Modular Intrusion Prevention System

1st Ikenna Achinuhu
a.c.i.achinuhu@student.tudelft.nl

2nd Jorrit van Assen
j.s.vanassen@student.tudelft.nl

3rd Kevin Nanhekhan
k.r.nanhekhan@student.tudelft.nl

4th Mohamed Rashad
M.Rashad@student.tudelft.nl

5th Mostafa Khattat
m.khattat@student.tudelft.nl

6th DUC HUY Nguyen
duchuynguyen@student.tudelft.nl

***Abstract*—With the increasing usage of devices that are connected to networks, the risk of being targeted by malicious actors aiming to gain access or attain information from your device also increases. This paper will showcase a system that is created to prevent some of the attacks malicious actors may use. In particular, an Intrusion Prevention System is created that is able to ban these bad actors from the network when they try and access certain services more times than the threshold allows in a certain time span. The system is a modular Intrusion Prevention System, that consists of parses for the log files of each service, a database and components that observe the database and then ban IP addresses exceeding the configured thresholds, together with an interface that gives the user control over the system. Testing the system of some pre-configured log in attempts showed that the system was functional and had a lower response time than Fail2Ban, although improvements could be made to better handle log rotation.**

***Index Terms*—intrusion prevention system, network security, Intrusion detection**

I. Introduction

In the new digitalised and networked world in which every device is connected via the Internet by either wired or a wireless medium, it is a huge risk accompanied by the enormous potential of linked machines. The risk of violating security in which unauthorised disclosure of information, improper system altercation, and unavailability of correct services occur constantly exists and are unexpectedly growing strong [1].

Besides intricately strategised and carefully planned attacks with the malicious scheme, there are also attacks that are very easily implementable but could still cause serious damage when not prevented. These types of attacks can be done by anyone with a personal computer, but can also be done by more experienced and sophisticated adversaries. Examples of such attacks are denial-of-service and brute-forcing, both of which rely on making a large number of requests, either to continuously try and guess a password to gain access or disrupt the service of the victim.

This project aims to create a system that is capable of stopping these attacks from happening against certain applications. The way this can be achieved is by creating an Intrusion Prevention System (IPS), which is simply a system that monitors for malicious activities and once spotted tries to prevent them.

The version of such a system this project aims to create, monitors by keeping track of log-in attempts for four services, SSH [2], phpMyAdmin [3], Joomla [4] and WordPress [5]. It then also prevents malicious activities by banning the perpetrators from these attempts if they cross a certain threshold. Additionally, the system is implemented to be modular and allows on the fly changes to certain parameters for these thresholds in order to either tighten security or relax it.

To sufficiently prevent attackers, the system counts the number of times a certain IP address tries to access one of the services within a certain amount of time and then bans those for a certain period. It also makes use of a web interface to give the user control of the parameters and allow them to manually unban one of the previously banned IP addresses.

So, this report will discuss the IPS that was made in order to achieve these goals and go into depth on the components of which it comprises. Also, the report will showcase experiments that were carried out in order to confirm its functionality and other experiments that were done to measure its performance and whether it lives up to some of the requirements that were set. Together with those experiments the consequent results will also be talked about and compared to already existing IPS systems.

II. Background

In this project, we created an IPS system called MIPS in which activities of four services, SSH, phpMyAdmin, Joomla, and WordPress, are logged, monitored, and based on such data precautious measures are taken.

WordPress [5] is an open-source software with which users can establish their website. The owner of a WordPress website can perform multi-purpose activities with the help of a rich selection of plugins. To optimize the operation and management of WordPress, users are able to host the website using their computers. WordPress does not provide logs of failed user attempts requiring either third party plugins or logging through the web server. One such webserver is Apache, which has multiple ways to help the admin monitor successful and failed attempts to get information about suspicious activities.

Joomla [4] is also an open-source content management system with which users can create their website or application. Normal users can create, edit, or publish website content. Besides, Joomla offers an advanced feature for developers who want to elevate their websites. This is done via Joomla Framework written in PHP which offers a variety of libraries or packages. To record activities, the webserver of Joomla provides the error log file in order to record failed login attempts.

phpMyAdmin [3] is a free software tool that was created for users to administrate MySQL using the website. Besides, phpMyAdmin also supports operations over MariaDB. phpMyAdmin allows users to manage entities in the database such as tables, columns and indexes via a user interface. phpMyAdmin is very well-supported by books and communities that help to ease usage to a wide range of people. To track the activity, phpMyAdmin utilises system log via AUTH facility.

SSH [2], which stands for the Secure Shell or Secure Socket Shell, is a network protocol that provides security features such as transmitting encrypted data over a network and user authentication. SSH follows a client-server architecture where the Secure shell client application that displays a session interacts with an SSH server where the actual session runs. Essentially all Unix systems come with support for the ssh utilities from OpenSSH. Just like PhpMyAdmin, SSH also utilises the system log for tracking activity.

III. Methodology

In order for the IPS to be able to prevent attackers from making more than a certain number of log-in attempts, the IPS has to meet a couple of requirements. First, the IPS needs to be able to extract information from the services that are being tracked and protected by the IPS and convert this into a useable format. It then needs to be able to use this information to decide which IP address to ban and which to let through, after which the next step is to ban those that exceed the specified thresholds.

Figure.1 shows the detailed schematic of the system and out of which components it is made. As the figure shows, the information from the services is gathered by log-parsers and saved in a storage module. From here a module running separately from the parses uses the information stored, decides which IP address to ban and according to the configurations and calls the ban module. The relationships between each component are also showcased as a diagram in the figure.2 Additionally the IPS must be user friendly and to that end it contains as figure.1 shows a web interface consisting of a back end and a front end interface. This allows the user control over the process shown in figure.2 by means of changing parameters or even manually unbanning IP addresses.

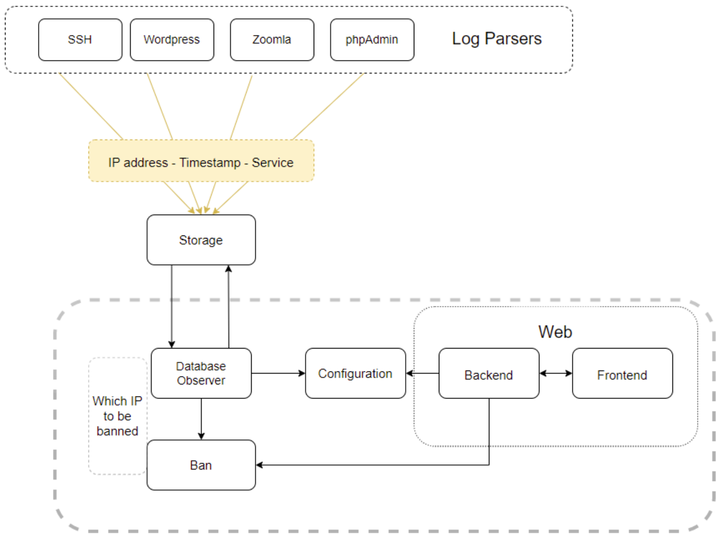

Fig. 1. Schematic of the IPS. Build out of log parses for each service that saves log in attempts as entries in the database. From which a database observer can then use the information to do calculations and call a ban module capable of banning certain IP addresses.

A. Service interface/Observer

In order to extract information from failed log-in attempts for the different services, a general service interface was developed. This is to have a common way for services to be handled while still allowing for the individual parsing to be done differently. This service interface utilises the python library watchdog [6], in order to monitor changes to the log files for each service, parse that information and store those as log record entries in the database.

B. Database

An important component of the project is the database. The database is used to store LogRecords. LogRecords contain the following information: the banned IP address, the service name that triggered the ban and the timestamp at ban became effective. Each time a failed login is detected, a LogRecord is created in the corresponding service interface. We decided to use SQLAlchemy library [7] for storing these logRecords. The database engine that runs under SQLAlchemy is SQLite 3 since the goal is to store banned IP addresses locally without compromising speed which SQLite 3 is good at. SQLAlchemy makes use of ORM which stands for Object-Relational Mapping and it makes it possible to interact with the underlying database in an object-oriented manner. Programming using the object-oriented paradigm enables the possibility of making the code responsible for database interaction more modular. This is achieved by creating classes for each table that needs to be created on the database. The tables inherit from a Base class which is used by SQLAlchemy to keep track of the tables and to handle queries to each table correctly since it

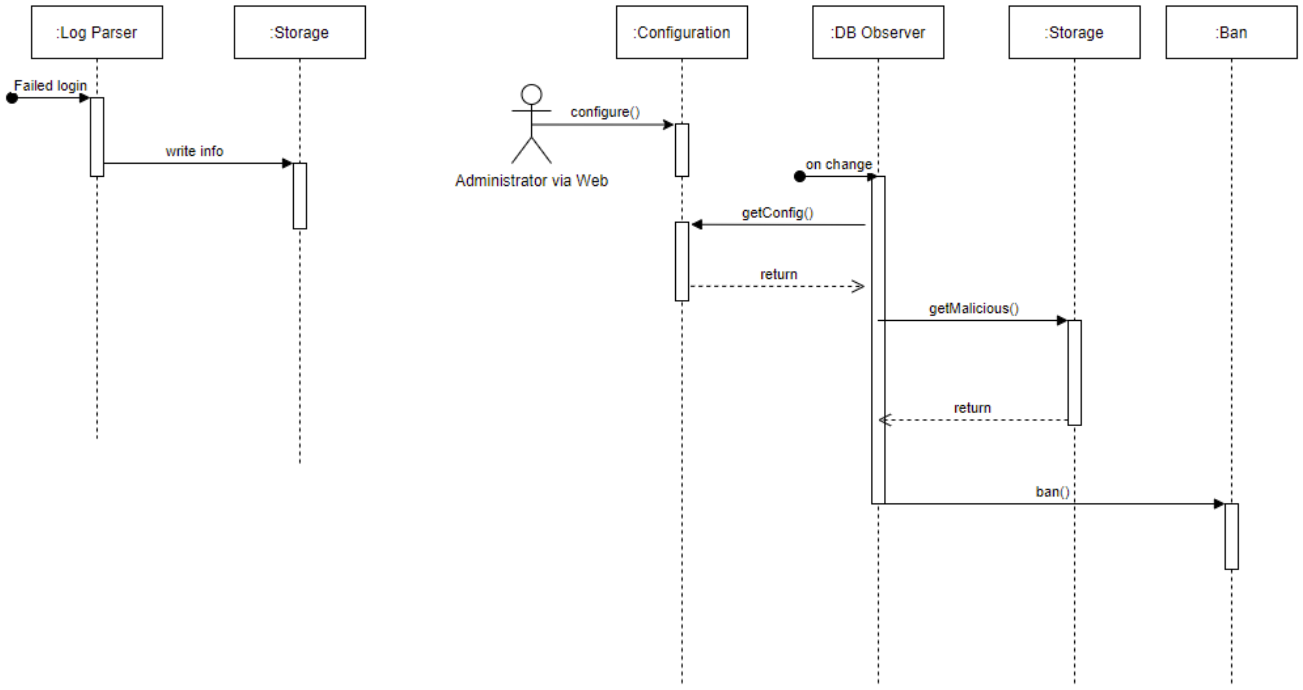

Fig. 2. The process that a failed log-in attempt goes through. Showcase the interactions of the different components and wherein the process they take place. While the diagram is sequential different parts of the program run in parallel. Meaning that multiple failed login can be handled according to the diagram at the same time.

then can derive information about the underlying structure of the tables and what is expected to be stored in each table. To communicate queries to the database, sessions are used. Sessions are responsible for all the communications with the database. The session only becomes stateful when objects need to be persisted or a connection to the database needs to be established. Each object that is added to the session is tracked and only when a transaction/query needs to be executed against the database, the session flushes the commits the changes to the database. Furthermore, SQLAlchemy uses lazy loading which makes it operate in an efficient way and consumes less memory.

C. *Parsers*

In order to retrieve information (IP address, timestamp, and Service) in one format from various log lines, we established parser files for each service in which they read the input of log lines and produce a group of information.

The location for where services log failed authentication can be manually changed for certain services but for this project a Ubuntu system was used with certain configuration. For the SSH service and phpMyAdmin, the logs are stored at ”/var/log/auth.log”, and the log files of Joomla and WordPress failed authentication are stored at ”/var/log/apache2/access.log”.

The common method that we used for extracting information from each log was to split them into components that automatically contained the IP address in one of the components in that resulted list. The only needed information left was timestamp which was indifferent formats between auth.log, other_vhosts_access.log, and error.php. At this stage, we implemented regex to recognize the pattern of each date-time format, extract the needed components (year, month, day, minute, and hour), and finally arrange them into a readable format for our system.

D. *DB observer*

The DB Observer is an integral part of the system. It is the part of the system that is responsible for connecting the correct information from the parsers and database to the actual component responsible for banning malicious actors. To achieve this, the DB Observer consists out of two parts: An Filtering process to find out which IP addresses need to be banned, and an observer part that notifies the DB Observer when to start this filtering process. See figure.3 for an overview of the observer.

For the observer part, the same python library watchdog [6] is used as in the service interface part to continuously observe the database file and in case of any changes to then steer the program towards the filter process.

After a change or rather a so-called event has taken place the DB observer can then start filtering the IP addresses

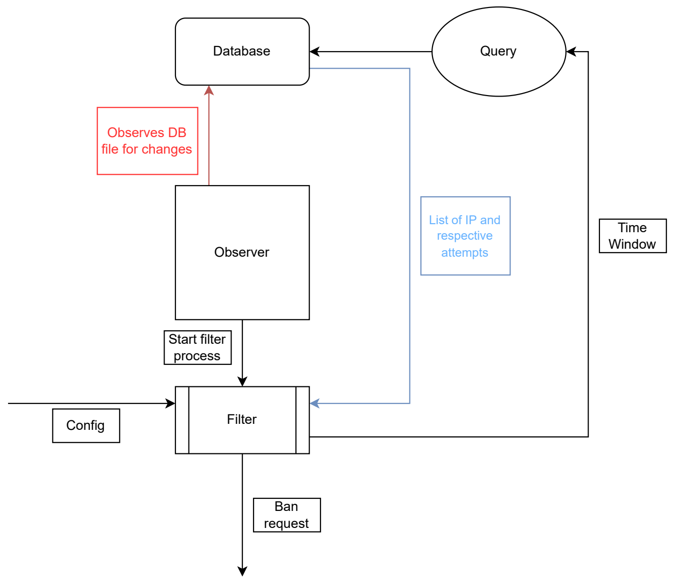

Fig. 3. Block diagram of the Database observer responsible for observing changes in the database and also filtering and counting the entries in order to find out which IP addresses to call a ban request for.

currently recorded in the database. In order to limit the number of database accesses, rather than storing the DB entries in the memory and doing the bulk of the filtering in the Observer instance, the observer instead sends a query to the database where it can make use of already built-in and optimized SQL functions. The main ones used in this case are the GROUP BY, which aggregates all attempts from the same IP together, the COUNT function which returns the numerical value of this grouping and also the FILTER function which is able to implement the aforementioned functions on only those entries within a certain time-frame.

Additionally, it is important for the filtering to be done according to set parameters. Hence why the observer imports the configuration file containing those and passes the time window information along with the query. After the query is done aggregating all the correct information in the database, it returns a list of tuples consisting of the IP address and number of accesses.

This list can then be checked against the threshold from the configuration file in the database observer. If this threshold is exceeded by any of the IP addresses, it will be banned for the configured time out, preventing the actor from accessing any of the services.

E. Banning

The banning module is designed with modularity in mind and can be easily swapped out by another implementation. For our implementation, banning and unbanning are handled by the ban_nft module which is based around NFTables. We chose to use NFTables framework for handling our banning. NFTables [8] is a network packet filtering framework created as part of the Netfilter project in 2013. It is designed as a modern replacement for iptables, ip6tables, arptables and ebtables unifying the utilities and taking a more efficient and maintainable approach. By using this framework instead of iptables we are able to make use of new features like sets and, at the same time, we allow our project to be more futureproof. The netfilter project provides the python3-nftables module, which is a python interface to their user space tool nft. We use this module to make configure the firewall, without having to call an external binary

Our implementation exposes methods for the initialization of the firewall, cleaning of the firewall, banning an IP for a given period, manually banning and unbanning an address and retrieving information about currently banned IP addresses.

A the start of the application is the firewall is configured. The application makes use of its own table, to which a chain and a blacklist set are added. The chain drops all packets at the INPUT hook when the source IP is in the blacklist. This means that only packets destined for an application on the machine are dropped, packets that would have been forwarded are still forwarded.

Whenever an IP address is banned it is registered and added to the blacklist set. The banning will also cause a Timer thread to start that waits for the duration of the ban. When the thread is done waiting, it checks if the perpetrator is still registered as banned and unbans its address. Unbanning is done by removing the address from the blacklist and removing the log entries in the database that caused the ban. This last part is done to ensure that an IP address that is unbanned manually, will not immediately be banned again the next time the database observer is triggered.

A list of currently banned addresses can be queried from the module. It provides information for each IP about what service the sender tried to access when the ban happened and for how long the IP is banned.

F. Back-end

The back-end is an essential component of the project. It is responsible for two important functionalities: bootstrap the core service of the system and handling communication with the front-end. The bootstrapping process sets up everything needed for the whole system to function collectively before any communication between the back-end and the front-end takes place. The back-end and the front-end communicate with each other by means of restful API calls. The API calls are handled using the Flask framework. The decision is made to use Flask instead of other frameworks so that the core and the server are written in the same programming language. This allows for easier adaptability between the core and the back-end parts since they are both coded using Python.

First, the back-end initializes the tables in the database that are needed for storing the banned IP addresses. Thereafter, the Flask app starts the core services and listens for requests from the front-end. There are six important endpoints that are implemented in the back-end: unbanning endpoint, change config endpoint, current config endpoint, toggle service endpoint, service status endpoint and an endpoint to fetch information regarding the currently banned addresses. The unbanning endpoint takes in a query string that contains the IP to unban. It delegates the unbanning functionality to the Unban/Ban module. Change config endpoint takes in three query string values: threshold, block time and time window. The threshold represents the maximum amount of requests that are allowed by an IP. The block time represents the time for an IP to be blocked. The value for the time window is used to check whether there are any IP adresses that reached the threshold in the time interval given by the time window. The current config endpoint is responsible for sending the current values of the config to the front-end module. The toggle service endpoint takes in service and a toggle boolean value in a query string. The service value represents the service to toggle and the toggle value is used as an indication of whether to stop or start the given service. Furthermore, the service status endpoint is responsible for fetching the information about the services and sending it to the front-end to show which services are running and which are not. Lastly, the back-end also contains an endpoint that is used by the front-end to gather the banned IP adress from the NFT module cache.

G. Front-end

The front-end is another important component of the project which is used for allowing the users to interact with the system through a web interface. The web interface has been created with the framework Vue.js, consisting of HTML, CSS and Javascript. This framework has been chosen as it allows for creating a single-page application and also allows for flexible changes to its structure [9]. Displayed through the front-end are the services that will be monitored which can be toggled on or off, the thresholds that can be altered and a list of banned IP addresses that can be unbanned.

IV. Experiment

To test the performance of our system we have conducted experiments on different parts of our system.

A. Response time

Because a quick response to threats is required to ensure the health of the server, the system should be able to respond quickly to malicious activities. To test this we came up with the following approach. An attacker tries to brute force the password of the PHPMyAdmin root user. He does this by sending requests repeatedly to the server. We measure the amount of requests the attacker can successfully make and the time it takes before the IP address is banned.

The attacker script was written in Python and made use of the requests library. Although python can be outperformed by other programming languages in terms of speed, it still allows for generating hundreds of login attempts per second. Additionally, python is a popular tool for writing exploit scripts. The script spawns 50 threads to send the requests. First each of them sends a get request to the site to find the hidden token. Afterwards, this token is used to make the login attempt. The threads will either print the timestamp of the response, or timeout after 1 second of no answer from the server. The difference in time between the 3rd request and the last request will be used as the response time.

B. Log file size:

For the second experiment, log files are used with a different number of lines. Here it is checked how long it takes from monitoring a change until it has successfully parsed the last line into the correct format. The motivation for this type of experiment is that the system is meant for users that want to monitor one or multiple services. This means that under certain conditions the log files are expected to contain a large number of lines. Therefore the system needs to be evaluated in terms of performance by looking at how it behaves under heavy load.

Based on the original service log files (namely auth.log, error.log and access.log), we produced different replica files with various sizes: *0, 250.000, 500.000, 750.000* and *1.000.000* log lines. Next, we designed a method to measure the elapsed time which started from the instant the system detected a new line was added to the simulation files. This is the moment when the system has to read a whole log file, where the size of the file can act as a bottleneck for the system causing a delay. The measurement ends when the last line of the log file is read and successfully parsed. This experiment was done for every one of the services and executed 5 times for each file and took the average result.

From this experiment, the initial expectation was that the run time for reading each simulated log file increases along with the size, where the system will still function properly rather than crash.

C. System responsiveness under load:

To test the responsiveness of the system when large amount of requests are issued, couple of experiments have been conducted. For the experiments, the Joomla and the Wordpress services are not used since the load on the system needs to be tested so which services are used to issue large amount of requests does not matter. For this reason, the services SSH and PhpMyAdmin are used for the experiments.

The system works by detecting changes made in the log files of each service. Each time failed login attempt is appended to it, a service handler is triggered in the system and adds the LogRecord information that can be derived from the log line into the database. To test the performance and the responsiveness of the system, DOS attack needed to be mimicked. This is done by creating two scripts that append forged log lines to the log files of the SSH and PhpMyAdmin services. This is done so that the system gets triggered and handles the logged lines. Each generated log line contains randomly generated IP addresses. The log lines are formatted in the right structure that the parsers in each of the service handler expect. This is done so to imitate how the messages are logged by the services themselves.

The experiment is conducted in two phases. The first phase is to generate forged log lines in SSH. In phase two, random log lines are also generated but now in both SSH and PhpMyAdmin log files at the same time. The first step that needs to be taken for conducting the experiments is to create functions that determine the time measurements of the experiment. This is done for fail2ban system and the MIPS system that we created. Fail2ban measurements are the base line measurements for this experiment. The config of both systems is the following: maxtry = three attempts; timewindow = five minutes; bantime = three minutes. The log files are flushed and the system is rebooted for the experiments. The attack speed of DOS is throttled down a bit to make it more similar to practical situations.

In the first phase, the total time it takes to ban a certain IP address by fail2ban is printed out. For the MIPS system, functions are added to the NFT module to measure how long it takes for the same address to be banned. Then in the next step the server is turned on and the scripts are run to append forged log lines. The scripts expect two arguments: the threshold of requests that can be derived from the config and how many log lines with IP addresses need to be generated. The threshold should be exactly the same as the number expected from the server config so that the database observer gets triggered for banning. The experiment is conducted on 1000 IP adresses and 5000 IP adresses. The results for phase one are shown in table IV.

In phase two, the same experiment as in phase one is conducted but now for both services (SSH and PhpMyAdmin) certain amount of log lines are appended to their corresponding log files simultaneously. The experiment is conducted on 2000 IP adresses and 10000 IP adresses, so that 1000 IP adresses are generated in each service log file and 5000 IP adresses are generated in each service log file. The results for phase one are shown in table V.

V. Results

A. Response time

The metrics are gathered over 10 runs for both Fail2Ban and our system. To ensure the validity of the test, the runs were alternated for the different systems. Both fail2ban and MIPS are configured to allow up to 3 requests, before banning the IP address. The response time for each run is given in table I and the overview is given in table II.

B. Log file size:

The data listed in table III represents reading large-sized log files for each service respectively that is used by the system. The results are in the order of milliseconds and are done over 5 trial runs.

VI. Discussion

A. Response time

The response time was on average 87.8 milliseconds and 226 milliseconds for MIPS and Fail2Ban respectively. Additionally the maximum response time measured during the test runs for MIPS was lower than the average response time for Fail2Ban.

Some circumstances may have influenced the test. First of all, what we measured was the moment the response came through on the attackers side. Depending on the round trip time of each request, the true response time could be higher or lower. It should also be noted that the time was measured using the time.time_ns method in python, possibly introducing slight measurement accuracy. Lastly, the timestamp of the last successful request was recorder, not the first unsuccessful request. Given that our script send more than 250 requests per second on average, this means that the response time could be around 2 ms higher.

B. Log file size

It can be observed that when feeding the system with large log files of various sizes, the system still functions properly and the result did not make any considerably difference. To be specific, when measuring the elapsed time of a log file in 5 trials, there were small fluctuations between measurements. The average values of those measurements were also witnessed at quite the similar values. The range of our obtained data was approximately around 2 - 5 milliseconds with some outliers such as 8.91 ms or 7.28 ms. One observation surprising to us was that although there was a significant different in sizes between log files with 0 lines (0 bytes) and log files with 1.000.000 lines (231 MB), the time differences were not very significant. This proved that fact that even when starting with very large files, the performance of our system was stable.

C. System responsiveness under load

As can be observed from the results, the MIPS system is faster than Fail2ban system in terms of banning IPs. However, MIPS cannot handle banning too many IP adresses in short period of time compared to Fail2ban. MIPS uses Nft tables for blocking IP addresses, but it cannot handle too many blocks in short period of times which results into a crash. Since Fail2ban does not use Nft tables, it does not crash when met with too many IP addresses but it does not block enormous amount of IP addresses in short period of time as shown in table V.

TABLE I
RESPONSE TIME PER RUN

	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10
MIPS	61.9 ms	54.4 ms	37.6 ms	142 ms	142 ms	43.1 ms	132 ms	135 ms	70.1 ms	60.2 ms
fail2ban	140 ms	44.1 ms	215 ms	126 ms	186 ms	371 ms	309 ms	406 ms	142 ms	323 ms

TABLE II
OVERVIEW OF RESPONSE TIME

	MIPS	Fail2Ban
Mean	87.8 ms	226 ms
Variance	1937 ms	14339 ms
Minimum	37.6 ms	44.1 ms
Maximum	142 ms	406 ms

TABLE III
RUN TIMES (MS) FOR PARSING EACH SERVICE LOG FILE SIZE PER RUN

Wordpress						
Log lines	Run 1	Run 2	Run 3	Run 4	Run 5	Average
0	3.36	3.12	2.79	2.92	3.11	3.06
250.000	3.64	3.01	3.06	4.16	3.06	3.40
500.000	4.21	3.04	3.60	4.43	3.18	3.69
750.000	2.93	4.12	2.91	3.23	3.29	3.30
1.000.000	4.25	3.87	3.01	4.53	3.20	3.77

Joomla						
Log lines	Run 1	Run 2	Run 3	Run 4	Run 5	Average
0	4.64	2.50	2.32	3.84	2.30	3.12
250.000	2.41	2.45	1.97	2.44	2.26	2.31
500.000	7.28	3.32	2.72	2.39	2.81	3.70
750.000	3.67	2.90	3.31	2.43	1.93	2.85
1.000.000	4.36	2.02	2.90	5.74	2.68	3.54

PhpMyAdmin						
Log lines	Run 1	Run 2	Run 3	Run 4	Run 5	Average
0	3.42	2.86	3.78	3.29	3.11	3.30
250.000	4.77	5.84	4.03	3.36	4.63	4.53
500.000	4.71	4.96	5.57	3.92	4.20	4.67
750.000	4.24	3.34	5.94	4.22	4.49	4.45
1.000.000	3.51	8.91	4.41	4.21	5.58	5.32

SSH						
Log lines	Run 1	Run 2	Run 3	Run 4	Run 5	Average
0	1.21	3.87	2.70	4.06	2.97	2.96
250.000	3.31	3.66	3.61	2.86	3.45	3.38
500.000	2.83	3.06	3.16	3.88	3.52	3.29
750.000	3.08	5.46	5.47	4.17	4.84	4.60
1.000.000	5.03	3.97	6.57	3.09	3.89	4.51

TABLE IV
TIME TAKEN (MS) FOR BANNING IPS ON SSH IN FAIL2BAN AND MIPS SYSTEMS FROM PHASE 1

IPs	Fail2Ban	MIPS
1000	30976	7435
5000	340785	-

TABLE V
TIME TAKEN FOR BANNING IPS ON SSH AND PHPMYADMIN SIMULTANEOUSLY IN FAIL2BAN AND MIPS SYSTEMS FROM PHASE 2

IPs	Fail2Ban	MIPS
2000	68276	34561
10000	-	-

VII. LIMITATION

One of the limitations of our system is the lack of support for detecting the log rotations. If the service application performs a log rotation, the intrusion detection system must be restarted. Furthermore, currently, it is not possible to support services that do not log their events into a text file but use another service e.g, Systemd for logging. Logging files of Systemd are binary files and can be accessed by the journalctl command to read.

While the system is capable of finding and banning malicious IP-addresses, it only does this for the specified applications. This introduces another limitation of the system and that is the application dependency. The IPS doesn't work on a system level rather the way malicious log in attempts are observed is by referring to the each specific application by means of the log files. This however does mean that even though the IPS is capable of protecting some of the applications on the system, others are still not protected.

Luckily it is possible to add more applications to the IPS. There are a couple difficulties in doing so however. First off, an application can only be added if it has a readable log file accessible to the owner of the system. Specifically readable means that the program is able to open the log-file and read the contents line by line, since an application might opt to save their log files in a format that can't be opened or read for security reasons. Also the attempted accesses must be noted down inside this log-file along the the IP-address in order to be useful. As for the second difficulty, another issue that exist is that the new application needs to be manually programmed into the IPS. Meaning, a new parser specific to it's logfile needs to be programmed in python, as well as the need to make other modifications in the code.

Another limitation of our system is that the simulation environment runs on our local area network. Due to constraints in router accessibility, we were unable to expose a server to the internet. Therefore we used an virtual machine image to share our setup and we ran the services only locally. This offered us some advantages during the making of the project, in terms of parallel workflow in different code version and less overhead, but also brought the considerable disadvantage that we could not test our system over the internet. This means that they were limited in the amount of machines we could run to test the stability of our system.

With what could be tested in the simulation environment, the system that was tested for could not handle failed authentication from too many IP addresses. This is due to the NFT kernel crashe within the interval of 2.000 to 5.000 IP addresses, causing the system to not function properly anymore.

VIII. Related Work

Fail2Ban [10] is one of the existing intrusion prevention software frameworks that prevents attacks such as brute-force attacks to a system. Basically, Fail2Ban will access activity log files in a system and detect which IP has many password failures. After a certain number of failed attempts, this software will ban that IP address by updating firewall rules to reject the IP address. As there are various types of log files from different plugins on the market, Fail2Ban can read multiple log files such as sshd, apache, or qmail. When using Fail2Ban, users can define the filter which is the pattern of log-in failtures, action which is commands executed at different moment, directory leading to the log file, and the maximum number of entries. This can be done in the fail.conf. Functioning based on log files, Fail2Ban can only start to work when there are data in those files, which is one of its the disadvantages and similar systems.

Another modern alternative for Fail2Ban is CrowdSec [11]. It is also an open-source software that allows users to detect and block suspiciously malicious activities from attackers. Similar to Fail2Ban, CrowdSec functions based on the information retrieved from log files which can be sshd, apache, or nginx. Users configure CrowdSec based on their needs in the file config.yaml in which they can provide access to database or set up the APIs. In recent years since the appearance of CrowdSec, it gradually gains its popularity as well as market share from Fail2Ban. This situation stems from the differences between two softwares. The first in the list of discrepancies was the language that was used to build 2 platforms. While Fail2Ban was created using Python, its competitor was written in Golang which provides a better speed, especially when scanning a large number of log files. With such advantage, more and more people are moving from Fail2Ban to CrowdSec.

The above solutions are two prominent software in the market. Besides, there are other alternatives including SSHGuard, IPBan, or Hookem-Banem, which provide a wide range of options for users.

IX. Conclusions

In this paper, we introduced our Intrusion Prevention System to detect malicious actions toward the administrative panel such as Joomla, Wordpress, or phpMyAdmin and to execute banning actions toward those suspiciously potential attackers. Our system allows users to configure the threshold, number of failed attempts in a certain amount of time and banning time, via a modern web interface. The result showed that even though the system has some limitations, it still remains functional under heavy load. Additionally, it performs better than Fail2Ban in terms of response time. We believe that the combination of modular approach and future proof technologies provide a viable system for small case real life environments.

As mentioned in the section on limitations, there are still a lot of areas with room for improvement, such as making the system compatible with more operating systems or making it easier to add more applications without having to reprogram the system. Our MIPS was way faster compared to other intrusion prevention systems regarding the banning of IP addresses. On the other hand fail2ban was more stable.

In the future more research can be done on alternatives to some of the components that the system is build out off. For example looking at alternatives to the ban method, or different storages apart from the sql-lite databases currently used. That way comparisons can be drawn in order to see if the system has room for further improvement with respect to efficiency and functionality. Thus each component could be looked at separately and investigated

References

[1] D. Stiawan, H. Abdullah, and Y. Idris, "Characterizing network intrusion prevention system," *International Journal of Computer Applications*, vol. 14, 01 2011.

[2] T. Ylonen, "SSH - secure login connections over the internet," in *6th USENIX Security Symposium (USENIX Security 96)*. San Jose, CA: USENIX Association, Jul. 1996. [Online]. Available: https://www.usenix.org/conference/6th-usenix-security-symposium/ssh-secure-login-connections-over-internet

[3] S. F. Conservancy, "Phpmyadmin," 2022. [Online]. Available: https://www.phpmyadmin.net/

[4] I. Open Source Matters, "Joomla," 2022. [Online]. Available: https://www.joomla.org/

[5] A. Inc., "Wordpress," 2022. [Online]. Available: https://wordpress.org/

[6] "watchdog," 2022. [Online]. Available: https://github.com/gorakhargosh/watchdog

[7] M. Bayer, "Sqlalchemy," in *The Architecture of Open Source Applications Volume II: Structure, Scale, and a Few More Fearless Hacks*, A. Brown and G. Wilson, Eds. aosabook.org, 2012. [Online]. Available: http://aosabook.org/en/sqlalchemy.html

[8] "Nftables," 2022. [Online]. Available: https://www.nftables.org/

[9] P. Pšenák and M. Tibensky, "The usage of vue js framework for web application creation," *Mesterséges intelligencia*, vol. 2, pp. 61–72, 01 2020.

[10] C. Jaquier, "Fail2ban," 2022. [Online]. Available: https://www.fail2ban.org/wiki/index.php/Main_Page

[11] "Crowdsec," 2022. [Online]. Available: https://crowdsec.net/