoS and File Disclosure attacks, the agents directly analyse the suspected logs on the fly to detect these attacks and apply bans. As for the SQLi ML models, the training and the testing steps use already existing dataset and the evaluation step uses the suspected logs in the locally generated logs. These logs are the result of simulated SQLi requests using sqlmap[6] tool. The perspective is to use streaming log data and online machine learning for real-time detection. Furthermore, Researchers indicate that blending data from various sources can introduce sampling bias [1]. To avoid this, we trained each component of MIDPS using distinct datasets, bypassing the need for any data integration. This approach provides a more robust and unbiased security framework.

4.2 Cyber attacks detection and prevention

(Distributed) Denial of Service

The target web server has been monitored over a 40-second interval, divided into the same four 10-second windows (4 steps). These steps highlight server activity using the access logs, the attack script logs, and the Fail2Ban logs over 10-second windows. Fig. 3a summarises these logs based on the requests per second rate. The *blue* line shows the number of active requests while a (D)DoS attack is in progress and its evolution while MIDPS is not active (*first scenario*). In contrast, the *orange* line presents the same statistic while having the MIDPS solution running and monitoring the protected web server (*second scenario*). During the first two steps (*Timeframe indices* 1 and 2–the gray vertical bars in Fig. 3a), the total number of active attack requests is over 3000 requests per second. MIDPS runs in the *log-collecting* phase. Thus, no actual log analysis is being performed. However, by the end of step 2, MIDPS detects the suspicious activity, activates the *Log Analysis* module, and enacts the *ban/blocking* components. These rules are activated on a per IP address basis, as shown in Section 3. The effect of this first wave of bans is clearly observable in step 3 (*Timeframe index* 3), and the attack is completely rejected by the end of step 4 (*Timeframe index* 4). These results demonstrate the efficacy against (D)DoS attacks. Furthermore, it is worth pointing out that the actual attack attempt is countered in less than 40 seconds after the initial suspicious activity is detected. Prevention is also ensured by actively maintaining the Fail2Ban module. The module achieves 100% detection rate by dynamically analyzing the suspicious activities/logs on the fly and promptly implementing bans to block potential attacks.

File Disclosure

[5] https://kaggle.com

[6] https://github.com/sqlmapproject/sqlmap

The experiments on MIDPS's effectiveness against File Disclosure attacks followed the same pattern as before. The target web server has been monitored over a 40-second interval, divided into the same four 10-second windows (or steps). Both test scenarios were kept –i.e., web server without MIDPS active and with MIDPS active. The results are summarised in Fig. 3b and follow the same color convention. Attack requests were successful during the first scenario, as can be observed by following the *blue* line in Fig. 3b. The malicious requests increased slowly, mainly guided by the increasing number of simulated attackers. As expected, once we have activated MIDPS during the second scenario, the success rate of the attacks drastically decreased between steps 2 and 3 (Fig. 3b–*orange* line between *Timeframe indices* 2 and 3). Full rejection of these malicious actors occurred by the end of step 4–yet again in less than 40 seconds since the initial attacks. Also, this module achieves 100% detection rate by dynamically analyzing the suspicious activities/logs on the fly and promptly implementing bans to block potential attacks.

SQL Injection

To train and test our model, we used the dataset provided in [8]. Furthermore, we tested the effectiveness of the SQL injection detection module following the same pattern as described above. The target web server has been monitored over a 40-second interval, divided into the same four 10-second windows (or steps). We simulated and SQL injection attack by using sqlmap[7]. As in the other experiments, the analysis is performed over two test scenarios, when MIDPS is inactive and when MIDPS is active. The results can be observed in Fig. 3c and follow the same color convention. The first scenario (blue line) shows how the number of attack requests keeps increasing over the course of the analysed period, stalling at around 60 requests per frame. The second scenario illustrates how the attack is detected and stopped one step after MIDPS is activated (Fig. 3b–*orange* line between *Timeframe indices* 1 and 2). The attack attempt was neutralised in less than 40 seconds since its launch. As mentioned in Table 2, the used Gradient Boosting Classifier achieved a 95% accuracy in detecting SQLi attacks. Moreover, for the evaluation step using generated logs using sqlmap, we succeeded to detect/block 100% of the related requests, which can be explained by the fact that the generated SQLi payloads are relatively easier to detect compared to the ones in the Kaggle SQLi dataset [8].

5 Conclusion and Future works

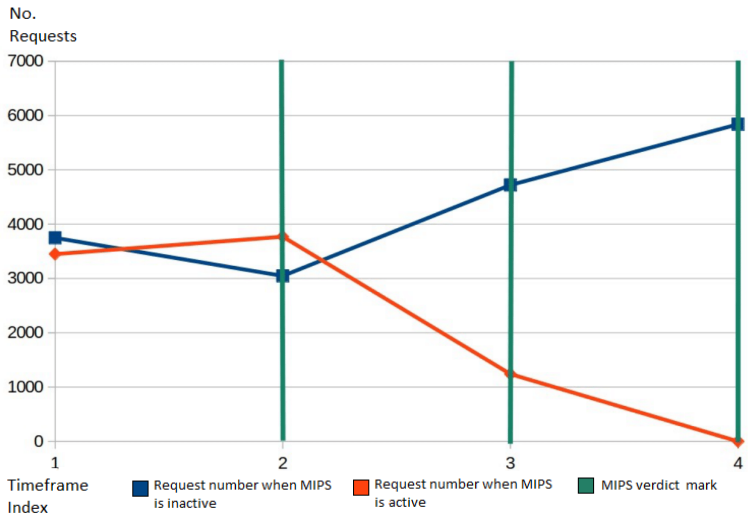
This paper proposed MIDPS, Multi-agent Intrusion Detection and Prevention System comprising two agents, the *Normal Access Patterns Generator Agent* and the *Real-time File Structure Crawler Agent*, both of which are included in the MIDPS Backbone. Additionally, a *Normal Logs Monitor* was integrated. As for *Attack Detection Modules*, the system can detect and prevent three types of attacks: File Disclosure, (D)DoS, and SQLi. The detection mechanisms for (D)DoS and File Dis-

[7] https://github.com/sqlmapproject/sqlmap

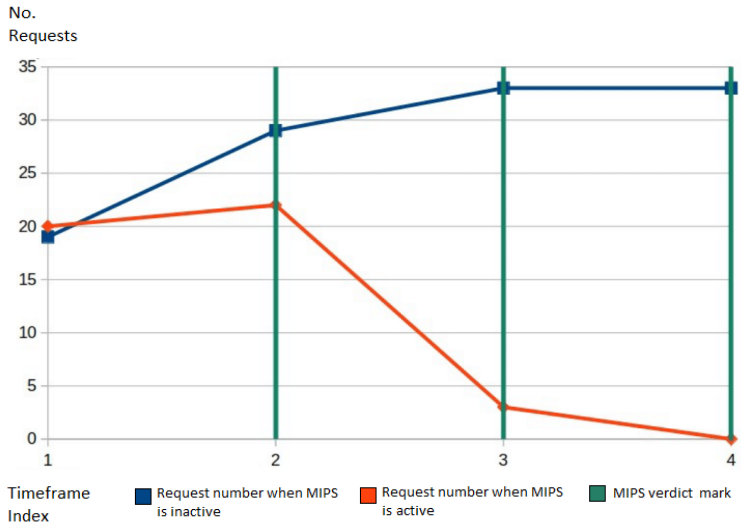
closure are rule-based. The (D)DoS approach calculates the request rate against a given threshold, which is computed in accordance with the web application's longest-length request. For File Disclosure, the web application's file structure graph is computed and checked to ascertain whether the resources in incoming requests are contained within this graph. The proposed system is most effective on servers primarily serving static content. Moreover, during the development of MIDPS, a unique dataset was created using HTTP server logs for the experiments. The experiments showed that the File Disclosure module achieved 100% detection rate, and due to its prevention module, it can also identify other attacks. (D)DoS attacks were recognised immediately after the second log frame, an impressive result. The solution is tailored for the monitored website, avoiding the reliance on a training dataset. Moreover, implementing additional ML algorithms for attack detection could further enhance the system's capabilities. Another future improvement would be expanding the degree of MIDPS parallelization, as the detection modules currently run sequentially. Finally, it is worth mentioning that this work is extendable and other attacks can be easily integrated into the attack detection modules.

References

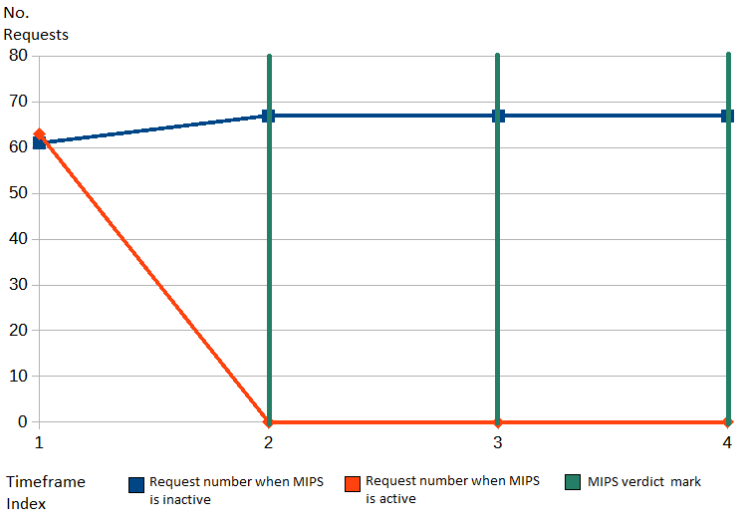
1. Arp, D., Quiring, E., Pendlebury, F., Warnecke, A., Pierazzi, F., Wressnegger, C., Cavallaro, L., Rieck, K.: Dos and don'ts of machine learning in computer security. arXiv preprint arXiv:2010.09470 (2020)
2. Hoang, D., Hung, N.: Detecting common web attacks based on supervised machine learning using web logs. Journal of Theoretical and Applied Information Technology **99**, 1339–1350 (2021)
3. Kruegel, C., Vigna, G.: Anomaly detection of web-based attacks. In: 10th ACM conference on Computer and communications security. pp. 251–261 (2003)
4. Landauer, M., Wurzenberger, M., Skopik, F., Hotwagner, W., Höld, G.: Aminer: A modular log data analysis pipeline for anomaly-based intrusion detection. Digital Threats **4**(1) (mar 2023). https://doi.org/10.1145/3567675
5. Lee, W.: A data mining framework for constructing features and models for intrusion detection systems. Columbia University (1999)
6. Mironeanu, C., Archip, A., Atomei, G.: Application of association rule mining in preventing cyberattacks. Bulletin of the Polytechnic Institute of Iași. Electrical Engineering, Power Engineering, Electronics Section **67**(4), 25–41 (2021)
7. Ross, K.: SQL Injection Detection Using Machine Learning Techniques and Multiple Data Sources. San Jose State University (2018)
8. SHAH, S.S.H.: sql injection dataset. Kaggle (2022), `https://www.kaggle.com/datasets/syedsaqlainhussain/sql-injection-dataset`
9. Sommer, R., Paxson, V.: Outside the closed world: On using machine learning for network intrusion detection. In: 2010 IEEE Symposium on Security and Privacy. pp. 305–316 (2010). https://doi.org/10.1109/SP.2010.25
10. Wan, W., Shi, X., Wei, J., Zhao, J., Long, C.: Elsv: An effective anomaly detection system from web access logs. In: 2021 IEEE International Performance, Computing, and Communications Conference (IPCCC). pp. 1–6 (2021). https://doi.org/10.1109/IPCCC51483.2021.9679413
11. Zolotukhin, M., Hämäläinen, T., Kokkonen, T., Siltanen, J.: Analysis of http requests for anomaly detection of web attacks. In: 2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing. pp. 406–411 (2014). https://doi.org/10.1109/DASC.2014.79



(a) (D)DoS Detection and Prevention



(b) File Disclosure Detection and Prevention



(c) SQL Injection Detection and Prevention

Fig. 3: Attacks detection and prevention using MIDPS