

**A Project Report**

on

***Internet Wide Scanning and Building Intrusion  
Detection System Based on the Analysis of Honeypot  
logs***

by

**SARADHI RAMAKRISHNA**

2017H1030081H

M.E Computer Science

Under the supervision of

**PROFESSOR CHITTARANJAN HOTA**

**SUBMITTED IN FULFILMENT OF THE REQUIREMENTS OF  
BITS G540 RESEARCH PRACTICE**



**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI  
HYDERABAD CAMPUS  
(APRIL 2018)**

## **ACKNOWLEDGMENTS**

I owe my sincere gratitude to Prof. CHITTARANJAN HOTA, a distinguished faculty at BITS Pilani ,Hyderabad Campus for this project, for his constant guidance, invaluable suggestions and advice at every stage of the project.



**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI**  
**HYDERABAD CAMPUS**

**Certificate**

This is to certify that the project report entitled “Internet Wide Scanning and Building Intrusion Detection System Based on the Analysis of Honeypot logs” submitted by Mr SARADHI RAMAKRISHNA (ID No. 2017H1030081H) in partial fulfilment of the requirements of the course BITS G540, Research Practice, embodies the work done by him under my supervision and guidance.

**(CHITTARANJAN HOTA)**

BITS- Pilani, Hyderabad Campus

Date : 10-04-2018

# Contents

<b>1</b>	<b>INDO-DUTCH Project</b>	<b>1</b>
1.1	Introduction . . . . .	1
1.2	Tools . . . . .	1
1.2.1	Shodan . . . . .	1
1.2.2	Nmap . . . . .	2
1.2.3	ZMap . . . . .	2
1.2.4	Censys . . . . .	3
1.2.5	Google BigQuery . . . . .	3
<b>2</b>	<b>Analysis and Design of IDS using Honeypots</b>	<b>8</b>
2.1	Introduction of Tools . . . . .	8
2.1.1	ELK Stack . . . . .	8
2.1.2	Honeypots . . . . .	9
2.1.3	VirusTotal . . . . .	10
2.1.4	Github . . . . .	11
2.2	Installation of Tools . . . . .	11
2.2.1	Elastic Search . . . . .	11
2.2.2	Logstash . . . . .	12
2.2.3	Kibana . . . . .	12
2.2.4	Cowrie Honeypot . . . . .	13
2.3	Working Model . . . . .	14
2.4	Monitoring System . . . . .	16
2.4.1	Stochastic Process . . . . .	21
2.5	Intrusion Detection System . . . . .	22
2.5.1	Malicious-URLs Table . . . . .	24
2.5.2	Stats Table . . . . .	28
2.6	Conclusion . . . . .	43

# List of Figures

1.1	Google BigQuery Engine . . . . .	3
2.1	Elastic Search Output . . . . .	12
2.2	Kibana Dashboard . . . . .	13
2.3	Working Model . . . . .	15
2.4	Monitoring System . . . . .	16
2.5	Daily Attacks Count Area Graph . . . . .	19
2.6	Top 20 Usernames Histogram . . . . .	19
2.7	Unique Count of IP's . . . . .	20
2.8	Global Attacks Map . . . . .	21
2.9	Stochastic Process . . . . .	22
2.10	Intrusion Detection System . . . . .	23
2.11	IP Identified as NOT Malicious by IDS . . . . .	42
2.12	IP Identified as Malicious by IDS . . . . .	43

# List of Tables

1.1	List of Devices in Netherlands . . . . .	4
1.2	List of Devices in India . . . . .	6
1.3	List of Ports and their counts in Netherlands . . . . .	7
1.4	List of Ports and their counts in India . . . . .	7
2.1	List of Antivirus Engines . . . . .	11
2.2	List of Tables in PostgreSQL Database . . . . .	23
2.3	Schema of Malicious-URLs Table . . . . .	24
2.4	Schema of Stats Table . . . . .	28
2.5	DOS Attacks Clustering Table . . . . .	35
2.6	Modified Naive Bayes and K-Means Results . . . . .	40
2.7	List of DDos Binaries . . . . .	40
2.8	List of Malicious URLs . . . . .	40

# Listings

1.1	All open devices in India and Netherlands . . . . .	4
1.2	All open ports and their counts . . . . .	5
2.1	Cowrie Config File . . . . .	16
2.2	Get Malicious URLs From Virus Total . . . . .	24
2.3	Add URL's to PostgreSQL Database . . . . .	25
2.4	Submit URL's for Scan to VirusTotal . . . . .	25
2.5	Get URL Scan results from VirusTotal . . . . .	26
2.6	Virus Total API Request Sample . . . . .	27
2.7	Get Unique IPs from Elastic Search . . . . .	28
2.8	Elastic Search API Request Sample . . . . .	29
2.9	Get Commands Executed by IPs from Elastic Search . . . . .	29
2.10	Get Login Attempts by IPs from Elastic Search . . . . .	31
2.11	Generate Login Attempts in a CSV File . . . . .	32
2.12	Applying Spark K-Means Algorithm . . . . .	32
2.13	Counting of DOS attacks . . . . .	33
2.14	Modified Naive Bayes Algorithm Code . . . . .	35
2.15	K-Means Algorithm using Spark . . . . .	36
2.16	Classifying IPs . . . . .	37
2.17	Intrusion Detection System Code . . . . .	41

# Chapter 1

## INDO-DUTCH Project

### 1.1 Introduction

I was fortunate enough to be able to work on a budding project under the supervision of Professor Chittaranjan Hota from BITS PILANI and Professor Herbert Bos from VRIJE University Amsterdam. The Project was to determine the availability of all IoT devices on the internet and vulnerabilities of such devices in India and Netherlands.

### 1.2 Tools

There are many tools available for wider scanning of the internet and finding all the systems which are having specified open ports. Some of the tools are listed below.

#### 1.2.1 Shodan

Shodan is a search engine that lets the user find specific types of computers (webcams, routers, servers, etc.) connected to the internet using a variety of filters. Some have also described it as a search engine of service banners, which are metadata that the server sends back to the client. This can be information about the server software, what options the service supports, a welcome message or anything else that the client can find out before interacting with the server.

**<https://www.shodan.io/explore>**

Above is the link of the search engine where user can query and get the results. Unfortunately it allows only first 10,000 results to be fetched at a time. If a user wants more than that, they have to send a request to shodan team for getting access to more than 10K records.



Shodan gives the most of metadata using which we can identify what type of device it is and what are the open ports. Shodan collects data mostly on web servers (HTTP/HTTPS - port 80, 8080, 443, 8443), as well as FTP (port 21), SSH (port 22), Telnet (port 23), SNMP (port 161), IMAP (port 993), SIP (port 5060), and Real Time Streaming Protocol (RTSP, port 554). The latter can be used to access webcams and their video stream. It also has REST API support which can be easily integrated with other applications.

### 1.2.2 Nmap

Nmap (Network Mapper) is a security scanner, used to discover hosts and services on a computer network, thus building a "map" of the network. To accomplish its goal, Nmap sends specially crafted packets to the target host(s) and then analyzes the responses. The software provides a number of features for probing computer networks, including host discovery and service and operating-system detection. These features are extensible by scripts that provide more advanced service detection, vulnerability detection, and other features. Nmap can adapt to network conditions including latency and congestion during a scan.

Nmap features include:

- Host discovery
- Port scanning
- Version detection
- OS detection

### 1.2.3 ZMap

ZMap is a fast single packet network scanner designed for Internet-wide network surveys. On a computer with a gigabit connection, ZMap can scan the entire public IPv4 address space in under 45 minutes.

ZMap operates on GNU/Linux, Mac OS, and BSD. ZMap currently has fully implemented probe modules for TCP SYN scans, ICMP, DNS queries, UPnP, BACNET, and can send a large number of UDP probes.

ZMap also provides online platform for searching the IPV4 address space. It is called Censys. It is useful when we are having limitation on computational capability and network reachability.

## 1.2.4 Censys

Censys is a platform that helps information security practitioners discover, monitor, and analyze devices that are accessible from the Internet. It regularly probe every public IP address and popular domain names, curate and enrich the resulting data, and make it intelligible through an interactive search engine. Censys also provides API's using which any user can interact with their database. Online search engine link is given below.

<https://censys.io/>

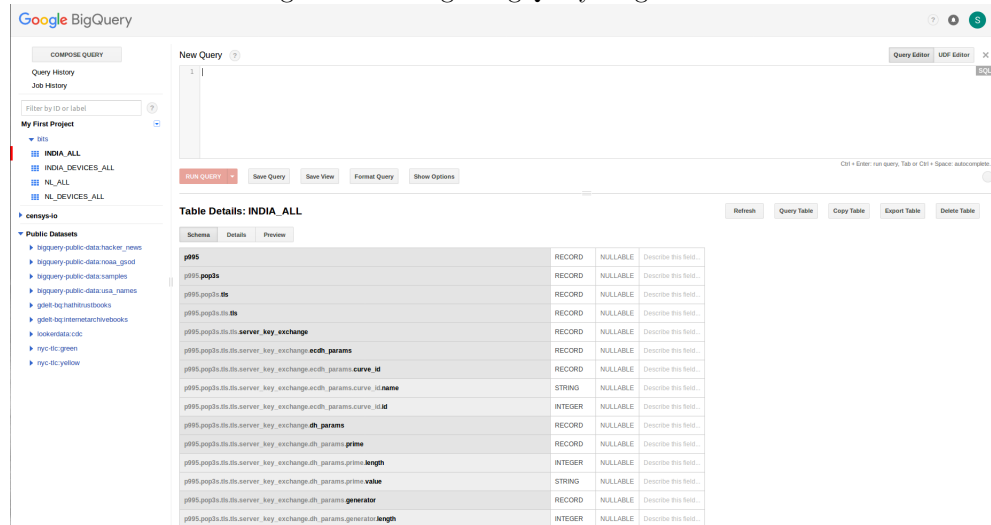
After carefully analyzing all the tools , their functionalities and their limitations, it has been decided to go with Censys Search engine provided by ZMap project team. But it has a limitation that a normal user can only get access to first 10,000 results of their database.

So, Mails are sent to Censys team and necessary privileges have been granted for us. Using the rest API's , we were able to access only first 10,000 results. So they provided access to their central database using Google BigQuery platform.

## 1.2.5 Google BigQuery

BigQuery is Google's serverless, highly scalable, low cost enterprise data warehouse designed to make all your data analysts productive. BigQuery enables us to analyze all our data by creating a logical data warehouse over managed, columnar storage as well as data from object storage, and spreadsheets.

Figure 1.1: Google BigQuery Engine



The screenshot shows the Google BigQuery web interface. On the left, there's a sidebar with 'Public Datasets' including 'INDIA\_ALL', 'INDIA\_DEVICES\_ALL', 'NL\_ALL', and 'NL\_DEVICES\_ALL'. The main area displays 'Table Details: INDIA\_ALL' with a list of tables and their schemas. The tables listed are:

Table Name	Schema	Record Count	Nullable	Description
pg95		RECORD	NULLABLE	Describe this field.
pg95.pg95		RECORD	NULLABLE	Describe this field.
pg95.pg95.ds		RECORD	NULLABLE	Describe this field.
pg95.pg95.ds.ds		RECORD	NULLABLE	Describe this field.
pg95.pg95.ds.ds.server_key_exchange		RECORD	NULLABLE	Describe this field.
pg95.pg95.ds.ds.server_key_exchange.ecdh_params		RECORD	NULLABLE	Describe this field.
pg95.pg95.ds.ds.server_key_exchange.ecdh_params.curve_id		RECORD	NULLABLE	Describe this field.
pg95.pg95.ds.ds.server_key_exchange.ecdh_params.curve_id.name		STRING	NULLABLE	Describe this field.
pg95.pg95.ds.ds.server_key_exchange.ecdh_params.curve_id.id		INTEGER	NULLABLE	Describe this field.
pg95.pg95.ds.ds.server_key_exchange.ecdh_params		RECORD	NULLABLE	Describe this field.
pg95.pg95.ds.ds.server_key_exchange.ecdh_params.prime		RECORD	NULLABLE	Describe this field.
pg95.pg95.ds.ds.server_key_exchange.ecdh_params.prime.length		INTEGER	NULLABLE	Describe this field.
pg95.pg95.ds.ds.server_key_exchange.ecdh_params.prime.value		STRING	NULLABLE	Describe this field.
pg95.pg95.ds.ds.server_key_exchange.ecdh_params.generator		RECORD	NULLABLE	Describe this field.
pg95.pg95.ds.ds.server_key_exchange.ecdh_params.generator.length		INTEGER	NULLABLE	Describe this field.

Google BigQuery engine looks like above. After we got access to their entire database, we started running queries to get the data of India and Netherlands entire database of all open devices and their ports.

India and Netherlands databases are huge in size comprising around 40 to 50 GB. We saved them as tables in Google BigQuery engine. Later we exported them to our local Bits Server machines in PostgreSQL databases for running our queries locally. Below are some of the statistics.

DEVICE-TYPE	COUNT OF DEVICES
nas (Network Access Storage)	73247
soho router	35951
IPMI	12039
infrastructure router	11315
network	7388
cable modem	3572
DSL/cable modem	3076
camera	2007
printer	1437
power distribution unit	1110
DSL modem	698
firewall	644
scada controller	531
alarm system	225
kvm	103
switch	83
set-top box	59
scada gateway	57
programmable logic controller	25
scada server	21
solar panel	15
hvac	14
DVR	12
laser printer	8
environment monitor	5
scada router	5

Table 1.1: List of Devices in Netherlands

There were a total of close to 1.6 million open devices in Netherlands which is a huge number and can be taken advantage of which can cause severe damage. Python code has been written for getting all the devices in India and Netherlands.

Listing 1.1: All open devices in India and Netherlands

```

1 import psycopg2
2
3 dbName = "censys"
4 table = "INDIA_DEVICES_ALL"
5 try:
6     conn = psycopg2.connect("dbname="+str(dbName)+" user='
7     postgres' host='localhost' password='postgres'")
8 except Exception as e:
9     print e
10 cur = conn.cursor()
11 cur.execute("select metadata_device_type,count(metadata_device_type
12 ) count from "+table+" group by metadata_device_type order by
13 count desc")
14 rows = cur.fetchall()
15 for row in rows:
16     portsString = "["
17     cur.execute("select ports from "+table+" where
18     metadata_device_type="+str(row[0])+" group by ports")
19     ports = cur.fetchall()
20     for port in ports:
21         portsString += str(port[0]) + ":"
22     portsString = portsString[0:len(portsString)-1] + "]"
23     print str(row[0])+" ,"+str(row[1])

```

The total number of ports and their counts is also found out by running the queries using python. It is given below.

Listing 1.2: All open ports and their counts

```

1 import psycopg2
2 from collections import Counter
3
4 dbName = "censys"
5 table = "NL_DEVICES_ALL"
6 device = "nas"
7 ports = []
8
9 try:
10     conn = psycopg2.connect("dbname="+str(dbName)+" user='
11     postgres' host='localhost' password='postgres'")
12 except Exception as e:
13     print e
14 cur = conn.cursor()
15 cur.execute("select ports from "+table+" ;")
16 rows = cur.fetchall()
17 for row in rows:
18     ports.append(row[0])
19
20 ports = Counter(ports)
21 for port in ports:
22     print str(port) + " , " + str(ports[port])

```

DEVICE-TYPE	COUNT OF DEVICES
network	135522
infrastructure router	43909
soho router	10268
camera	9491
DSL/cable modem	3356
nas	2537
IPMI	1135
printer	931
firewall	667
DSL modem	298
power distribution unit	242
DVR	182
kvm	50
environment monitor	24
solar panel	13
scada server	10
switch	9
scada router	9
cable modem	8
set-top box	8
wireless modem	8
scada controller	6
scada gateway	2

Table 1.2: List of Devices in India

There were close to 2 Million devices in India. Open ports and their counts are given below for both Netherlands and India.

There were in total of 35 Million devices in India and 67 Million devices in Netherlands. 2 tables were created in the PostgreSQL database for devices in India and Netherlands and queries were run against those tables. Below is the command for connecting to database "censys".

```
sudo -u postgres psql -d censys
```

After they are satisfied with our results, entire database has been handed to them for future work.

PORT	COUNT			PORT	COUNT
443	2670606			80	1925425
22	409455			25	262318
21	218763			143	192846
110	177016			995	174642
8080	171372			993	162578
53	134094			465	108951
587	45178			23	28813
8888	24090			445	5637
7547	2545			2323	2257
1911	931			1900	647
502	347			102	107
47808	75			20000	1

Table 1.3: List of Ports and their counts in Netherlands

PORT	COUNT			PORT	COUNT
443	1427660			80	683484
8080	263629			22	184335
53	169409			8888	160570
110	103431			21	98719
25	90601			143	84316
23	74346			995	59108
993	49866			465	39585
445	39177			587	9668
7547	7810			1900	2197
2323	1048			502	106
47808	11			102	7
1911	2			20000	1

Table 1.4: List of Ports and their counts in India

## Chapter 2

# Analysis and Design of IDS using Honeypots

### 2.1 Introduction of Tools

In this section, i will be introducing the tools ,languages and API's which will be used throughout the report.

#### 2.1.1 ELK Stack

”ELK” is the acronym for three open source projects: Elasticsearch, Logstash, and Kibana.

##### **Elastic Search**

Elasticsearch is a search engine based on Lucene. It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents.

##### **Logstash**

Logstash is an open source tool for collecting, parsing, and storing logs for future use.

##### **Kibana**

Kibana is an open source data visualization plugin for Elasticsearch. It provides visualization capabilities on top of the content indexed on an Elasticsearch cluster. Users can create bar, line and scatter plots, or pie charts and maps on top of large volumes of data.

### 2.1.2 Honeypots

In computer terminology, a honeypot is a trap set to detect, deflect or in some manner counteract attempts at unauthorized use of information systems.

In other words, A server that is configured to detect an intruder by mirroring a real production system. It appears as an ordinary server doing work, but all the data and transactions are phony. Located either in or outside the firewall, these are used to learn about an intruder's techniques as well as determine vulnerabilities in the real system."

Based on design criteria, honeypots can be classified as

1. Low-interaction honeypots
2. Medium-interaction honeypots
3. High-interaction honeypots

#### **Low-interaction honeypots**

Low-interaction honeypots simulate only the services frequently requested by attackers. Since they consume relatively few resources, multiple virtual machines can easily be hosted on one physical system, the virtual systems have a short response time, and less code is required, reducing the complexity of the security of the virtual systems.

Low-interaction honeypots present the hacker emulated services with a limited subset of the functionality they would expect from a server, with the intent of detecting sources of unauthorized activity. For example, the HTTP service on low-interaction honeypots would only support the commands needed to identify that a known exploit is being attempted.

#### **Medium-interaction honeypots**

Medium-interaction honeypots might more fully implement the HTTP protocol to emulate a well-known vendor's implementation, such as Apache. However, there are no implementations of a medium-interaction honeypots and for the purposes of this paper, the definition of low-interaction honeypots captures the functionality of medium-interaction honeypots in that they only provide partial implementation of services and do not allow typical, full interaction with the system as high-interaction honeypots.

#### **High-interaction honeypots**

High-interaction honeypots imitate the activities of the real systems that host a variety of services. It let the hacker interact with the system as they would



any regular operating system, with the goal of capturing the maximum amount of information on the attacker's techniques. Any command or application an end-user would expect to be installed is available and generally, there is little to no restriction placed on what the hacker can do once he/she comprises the system.

According to recent researches in high interaction honeypot technology, by employing virtual machines, multiple honeypots can be hosted on a single physical machine. Therefore, even if the honeypot is compromised, it can be restored more quickly. Although high interaction honeypots provide more security by being difficult to detect, but it has the main drawback that it is costly to maintain.

### **Cowrie Honeypot**

Cowrie is used for our research purposes. Cowrie is a medium interaction SSH and Telnet honeypot designed to log brute force attacks and the shell interaction performed by the attacker.

#### **Interesting Features of Cowrie Honeypot**

1. Fake filesystem with the ability to add/remove files. A full fake filesystem resembling a Debian 5.0 installation is included.
2. Possibility of adding fake file contents so the attacker can cat files such as /etc/passwd. Only minimal file contents are included.
3. Logging in JSON format for easy processing in log management solutions.

### **2.1.3 VirusTotal**

VirusTotal is a website created by the Spanish security company Hispasec Sistemas. Launched in June 2004, it was acquired by Google Inc. in September 2012. The company's ownership switched in January 2018 to Chronicle, a subsidiary of Alphabet Inc. (Google's parent company).

VirusTotal aggregates many antivirus products and online scan engines to check for viruses that the user's own antivirus may have missed, or to verify against any false positives. Files up to 256 MB can be uploaded to the website or sent via email. Anti-virus software vendors can receive copies of files that were flagged by other scans but passed by their own engine, to help improve their software and, by extension, VirusTotal's own capability.

AegisLab	Agnitum	AhnLab	Anity
Aladdin	Avast	AVG	Avira
BluePex	Baidu	BitDefender	Bkav
ByteHero	Quick Heal	CMC Antivirus	CYREN
ClamAV	Comodo	CrowdStrike	Doctor Web Ltd.
Emsisoft	Endgame	Eset Software	Fortinet
F-Prot	F-Secure	G Data	Hacksoft
Hauri	IKARUS	nProtect	Invincea
Jiangmin	K7AntiVirus	Kaspersky	Kingsoft
Malwarebytes	McAfee	Microsoft	eScan
Nano Security	Norman	Panda	Rising
Symantec	VIPRE	TotalDefense	TrendMicro

Table 2.1: List of Antivirus Engines

## List of Anti-Virus Engines used by VirusTotal

### 2.1.4 Github

GitHub (originally known as Logical Awesome LLC) is a web-based hosting service for version control using git. It is mostly used for computer code. It offers all of the distributed version control and source code management (SCM) functionality of Git as well as adding its own features. It provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project.

The code which is written for this project is entirely pushed to Github where it is bug free and version safe.

## 2.2 Installation of Tools

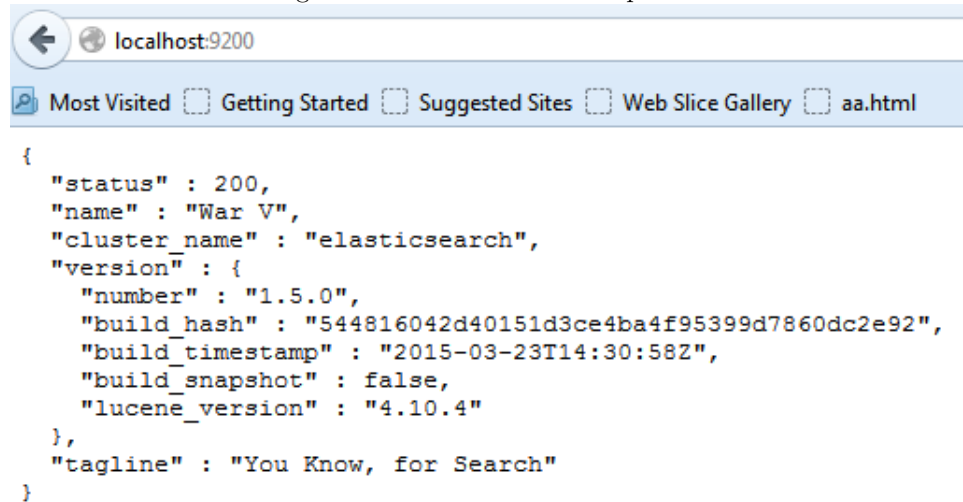
In this section, we will be mentioning about the installation of various tools which will be used in the project.

### 2.2.1 Elastic Search

1. Download the Debian Installation file from the below link
2. Install it using the below command  
*iotsys3@iotsys3-Precision-Tower-3420: dpkg -i elastic-search-6.2.0.deb*
3. Now start the service using below command  
*iotsys3@iotsys3-Precision-Tower-3420: sudo service elasticsearch start*

Open the link `http://localhost:9200` in the browser and check for the output which should look like below.

Figure 2.1: Elastic Search Output



We can check for the working of elastic search in the browser by typing the url `http://localhost:9200` and verifying the output.

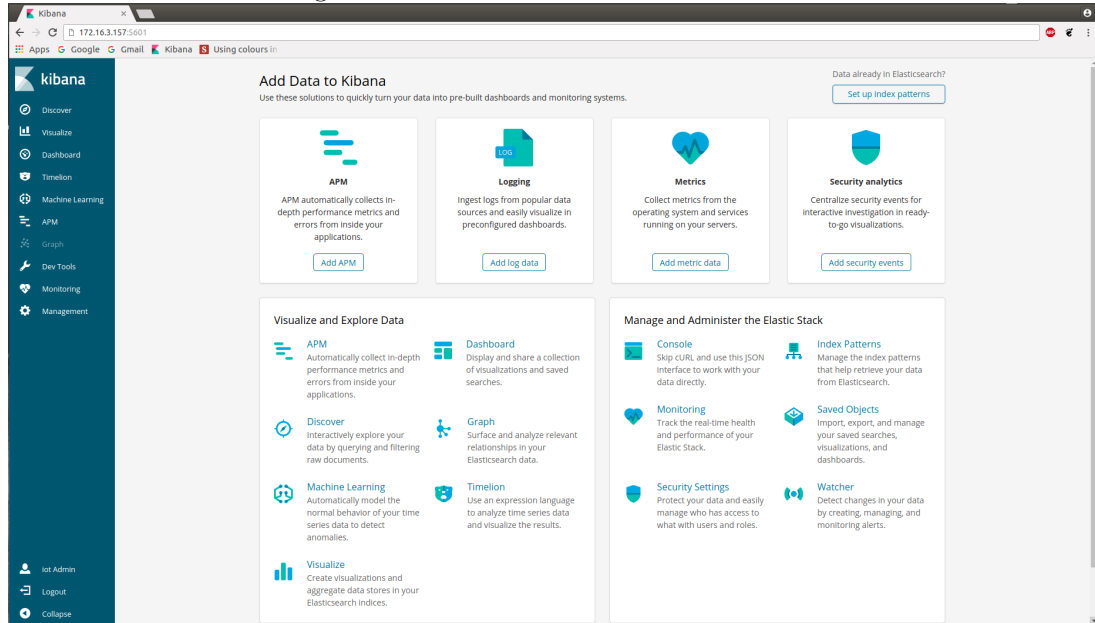
### 2.2.2 Logstash

1. Download the logstash debian installation file from below link  
`https://www.elastic.co/downloads/logstash`
2. Install it using the below command  
`iotsys3@iotsys3-Precision-Tower-3420: dpkg -i logstash-6.2.2.deb`
3. Now start the service using below command  
`iotsys3@iotsys3-Precision-Tower-3420: sudo service logstash start`

### 2.2.3 Kibana

1. Download the Kibana debian installation file from below link  
`https://www.elastic.co/downloads/kibana`
2. Install it using the below command  
`iotsys3@iotsys3-Precision-Tower-3420: dpkg -i kibana-6.2.2.deb`
3. Now start the service using below command  
`iotsys3@iotsys3-Precision-Tower-3420: sudo service kibana start`

Figure 2.2: Kibana Dashboard



Open the link (<http://localhost:5601>) in the browser and check for the Kibana Dashboard which looks like above.

## 2.2.4 Cowrie Honeypot

Cowrie honeypot is installed in a separate machine with public IP which is connected to the internet. It can be installed by following the steps given below.

### Install dependencies

```
iotics3@iotics3-Precision-Tower-3420: sudo apt-get install git python-virtualenv  
libssl-dev libffi-dev build-essential libpython-dev python2.7-minimal authbind
```

### Create a user account

```
iotics3@iotics3-Precision-Tower-3420: sudo adduser --disabled-password cowrie  
iotics3@iotics3-Precision-Tower-3420: sudo su - cowrie
```

### Checkout the code

```
iotics3@iotics3-Precision-Tower-3420: git clone http://github.com/micheloosterhof/cowrie  
iotics3@iotics3-Precision-Tower-3420: cd cowrie
```

## Setup Virtual Environment

```
iotsys3@iotsys3-Precision-Tower-3420: pwd  
iotsys3@iotsys3-Precision-Tower-3420: virtualenv cowrie-env
```

## Port redirection

```
iotsys3@iotsys3-Precision-Tower-3420: sudo iptables -t nat -A PREROUTING  
-p tcp -dport 22 -j REDIRECT --to-port 2222
```

In the same folder, create a copy of *cowrie.cfg.dist* and modify the file according to your needs like changing the name, SSH port number etc.

## Starting Cowrie

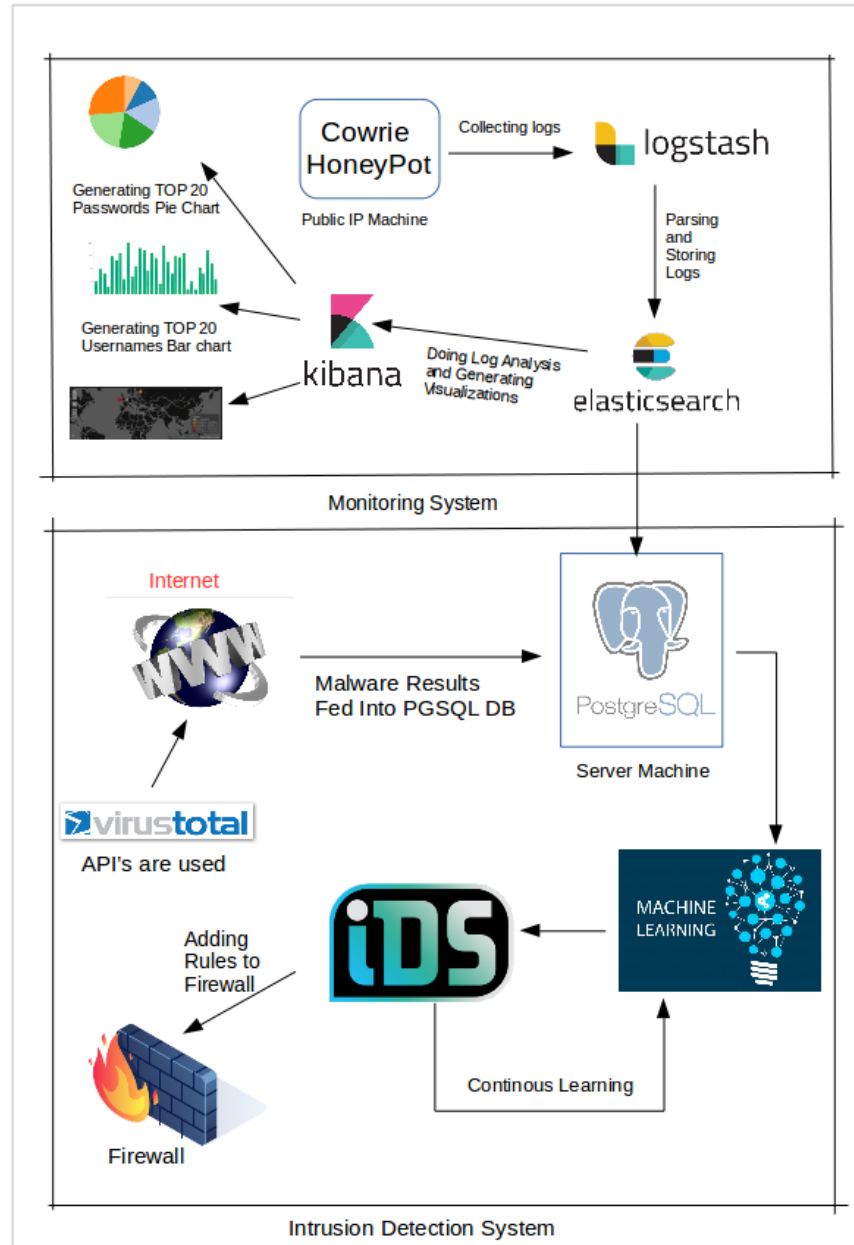
```
iotsys3@iotsys3-Precision-Tower-3420: bin/cowrie start
```

Cowrie will be up and running on port 22 which will monitor all SSH logins and commands executed by the attackers or intruders.

## 2.3 Working Model

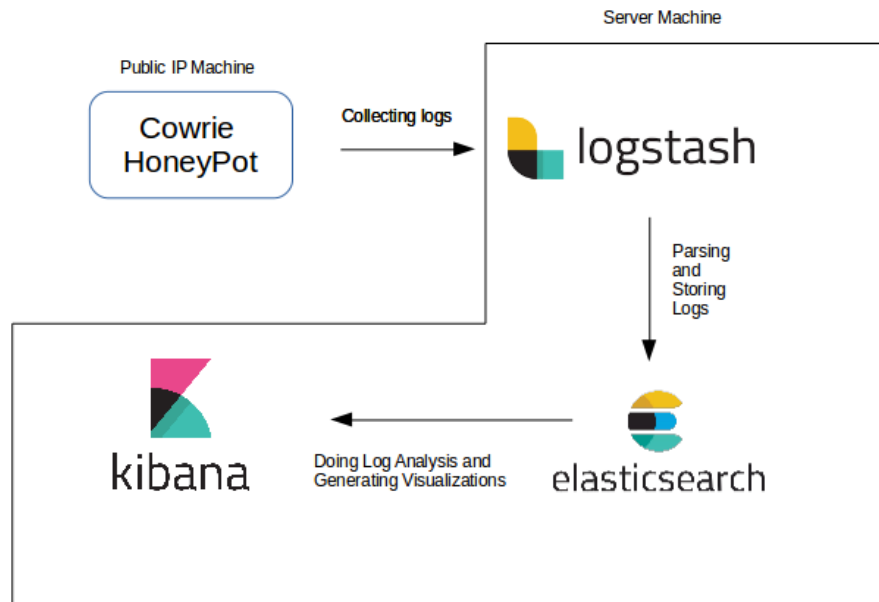
All the tools listed above are integrated to form a complete Working Model which looks like below.

Figure 2.3: Working Model



## 2.4 Monitoring System

Figure 2.4: Monitoring System



Cowrie will be generating the logs in the form of JSON format which will be available in public IP machine as specified in the figure-3. In order to stream the logs from the public IP machine to Server machine, filebeat can be used which is an open source tool. Unfortunately we did not get to use that tool as there was no physical connection between public IP machine and server machine due to security reasons.

So all the logs are copied from public IP machine to server machine on daily basis. These log files are supplied as input to Logstash which will parse, add metadata and push them to Elastic Search database. Logstash works on config files. Each config file has three parameters namely input, filter and output. Our config file is given below.

Listing 2.1: Cowrie Config File

```
1 input {
2   file {
3     path => "/home/iotsys3/Documents/Cowrie_Logs/*"
4     start_position => "beginning"
5     ignore_older => 0
6   }
}
```

```

7 }
8 filter {
9
10   json {
11     source => message
12   }
13
14   date {
15     match => [ "timestamp", "ISO8601" ]
16   }
17
18   if [src_ip] {
19
20     dns {
21       reverse => [ "src_host", "src_ip" ]
22       action => "append"
23     }
24
25     geoip {
26       source => "src_ip" # With the src_ip field
27       target => "geoip" # Add the geoip one
28       # Using the database we previously saved
29       database => "/home/iotsys3/Downloads/GeoLite2-
City_20180206/GeoLite2-City.mmdb"
30       add_field => [ "[geoip][coordinates]", "%{[geoip][
longitude]}" ]
31       add_field => [ "[geoip][coordinates]", "%{[geoip][
latitude]}" ]
32     }
33
34     # Get the ASN code as well
35     geoip {
36       source => "src_ip"
37       database => "/home/iotsys3/Downloads/GeoLite2-
ASN_20180206/GeoLite2-ASN.mmdb"
38     }
39
40     mutate {
41       convert => [ "[geoip][coordinates]", "float" ]
42     }
43   }
44 }
45 }
46 output {
47
48   elasticsearch {
49     hosts => ["localhost:9200"]
50     sniffing => true
51     manage_template => false
52     index => "logstash-cowrie-%{+YYYY.MM.dd}"
53     user => "elastic"
54     password => "63TxZjWQNkB4tArR7hri"
55   }
56
57   stdout {
58     codec => rubydebug
59   }

```



### 1. Input

It contains a parameter file which has options path, start-position and ignore-older. Path is where log files are present and start-position indicates from which point Logstash should start reading the logs from.

### 2. Filter

Filter is where entire processing happens in logstash. It has many options out of which json option is used as the logs are in JSON format. Some more information is added to the already present logs like country , Continent , latitude and longitude information which are specified in geoip option. Finally latitude and longitude are converted into float variables for plotting on map.

### 3. Output

It specifies options as to where the data should go after the processing is done. There are many options out of which we are using elastic search where data is stored in json format which will be in key value format.

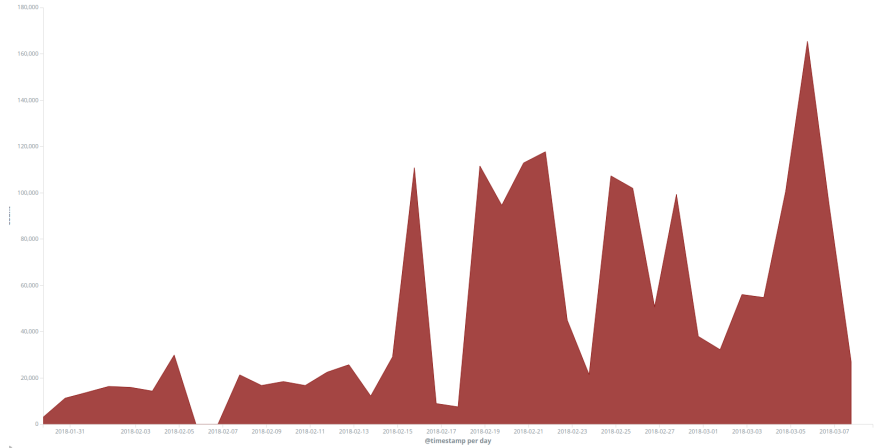
Data collected in the elastic search will be picked up by the Kibana by default as specified in the Kibana config files. Using the data visualizations are created for analyzing and monitoring the regular changes in data. Monitoring system was deployed for 40 days and collected nearly **2 Million Records**.

Using Kibana different visualizations are created which are listed below.

#### 1. Daily Attacks Count Area Graph

An area graph was plotted against the count of attacks on daily basis. X-axis represents the date and Y-axis represents the count of attacks. It can be observed that, attacks kept on increasing at the end.

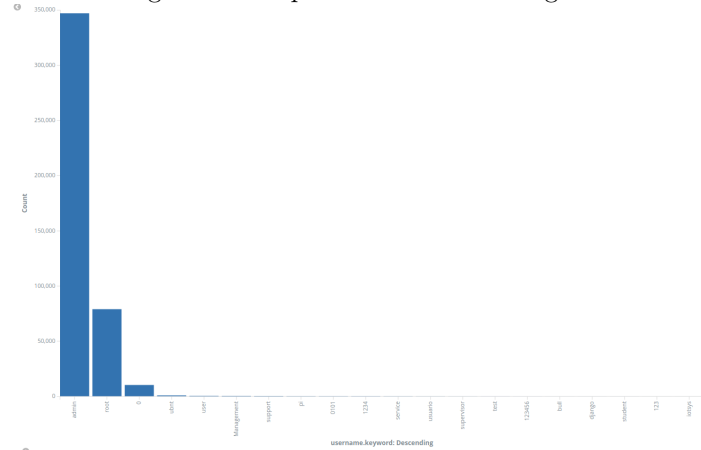
Figure 2.5: Daily Attacks Count Area Graph



## 2. Top 20 Usernames Histogram

A Histogram was used to represent the top 20 usernames that have been used by the attackers and intruders.

Figure 2.6: Top 20 Usernames Histogram

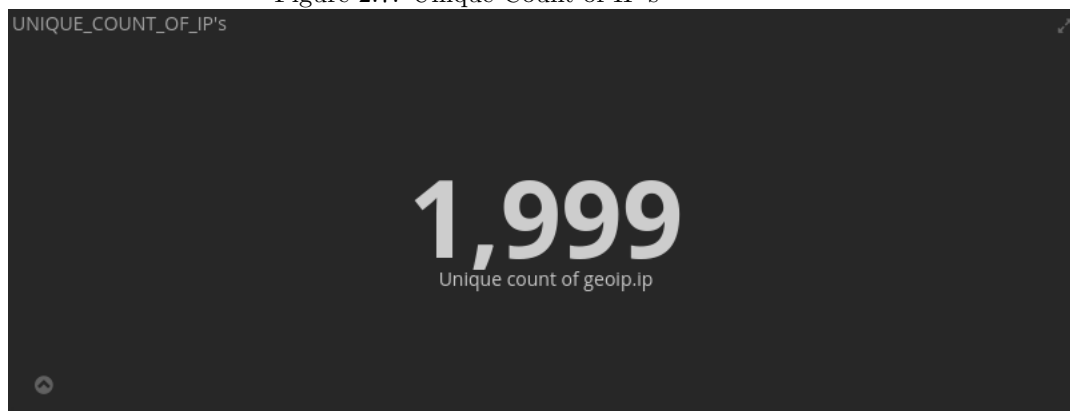


From the graph, it can be observed that usernames "admin", "root" and "ubnt" are mostly used by attackers.

## 3. Unique Count Of IP's

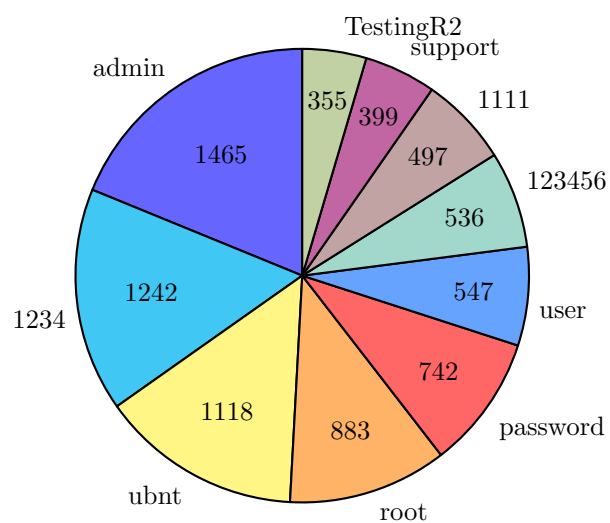
A Kibana Utility graph named Unique Count has been used for counting the Unique Number of IP's till now monitoring system has captured. It has found to be nearly 2000 IP's.

Figure 2.7: Unique Count of IP's



#### 4. Top 10 Passwords Pie

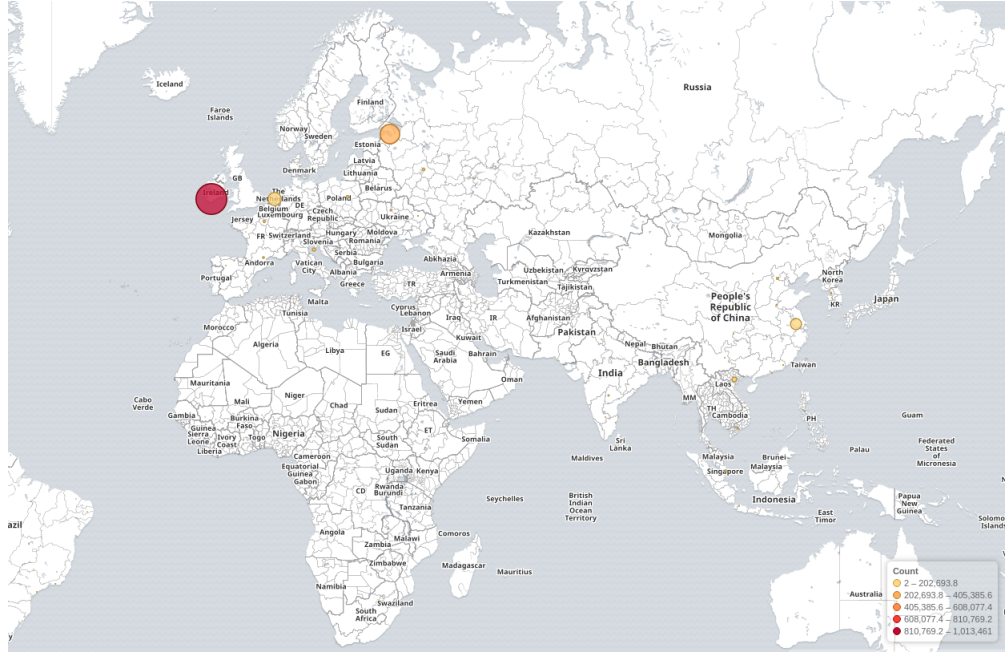
A pie chart has been drawn against the top 10 passwords used by the attackers and intruders. It can be observed that passwords "ubnt" and "admin" are mostly used.



#### 5. Global Attacks Map

A map is also plotted against the attacks receiving from different countries and continents using the latitude and longitude values. Most of the attacks are received from European continent and China country.

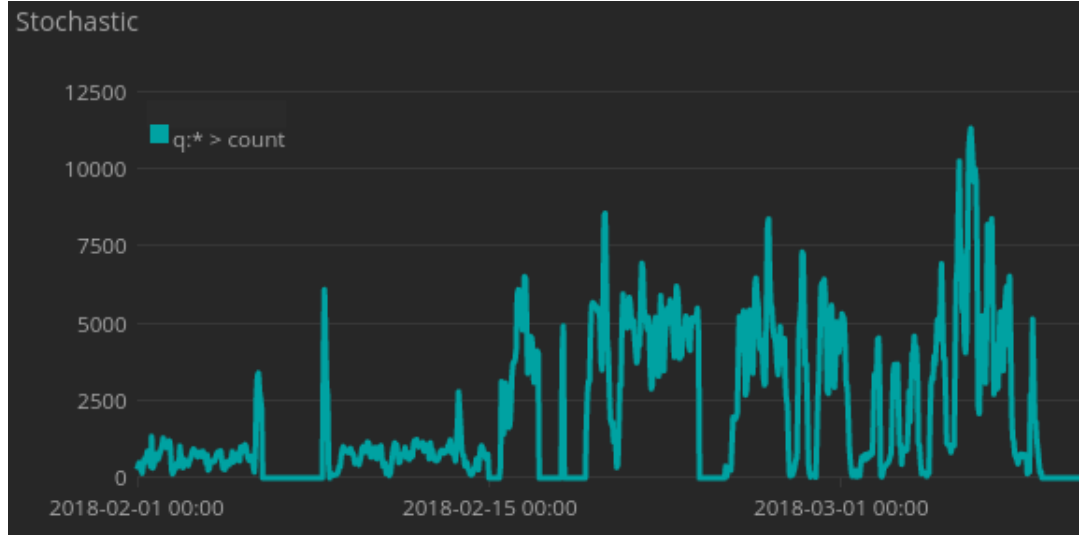
Figure 2.8: Global Attacks Map



### 2.4.1 Stochastic Process

Stochastic process is also called random process. It contains random variables with probabilities associated to it. While exact path cannot be inferred, assumptions can be made based on the probabilities of random variables. Attacks pattern observed is following random process or stochastic process as number of attacks on daily basis is not following any pattern.

Figure 2.9: Stochastic Process



## 2.5 Intrusion Detection System

An Intrusion Detection System is a software application which regularly monitors the network for malicious activities and reports them to central authority for fail-safe measures. After the monitoring system is setup and data is collected, task is now to design an Intelligent Intrusion Detection System which applies machine learning algorithms on the data and comes up with firewall rules to block malicious activities and IP's. For designing, tools like VirusTotal API's, PostgreSQL database, Spark K-Means Algorithm etc. are used which is given below in detail.

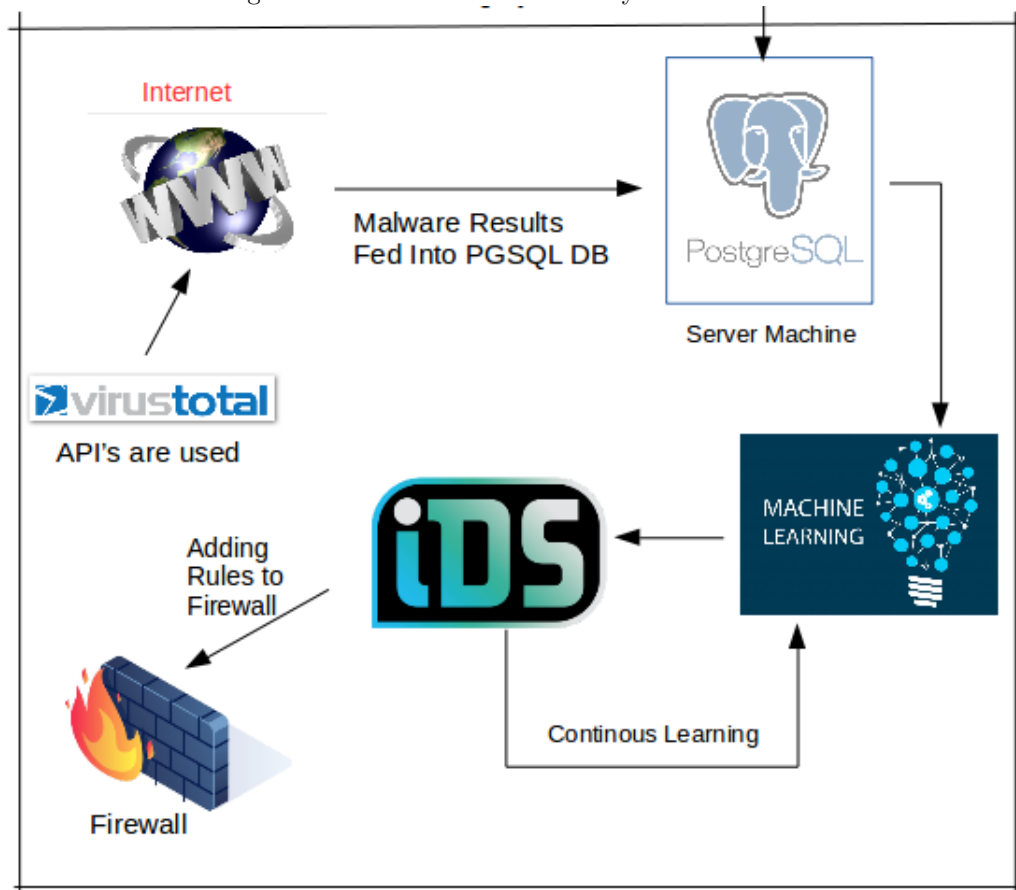
Virus Total is a website where details about different malwares and IP's can be fetched by using their web interface or REST API's. For our convenience, API's are used to integrate with our python programs. Virus Total gives a normal user on an average of 1000 API requests per day combining malware samples and IP's. We have sent a mail to VirusTotal requesting Research User privileges which has access to 10,000 API requests per day and all malware samples present in their database and got their approval.

In PostgreSQL, database named "cowrie" has been created and below tables were created to store the required data from Elastic Search and VirusTotal.

S.No.	Table Name	Schema
1	Dos-Attacks	public
2	Malicious-URLs	public
3	Stats	public

Table 2.2: List of Tables in PostgreSQL Database

Figure 2.10: Intrusion\_Detection\_System



### 2.5.1 Malicious-URLs Table

S.No.	Column Name	Column Datatype
1	url	text
2	scan-url	text
3	positives	integer
3	total	integer
3	identified-engines	text

Table 2.3: Schema of Malicious-URLs Table

Column "url" is used to store the malicious url and "scan-url" is the ID to identify the job submitted to VirusTotal for malicious url. "positives" contains the number of antivirus engines identified the url as malicious and "total" indicated the number of antivirus engines present in VirusTotal. "identified-engines" contains the names of all anti-virus engines. Malicious-urls table gets the data from VirusTotal API's. Python codes and their explanations are given below.

Listing 2.2: Get Malicious URLs From Virus Total

```
1 import requests
2 import json
3 import psycopg2
4
5 apikey = "
6     f0aa0772e518487aa5daf1dcf2be2d37c9b9b361b1a0d8339faf972210b36d41
7     "
8
9 scan_url = 'https://www.virustotal.com/vtapi/v2/ip-address/report?
10     apikey='+apikey+'&ip='
11 uniqueURLs = set()
12
13 dbName = "cowrie"
14 offset = 1813
15
16 try:
17     conn = psycopg2.connect("dbname="+str(dbName)+" user='
18     postgres' host='localhost' password='postgres'")
19 except Exception as e:
20     print e
21 cur = conn.cursor()
22
23 #cur.execute("CREATE TABLE IF NOT EXISTS MALICIOUS_URLS(URL TEXT)")
24 #cur.execute("TRUNCATE TABLE MALICIOUS_URLS")
25 #conn.commit()
26
27 cur.execute("SELECT IP FROM STATS OFFSET "+str(offset));
28 rows = cur.fetchall()
29 i = offset
30 for row in rows:
31     print "*****"
32     print i
33     i += 1
34     print row[0]
```

```

29 r = requests.get(str(scan_url)+str(row[0]))
30 resp = json.loads(r.content)
31 #print resp
32 if "detected_urls" in resp:
33     if resp["detected_urls"] != []:
34         #print resp["detected_urls"]
35         for url in resp["detected_urls"]:
36             print url["url"]
37             cur.execute("INSERT INTO MALICIOUS.URLS(URL) VALUES
('\" + url[\"url\"] + '\"')")
38             conn.commit()

```

Listing 2.3: Add URL's to PostgreSQL Database

```

1 import psycopg2
2
3 dbName = "cowrie"
4
5 try:
6     conn = psycopg2.connect("dbname="+str(dbName)+" user='
postgres' host='localhost' password='postgres'")
7 except Exception as e:
8     print e
9 cur = conn.cursor()
10
11 with open("urls.txt") as fp:
12     url = fp.readline()
13     while url:
14         print url
15         cur.execute("INSERT INTO MALICIOUS.URLS(URL) VALUES('\" + str
(url) + '\"')")
16         conn.commit()
17         url = fp.readline()

```

Listing 2.4: Submit URL's for Scan to VirusTotal

```

1 import json
2 import psycopg2
3 from subprocess import check_output
4
5 dbName = "cowrie"
6 apikey = "
f0aa0772e518487aa5daf1dcf2be2d37c9b9b361b1a0d8339faf972210b36d41
"
7 scan_url = 'https://www.virustotal.com/vtapi/v2/url/scan '
8
9 offset = 805
10
11 try:
12     conn = psycopg2.connect("dbname="+str(dbName)+" user='
postgres' host='localhost' password='postgres'")
13 except Exception as e:
14     print e
15 cur = conn.cursor()
16
17
18 cur.execute("SELECT URL FROM MALICIOUS.URLS ORDER BY URL OFFSET "+
str(offset))

```



```

19 urls = cur.fetchall()
20 i=offset
21 for url in urls:
22     print "*****"
23     print i
24     i += 1
25     res = json.loads(check_output(["curl", "--request", "POST",
26                                   "--url", "https://www.virustotal.
27                                   com/vtapi/v2/url/scan",
28                                   "--data", "apikey=" + apikey,
29                                   "--data", "url=" + url[0]]))
30     print res['scan_id']
31     cur.execute("UPDATE MALICIOUS_URLS SET SCAN_URL='"+str(res['
scan_id'])+"' WHERE URL='"+str(url[0])+"'")
32     conn.commit()

```

Listing 2.5: Get URL Scan results from VirusTotal

```

1 import requests
2 import psycpg2
3 from collections import Counter
4 import json
5
6 dbName = "cowrie"
7 apikey = "
f0aa0772e518487aa5daf1dcf2be2d37c9b9b361b1a0d8339faf972210b36d41
"
8 offset = 3
9
10
11 def getScanResults(scan_id):
12     print scan_id
13     softwares = ""
14     report_url = 'https://www.virustotal.com/vtapi/v2/url/report?'
15     \
16         'apikey=' + str(apikey) + '&' \
17         'resource=' + str(scan_id) + '&' \
18         'allinfo=false'
19
20     response = requests.get(report_url)
21     res = response.json()
22     #print json.dumps(res['scans'], indent=4, sort_keys=True)
23     for r in res['scans']:
24         if res['scans'][r]['detected'] == True:
25             softwares += str(r) + ";"
26     print softwares
27     softwares = softwares[0:len(softwares)-1]
28     positives = res['positives']
29     total = res['total']
30     cur.execute("UPDATE MALICIOUS_URLS SET "
31               "POSITIVES="+str(positives)+" , "
32               "TOTAL="+str(total)+" , "
33               "IDENTIFIED_ENGINES="+str(softwares)+" WHERE
SCAN_URL='"+scan_id+"'")
34     conn.commit()
35
36

```

```

37 try:
38     conn = psycopg2.connect("dbname='"+str(dbName)+"' user='
        postgres' host='localhost' password='postgres'")
39 except Exception as e:
40     print e
41 cur = conn.cursor()
42
43 cur.execute("SELECT SCAN.URL FROM MALICIOUS.URLS OFFSET "+str(
        offset))
44 scan_ids = cur.fetchall()
45 i = offset
46 for id in scan_ids:
47     print "
        *****"
48     print i
49     i = i+1
50     if(id[0]!=None):
51         getScanResults(id[0])

```

Above python codes uses Virus Total API and sends requests for all the malicious urls identified for the specific IP. The response is decoded and then stored into the table. response looks like below.

Listing 2.6: Virus Total API Request Sample

```

1 {
2   'response_code': 1,
3   'verbose_msg': 'Scan finished , scan information embedded in this
        object',
4   'scan_id': '1
        db0ad7dbcec0676710ea0eaacd35d5e471d3e11944d53bcbd31f0cbd11bce31
        -1390467782',
5   'permalink': 'https://www.virustotal.com/url/_urlsha256_/_/
        analysis/1390467782/',
6   'url': 'http://www.virustotal.com/',
7   'scan_date': '2014-01-23 09:03:02',
8   'filescan_id': null,
9   'positives': 0,
10  'total': 51,
11  'scans': {
12    'CLEAN MX': {
13      'detected': false,
14      'result': 'clean site'
15    },
16    'MalwarePatrol': {
17      'detected': false,
18      'result': 'clean site'
19    }
20  }
21 }

```

This response is parsed and requires fields are extracted to update the table in the database. These fields include "positives" which is the no of engines identified it as positive, "total" which is the count of the engines in total and "identified-engines" which contains the names of all the engines in the response.

### 2.5.2 Stats Table

S.No.	Column Name	Column Datatype
1	ip	character varying(20)
2	commands	text
3	countofcommands	integer
4	loginattempts	integer
5	doscluster	text
6	sentiment	integer

Table 2.4: Schema of Stats Table

Column "IP" contains the ip of the attacker or intruder and "commands" contains all the commands executed by the attacker or the intruder till now in cowrie honeypot. "Count-of-commands" contains the number of commands attacker executed. "login-attempts" contains the number of times attacker tried to login into the cowrie. "dos-cluster" contains the cluster number to which ip is clustered which will be updated dynamically. "sentiment" contains the value of ip which is updated after running machine learning algorithms.

All the python codes used for filling the table are given below one by one.

Listing 2.7: Get Unique IPs from Elastic Search

```
1 import json
2 import requests
3 import psycopg2
4 from Globals import *
5
6
7 uniqueIPs = set()
8 dbName = "cowrie"
9
10 try:
11     conn = psycopg2.connect("dbname='"+str(dbName)+"' user='
12     postgres' host='localhost' password='postgres'")
13 except Exception as e:
14     print e
15 cur = conn.cursor()
16
17 def createTables():
18     cur.execute("CREATE TABLE IF NOT EXISTS STATS("
19                 "IP VARCHAR(20) PRIMARY KEY NOT NULL,"
20                 "LOGINATTEMPTS INT,"
21                 "COUNTOFCOMMANDS INT,"
22                 "DOSCLUSTER INT,"
23                 "SENTIMENT INT,"
24                 "COMMANDS TEXT)")
25
26     cur.execute("CREATE TABLE IF NOT EXISTS DOS_ATTACKS("
27                 "IP VARCHAR(20) PRIMARY KEY NOT NULL,"
28                 "LOGINATTEMPTS INT,"
```

```

28         "COUNTOFCOMMANDS INT,"
29         "COMMANDS TEXT")
30
31     conn.commit()
32     cur.execute("TRUNCATE TABLE STATS")
33     conn.commit()
34
35 createTables()
36
37
38 url = ELASTIC_URL + str("?size=1")
39 response = requests.get(url)
40 countOfData = json.loads(response.content)["hits"]["total"]
41 print countOfData
42
43 size = 10000
44
45 for i in range(0, countOfData+1, size):
46     url = ELASTIC_URL + str("?_source=geoip.ip&size=")+str(size)+"&
47     from="+str(i)
48     response = requests.get(url)
49     IPs = json.loads(response.content)["hits"]["hits"]
50     print len(IPs)
51     for IP in IPs:
52         if IP["_source"] != {}:
53             ip = IP["_source"]["geoip"]["ip"]
54             uniqueIPs.add(ip)
55             print "Completed : from = "+str(i)+" and end = "+str(i+size)
56
57 for ip in uniqueIPs:
58     cur.execute("INSERT INTO STATS(IP) VALUES('"+ip+"')")
59     conn.commit()
60
61 print "Total Number of Unique IP's = " + str(len(uniqueIPs))

```

The above python code fetches all the IP's from elastic search, puts them in a set to avoid duplicates and copies all the IP's to the column named "ip" in Stats table. It uses the Elastic Search API's which looks like below.

Listing 2.8: Elastic Search API Request Sample

```

1 POST localhost:9200/logstash-*/_search
2 {
3     "size" : 0,
4     "aggs" : {
5         "distinct_ip" : {
6             "cardinality" : {
7                 "field" : "geoip.ip"
8             }
9         }
10    }
11 }
12 }

```

Listing 2.9: Get Commands Executed by IPs from Elastic Search

```

1 import json

```

```

2 import requests
3 import psycopg2
4 from Globals import *
5
6
7 dbName = "cowrie"
8 IPsArray = []
9
10 try:
11     conn = psycopg2.connect("dbname='"+str(dbName)+"' user='
12     postgres' host='localhost' password='postgres'")
13 except Exception as e:
14     print e
15 cur = conn.cursor()
16
17 def getNoOfUniqueIPsFromDB():
18     cur.execute("SELECT COUNT(*) FROM STATS");
19     rows = cur.fetchall()
20     for row in rows:
21         return row[0]
22
23 def getIPsFromDB():
24     cur.execute("SELECT IP FROM STATS");
25     rows = cur.fetchall()
26     for row in rows:
27         IPsArray.append(row[0])
28
29 print getNoOfUniqueIPsFromDB()
30 getIPsFromDB()
31
32 for i in range(0, getNoOfUniqueIPsFromDB()):
33     tillNowCommandsExecuted = ""
34     data = {
35         "query": {
36             "query_string": {
37                 "fields": ["src_ip", "eventid"],
38                 "query": ""+str(IPsArray[i])+" AND cowrie.command.
39             input"
40             },
41             "_source": ["geoip.ip", "eventid", "message"]
42         }
43     }
44     headers = {"Content-Type": "application/json"}
45     url = ELASTIC_URL + str("?size=10000")
46     response = requests.post(url, data=json.dumps(data), headers=
47     headers)
48     countOfCommands = json.loads(response.content)["hits"]["total"]
49     cur.execute("UPDATE STATS SET COUNTOFCOMMANDS=" + str(
50     countOfCommands) + "WHERE IP=" + IPsArray[i] + " ;")
51     conn.commit()
52     commands = json.loads(response.content)["hits"]["hits"]
53     if commands != []:
54         for command in commands:
55             tillNowCommandsExecuted += str(command["_source"] ["
56             message"].replace("CMD:", "")) + " "
57     print tillNowCommandsExecuted

```

```

54 cur.execute("UPDATE STATS SET COMMANDS=" + str(
    tillNowCommandsExecuted.replace(" ", ";").strip()) + " ' WHERE IP
55 =" + IPsArray[i] + "';")
    conn.commit()

```

The above python code fetches all the commands executed per IP , appends them together separated by semicolon and puts them in the column named "commands" in Stats table. It also updates the "countofcommands" column.

Listing 2.10: Get Login Attempts by IPs from Elastic Search

```

1 import json
2 import requests
3 import psycopg2
4 from Globals import *
5
6
7 dbName = "cowrie"
8 IPsArray = []
9
10 try:
11     conn = psycopg2.connect("dbname="+str(dbName)+" ' user='
        postgres ' host='localhost ' password='postgres '")
12 except Exception as e:
13     print e
14 cur = conn.cursor()
15
16 def getNoOfUniqueIPsFromDB():
17     cur.execute("SELECT COUNT(*) FROM STATS");
18     rows = cur.fetchall()
19     for row in rows:
20         return row[0]
21
22 def getIPsFromDB():
23     cur.execute("SELECT IP FROM STATS");
24     rows = cur.fetchall()
25     for row in rows:
26         IPsArray.append(row[0])
27
28
29 print getNoOfUniqueIPsFromDB()
30 getIPsFromDB()
31
32 for i in range(0, getNoOfUniqueIPsFromDB()):
33     tillNowCommandsExecuted = ""
34     data = {
35         "query": {
36             "query_string": {
37                 "fields": ["src_ip", "eventid"],
38                 "query": str(IPsArray[i])+" AND cowrie.login.*"
39             },
40         },
41         "_source": ["geoip.ip", "eventid", "message"]
42     }
43     headers = {"Content-Type": "application/json"}
44     url = ELASTIC_URL + str("?size=1")

```

```

45     response = requests.post(url,data=json.dumps(data),headers=
46         headers)
47     noOfLoginAttempts = json.loads(response.content)["hits"]["total"]
48     cur.execute("UPDATE STATS SET LOGINATTEMPTS=" + str(
49         noOfLoginAttempts) + "WHERE IP=" + IPsArray[i] + "';")
50     conn.commit()

```

The above python code gets the number of login attempts done by each IP from the elastic search. It counts the attempts and stores them in column named "loginattempts" in Stats table.

## DOS Attacks

This "loginattempts" login is used to predict whether IP is performing DOS attack or not. For this , only those IP's are considered where after they log in , they don't execute any commands and just leave. Spark K-Means Machine learning algorithm has been used to divide the IP's into 2 categories "High" and "Low". CSV file is generated which is supplied as input to K-Means algorithm. Pseudo code is given below.

Listing 2.11: Generate Login Attempts in a CSV File

```

1
2 import psycopg2
3
4 dbName = "cowrie"
5
6
7 try:
8     conn = psycopg2.connect("dbname="+str(dbName)+" user='
9         postgres' host='localhost' password='postgres'")
10 except Exception as e:
11     print e
12 cur = conn.cursor()
13
14
15
16 cur.execute("SELECT LOGINATTEMPTS FROM STATS WHERE COUNTOFCOMMANDS
17     = 0 ORDER BY LOGINATTEMPTS DESC")
18 rows = cur.fetchall()
19 print "Login_Attempts"
20 for row in rows:
21     print(row[0])

```

Listing 2.12: Applying Spark K-Means Algorithm

```

1 from pyspark.ml.linalg import Vectors
2 from pyspark.ml.feature import StandardScaler
3 from pyspark.ml.feature import VectorAssembler
4 from pyspark.ml.clustering import KMeans
5 from pyspark.sql import SparkSession
6

```

```

7
8 spark = SparkSession.builder.appName("TEST").getOrCreate()
9
10 Create Dataframe From CSV
11
12 cluster_df = spark.read.csv("/home/iotsys3/PycharmProjects/krishna/
    Cowrie-Scripts/loginattempts.csv",
13     header=True,
14     inferSchema=True,
15 )
16
17
18
19 Removing Columns that are not needed from DataFrame
20
21 #cluster_df = cluster_df.drop("id","date")
22
23
24
25 Convert Data to Vector as Standardised Scaler
26 and Kmeans can only operate on that data
27
28 vectorAssembler = VectorAssembler(
29     inputCols=["Login_Attempts"],
30     outputCol="features"
31 )
32
33
34 vcluster_df = vectorAssembler.transform(cluster_df)
35 #vcluster_df.show()
36 test_df = vectorAssembler.transform(cluster_df)
37
38
39 kmeans = KMeans().setK(3) # set number of clusters
40 kmeans.setSeed(1) # set start point
41 kmodel = kmeans.fit(vcluster_df)
42
43 centers = kmodel.clusterCenters()
44
45 pred_df = kmodel.transform(test_df)
46
47 print("\t**OUTPUT**\n\n\n")
48 print(vcluster_df.show())
49 print("Centers RK = " + str(centers))
50 print(kmodel.summary.clusterSizes)
51 print("\n\n\n")
52 pred_df.show(10)
53 print "Center -1 " + str(centers[0])
54 print "Center -2 " + str(centers[2])
55 print "Center -3 " + str(centers[1])

```

Listing 2.13: Counting of DOS attacks

```

1 import psycopg2
2
3 dbName = "cowrie"
4
5

```



```

6 cluster1center = 25.21769815
7 cluster2center = 11354.0
8 cluster3center = 20776.75
9
10 try:
11     conn = psycopg2.connect("dbname="+str(dbName)+" user='
12     postgres' host='localhost' password='postgres'")
13 except Exception as e:
14     print e
15 cur = conn.cursor()
16
17 def selectCluster(dis1,dis2,dis3):
18     if (dis1 < dis2) and (dis1 < dis3):
19         return 1
20     elif (dis2 < dis1) and (dis2 < dis3):
21         return 2
22     else:
23         return 3
24
25 cur.execute("SELECT IP,LOGINATTEMPTS FROM STATS WHERE
26     COUNTOFCOMMANDS = 0 ORDER BY LOGINATTEMPTS DESC")
27 rows = cur.fetchall()
28 for row in rows:
29     clusterNo = selectCluster(abs(row[1]-cluster1center),abs(row
30     [1]-cluster2center),abs(row[1]-cluster3center))
31     cur.execute("UPDATE STATS SET DOSCLUSTER="+str(clusterNo)+"
32     WHERE IP="+str(row[0])+" ';'")
33     conn.commit()
34
35 countOfSevereDosAttacks = 0
36 countOfMediumDosAttacks = 0
37 countOfLowDosAttacks = 0
38
39 cur.execute("SELECT COUNT(IP) FROM STATS WHERE DOSCLUSTER=1")
40 rows = cur.fetchall()
41 for row in rows:
42     countOfLowDosAttacks = row[0]
43     break
44
45 cur.execute("SELECT COUNT(IP) FROM STATS WHERE DOSCLUSTER=2")
46 rows = cur.fetchall()
47 for row in rows:
48     countOfMediumDosAttacks = row[0]
49     break
50
51 cur.execute("SELECT COUNT(IP) FROM STATS WHERE DOSCLUSTER=3")
52 rows = cur.fetchall()
53 for row in rows:
54     countOfSevereDosAttacks = row[0]
55     break
56
57 print "+++++-----+"
58 print "+++++| Severe DOS Attacks = "+str(countOfSevereDosAttacks)
59 print "+++++| Medium DOS Attacks = "+str(countOfMediumDosAttacks)
60 print "+++++| Low DOS Attacks = "+str(countOfLowDosAttacks)
61 print "+++++-----+"

```

After the above python file is executed, DOS attacks table looks like below. Some IP's are also performing sniffing attacks where in they login and check for busybox utility.

S.No.	DOS Cluster	No. of IP's
1	Severe	16
2	Medium	5
3	Low	1842
	<b>Total</b>	<b>1863</b>

Table 2.5: DOS Attacks Clustering Table

### Modified Naive Bayes With K-Means for Classification

Based on the commands executed by the IP, they have been classified into two classes as high and low using Modified Naive Bayes. As all the data is unlabeled, Sentiment analysis cannot be applied using original Naive Bayes. In modified naive Bayes, prior probability is assumed to be one as we have all data which belongs to only attack class. Effective algorithm is given below.

Listing 2.14: Modified Naive Bayes Algorithm Code

```

1 import psycopg2
2 from collections import Counter
3 import math
4
5 dbName = "cowrie"
6 totalWords = []
7 uniqueWords = set()
8 probabilities = {}
9 sentiments = []
10
11 try:
12     conn = psycopg2.connect("dbname="+str(dbName)+" user='
13     postgres' host='localhost' password='postgres'")
14 except Exception as e:
15     print e
16 cur = conn.cursor()
17 cur.execute("SELECT COMMANDS FROM STATS")
18 rows = cur.fetchall()
19 for row in rows:
20     commands = row[0].replace(";", " ").split(" ")
21     for command in commands:
22         if command != " ":
23             uniqueWords.add(command.strip())
24             totalWords.append(command.strip())
25
26 counts = Counter(totalWords)
27 for word in uniqueWords:
28     probability = (float)(counts.get(word) + 1) / (float)(len(
29         totalWords) + len(uniqueWords))
30     probabilities.setdefault(word, math.exp(probability))

```

```

30
31
32 cur.execute("SELECT IP,COMMANDS FROM STATS")
33 rows = cur.fetchall()
34 for row in rows:
35     TotalProb = 1.0
36     commands = row[1].replace(";", " ").split(" ")
37     for command in commands:
38         if command != " ":
39             if probabilities.get(command) != None:
40                 TotalProb *= probabilities.get(command)
41     #print str(row[0]) + "=====>" + str(TotalProb)
42     if TotalProb != 1.0:
43         sentiments.append((row[0], TotalProb))
44
45 print "Sentiments"
46 for sentiment in sentiments:
47     print sentiment[1]
48
49
50 print bcolors.WARNING + "Warning: No active frommets remain.
    Continue?" + bcolors.ENDC

```

After the application of Modified Naive Bayes, Posterior Probabilities are generated which are given as input to K-Means algorithm which groups them into clusters. Clusters are designed to be of two types naming high and low. Based on the results, below table has been deduced.

Listing 2.15: K-Means Algorithm using Spark

```

1 from pyspark.ml.linalg import Vectors
2 from pyspark.ml.feature import StandardScaler
3 from pyspark.ml.feature import VectorAssembler
4 from pyspark.ml.clustering import KMeans
5 from pyspark.sql import SparkSession
6
7
8 spark = SparkSession.builder.appName("TEST").getOrCreate()
9 '''
10 Create Dataframe From CSV
11 '''
12 cluster_df = spark.read.csv("/home/iotsys3/PycharmProjects/krishna/
    Cowrie_Scripts/sentiments.csv",
13     header=True,
14     inferSchema=True,
15 )
16
17
18 '''
19 Removing Columns that are not needed from DataFrame
20 '''
21 #cluster_df = cluster_df.drop("id","date")
22
23
24 '''
25 Convert Data to Vector as Standardised Scaler

```

```

26 and Kmeans can only operate on that data
27 '''
28 vectorAssembler = VectorAssembler(
29     inputCols=["Sentiments"],
30     outputCol="features"
31 )
32
33 '''
34 vcluster_df = vectorAssembler.transform(cluster_df)
35 #vcluster_df.show()
36 test_df = vectorAssembler.transform(cluster_df)
37
38
39 kmeans = KMeans().setK(2) # set number of clusters
40 kmeans = kmeans.setSeed(1) # set start point
41 kmodel = kmeans.fit(vcluster_df)
42
43 centers = kmodel.clusterCenters()
44
45 pred_df = kmodel.transform(test_df)
46
47 print("\t**OUTPUT**\n\n")
48 print(vcluster_df.show())
49 print("Centers RK = " + str(centers))
50 print(kmodel.summary.clusterSizes)
51 print("\n\n")
52 pred_df.show(10)
53 print "Center -1 " + str(centers[0])
54 print "Center -2 " + str(centers[1])

```

Listing 2.16: Classifying IPs

```

1 import psycopg2
2 from collections import Counter
3 import math
4
5 dbName = "cowrie"
6 totalWords = []
7 uniqueWords = set()
8 probabilities = {}
9 sentiments = []
10 classification = []
11 center_1 = 1.65478756
12 center_2 = 2.87508731e+08
13 class_1=0
14 class_2=0
15 class_3=0
16
17 def classify(dis):
18     dis1 = abs(center_1 - dis)
19     dis2 = abs(center_2 - dis)
20     if dis2 > dis1:
21         return 2
22     else:
23         return 3
24
25
26 try:

```

```

27     conn = psycopg2.connect("dbname='"+str(dbName)+"' user='
    postgres' host='localhost' password='postgres'")
28 except Exception as e:
29     print e
30 cur = conn.cursor()
31
32 cur.execute("SELECT COMMANDS FROM STATS")
33 rows = cur.fetchall()
34 for row in rows:
35     commands = row[0].replace(";", " ").split(" ")
36     for command in commands:
37         if command != "":
38             uniqueWords.add(command.strip())
39             totalWords.append(command.strip())
40
41 counts = Counter(totalWords)
42 for word in uniqueWords:
43     probability = (float)(counts.get(word) + 1) / (float)(len(
    totalWords) + len(uniqueWords))
44     probabilities.setdefault(word, math.exp(probability))
45
46 cur.execute("SELECT IP, COMMANDS FROM STATS")
47 rows = cur.fetchall()
48 for row in rows:
49     TotalProb = 1.0
50     commands = row[1].replace(";", " ").split(" ")
51     for command in commands:
52         if command != "":
53             if probabilities.get(command) != None:
54                 TotalProb *= probabilities.get(command)
55     sentiments.append((row[0], TotalProb))
56
57 for sentiment in sentiments:
58     if sentiment[1] == 1.0:
59         classification.append((sentiment[0], sentiment[1], 1))
60     else:
61         classification.append((sentiment[0], sentiment[1], classify(
    sentiment[1])))
62
63 for clasif in classification:
64     print str(clasif[0])+"====="+str(clasif[1])+str("====="
    )+str(clasif[2])
65     if clasif[2] == 1:
66         class_1 += 1
67     elif clasif[2] == 2:
68         class_2 += 1
69     else:
70         class_3 += 1
71
72
73
74 print "+++++----- No Of IPs -----+"
75 print "+++++| Critical = "+str(class_3)
76 print "+++++| Medium   = "+str(class_2)
77 print "+++++| Low      = "+str(class_1)
78 print "+++++-----+"

```

---

**Algorithm 1** Modified Naive Bayes With K-Means Clustering Algorithms

---

1: **Probability**(*IP is Malicious / Commands executed by IP*) =

$$\frac{\mathbf{Probability}(IP \text{ is Malicious}) * \mathbf{Probability}(\text{Commands executed by IP} / IP \text{ is Malicious})}{\mathbf{Probability}(\text{Commands executed by IP})}$$

2: **Note** :As we are working on Unclassified Data, We are assuming below values.

3: **Probability**(*IP is Malicious*) = 1

4: **Probability**(*Commands executed by IP*) = 1

5: **Probability**(*Commands executed by IP / IP is Malicious*) =

$$\frac{\text{Frequency of Word} + 1 \text{ (Smoothing Factor)}}{\text{Total Number of Words} + \text{Unique Number of Words}}$$

6: **Note** : *Commands executed by each IP is split into words and their frequencies are stored in Database*

7: *Total No of Words*  $\leftarrow N$

8: *Unique No of Words*  $\leftarrow \text{dict}$

9: **while** *IP List not Empty* **do**

10:   *No of Words*  $\leftarrow$  *Splitting Commands into words*

11:   **Probability**(*IP is Malicious / Commands executed by IP*)

12:   = **Probability** (*Commands executed by IP / IP is Malicious*)

13:   =  $\prod_{i=1}^{\text{No of Words}}$  **Probability**(*Word<sub>i</sub> in Command Executed/ IP is Malicious*)

14:   =  $\prod_{i=1}^{\text{No of Words}} \frac{n_i + 1}{N + \text{dict}}$

15: **done**

16: **Note** : *Each IP Final Probability will be stored in Database*

17: **Note** : *After Storing the IP's Probabilities in Database, K-Means*

*Clustering algorithm is applied to classify the IP's into two classes as high and Low.*

\*\*\*\*\*

**K-Means is given below**

\*\*\*\*\*

18: **Input** : *Set of Data Points D and No. of Clusters K.*

19: **Output** : *Cluster Centers that minimizes the squared error distortion.*

20: **Algorithm** :

1. *Pick K Data points randomly from D to form cluster centers.*
  2. *Assign each data point to its nearest cluster center by calculating and taking the minimum Euclidean Squared distance metric.*
  3. *After all data points are assigned to their clusters, move each cluster center to mean of its assigned data points.*
  4. *With new cluster centers, repeat 2 to 3 until there is no convergence.*
-

S.No.	Cluster Name	No. of IP's
1	High	136
2	Low	1842
	<b>Total</b>	<b>1978</b>

Table 2.6: Modified Naive Bayes and K-Means Results

### DDOS Binaries

Attackers who were able to login successfully has used their access to full effect by deploying the binaries and executable files on the machine and running them remotely whenever needed. Fortunately Cowrie happened to be a medium interaction honeypot which would not allow running any binaries or executable files. Some of the captured binaries are sent to VirusTotal and all of them are identified as serious DDos attacking binaries. Some of them are listed below.

./lPg5Am8r	./AOBM55mP	./ScDrDSSSt	./YL88yLHr
./Tr5l603l	./yu5LvV97	./VdPacLUl	./uzsiO4Hx
./BRB3bpfb	./PWnQ7Tcn	./GgzxWgHv	./zanTB34C
./udp4858	./kEbIZq9x	./MvlFggnh	./6PcycAuj
./o3e1ROxG	./b1fNqt0C	./iUk3up10	./M7vu30G4
./KAFsqrpv	./GYOIsFtx	./tf2SD7fn	

Table 2.7: List of DDos Binaries

Some of the malicious URL's accessed by the attackers is captured by the cowrie. These are given below in the table.

http://www.bizqsoft.com/tp2/r6.log
http://185.165.29.196/lmao.sh
http://mdb7.cn:8081/exp
http://203.146.208.208/drago/images/.ssh/y.txt
http://162.248.4.130/isu80
https://doiaru.000webhostapp.com/zeni.txt
http://61.147.112.10:1242/udp4858
http://104.216.151.157/i3306m
http://23.228.113.240/do3309
http://23.228.113.240/ys808e
http://14.142.118.25/m.sh
http://107.189.130.20/ys808e
http://192.200.218.166/g3308l

Table 2.8: List of Malicious URLs

Based on all the data listed above and tables created in PostgreSQL database and API's from VirusTotal, an intrusion detection system software has been developed which continuously monitors the network where commands executed by any IP is noted and sent to the database built. If it is found in the database, it will be reported as malicious. If no information is found about the data in database, it keeps that in a separate column and keeps monitoring. If any anomaly is found, then the database will be updated with the new one. So it is a continuous learning process for the software. Code is given below.

Listing 2.17: Intrusion Detection System Code

```

1 import psycpg2
2 from collections import Counter
3 from prettytable import PrettyTable
4 from termcolor import colored
5 import sys
6
7 IdentifiedEnginesCount = PrettyTable(['Antivirus Engine', '
    Identified Count'])
8 MaliciousURLSCount = PrettyTable(['URL', 'Positives', 'Total'])
9
10 uniqueURLs = set()
11 dbName = "cowrie"
12 #ipToTest = "192.200.218.166"
13 #print str(sys.argv)
14 ipToTest = str((sys.argv)[1])
15 engines = []
16 malicious = 0;
17
18 try:
19     conn = psycpg2.connect("dbname="+str(dbName)+" user='
        postgres' host='localhost' password='postgres'")
20 except Exception as e:
21     print e
22 cur = conn.cursor()
23
24 cur.execute("SELECT URL, POSITIVES, TOTAL, IDENTIFIED_ENGINES FROM
    MALICIOUS_URLS WHERE URL LIKE '%" + str(ipToTest) + "%'")
25 rows = cur.fetchall()
26 for row in rows:
27     for engine in row[3].split(";"):
28         malicious = 1
29         engines.append(engine)
30         if row[0] != "":
31             uniqueURLs.add(row[0].strip())
32
33
34 for url in uniqueURLs:
35     cur.execute("SELECT URL, POSITIVES, TOTAL FROM MALICIOUS_URLS
        WHERE URL LIKE '%" + str(url) + "%' LIMIT 1")
36     rows = cur.fetchall()
37     for row in rows:
38         MaliciousURLSCount.add_row([row[0].strip(), row[1], row[2]])
39
40 if(malicious == 1):

```



```

41 print "
42 +-----+
43 print "| IP : " + str(ipToTest) + " is Classified as " + colored('
Malicious', 'red', attrs=['bold']) + " |";
44 print "
45 +-----+
46 print " | Identified as " + colored('Malicious', 'red', attrs=['
bold']) + " By Below Antivirus Engines |"
47 print "
48 +-----+
49 counts = Counter(engines)
50 for engine in counts:
51     IdentifiedEnginesCount.add_row([engine, counts.get(engine)])
52
53 print IdentifiedEnginesCount
54 print "
55 +-----+
56 | Identified Below URLs as Malicious pertaining to
the given IP |"
57 print "
58 +-----+
59
60 print MaliciousURLSCount
61
62 else:
63 print "
64 +-----+
65 print "| No Data Found in the Database for the Specified IP
|"
66 print "| IP : " + str(ipToTest) + " is " + colored('Not
Malicious', 'green', attrs=['bold']) + " |";
67 print "
68 +-----+

```

Figure 2.11: IP Identified as NOT Malicious by IDS

```

+-----+
| No Data Found in the Database for the Specified IP |
| IP : 192.200.218.165 is Not Malicious |
+-----+

```

Based on the output from the IDS, firewall rules will be updated with the specific IP as blocking or nonblocking.

**Blocking :** *iptables -A INPUT -s 192.200.218.166 -j DROP*

**Unblocking :** *iptables -D INPUT -s 192.200.218.165 -j DROP*

Figure 2.12: IP Identified as Malicious by IDS

IP : 192.200.218.166 is Classified as <b>Malicious</b>		
Identified as <b>Malicious</b> By Below Antivirus Engines		
Antivirus Engine	Identified Count	
CyRadar	5	
AegisLab WebGuard	7	
Yandex Safebrowsing	8	
SCUMWARE.org	1	
PREBYTES	5	
Sophos	7	
Nucleon	3	
Kaspersky	3	
BitDefender	8	
Avira	4	
Fortinet	7	
ZeroCERT	1	
Identified Below URLs as Malicious pertaining to the given IP		
URL	Positives	Total
http://192.200.218.166/i3306m	8	68
http://192.200.218.166/isu80	8	68
http://192.200.218.166/ps23e	8	68
http://192.200.218.166/is80	3	67
http://192.200.218.166/g3308l	8	68
http://192.200.218.166/ys808e	8	67
http://192.200.218.166/ys53a	8	67

## 2.6 Conclusion

In this report, we were able to figure out how vulnerable most of the internet devices are and what must be done in order to secure those devices. Recent developements in IOT has forced our hand to look at the security features at a microscopic level and improvements to be done. We have also tried to classify the different types of attacks on different IOT devices and on different ports. We have also tried to design a detection layer called Intrusion Detection System using all the historical and present data and were able to prevent if not detect some of those attacks. A lot has to be done in the security related work and hope everyone will put their hand towards acheiving it.

# Bibliography

- [1] Performance of ELK Stack and Commercial System in Security Log Analysis by *Sung Jun Son and Youngmi Kwon*
- [2] Building an IoT Data Hub with Elasticsearch, Logstash and Kibana by *Marcin Bajer*
- [3] Network security enhancement through effective log analysis using ELK by *Ibrahim Yahya Mohammed Al-Mahbashi and M. B. Potdar and Prashant Chauhan*
- [4] Malware capturing and detection in dionaea honeypot by *P Dilsheer Ali and T. Gireesh Kumar*
- [5] Integration of network intrusion detection systems and honeypot networks for cloud security by *Varan Mahajan and Sateesh K Peddoju*
- [6] Honeynet Data Analysis and Distributed SSH Brute-Force Attacks by *Gokul Kannan Sadasivam and Chittaranjan Hota and Anand Bhojan*
- [7] Detection of Severe SSH Attacks using Honeypot Servers and Machine Learning Techniques by *Gokul Kannan Sadasivam and Chittaranjan Hota and Anand Bhojan S*
- [8] Classification of SSH attacks using Machine learning algorithms by *Gokul Kannan Sadasivam and Chittaranjan Hota and Anand Bhojan*
- [9] IoT honeypot: A multi-component solution for handling manual and Mirai-based attacks by *Sasa Mrdovic and Haris Šemić*
- [10] Analysis of modern intrusion detection system by *Aleksey A. Titorenko and Alexey A. Frolov*
- [11] Clustering Approach Based on Mini Batch Kmeans for Intrusion Detection System Over Big Data by *Parisa Lotfallahabrizi and Yasser Morgan*
- [12] Feature selection based intrusion detection system using the combination of DBSCAN, K-Mean++ and SMO algorithms by *Vandana Shakya and Rajni Ranjan Singh Makwana*

- [13] DDoS Attack Detection System: Utilizing Classification Algorithms with Apache Spark by *Amjad Alsirhani and Srinivas Sampalli and Peter Bodorik*
- [14] Testing of algorithms for anomaly detection in Big data using apache spark by *Sheeraz Niaz Lighari and Dil Muhammad Akbar Hussain*
- [15] Feature selection based intrusion detection system using the combination of DBSCAN, K-Mean++ and SMO algorithms by *Vandana Shakya and Rajni Ranjan Singh Makwana*
- [16] <https://www.virustotal.com>
- [17] Privacy and Security of Cloud-Based Internet of Things (IoT) by *Tanupriya Choudhury and Ayushi Gupta and Saurabh Pradhan and Praveen Kumar; Yogesh Singh Rathore*
- [18] Internet of Things (IoT) security platforms by *Ibrahim R. Waz and Mohamed Ali Sobh and Ayman M. Bahaa-Eldin*
- [19] A Search Engine Backed by Internet-Wide Scanning by *Zakir Durumeric and David Adrian and Ariana Mirian and Michael Bailey and J. Alex Halderman*