Report Titled

THREAT INTELLIGENCE

Submitted in partial fulfillment of
the requirements of the degree of

Bachelor of Technology in Electronics Engineering

By

Soham Deshmukh	151060022
Rahul Rade	151060050
Ajay Unny	151060062
Ruturaj Nene	151060064

Under the guidance of

Dr. Faruk Kazi

Department of Electrical Engineering
Veermata Jijabai Technological Institute
Mumbai – 400019

(2018 - 2019)

DECLARATION OF STUDENT

I declare that work embodied in this project titled "**Threat Intelligence**", forms my own contribution of work under the guidance of **Dr. Faruk Kazi** at the Department of Electrical Engineering, Veermata Jijabai Technological Institute. The report reflects the work done during period 2018-19.

Soham Deshmukh (151060022)
Rahul Rade (151060050)
Ajay Unny (151060062)
Ruturaj Nene (151060064)

ACKNOWLEDGEMENT

I would like to express gratitude to all those people whose support and cooperation has been an invaluable asset during stages - I and II of this project. I would also like to thank our guide **Dr. Faruk Kazi** for guiding me throughout the duration of this project.

I am also grateful to **Dr. N. M. Singh**, Head of Department for his motivation and providing various facilities, which helped me greatly in the whole process of this stage of project.

I would also like to thank all other teaching and non-teaching staff members of the Electrical Engineering Department for directly or indirectly helping me for the completion of the project and the resources provided.

Soham Deshmukh (151060022)
Rahul Rade (151060050)
Ajay Unny (151060062)
Ruturaj Nene (151060064)

CERTIFICATE

This is to certify that Soham Deshmukh, Rahul Rade, Ajay Unny and Ruturaj Nene, students of B. Tech Electronics, Veermata Jijabai Technological Institute, Mumbai have successfully completed the project titled **"Threat Intelligence"** under the guidance of **Dr. Faruk Kazi**.

Dr. Faruk Kazi	**Dr. N. M. Singh**
Project Guide	Head, Department of Electrical Engg.

Soham Deshmukh	(151060022)
Rahul Rade	(151060050)
Ajay Unny	(151060062)
Ruturaj Nene	(151060064)

Date: **Place:**

CERTIFICATE

This is to certify that Soham Deshmukh, Rahul Rade, Ajay Unny and Ruturaj Nene, students of B. Tech Electronics, Veermata Jijabai Technological Institute, Mumbai have successfully completed the project titled **"Threat Intelligence"** under the guidance of **Dr. Faruk Kazi**.

Dr. Faruk Kazi	**External Examiner**
Project Guide	

Soham Deshmukh	(151060022)
Rahul Rade	(151060050)
Ajay Unny	(151060062)
Ruturaj Nene	(151060064)

Date: **Place:**

ABSTRACT

Big data analytics solutions, backed by machine learning and artificial intelligence, can help tackle the problems of cybersecurity breach and hacking. By employing the power of big data and machine learning, we can significantly improve cyber threat detection mechanisms. Today, cyber threat analysis is one of the emerging focus of information security. Its main functions include identifying the potential threats and predicting the nature of an attacker. Understanding the behaviour of an attacker remains one of the most important aspect of threat analysis, although much work has been focused on the detection of concrete network attacks using Intrusion Detection System to raise an alert which subsequently requires human attention. However, we think inspecting the behavioural aspect of an attacker is more intuitive in order to take necessary security measures. In this project, we propose a novel approach to analyze the behaviour of an attacker in cowrie honeypot. First, we introduce the concept of Honeypot and then model the data using semi-supervised Markov Chains and Hidden Markov Models. We evaluate the suggested methods on a dataset consisting of over a million simulated attacks on a cowrie honeypot system. Along with proposed stochastic models, we also explore the use of Long Short-Term Memory (LSTM) based model for attack sequence modelling. The LSTM based model was found to be better for modelling of long attack sequences as compared to Markov models due to their inability to capture long term dependencies. The results of these models are used to analyze different attack propagation and interaction patterns in the system and predict attacker's next action. These patterns can be used for a better understanding of the existing or evolving attacks and may also aid security experts to comprehend the mindset of an attacker. Finally, we incorporate these machine learning algorithms into a big data system utilizing Kafka streaming. The system can store large amounts of data and help analysts examine, observe, and detect irregularities within a network. The predictive models help predict and gear up for possible consequences caused by attacker and can issue an alert as soon as it sees an entry point for a cybersecurity attack.

Keywords: Cyber security, Threat Intelligence, Cowrie Honeypot, Big Data Analytics, Machine Learning, Markov Chain, Hidden Markov Models, Attacker Behavioral Analysis, Sequence Modelling using LSTM, Apache Kafka.

CONTENTS

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 Introduction

Data is more pertinent and prevalent now than it has ever been — as consumers, we're now at a 2.5-quintillion-bytes-of-data-per-day level. Threat data is no exception: Cybercriminals add to its abundance as they continuously up their game by tweaking old and creating new threats to evade detection. To address the vast amounts of threat data, security providers turn to threat intelligence based solutions, thus trying to automate processes and improve security solutions. Threat intelligence is knowledge that allows you to prevent or mitigate those attacks. Rooted in data, threat intelligence provides context — like who is attacking you, what their motivation and capabilities are, and what indicators of compromise in your systems to look for — that helps you make informed decisions about your security.

"Threat intelligence is evidence-based knowledge, including context, mechanisms, indicators, implications and action-oriented advice about an existing or emerging menace or hazard to assets. This intelligence can be used to inform decisions regarding the subject's response to that menace or hazard." — Gartner

Today, the cybersecurity industry faces numerous challenges — increasingly persistent and devious threat actors, a daily flood of data full of extraneous information and false alarms across multiple, unconnected security systems, and a serious shortage of skilled professionals. Some organizations try to incorporate threat data feeds into their network, but don't know what to do with all that extra data [26], adding to the burden of analysts who may not have the tools to decide what to prioritize and what to ignore. A cyber threat intelligence solution can address each of these issues. The best solutions use machine learning to automate data collection and processing, integrate with your existing solutions, take in unstructured data from disparate sources, and then connect the dots by providing context on indicators of compromise (IoCs) and the tactics, techniques, and procedures (TTPs) of threat actors. Threat intelligence is

actionable — it's timely, provides context, and is able to be understood by the people in charge of making decisions.

"In warfare, information is power. The better you understand your enemy, the more able you are to defeat him" [1]. In order to defeat attackers, security professionals need to learn and understand the attacker's methods, tactics, motivation and tools. Although information technology (IT) systems provide a rich set of functionalities, they are also susceptible to a more challenging set of threats each unique to the attacker. The need to protect and secure critical infrastructure has led to recent developments in threat analytic. Different threat analysis methods have been used to identify the elements of a threat against critical infrastructure assets and also analyze its behavioral patterns.

In order to execute useful response action, the deployed models should be able to detect future or ongoing intrusion attacks to the system. The current research literature mainly focuses on developing sophisticated Intrusion Response System (IRS) to satisfy this requirement [2]. The most important part determining the effectiveness of an IRS is its prediction model which helps the system to take accurate countermeasures before the intrusion happens. Thus, an accurate prediction of attacker's action will help the system to isolate and contain the attack without excessively affecting the whole network. Moreover the response should be targeted specifically to the attacker.

Predictive Threat Intelligence is one of the emerging focus of information security. In short, Predictive Threat Intelligence is based on understanding attacker behaviour and attack pattern to better predict his/her future actions. The foresight of attackers next actions, intentions and targets can help industries and organizations to take better precautionary decisions beforehand. Moreover, the foresight can expose major vulnerabilities or leaks in the existing system. This will help the industries and organizations fix unauthorized access points in their system before they can be used for intrusion.

This being an active research area, researchers are working on different ways to accurately model the temporal actions of attacker. The model developed has to be sophisticated to understand complex pattern and at the same time generalizable to

new unseen attack patterns. This can mainly lead to researchers taking two approaches:

- Anomaly detection: This is the most common approach [25][28] used, where an attack sequence or pattern is considered deviation from normal and the problem is framed as detection this deviation
- Statistical Modelling: This is a promising approach to model attacker behaviour and then using the trained model to predict attacker state. The research community has not be able to fully extract the benefit of this approach due to hurdles like lack of real attack data, lack of benchmarks for evaluating results, and difficulty in modelling complex discrete patterns.

In this work, we focus on accurately modelling the attacker behavior and predicting his next action. We propose a model based on Markov Chain and HMM for behavior analysis on cowrie honeypot data for short sequences. Later, we focus on Long Short Term Memory (LSTM) architecture to analyze long sequence data. We model the behavior of an attacker using the mentioned models and use these models to predict future behavior of an attacker. This can aid the organization in taking the necessary preventive measures through human intervention or automated system.

The key contributions of this work algorithm wise are

- Successfully modelling the behavior of an attacker who tries to penetrate a file based Cowrie honeypot system.
- Providing an estimate of the next action or step the attacker might take and thus, enable the honeypot to take appropriate actions to slow down an attacker or prevent his actions.
- Detailed analysis of the honeypot logs providing insight on attacker behavior and trends.

Motivated by the aforementioned information, this work proposes a tiered architecture that leverages several big data paradigms [21] rooted in network traffic stream processing, distributed session handling and analysis, coordinated message queuing, and distributed high-performance storage. The aim is to automate and systematize the capturing, handling, and feature-generation of both real-time and previously captured network traffic, to generate effective artifacts that would be promptly available to be employed in machine learning algorithms for anomaly

training and detection. An auxiliary added value is the capability to generate recent, highly diverse cyber security datasets, that could effectively be used for evaluation purposes as well as shared with the research community at large. In particular, we frame the projects contributions in big data architecture in the following three threads:

- Designing and prototyping a multi-tiered big data architecture, which aims at automating the generation of cyber threat artifacts that could effectively be employed for adaptive cyber threat intelligence using abundant machine learning techniques.
- Exploiting several big data paradigms, tools and technology, which enables the integration of heterogeneous network traffic to generate consolidated artifacts. Such an architecture seamlessly and effectively integrates the cyber security and data analytics communities, in an effort to further improve the adoption of machine learning in cyber security initiatives.
- Evaluating and validating the proposed architecture under three diverse case studies by employing over 100 GB of real network and malware traffic.

The generated inferences provide concrete evidence of the effectiveness of the architecture in the context of attack classification and zero day malware inference.

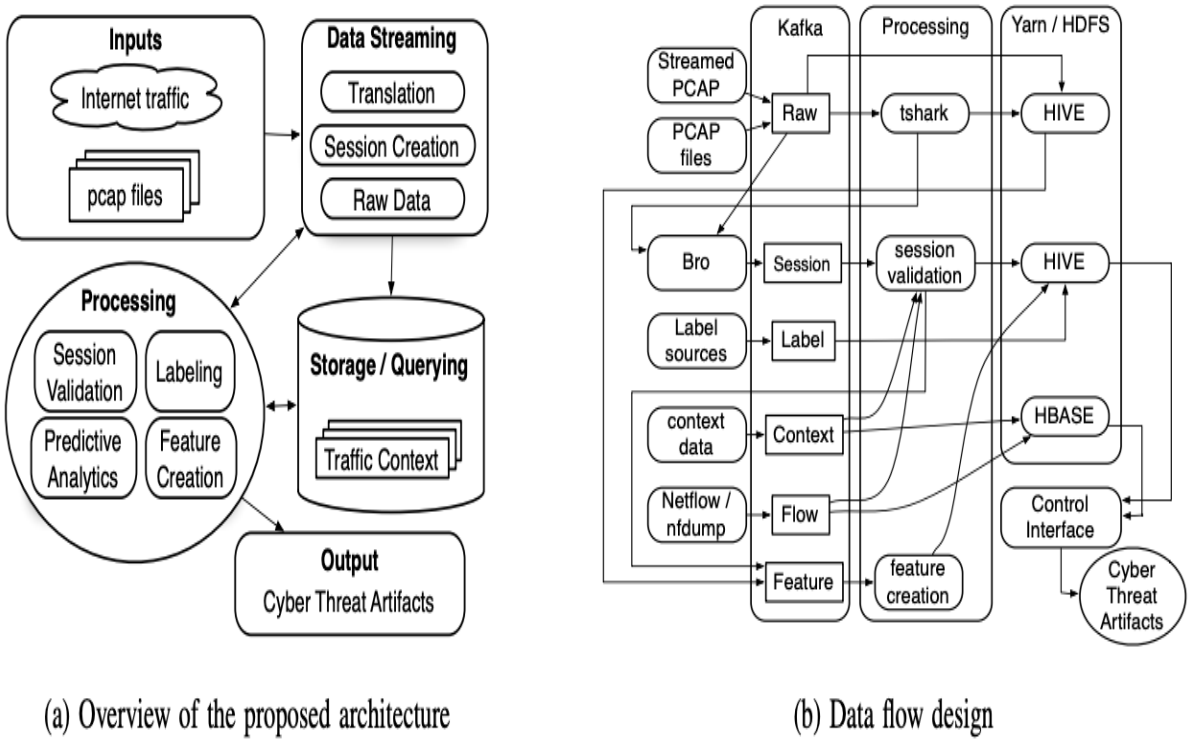

Fig. 1. Proposed System Architecture

1.2 Threat Intelligence Lifecycle

So, how does cyber threat intelligence get produced? Raw data is not the same thing as intelligence — cyber threat intelligence is the finished product that comes out of a six-part cycle of data collection, processing, and analysis. This process is a cycle because new questions and gaps in knowledge are identified during the course of developing intelligence, leading to new collection requirements being set. An effective intelligence program is iterative, becoming more refined over time.

To maximize the value of the threat intelligence you produce, it's critical that you identify your use cases and define your objectives before doing anything else.

1. Planning and Direction

The first step to producing actionable threat intelligence is to ask the right question.
The questions that best drive the creation of actionable threat intelligence focus on a single fact, event, or activity — broad, open-ended questions should usually be avoided. Prioritize your intelligence objectives based on factors like how closely they adhere to your organization's core values, how big of an impact the resulting decision will have, and how time sensitive the decision is.

One important guiding factor at this stage is understanding who will consume and benefit from the finished product — will the intelligence go to a team of analysts with technical expertise who need a quick report on a new exploit, or to an executive that's looking for a broad overview of trends to inform their security investment decisions for the next quarter?

2. Collection

The next step is to gather raw data that fulfills the requirements set in the first stage. It's best to collect data from a wide range of sources — internal ones like network event logs and records of past incident responses, and external ones from the open web, the dark web, and technical sources.

Threat data is usually thought of as lists of IoCs, such as malicious IP addresses, domains, and file hashes, but it can also include vulnerability information, such as the personally identifiable information of customers, raw code from paste sites, and text from news sources or social media.

3. Processing

Once all the raw data has been collected, you need to sort it, organizing it with metadata tags and filtering out redundant information or false positives and negatives. Today, even small organizations collect data on the order of millions of log events and hundreds of thousands of indicators every day. It's too much for human analysts to process efficiently — data collection and processing has to be automated to begin making any sense of it.

Solutions like SIEMs are a good place to start because they make it relatively easy to structure data with correlation rules that can be set up for a few different use cases, but they can only take in a limited number of data types. If you're collecting unstructured data from many different internal and external sources, you'll need a more robust solution. Recorded Future uses machine learning and natural language processing to parse text from millions of unstructured documents across seven different languages and classify them using language-independent ontologies and events, enabling analysts to perform powerful and intuitive searches that go beyond bare keywords and simple correlation rules.

4. Analysis

The next step is to make sense of the processed data. The goal of analysis is to search for potential security issues and notify the relevant teams in a format that fulfills the intelligence requirements outlined in the planning and direction stage.

Threat intelligence can take many forms depending on the initial objectives and the intended audience, but the idea is to get the data into a format that the audience will understand. This can range from simple threat lists to peer-reviewed reports.

5. Dissemination

The finished product is then distributed to its intended consumers. For threat intelligence to be actionable, it has to get to the right people at the right time. It also needs to be tracked so that there is continuity between one intelligence cycle and the next and the learning is not lost. Use ticketing systems that integrate with your other security systems to track each step of the intelligence cycle — each time a new intelligence request comes up, tickets can be submitted, written up, reviewed, and fulfilled by multiple people across different teams, all in one place.

6. Feedback

The final step is when the intelligence cycle comes full circle, making it closely related to the initial planning and direction phase. After receiving the finished intelligence product, whoever made the initial request reviews it and determines whether their questions were answered. This drives the objectives and procedures of the next intelligence cycle, again making documentation and continuity essential.

1.3 Big Data and Machine Learning for Threat Intelligence

Big data and machine learning go hand in hand in cybersecurity. Threat data provides the necessary information for cybersecurity solutions to work effectively. While big data is essential for analysis, collection and processing threat data, a large threat dataset enables a machine learning system to spot a wider variety of threats — even variants — and to decide how to best mitigate them before they infect endpoints and networks. It appears that the more data a security vendor has, the better the threat intelligence it uses in defending against cyberattacks.

Businesses are growing more digitized today. As this happens, cybersecurity threats are rising as well. Companies are placed at an increasing risk, which is why they need help from big data analysis. In fact, KuppingerCole conducted a study entitled “Big Data and Information Security.” study looks in-depth at current deployment levels and the benefits of big data security analytics solutions, as well as the challenges they face.

The Rise in Cybersecurity Threats Today

Recently, the notion of a corporate security perimeter has disappeared since more companies are adopting cloud and mobile services. Now cyber threat intelligence helps in detecting the threats within an organization. This is because traditional tools are growing outdated. Technology, however, helps businesses monitor and detect any malicious activities within their corporate network today. This paradigm shift is a good thing because today's cybercriminals are growing more advanced too. Many times they're actually working as part of an "inside job." Several recent large-scale security breaches have demonstrated this. They've also shown us that it's time we ramp up our security instead of depending on traditional approaches. Unfortunately, there are some challenges that stand in business' way here. These are important to pay

attention to since malware attacks are growing in volume and complexity. According to Data Meer, the two biggest challenges include:

- Data volume: Cybersecurity labs have to analyze the more than 300,000 potentially malicious files that are reported to them daily. With so many reports to look into, it's impossible to keep up with all of them. This places businesses at danger.
- Scalability: Unfortunately, SQL-based tools and infrastructure don't scale well. They're also quite costly to maintain.

How Big Data Analytics Fits In

With all the cybersecurity threats that happen here, businesses are fortunate to have big data analytics step in to help them. It will improve the detection of such threats. There are several ways in which this is done, including:

- Identifying changing use patterns
- Executing complex analysis so quickly it's close to real-time
- Performing complex correlations across various data sources ranging from server and application logs to network events and user activities

For any of these things to happen though, advanced analytics beyond the simple rule-based approaches must occur. Businesses will need to analyze a lot of data – both current and historic. Since big data analysis combines current analytics with security, it makes business more cyber resilient. Of course, this is something that's “new” today. In fact, you can think of it as the security industry’s response to the cyber challenges it's facing today. These are beneficial because they can collect, store and analyze huge amounts of security data across your whole enterprise in real-time. This is then enhanced by additional context data and external threat intelligence. It's analyzed with various correlation algorithms that detect anomalies and identify any possible malicious activities.

Unlike traditional solutions these tools operate in near real-time. They'll send you some security alerts. These are ranked by severity. Additional forensic details are also available, helping to simplify security analyst job. Now they can quickly and easily detect and mitigate any type of cyberattack. The combination of big data analysis and cybersecurity makes most threats unsuccessful.

All of this is a part of the new PDR paradigm that cybersecurity experts are teaching today – prevent, detect, and respond fast. With big data analytics this is indeed possible. It's now easier to overcome cyber challenges. In fact, this is something that the healthcare industry has really come to appreciate. The technology research firm, Gartner Inc. says that at least 25% of large, global, healthcare facilities have adopted big data analytics for at least one security or fraud detection use case today. Another independent analyst and consultancy firm, Ovum, advises businesses that they really do need to incorporate big data too. They say this is an important way to fight security threats.

Rethinking CyberSecurity

Bi Survey says that big data analytics gives us a reason to rethink cybersecurity today. They believe that while analytics is the key element in leveraging cyber resilience attacks are growing more persistent and advanced in nature. As this happens, there's a new, simple fact at play: Every business needs to protect itself against many different types of attacks. This is because, unfortunately, an attacker only needs one successful attempt to ruin your business. For this reason, you need to rethink all of your cybersecurity concepts and move towards the PDR paradigm.

Fortunately, this is something that most companies already realize. While they know that data is what cyber attackers are attempting to steal, these companies are also learning that data can save their business too. It's a matter of knowing how to use it correctly. With this in mind, here are some things that data analytics can do to combat cyber threats today:

- It can identify anomalies in how a device is behaving. This is important because an employee's device could be used as a Trojan horse. In layman's terms, this means the device could be used to access and steal data. Fortunately, this is something that you can stop by using big data analytics.
- It can identify anomalies in employee and contractor behavior. So, when an employee attempts to download large amounts of data, you can not only detect it but you can also stop it.
- It can detect anomalies in the network such as new threats without known signatures. By looking at many different data attributes big data can understand the nature of various attacks.

- It can analyze data to assess network vulnerabilities and risks. This allows you to eliminate serious potential sources of risk, especially when dealing with customer-identifying information.

With the growing number of cyber threats today, it's time to look into big data analysis. When combined with cybersecurity majority business will stay safe and protected from malicious attacks.

CHAPTER 2
LITERATURE SURVEY

A lot of efforts are put in understanding the user interaction with the system and finding underlying pattern. Researchers have proposed generative and discriminative models to achieve this goal. Previously one of the approach is used to understand the user browsing behaviour modelled by Markov models [3]. Also, a lot of previous work has been focused on classifying web sessions into “attack” and “normal” classes with the help of Intrusion Detection System (IDS) [22][23] wherein these web sessions were usually modelled using supervised machine learning algorithms that were used to distinguish sessions [4][5][6]. Similarly, previous works have also focused on the analysis of polymorphic worms which can be considered complementary work to our suggested method [7][8]. In some of these methods, signature models are used to identify the worms [7]. While these methods are necessary they do not provide better understanding of the nature of existing and future threats. Having the ability to predict attacker actions is of paramount importance for taking preventive rather than damage control based actions. Thus, in our work, we use Markov chain and Hidden Markov Model, along with LSTMs, to model different types of attacker behaviour that can be observed in a common file based honeypot system.

In the context of big data for network security [24], Cardenas et al. [5] discussed advances in security analytics that could exploit big data. The authors considered the fact that advances in hardware and software are enabling the analysis of large-scale, heterogeneous datasets at unprecedented scales and speeds. Further, they stated that these technologies are transforming security analytics by facilitating the storage, maintenance, and analysis of security information. The authors concluded by highlighting some challenges related to big data analytics for security, including, privacy concerns, suitable and sound approaches to handle and manage such data, and the need for visual analytics based on human-computer interactions. In another work [8], Colin Tankard focused on the security issues related to big data. The author

provided a comprehensive review of many aspects of big data, including, big data use cases, challenges facing consumers of big data, various factors surrounding the adoption of big data platforms such as Hadoop, and new opportunities provided by this paradigm. In this work, we realize the benefits of big data paradigms, which aim to automate the labeling process as well as generating imperative cyber threat artifacts. In the context of machine learning-based approaches for cyber security, existing methods commonly follow two types of approaches, namely, misuse-based detection and anomaly-based detection [7, 9]. The former compares network activities with respect to pre-discovered patterns to identify suspicious activities, where patterns are commonly captured by using supervised learning approaches [10] to build effective classifiers. The latter typically employs clustering [11][28] or association rule mining based approaches [12] to find outliers, thus flagging them as suspicious network behavior. In this work, we uniquely propose and empirically evaluate a big data architecture in diverse network scenarios through experimenting with various types of network traffic to demonstrate its effectiveness in amalgamating heterogeneous data for cyber threat intelligence purposes.

CHAPTER 3
DATA COLLECTION AND ANALYSIS

3.1 Honeypot

Honeypot is a decoy system [9] that can simulate one or many vulnerable hosts, with the intent of tricking the attacker with an easy target. The honeypot does not have any other role to fulfil and therefore all connection attempts are deemed suspicious. It can emulate the operating system services to detect attacks.The main objectives of honeypots are to divert malicious traffic away from important systems, get early warning of an imminent attack before critical systems are compromised in order for the system to be secured, and gather information about the attacker and his/her behaviour for further analysis. Honeypot is used as a bait for the attacker to carry out his/her attack on a non-critical, well-monitored and a decoy system, significant intuition can be gained into their attack methods, and subsequent information can be derived for further analysis or legal purposes. Another purpose of a honeypot is to take attackers attention from legitimate targets, and make them waste their time while the original entry point is secured.

Types of Honeypots

Honeypots are usually classified based on their deployment and based on their level of involvement. Based on deployment, honeypots may be classified as:

1. Production honeypots
2. Research honeypots

Production Honeypots: Production honeypots are placed inside the production network with other production servers to safeguard an organization or an enterprise. Normally, production honeypots are easier to deploy.

Research Honeypots: They are run to gather information about the behaviour of an attacker of targeting different networks. Although they do not add any security value to an organization, these honeypots are used to analyse the nature of threats an

organizations can face. Research honeypots are complex to deploy and maintain, are used primarily by research, military, or government organizations.

Based on design criteria, honeypots can be classified as

1. Low-interaction honeypots
2. Medium-interaction honeypots
3. High-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. Some authors classify a third category, medium-interaction honeypots, as providing expanded interaction from low-interaction honeypots but less than high-interaction systems

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 project, 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 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. If virtual machines are not available, one honeypot must be maintained for each physical computer, which can also lead to an increase of cost. Example: Honeynet.

Advantages and Disadvantages of a Honeypot

Although they require significant resources, honeypots provide significant advantages as well. Some of the benefits of using a honeypot include:

- **Collect real data**: Honeypots collect data from actual attacks and other unauthorized activities, providing analysts with a rich source of useful information.
- **Reduce false positives**: Ordinary cybersecurity detection technologies generate alerts that can include a significant volume of false positives, but honeypots reduce this volume because there is no reason for legitimate users to access them.
- **Cost-effective**: Honeypots can be good investments because they do not require high-performance resources to process large volumes of network traffic looking for attacks, because they only interact with malicious activities.
- **Encryption**: Honeypots capture malicious activity, even if an attacker is using encryption.

However, honeypots do hold several disadvantages. The most pressing issues include:

- **Data**: Honeypots only collect information when an attack occurs. Zero attempts to access the honeypot means there is no data to analyze.
- **Honeypot network**: Malicious traffic that has been captured is only collected when an attack targets the honeypot network; if attackers suspect a network is a honeypot, they will avoid it.
- **Distinguishable**: Honeypots are often distinguishable from legitimate production systems, which means experienced hackers can often differentiate a production system from a honeypot system using system fingerprinting techniques.

In this work, we use the logs generated by Cowrie honeypot for evaluating our approach.

3.2 Cowrie Honeypot

Cowrie [19] is a medium interaction SSH and Telnet honeypot designed to log brute force attacks and the shell interaction performed by the attacker.

Some interesting features:

- Fake filesystem with the ability to add/remove files. A full fake filesystem resembling a Debian 5.0 installation is included
- Possibility of adding fake file contents so the attacker can cat files such as /etc/passwd. Only minimal file contents are included
- Session logs are stored in an UML Compatible format for easy replay with original timings with the bin/playlog utility.
- Cowrie saves files downloaded with wget/curl or uploaded with SFTP and scp for later inspection
- SFTP and SCP support for file upload
- Support for SSH exec commands
- Logging of direct-tcp connection attempts (ssh proxying)
- Forward SMTP connections to SMTP Honeypot (e.g. mailoney)
- Logging in JSON format for easy processing in log management solutions
- Many, many additional commands

Using Cowrie honeypot, user activity is logged in JSON files with distinct focus on terminal inputs and messages. The default JSON logging format employed is shown below:


```
{
        "eventid": "cowrie.client.version",
        "macCS": ["hmac-sha1", "hmac-sha1-96", "hmac-md5", "hmac-md5-96",
                 "hmac-ripemd160", "hmac-ripemd160@openssh.com"],
        "timestamp": "2017-07-03T18:30:08.235671Z",
        "session": "577076dfb79e",
        "kexAlgs": ["diffie-hellman-group14-sha1",
                    "diffie-hellman-group-exchange-sha1",
              "diffie-hellman-group1-sha1"],
        "keyAlgs": ["ssh-rsa", "ssh-dss"],
        "message": "Remote SSH version: SSH-2.0-PUTTY",
        "system": "HoneyPotSSHTransport,523,116.31.116.16",
        "isError": 0,
        "src_ip":  "116.31.116.16",
        "version": "SSH-2.0-PUTTY",
        "dst_ip": "10.10.0.13",
         "compCS": ["none"],
        "sensor": "DC-NIC-Mumbai",
        "encCS": ["aes128-ctr", "aes192-ctr", "aes256-ctr", "aes256-cbc",
              "rijndael-cbc@lysator.liu.se", "aes192-cbc", "aes128-cbc",
```

```
            "blowfish-cbc", "arcfour128","arcfour",
            "cast128-cbc","3des-cbc"]
}
```

eventid is a tag given to the activity of the user, in our case this implies the hacker is checking the version of cowrie honeypot

timestamp: the exact time of login or the action being performed in

message: command prompt logs

session: current session id

src_port: the source port

system: honeypot system which is being intruded

isError: did the given action performed lead to any error

src_ip: source ip of the attacker

dst_ip: destination ip

The JSON file is parsed into a csv file for further processing and exploration.

3.3 Data Analysis

The dataset was explored for outliers and abnormalities which can subsequent affect the performance for further modelling. The following points were considered:

- The occurrence of particular ‘event id’
- The source IPs which are hitting the honeypot the most
- Most common passwords used while logging in the honeypot
- The frequency of hit on different honeypot systems
- The most common commands used in terminal

As shown in fig. 2, cowrie.command.success is the most common action being recorded indicates the actions of hacker are being completely executed. This implies that the malicious action of hackers are recorded in logs without fail. Cowrie.command.input is then the second most common action being taken. This includes the actions of deleting, downloading and injecting files into the file system.

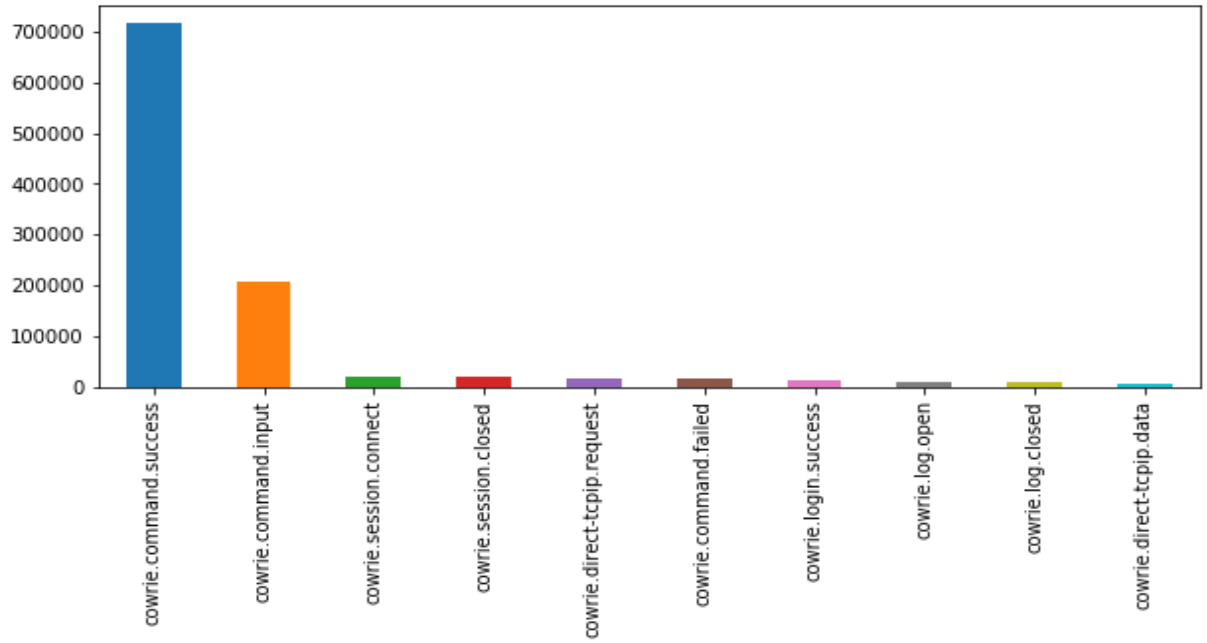

Fig. 2. Count of most occurring events in attack sessions

On further analysis of cowrie.command.input, the most common type of attack performed by hacker is a delete operation followed by input. On further analysis one can exactly determine if hackers are after some particular file or folder, and are they interested in perturbing it with virus or directly deletion.

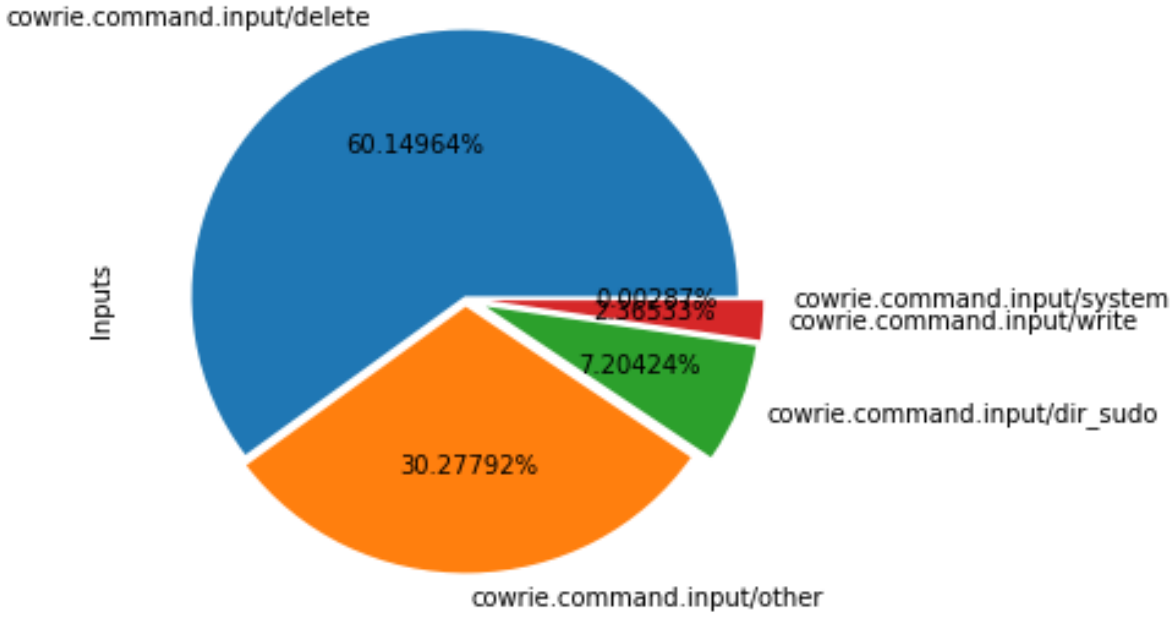

Fig. 3. Input events distribution

Fig. 4 gives an insight that majority of attacks are from a particular malicious source IP, blocking those top 20 percentage of source IP will be the first tasks to providing preliminary security to the system. Moreover, the attacks originated from 1924 distinct IP addresses spreading across 90 countries.

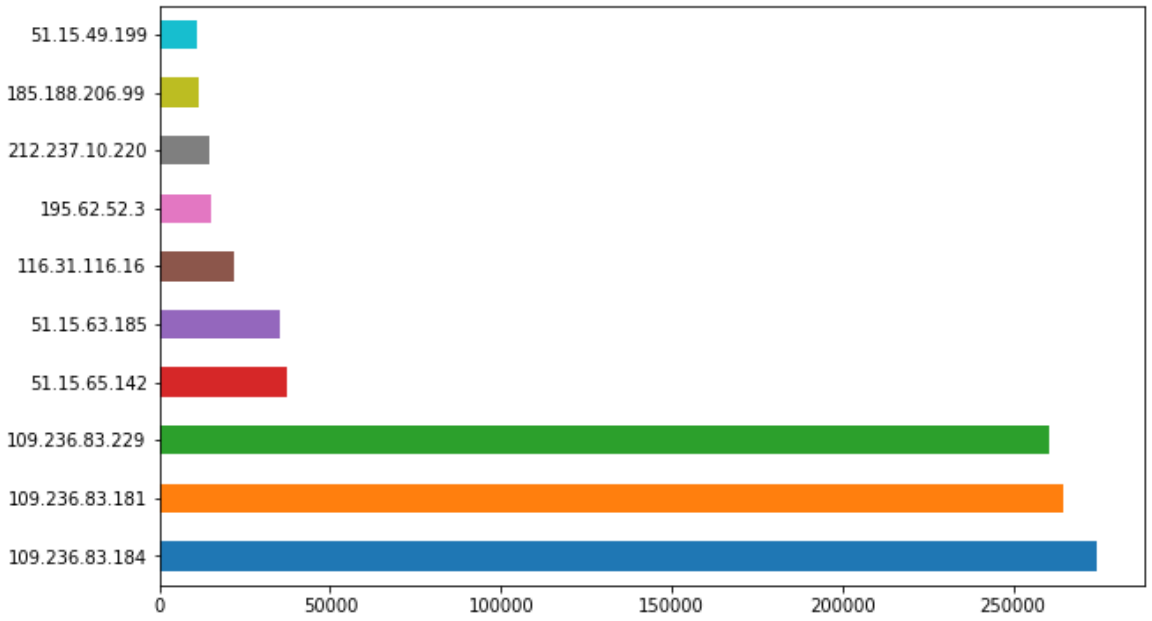

Fig. 4. Most common IPs involved in attacks

The most attacked system in the honeypot are shown in fig. 5 which provides insight on which is a higher target for hackers and the priority of securing systems.

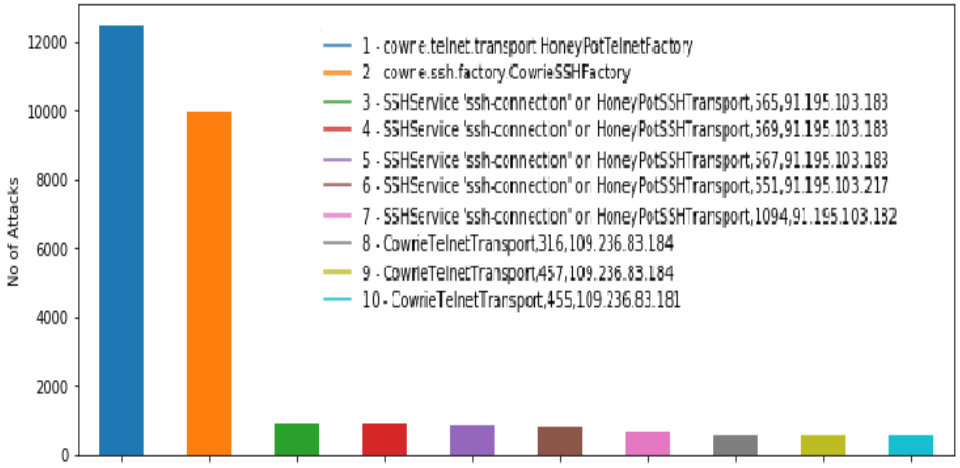

Fig. 5. Most attacked honeypot systems

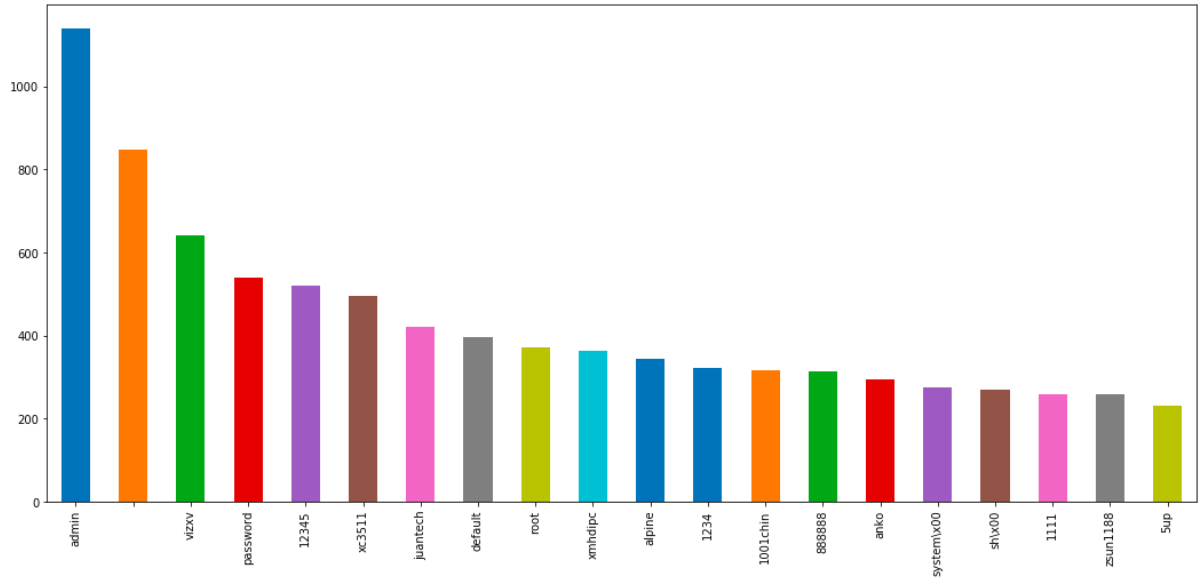

Fig. 6. The most commonly used passwords by hackers

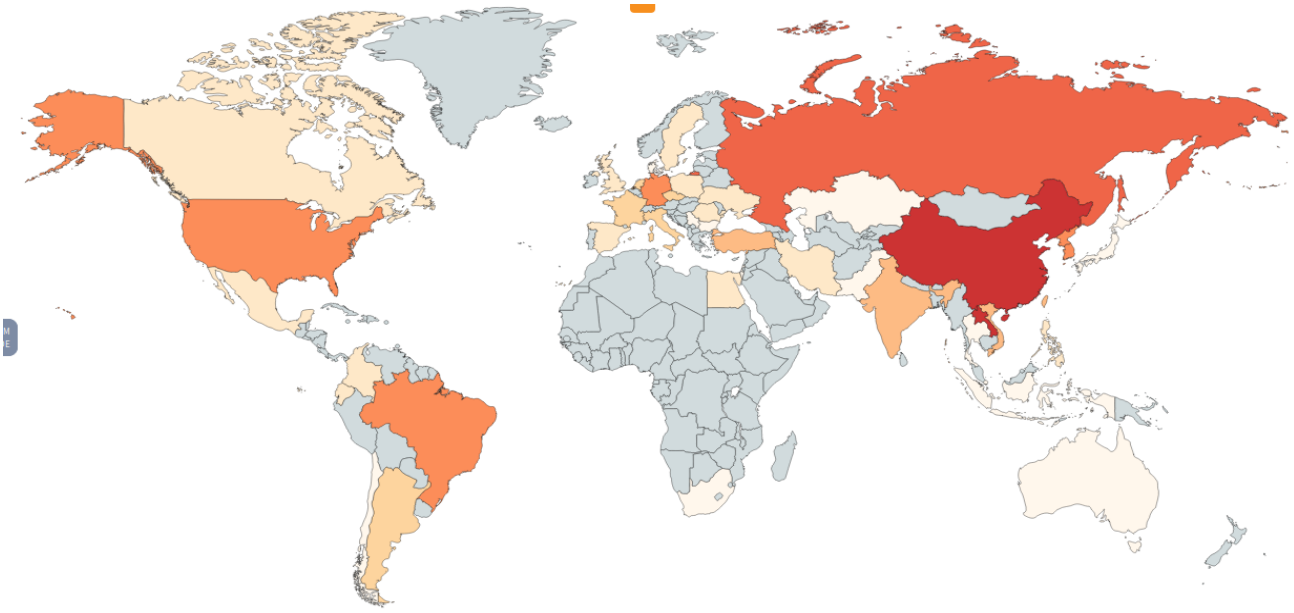

Fig. 7. Distribution of IPs across various countries

Honeypot Logs Dataset Processing for Modelling

The logged cowrie activity was modelled further into 14 states with focus on input commands. The input command was then divided into 5 commands specifically write, system, sudo, delete and other with prominently emphases the malicious behaviour of attacker. The total 19 states are given below along with their corresponding IDs.

{

0: 'cowrie.client.size',
1: 'cowrie.client.version',
2: 'cowrie.command.failed',
3: 'cowrie.command.input/delete',
4: 'cowrie.command.input/dir_sudo',
5: 'cowrie.command.input/other',
6: 'cowrie.command.input/system',
7: 'cowrie.command.input/write',
8: 'cowrie.command.success',
9: 'cowrie.direct-tcpip.data',
10: 'cowrie.direct-tcpip.request',
11: 'cowrie.log.closed',
12: 'cowrie.log.open',
13: 'cowrie.login.failed',
14: 'cowrie.login.success',
15: 'cowrie.session.closed',
16: 'cowrie.session.connect',
17: 'cowrie.session.file_download',
18: 'cowrie.session.input'

}

The data is grouped by session id for considering each sequence where each session id corresponds to the sequence of actions taken by hacker. The assumption made here is different session id are independent of individual attacker characteristics and hence dividing depending on session ID rather than source IP wouldn't affect the modelling and prediction by a large factor.

CHAPTER 4

MACHINE LEARNING FOR THREAT INTELLIGENCE

In this section, we provide a detailed review of the three models used for modelling the attack sequences occurring on Cowrie honeypot and predicting the future attack sequences which might occur on the honeypot system. These models can be used for any other file-based honeypot systems which log detailed information of the attack sequences. The overall workflow consists of three prediction parallel models and a heuristic threshold based approach as shown in figure 8 where a particular model is used for prediction depending on length of attack sequence per session.

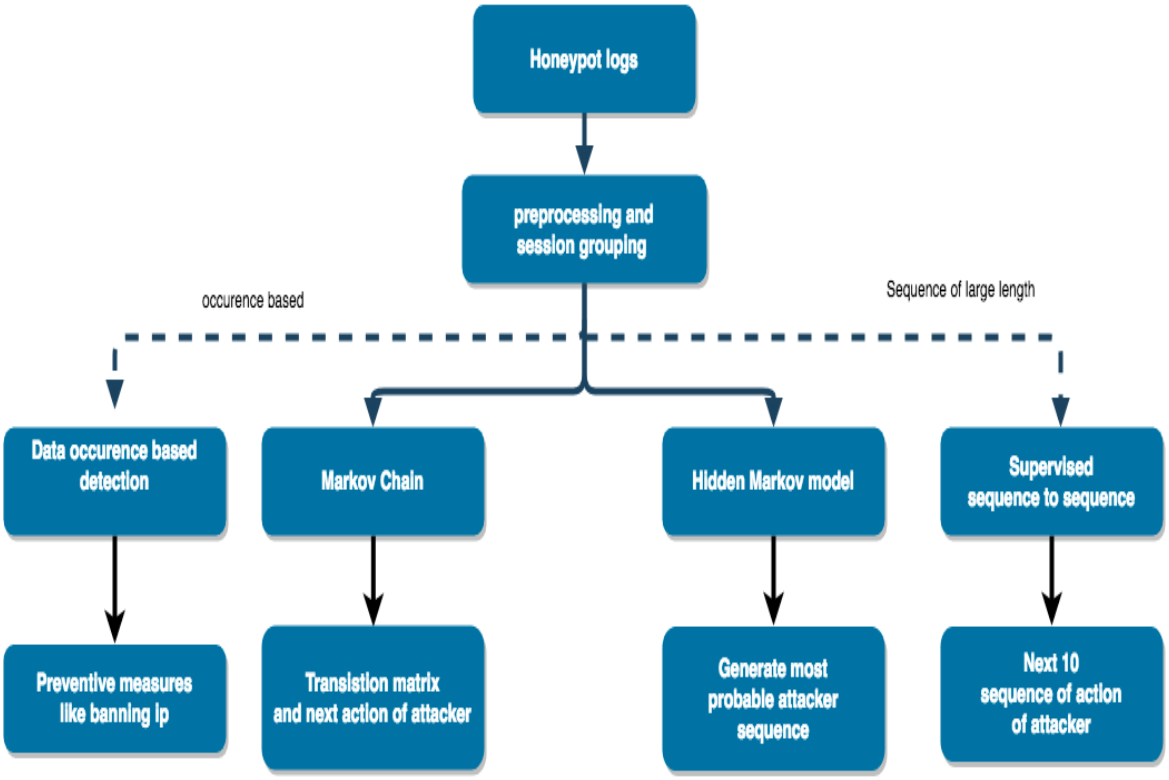

Fig. 8. Proposed Workflow

4.1 Markov Chain

A Markov chain [12] is a stochastic process that sequentially moves from one state to another in the state space. A Markov chain [13] is defined by a state space X = {x_i: i = 0, 1, ...} and the probabilities of transition between the states in the state space. In case of a Markov chain with a finite number of distinct states, these transition probabilities can be represented by a matrix that is always non-negative and where the sum of elements in each row equals one. Such a transition matrix is called one-step transition matrix. The main distinction of a Markov chain is the existence of memory which can help in efficiently modelling systems which follow a series of states, where the next state depends on the current state of the system.

In context of analyzing honeypot attack sequences, the main idea is to train a probabilistic model which efficiently models the intruder attack sequences on Cowrie honeypot. A first-order Markov chain having states $X = \{ x_i : i = 0, 1, ..., 18\}$ is being used for the purpose, where each state represents the ID of the event logged by the honeypot in response to the action taken by the attacker. According to a first order Markov chain, the probability of a state *X* at instant *t* is dependent only on the state at instant *t-1* i.e.,

$$P (X_t = x \mid X_{t-1}, X_{t-2}, X_{t-3}....X_1, X_0) = P (X_t \mid X_{t-1}) \quad (1)$$

where X_t is the state of the model at instant t and $x \in X$.

We are interested in finding the probabilities P ($X_i \mid X_j$) that if X_j is the current step which the attacker takes then the next step would be X_i. Training a Markov model involves calculating these transition probabilities using the number of transitions from state i to state j in the training sequences. Thus, the transition probability from state X_j to state X_i is calculated as

$P (X_i \mid X_j)$ = no of transitions from state *j* to state *i* / total no of transitions from state *j* to all states (2)

The resulting Markov chain reveals the sequence of steps taken by an intruder while attacking a Cowrie honeypot system. Moreover, it can be used to predict the next step which the attacker might take depending on the current step of the attacker.

4.2 Hidden Markov Model

The Hidden Markov Model (HMM) has gained popularity over the past few years. HMM [10][14] is a statistical Markov model with the unobserved state. In HMM the states are not visible, but the output depends on states that are visible which makes it fast and useful in Cybersecurity systems.

In the Markov chain, each state corresponds to an observable event, i.e. given an event, we can say with 100% accuracy which state we are in. In the HMM, the observation is a probabilistic function of the state. An HMM is basically a Markov chain where the output observation is a random variable X generated according to an output probabilistic function associated with each state. This means there is no longer a one-to-one correspondence between the observation sequence and the state sequence so you cannot know for certain the state sequence for a given observation sequence.

A Hidden Markov Model is defined by:

- $\boldsymbol{O} = \{o_1, o_2, ..., o_M\}$ - An output observation alphabet. The observation symbols correspond to the physical output of the system being modeled. $\boldsymbol{\Omega} = \{1, 2, ..., N\}$ - A set of states representing the state space. s_t is denoted as the state at time t.
- $\boldsymbol{A} = \{ a_{ij} \}$ - A transition probability matrix, where a_{ij} is the probability of taking a transition from state i to state j.
- $\boldsymbol{B} = \{ b_i(k) \}$ - An output probability distribution, where $b_i(k)$ is the probability of emitting symbol o_k when in-state i.
- $\boldsymbol{\pi} = \{ \boldsymbol{\pi}_i \}$ - An initial state distribution where $\boldsymbol{\pi}_i = P(s_0 = i), \quad 1 \leq i \leq N$

A complete specification of an HMM includes two parameters N and M, representing the total number of states and the size of the observation alphabet, the observation alphabet O, and three sets of probability measures A, B, $\boldsymbol{\pi}$. An HMM is often denoted by ϕ, where $\phi = (A, B, \boldsymbol{\pi})$. Given the definition above, three basic problems of interest must be addressed before HMMs can be applied to real-world applications:

1. **Evaluation Problem**: Given a model ϕ and a sequence of observations $X = \{ X_1, X_2, ..., X_T \}$, what is the probability that the model generates the observations, $P(X | \phi)$.

2. **Decoding Problem**: Given a model ϕ and a sequence of observations $X = \{ X_1, X_2, ..., X_T \}$, what is the most likely state sequence $S = \{ s_o, s_1, ,..., s_T \}$ in the model that produces the observations.
3. **The Learning Problem**: Given a model ϕ and a sequence of observations $X = \{ X_1, X_2, ..., X_T \}$, how can we adjust the model parameter ϕ to maximize the joint probability $\Pi_X \ P(X|\phi)$ i.e. train the model to best characterize the states and observations.

Evaluating an HMM

The aim of evaluating an HMM is to calculate the probability $P(X|\phi)$ of the observation sequence $X = \{ X_1, X_2, ..., X_T \}$, given the HMM ϕ. The steps involved in calculating $P(X|\phi)$ are shown below. The algorithm is called the forward algorithm.

To calculate the likelihood of a sequence of observations, you would expect that the underlying state sequence should also be known (since the probability of a given observation depends on the state). The forward algorithm gets around this by summing over all possible state sequences. A dynamic programming algorithm is used to do this efficiently.

You may wonder why we would want to know the probability of a sequence without knowing the underlying states? It gets used during training e.g. we want to find parameters for our HMM $\phi = (A, B, \pi)$ such that the probability of our training sequences is maximised. The Forward algorithm is used to determine the probability of an observation sequence given an HMM.

- **Step 1**: Initialisation

$$\alpha_1(i) = \pi_i b_i(X_1) \qquad 1 \leq i \leq N \tag{3}$$

- **Step 2**: Induction

$$\alpha_t(j) = \left[\sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} \right] b_j(X_t) \qquad 2 \leq t \leq T, 1 \leq j \leq N \tag{4}$$

- **Step 3**: Termination

$$P(\mathbf{X}|\mathbf{\Phi}) = \sum_{i=1}^{N} \alpha_T(i) \tag{5}$$

If the HMM must end in a particular exit state (final state s_F) then,

$$P(\mathbf{X}|\mathbf{\Phi}) = \alpha_T(s_F) \tag{6}$$

Forward Algorithm

The forward algorithm [14] can be understood to be working on a matrix of $\alpha_t(i)$ values. The values of $\alpha_t(i)$ represent the likelihood of being in state i at time t given the observations up to time t. You can visualize the alpha values like this:

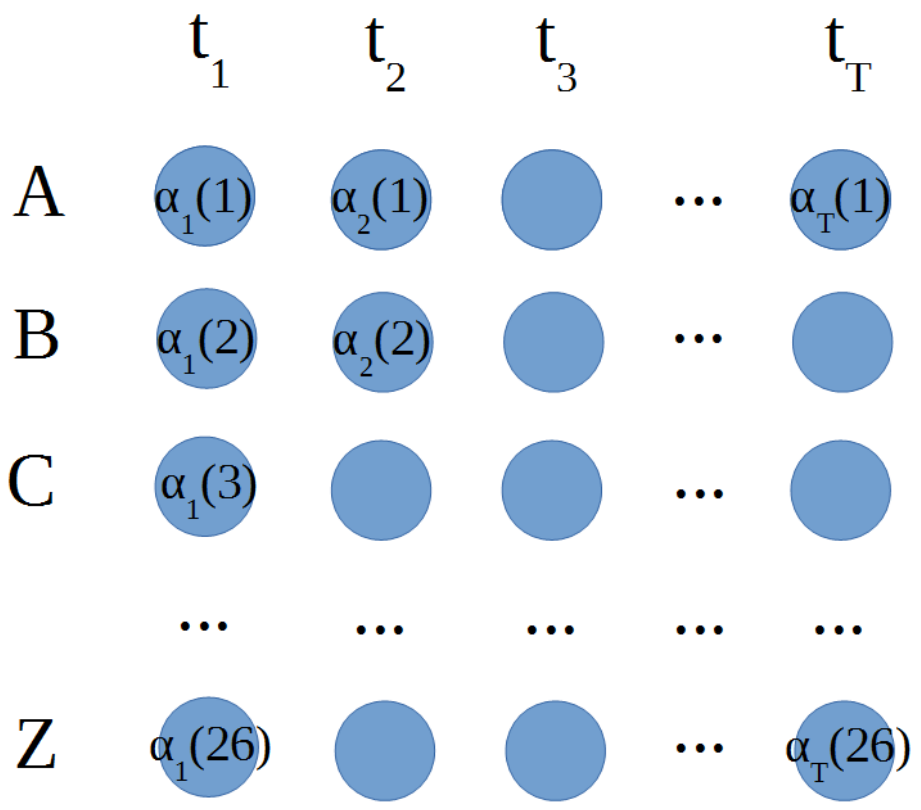

Fig. 9. The lattice of alphas calculated by the forward algorithm.

The blue circles are the values of alpha. $\alpha_1(1)$ is the top left circle etc. The initialization step from the previous section fills the first column. Since there is no previous state, the probability of being in e.g. state 1 at time 1 is just the probability of starting in state

1 times the probability of emitting observation X_1 i.e. $\alpha_1(i) = \pi_i\, b_i\, (X_1)$. Once we have done that for all the states initialisation is complete.

The induction step takes into account the previous state as well. If we know the previous state is e.g. state 2, then the likelihood of being in state 4 in the current time step is $\alpha_{t+1}(4) = \alpha_t(2)\, a_{24}\, b_4\, (X_t)$ i.e. the likelihood of being in state 2 previously, times the transition probability of going from state 2 to state 4, times the probability of seeing observation X_t in state 4. Of course, we don't know the previous state, so the forward algorithm just sums over all the previous states i.e. we marginalize over the state sequence.

Remember the job of the forward algorithm is to determine the likelihood of a particular observation sequence, regardless of state sequence. To get the likelihood of the sequence we just sum the final column of alpha values.

The Viterbi algorithm is almost identical to the forward algorithm, except it uses argmax instead of sum. This is because it wants to find the most likely previous state, instead of the total likelihood. To find the best state sequence we backtrack through the lattice, reading off the most likely previous states until we get back to the start.

Decoding an HMM

The aim of decoding [14] an HMM is to determine the most likely state sequence $S = \{ s_o, s_1, ..., s_T\}$ given the observation sequence $X = \{ X_1, X_2, ..., X_T\}$ and the HMM ϕ. The steps involved in calculating $P(X|\phi)$ are shown below. The algorithm is called the Viterbi algorithm.

The Viterbi algorithm is very similar to the forward algorithm, except it uses argmax instead of summing over all possible sequences. This is necessary to enable us to pick the most likely sequence. It would seem like argmax is discarding a lot of information and should result in suboptimal results, but in practice it works well.

- **Step 1**: Initialisation

$$V_1(i) = \pi_i b_i(X_1) \qquad 1 \leq i \leq N$$
$$B_1(i) = 0 \tag{7}$$

- **Step 2**: Induction

$$V_t(j) = \max_{1 \le i \le N} [V_{t-1}(i) a_{ij}] \, b_j(X_t) \qquad 2 \le t \le T, 1 \le j \le N$$

$$B_t(j) = \underset{1 \le i \le N}{\operatorname{argmax}} [V_{t-1}(i) a_{ij}] \qquad 2 \le t \le T, 1 \le j \le N \tag{8}$$

- **Step 3**: Termination

$$\text{best score} = \max_{1 \le i \le N} [V_t(i)] \tag{9}$$

$$s_T^* = \underset{1 \le i \le N}{\operatorname{argmax}} [B_t(i)] \tag{10}$$

- **Step 4**: Backtracking

$$s_t^* = B_{t+1}(s_{t+1}^*) \qquad t = T-1, T-1, \ldots, 1 \tag{11}$$

$$\mathbf{S}^* = (s_1^*, s_2^*, \ldots, s_T^*,) \quad \text{is the best sequence} \tag{12}$$

Estimating HMM Parameters

Estimating the HMM parameters is the most difficult of the three problems because there is no known analytical method that maximises the joint probability of the training data in a closed form. Instead, the problem can be solved by the iterative Baum-Welch algorithm, also known as the forward-backward algorithm. The forward-backward algorithm is a type of Expectation Maximisation algorithm.

The idea is to find values for $\phi = (A, B, \pi)$ such that the probability of the training observations (calculated using the forward algorithm) is maximised. We perturb the parameters until they can no longer be improved. I won't go into the details here, an outline is given below.

1. Initialisation. Choose an initial estimate ϕ.
2. E-step. Compute auxiliary function $Q(\phi, \phi')$ based on ϕ.
3. M-step. Compute ϕ' according to the re-estimation equations on page 392 of Spoken Language Processing to maximise Q.
4. Iteration. Set $\phi = \phi'$, repeat from step 2 until convergence.

4.3 Generative Sequence modelling using LSTM

Recurrent Neural Network

In the conventional feed-forward neural networks, all test cases are considered to be independent. Recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor.

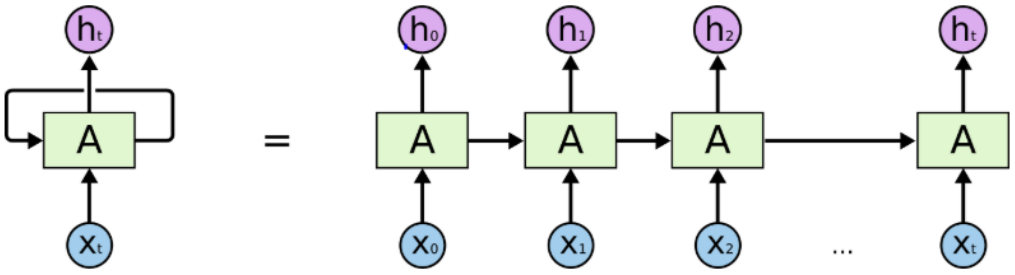

Fig. 10. RNN

In the above diagram, a chunk of neural network, *A*, looks at some input x_t and outputs a value h_t. A loop allows information to be passed from one step of the network to the next. Recurrent Neural Networks work just fine when we are dealing with short-term dependencies. The relevant information may be separated from the point where it is needed, by a huge load of irrelevant data. The Recurrent Neural Network fails in such cases [17].

The reason behind this is the problem of Vanishing Gradient [20]. A conventional feed-forward neural network, the weight updating that is applied on a particular layer is a multiple of the learning rate, the error term from the previous layer and the input to that layer. Thus, the error term for a particular layer is somewhere a product of all previous layers' errors. When dealing with activation functions like the sigmoid function, the small values of its derivatives (occurring in the error function) gets multiplied multiple times as we move towards the starting layers. As a result of this, the gradient almost vanishes as we move towards the starting layers, and it becomes difficult to train these layers.

A similar case is observed in Recurrent Neural Networks. RNN remembers things for just small durations of time [17], i.e. if we need the information after a small time it may be reproducible, but once a lot of states are fed in, this information gets lost somewhere. This issue can be resolved by applying a slightly tweaked version of RNNs – the Long Short-Term Memory Networks.

Markov chain or hidden Markov model fall short on modelling sequences of large number of actions. Long Short-Term Memory (LSTM) or Long Short-term memory is a modified version of recurrent neural network introduced by Hochreiter & Schmidhuber[16]. LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. It forms a chain like structure where each cell can be assumed to be a time frame [16]. LSTM solves the vanishing gradient problem commonly faced in RNN which allows it to remember long sequences because LSTM architecture allows disabling of writing to a cell by turning off the gate, thus preventing any changes to the contents of the cell over many cycles [15] [18].

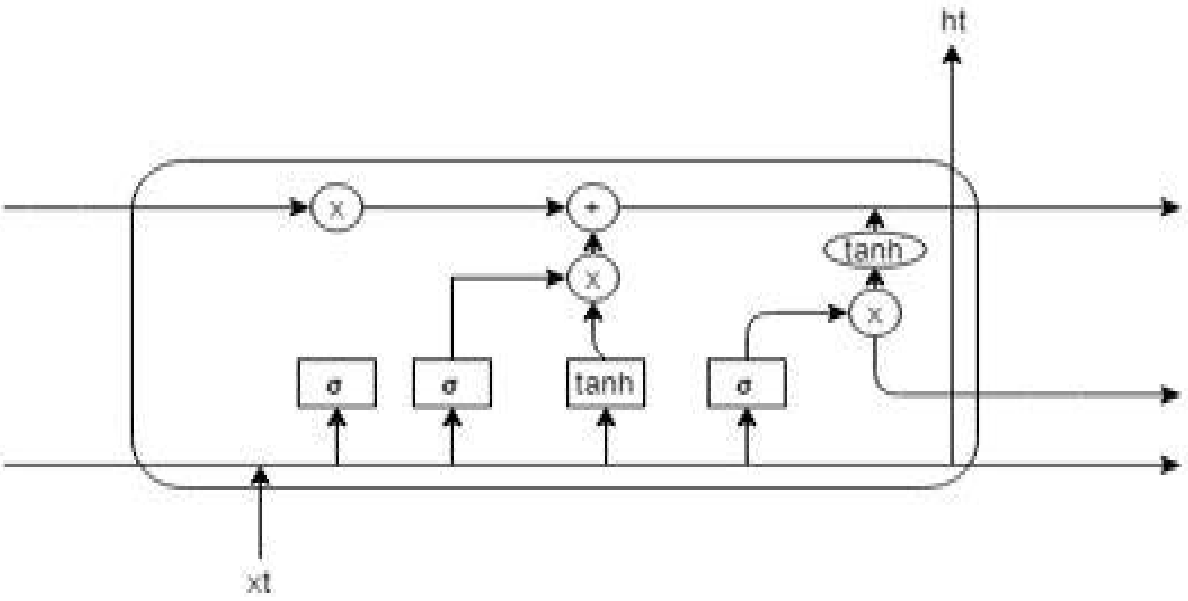

Fig. 11. LSTM cell

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \quad (13)$$

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \quad (14)$$

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \quad (15)$$

$$c_t = f_t \times c_{t-1} + i_t \times \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \quad (16)$$

$$h_t = o_t \times \sigma_h(c_t) \tag{17}$$

where,

x_t: input vector to the LSTM unit

f_t: forget gate's activation vector

i_t: input gate's activation vector

o_t: output gate's activation vector

h_t: output vector of the LSTM unit

c_t: cell state vector

These long-term dependencies can greatly influence the predictions of future action as most proficient hacker lurk in the system doing benign actions for a long time. This make it essential to be able to capture this sequence of actions, as a false negative in this case would greatly affect the system. Formatting and modelling data in a supervised manner was essential for working of LSTM. The processing of data was done as follows:

1. Each attack sequence consisting of various actions was padded to a fixed length of hundred and ten future actions were considered as output. A sliding window approach was used for maximum utilization of data.
2. Input sequences of action less than hundred and output sequences of action less than ten were padded with 0.
3. The output was then normalized between one and zero and 20% percent of data was reserved as testing data.

The architecture of model consisted of two LSTM layers, where the output of first LSTM is fed to second LSTM. In the second layer of LSTM, the outputs of previous cells are ignored and only the output of last cell is taken. The output activation function is tanh and returns a output of dimension (1, 10). The problem was framed as regression instead of classification. Mean Squared Error is used as loss function and the model is trained of 1000 epochs with a batch size of 256. For evaluating the model, the predicted model score was floored and then the predicted value and truth value were used to compute accuracy. Hyperparameter like sequence length played a important role in the ability of LSTM to predict future sequence length.

4.4 Experiments

In this section, we present the results from applying the suggested methods on a massive data set of honeypot attack sequences. The dataset was extracted from the logs which the Cowrie honeypot generated from April 2017 to July 2017. Based on these logs, we generated 22,499 distinct attack sessions involving the series of steps taken by a particular source ip starting from cowrie.session.connect to cowrie.session.closed. The attack sessions lasted from 2 steps to over 1400 steps. However, when we further investigated the sessions, we observed that over 30% sessions lasted for more than 20 steps. These 30% sequences are of major significance for the system administrators as they were found to be involving some alarming actions such as deleting system files and trying to search system information.

First, we trained a first order Markov chain on the honeypot attack sessions considering 19 different events as 19 possible states forming the state space of the chain. Investigating the transition probabilities generated by the Markov chain, we produced a graph with 19 nodes and 69 edges with corresponding non-zero transition probabilities. The state propagation graph generated is shown in fig. 12.

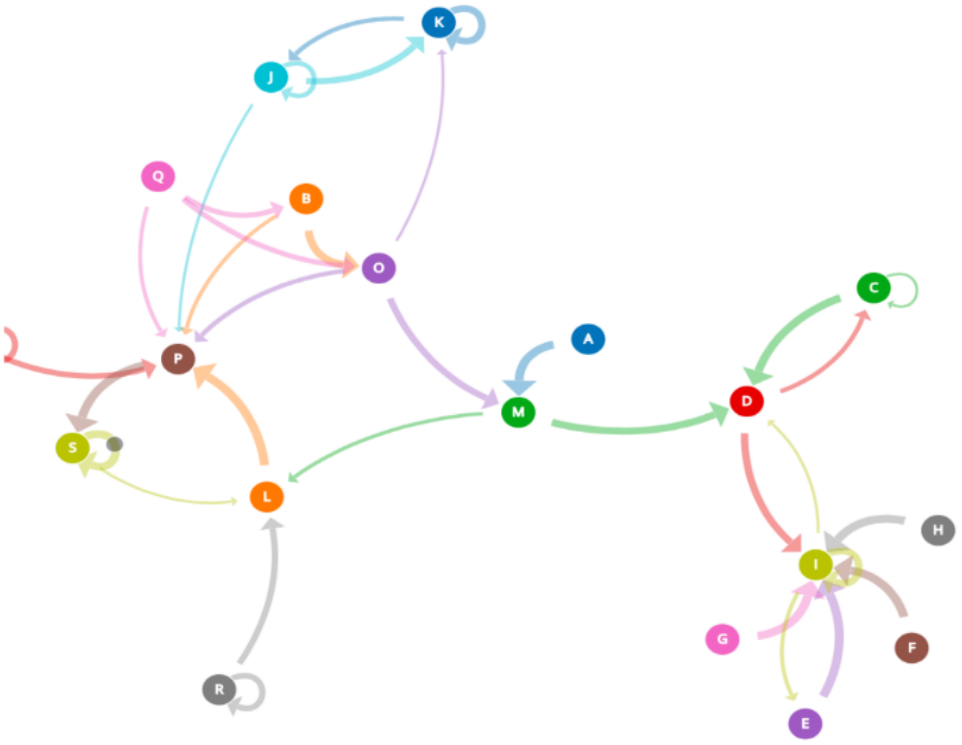

Fig. 12. Attack Propagation Graph generated using Markov chain

A high transition probability of 0.74 from state 3 (cowrie.command.input/delete) to state 8 (cowrie.command.success) reveals that the attackers are generally successful in deleting some files on the honeypot system. Although majority of the states in the attack sessions involve success and input states, only 2.12% were found to be involving downloading files on the honeypot system while 1% states involved deleting files on the system. In addition to these statistics, the Markov chain also provides an insight of the further actions the attacker is about to take.

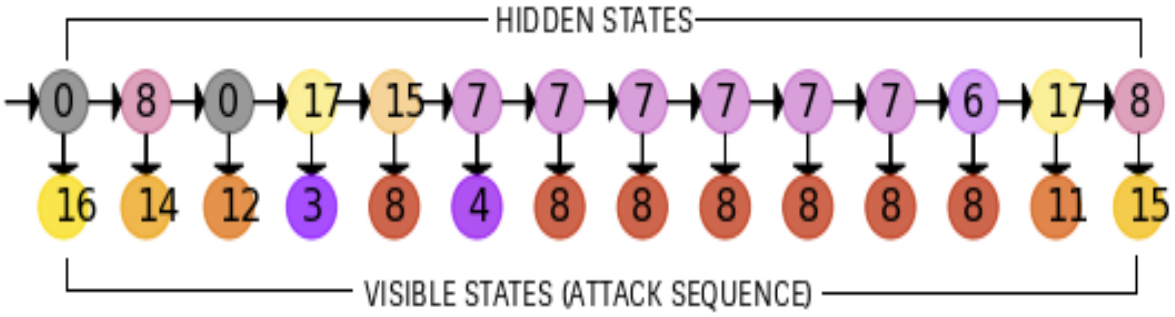

Fig. 13. Most probable sequence of length 14 generated by HMM

The results of Hidden Markov Model were similar to that obtained from Markov Chain. But testing on real time logs of attacker actions for 1 month (which we did not use for training) showed a prediction accuracy of 77% as compared to Markov chain which had 72% accuracy. The most probable attack sequence of length 14 generated by HMM is presented in fig. 13 along with the corresponding sequence of hidden states. Fig. 14 shows the convergence of the Hidden Markov Model being trained using expectation maximization algorithm.

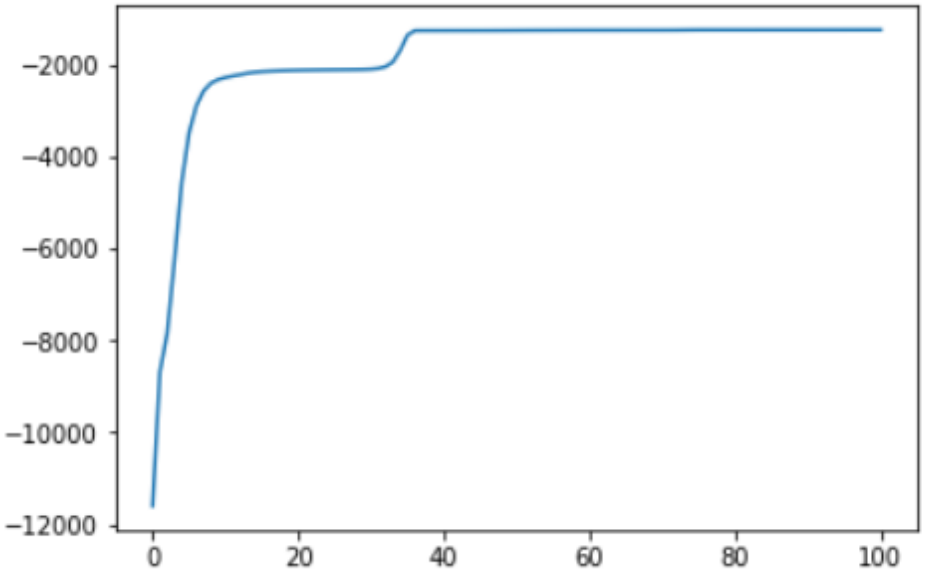

Fig. 14. The logarithmic probabilities every iteration

The accuracies of these two stochastic approaches along with that of LSTM based generative model are presented in table I.

Table I. Model used vs Percentage Accuracy

Model	Accuracy
Markov chain	72
HMM	77
LSTM	86

It can be seen that LSTM outperforms Markov chain and Hidden Markov Model on large sequences of actions. Given previous 100 actions, LSTM sequence model predicts the future 10 states in time with an accuracy of 86 to 74 percent for 1st event to 10th event respectively. The graph in fig. 15 shows train and test accuracy in predicting 10 states in future. The red plot shows accuracy of training data and blue plot shows accuracy of test data. Here timeframe refers the states prediction in future time.

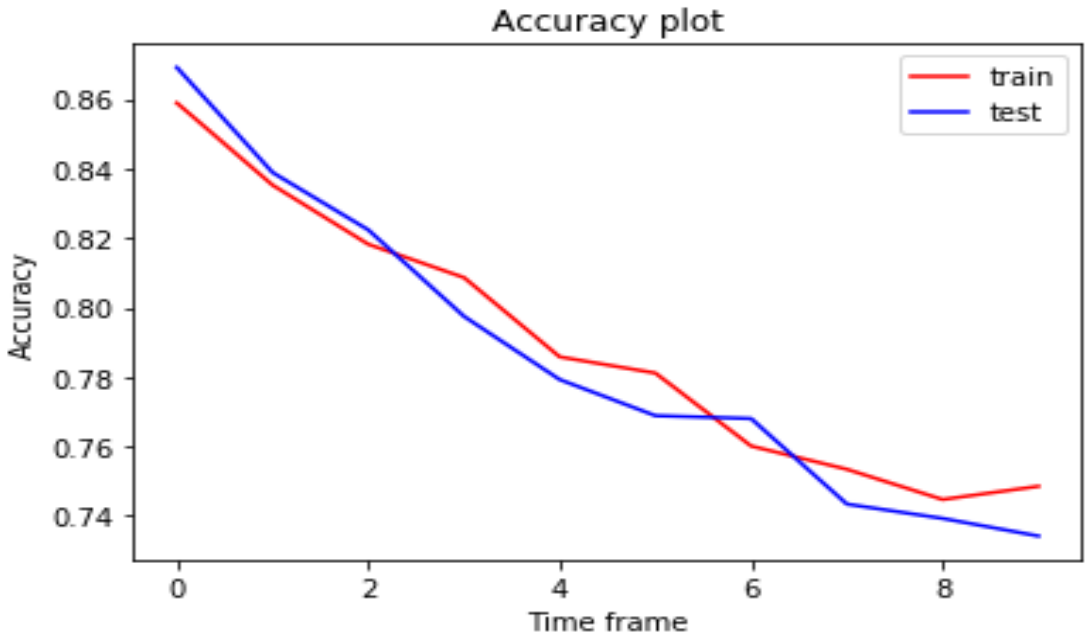

Fig. 15. Accuracy Plot

An added advantage of LSTM based model is that it can predict 10 future states with a much greater accuracy; even the accuracy for 10th future state predicted by LSTM is comparable to the accuracy for 1st future state predicted by Markov chain. For example, the future 10 states predicted by LSTM based model for an input sequence [16, 14, 12, 3, 3, 8, 3, 2, 3, 2, 3, 8, 5, 8, 8, 8, 8, 8, 5, 8, 8, 5] are [8, 8, 6, 8, 8, 6, 8, 8, 5, 7], whereas the actual 10 stated are [8, 8, 5, 8, 8, 5, 8, 8, 5, 8].

The predictions of these models along with the actual attack sequences are shown below.

Actual Attack Seq.	[16, 14, 12, 3, 8, 3, 2, 3, 2, 3, 8, 3, 8, 3, 8, 8, 8, 3, 8, 8, 8, 11, 15]
Markov Chain	[16, 14, 12, 3, 8, 8, 8, 3, 8, 3, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 15]
HMM	[16, 14, 12, 3, 8, 3, 8, 3, 8, 3, 8, 3, 8, 3, 8, 8, 8, 3, 8, 8, 8, 3, 15]
LSTM	[16, 9, 11, 3, 8, 3, 2, 3, 2, 3, 8, 3, 8, 3, 8, 8, 8, 3, 8, 8, 8, 3, 12]

Actual Attack Seq.	[16, 14, 12, 3, 8, 3, 2, 3, 2, 3, 8, 3, 8, 3, 8, 8, 8, 3, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 3, 8, 4, 8, 8,

	8, 4, 8, 8, 8, 4, 8, 8, 8, 4, 8, 8, 8, 4, 8, 8, 8, 4, 8, 8, 8, 4, 8, 8, 8, 4, 8, 8, 8, 4, 8, 8, 8, 4, 8, 8, 8, 4, 8, 8, 8, 4, 8, 8, 8, 4, 8, 8, 8, 4, 8, 8, 8, 5, 8, 7, 8, 8, 8, 8, 8, 8, 3, 8, 8, 3, 8, 11, 15]
Markov Chain	[16, 14, 12, 3, 8, 8, 8, 3, 8, 3, 8, 15]
HMM	[16, 3, 3, 3, 8, 3, 8, 3, 8, 3, 8, 3, 8, 3, 8, 3, 8, 4, 8, 3, 8, 4, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 4, 8, 3, 8, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 4, 8, 3, 8, 3, 8, 8, 3, 3]
LSTM	[16, 9, 11, 4, 8, 3, 2, 3, 2, 3, 8, 3, 8, 4, 8, 8, 8, 3, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 4, 8, 8, 8, 8, 8, 8, 3, 8, 5, 8, 8, 8, 4, 8, 8, 8, 4, 8, 8, 8, 4, 8, 8, 8, 4, 8, 8, 8, 4, 8, 8, 8, 4, 8, 8, 8, 4, 8, 8, 8, 4, 8, 8, 8, 4, 8, 8, 8, 4, 8, 8, 8, 4, 8, 8, 8, 4, 8, 8, 8, 4, 8, 8, 8, 4, 8, 8, 8, 5, 8, 8, 8, 8, 8, 8, 8, 8, 4, 8, 8, 3, 8, 10, 11]

The effectiveness of LSTM for longer sequences is evident from the above examples. Additionally, it can be seen that HMM performs considerably better than Markov chain which gets biased towards high probability states. Moreover, the predictions of these models can provide a significant aid for taking preventive measures for improving the security of file based systems.

CHAPTER 5
BIG DATA FOR THREAT INTELLIGENCE

5.1 Basics of Kafka

Apache Kafka [29][30] is a publish/subscribe messaging system designed to solve this problem. It is often described as a “distributed commit log”. A filesystem or database commit log is designed to provide a durable record of all transactions so that they can be replayed to consistently build the state of a system. Similarly, data within Kafka is stored durably, in order, and can be read deterministically. In addition, the data can be distributed within the system to provide additional protections against failures, as well as significant opportunities for scaling performance.

Messages and Batches

The unit of data within Kafka is called a message [30]. If you are approaching Kafka from a database background, you can think of this as similar to a row or a record. A message is simply an array of bytes, as far as Kafka is concerned, so the data contained within it does not have a specific format or meaning to Kafka. Messages can have an optional bit of metadata which is referred to as a key. The key is also a byte array, and as with the message, has no specific meaning to Kafka. Keys are used when messages are to be written to partitions in a more controlled manner. The simplest such scheme is to treat partitions as a hash ring, and assure that messages with the same key are always written to the same partition.

For efficiency, messages are written into Kafka in batches. A batch is just a collection of messages, all of which are being produced to the same topic and partition. An individual round trip across the network for each message would result in excessive over– head, and collecting messages together into a batch reduces this. This, of course, presents a tradeoff between latency and throughput: the larger the batches, the more messages that can be handled per unit of time, but the longer it takes an individual message to propagate. Batches are also typically compressed, which

provides for more efficient data transfer and storage at the cost of some processing power.

Schemas

While messages are opaque byte arrays to Kafka itself, it is recommended that additional structure be imposed on the message content so that it can be easily understood. There are many options available for message schema, depending on your application's individual needs. Simplistic systems, such as Javascript Object Notation (JSON) and Extensible Markup Language (XML), are easy to use and human readable. However they lack features such as robust type handling and compatibility between schema versions. Many Kafka developers favor the use of Apache Avro, which is a serialization framework originally developed for Hadoop. Avro provides a compact serialization format, schemas that are separate from the message payloads and that do not require generated code when they change, as well as strong data typing and schema evolution, with both backwards and forwards compatibility.

A consistent data format is important in Kafka, as it allows writing and reading messages to be decoupled. When these tasks are tightly coupled, applications which subscribe to messages must be updated to handle the new data format, in parallel with the old format. Only then can the applications that publish the messages be updated to utilize the new format. New applications that wish to use data must be coupled with the publishers, leading to a high-touch process for developers. By using well- defined schemas, and storing them in a common repository, the messages in Kafka can be understood without coordination.

Topics and Partitions

Messages in Kafka are categorized into topics. The closest analogy for a topic is a database table, or a folder in a filesystem. Topics are additionally broken down into a number of partitions. Going back to the "commit log" description, a partition is a single log. Messages are written to it in an append-only fashion, and are read in order from beginning to end. Note that as a topic generally has multiple partitions, there is no guarantee of time-ordering of messages across the entire topic, just within a single partitions are also the way that Kafka provides redundancy and scalability. Figure 16 shows a topic with 4 partitions, with writes being appended to the end of each one. . Each partition can be hosted on a different server, which means that a single topic can

be scaled horizontally across multiple servers to provide for performance far beyond the ability of a single server.

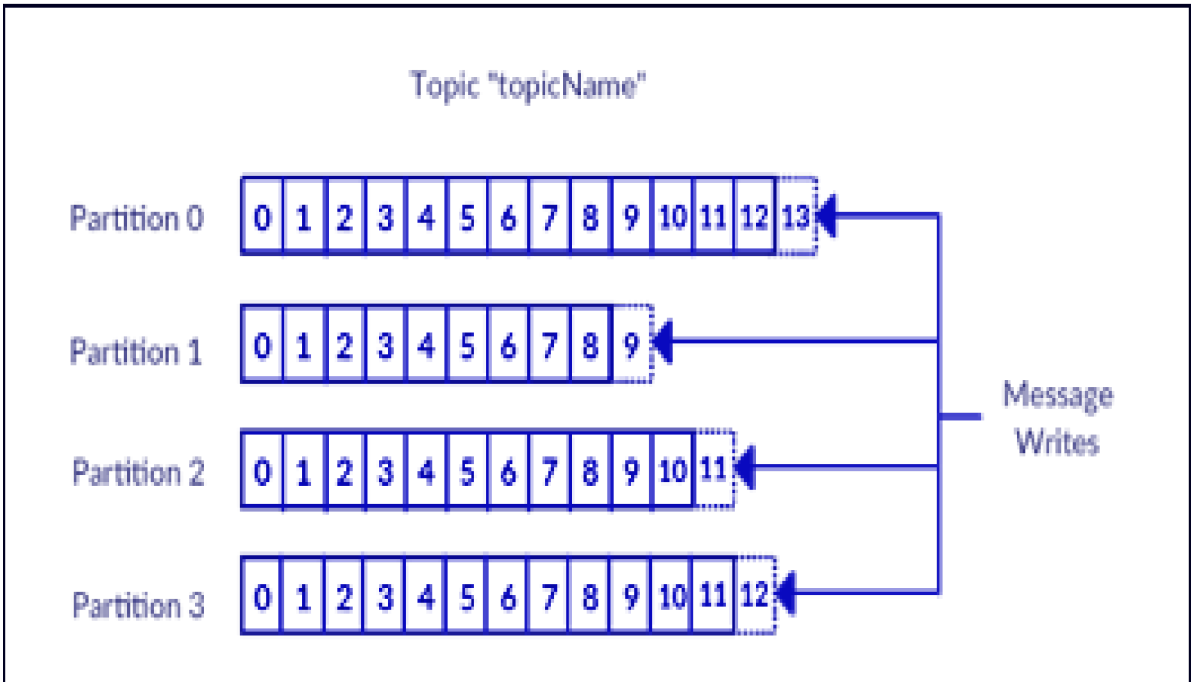

Fig. 16. Representation of a topic with multiple partitions

The term stream is often used when discussing data within systems like Kafka. Most often, a stream is considered to be a single topic of data, regardless of the number of partitions. This represents a single stream of data moving from the producers to the consumers. This way of referring to messages is most common when discussing stream processing, which is when frameworks, some of which are Kafka Streams, Apache Samza, and Storm, operate on the messages in real time. This method of operation can be compared to the way offline frameworks, namely Hadoop, are designed to work on bulk data at a later time.

Producers and Consumers

Kafka clients are users of the system, and there are two basic types: producers and consumers.

Producers create new messages. In other publish/subscribe systems, these may be called publishers or writers. In general, a message will be produced to a specific topic. By default, the producer does not care what partition a specific message is written to and will balance messages over all partitions of a topic evenly. In some cases, the

producer will direct messages to specific partitions. This is typically done using the message key and a partitioner that will generate a hash of the key and map it to a specific partition. This assures that all messages produced with a given key will get written to the same partition. The producer could also use a custom partitioner that follows other business rules for mapping messages to partitions.

Consumers read messages. In other publish/subscribe systems, these clients may be called subscribers or readers. The consumer subscribes to one or more topics and reads the messages in the order they were produced. The consumer keeps track of which messages it has already consumed by keeping track of the offset of messages. The offset is another bit of metadata, an integer value that continually increases, that Kafka adds to each message as it is produced. Each message within a given partition has a unique offset. By storing the offset of the last consumed message for each partition, either in Zookeeper or in Kafka itself, a consumer can stop and restart without losing its place.

Consumers work as part of a consumer group. This is one or more consumers that work together to consume a topic. The group assures that each partition is only consumed by one member. In Figure 17, there are three consumers in a single group consuming a topic. Two of the consumers are working from one partition each, while the third consumer is working from two partitions. The mapping of a consumer to a partition is often called ownership of the partition by the consumer.

In this way, consumers can horizontally scale to consume topics with a large number of messages. Additionally, if a single consumer fails, the remaining members of the group will rebalance the partitions being consumed to take over for the missing member.

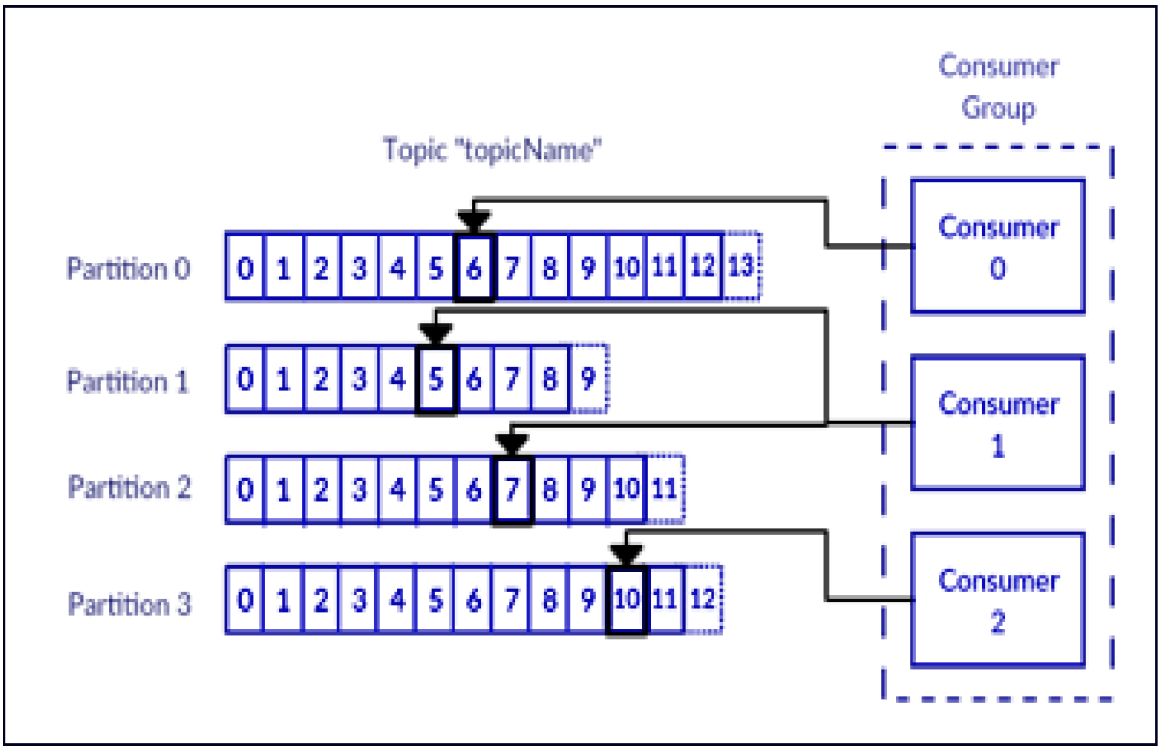

Fig. 17. A consumer group reading from a topic

Brokers and Clusters

A single Kafka server is called a broker. The broker receives messages from producers, assigns offsets to them, and commits the messages to storage on disk. It also services consumers, responding to fetch requests for partitions and responding with the messages that have been committed to disk. Depending on the specific hardware and its performance characteristics, a single broker can easily handle thousands of partitions and millions of messages per second.

Kafka brokers are designed to operate as part of a cluster. Within a cluster of brokers, one will also function as the cluster controller (elected automatically from the live members of the cluster). The controller is responsible for administrative operations, including assigning partitions to brokers and monitoring for broker failures. A partition is owned by a single broker in the cluster, and that broker is called the leader for the partition. A partition may be assigned to multiple brokers, which will result in the partition being replicated (as in Figure 18). This provides redundancy of messages in the partition, such that another broker can take over leadership if there is a broker

failure. However, all consumers and producers operating on that partition must connect to the leader.

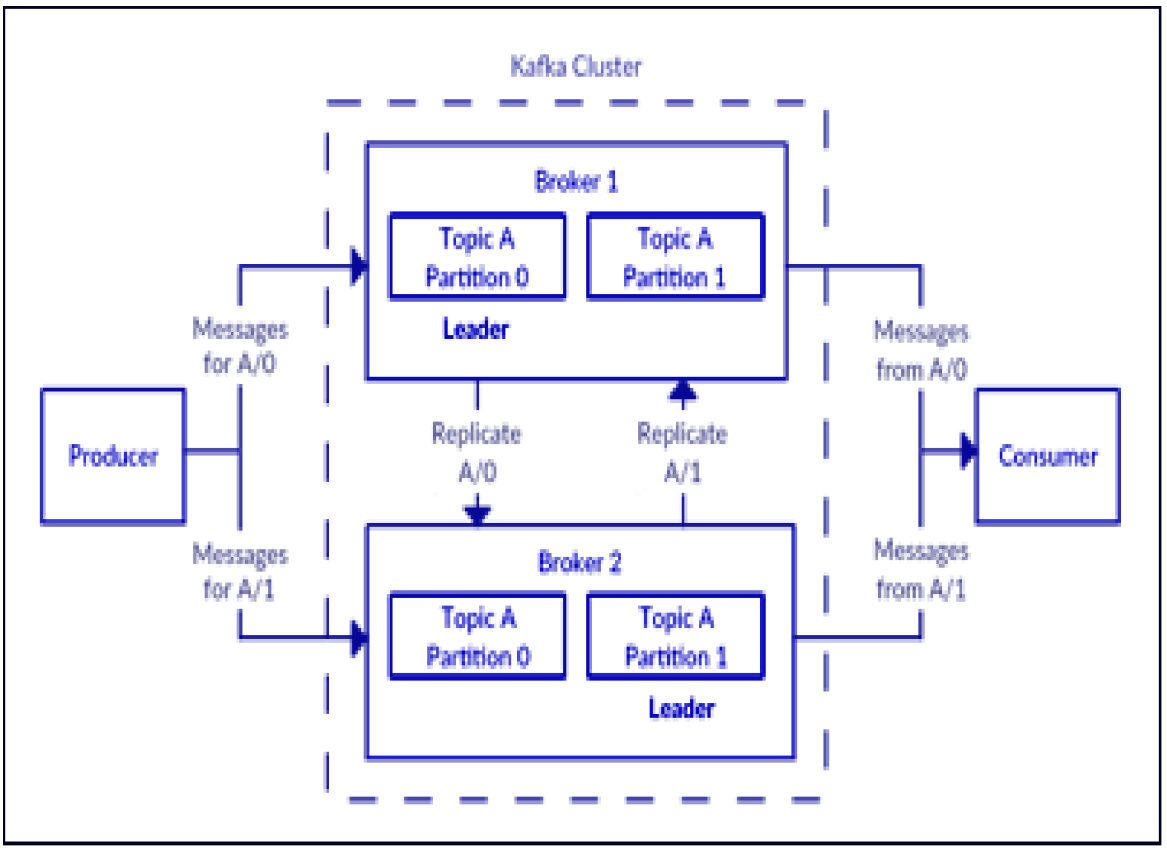

Fig. 18. A partition assigned to multiple brokers

A key feature of Apache Kafka is that of retention, or the durable storage of messages for some period of time. Kafka brokers are configured with a default retention setting for topics, either retaining messages for some period of time (e.g. 7 days) or until the topic reaches a certain size in bytes (e.g. 1 gigabyte). Once these limits are reached, messages are expired and deleted so that the retention configuration is a minimum amount of data available at any time. Individual topics can also be configured with their own retention settings, so messages can be stored for only as long as they are useful. For example, a tracking topic may be retained for several days, while application metrics may be retained for only a few hours. Topics may also be configured as log compacted, which means that Kafka will retain only the last message produced with a specific key. This can be useful for changelog-type data, where only the last update is interesting.

Multiple Clusters

As Kafka deployments grow, it is often advantageous to have multiple clusters. There are several reasons why this can be useful:

- Segregation of types of data
- Isolation for security requirements
- Multiple datacenters (disaster recovery)

When working with multiple datacenters, in particular, it is usually required that messages be copied between them. In this way, online applications can have access to user activity at both sites. Or monitoring data can be collected from many sites into a single central location where the analysis and alerting systems are hosted. The replication mechanisms within the Kafka clusters are designed only to work within a single cluster, not between multiple clusters.

The Kafka project includes a tool called Mirror Maker that is used for this purpose. At its core, Mirror Maker is simply a Kafka consumer and producer, linked together with a queue. Messages are consumed from one Kafka cluster and produced to another. Figure 19 shows an example of an architecture that uses Mirror Maker, aggregating messages from two “Local” clusters into an “Aggregate” cluster, and then copying that cluster to other data centers. The simple nature of the application belies its power in creating sophisticated data pipelines, however.

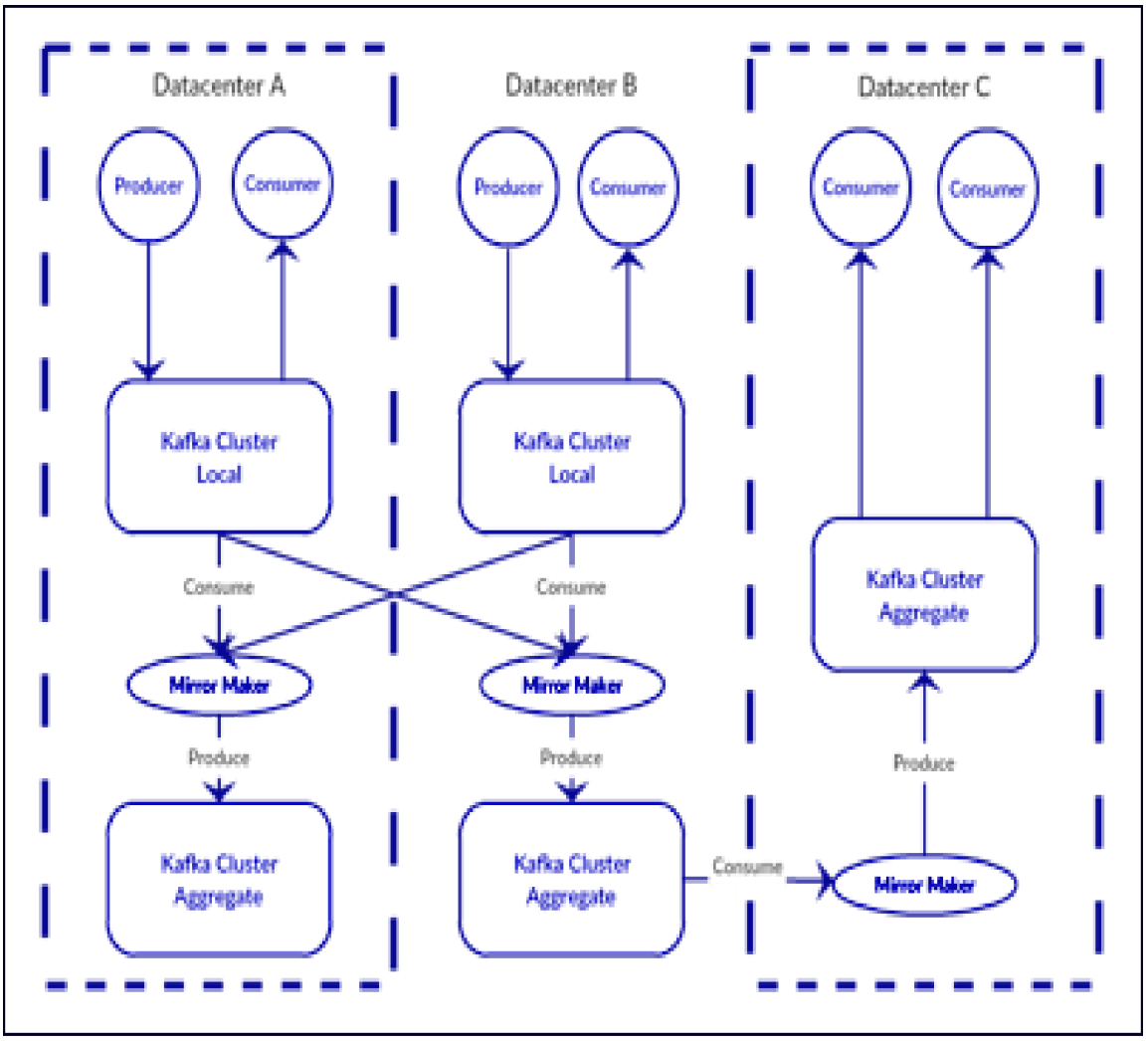

Fig. 19. Multiple datacenter architecture

5.2 Why Kafka?

There are many choices for publish/subscribe messaging systems, so what makes Apache Kafka a good choice?

Multiple Producers

Kafka is able to seamlessly handle multiple producers, whether those clients are using many topics or the same topic. This makes the system ideal for aggregating data from many front end systems and providing the data in a consistent format. For example, a site that serves content to users via a number of microservices can have a single topic for page views which all services can write to using a common format. Consumer

applications can then received one unified view of page views for the site without having to coordinate the multiple producer streams.

Multiple Consumers

In addition to multiple consumers, Kafka is designed for multiple consumers to read any single stream of messages without interfering with each other. This is in opposition to many queuing systems where once a message is consumed by one client, it is not available to any other client. At the same time, multiple Kafka consumers can choose to operate as part of a group and share a stream, assuring that the entire group processes a given message only once.

Disk-based Retention

Not only can Kafka handle multiple consumers, but durable message retention means that consumers do not always need to work in real time. Messages are committed to disk, and will be stored with configurable retention rules. These options can be selected on a per-topic basis, allowing for different streams of messages to have different amounts of retention depending on what the consumer needs are. Durable retention means that if a consumer falls behind, either due to slow processing or a burst in traffic, there is no danger of losing data. It also means that maintenance can be performed on consumers, taking applications offline for a short period of time, with no concern about messages backing up on the producer or getting lost. The consumers can just resume processing where they stopped.

Scalable

Flexible scalability has been designed into Kafka from the start, allowing for the ability to easily handle any amount of data. Users can start with a single broker as a proof of concept, expand to a small development cluster of 3 brokers, and move into production with a larger cluster of tens, or even hundreds, of brokers that grows over time as the data scales up. Expansions can be performed while the cluster is online, with no impact to the availability of the system as a whole. This also means that a cluster of multiple brokers can handle the failure of an individual broker and continue servicing clients. Clusters that need to tolerate more simultaneous failures can be configured with higher replication factors.

High Performance

All of these features come together to make Apache Kafka a publish/subscribe messaging system with excellent performance characteristics under high load. Producers, consumers, and brokers can all be scaled out to handle very large message streams with ease. This can be done while still providing sub-second message latency from producing a message to availability to consumers.

The Data Ecosystem

Many applications participate in the environments we build for data processing. We have defined inputs, applications that create data or otherwise introduce it to the system. We have defined outputs, whether that is metrics, reports, or other data products. We create loops, with some components reading data from the system, performing operations on it, and then introducing it back into the data infrastructure to be used elsewhere. This is done for numerous types of data, with each having unique qualities of content, size, and usage.

Apache Kafka provides the circulatory system for the data ecosystem. It carries messages between the various members of the infrastructure, providing a consistent interface for all clients. When coupled with a system to provide message schemas, producers and consumers no longer require a tight coupling, or direct connections of any sort. Components can be added and removed as business cases are created and dissolved, while producers do not need to be concerned about who is using the data, or how many consuming applications there are.

CHAPTER 6
OVERALL SYSTEM WORKFLOW

6.1 Cowrie Honeypot

The user interacts with the honeypots with the aim to explore the system. These actions are logged into the system with these information columns.


```
{
        "eventid": "cowrie.client.version",
        "macCS": ["hmac-sha1", "hmac-sha1-96", "hmac-md5", "hmac-md5-96",
                  "hmac-ripemd160", "hmac-ripemd160@openssh.com"],
        "timestamp": "2017-07-03T18:30:08.235671Z",
        "session": "577076dfb79e",
        "kexAlgs": ["diffie-hellman-group14-sha1",
                    "diffie-hellman-group-exchange-sha1",
              "diffie-hellman-group1-sha1"],
        "keyAlgs": ["ssh-rsa", "ssh-dss"],
        "message": "Remote SSH version: SSH-2.0-PUTTY",
        "system": "HoneyPotSSHTransport,523,116.31.116.16",
        "isError": 0,
        "src_ip":  "116.31.116.16",
        "version": "SSH-2.0-PUTTY",
        "dst_ip": "10.10.0.13",
         "compCS": ["none"],
        "sensor": "DC-NIC-Mumbai",
        "encCS": ["aes128-ctr", "aes192-ctr", "aes256-ctr", "aes256-cbc",
              "rijndael-cbc@lysator.liu.se", "aes192-cbc", "aes128-cbc",
              "blowfish-cbc", "arcfour128","arcfour",
              "cast128-cbc","3des-cbc"]
}
```


The data is stored in the MongoDB database at the end of the session.

6.2 Kafka

Producer

Each honeypot device has its own producer module. The producer is responsible for publishing the honeypot log data to the brokers. Each producer node subscribes to a cluster of Kafka brokers which automatically distributes the load among themselves. The producer constantly monitors the log produced by honeypot device, converts it to

JSON format acceptable to the streamer and publishes it to an input topic. The JSON file consists of attacker details as well as the input offset of the attacker. Topics are additionally broken down into a number of partitions. The partition in which a data is placed is determined by its key. The data with the same input key will be grouped together in the same partition at the broker end. To maintain the order of the input data, we assign the unique session id of each attacker as the key to the input topic.

Producer Code:

```
while (cursor.hasNext()) {
     Document temp = cursor.next();
     String session = temp.get("session").toString();
     String eventid = temp.get("eventid").toString();
     String timestamp = temp.get("timestamp").toString();
     System.out.println(temp);

     try{
          producer.send(produce(session,topic,eventid,times
                          tamp));
     }
     catch (Exception e)
     {    break;    }
     }
} finally {
     cursor.close();
}
```

Kafka Streamer

Kafka Streamer is responsible for real-time prediction of the future attack sequence. It performs a set of map and reduce operations to aggregate and compute the results. The streamer is subscribed to the input topic which is periodically updated by the kafka producer. Each kafka streamer has its own flask server which is responsible for computing the predicting via various machine learning algorithms.

The first map operation maps the input stream of data into a numeric value between 0-18 and stores the result in a custom data structure named as streamList. A groupByKey() operation, then aggregates the mapped values with respect to the session Id. This is followed by a reduce operation which combines the data of each group and produces a final output. This output is in the form of a streamList data

structure which consist of a list of sequence generated by each attacker sequence. This reduce operation also ensures the list does not exceed the maximum limit of aggregated data to be preserved. The streamList corresponding to each attacker sequence is stored in a KTable.

This is followed by a second map phase to produce a stream of predicted output. At this stage, the streamer sends a POST request to the flask server. The body of this request is in the form of an array of attacker sequence. The server computes the prediction and sends the predicted output back to the streamer. The prediction is stored in JSON format. This map phase produces data in the form of a KStream. This KStream data is flushed back to the kafka broker via an output topic.

Producer Streamer:

```
KTable<String,streamList> table = input.map((k, v) ->
     {
          streamList newClass = new streamList();
          String event_id = v.get("event_id").asText();
          int val = utils.map(event_id);
          int offset = v.get("offset").asInt();
          newClass.l1 = new LinkedList<Integer>();
          newClass.l1.add(val);
          newClass.offset = offset;
          return new KeyValue<String, streamList>(k,
                                   newClass);
     }
).groupByKey(Serialized.with(Serdes.String(),utils.getstreamL
istSerde())).reduce((x, y) ->
                {
                     x.l1.addAll(y.l1);
                     x.offset = y.offset;
                     int sizeCounter = x.l1.size();
                     return x;
                }
);
KStream<String,JsonNode> prediction = table.toStream().map(
(k,v) ->{
     ObjectNode node
               =JsonNodeFactory.instance.objectNode();
     node = utils.createArray(node);
     try {
          node = utils.addToArray(node,v.l1);
     }
```

```
      catch (IOException e) {
           e.printStackTrace();
      }

      ObjectNode predictionJson
                     =JsonNodeFactory.instance.objectNode();
      String pred = "12";
      int[] buffer = utils.toIntArray(v.l1);
      predictionJson = Predict.prediction(buffer,k);
      //pred = Predict.prediction();
      node.put("key",k);
      node.put("offset",v.offset);

      node.put("prediction",predictionJson.get("predictions"))
      ;
      return new KeyValue<String, JsonNode>(k, node);
});
```

Kafka Consumer

Each honeypot device has its own consumer module. The consumer module is responsible for publishing the honeypot log data to the MongoDB for the storage. Each consumer node subscribes to a cluster of Kafka brokers which automatically distributes the load among themselves. The consumer constantly monitors the predictions made by Machine learning models, converts it to JSON format acceptable to the MongoDB and publishes it to a corresponding topic. The JSON file consists of predicted states as well as the probability of the attacker sequence.

Consumer Code:

```
consumerRecords.forEach(record -> {
   ObjectMapper mapper = new ObjectMapper();
   JsonNode node = record.value();
   String predictions =
     record.value().get("prediction").toString().replaceAll
                                                  ("\\\\","");
   try {
       JsonNode array =
               mapper.readTree(predictions).get("");
       System.out.println(array);
   } catch (IOException e) {
       e.printStackTrace();
     }
```

```
        Document document = new Document("session",record.key())
                    .append("prediction",
        record.value().get("prediction").toString().replaceAll("
        \\\\","")).append("true", true);
        collection.insertOne(document);
});
```

6.3 Machine Learning Models

The Flask server which is responsible for the prediction and analysis of the sequence runs the machine learning models on input data from the streamer. The streamer sends a sequence of the states ranging from 0-18 each representing action taken by the user. The output of the Machine learning model is the prediction of the next state. The machine learning Model can be either Markov Chain, HMM or the LSTM Model.

6.4 MongoDB database

The MongoDB database is the storage component of the system. The log files which are created through interaction can be stored and utilised to improve the accuracy of the Machine learning model by retraining the model on generated data. Also the predicted states for each sequence can be used as measure of accuracy for the Models. The mongoDB interacts with Kafka subsystem through the producers and the consumers to send and receive the data in JSON format. A sample raw log record stored in MongoDB is shown below -


```
{
            "_id" : ObjectId("5ce39cb466e6b811404954ed"),
            "dst_ip" : null,
            "dst_port" : null,
            "duration" : null,
            "eventid" : "cowrie.command.success",
            "input" : "/bin/busybox 81c46036tftp",
            "keyAlgs" : null,
            "message" : "Command found: /bin/busybox 81c46036tftp",
            "password" : null,
            "sensor" : "DC-NIC-Mumbai",
            "session" : "274c501ed69f",
            "src_ip" : "190.236.71.103",
            "src_port" : null,
            "system" : "CowrieTelnetTransport,1253,190.236.71.103",
            "timestamp" : "2017-07-03 18:31:54.918724",
            "username" : null                                          }
```

6.5 Overall System Data-flow

The data-flow in the overall system is shown in figure 20.

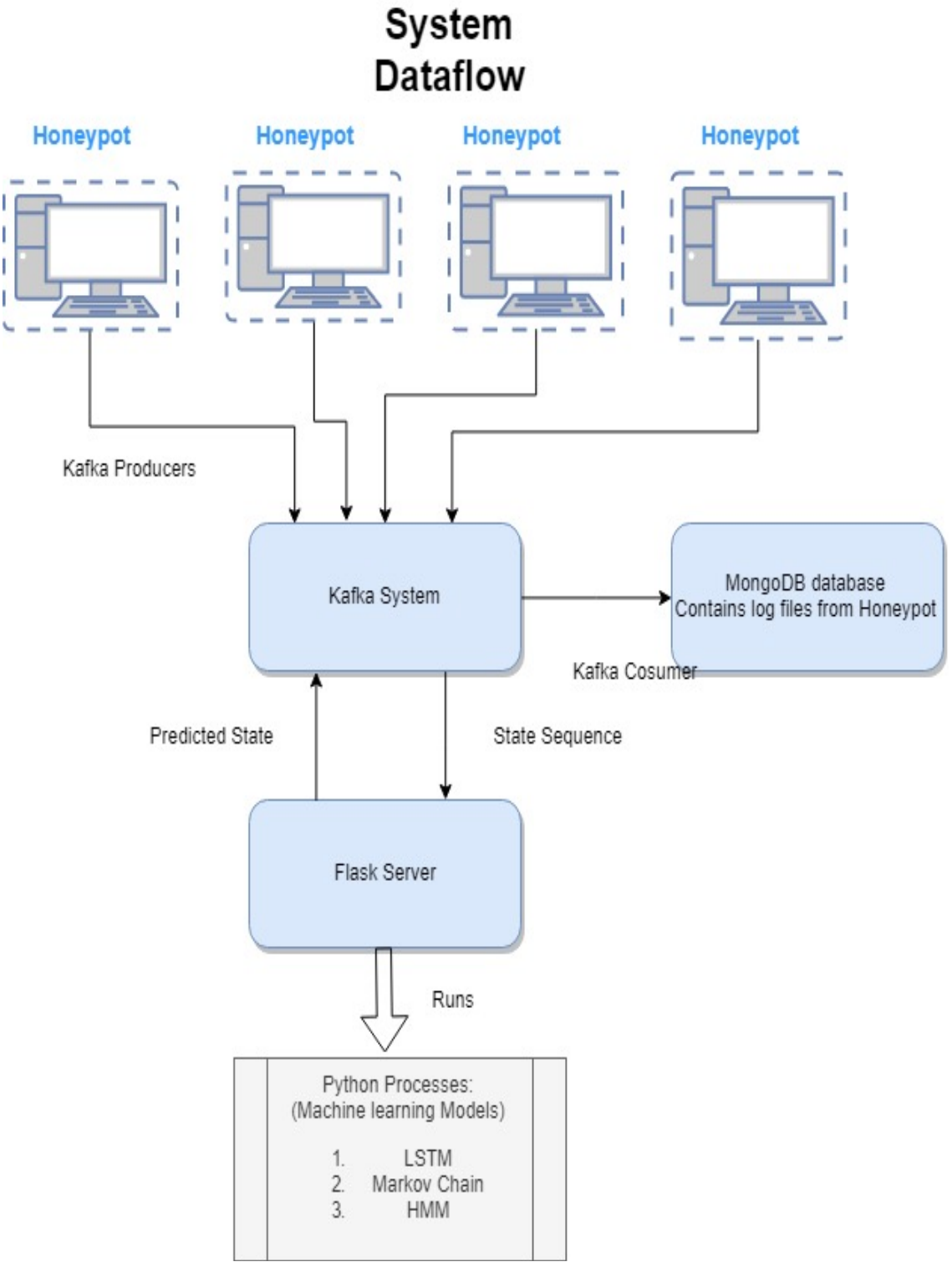

Fig. 20. Overall System Data-flow

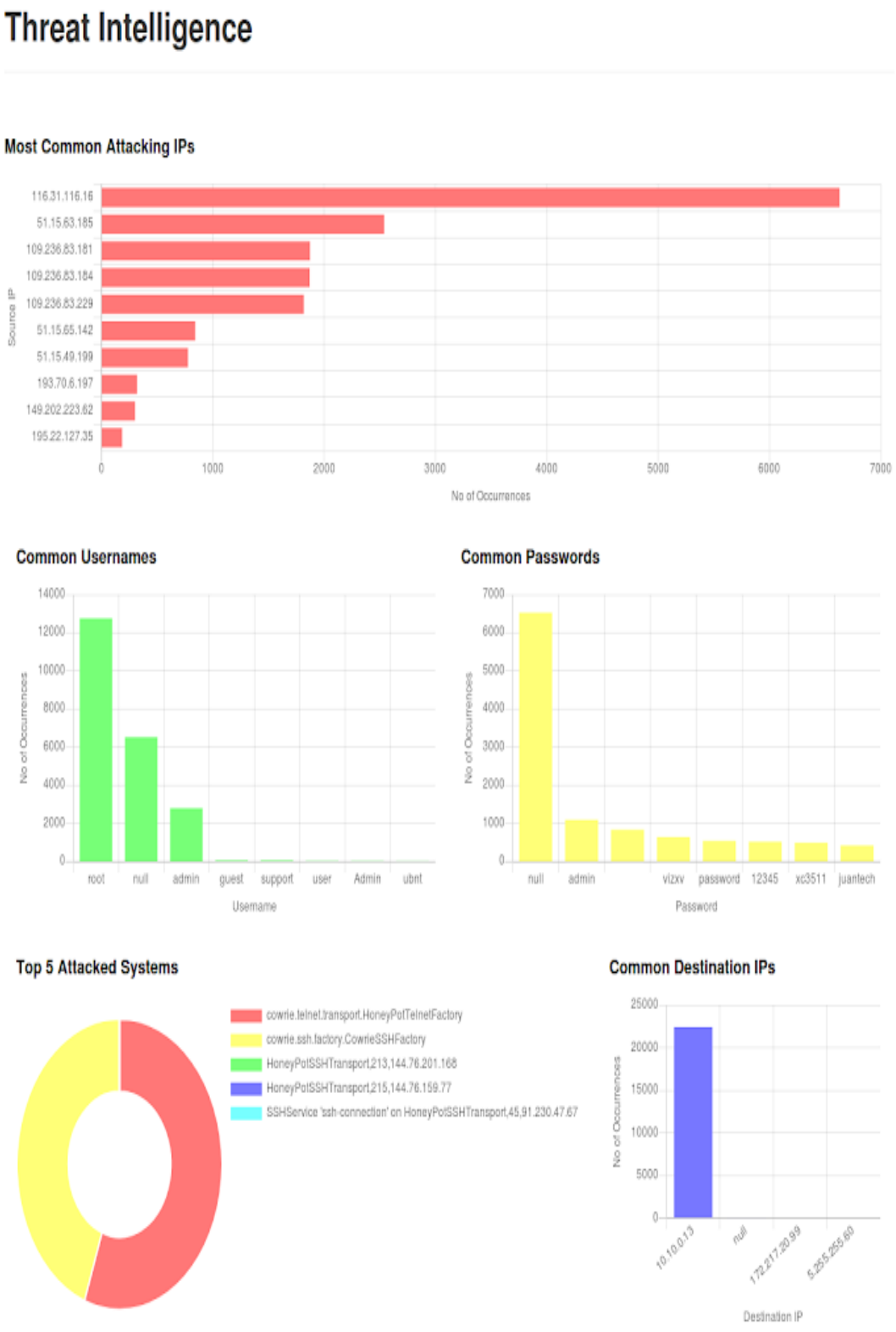

Fig. 21. Dashboard Screenshot - 1

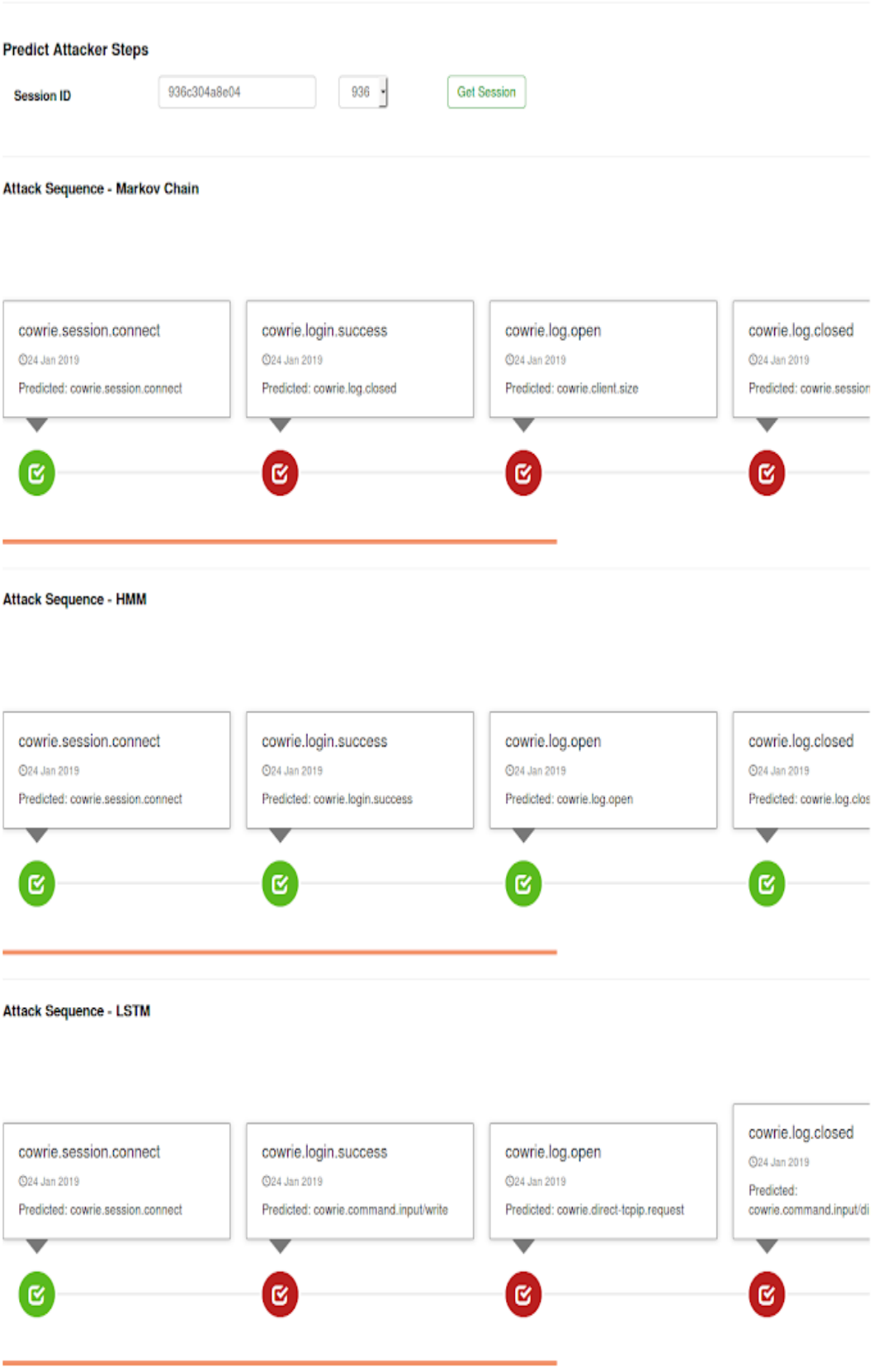

Fig. 22. Dashboard Screenshot - 2

CHAPTER 7
CONCLUSION AND FUTURE WORK

In this work, we proposed a threat intelligence platform powered by machine learning and big data to tackle the evolving threats posed by cyber attacks. The machine learning system is able to spot a wider variety of threats — even variants — and to decide how to best mitigate them before they infect endpoints and networks. As a part of machine learning system, we investigated the use of probabilistic models for modelling attack sequences occurring on Cowrie honeypots based on the logs generated by the honeypots. Further, we presented two novel approaches to effectively model the behaviour of an intruder, thereby providing an estimate of emerging threats posed by the professional attackers. Using first order Markov chain along with variable length Hidden Markov Model, we were successfully able to model short to medium attack sequences. These trained models are capable of predicting next highly probable steps which the attackers might take along with an insight of the most common attacking trends and new and evolving techniques which attackers use. On the other hand, an LSTM model played a critical role in modelling long term dependencies present in sequences consisting of more than hundred actions and with a commendable accuracy can predict the future actions of attacker. Analysing the sequences generated by the proposed models may shed light on new and evolving techniques which attackers take which intruding into a system. Thus, the system administrators may take more stringent steps to tackle these emerging threats. In addition, the developed machine learning algorithms can be extended further from file system logs and used to develop a retaliatory sequence of action with two end benefits. First to ban the attacker if he possess an immediate threat to the system or any way could compromise the system. Second to stall the attacker in the honeypot long enough to generate his accurate logs. Along with the intelligence provided by these algorithms, we employ a Kafka-based architecture which is capable of real time

data processing using its streaming module. This simply improves the speed and scalability of the system.

Furthermore, this works truly highlights the potential of development of a reinforcement-based algorithm wherein the agent is a system which can take preventive or retaliatory actions and environment consists of attackers and his action. Massive number of interactive logs will play a pivotal role in learning an appropriate policy based on traditional bellman equation

REFERENCES

[1] Schneier, B., “Honeypots and the Honeynet Project”, 2001, [Online]. Available via: [http://www.cs.rochester.edu/~brown/Crypto/news/3.txt], Accessed 26 July. 2018

[2] B. C. Cheng, G. T. Liao, C. C. Huang, and M. T. Yu, “A novel probabilistic matching algorithm for multi-stage attack forecasts,” IEEE Journal on Selected Areas in Communications, vol. 29, no. 7, pp. 1438–1448, August 2011.

[3] Diwakar Shukla, Rahul Singhai (2010). Analysis of Users Web Browsing Behavior Using Markov chain Model, Int. J. 2. 824-830.

[4] Mohammad Reza Norouzian,Sobhan Merati. (2011). Classifying attacks in a network intrusion detection system based on artificial neural networks - IEEE Conference Publication.Paper presented at the 13th International Conference on Advanced Communication Technology (ICACT 2011), Seoul, South Korea, 13-16 Feb. 2011

[5] Bisyron Wahyudi Masduki,Kalamullah Ramli,Ferry Astika Saputra,Dedy Sugiarto. (2016). Study on implementation of machine learning methods combination for improving attacks detection accuracy on Intrusion Detection System (IDS).Paper presented at the 2015 International Conference on Quality in Research (QiR), Lombok, Indonesia, 10-13 Aug. 2015

[6] Kwangjo Kim,Muhamad Erza Aminanto (2018). Deep learning in intrusion detection perspective: Overview and further challenges.Paper presented at the 2017 International Workshop on Big Data and Information Security (IWBIS), Jakarta, Indonesia, 23-24 Sept. 2017

[7] Kolesnikov, Oleg, Lee, Wenke Advanced Polymorphic Worms: Evading IDS by Blending in with Normal Traffic(2005) : CC Technical Report; GIT-CC-05-09 ,Georgia Institute of Technology, Available via: [http://hdl.handle.net/1853/6485], accessed 26 July. 2018

[8] Venkata SreeKrishna Koganti,Lavanya K. Galla,Nagarjuna Nuthalapati (2018). Internet worms and its detection. Paper presented at the 2016 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT), Kumaracoil, India, 16-17 Dec. 2016

[9] Jianchao Hong and Ying Hua 2018 IOP Conf. Ser.: Mater. Sci. Eng. 322 052033. Avaliable via http://iopscience.iop.org/article/10.1088/1757-899X/322/5/052033/pdf] Accessed 26 July 2018

[10] Lawrence R. Rebiner A tutorial on hidden Markov models and selected applications in speech recognition - Proceedings of the IEEE , 1989

[11] Rose Hoberman,Dannie Durand (2006) HMM Lecture Notes [http://www.cs.cmu.edu/~durand/03-711/2006/Lectures/hmm-bw.pdf] Accessed 26 July. 2018

[12] Grinstead, C. M., & Snell, J. L. (2012). Introduction to probability. American Mathematical Soc

[13] Chan, Ka & T. Lenard, C & Mills, Terence. (2012). An Introduction to Markov Chains. 10.13140/2.1.1833.8248.

[14] Rabiner, Lawrence R., and Biing-Hwang Juang. "An introduction to hidden Markov models." ASSP Magazine, IEEE 3.1 (1986): 4-16.

[15] Cho, K., van Merrienboer, B., Gulcehre, C., Bougares, F., Schwenk, H., and Bengio, Y. (2014a). Learning phrase representations using RNN encoder-decoder for statistical machine translation. In Proceedings of the Empirical Methods in Natural Language Processing (EMNLP 2014). to appear.

[16] Graves, A. (2013). Generating sequences with recurrent neural networks. arXiv: 1308.0850 [cs.NE]

[17] Bengio, Y., Frasconi, P., and Simard, P. (1993). The problem of learning long-term dependencies in recurrent networks. pages 1183–1195, San Francisco. IEEE Press.

[18] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9(8), 1735–1780.

[19] Official repository for the Cowrie SSH and Telnet Honeypot effort. Available via [https://github.com/micheloosterhof/cowrie], Accessed 26 July 2018.

[20] Razvan Pascanu, Tomas Mikolov, Yoshua Bengio(2013) . On the difficulty of training Recurrent Neural Networks. arXiv:1709.03082v7 [cs.NE] 10 Mar 2018.

[21] Alvaro A Cardenas, Pratyusa K Manadhata, and Sreeranga P Rajan. Big data analytics for security. IEEE Security & Privacy, (6):74–76, 2013.

[22] Kanubhai K Patel and Bharat V Buddhadev. Machine learning based research for network intrusion detection: A state-of-the-art. International Journal of Information and Network Security (IJINS), 3(3), 2014.
[23] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In Proc. of the 2010 IEEE Symposium on Security and Privacy, pages 305–316. IEEE, 2010.
[24] Colin Tankard. Big data security. Network security, 2012(7):5–8, 2012.
[25] M. Bhuyan et al. Network anomaly detection: Methods, systems and tools. IEEE Communications Surveys & Tutorials, 16(1):303–336, 2014.
[26] Shan Suthaharan. Big data classification: problems and challenges in network intrusion prediction with machine learning. ACM SIGMETRICS Performance Evaluation Review, 41(4):70–73, 2014.
[27] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In Proc. 17th USENIX Security Symposium, 2008.
[28] Daniela Brauckhoff et al. Anomaly extraction in backbone networks using association rules. IEEE/ACM Transactions on Networking (TON), 20(6):1788–1799, 2012.
[29] Apache Kafka Project Home [http://kafka.apache.org/].
[30] “Kafka: The Definitive Guide: Real-Time Data and Stream Processing at Scale”, ASIN: B0758ZYVVN, a book by Gwen Shapira, Neha Narkhede, and Todd Palino