Online Security Analytics
Hélder Figueiredo Reia, 86214, MEEC
Instituto Superior Técnico, Academia Militar
e-mail: helder.reia@gmail.com

Abstract—In this work, the difficulties presented in data analytics are analyzed for a cybersecurity system, as well as the flaws and limitations presented in the current detection systems and how to create a more reliable system that can tackle these limitations.

The main objective of this document is to create security monitoring system with means for automation, with near real-time threat detection and awareness.

To do this, different machine learning techniques, both supervised and unsupervised, were combined to make the detection process of network anomalies as automatic as possible. The used methods were features extraction from the input data, clustering using an unsupervised machine learning method to aggregate the entities based on the previous feature extraction, extracting features from the obtained clusters with a genetic algorithm, classification of the obtained clusters, based on feature values and output the misbehaving groups as well as the most important features that characterizes this data.

Two different datasets were used corresponding to flows of two real networks, where from each one a set of features were extracted. The main goal of the developed work is to extract useful information to detect cyberattacks from streaming data, without information of previous patterns. After this, a genetic algorithm was applied to extract the best features for the given dataset, in order to characterize the groups with suspicious behavior better.

This approach was implemented by leveraging two open source processing and storage frameworks, in order to obtain a system with more redundancy and strength. Two datasets were used to evaluate experimentally the approach and its implementation.

This system was presented to some people inside the Portuguese Army, having been approved as an important project with applicability in developing in the present and future with the goal to complement the intrusion detection systems.

Index Terms—data analytics; cybersecurity; machine learning; processing frameworks; intrusion detection systems.

I. INTRODUCTION

Since the end of last century, there were rapid advancements in telecommunications, computing hardware and software, and cryptography. The rapid growth and widespread use of electronic data processing and electronic business conducted through the Internet, along with numerous occurrences of international cybercrime and increased cyberattacks, fuelled the need for better methods of protecting the computers and the information they store, process and transmit. Furthermore, according to several security reports, the average time a company takes to detect a successful attack is about 100 days [1].

To try to detect and prevent this increasing number of attacks, intrusion detection systems (IDS) have been proposed [2]. IDSs are systems that can detect intrusions from events and can be classified by analysed activity or by detection method employed. By analysed activity there are: network intrusion detection systems (NIDS) which are placed at strategic points within the network to monitor traffic to and from devices, the main source being TCP/IP traffic; host intrusion detection systems (HIDS) on the other hand, run on individual hosts or devices on the network and the main sources are file integrity analysis, security events from the operating system or even file signatures. By detection method employed, IDSs can be divided in: signature-based, referring to the detection of attacks by looking for specific patterns; anomaly-based, to detect unknown previous attacks by analysing a network and classify the activity as normal or anomalous, using heuristics instead of patterns. These systems have deserved a lot of attention in the computer security area and are under continuous development by industry because of the growing need to stop cyberattacks. Thus, the Cloud Security Alliance (CSA) proposed that IDSs have gone through three stages of evolution [3]:

• IDS – The first detection systems that were used to detect known attacks using signatures and abnormal behaviour based on statistics derived from normal patterns;
• Security Information and event management (SIEM) systems – They can collect and manage relevant data from devices in the network, providing an increased network security by aggregating and filtering alarms.
• Second generation SIEM – They can handle and collect the largest proportion of the big data with reduced time and contextualizing more diverse and unstructured security data.

The data that is most commonly used in cybersecurity for intrusion detection are the logs generated by almost every network device. These logs have information of the behaviour of all devices and the users that use these devices. This leads to a big data problem, where there are a lot of network events that we need to filter so we can use them in cybersecurity.

Nowadays, the common form is to look for signatures or (patterns) [4] of the known attacks in the data logs, trying to compare what is supposed to be a normal activity with a malicious one. The signature-based detection is thoroughly used by companies to detect known attacks, but it still requires a lot of manual analysis [5] to improve the detecting system. Furthermore, it can only be used to detect attacks that are already known, meaning that if there are new attacks, with a different pattern from the ones that are known, they will be hardly detected.

Currently, SIEM systems are commonly used but they cannot handle the big data problems nor extract useful information contained in big data. SIEMs collect and manage
security relevant data from different devices in the network (e.g., IDSs, firewalls, authentication servers). These systems provide increased network visibility by aggregating and filtering alarms while providing actionable information to security analysts [3]. In the near future, the SIEM systems should be able to handle the big data problem, providing a log-term storage for correlating data [6].

II. THE STATE OF ART

A. Big Data Challenges

Due to an increasing number of people connected to the internet, the ability to communicate, share, and access data is getting more and more important. Thus, the logs generated by the devices are increasing each day, creating a big data problem for the security systems and analysts.

In the past, storing this volume of data would be very difficult but nowadays, with the decreasing of storage costs, there are also algorithms and better solutions to store all the significant data [6]. Having so much data stored creates a great problem for analysts since they must search through a lot more quantity of information. Big data can be defined with 3V’s [7]: Volume, which refers to large amounts of data that traditional computing techniques cannot handle; Velocity, which is the speed at which data is processed, and there can be a big data challenge when the rate of data is moving too quickly to process and Variety refers to the complexity of the data, and there can be a big data challenge when the data includes complex problems such as high dimensionality, data from many sources, or data having many different data structures: all of these problems can cause difficulty in processing with traditional computing techniques. Nowadays all information faces a new dimension, the Value of the information, which is the veracity of the information that one has plus the quality of this information.

The usage of big data is widely used, such that many services and enterprises are currently trying to get advantages from it. According to EMC Digital Universe with Research & Analysis by International Data Corporation (IDC), data is growing at an annual rate of 40% into the next decade [8]. What is significant is that the smart devices that are connected to the Internet – Internet of Things (IoT) – will contribute significantly to the data volumes, velocity, and heterogeneity.

The main big data challenges are data quality, data streams, dynamically evolving data, data heterogeneity and data modelling, multi-model databases, client and query interfaces, data compression, data encryption, access control and authorization, and deployment on cloud-hosted cluster computers. One task that crosscuts all the above challenges is identifying a subset of big data that has high value. This requires separating the data that is contaminated by spam, noise and bias from that which is uncontaminated [8].

B. Machine Learning Methods

Managing such amount of data is beyond human capabilities and the usage of machine learning methods is becoming more useful to extract information from large and multi-dimensional data.

The machine learning techniques are commonly divided in two categories: supervised and unsupervised learning. Next it will be introduced these categories and further ahead it will be shown some examples that are already use machine learning methods.

In supervised learning, the training data must be labelled by humans for every data point. A supervised learning algorithm analyses the training data and produces an inferred function, which can be used for mapping new examples.

One of the most important use cases of supervised learning in the cybersecurity domain are anti-spam systems. This system receives as input a set of discriminating features of the e-mail, which can help in predicting if it is spam or not. The system outputs whether an e-mail is likely to be spam or ham (not spam). Another very important use case is the pattern recognition, that is normally used in Network Security, in signature-based IDS, where it checks every request for access to the network against a set of existing attack signatures in order to detect possible attacks [9].

Unlike the previous learning category, unsupervised algorithms do not require previous labelled data. Unsupervised learning is useful when one wants to explore data but do not yet have a specific goal or if one is not sure what information the data contains. It is also a good way to reduce the dimensions of data.

Unsupervised learning methods can be applied in cybersecurity when we are trying to create a system to detect attacks that we do not know because they can compute the outliers that can correspond to attackers. In supervised learning this is not possible because supervised learning methods need to have desired outputs – known attacks.

The main tools used for unsupervised learning are K-means clustering, gaussian mixture modelling, hierarchical clustering, principal component analysis (PCA) and spectral clustering.

C. Frameworks

To tackle the big data problem generated by huge volume of security events, a suitable framework must be adopted. Nowadays these frameworks are still under development but there is some research being done in this area.

Currently there are three major frameworks used in cybersecurity applications: Apache Spot, Elastic Stack (previously ELK Stack) and Apache Metron. All these frameworks are based on Hadoop platform and use the MapReduce programming model.

Apache Hadoop is an open source Apache project used for distributed storage and processing of dataset of big data using the MapReduce programming model. MapReduce is a software framework for easily writing applications which process vast amounts of data in-parallel on large clusters of commodity hardware in a reliable, fault-tolerant manner.

Even though Hadoop is good for batch processing, it is considered unsuitable for data streams because a Hadoop job assumes that all data exists in files on various nodes, and will start its Map and Reduce phases on a fixed amount of input to generate a fixed amount of output, and this is a problem because, for streaming applications, there is a steady stream of data that does not stop. Hadoop is also not suitable for algorithms that are iterative and depend on previously computed values. This class of algorithms includes many types of machine learning algorithms that are critical for sophisticated data analytics, such as online learning algorithms.

Apache Spark is an open source cluster computing system that has as primary objective to speed up data analytics in runtime and in development. To do this, Spark provides primitives for in-memory cluster computing: a job can load data into memory and query it repeatedly much more quickly.
than with disk-based systems such as Hadoop MapReduce. One of the biggest difficulties was to perform interactive queries on streaming data but the unified Spark stack is designed to handle these requirements efficiently and scalably.

Spark fits with Hadoop ecosystem as an alternative to MapReduce, while using the same underlying infrastructure such as YARN and the HDFS.

Apache Spot is an open source software for cybersecurity to give security insights from packet analysis. Spot expedites threat detection, investigation, and remediation via machine learning and consolidates all enterprise security data into a comprehensive information technology hub based on open data models. Spot’s scalability and machine learning capabilities support an ecosystem of machine learning based applications that can run simultaneously on a single, shared, enriched data set to provide organizations with maximum analytic flexibility.

Apache Spot uses tools to provide functionalities in preventing attacks, identifying unknown threats, and providing cybersecurity. Spot has four layers, being the first layer the perimeter flows, where can be caught the intrusion scanning of our network, unusual data flows or beaconing. The second layer is the internal layer of DNS where Spot can detect DNS Tunneling, DNS directed attacks. The third layer if the proxy layer where the command and control is made. The fourth and last layer is the internal flows, that provides the analyst a complete threat visibility.

Apache Metron [10] is a cyber security application framework released in 2016 that provides organizations the ability to ingest, process and store diverse security data feeds at scale in order to detect cyber anomalies.

Metron integrates a variety of open source technologies in order to offer a centralized tool for security monitoring and analysis. Metron provides capabilities for log aggregation, full packet capture indexing, storage, advanced behavioural analytics and data enrichment while applying threat-intelligence information to security telemetry. Metron can be divided into 4 areas: A mechanism to capture, store and normalize any type of security telemetry at extremely high rates (due to the constant generation of security telemetry data, it is required to have a method that ingests data at high speeds, redirecting it to various processing units); Real time processing and application of enrichments (such as threat intelligence, geolocation, and DNS information to telemetry being collected); Efficient information storage (Logs and telemetry are stored so that they can be mined and analysed); An interface that gives a security investigator a centralized view of data and alerts passed through the system (interface that presents alert summaries with threat intelligence and enrichment data specific to that alert on one single page).

Apache Metron has been thoroughly used because it integrates capabilities, using other platforms such as Storm for Data Processing, and it has a portal, dashboard and user interface for each different user.

## III. System Description

The main goal of the developed work is to extract useful information to detect cyberattacks from streaming data, without information of previous patterns, as used in anomaly-based IDSs. Also, anomaly-based IDSs are not what we want because they need to be trained with a lot of non-malicious data. Thus, it was chosen to develop a detection system with the goal to find attacks without previous information about them, by grouping and understanding entities with similar behaviours. It is intended to generate classification models dynamically using: clustering to extract information from logs using generic features; application of a genetic algorithm to search for the best specific features to understand better which features are more important to our set of data. By doing this, it will be possible to get more knowledge about the data logs and apply this knowledge on developing better classification models that can be focused on detecting specific behaviours.

### A. Pre-runtime phases

The first two phases are executed before runtime to train the system and provide classifiers for the runtime phase.

The first phase, represented in Figure 1, is the feature definition and classification. This phase is defined in three steps: the definition of the normalization of the data, to transform the data so that we know what parameters to normalize to make all data consistent; the feature selection, to know which features are going to be extracted to characterize the available data, defining the time periods (to aggregate time windows) to extract the features; the definition of how the features are extracted from the data logs, given our set of data.

![Figure 1 - Flowchart of the first phase: feature and normalization definition](image)

The second phase is the training of the classifier used in runtime. In this phase, the input data will be a training set of data logs. After the normalization, the clustering is performed to group entities with the same behaviour. As said previously, this grouping is only based on the given data, without having data from each entity beforehand. After this process, a feature extraction of these clusters is done. These features taken from the clusters will be an input to a genetic algorithm. The genetic algorithm is used to understand which features are important to identify an entity as belonging to a suspicious cluster. Also, this genetic algorithm was used to understand which features characterize better the data. From this, the analyst will confirm and identify the root cause of the outlier. If the content is inside a group considered malicious, it will be given to the analyst with the set of features that were considered more important from the genetic algorithm so that he can have more information about the misbehaviour. The analysed clusters are only the outliers, that are more likely to be the entities with misbehaviour.

### B. Runtime phase

The last and most important phase is the execution of the detection system in runtime. In the first part, the classifiers trained from the second phase will classify the input data. If the input data is malicious, the classification method will output the label given by the classifier, so that the analyst can know what the malicious content is. If this data is not malicious, it will be forwarded to a system similar to the second phase of pre-runtime, where the data will be clustered and inputted to the genetic algorithm, giving more information to the analyst so that he can update the classifier. This is a method to rapidly analyse the content that is already known as malicious since the input data only needs to go through the first classification. It also provides the detection
of new or unknown attacks, by giving to the analyst every new suspicious content so that the classification can be updated.

The runtime period can be different if we want to detect different suspicious behaviours. The normal runtime can be configured (e.g. to be in 1 hour, the classification of the input data is normalized for 1 hour) but, if we want to analyse the data in a period of a day, it is possible by changing the clustering to aggregate the logs in a different time window (e.g. 24 hours).

C. Framework Description

Given that Apache Spot and Apache Metron are still under development, a new framework was proposed, based on a processing module and a distributed storage module. The outcome would have to cover the tools that we wanted to use as the previous 2 frameworks would. The storage module has the objective of store the data that arrives from other sources and the processing module has the objective of processing this large quantity of data in a distributed way. To do so, the architecture of the Figure 2 was proposed. In order to explain this framework, it was divided in 2 sections: storage and processing modules, shown in Figure 2 on the left and right respectively.

![Figure 2 - System Architecture](image)

D. Storage Module – Apache Kafka

The first part of this framework is the storage unit, where the data that we want to input on our system will be stored. To do so, the Apache Kafka framework was used.

Apache Kafka is a distributed streaming platform, with capabilities of publish and subscribe streams of records, similar to a message queue or enterprise messaging system; store streams of records in a fault-tolerant and durable way and process streams of records as they occur.

Apache Kafka has four core APIs: the Producer API, the Consumer API, the Streams API and the Connector API.

To accomplish this, Kafka stores streams of records in categories called topics. A topic is a category or feed name to which records are published. Topics in Kafka are always multi-subscriber, that is, a topic can have zero, one, or many consumers that subscribe to the data written to it. Kafka topics are divided in partitions. Partitions allow one to parallelize a topic by splitting the data in a particular topic across the multiple servers.

Each partition of the log is an ordered, immutable sequence of records that is continually appended to—a structured commit log. The records in the partitions are each assigned a sequential id number called the offset that uniquely identifies each record within the partition. This offset gives an order inside a partition, but this order is only guaranteed inside its partition. The data within a partition is immutable, which means that after the data is written, it cannot be changed.

When a consumer is assigned to a topic, meaning that it will input some data in the topic, the data is randomly assigned to a partition within this topic. This can be changed if we want to have the order within each partition, by adding a key that guarantee that all messages for that key will always go to the same partition, but in our case, it was not necessary because the data was being read as it was being received to the topic.

In order to guarantee the correct operation of the brokers (servers), a broker manager is needed. To do that, Apache Zookeeper is used. Zookeeper is a framework that provides multiple features for distributed applications like distributed configuration management, consensus building through coordination and locks. Zookeeper manages the brokers, keeping a list of those brokers and helps in performing leader election for partitions. Apart from this, Zookeeper also sends notification to Kafka in case of changes (e.g. new topic creation, broker dies, delete topics). Zookeeper usually operates in a quorum — in our case there is a Zookeeper quorum of 3 servers, with one leader and the other two as followers, guaranteeing that there can be one server down and yet the Zookeeper quorum still works.

The producer API consists on how the producer writes data to topics. The producer only has to specify the topic name that he wants to write data and a broker to connect to, and Kafka will automatically take care of routing the data to the right brokers. Also, a producer can choose to receive acknowledgement of data writes, being it “0”, where the producer will not wait for acknowledgement of the topic, “1”, where the producer will wait for leader acknowledgement and “all”, where the producer will wait for both leader and replica to acknowledge.

The consumer API reads data from a topic. The consumer only has to specify the topic name from where he wants to read the data and one broker server to connect to, and Kafka will again take care of pulling the data from the right brokers. The data in a topic is read in order for each partition, which means that the first messages received in a partition, will be the first ones to be read by the consumer. As it was presented previously, Kafka stores the offsets of the data inside a partition. Similarly, Kafka also stores the offsets at which a consumer group has been reading, so that if a consumer process dies, it will be able to read back from where it left, by getting the latest consumer offset.

The last two APIs, the connectors API and the streams API, are very similar being the streams API a client library for building applications and microservices, where the input and output data are stored in Kafka clusters and the connectors API is a tool for scalably and reliably streaming data between Kafka and other systems. The major differences between the two is that the streams API is used when we want to connect to other applications while the connector API is used when we want to access a database or another framework. In this work the streams API was not used but the connectors API was used to connect our data in Kafka to Apache Spark, as we will see ahead.

In this work it was created a Kafka topic with replication factor 3, where the data was inputted. Figure 3 shows the described topic.

![Figure 3 - Kafka Topic](image)
As the previous image shows us, the topic “topic1” was created with 3 partitions with a replication factor of 3. Here we can see all the partitions, the leaders for each partition, how the replicas are distributed and the in-sync replicas. Due to our configuration, the replicas are all synchronized, that is why there is the same number of replicas and ISR.

E. Processing Module – Apache Spark

The second unit of the framework is the processing unit, where all the computation to achieve the results from the proposed solution are made. To implement the processing section, the Apache Spark framework was used. Apache Spark is a distributed and highly scalable in-memory data analytics system with four main submodules: Spark SQL, Spark Streaming, MLlib and GraphX.

In this work, only the Spark Streaming and MLlib modules were used. The Spark MLlib module was thoroughly used, being able to run advanced data analysis algorithms scalably. Apache Spark, in its core, provides the runtime for massive parallel data processing, and different parallel machine learning libraries are running on top of it. As soon as more data is loaded to the available main memory of the system, it crashes. This was mainly the reason why Apache Spark was used, because it allowed these kinds of operations keeping the viability, due to Apache Spark make use of multiple computer nodes to form a cluster and even on a single node can spill data transparently to disk therefore avoiding the main memory bottleneck.

The last module, Spark Streaming, involves the processing of continual streams of data in real time, with the functionality of checkpointing and windowing, being explained latter ahead in this project.

Similarly to Apache Kafka, Spark also has to have a cluster manager, which then allocates resources across the worker nodes for the application. The cluster manager allocates executors across the cluster worker nodes. The available cluster managers are: Local, by specifying a Spark configuration local URL, it is possible to have the application run locally, being this a useful development and test option because one can also test some sort of parallelization scenarios but keep all log files on a single machine; Standalone, a mode that uses a basic cluster manager that is supplied with Apache Spark, supporting only first-in-first-out (FIFO) scheduling; Apache Yet Another Resource Negotiator (YARN), a mode that is better in larger scale and runs in two modes: if the Spark master value is set as yarn-cluster, then the application can be submitted to the cluster and then terminated, being the cluster the one that takes care of allocating resources and running tasks. If the application master is submitted as yarn-client, then the application stays alive during the life cycle of processing, and requests resources from YARN. Lastly, the last cluster manager available for Spark is Apache Mesos, an open source system for resource sharing across a cluster. It allows multiple frameworks to share a cluster by managing and scheduling resources. This last cluster manager is very used when there are thousands of nodes with various frameworks and applications running.

In this work, the cluster manager used was the standalone mode due to its simplicity and accessibility.

Every Spark application consists on a driver program that runs a user main function and executes various parallel operations in a cluster. The main abstraction Spark provides is the resilient distributed dataset (RDD), which is a collection of elements – similar to an array – partitioned across the nodes of the cluster that can be operated on in parallel. It is also possible to ask Spark to persist an RDD in memory, allowing it to be reused efficiently across parallel operations.

At this point is important to see how the frameworks are implemented in the servers. The 3 available servers are named Machine1, Machine2 and Machine3, being Machine1 the master server and the other two the slaves. For Zookeeper, Machine1 is the leader server and the other two are the followers. For Spark, it works the same way, Machine 1 is the master and Machine 2 and 3 are the worker nodes. Table 1 shows all of the configurations made in the servers:

<table>
<thead>
<tr>
<th>Name</th>
<th>IP</th>
<th>Zookeeper</th>
<th>Spark</th>
<th>Memory (Spark)</th>
<th>Cores (Spark)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine 1</td>
<td>192.168.0.144</td>
<td>Leader</td>
<td>Master</td>
<td>Dynamically</td>
<td>Dynamically</td>
</tr>
<tr>
<td>Machine 2</td>
<td>192.168.0.145</td>
<td>Follower</td>
<td>Worker</td>
<td>8 Gb</td>
<td>4</td>
</tr>
<tr>
<td>Machine 3</td>
<td>192.168.0.146</td>
<td>Follower</td>
<td>Worker</td>
<td>8 Gb</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 1 - Server Configuration

This cluster was created with servers of rack 1U type provided by the Communications and Information Systems Department of the Portuguese Army (DCSI), in order to validate the framework with commodity servers, since there was a restriction of processing data only in these servers inside DCSI infrastructure.

IV. ALGORITHMS AND CODE SPECIFICATIONS

After the designed framework was implemented, the intended code was developed to perform the analysis of the input data and to try to discover the anomalous traffic. Thus, the developed code was divided in: feature extraction, clustering and online mode with window analysis.

Before performing the feature extraction, we need to parse the data. The data given as input to Spark is in Unicode, so it is first decoded to a string and then it is splitted to get each parameter of the logs, so that afterwards we can take each parameter to do the feature extraction. This splitting has to be done because when data given as input in Spark, it reads as a string, so a separation of values had to be done.

A. Feature Extraction

The features to select are one of the most important steps when analysing a set of data because it is based on these features that we will be able to detect some attacks. The selected features for this work were based on the ones studied in the related work and that were able to implement using our datasets, given that these datasets used were netflow logs. With this, a set of 26 features was used, which are divided in two distinct groups: the first half are features extracted from the source computer and the other half are extracted from the destination computer. This was made to analyse both potential attackers and entities that are being attacked. The required transformations and operations needed to perform the feature extraction were the Map transformation, to group pairs of values, the CountByValue operation, because there were some features that counted the frequency of an occurrence (e.g. number of connections made) and the reduceByKey operation, to remove the repeated values by summing the values for each entity (e.g. a Computer).
The 13 other features are: number of connections received, number of ports used by a source, analysing a receiver, number of ports contacted in a receiver, sum of packets received, sum of packets received to port 80, sum of packets received from port 80, sum of packets received to port 194, sum of packets received from port 194, sum of packets received to port 25, sum of packets received from port 25, sum of packets received to port 22, sum of packets received from port 22 and the sum of bytes received.

There are some behaviours that are simple to observe through these features. For example, the features related to number of connections made, ports used and contacted are easily related because there can be some entities that contact with a lot of other entities (e.g. a server that is being contacted by a large group of computers replies to these computers, creating a large number of connections and can have also a large number of ports used/contacted). The major number of features are based on specific ports. These ports were selected based on their relevance and if they were present in the used datasets. In the first dataset used, the addresses and ports were modified, with the exception of the ones obtained in this feature extraction: port 80, the standard port for an Hypertext Transfer Protocol (HTTP) client to establish a Transmission Control Protocol (TCP) connection; port 194, the standard port for Internet Relay Chat (IRC) Protocol; port 25, the standard port for Simple Mail Transfer Protocol (SMTP) and port 22, the standard port for the Secure Shell (SSH) Protocol. After these features are extracted, the output consists in an entity (IP) followed by every value for each feature. This output was also saved in a file so that the processing of the feature extraction would not be repeated.

B. Clustering

After the data is normalized, it is given as input to the clustering algorithm. Our approach aims to separate well-behaved from misbehaving entities, so the clustering algorithm has to separate entities with different behaviour, being different behaviour expressed by different values of features. The available algorithms in Apache Spark MLlib are: K-means, Gaussian mixture, power iteration clustering (PIC), Latent Dirichlet allocation (LDA), bisecting K-means and Streaming K-Means. LDA and Gaussian mixture were not chosen because they are still experimental features under development for Apache Spark MLlib. PIC was also not selected because this is an algorithm that must have as input tuples of entities-similarity. The chosen algorithm was then K-Means because it is one of the most commonly used clustering algorithms that clusters the data points into a predefined number of clusters. Its implementation has the following parameters:

- \( k \) is the number of desired clusters
- \( \text{maxIterations} \) is the maximum number of iterations to run;
- \( \text{initializationMode} \) specifies either random initialization or initialization via k-means++ (a variant of the standard k-means algorithm);
- \( \text{epsilon} \), parameter that determines the distance threshold within which we consider k-means to have converged;
- \( \text{initialModel} \) is an optional set of clusters centres used for initialization. If this parameter is supplied, only one run is performed.

The output of this algorithm is the cluster number of which each entity belongs to and the central points of each cluster.

C. Genetic Algorithm

After the input data is clustered, it is important to know which features are important to our set of data, specially to identify which features better compose a malicious cluster. As presented previously in the system architecture, to extract the most important features to build the clusters for each dataset, a genetic algorithm is used. To do so, an evolutionary algorithm framework was used: Distributed Evolutionary Algorithms in Python (DEAP). This was the chosen framework because it is oriented to Python and our system was developed in that language. DEAP is a novel evolutionary computation framework for rapid prototyping and testing of ideas, seeking to make algorithms explicit and data structures transparent [11].

There are four important concepts to be aware of: a gene, an individual, a population and a generation. A gene is nothing more than a set of properties that an individual has. The individual is a group of genes, being each individual a solution to the problem that one wants to solve. A population is a set of individuals, which can be modified from generation to generation, being a generation a new population, normally inheriting individuals from previous populations or modifying them (e.g. with operators such as crossover and mutation).

The first type that has to be created is the fitness class, which is an abstract class that needs weights in order to be functional. A fitness can be either a minimizing fitness, a maximizing fitness or a multiple weight fitness, with both negative and positive. In this work, a maximizing fitness was used because the fitness function used, explained ahead, had the objective of getting the higher value possible.

After creating the fitness class, the individual class is created to specify how the individuals are created. When an individual is created there are only three parameters given: the name of the class, the class which it will inherit and the fitness used. In this work, the individual class was created inheriting from the list class with the fitness being a maximizing fitness, defined earlier. After creating the individual, we must register it, taking two steps, firstly by specifying what will the individual be composed of, i.e. the genes that are in this individual, and secondly how a new individual is created. In this work, we applied a genetic algorithm to obtain the features that best describe the set of data. To do so, we will maintain only some features, and compare the results obtain with a smaller set of features compared to all the features used. The number of features must be defined earlier because if this is not set, the genetic algorithm will try to get the best solution, meaning that it will try to have as many features available from the original as possible.

The first operator registered is the evaluation operator. The evaluation is the most personal part of an evolutionary algorithm because it’s the part of this genetic algorithm that correlates the data from outside the genetic algorithm. In this work, the evaluation function that was made was a function that, receiving as input the individual to evaluate, selects from our data only the features corresponding to ones in our individual and applies the clustering algorithm. After this, the algorithm will compare the clusters obtained with the smaller number of features versus the clusters obtained with every feature used. This evaluation returns a score for this evaluation based on the proximity of the number of entities inside a cluster in both cases.
After getting this evaluation, the best individuals (the ones that obtained a best score) were selected for the next generation. To create a new population, the crossover and mutation operators were used. The crossover operation requires two individuals that came from the previous generation, called the parents, and the offspring are created by exchanging genes of parents among themselves until the crossover point is reached.

After the crossover operation is made, the mutation operator was applied. Mutation is applied to a new offspring formed to change their genes given a low random probability. It occurs to maintain diversity within the population and prevent premature convergence.

Both crossover and mutation operations used functions that shuffled the attributes of the input individual and return the mutant.

In this work the individual has a set of 26 genes. A population is made with 32 individuals, being the first generation generated randomly. There is a total of 40 generations and each generation is created with half of the best individuals from the previous generation plus a set of individuals generated with crossover and mutation operations. This way, we keep our best individuals and mutate from them to try to obtain better individuals.

D. Streaming Processing

The streaming processing was the next step to develop. To do so, a streaming of data over time was needed. To do that, a Python script was developed to input data (from the used dataset) into Apache Kafka over time. The configurations done in batch processing was also done in streaming processing. Also, the extracted features and the clustering algorithm were the same. Internally, Spark Streaming works as shown in Figure 4, receiving live input data streams and divides the data into batches.

The used operations used in Spark Streaming are similar to the ones used in Apache Spark, but with a time parameter that has to be inputted. For the feature extraction, the main operation used was reduceByKey andWindow, that is similar to reduceByKey (used in batch mode) but with two more parameters: window length, being the total duration of the window, and sliding interval, being the interval at which the window operation is performed. As shown in Figure 15, every time the window slides over a source DStream, the source RDDs that fall within the window are combined and operated upon to produce the RDDs of the windowed DStream. The Stream processing is started with the stream context start method, and the awaitTermination method indicated that it should process until stopped. The processing stops either by the user terminating the session or if Spark Streaming does not receive new data for 10 consecutive time windows.

V. System Evaluation

After the framework was settled, it was necessary to do a study, working in offline mode, to understand how to parse and normalize the data and which features were important to extract. To conduct the experimental investigation, two kinds of data were used: a dataset from Los Alamos National Laboratory's corporate internal computer network and a dataset with real data provided by DCSI corresponding to the administrative computer network.

A. Data normalization

After the feature extraction was done, it was important to know how the values for each feature were distributed. To do so, the frequency for each feature was plotted to see its distribution. As it can be analyzed in Figures 5, the distribution values within each feature is closer to the left side of the plot, which means that there are a lot of small values and some big values.

Thus, this data needs to be normalized to make it more evenly distributed. The best normalization to this set of data is the logarithmic normalization, due to its asymmetry, skewed with most of its mass to the left. After this normalization is done, the data became more evenly distributed, as shown in Figure 6:

Apart from the logarithmic normalization, there was also needed to perform a normalization of the values of the extracted features between -1 and 1 because afterwards, this data will be inputted in a clustering algorithm that has as requirement to have the data distributed between -1 and 1.

B. Clustering Results

After the previous normalization, the extracted features were ready to be grouped and inputted in a clustering algorithm with K-Means algorithm and the input to this algorithm was the number of clusters we wanted plus our set of features per entity and the output is the number of the cluster that each entity belongs to, plus the centroids of these
clusters. From this, it was possible to sketch a plot to see the distribution of points inside the clusters. Since there are 26 dimensions (26 features), they could not be all sketched in one plot. Thus, the arranged solution to do this was creating a two-dimension plot with colours, where the axis are two features (e.g. connections made and ports contacted) and the colours are the distinct clusters, so that we can differentiate which points are from each cluster. Each point represents an entity (computer). Also, the central points of the clusters were printed. The number of centroids that was chosen was 15, chosen after performing some experiments where the number of centroids varied. In Figures 7 we can analyse the output clusters with $k=15$, for the pair of features connections made – ports contacted.

Apart from these plots, it was also sketched a heatmap to understand better what characterizes each cluster of outliers. Thus, it was analysed if the entities presented in the readteam file were inside the outlier clusters / clusters that had a suspicious behaviour.

In Figure 8 we can see an heatmap of the previous clusters, where in the vertical axis is the number of the cluster, the horizontal axis is the features and the colours correspond to the value of that feature for a given cluster.

**C. Streaming Output for Los Alamos Dataset**

The previous analysis was for a full month of data, which can lead to some behaviours be more hidden with the rest of the data. Thus, a streaming processing was made and outputted in smaller windows in order to try to get these suspicious behaviours. From this, the datasets were inputted in Spark Streaming and analysed over some windows of time. During this day only C17693 and C19932 had logs where they were source computers, but this was the chosen day because it had a lot of logs specially from C17693. This day was called ‘attday’ for further reference. From this, an heatmap was also plotted, as shown in Figure 9 and the clusters and the entities per cluster were again printed, to be again compared, shown in Table 2:

**Table 2 - Number of entities per cluster for “attday”**

<table>
<thead>
<tr>
<th>Cluster Number</th>
<th>Number of Entities</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>63</td>
</tr>
<tr>
<td>5</td>
<td>264</td>
</tr>
<tr>
<td>9</td>
<td>298</td>
</tr>
<tr>
<td>10</td>
<td>314</td>
</tr>
<tr>
<td>6</td>
<td>315</td>
</tr>
<tr>
<td>14</td>
<td>353</td>
</tr>
<tr>
<td>8</td>
<td>355</td>
</tr>
<tr>
<td>12</td>
<td>373</td>
</tr>
<tr>
<td>3</td>
<td>462</td>
</tr>
<tr>
<td>13</td>
<td>485</td>
</tr>
<tr>
<td>4</td>
<td>558</td>
</tr>
<tr>
<td>7</td>
<td>565</td>
</tr>
<tr>
<td>11</td>
<td>724</td>
</tr>
<tr>
<td>2</td>
<td>967</td>
</tr>
<tr>
<td>1</td>
<td>1695</td>
</tr>
</tbody>
</table>

**Figure 7 - Clustering output for 15 clusters for Los Alamos dataset**

**Figure 8 - Heatmap for the previous clusters**

**Figure 9 - Heatmap for the clusters of “attday”**

Analysing both the heatmap and the table, we can see that the outliers, cluster 0 and 5 are the ones with a more suspicious behaviour – cluster 0 because had a lot of ports contacted with a low number of ports used, a high number of connections made and also a high number of packets and bytes sent while cluster 5 had a low number of connections made but with a large number of packets and bytes sent.

With this smaller window, it was possible to reduce the number of entities that are considered suspicious, having now 327 out of 7791 (entities in cluster 0 and 5, considered suspicious), which makes 4.2% of the population and a reduction in more than 10% of the population when compared to the analysing a full month of data. Also, if we had more contextual information about each entity we could remove any server or entity that normally had a behaviour that is more specific.

After this, a genetic algorithm, explained earlier, was used, to try to understand which features were more important to characterize the dataset and, more specifically, the suspicious clusters. The output of the genetic algorithm, shown in Figure 10, shows us that for this dataset, our set of 6 more important features are: Number of different ports used, Number of different ports contacted, Sum of packets sent from port 80, Number of different ports contacted when analysing a destination, Sum of packets received to port 22 and Sum of bytes received, and the output clusters are 81.2% equal to the previous ones.

**Figure 10 - Output of genetic algorithm for Los Alamos dataset**

**D. Clustering Output for DCSI Dataset**

After the analysis of the dataset from Los Alamos was done, we used our real dataset from DCSI. This dataset
corresponded also to a full month of logs extracted from a SIEM, with a network of around 5500 computers. As said earlier in this work, these logs cannot be disclosed. The data extracted from SIEM was in raw, to simulate a real case where data is arriving directly from computer sources. To evaluate the system, some attacks were injected in order to try to catch them with this system. In Table 3, it is possible to see which attacks were injected:

<table>
<thead>
<tr>
<th>Source IP</th>
<th>Dest IP</th>
<th>Attack</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1 D1</td>
<td></td>
<td>Stealth Port Scan</td>
<td>Performed in 1s and 5s intervals</td>
</tr>
<tr>
<td>S2 D2</td>
<td></td>
<td>Stealth Dictionary Attack</td>
<td>SSH and RDP auth requests performed in 30s and 2 min intervals</td>
</tr>
<tr>
<td>S3 D3</td>
<td></td>
<td>Data exfiltration</td>
<td>Big volume of data sent to one entity</td>
</tr>
<tr>
<td>S4 D4</td>
<td></td>
<td>Beacon traffic</td>
<td>HTTP packets sent with low volume in 15min interval</td>
</tr>
</tbody>
</table>

Table 3 - Injected attacks for DCSI dataset

The sketches of the plots of both cluster output and heatmap of this dataset could not be provided since this data is classified, but both plots are similar to the ones analysed from the Los Alamos dataset.

The first analysis of this dataset was made in a big window (one month), just to characterize the data. In the first sketched plot, the output of the different entities grouped in 15 clusters, similarly to what was presented previously with the Los Alamos dataset. In DCSI dataset, there are 5461 different entities.

In the next plot it was sketched an heatmap of the corresponding 15 clusters with each feature. For this dataset, only 22 features were considered since the following features were not verified: Sum of packets sent from port 25, Sum of packets sent from port 22, Sum of packets received from port 25 and Sum of packets received from port 22. Also, it is possible to analyse that the number of clusters had to be adjusted since there are some clusters that are very similar.

To reduce groups with similar behaviours, new plots were sketched with an heatmap with different numbers of clusters, that grouped better our data. The best result obtained, where there were no more similar groups was with 8 clusters. From the plot with 8 clusters, clusters 0 and 5 are clusters more suspicious to have misbehaviour, since both have the highest number of connections made, ports used, and sum of bytes sent. Also, cluster 1 and 2 were considered suspicious because, the first one, had a high ratio between sum of packets sent per number of ports contacted, and cluster 2 had high values of packets sent to port 80.

Comparing these results with the suspicious clusters obtained and with what attack each entity did, it is easy to analyse that, for entity S1 it is in a cluster that corresponds with it behaviour, since cluster 0 is a cluster of entities with the highest values of connections made and ports contacted, for entity 3 it is also in a cluster that corresponds to its behaviour, since cluster 1 has the highest ratio of number of packets / number of bytes per connections and for entity 4 it is in a cluster that corresponds to its behaviour, because cluster 2 is a cluster with a high value of packets sent to port 88 (being the second highest value, being cluster 5 the one with the highest value in this feature). Entity 4 was the only feature that was not in a group considered suspicious and this happened because, analysing the heatmap, it is possible to see that there is no cluster with a high value for the feature packets sent to port 22 (SSH). This behaviour happens because this feature is not relevant when looking to the rest of the entities, meaning that even though this entity (S2) had this behaviour, it is grouped with other entities that do not have the same behaviour in this particular feature.

Since these values do not have any prominence, given that the used time window was too big to detect something, two new time windows were tested: one for the whole day where was more misbehaving entities and after this a window of 10 minutes, the smallest time window used in our streaming process, chosen after analysis on which hour had more misbehaving entities, in which entities S3 and S4 were present.

In the time window regarding ten minutes, when analysing the heatmap and the number of entities for each cluster, presented in Table 4, it is much easier to identify suspicious clusters, being them clusters 2 and 5. Cluster 2 has entities that performed a high number of connections with a high number of packets sent, specifically to port 80. Cluster 5, on the other hand, has entities that performed few connections but sent a lot of packets and bytes.

<table>
<thead>
<tr>
<th>Cluster Number</th>
<th>Number of Entities</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>76</td>
</tr>
<tr>
<td>5</td>
<td>79</td>
</tr>
<tr>
<td>3</td>
<td>125</td>
</tr>
<tr>
<td>6</td>
<td>214</td>
</tr>
<tr>
<td>7</td>
<td>228</td>
</tr>
<tr>
<td>4</td>
<td>231</td>
</tr>
<tr>
<td>0</td>
<td>324</td>
</tr>
<tr>
<td>1</td>
<td>450</td>
</tr>
</tbody>
</table>

Table 4 - Number of entities per cluster for 10 minutes attack period

Comparing this result with our misbehaving entities, we see that entity S3 was grouped in cluster 5 and entity S4 in cluster 2, which makes sense since these two entities fit in the description of the corresponding clusters, presented earlier. This time window had fewer entities present, only 1727 entities communicated in this ten minute window and, given our suspicious clusters, 155 entities were considered suspicious, which gives us 9% of suspicious entities out of the total entities for this time window.

After this analysis was done, the application of the genetic algorithm was proceeded. The parameters used were, again, 32 individuals per population, with 40 Generations and a number of output features equal to 6. The output of the genetic algorithm, shown in Figure 11, shows us that for this time window the 6 more important features are: number of connections made, number of ports used, sum of packets sent, sum of packets sent to port 80, sum of packets sent from port 80 and sum of packets received. The best individual presented 78% of accuracy, meaning that the obtained clusters were 78% equal to the previous ones.

Figure 11 - Output of genetic algorithm for Los Alamos dataset

After this analysis made, the genetic algorithm was applied again, but this time with 8 features instead of 6 to see which new features were now presented. The parameters used were the same, 32 individuals per population and 40
generations. Apart from the previous features, the new two features obtained were: sum of bytes sent and sum of packets sent received from port 80.

E. Performance Analysis

With all of the processing done and results obtained, a comparison of performance was done using 2 different approaches: performing the data analysis in batch mode and in streaming mode. These approaches were made to compare the times between them, to understand which one was more efficient. To do so, a part of the dataset from LosAlamos containing 3 days was processed – made the feature extraction, normalization and clustering. For the batch mode, this data was all loaded in memory and then processed while in streaming processing the data was being processed in a 10 minutes window, then the data was aggregated in a 1 hour window, 8 hours window, 1 day window and finally in the 3 days window, corresponding to the whole part of the dataset to test the performance. These 3 days correspond to 1.27 Gigabytes of data. For streaming processing, the time that was marked was the time of processing the last 10 minutes window plus the aggregation in the other windows.

From the data obtained from Spark UI it is possible to analyse that the obtained time for 3 days using batch processing was 47 minutes while performing a streaming analysis only took 13 minutes. This was expected since the batch processing has to load all the data into memory, which leaves low resources to do processing while streaming processing saves metadata with checkpointing, regarding the time windows that are still aggregating, only using memory to store the smaller window (10 minutes), leaving the rest to process the data.

Apart from the previous performance analysis, it was also tested if Spark Streaming could process data quicker than the smaller time window. Regarding our system, the processing of each 10 minutes window was 4 to 5 minutes, which means that our time windows are well defined and do not have more data than it is supposed to.

VI. CONCLUSION

All of the investigation and developed work in this thesis cover an important area not only for the Portuguese Army as well as any company that wants to have a solid cybersecurity framework.

Firstly, the system developed in this work aims to tackle the big data problems presented in previous sections. To do this and to tackle big data problems and to produce results quicker and more efficiently, a framework of combining data storage and data processing was used. This framework received netflow logs from different sources (Los Alamos and DCSI datasets) and returned a set of entities considered to have suspicious behaviour plus which features were considered more important for that set of data.

To validate this system two datasets were used: firstly, a dataset from Los Alamos was used and then a dataset containing information from DCSI network with some simulated attacks. The obtained results in both datasets reveal that there is a significant reducing in the number of entities to analyse, by discovering anomalous behaviours inside the set of data for different time windows. The obtained results could be even better if one has access to contextual information about the networks (e.g. which ones were servers, that had suspicious behaviour).

For future work, one of the improvements would be correlate different time windows not only vertically, comparing different smaller and bigger time windows and the behaviour of the entities inside these windows, but also horizontally, between the time windows with the same size (e.g. if an entity has the same behaviour in most 10 minutes windows). With this, it is possible to filter even more entities that have suspicious behaviour but not misbehaviour.

Another improvement would be analysing other sources rather than just netflow logs, such as firewall and host-based events. Correlating netflow logs with these events should provide more information about every entity, increasing the complexity on how data is processed. Also, there should be analysed different features to try to detect more attacks and best characterize the data.

Lastly, another improvement would pass to use different algorithms for both clustering and features selection, to understand which ones obtained better performance results. This could be obtained by implementing the new frameworks that could not be implemented (e.g. Apache Spot) that have a bigger amount of machine learning algorithms to process data and a visualization interface to present data in a more perceptible way.

REFERENCES