Contents

1 Introduction of Tools **1**
- 1.1 ELK Stack . . . 1
  - 1.1.1 Elastic Search . . . 1
  - 1.1.2 Logstash . . . 1
  - 1.1.3 Kibana . . . 1
- 1.2 Honeypots . . . 1
  - 1.2.1 Low-interaction honeypots . . . 2
  - 1.2.2 Medium-interaction honeypots . . . 2
  - 1.2.3 High-interaction honeypots . . . 2
  - 1.2.4 Cowrie Honeypot . . . 2
- 1.3 VirusTotal . . . 3
  - 1.3.1 List of Anti-Virus Engines used by VirusTotal . . . 3
- 1.4 Github . . . 3

2 Installation of Tools **4**
- 2.1 Elastic Search . . . 4
- 2.2 Logstash . . . 5
- 2.3 Kibana . . . 5
- 2.4 Cowrie Honeypot . . . 6
  - 2.4.1 Install dependencies . . . 6
  - 2.4.2 Create a user account . . . 6
  - 2.4.3 Checkout the code . . . 6
  - 2.4.4 Setup Virtual Environment . . . 6
  - 2.4.5 Port redirection . . . 6
  - 2.4.6 Starting Cowrie . . . 6

3 Working Model **7**

4 Monitoring System **8**
- 4.1 Stochastic Process . . . 13

5 Intrusion Detection System **14**
- 5.1 Malicious-URLs Table . . . 16
- 5.2 Stats Table . . . 20

List of Figures

1 Elastic Search Output . . . 4
2 Kibana Dashboard . . . 5
3 Working Model . . . 7
4 Monitoring System . . . 8
5 Daily Attacks Count Area Graph . . . 11
6 Top 20 Usernames Histogram . . . 11
7 Unique Count of IP's . . . 12

8 Top 20 Passwords Pie 12
9 Global Attacks Map 13
10 Stochastic Process 14
11 Intrusion Detection System 15

List of Tables

1 List of Antivirus Engines 3
2 List of Tables in PostgreSQL Database 15
3 Schema of Malicious-URLs Table 16
4 Schema of Stats Table 20
5 DOS Attacks Clustering Table 27

Listings

1 Cowrie Config File 8
2 Get Malicious URLs From Virus Total 16
3 Add URL's to PostgreSQL Database 17
4 Submit URL's for Scan to VirusTotal 17
5 Get URL Scan results from VirusTotal 18
6 Virus Total API Request Sample 19
7 Get Unique IPs from Elastic Search 20
8 Elastic Search API Request Sample 21
9 Get Commands Executed by IPs from Elastic Search 21
10 Get Login Attempts by IPs from Elastic Search 23
11 Generate Login Attempts in a CSV File 24
12 Applying Spark K-Means Algorithm 24
13 Counting of DOS attacks 25

1 Introduction of Tools

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

1.1 ELK Stack

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

1.1.1 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.

1.1.2 Logstash

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

1.1.3 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.

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

1.2.1 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.

1.2.2 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.

1.2.3 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.

1.2.4 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.

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.

1.3.1 List of Anti-Virus Engines used by VirusTotal

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 1: List of Antivirus Engines

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 Installation of Tools

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

2.1 Elastic Search

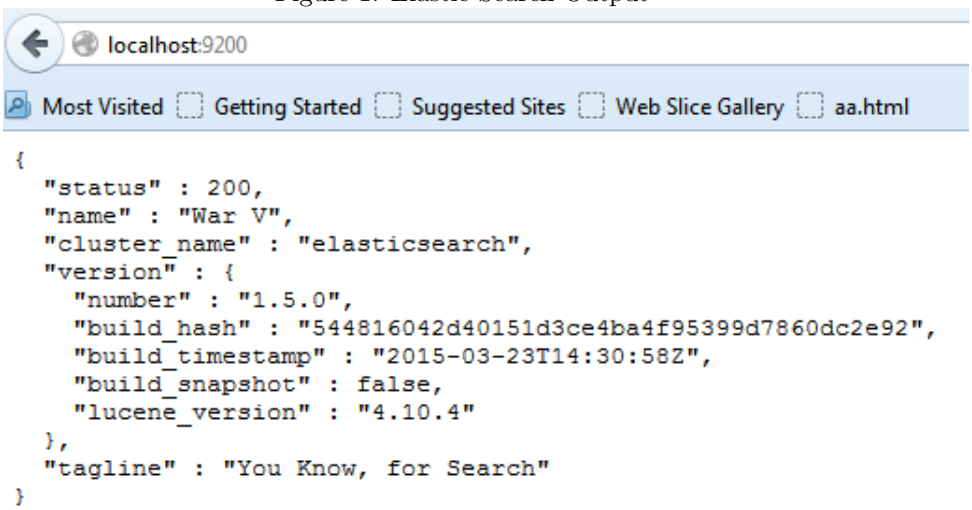
1. Download the Debian Installation file from the below link

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 ouput which should look like below.

Figure 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 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

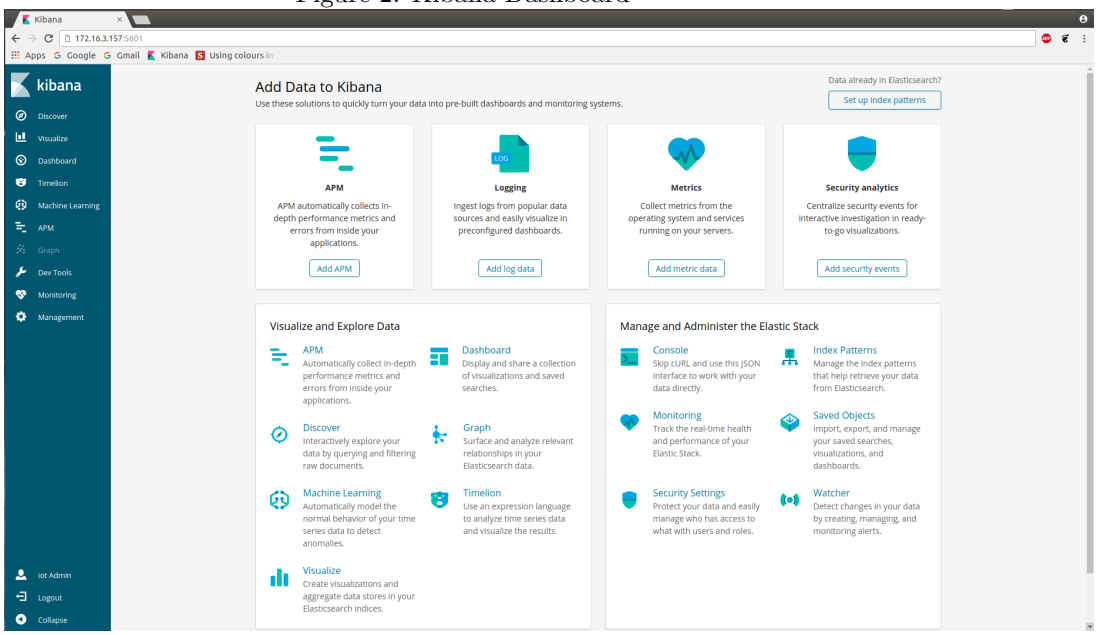
2.3 Kibana

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: Kibana Dashboard



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

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.

2.4.1 Install dependencies

iotsys3@iotsys3-Precision-Tower-3420: sudo apt-get install git python-virtualenv libssl-dev libffi-dev build-essential libpython-dev python2.7-minimal authbind

2.4.2 Create a user account

iotsys3@iotsys3-Precision-Tower-3420: sudo adduser –disabled-password cowrie
iotsys3@iotsys3-Precision-Tower-3420: sudo su - cowrie

2.4.3 Checkout the code

iotsys3@iotsys3-Precision-Tower-3420: git clone http://github.com/micheloosterhof/cowrie
iotsys3@iotsys3-Precision-Tower-3420: cd cowrie

2.4.4 Setup Virtual Environment

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

2.4.5 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.

2.4.6 Starting Cowrie

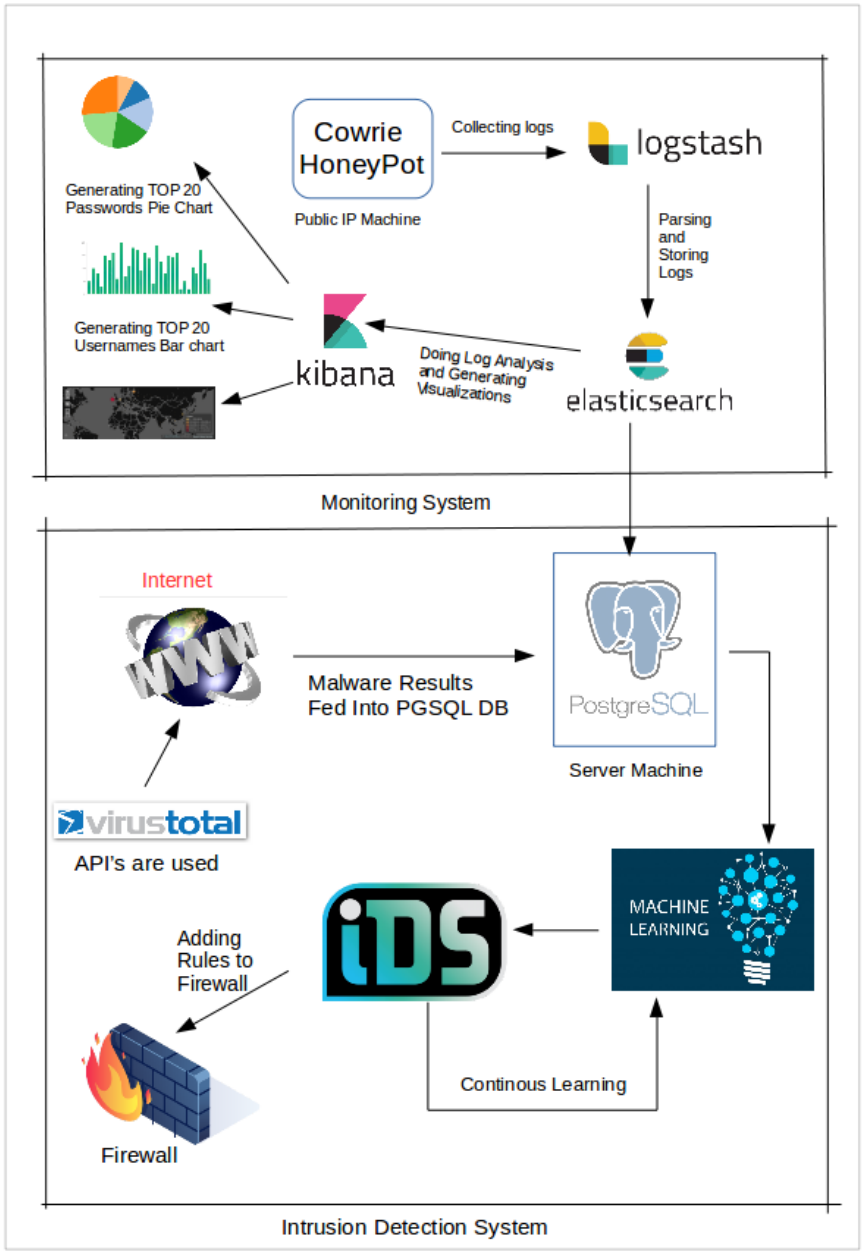
iotsys3@iotsys3-Precision-Tower-3420: bin/cowrie start

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

3 Working Model

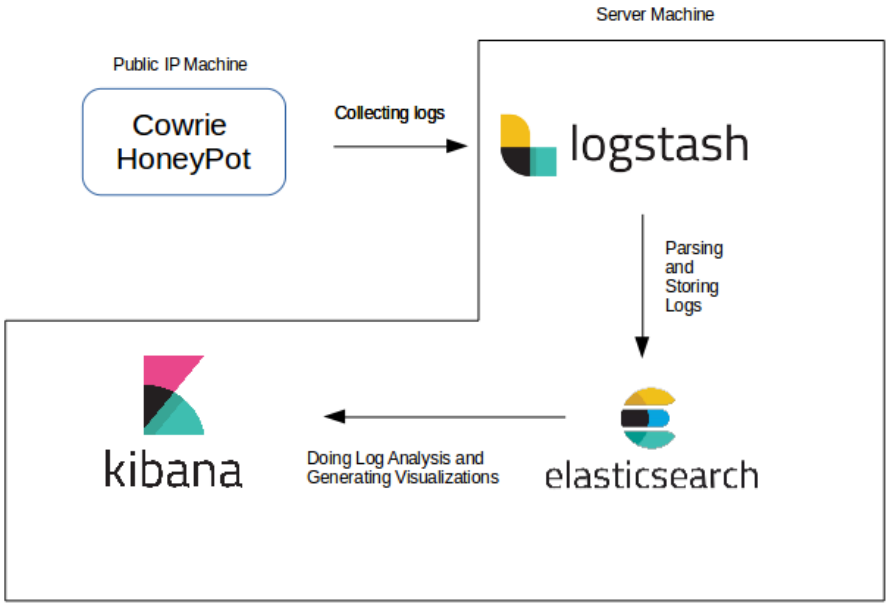
All the tools listed above are integrated to form a complete Working Model which looks like below.

Figure 3: Working Model



4 Monitoring System

Figure 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 1: Cowrie Config File

```
input {
  file {
    path => "/home/iotsys3/Documents/Cowrie_Logs/*"
    start_position => "beginning"
    ignore_older => 0
  }
```

```
}
filter {

  json {
            source => message
        }

        date {
            match => [ "timestamp", "ISO8601" ]
        }

        if [src_ip]  {

            dns {
                reverse => [ "src_host", "src_ip" ]
                action => "append"
            }

            geoip {
                source => "src_ip"  # With the src_ip field
                target => "geoip"   # Add the geoip one
                # Using the database we previously saved
                database => "/home/iotsys3/Downloads/GeoLite2-City_20180206/GeoLite2-City.mmdb"
                add_field => [ "[geoip][coordinates]", "%{[geoip][longitude]}" ]
                add_field => [ "[geoip][coordinates]", "%{[geoip][latitude]}"  ]
            }

            # Get the ASN code as well
            geoip {
                source => "src_ip"
                database => "/home/iotsys3/Downloads/GeoLite2-ASN_20180206/GeoLite2-ASN.mmdb"
            }

            mutate {
                convert => [ "[geoip][coordinates]", "float" ]
            }
        }

}
output {

  elasticsearch {
            hosts => ["localhost:9200"]
            sniffing => true
            manage_template => false
            index => "logstash-cowrie-%{+YYYY.MM.dd}"
          user => "elastic"
            password => "63TxZjWQNkB4tArR7hri"
        }

  stdout {
            codec => rubydebug
        }
```

```
60 }
```

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 elastc search where data is stored in json format which will be in key value format.

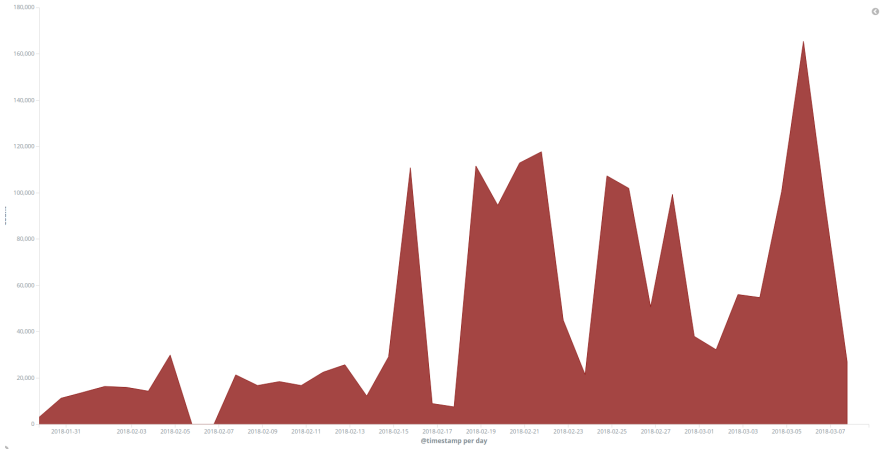
Data collected in the elastic search will be picked up by the Kibana by default as specified in the Kibana config files. Using the data visualizations are created for analyzing and monitoring the regular changes in data. Monitoring system was deployed for 40 days and collected nearly **2 Million Records**.

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 observed that, attacks kept on increasing at the end.

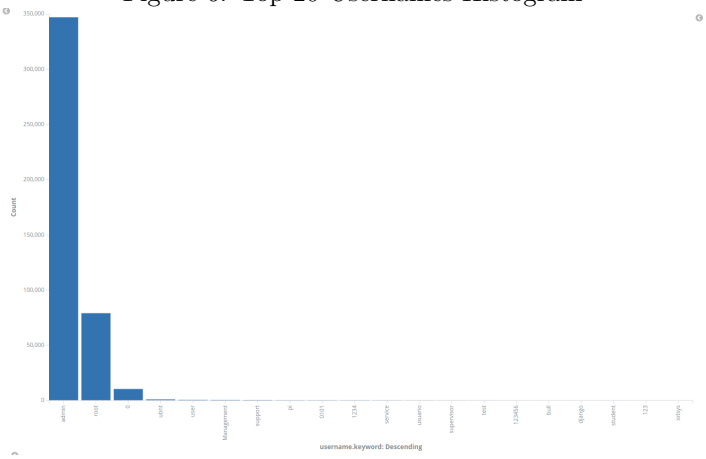
Figure 5: Daily Attacks Count Area Graph



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 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

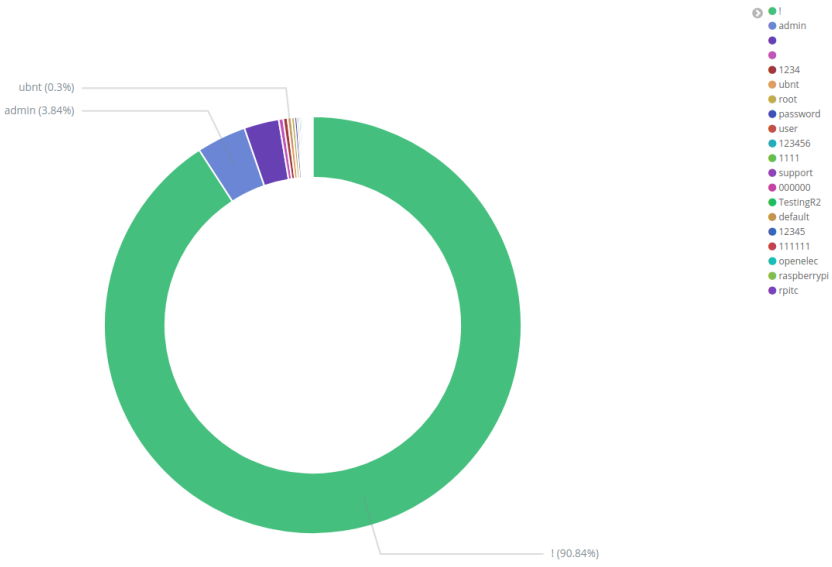
A Kibana Utility graph named Unique Count has been used for counting the Unique Number of IP's till now monitoring system has captured. It has found to be nearly 2000 IP's.

Figure 7: Unique Count of IP's

4. Top 20 Passwords Pie

A pie chart has been drawn against the top 20 passwords used by the attackers and intruders. It can be observed that passwords "ubnt" and "admin" are mostly used.

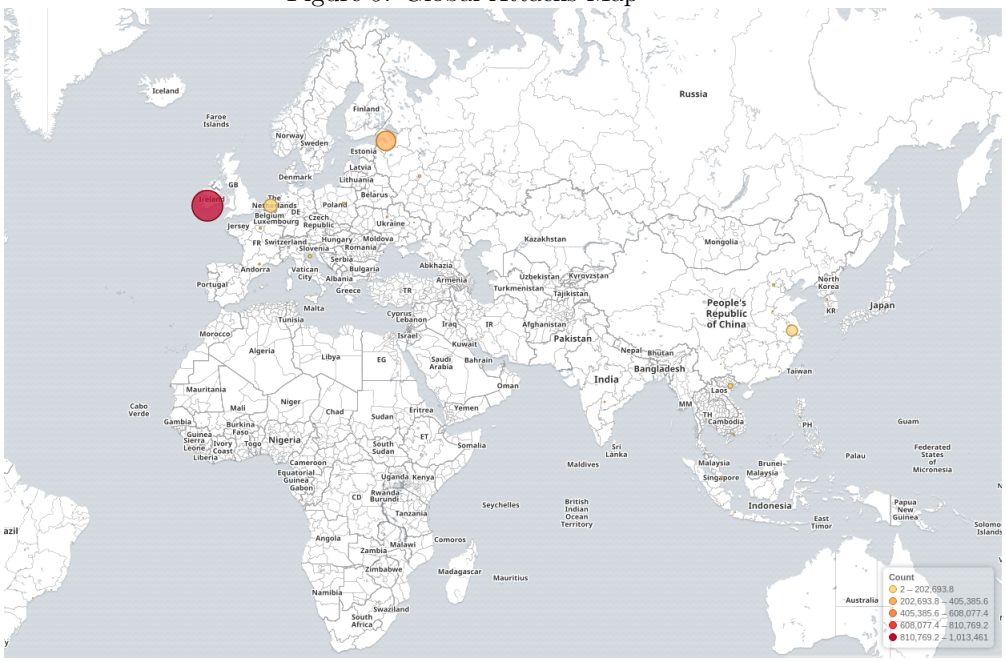
Figure 8: Top 20 Passwords Pie



5. Global Attacks Map

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

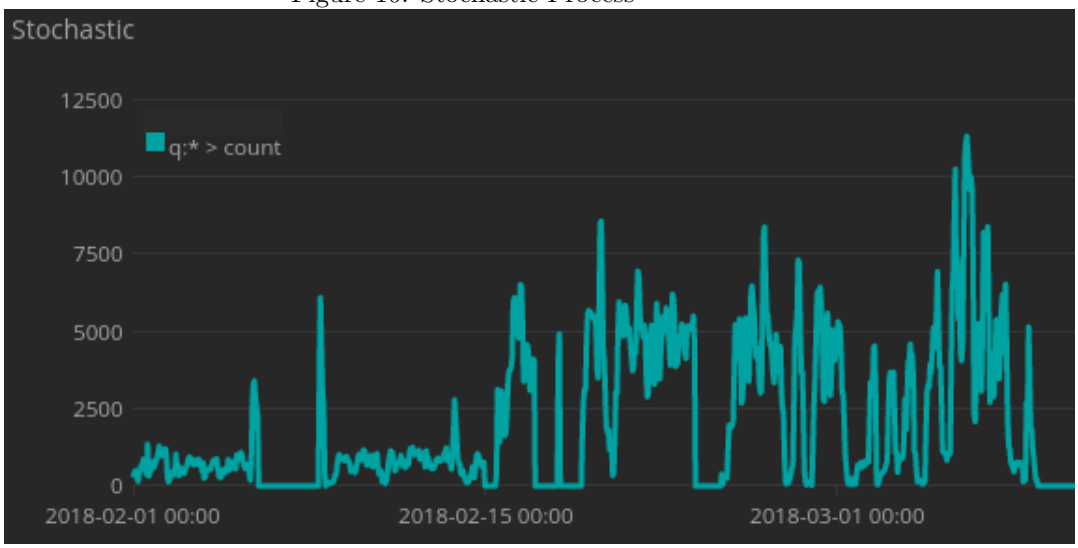
Figure 9: Global Attacks Map



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 10: Stochastic Process



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 activites 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.

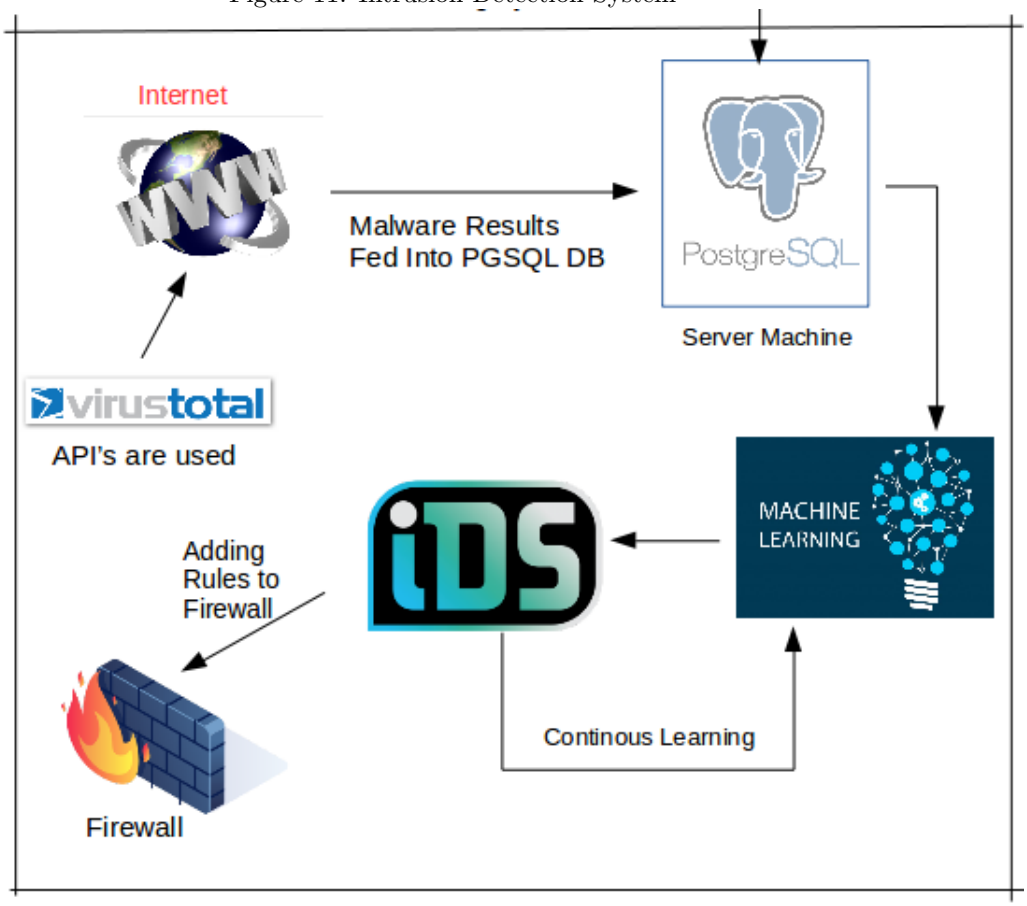
Virus Total is a website where details about different malwares and IP's can be fetched by using their web interface or REST API's. For our convenience, API's are used to integrate with our python programs. Virus Total gives a normal user on an average of 1000 API requests per day combining malware samples 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: List of Tables in PostgreSQL Database

Figure 11: Intrusion Detection System



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 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: Get Malicious URLs From Virus Total

```
import requests
import json
import psycopg2

apikey = "
    f0aa0772e518487aa5daf1dcf2be2d37c9b9b361b1a0d8339faf972210b36d41
    "
scan_url = 'https://www.virustotal.com/vtapi/v2/ip-address/report?
    apikey='+apikey+'&ip='
uniqueURLs = set()

dbName = "cowrie"
offset = 1813
try:
    conn = psycopg2.connect("dbname='"+str(dbName)+"' user='
    postgres' host='localhost' password='postgres'")
except Exception as e:
    print e
cur = conn.cursor()

#cur.execute("CREATE TABLE IF NOT EXISTS MALICIOUS_URLS(URL TEXT)")
#cur.execute("TRUNCATE TABLE MALICIOUS_URLS")
#conn.commit()

cur.execute("SELECT IP FROM STATS OFFSET "+str(offset));
rows = cur.fetchall()
i = offset
for row in rows:
    print "*********************************"
    print i
    i += 1
    print row[0]
```

```
    r = requests.get(str(scan_url)+str(row[0]))
    resp = json.loads(r.content)
    #print resp
    if "detected_urls" in resp:
        if resp["detected_urls"] != []:
            #print resp["detected_urls"]
            for url in resp["detected_urls"]:
                print url["url"]
                cur.execute("INSERT INTO MALICIOUS_URLS(URL) VALUES
    ('" + url["url"] + "')")
                conn.commit()
```

Listing 3: Add URL's to PostgreSQL Database

```
import psycopg2

dbName = "cowrie"

try:
    conn = psycopg2.connect("dbname='"+str(dbName)+"' user='
    postgres' host='localhost' password='postgres'")
except Exception as e:
    print e
cur = conn.cursor()

with open("urls.txt") as fp:
   url = fp.readline()
   while url:
       print url
       cur.execute("INSERT INTO MALICIOUS_URLS(URL) VALUES('" + str
    (url) + "')")
       conn.commit()
       url = fp.readline()
```

Listing 4: Submit URL's for Scan to VirusTotal

```
import json
import psycopg2
from subprocess import check_output

dbName = "cowrie"
apikey = "
    f0aa0772e518487aa5daf1dcf2be2d37c9b9b361b1a0d8339faf972210b36d41
    "
scan_url = 'https://www.virustotal.com/vtapi/v2/url/scan'

offset = 805

try:
    conn = psycopg2.connect("dbname='"+str(dbName)+"' user='
    postgres' host='localhost' password='postgres'")
except Exception as e:
    print e
cur = conn.cursor()


cur.execute("SELECT URL FROM MALICIOUS_URLS ORDER BY URL OFFSET "+
    str(offset))
```

```
urls = cur.fetchall()
i=offset
for url in urls:
    print "************************************************"
    print i
    i += 1
    res = json.loads(check_output(["curl", "--request", "POST",
                                   "--url", "https://www.virustotal.
    com/vtapi/v2/url/scan",
                                   "--data", "apikey=" + apikey,
                                   "--data", "url=" + url[0]]))
    print res['scan_id']
    cur.execute("UPDATE MALICIOUS_URLS SET SCAN_URL='"+str(res['
    scan_id'])+"' WHERE URL='"+str(url[0])+"';")
    conn.commit()
```

Listing 5: Get URL Scan results from VirusTotal

```
import requests
import psycopg2
from collections import Counter
import json

dbName = "cowrie"
apikey = "
    f0aa0772e518487aa5daf1dcf2be2d37c9b9b361b1a0d8339faf972210b36d41
    "
offset = 3


def getScanResults(scan_id):
    print scan_id
    softwares = ""
    report_url = 'https://www.virustotal.com/vtapi/v2/url/report?'
    \
                 'apikey=' + str(apikey) + '&' \
                 'resource=' + str(scan_id) + '&' \
                 'allinfo=false'

    response = requests.get(report_url)
    res = response.json()
    #print json.dumps(res['scans'], indent=4, sort_keys=True)
    for r in res['scans']:
        if res['scans'][r]['detected'] == True:
            softwares += str(r) + ";"
    print softwares
    softwares = softwares[0:len(softwares)-1]
    positives = res['positives']
    total = res['total']
    cur.execute("UPDATE MALICIOUS_URLS SET "
                    "POSITIVES="+str(positives)+", "
                    "TOTAL="+str(total)+" ,"
                    "IDENTIFIED_ENGINES='"+str(softwares)+"' WHERE
    SCAN_URL='"+scan_id+"';")
    conn.commit()


```

```
try:
    conn = psycopg2.connect("dbname='"+str(dbName)+"' user='
    postgres' host='localhost' password='postgres'")
except Exception as e:
    print e
cur = conn.cursor()

cur.execute("SELECT SCAN_URL FROM MALICIOUS_URLS OFFSET "+str(
    offset))
scan_ids = cur.fetchall()
i = offset
for id in scan_ids:
    print "
    ************************************************************"
    print i
    i = i+1
    if(id[0]!=None):
        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 6: Virus Total API Request Sample

```
{
 'response_code': 1,
 'verbose_msg': 'Scan finished, scan information embedded in this
    object',
 'scan_id': '1
    db0ad7dbcec0676710ea0eaacd35d5e471d3e11944d53bcbd31f0cbd11bce31
    -1390467782',
 'permalink': 'https://www.virustotal.com/url/__urlsha256__/
    analysis/1390467782/',
 'url': 'http://www.virustotal.com/',
 'scan_date': '2014-01-23 09:03:02',
 'filescan_id': null,
 'positives': 0,
 'total': 51,
 'scans': {
    'CLEAN MX': {
      'detected': false,
      'result': 'clean site'
    },
    'MalwarePatrol': {
      'detected': false,
      'result': 'clean site'
    }
  }
}
```

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.

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 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 noe 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 7: Get Unique IPs from Elastic Search

```
import json
import requests
import psycopg2
from Globals import *


uniqueIPs = set()
dbName = "cowrie"

try:
    conn = psycopg2.connect("dbname='"+str(dbName)+"' user='
    postgres' host='localhost' password='postgres'")
except Exception as e:
    print e
cur = conn.cursor()

def createTables():
    cur.execute("CREATE TABLE IF NOT EXISTS STATS("
                "IP VARCHAR(20) PRIMARY KEY NOT NULL,"
                "LOGINATTEMPTS INT,"
                "COUNTOFCOMMANDS INT,"
                "DOSCLUSTER INT,"
                "SENTIMENT INT,"
                "COMMANDS TEXT)")

    cur.execute("CREATE TABLE IF NOT EXISTS DOS_ATTACKS("
                "IP VARCHAR(20) PRIMARY KEY NOT NULL,"
                "LOGINATTEMPTS INT,"
```

```
                    "COUNTOFCOMMANDS INT,"
                    "COMMANDS TEXT)")

    conn.commit()
    cur.execute("TRUNCATE TABLE STATS")
    conn.commit()

createTables()


url = ELASTIC_URL + str("?size=1")
response = requests.get(url)
countOfData = json.loads(response.content)["hits"]["total"]
print countOfData

size = 10000

for i in range(0,countOfData+1,size):
    url = ELASTIC_URL + str("?_source=geoip.ip&size=")+str(size)+"&
    from="+str(i)
    response = requests.get(url)
    IPs = json.loads(response.content)["hits"]["hits"]
    print len(IPs)
    for IP in IPs:
        if IP["_source"] != {}:
            ip = IP["_source"]["geoip"]["ip"]
            uniqueIPs.add(ip)
    print "Completed : from = "+str(i)+" and end = "+str(i+size)

for ip in uniqueIPs:
    cur.execute("INSERT INTO STATS(IP) VALUES('"+ip+"')")
    conn.commit()

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 8: Elastic Search API Request Sample

```
POST localhost:9200/logstash-*/_search
{
  "size" : 0,
    "aggs" : {
        "distinct_ip" : {
            "cardinality" : {
              "field" : "geoip.ip"
            }

        }
    }
}
```

Listing 9: Get Commands Executed by IPs from Elastic Search

```
import json
```

```
import requests
import psycopg2
from Globals import *


dbName = "cowrie"
IPsArray = []

try:
    conn = psycopg2.connect("dbname='"+str(dbName)+"' user='
    postgres' host='localhost' password='postgres'")
except Exception as e:
    print e
cur = conn.cursor()

def getNoOfUniqueIPsFromDB():
    cur.execute("SELECT COUNT(*) FROM STATS");
    rows = cur.fetchall()
    for row in rows:
        return row[0]

def getIPsFromDB():
    cur.execute("SELECT IP FROM STATS");
    rows = cur.fetchall()
    for row in rows:
        IPsArray.append(row[0])


print getNoOfUniqueIPsFromDB()
getIPsFromDB()

for i in range(0,getNoOfUniqueIPsFromDB()):
    tillNowCommandsExecuted = ""
    data = {
       "query": {
            "query_string" : {
                "fields" : ["src_ip","eventid"],
                "query" : ""+str(IPsArray[i])+" AND cowrie.command.
    input"
            }
        },
        "_source": ["geoip.ip","eventid","message"]
    }
    headers = {"Content-Type": "application/json"}
    url = ELASTIC_URL + str("?size=10000")
    response = requests.post(url,data=json.dumps(data),headers=
    headers)
    countOfCommands = json.loads(response.content)["hits"]["total"]
    cur.execute("UPDATE STATS SET COUNTOFCOMMANDS=" + str(
    countOfCommands) + "WHERE IP='" + IPsArray[i] + "';")
    conn.commit()
    commands = json.loads(response.content)["hits"]["hits"]
    if commands != []:
        for command in commands:
            tillNowCommandsExecuted += str(command["_source"]["
    message"].replace("CMD:","")) + " "
        print tillNowCommandsExecuted
```

```
    cur.execute("UPDATE STATS SET COMMANDS='" + str(
    tillNowCommandsExecuted.replace("'","").strip()) + "' WHERE IP
    ='" + 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 10: Get Login Attempts by IPs from Elastic Search

```
import json
import requests
import psycopg2
from Globals import *


dbName = "cowrie"
IPsArray = []

try:
    conn = psycopg2.connect("dbname='"+str(dbName)+"' user='
    postgres' host='localhost' password='postgres'")
except Exception as e:
    print e
cur = conn.cursor()

def getNoOfUniqueIPsFromDB():
    cur.execute("SELECT COUNT(*) FROM STATS");
    rows = cur.fetchall()
    for row in rows:
        return row[0]

def getIPsFromDB():
    cur.execute("SELECT IP FROM STATS");
    rows = cur.fetchall()
    for row in rows:
        IPsArray.append(row[0])


print getNoOfUniqueIPsFromDB()
getIPsFromDB()

for i in range(0,getNoOfUniqueIPsFromDB()):
    tillNowCommandsExecuted = ""
    data = {
       "query": {
            "query_string" : {
                "fields" : ["src_ip","eventid"],
                "query"  : str(IPsArray[i])+" AND cowrie.login.*"
            }
        },
        "_source": ["geoip.ip","eventid","message"]
    }
    headers = {"Content-Type": "application/json"}
    url = ELASTIC_URL + str("?size=1")
```

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

5.2.1 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'e 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 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='
       postgres' host='localhost' password='postgres'")
9  except Exception as e:
10     print e
11 cur = conn.cursor()
12
13
14
15
16 cur.execute("SELECT LOGINATTEMPTS FROM STATS WHERE COUNTOFCOMMANDS
       = 0 ORDER BY LOGINATTEMPTS DESC")
17 rows = cur.fetchall()
18 print "Login_Attempts"
19 for row in rows:
20    print(row[0])
```

Listing 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
```

```

spark = SparkSession.builder.appName("TEST").getOrCreate()
'''
Create Dataframe From CSV
'''
cluster_df = spark.read.csv("/home/iotsys3/PycharmProjects/krishna/
    Cowrie_Scripts/loginattempts.csv",
  header=True,
  inferSchema=True,
  )


'''
Removing Columns that are not needed from DataFrame
'''
#cluster_df = cluster_df.drop("id","date")


'''
Convert Data to Vector as Standardised Scaler
and Kmeans can only operate on that data
'''
vectorAssembler = VectorAssembler(
    inputCols=["Login_Attempts"],
    outputCol="features"
  )

''''''
vcluster_df = vectorAssembler.transform(cluster_df)
#vcluster_df.show()
test_df = vectorAssembler.transform(cluster_df)


kmeans = KMeans().setK(3)  # set number of clusters
kmeans = kmeans.setSeed(1)  # set start point
kmodel = kmeans.fit(vcluster_df)

centers = kmodel.clusterCenters()

pred_df = kmodel.transform(test_df)

print("\t**OUTPUT**\n\n\n")
print  vcluster_df.show()
print("Centers RK = "+ str(centers))
print(kmodel.summary.clusterSizes)
print("\n\n\n")
pred_df.show(10)
print "Center -1 " + str(centers[0])
print "Center -2 " + str(centers[2])
print "Center -3 " + str(centers[1])
```

Listing 13: Counting of DOS attacks

```
import psycopg2

dbName = "cowrie"


```

```
cluster1center = 25.21769815
cluster2center = 11354.0
cluster3center = 20776.75

try:
    conn = psycopg2.connect("dbname='"+str(dbName)+"' user='
    postgres' host='localhost' password='postgres'")
except Exception as e:
    print e
cur = conn.cursor()

def selectCluster(dis1,dis2,dis3):
    if (dis1 < dis2) and (dis1 < dis3):
        return 1
    elif (dis2 < dis1) and (dis2 < dis3):
        return 2
    else:
        return 3


cur.execute("SELECT IP,LOGINATTEMPTS FROM STATS WHERE
    COUNTOFCOMMANDS = 0 ORDER BY LOGINATTEMPTS DESC")
rows = cur.fetchall()
for row in rows:
    clusterNo = selectCluster(abs(row[1]-cluster1center),abs(row
    [1]-cluster2center),abs(row[1]-cluster3center))
    cur.execute("UPDATE STATS SET DOSCLUSTER="+str(clusterNo)+"
    WHERE IP='"+str(row[0])+"';")
    conn.commit()

countOfSevereDosAttacks = 0
countOfMediumDosAttacks = 0
countOfLowDosAttacks = 0

cur.execute("SELECT COUNT(IP) FROM STATS WHERE DOSCLUSTER=1")
rows = cur.fetchall()
for row in rows:
    countOfLowDosAttacks = row[0]
    break

cur.execute("SELECT COUNT(IP) FROM STATS WHERE DOSCLUSTER=2")
rows = cur.fetchall()
for row in rows:
    countOfMediumDosAttacks = row[0]
    break

cur.execute("SELECT COUNT(IP) FROM STATS WHERE DOSCLUSTER=3")
rows = cur.fetchall()
for row in rows:
    countOfSevereDosAttacks = row[0]
    break

print "+++++-------------------------------+"
print "++++| Severe DOS Attacks = "+str(countOfSevereDosAttacks)
print "++++| Medium DOS Attacks = "+str(countOfMediumDosAttacks)
print "++++| Low DOS Attacks    = "+str(countOfLowDosAttacks)
print "+++++-------------------------------+"
```

After the above python file is executed, DOS attacks table looks like below.

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

Table 5: DOS Attacks Clustering Table

5.2.2 Modified Naive Bayes With K-Means for Sentiment Analysis

Based on the commands executed by the

Algorithm 2 Modified Naive Bayes For Sentiment Analysis Algorithm

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

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

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

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

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

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

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

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

Total No of Words $\leftarrow N$

Unique No of Words $\leftarrow dict$

while *IP List* **not** *Empty* **do**

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

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

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

$= \prod_{i=1}^{No\ of\ Words}$ **Probability** *($Word_i$ in Command Executed/ IP is Malicious)*

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

done

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

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

****

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

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

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.*