**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, 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, those IPs 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) which captures the authentication failures to the log, thereby avoiding the burst login retries from a client.

## Introduction

Intrusion Prevention Systems are security appliances which monitors systems for malicious activities. The main functions of these systems are to identify malicious activities based on the log information and there by attempts to block the actions/traffic and report it [1]. fail2ban is one such application monitor which monitors the authentication attempts and temporarily blocks traffic from the corresponding clients based on threshold number of retries during a particular time window. The current implementation is similar to fail2ban like monitor which monitors applications like SSH, phpMyAdmin, Joomla and Wordpress. This application monitors the authentication failure logs and identifies it as a failure attempt and keeps track of all such events from a client IP address. Upon 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 to that web application, the block is done only for a certain configurable period of time. All the configurable parameters are initialized/modified from the admin console.  The admin console also provide a flexibility to unblock the IPs of interest. So when a user fails to enter the password many times, can still get unblocked upon consulting the administrator.

Even though this application doesn’t completely eliminate all the security attacks like brute force attacks and dictionary attacks. However, it makes these attacks even 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 to monitor 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.

## 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 the use cases and constraints.

**Use Cases:**

1. The logs need to be parsed to identify a login failure.
2. Upon threshold number of retries due to login failures, the client IP should be blocked.
3. The blocked IPs should appear at the admin console.
4. The Clients which are blocked should not be able to access the server for a certain period of time.
5. The administrator should be able to remove/unblock one or more IP address(s) from the blocked list.
6. The number of failed login attempts, the time and blocking period should be configurable and must be available only for the administrator.

**Components:**

The major components (Figure 1) of the system are the applications (like Joomla, wordpress, phpmyadmin), log files where the unsuccessful login attempts are logged, admin interface where the administrator can manage all the blocked IPs and modify configurable parameters, PostgreSQL database, system iptables  and the central modular intrusion prevention system that controls the communication between all the above components.

1. **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 to be in the blocked list. 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.

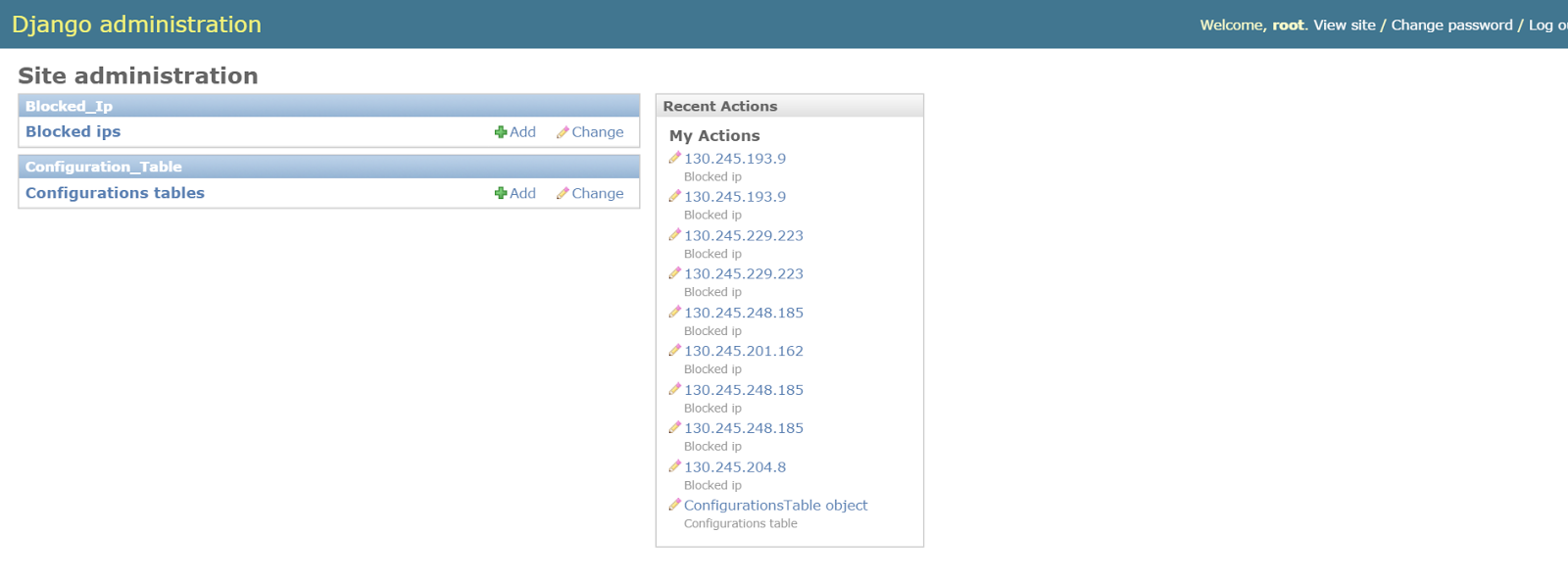


Figure 1: Admin Interface to manage the blocked client IP addresses and configuration parameters.



Figure 2: Configuration parameters

1. **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 a open source library, [SQLAlchemy](http://www.sqlalchemy.org/) and a python driver, [pyscopg2](http://initd.org/psycopg/) 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\_reties: 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\_reties* is the maximum number of acceptable retries. If there are more than this value in the last *time\_duration* minutes, then the ip is potential candidate for blocking. And 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.

Note that there is only one row in this table. The reason for holding these values in the database instead of a configuration file is to allow the admin to edit these values dynamically. These values are read from the database whenever needed by the application. They are not cached in the database. This way, they are immediately reflected once the admin re-configures them.

2.Ip\_hits:

|  |  |
| --- | --- |
| client\_ip: Char | hit\_time : Date\_time |

    Table 2: Columns of *ip\_hits* table

*ip\_hits* table holds all the failed login attempts. Upon a failure login attempt to any of the above mentioned applications, the client IP and the current time 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 :

|  |  |  |
| --- | --- | --- |
| 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* addressesthat 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 further allowing the traffic from this particular source.  Additionally a field, *force\_remove* is provided for the admin to manage or 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 allow traffic from a blocked *ip*.

Note that admin is **not** supposed to directly remove an entry from the *blocked\_ip* table. This is because every entry in this table has a equivalent entry in the *System IP table*. So, every deletion from this table should be followed by a deletion from *System IP table* or else the traffic from this *ip* would remain blocked forever.

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

sys-sec (2).png

Figure 3: Block diagram of the Intrusion Prevention System

**WorkFlow:** The workflow consists of a work queue, multiple producer threads (one per application), a single consumer thread and a single unblocking thread. The job of Producer threads is to parse the log files and add the new unsuccessful login attempts in a Work Queue. The producer threads polls the log files for any event of failed login. Upon a failed login event, this thread adds the parsed ip address into the queue\*. Consumer thread dequeues the ip addresses from the queue and inserts those addresses into the table *ip\_hits*. Also, at the same time it runs a sql query that checks if those ip addresses have made threshold login attempts within specified time interval and those addresses are removed from the table *ip\_hits* and inserted into the *blocked\_ip* table. The unblock thread takes the entries from the *blocked\_ip* table and inserts those addresses into *System IP table* to actually block the incoming traffic from that address. Also, this 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*. The Admin console displays these *blocked\_ip* table entries for managing the blocked ip addresses.

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

sys-sec3.png

Figure 4: Modular Intrusion Prevention System in detailed

## Evaluation

**Experimental Setup**

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

    Since Joomla, PhpMyAdmin, Wordpress don’t directly log the authentication issues to a log file, we need to explicitly modify the source code for these web applications to force write to the log files. Each of these are indicated clearly in the individual experiments.

**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. Since Joomla doesn’t log the authentication failures, below setup is required in-order to log the login failures from an IP address.

|  |
| --- |
| *cd <Intrusion Prevention Setup Directory>*  *cd joomla*  *cp php.ini /etc/php5/apache2/php.ini*  Note: Make sure the value of error\_log in the above configuration is present and has write permissions for the application to write logs to it.  *cp admin/controller.php /var/www/joomla/administrator/components/com\_login/controller.php*  Note: In-order to allow user logins also to be monitored, execute the below  *cp user/user.php /var/www/joomla/components/com\_users/controllers/user.php*  *sudo service apache2 restart* |

    The above setup ensures the login failures are logged as below.

|  |
| --- |
| *[18-Nov-2015 00:13:03 UTC] ERROR: joomla Login failed for the user: root, from 130.245.192.170* |

We conducted experiments to function test the implementation for joomla. As a part of this, we configured threshold retries as 5 for 2mins time duration. Upon threshold retries, the client IP address should be blocked for 5mins. And tried failure login attempts 5 times in the joomla administrative panel (mentioned in the below table). The application blocked 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.

|  |
| --- |
| <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. Since PhpMyAdmin doesn’t log the authentication failures, below setup is required in-order to log the login failures from an IP address.

|  |
| --- |
| *cd <Intrusion Prevention Setup Directory>*  *cd phpmyadmin*  *cp cookie.auth.lib.php /usr/share/phpmyadmin/libraries/auth/cookie.auth.lib.php*  Note: Make sure the value of error\_log in the above configuration is present and has write permissions for the application to write logs to it.  *sudo service apache2 restart* |

    The above setup ensures the login failures are logged as below.

|  |
| --- |
| *[18-Nov-2015 00:13:03 UTC] ERROR: phpMyAdmin Login failed for the user: root, from 130.245.192.170* |

We conducted experiments to function test the implementation for phpMyAdmin similar to *experiment 1*. As a part of this, we configured threshold retries as 5 for 2mins time duration. Upon threshold retries, the client IP address should be blocked for 5mins. And tried failure login attempts 5 times in the phpMyAdmin administrative panel (mentioned in the below table). The application blocked 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.

|  |
| --- |
| [http://104.196.46.6/phpmyadmin/](http://104.196.46.6/joomla/administrator/) |

**Experiment 3: Wordpress Admin Console Monitoring**

**Setup**

Installation instructions are explained in detailed in the below table.

|  |
| --- |
| Download and unzip wordpress from [here](http://wordpress.org/download/)  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/* |

Since Wordpress doesn’t log the authentication failures, we installed a third party plugin called *WP fail2ban.* Note that this plugin by default will neither manage nor make entries to the system iptables, though the name suggest the same.  This plugin will log the failed retries to the *auth.log* file. Steps to installing a new plugin are available from the below mentioned links.

|  |
| --- |
| <https://codex.wordpress.org/Managing_Plugins> |

The above setup ensures the login failures are logged as below.

|  |
| --- |
| *Nov 19 19:57:31 inst-1 wordpress(104.196.46.6)[9210]: Authentication failure for root from 130.245.223.98* |

We conducted experiments to function test the implementation for Wordpress, similar to *experiment 1*. As a part of this, we configured threshold retries as 5 for 2mins time duration. Upon threshold retries, the client IP address should be blocked for 5mins. And tried failure login attempts 5 times in the Wordpress administrative panel (mentioned in the below table). The application blocked 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.

|  |
| --- |
| <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* |

## We conducted experiments to function test the implementation for ssh, similar to *experiment 1*. As a part ot this, we configured threshold retries as 5 for 2mins time duration. Upon threshold retries, the client IP address should be blocked for 5mins. And tried failure login attempts 5 times. The ssh access got blocked from the client 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. And we were able to access the server from the ssh client.

**Experiments:**

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 fired 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.  An unsuccessful attempt once every 30 secs. Thus at any given point of time, there will only be 4 attempts in 2 mins which are not more than the threshold entries.  Thus ideally the ip should not be blocked. And the behaviour of our implementation worked exactly as expected. We have conducted this test over a time span of 10 mins.

Case 3: This test case makes sure that at any span of time if there are more than threshold entries in the given *time\_duration* the ip gets blocked. In the first second only 2 attempts were made. In the second and third second 3 attempts each were made. The 3rd attempt in the 3rd second is successfully blocked.

## Results

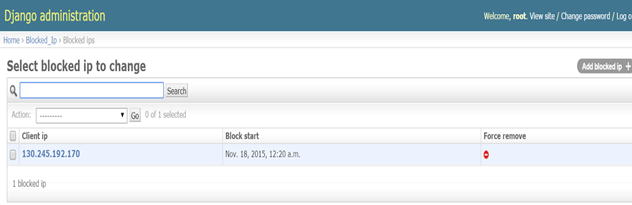


Figure 3: Blocked ip addresses on the admin console.

    The current implementation doesn’t add an overhead to the applications which are being monitored except from writing a one line log entry upon login failure.

## 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 which 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. There is no provision of blocking at the application port level granularity.

## 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. However, there is also a scope for fine grain blocking at the particular port numbers along with the IP address to avoid the traffic for a specific application.

## Acknowledgements

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

## Contributions

## 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>