Modular Intrusion Prevention System

Rajendra Kumar Raghupatruni, Ravi Chandra Sadineni
Harshavardhan chowdary Ellanti, Nitish Garg, Anshul Anshul
Stony Brook University

Abstract

Intrusion Prevention Systems are part of the defense-in-depth strategy of computing systems. One popular Intrusion Prevention System is called *fail2ban*. *Fail2ban* monitors failed authentication attempts and temporarily blocks traffic from IP addresses that exceed a configurable number. e.g. fail2ban can block all traffic from IP address 1.2.3.4 if that IP address tried to log in more than 10 times in one minute. Our application is similar to fail2ban in a way it monitors auth.log, errors.log etc., to identify the ip/hosts which tries to re-login multiple times within a short span of time. Upon identifying those IP address which reaches the threshold retries during the configured time window, they are blocked for a configurable period of time thus reducing the likelihood of successful dictionary attacks. Moreover, this application can be configured to support any application(s) that captures the authentication failures to the log, thereby avoiding the burst login retries from a client.

1. Introduction

Intrusion Prevention Systems are security appliances that monitor systems for malicious activities. These systems identify malicious activities based on the log information, block the actions/traffic and report it. *Fail2ban* is one such application monitor that monitors the authentication attempts and temporarily blocks traffic from the corresponding clients based on the threshold number of retries during a particular time window. The current implementation is similar to fail2ban like monitor that monitors applications like SSH, phpMyadmin, Joomla and Wordpress. Modular Intrusion Prevention System(MIPS) monitors the authentication failure logs and keeps track of all such events from the client IP address. When the number of failure attempts reach a certain threshold for some configurable duration of time, the traffic from the corresponding IP is blocked at the server. Since blocking indefinitely will cause a complete Denial of Service for the client, the block is only done for a certain configurable period of time. All the configurable parameters can be initialized or modified from the admin console. The admin console also provides the flexibility to unblock the IPs of interest. So when a legitimate user fails to enter the correct password many times, he can still get it unblocked by consulting the administrator.

Though this application doesn’t completely eliminate all the security attacks like brute force attacks and dictionary attacks, it makes these attacks harder to perform and take very long periods of time in order to achieve the objective of the attacker. Moreover, the current implementation supports as many applications as possible based on a configuration file, thus eliminating the need to rewrite the whole implementation for new application panels.

Section 2 describes the design and implementation of an intrusion prevention system. Section 3 describes the evaluation which includes the experimental setup and how they are configured per application. Section 4 presents the results of these experiments along with their screenshots. Section 5 is the conclusion. Section 6 includes future work.

2. Design and Implementation

Intrusion Prevention System utility monitors the auth.log and error.log to identify failure login attempts from a client IP address. The implementation of this utility is based on the below identified tasks.

- The logs need to be parsed to identify a login failure.
- Upon threshold number of unsuccessful retries, the client IP should be blocked.
- The blocked IPs should appear in the admin console.
- The clients which are blocked should not be able to access the server for a configured period of time.
- The administrator should be able to remove/unblock IP addresses from the blocked list.
- The number of unsuccessful attempts, the time and blocking period should be configurable (only by the administrator).

Components:

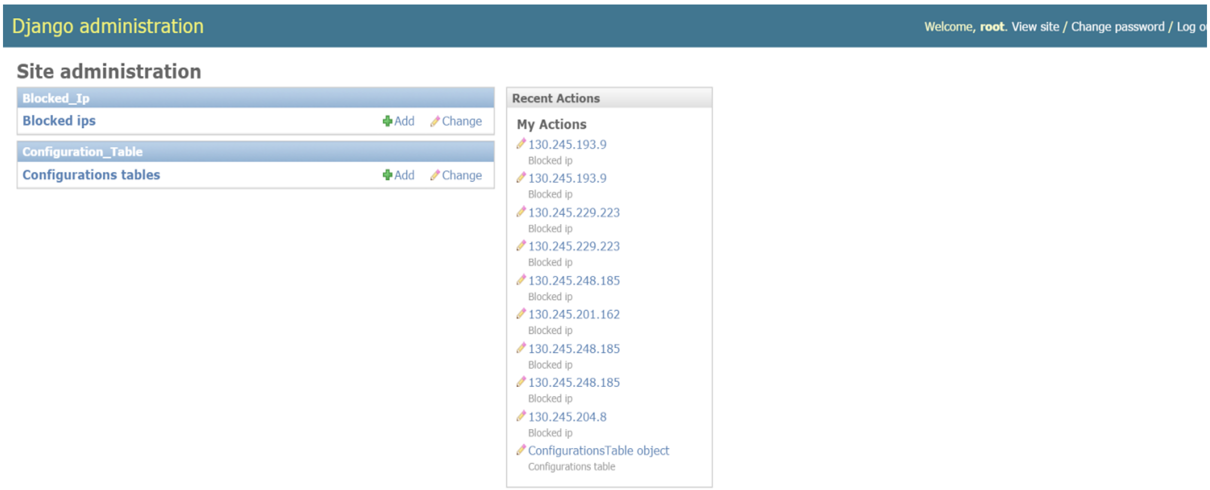
The major components (as indicated in Figure 3) of the system are the applications, log files, admin interface, Postgres database, system iptables and the central Modular Intrusion Prevention System.

- **Log Parser:** Every HTTP request is logged by the apache web server to */var/log/apache2/access.log.* We collected the pattern of URLs for the admin login and logout forms of Joomla, Wordpress and phpMyadmin. Whenever a client tries to login to any of the above mentioned applications, the hit count value is incremented for the IP address. And when the client logs out, all the entries pertaining to the client are removed from the database. However, this still leaves a possibility for a brute force attack by an attacker who has an access to an application and tries to gain access to any other application through brute force attack. For Eg: When an attacker has an access to phpMyadmin, he/she will have an active session and tries brute force approach to login to Joomla. Before threshold number of retries are reached for Joomla, he/she will try to logout from phpMyadmin which resets the counter to 0 for the attacker. Hence it will allow the attacker to continue his brute force attack without Denial of Service from the server.

  To address this possibility, we designed the application to use application name in conjunction to the client IP. Whenever the client tries to login (POST requests) to any of these applications, we increase the hit count of the login page for that particular application and IP combination. On reaching the threshold value which is set by the administrator, the traffic from the IP is blocked. If the client logs out of an application then we invalidate all the failures pertaining only to this application. Thus a subsequent failure attempt from the application and the IP combination is treated as a first attempt. And all the existing failure attempts from other applications are not invalidated. This eliminates the above mentioned brute force attack.

- **Admin Interface**: An admin interface provides an option to view and remove IP addresses from the blocked list. The administrator can also configure the threshold retries, time interval for which the requests are monitored, and the time interval for which an IP address has to be blocked. This user interface is built in python using Django framework. It communicates with PostgreSQL database which holds tables for storing the configuration and blocked IPs information. Details on the database schema are provided below. Note that user interface does not communicate directly with the

Intrusion Prevention System. It only communicates with the database. Thus both these applications run independently.



Django administration

Welcome, **root**. View site / Change password / Log o

Site administration

Blocked_Ip

Blocked ips ✚Add Change

Configuration_Table

Configurations tables ✚Add Change

Recent Actions

My Actions

130.245.193.9
Blocked ip

130.245.193.9
Blocked ip

130.245.229.223
Blocked ip

130.245.229.223
Blocked ip

130.245.248.185
Blocked ip

130.245.201.162
Blocked ip

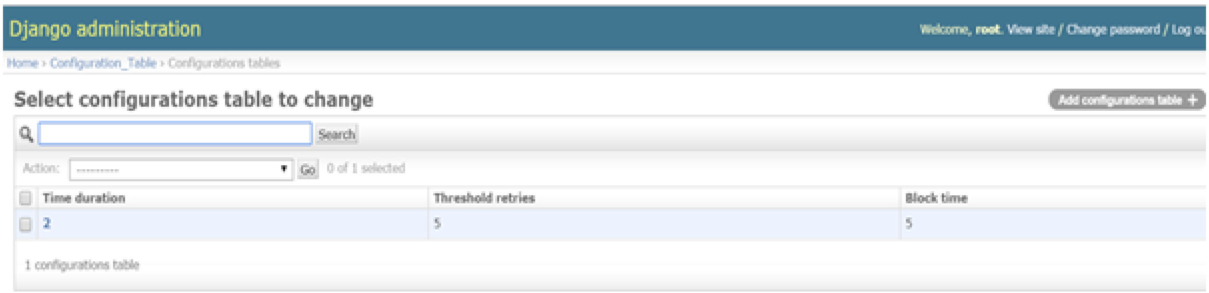
130.245.248.185
Blocked ip

130.245.248.185
Blocked ip

130.245.204.8
Blocked ip

ConfigurationsTable object
Configurations table

Figure 1: Admin Interface to manage the blocked client IP addresses and configuration parameters.



Django administration

Welcome, **root**. View site / Change password / Log ou

Home › Configuration_Table › Configurations tables

Select configurations table to change

Add configurations table ✚

Search

Action: --------- Go 0 of 1 selected

□	Time duration	Threshold retries	Block time
□	2	5	5

1 configurations table

Figure 2: Configuration parameters

- **Database:** Database acts as a persistent storage of the current state as well as a communication layer between the admin interface and the Intrusion Prevention System. There are three main tables, *blocked_ip*, *configuration_table*, and *ip_hits* used for the bookkeeping in the system. Admin interface communicates with the database using the Django ORM. Intrusion Prevention System uses an open source library, SQLAlchemy and a python driver, pyscopg2 for communicating with the PostgreSQL. A single application layer session is maintained and every write to the database is committed atomically to prevent database inconsistencies. Each of these tables are discussed below in detail.

  1. Configuration table

time_duration: int	threshold_retries: int	block_time: int

Table 1: Columns of *configuration_table* table

Configuration table holds all the fields that can be configured by the admin.

- *time_duration* represents the minutes prior to the current time for which the entries of *ip_hits* are considered while counting the retries.

- *threshold_retries* is the maximum number of acceptable retries. If there are more than *threshold_retries* in the last *time_duration* minutes, then the IP is added to the *blocked_ip* table.
- *block_time* indicates the number of minutes the traffic from this particular IP has to be blocked.

The reason for holding these values in the database instead of a configuration file is to allow the admin to edit these values dynamically. The application reads these value from the database whenever needed. This way, they are immediately reflected once the admin re-configures them.

2. Ip_hits table

client_ip: Char	hit_time : Date_time	appl_name : char

Table 2: Columns of *ip_hits* table

ip_hits table holds all the failed login attempts. On a failed login attempt to any of the above mentioned applications, the client IP and the current time and application name tuple is inserted into the table. The application then scans the table to see if there are *threshold_retries* number of entries from the same *IP* in the last *time_duration* minutes. Note that the *threshold_retries* and *time_duration* are read from the *configuration_table*. If there are more than threshold entries, then an entry is made into the *blocked_ip* table. Along with the entry into the *ip_hits* table, an entry is also made in the *System IP table*. *System IP table* acts like a firewall and blocks all the packets originating from that particular source. Once an entry is made into these two tables, *ip_hits* table is invalidated by deleting all the entries from this source. Also periodically this table is cleaned by removing all the old entries which are of no significance thereby maintaining the number of table entries small.

3. Blocked_ip table

client_ip : Char	block_start : Date Time	force_remove : boolean

Table 3: Columns of *blocked_ip* table

blocked_ip table holds the list of the *IP* addresses that are currently blocked along with the time at which they are blocked. The entry into this table is triggered by a failed login entry as discussed above. This table is periodically scanned for entries that are blocked for configured block time, upon the expiry of the block time period the *IP* is unblocked. All the entries are removed periodically from the table once the *IP* is unblocked. When removing the entry from *blocked_ip* table, it is also purged from the *System IP table* thus allowing the traffic from this particular source. Additionally a field, *force_remove* is provided for the admin to unblock *IP*'s of interest. If admin marks this field to *true* for a particular entry, then the entry is immediately removed irrespective of the time at which the entry is made. This way admin can re allow traffic from a blocked *IP*.

- **Intrusion Prevention System:** This is the main module of the system and is responsible to manage the flow of traffic in the system. It consists of work queue and various threads interacting with database, log files and with each other. The purpose of each of them is explained in workflow section below.

Workflow

The workflow (as indicated in Figure 4) consists of a work queue, multiple producer threads (one per application), a single consumer thread and a single unblocking thread. Each producer thread polls a log file for any event of failed login. On a failed login event, this thread adds the parsed IP address into a work queue*. Consumer thread dequeues the IP addresses from the queue and inserts those addresses into the *ip_hits* table. Also, at the same time it runs an SQL query that checks if there are more than threshold login attempts from the same IP address in the specified time interval. If there are threshold login attempts, they are removed from the table *ip_hits* and a single entry is made into the *blocked_ip* table as well as the *System IP table*. Note that an entry into the *System IP table* will actually block the incoming traffic from that address (like a firewall). The Admin console displays these *blocked_ip* table entries for managing the blocked IP addresses. The Unblock thread checks the *blocked_ip* table and *configuration_table* for any IP address that have been blocked for the *block_time* minutes and then removes the entries from both *blocked_ip* table and *System IP table*. And it is also responsible for allowing the traffic from the IP address.

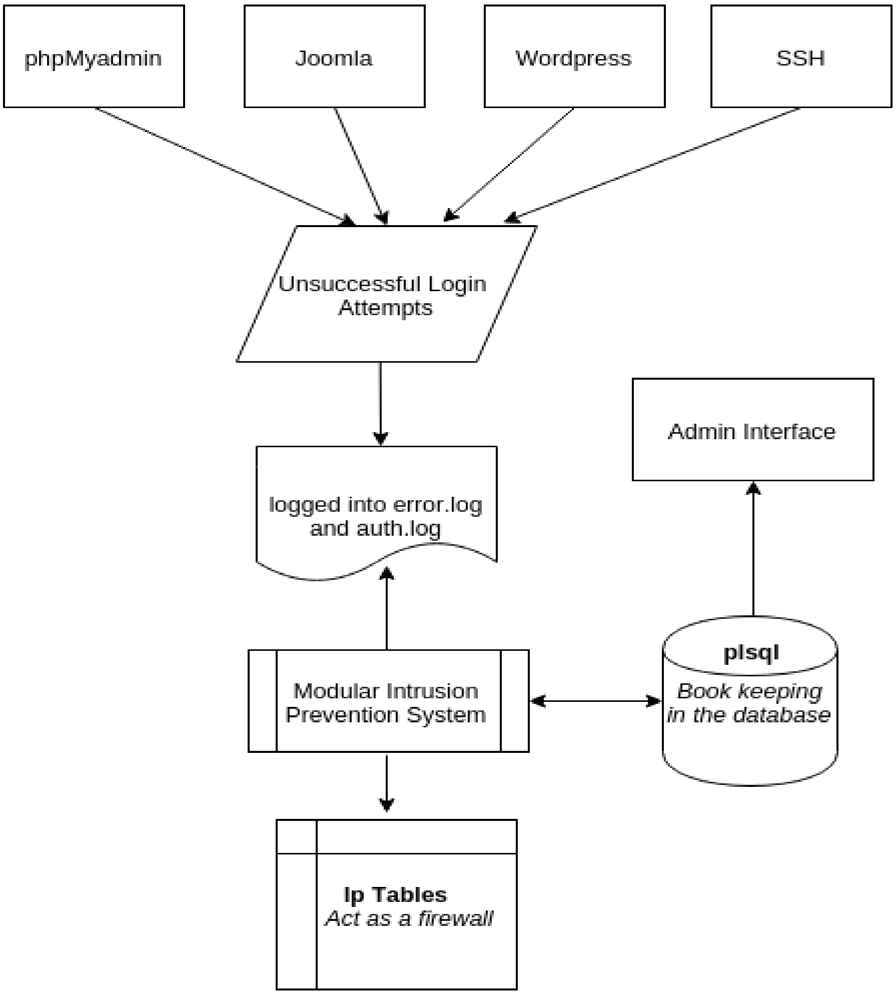

phpMyadmin
Joomla
Wordpress
SSH
Unsuccessful Login Attempts
Admin Interface
logged into error.log and auth.log
plsql
Book keeping in the database
Modular Intrusion Prevention System
Ip Tables
Act as a firewall

Figure 3: Block diagram of the Intrusion Prevention System

Note: *Queue in python is a synchronized class where the information can be exchanged safely among multiple threads without explicit locking mechanism.

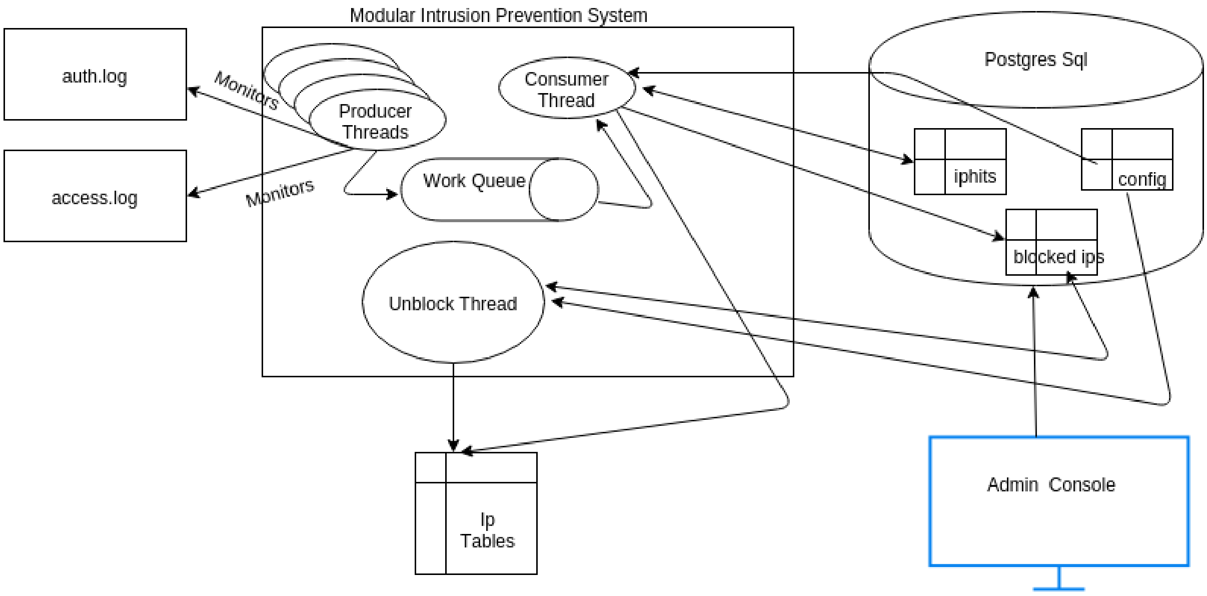


Figure 4: Modular Intrusion Prevention System

3. Evaluation

Experimental Setup

We conducted experiments on a Google Cloud node running on an Intel(R) Xeon(R) CPU @ 2.30GHz with 3GB main memory. The machine runs Linux kernel 3.16. We installed Python 2.7, Apache2, phpMyadmin, MySQL, Joomla and Wordpress on the machine. We used Django framework to build web interface for the administrator to update the threshold parameters. We installed PostgreSQL for bookkeeping and state management of the applications login failure attempts.

Every HTTP request is logged by the apache web server to */var/log/apache2/access.log*. We have collected the pattern of URLs for the admin login and logout forms of Joomla, Wordpress and phpMyadmin. When a client tries to login to any of these applications, we increase the hit count of the login page for that particular application and IP combination. On reaching the threshold value which is set by the administrator the IP is blocked. If the IP logged out of an application then we invalidate all the failure pertaining to the application. Thus a subsequent failure attempt from the same application and IP combination is treated as a first attempt. This eliminates the possible brute force attack by an attacker who has access to some other application on the same server.

Experiment 1: Joomla Admin Console Monitoring

Setup

Installation of Joomla can be done through the ***preInstall.sh*** utility which is a part of the source code bundle.

Below is the log format for login requests from the Joomla admin client.

```
130.245.230.194 - - [22/Nov/2015:20:43:06 +0000] "POST /joomla/administrator/index.php HTTP/1.1" 303
374 "http://104.196.46.6/joomla/administrator/index.php" "Mozilla/5.0 (X11; Linux x86_64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2490.80 Safari/537.36"
```

Login page for Joomla:

http://104.196.46.6/joomla/administrator/

Experiment 2: PhpMyAdmin Console Monitoring

Setup

Installation of phpMyadmin can be done through the ***preInstall.sh*** utility which is a part of the source code bundle. Below steps guides through the configuration of phpMyadmin

```
echo "Include /etc/phpmyadmin/apache.conf" >> /etc/apache2/apache.conf

sudo service apache2 restart
```

Below is the log format for login requests from the phpMyadmin admin client.

```
130.245.230.194 - - [22/Nov/2015:20:44:48 +0000] "POST /phpmyadmin/index.php HTTP/1.1" 302 632
"http://104.196.46.6/phpmyadmin/index.php?token=92de7bf121da11cd963f23108f239d49" "Mozilla/5.0
(X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2490.80 Safari/537.36"
```

Login page for phpMyAdmin:

http://104.196.46.6/phpmyadmin/

Experiment 3: Wordpress Admin Console Monitoring

Setup

Installation instructions are explained in detailed in the below table.

Download and unzip wordpress from here

Create a Database table for wordpress and a user with all the privileges to access and modify it.

Search for the file wp-config-sample.php and rename it to wp-config.php. Enter the details of Database connectivity and the user just created in the same file

Create a folder called wordpress in the /var/www/

Copy all the files in the unzipped folder to the folder just created

Now access the url *http://yoursite.com/wordpress/*

The actual blocking of the traffic is taken care by MIPS. Below is the log format for login requests from the Wordpress admin client.

```
130.245.230.194 - - [22/Nov/2015:20:45:58 +0000] "POST /wordpress/wp-login.php HTTP/1.1" 200 1609
"http://104.196.46.6/wordpress/wp-login.php" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/46.0.2490.80 Safari/537.36"
```

Login page for Wordpress:

http://104.196.46.6/wordpress/wp-login.php

Experiment 4: SSH Authentication Monitoring

Setup

Install the ssh server if there is not already installed on the server.

```
sudo apt-get install openssh-server
```

In-case of any ssh failed login attempts, the ssh-server logs the details to *auth.log* file. The above setup ensures the login failures are logged as below.

```
Nov 19 14:43:24 harp sshd[918]: Failed password for rraghupatrun from 104.196.46.6 port 59517 ssh2
```

Experiments:

We conducted experiments for functional testing of the implementation for all applications. As a part of this, we configured threshold retries as 5 for 2 mins time duration. Upon threshold retries, the client IP address should be blocked for 5 mins. On more than 5 failed login attempts, MIPS blocks the IP address thereby not allowing the client to reload or access the page for the next 5 minutes.

We also conducted experiments to unblock the IP address from the administrative panel by setting the *Force Remove* option to *True.* And the Intrusion Prevention System application successfully unblocked the IP address.

Along with the functional tests for individual applications, we ran the below tests for all the applications mentioned earlier. Current settings in the configuration table are as follows.

```
Time_Duration : 2 mins

Threshold_retries : 5

Block_time: 5 mins
```

Each of these parameters are explained above in Section 2 in detail. Below are the experimental tests we conducted to ensure the correctness and completion of the design and implementation. For all these tests, when an IP is blocked all the below properties hold true.

1. Any retry after 6th attempt should be blocked and the client should not receive any responses from the server.
2. Web server should not receive further requests from the client for the next 5 minutes.
3. Any attempt after the 6th attempt should have an entry in the *blocked_ip table* and an equivalent entry in the *System IP table.*
4. And the above entries should be deleted once the block time period expires.

Case 1: This test case makes sure that the normal functionality works properly. We made 7 consecutive attempts. The actual result matched exactly with the expected result.

Case 2: This test case makes sure that only entries in the time span of *time_duration* are considered. For an unsuccessful attempt once every 30 secs, there will only be 4 attempts in 2 mins which are not more than the threshold entries. Ideally, the IP should not be blocked. The behaviour of our implementation worked exactly as expected. We conducted this test over a time span of 10 mins.

Case 3: This test case makes sure that if a login retry is neglected at any span of time, it may still take part in the next time interval. In the first minute, 2 failure attempts are made. In the second minute another 2 failure attempts are made. And finally during the third minute, 3 attempts are made. After the 3rd attempt in the 3rd minute, the IP is successfully blocked.

4. Benchmark Results

We have conducted various experiments which are explained in Section 3 and the results are as expected. The current implementation doesn't add any overhead to the applications which are being monitored except from writing a one line log entry upon login failure. To test this, we measured the average response times by enabling logging and disabling logging of Wordpress, Joomla and phpMyadmin using Postman rest client plugin which indicates that the additional logging doesn't add a significant overhead. We disabled our MIPS application during this experiment.

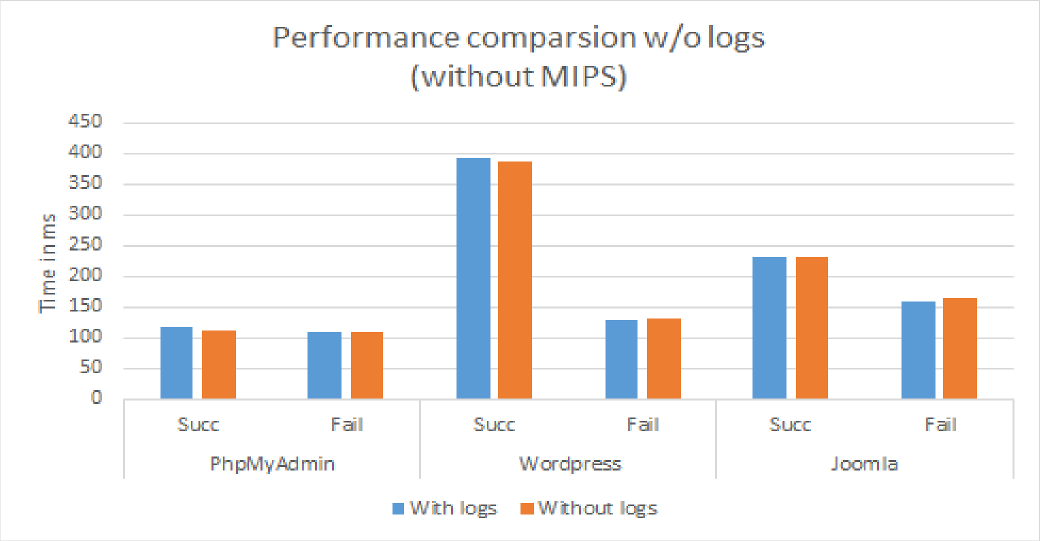


Figure 5: Performance comparison with and without logs and without MIPS.

The graph clearly shows that there is a very little overhead due to the additional logs in the failure and success scenarios. There is a maximum of 5ms overhead in both the successful and failure scenarios.

We also tested the performance of our MIPS to make sure that it is not creating any considerable performance degradation to the applications that are being monitored. The response times are measured with Postman rest client. These tests are done with MIPS enabled and disabled. The results are as shown in graph below.

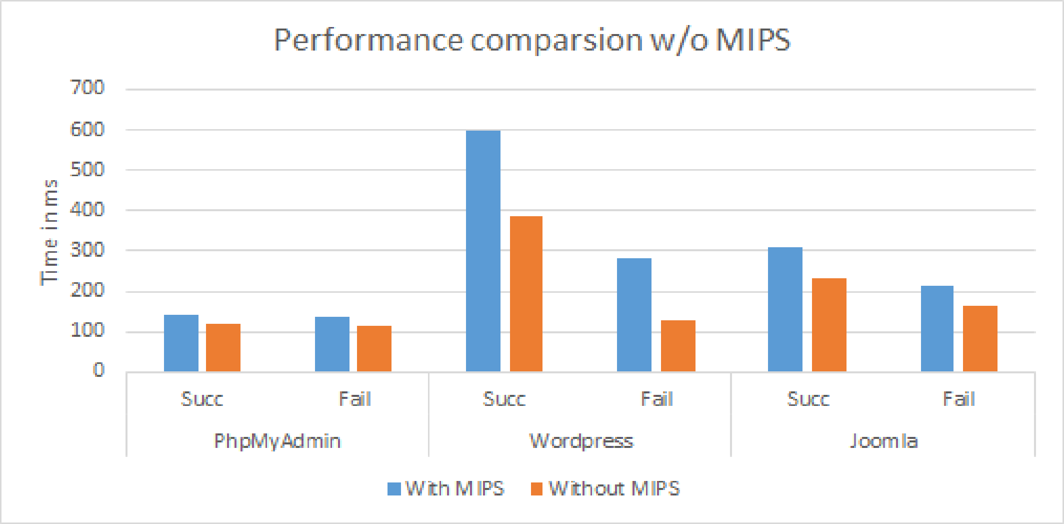
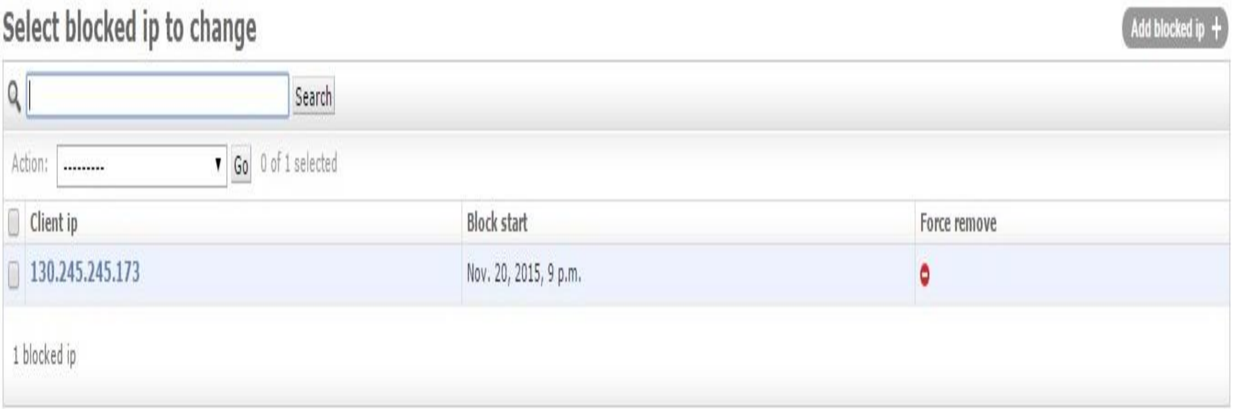


Figure 6: Performance comparison with and without MIPS.

The results clearly indicate that there is a little overhead with MIPS enabled. And even this small overhead is due to fact that our application is running on Google Cloud virtual machine with the system configuration (1vCPU, 3GB RAM) that is too less to handle the additional threads that constantly monitor logs, do the necessary updates in the backend tables and block IPs.

We also tested the feature *Force Remove* which is introduced to help the system administrators to unblock any IP addresses of interest. This gives the administrator an additional flexibility to manage the legitimate user login failure attempts. The screenshot below shows the typical blocked IP in the admin console. We can see the *Force Remove* option present in the third column is disabled by default.

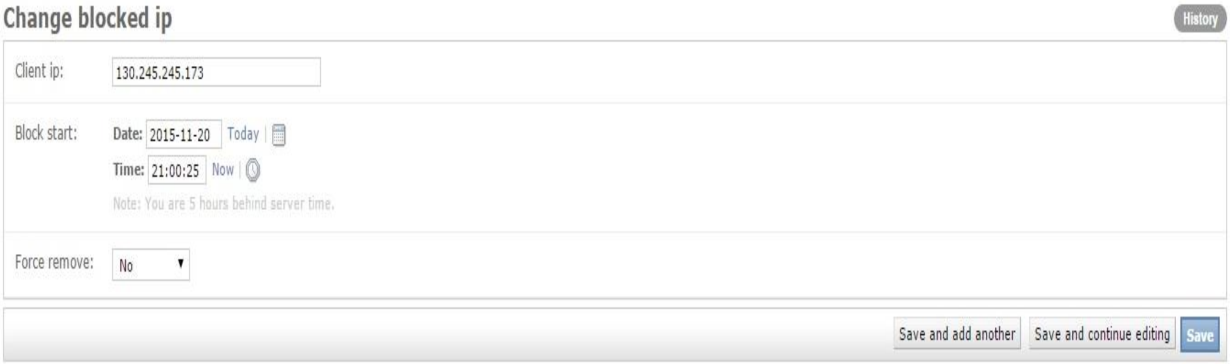


□ Client ip	Block start	Force remove
□ 130.245.245.173	Nov. 20, 2015, 9 p.m.	⊖

1 blocked ip

Figure 7: Force remove window.

If an administrator wants to remove the particular IP from the blocked list, then he have to select the corresponding IP and set the *Force Remove* option to *Yes*.



Change blocked ip

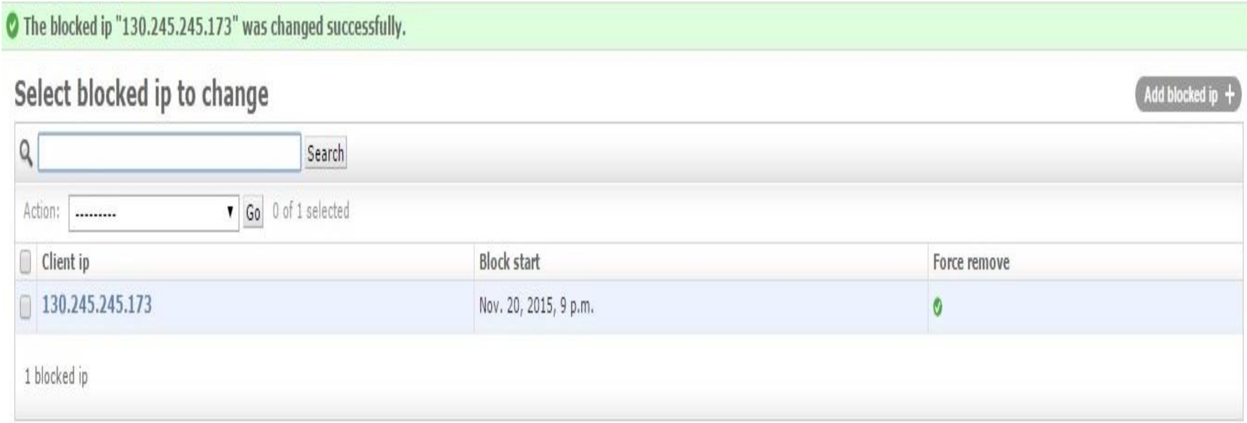
Client ip: 130.245.245.173

Block start: Date: 2015-11-20 Time: 21:00:25

Note: You are 5 hours behind server time.

Force remove: No

Figure 8: Administrator's view of the blocked IP



The blocked ip "130.245.245.173" was changed successfully.

Select blocked ip to change

Action: --------- 0 of 1 selected

Client ip	Block start	Force remove
130.245.245.173	Nov. 20, 2015, 9 p.m.	✓

1 blocked ip

Figure 9: Successfully force removed IP

The Unblock Thread which wakes up every 10 seconds will validate the *Force Remove* field for the IP and remove the particular IP from the system iptables thus unblocks the traffic for the IP address.

5. Conclusion

With the introduction of this monitor, we are able to reduce the likelihood of successful dictionary and brute force attacks. This monitor provides the flexibility to scale up to many web applications by just adding an entry to the configuration file and restart the application. Our application even provides an admin interface to customize/configure the threshold parameters like number of login retries, blocking time, time for which the number of retries to be monitored. It even provides a flexibility to unblock certain IPs of interest from the admin web interface. Moreover, our implementation doesn't add any extra overhead on the applications that are monitored. However, similar to *fail2ban* monitor our application doesn't prevent distributed brute-force attacks, and blocks the entire traffic from the client IP address when the conditions are met.

6. Future Work

There is scope of future work for introducing per application based configuration parameters like threshold retries for a duration, blocking time etc. Currently our implementation will block all the requests from the client IP address with maximum threshold retries. However, this application can be extended to block at a port number level granularity from an IP address to avoid the attack targeting a specific application.

Acknowledgements

We thank Professor Nick Nikiforakis for the guidance with the design and implementation suggestions for this project.

Contributions

Rajendra Kumar Ragupatruni: Designed and developed *IntrusionPreventionSystem.py (Multi-threading module). Deployed MySQL and Postgres.*

Nitish Garg: Deployed and modified code of Wordpress and Joomla to write logs in the required format.

Harshavardhan Chowdary Ellanti: Designed and developed *Admin Interface using Django Python Framework.*

Anshul: Wrote Log Parser and Deployed phpMyadmin and made code changes for the phpMyadmin to write logs in the required format.

Ravi Chandra Sadineni: Designed and developed *ConfigReader.py* and *IpTableManager.py* (ORM for Database Interaction for the Python module) using SQL Alchemy and pyscopg2.

We all have contributed towards the paper write-up, presentation and the demo video compilation.

References

- https://en.wikipedia.org/wiki/Intrusion_prevention_system
- https://en.wikipedia.org/wiki/Fail2ban
- Project Description: http://securitee.org/teaching/cse509/projects/project1.html
- Joomla Installation: http://www.liberiangeek.net/2014/09/install-joomla-ubuntu-14-04-apache2-mysql-php-support/
- PhpMyAdmin Installation: https://help.ubuntu.com/community/phpMyAdmin
- http://php.net/manual/en/function.error-log.php
- https://docs.python.org/2/library/queue.html
- Wordpress Installation: https://codex.wordpress.org/Installing_WordPress#Detailed_Instructions
- Django Framework : https://www.djangoproject.com/