

# Contents

<b>1</b>	<b>Introduction of Tools</b>	<b>1</b>
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
<b>2</b>	<b>Installation of Tools</b>	<b>4</b>
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
<b>3</b>	<b>Working Model</b>	<b>7</b>
<b>4</b>	<b>Monitoring System</b>	<b>8</b>
4.1	Stochastic Process	13
<b>5</b>	<b>Intrusion Detection System</b>	<b>14</b>

## 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	PostgreSQL Tables . . . . .	15
12	Intrusion Detection System . . . . .	15
13	Malicious URL's Table . . . . .	16
14	Stats Table . . . . .	20

## List of Tables

1	List of Antivirus Engines . . . . .	3
---	-------------------------------------	---

## Listings

1	Cowrie Config File . . . . .	8
2	Get Malicious URLs From Virus Total . . . . .	16
3	Get Malicious URLs From Virus Total . . . . .	17
4	Get Malicious URLs From Virus Total . . . . .	17
5	Get Malicious URLs From Virus Total . . . . .	18
6	Virus Total API Request Sample . . . . .	19

# 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

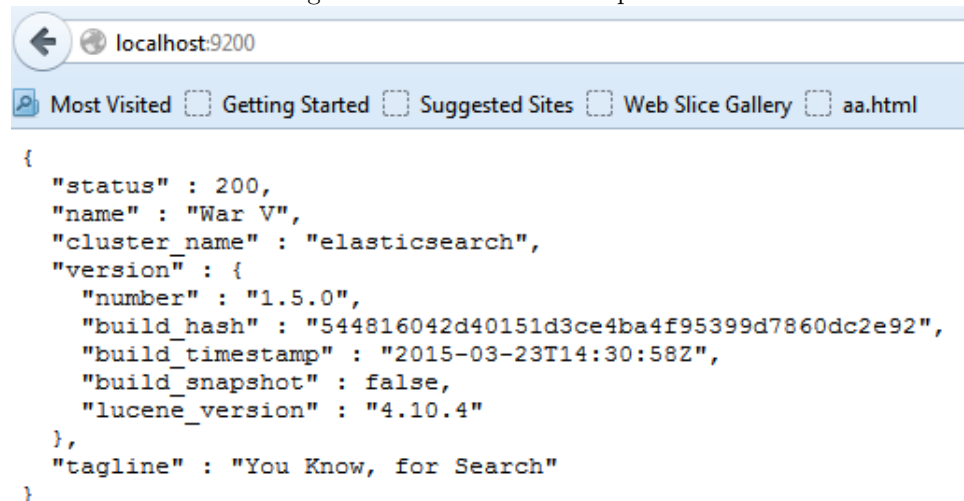
In this section, we will be mentioning about the installation of various tools which will be used in the project.

### 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 output 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.

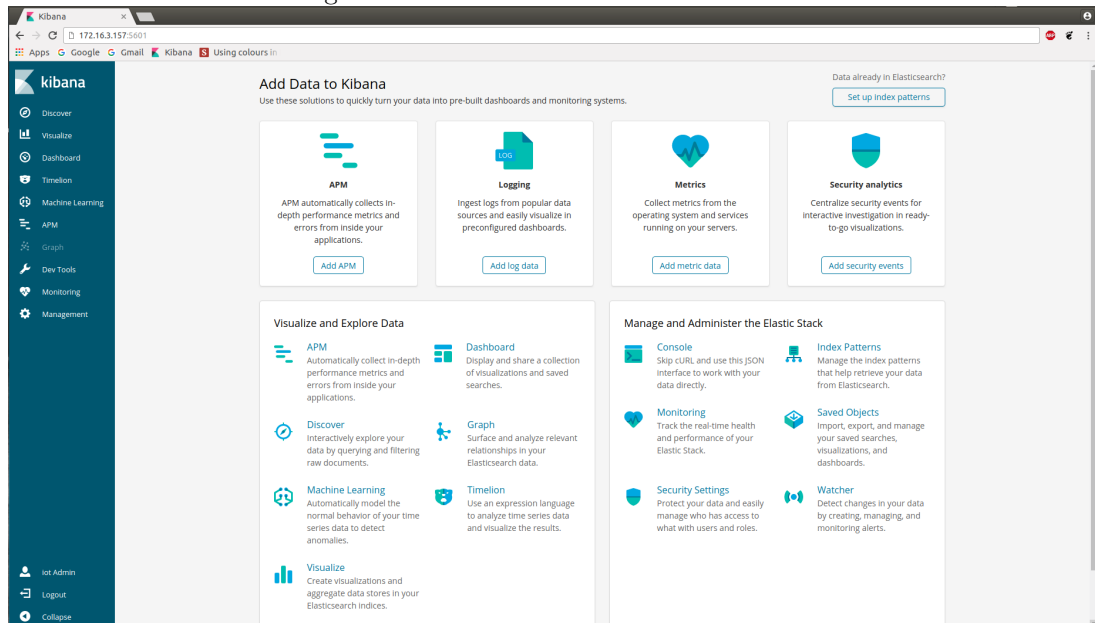
## 2.2 Logstash

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
```

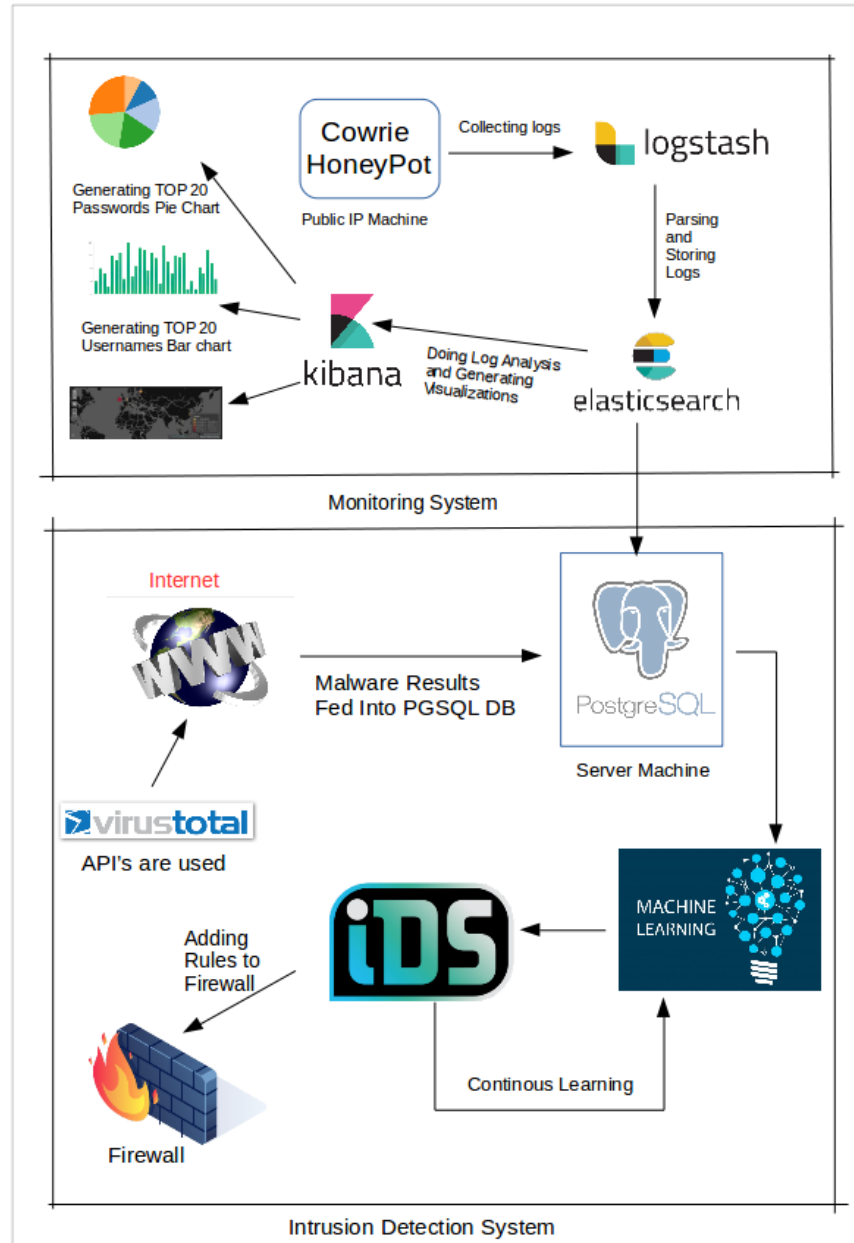
Cowrie will be up and running on port 22 which will monitor all SSH logins and commands executed by the attackers or intruders.



### 3 Working Model

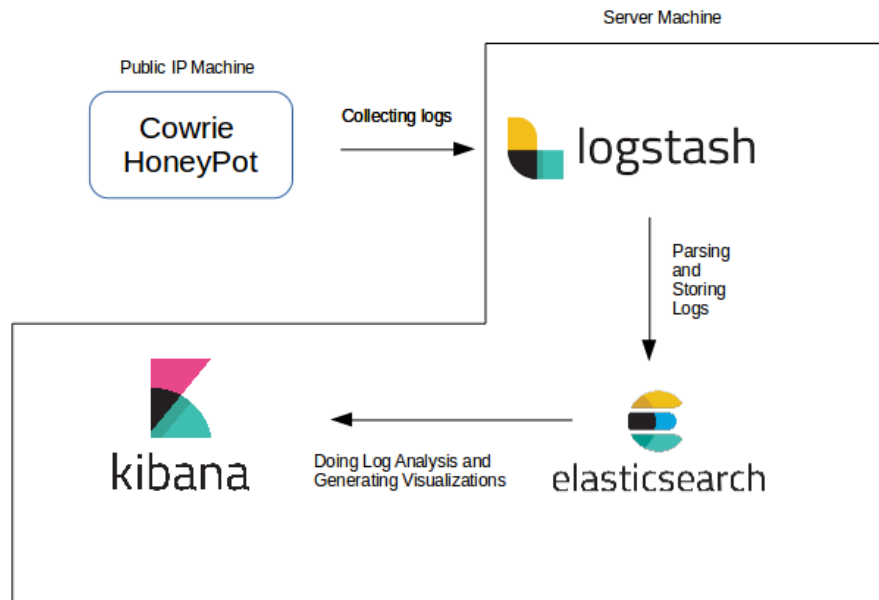
All the tools listed above are integrated to form a complete Working Model which looks like below.

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.

```
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     }

```

60 }

### Listing 1: Cowrie Config File

#### 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.

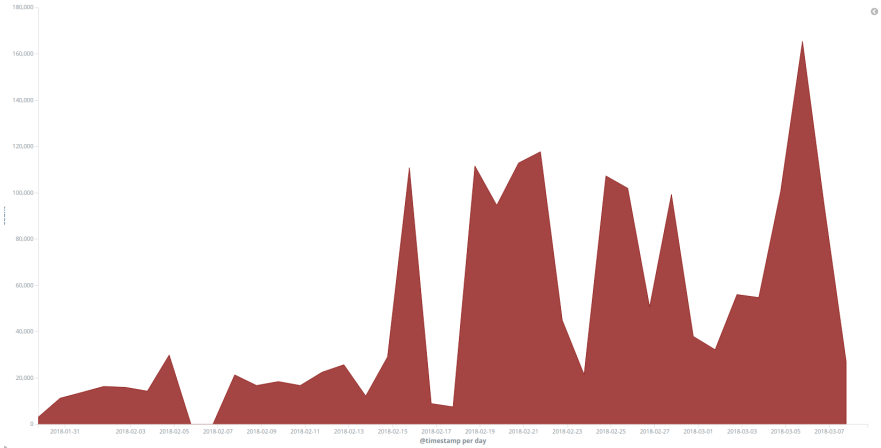
Data collected in the elastic search will be picked up by the Kibana by default as specified in the Kibana config files. Using the data visualizations are created for analyzing and monitoring the regular changes in data. Monitoring system was deployed for 40 days and collected nearly **2 Million Records**.

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.

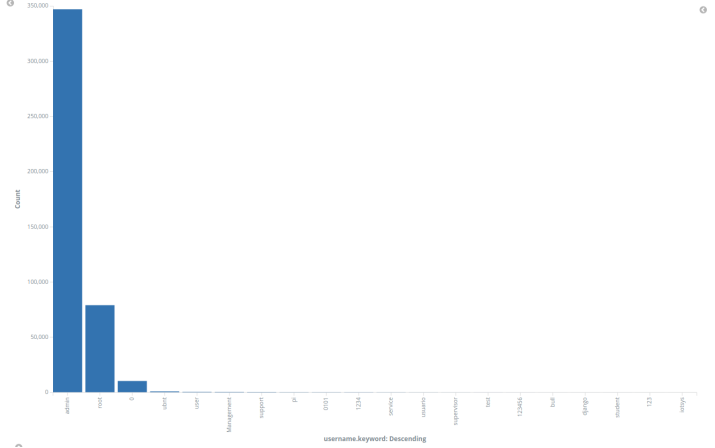
Figure 5: Daily Attacks Count Area Graph



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

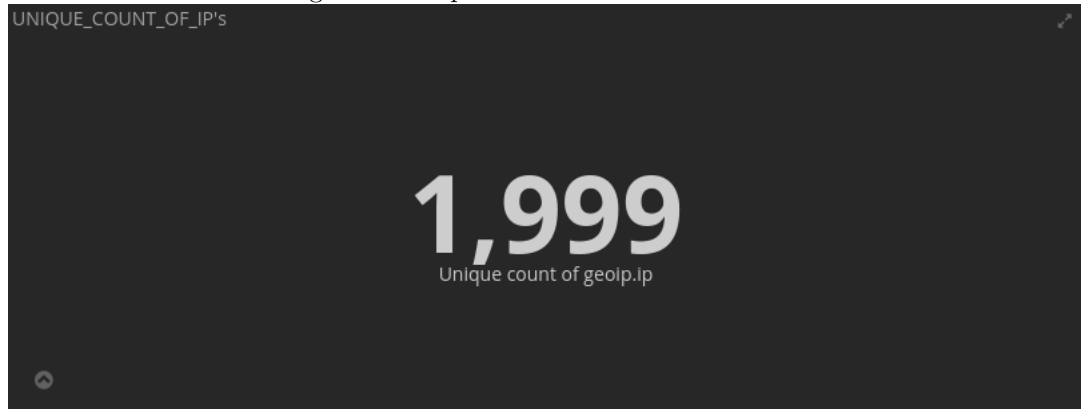


From the graph, it can be observed that usernames "admin", "root" and "ubnt" are mostly used by attackers.

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.

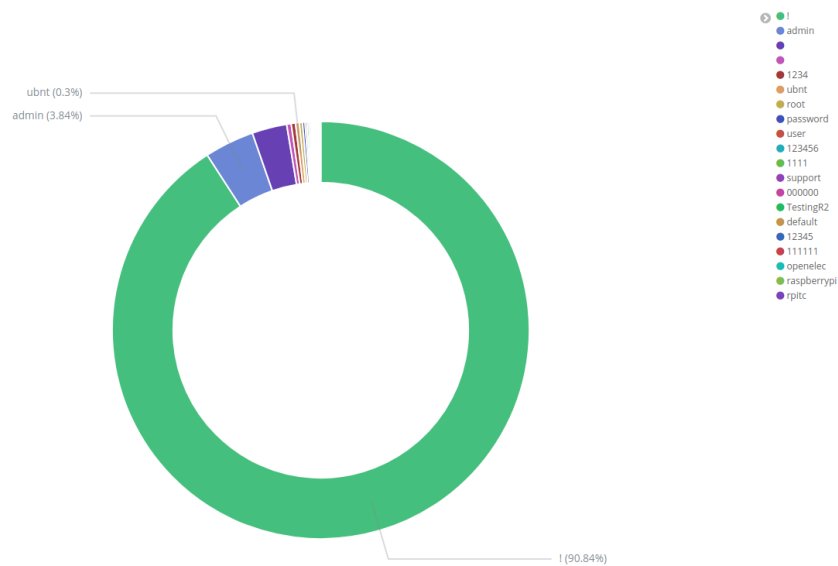
Figure 7: Unique Count of IP's



#### 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.

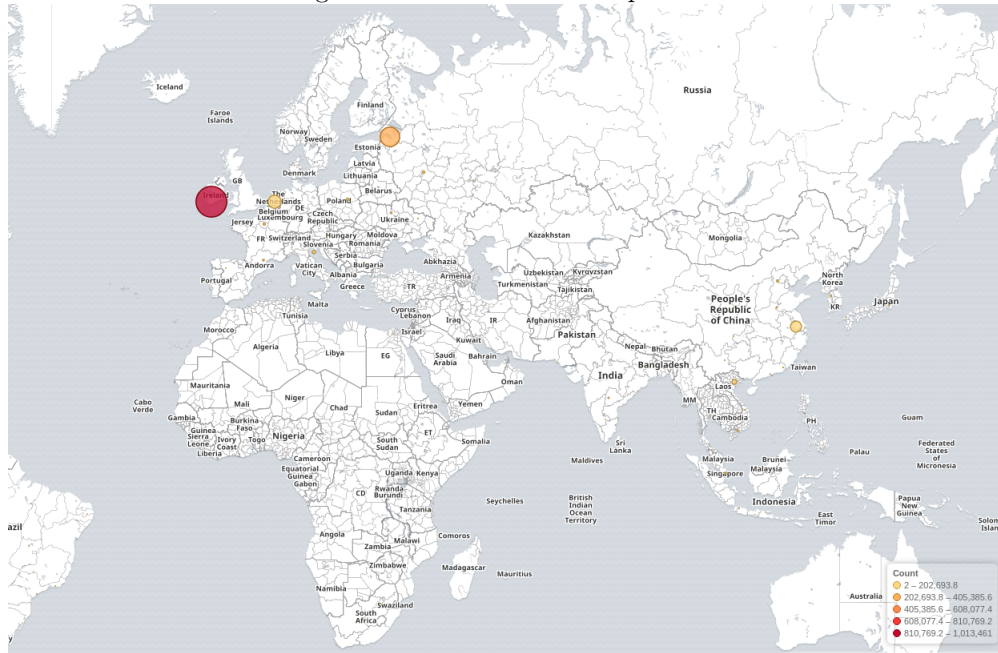
Figure 8: Top 20 Passwords Pie



## 5. Global Attacks Map

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

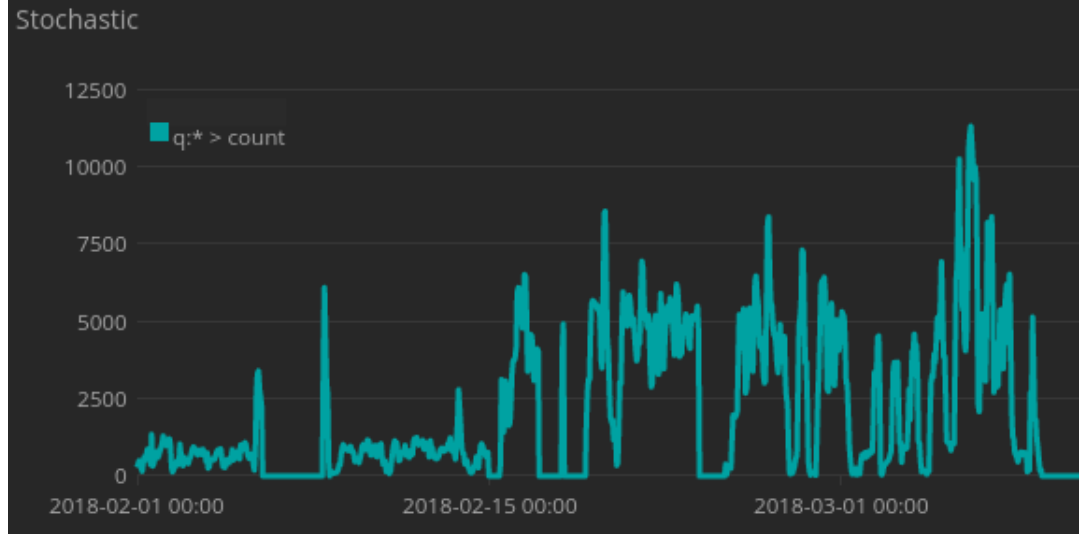
Figure 9: Global Attacks Map



### 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 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.



Figure 11: PostgreSQL Tables

```
cowrie=# \dt
```

List of relations			
Schema	Name	Type	Owner
public	dos_attacks	table	postgres
public	malicious_urls	table	postgres
public	stats	table	postgres

(3 rows)

Figure 12: Intrusion Detection System

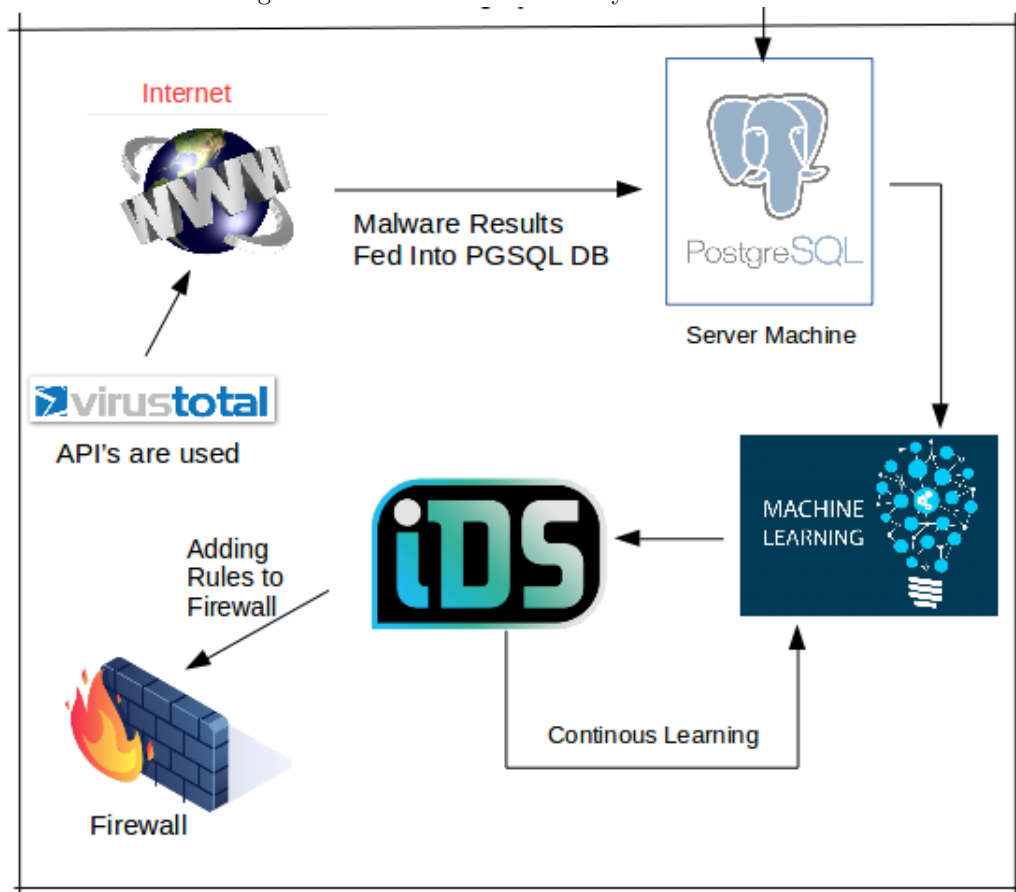


Figure 13: Malicious URL's Table

Table "public.malicious_urls"		
Column	Type	Modifiers
url	text	
scan_url	text	
positives	integer	
total	integer	
identified_engines	text	

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.

```

1 import requests
2 import json
3 import psycopg2
4
5 apikey = "
    f0aa0772e518487aa5daf1dcf2be2d37c9b9b361b1a0d8339faf972210b36d41
    "
6 scan_url = 'https://www.virustotal.com/vtapi/v2/ip-address/report?
    apikey='+apikey+'&ip='
7 uniqueURLs = set()
8
9 dbName = "cowrie"
10 offset = 1813
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 #cur.execute("CREATE TABLE IF NOT EXISTS MALICIOUS.URLS(URL TEXT)")
18 #cur.execute("TRUNCATE TABLE MALICIOUS.URLS")
19 #conn.commit()
20
21 cur.execute("SELECT IP FROM STATS OFFSET "+str(offset));
22 rows = cur.fetchall()
23 i = offset
24 for row in rows:
25     print "*****"
26     print i
27     i += 1
28     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: Get Malicious URLs From Virus Total

```

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 3: Add URL's to PostgreSQL Database

```

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.
com/vtapi/v2/url/scan",
27                                   "--data", "apikey=" + apikey,
28                                   "--data", "url=" + url[0]]))
29     print res['scan_id']
30     cur.execute("UPDATE MALICIOUS_URLS SET SCAN_URL='"+str(res['
scan_id'])+"' WHERE URL='"+str(url[0])+"'")
31     conn.commit()

```

Listing 4: Submit URL's for Scan to VirusTotal

```

1 import requests
2 import psycopg2
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])

```

Listing 5: Get URL Scan results from VirusTotal

Above python code 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.

```

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 }

```

Listing 6: Virus Total API Request Sample

This response is parsed and required 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 no of the engines in total and "identified-engines" which contains the names of all the engines in the response.

Figure 14: Stats Table

```
cowrie=# \d stats
```

Table "public.stats"		
Column	Type	Modifiers
ip	character varying(20)	not null
commands	text	
countofcommands	integer	
loginattempts	integer	
doscluster	integer	
sentiment	integer	

Indexes:

"stats\_pkey" PRIMARY KEY, btree (ip)

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.

---

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

---

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

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

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*) =

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

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*  $\leftarrow$  *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.*
-