information

Article

Apache Spark and MLlib-Based Intrusion Detection System or How the Big Data Technologies Can Secure the Data

Otmane Azeroual [1,*] and Anastasija Nikiforova [2,3]

[1] German Centre for Higher Education Research and Science Studies (DZHW), 10117 Berlin, Germany
[2] "Innovative Information Technologies" Laboratory, Programming Department, Faculty of Computing, University of Latvia, Raina Boulevard 19, LV-1050 Riga, Latvia; Nikiforova.Anastasija@gmail.com
[3] European Open Science Cloud (EOSC) Task Force "FAIR Metrics and Data Quality", 1050 Brussels, Belgium
* Correspondence: azeroual@dzhw.eu; Tel.: +49-30-206417738

Abstract: Since the turn of the millennium, the volume of data has increased significantly in both industries and scientific institutions. The processing of these volumes and variety of data we are dealing with are unlikely to be accomplished with conventional software solutions. Thus, new technologies belonging to the big data processing area, able to distribute and process data in a scalable way, are integrated into classical Business Intelligence (BI) systems or replace them. Furthermore, we can benefit from big data technologies to gain knowledge about security, which can be obtained from massive databases. The paper presents a security-relevant data analysis based on the big data analytics engine Apache Spark. A prototype intrusion detection system is developed aimed at detecting data anomalies through machine learning by using the k-means algorithm for clustering analysis implemented in Sparks MLlib. The extraction of features to detect anomalies is currently challenging because the problem of detecting anomalies is not actively and exhaustively monitored. The detection of abnormal data can be effectuated by using relevant data that are already in companies' and scientific organizations' possession. Their interpretation and further processing in a continuous manner can sufficiently contribute to anomaly and intrusion detection.

Keywords: big data; machine learning; k-means; clustering; data anomaly; security; intrusion; intrusion detection; IDS

Citation: Azeroual, O.; Nikiforova, A. Apache Spark and MLlib-Based Intrusion Detection System or How the Big Data Technologies Can Secure the Data. *Information* **2022**, *13*, 58. https://doi.org/10.3390/info13020058

Academic Editor: Giovanni Russello

Received: 22 December 2021
Accepted: 21 January 2022
Published: 24 January 2022

1. Introduction

Today, during the fourth industrial revolution, also known as Industry 4.0, closely linked with the Internet of Things (IoT) forming its core, more and more data are being produced, collected, processed and analyzed [1]. These data, which may be critical for decision-making, forecasting, marketing-related competition etc., are exposed to constant dangers. The data are not only a lucrative target for criminals, but they are also of high desirability in industrial espionage specialized in the tapping and manipulating of company data [2]. However, the threats to companies and research institutions are complex. Therefore, the protection of these data is an essential factor not only for IT departments but for the whole entity, i.e., a company or research institution. In particular, security goals of protecting against unauthorized access and data manipulation are essential for all organizations. These and other security objectives are often endangered by breaches of the facility's IT infrastructure. Therefore, the main objective of each security department is to prevent intrusion into the IT infrastructure.

Employees of organizations are in constant contact with the respective systems—they work with them and are generally the most important source of added value in organizations. However, at the same time, they are a great threat to data and systems they deal with. The use of weak passwords, unsafe and/or insecure hardware and software, unthought-through behavior, etc., are the most common sources of IT security problems. Therefore, people, as the central interface of the IT systems of an organization, are potentially also the

greatest risk and even threat. In order to improve IT security of organizations, all people involved should, and even must, be aware of the potential dangers in advance.

Today, the term of big data in our perception is typically related to data management, i.e., the production or collection of data and their further use, with potential derivation of benefit from their reuse. However, in addition to a variety of value-adding analyses, big data technologies can also be used for safety-related calculations/activities [3]. In general, the term big data refers to valuable, complex and poorly structured and/or fast-changing data sets, particularly in the corporate environment [4]. More precisely, big data is typically described as data compliant with an nV set of properties, where n is the number of those properties, which can vary. Models 3V and 5V are generally accepted and widely used, and they describe big data with a set of features, such as volume, velocity, variety, veracity and value. They are also characterized by multifaceted interpretation, where big data covers different areas, such as technologies, analytical methods, modeling and design processes, commercial concepts and legal frameworks [5].

Big data analytics is perhaps one of the most obvious big data-related areas of research, attracting both practitioners and researchers. Their advantages are widely recognized in different areas. Furthermore, big data, and more precisely big data analytics techniques, are also seen as a tool to help in crisis management and combating epidemics and pandemics, such as the current COVID-19 pandemic, and are characterized by a great capacity to predict, map, track, monitor and raise awareness of these emergencies. Here, one of the recent studies [6,7] found that the main sources of data come from social media and internet search engines. This applies to both the identification of first "signs" of emergency, as the first news of COVID-19 was found in Twitter messages, and combating it. The latter relates to the most common techniques for analyzing these data, which involve the use of statistics, such as correlation and regression, combined with other techniques, which can potentially (and have already proved to be such) play a key role in combating the emergency and crisis management, allowing for a variety of studies and experiments that have not been conceived until now. This makes big data very attractive to health and (bio)medical areas.

However, big data storage is also known for its lower level of data protection and considerably higher interest from attackers [8]. More precisely, NoSQL databases often have a large number of data leaks.

Due to the importance of security and the potential of big data with respect to this issue, our paper aims to develop a prototype to improve the security of IT systems by means of big data analysis methods. The prototype focuses on the systematic detection of threats and attacks on data, using machine learning (ML) with the help of big data technologies. In particular, the open-source big data analysis framework Apache Spark is used to develop an intrusion detection system. It is not a secret that current security information and event management systems (SIEM) already sometimes use big data analytics technologies [9]. Using these multidimensional security analyses, organizations can automatically define the "normal state" of their data or network, thereby comparing individual actions with this "base" and thus recognizing potential risks and threats. Thus, we explore and demonstrate the capabilities provided by Apache products and services and draw conclusions on their appropriateness for pattern-based analyses utilizing unsupervised machine learning aimed at improving protection of databases in use. This is conducted on the basis of covering these approaches and presenting the prototype developed based on them. This is followed by the validation of the tool by applying it to the real systems and assessing the results.

To meet these aims, the paper is structured as follows: Section 2 refers to the background and provides the reader with the general understanding on the topic; Section 3 covers materials and methods used; Section 4 provides description of the solution provided; while Sections 5 and 6 provide discussion and results.

2. Background: General Understanding of Security, Intrusion Detection and Prevention

Data protection describes the setting of security measures to achieve security goals, which should be the highest priority, particularly in an open-communication infrastructure, such as the internet. As an example, recent studies [10,11] have presented an IoTSE—Internet of Things Search Engines—Shodan and Binary Edge-based tool for non-intrusive testing of open data sources to detect their vulnerabilities and the extent thereof. Although it referred to the analysis of open databases only, the authors' observation showed some interesting results, i.e., although the total number of open databases accessible outside the organization was less than 2% of the data sources scanned, there were data sources with low security features where it was possible to connect to nearly all IP addresses by retrieving data and information from them, i.e., which may pose risks to organizations. What is more, in some cases, the databases that did not use security mechanisms, by which NoSQL databases are mainly characterized, have already been compromised. This was the case for such popular data sources as MongoDB, followed by PostgreSQL, ElasticSearch and Memcached.

The internal network of the organization should also be viewed as an open-communication structure, since it is potentially possible that each terminal within the network has access to communication channels. The basic security objectives, mainly mentioned in the literature, are described by the terms "Confidentiality, Integrity, Availability" (CIA) [11]. Information security is typically, or should be, accompanied by data protection. These security objectives are [12]:

- the protection of data against unauthorized access;
- the protection of data against unintentional changes;
- maximizing the likelihood that system or service requirements will be met within a specified time frame.

The prevention of break-ins by setting security objectives therefore plays an important role in every organization. The best way to protect a network or sources of information systems is to detect attacks as early as possible and protect the artifact against them, even before they can cause harm/damage. In recent years, organizations have invested significant resources, both financial, time and human, into defining and implementing new security products to protect against attacks [12,13]. Here, the intrusion detection systems (IDS) enter. An intrusion detection system mainly consists of three components providing the relevant functionality [14], i.e., it works in three stages: (1) the system should collect the data, where the data collection source may vary from one case to another or be a combination of various sources, and prepare them for analysis; (2) the accumulated data should be investigated; (3) in case of an intrusion or anomaly detected, the administrators should be informed and take action. To sum up, intrusion detection is a process of monitoring events in an information system or network.

Event monitoring, however, is used to analyze "signs"/characteristics of potential security incidents that violate computer security guidelines, usage guidelines or standard security practices [15]. They, however, can be complemented with ad-hoc guidelines depending on the particular case. Moreover, the avoidance of intrusion refers to the recognition and the attempt to prevent it. Accordingly, intrusion detection and prevention systems (IDPS) are software solutions that combine the detection and prevention of incidents [16]. They primarily focus on identifying and logging possible security incidents, preventing intrusions/breaches and informing security administrators about them. In some cases, organizations also use IDPS for other purposes, such as identifying problems related to existing security policies, documenting existing threats and preventing individuals from bypassing security policies [17]. Therefore, almost all organizations should consider it necessary to supplement the security infrastructure through IDPSs [18]. Another important component that complements IDPS is Security Information and Event Management (SIEM) solutions, which offer a combination of software products and services, which were previously presented under two separate terms, i.e., Security Information Management (SIM) and Security Event Management (SEM) [19]. SIEM technologies enable real-time analysis

of security warnings generated by network hardware and applications by using the data received from the IDPS and other relevant data [9]. They also use data from correlating events logged by different technologies, displaying data from many event sources and providing supporting information from other sources to help users verify the accuracy of IDPS alerts [20].

Overall, data protection should be an ongoing priority. It is important to regularly review information systems as sources and to be vigilant in the case of suspicious changes. This is particularly important because threats are evolving at an unprecedented pace, and data about organizations' networks and servers are the preferred target. IDPSs are very suitable for detecting and stopping network attacks, including those aimed at vulnerabilities in the application and operating system, and provide IT administrators with an additional layer of protection of their infrastructures. They ensure that attempted attacks and undesired data transfers and/or manipulations with them are reported quickly and preferably also prevented. The type of IDPS that is ultimately used depends on the size and architecture of the network concerned, as well as on the requirements of the IT department and management with regard to the functions of the product [20].

In this paper, we present a prototype of the intrusion detection system developed on the basis of Apache Spark. Therefore, let us provide some motivation for this choice by elaborating on the applicability and appropriateness of Apache Spark.

3. Materials and Methods: Big Data Approach with Apache Spark and MLlib

Apache Spark is an open-source big data analysis framework that allows computing of huge amounts of data [21]. Spark is able to process databases that are distributed across a large number of machines. This project was launched in 2010 at the University of California, Berkeley, and became an Apache project in 2013. Since then, Spark has proliferated and evolved immensely. In the meantime, Apache Spark has developed into a de facto open-source standard in the big data world and is traded as a successor to Apache Hadoop MapReduce technology, mainly due to its outstanding processing speed [22]. With advanced resource management and resource utilization (such as in-memory computing), Apache Spark enables this world-class speed. In addition, it can also be easily integrated into existing Apache Hadoop systems; for example, native support for the Hadoop Distributed File System (HDFS) is built into Spark [23].

Another benefit and key for success of the Spark framework is the fact that it provides interface to a variety of programing languages, including Java, Python and R, while Apache Spark itself has been developed in Scala. Therefore, for pragmatic reasons, the source code of the prototype proposed in this study has also been developed in Scala. In addition, the Scala programing language offers a variety of contemporary and easy-to-implement concepts, such as optional parameters, immutable objects, pattern-based searches, etc. In particular, anonymous functions are widespread in Scala and are also widely used in this work. Scala programs—and thus Spark—are compiled in Java bytecode and executed in the Java Runtime Environment.

Each Spark application consists of a driver program that starts and manages various parallel operations on a computer network. The driver includes the main function and defines data records distributed by the SparkContext instance, on which arithmetic operations are then carried out. These operations are divided into individual tasks and are processed by executors on nodes in the computer network. Figure 1 shows general relationships of the components for the distributed execution in Spark.

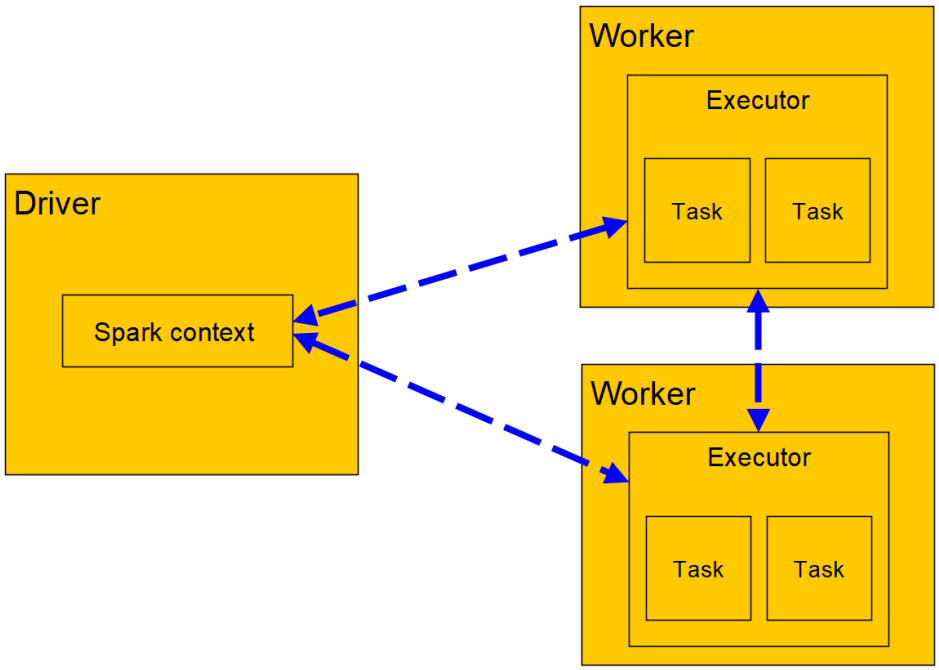

Figure 1. Components of Distributed Execution in Spark.

An essential core element and one of the most fundamental data structures of Spark is the Resilient Distributed Dataset (RDD) [24]. This is an abstraction of a database that is partitioned and therefore redistributable over a computer network. All data, e.g., data from log files or network streams, are converted to a generic RDD, which in turn is used by Spark as a processing medium. This layer of abstraction allows developers to keep data in the cluster node's memory during their processing. In this way, the costs of recalculating or outsourcing to slow data carriers can be avoided in the subsequent analysis. Thus, Spark can be considered sufficiently flexible and fault tolerant. For example, computer network node's failures or memory bottlenecks are automatically recognized during calculation and treated with different strategies without interrupting processing [25].

Another component recognized as important and beneficial for developers is Spark Machine Learning Library (MLlib). MLlib is a machine learning (ML) library optimized for use in a cluster [26]. MLlib aims to implement ML algorithms and techniques in a scalable and simple manner in Spark applications [27]. It has many pre-implemented ML algorithms, including classification, regression, clustering and others, to be re-used in developed Spark applications. Generally, the Spark Framework consists of five components, as shown in Figure 2. In light of the objectives of our paper, we are focusing and further using Spark Core and MLlib.

In computer science, machine learning (ML) is known as the sub-discipline of the generation of synthetic/artificial knowledge from data. The artificial knowledge gained allows the ML systems to make certain predictions. There are various ML algorithms that, once learned, are able to distinguish between normal/ordinary and uncommon data. If it is known in a data record or set which entries are considered "normal", and the system is informed that training can take place, for example in the form of a flag, this is called "supervised learning". The used algorithm is assigned to work in a specific predefined manner, based on known input and output data. This process is also known as training in the context of ML. Statistical classification methods are mainly used in this form of machine learning. With this method, the algorithm is able to develop decision trees to be able to prepare predictions for new data with similar patterns by means of regression analysis. Most algorithms that make predictions are based on classification and regression.

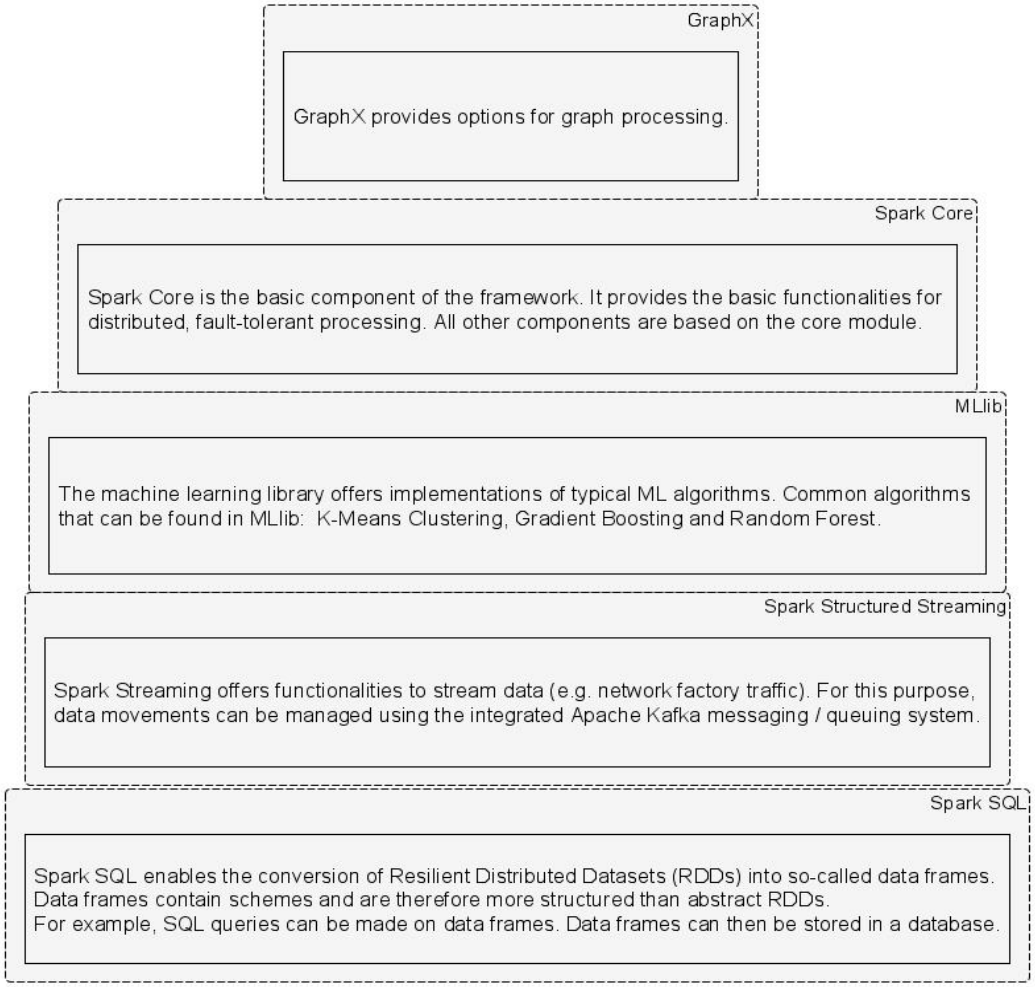

Figure 2. Apache Spark Framework.

At the same time, there are data sets for which it is previously not known which data should be considered normal or ordinary. For example, if new types of attack attempts are identified in the data, attack patterns may not be known. In this case, it is not possible to apply classification in advance. The system should rather use input data to identify independently which patterns are represented in a data record and thus evaluate which data do not match these patterns. "Unsupervised learning" can be a solution to these kinds of problems. In other words, unsupervised learning techniques are not trained on the basis of defined target data, as they are not available during the learning process. Rather, unsupervised learning algorithms capture structures in data and find groups of similar inputs or determine and learn which types of inputs might occur and which might not. Thus, in light of this, the intrusion detection system to be developed will be developed on the basis of unsupervised learning, by using the k-means clustering method implemented in MLlib from Spark.

The k-means clustering method is probably the best-researched and most widely used method of unsupervised learning. In this context, the term "cluster" should not be confused with the computer network. Cluster analysis tends to identify similarities in structures in large databases, where object groups or records that are structurally similar are called clusters. Clusters created from input data can then be used to compare new or unknown data with existing clusters. If the data to be checked exceeds a certain distance from the given clusters, i.e., the threshold value for the distance to existing clusters is exceeded, this data is referred to as an anomaly.

4. Results: Prototype of Apache Spark and MLlib-Based Intrusion Detection System from A to Z

In contrast to signature-based IDS solutions that prevail in the market, the prototype solution developed in this article is based on machine learning with the help of the big data cluster framework Apache Spark. Figure 3 shows a system framework diagram modeled in Business Process Model and Notation (BPMN) to understand the overall image of the actions performed in the system. In short, the developed system should be able to determine from the data set used, which parts of the data set are to be classified as harmless or regular, and which are conspicuous or irregular, and can quickly identify anomalies in the data. The k-means algorithm is used for cluster analysis, which is implemented in the Sparks MLlib component. The anomalies can be intentional or involuntary. The premeditated ones are malicious, and it is important to identify and discover them. The causes of anomalies vary, and if the anomaly was alerted in the solution, clustering techniques are used to perform the action that ML algorithms, such as K-means, use to divide data into groups. Instances far from all groups are identified as outliers.

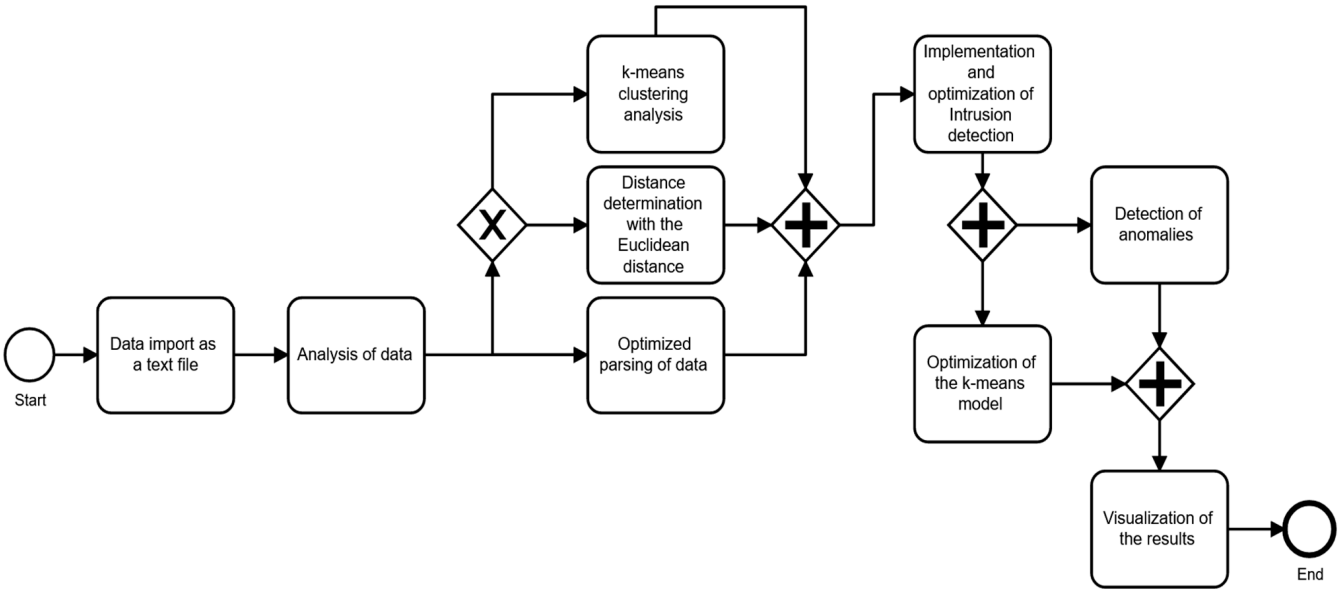

Figure 3. System Framework Diagram.

For the purposes of our experiments, we have created a database, in which we have created our own computer network for Spark and Hadoop applications. The cluster consists of a total of 24 processors and is equipped with 96 gigabytes of RAM. Spark version 1.3 is installed on the cluster. The database is available as a text file researchdatacup.data, which part is made available at https://github.com/OtmaneAzeroualDZHW/IDS-prototype/blob/main/researchdatacup.data (accessed on 20 December 2022). It is over 700 megabytes and consists of almost five million data records. Each data record consists of 42 attributes, with one data record per line, separated by commas (in a CSV-like format). The database used is sufficient both for building a rapid prototyping and as a big data example, since processing this database requires a large number of gigabytes of RAM. For this reason, a large part of the calculations was carried out within the computer cluster. In addition, feature names and feature types are listed in a further text file (researchdatacup.names). Interestingly, the organizers of the research data cup identified the type of connection or attack in each data set. There are labels such as buffer_overflow, smurf or normal. This marking could serve as a target definition of a supervised learning method. However, the main focus of the prototype under development is on detecting anomalies and potentially unknown attacks. Therefore, a method of independent, unsupervised learning is used, and the characteristics of the type of attack are deliberately ignored for actual processing. In

the course of this paper, it will also be found that some connections in the database are considered normal and are marked as such, but they represent anomalies in the sense of the analysis and are therefore to be classified as potentially dangerous.

In almost all cases of analysis, it is necessary to adapt the input data to a certain extent, so that the analysis system can process these data adequately. This process is called transformation. Therefore, the first step of any analysis is to review the data. For this purpose, we have used the Spark Shell, which offers a convenient tool for quick initial database analysis. It initializes the Spark context, which is essential for calculations (instance of the *SparkContext* class). In the following examples, the Spark Framework should be equipped with at least eight gigabytes of RAM. Therefore, Spark Shell is used on a Unix-like operating system with a parameter—*driver-memory 8 g*. Once the Spark context (identified in the following by the variable *sc*) is available, the data record may be read into an RDD (see Figure 1).

In the first line of the code provided in Figure 4, the data record is read in as RDD and in the unchangeable variable *rawData*, while line four transforms an array of strings of each comma-separated data record. The *map()* function of the RDD class is used here, which expects a function that is applied to each data record as a parameter. It should be noted that the mapped variable references a new RDD instance and that the *map()* method has not yet processed the data. When using RDDs, there are typically two types of instructions—transformations and actions.

```
val rawData = sc.textFile("researchdatacup.data")
// Output: rawData:org.apache.spark.rdd.RDD[String] =
// researchdatacup.data MapPartitionsRDD [1] at textFile at <console >:24
val mappedRdd = rawData.map(_.split(','))
val firstItem = mappedRdd.first()
firstItem.length
// Output: Int = 42
```

Figure 4. Analysis of data set in the Spark Shell.

Transformations enrich abstract RDD with meta-information, such as how RDD should be converted. However, only if there is an action Spark performs an actual, distributed calculation on an RDD and then takes into account and optimizes the previously defined transformations. The *map()* is a transformation ("convert every data set using the following function"), while the *first()* method is an action ("return the first line of a calculated RDD"). Therefore, only when *first()* is called in line seven the RDD is actually transformed, and the first data record of the transformed RDD is returned.

The first data set extracted by *first()* reveals the TCP connection to the HTTP service. In this particular case, 215 bytes were sent, and 45,076 bytes were received. All other features are listed in the *researchdatacup.names* text file.

Using the *length* method, we can determine that the data record in *firstItem* consists of 42 elements (also called columns, parameters). These 42 columns will be used as 42-dimensional vector features for clustering in the following steps. It is also important to note that the elements are not only numeric, i.e., some features are represented in Boolean (as 0 or 1), some take real values between 0 and 1, while others are symbolic (e.g., *tcp*).

As we have briefly mentioned above, *map()* expects a definition of the function. Here, we use a method provided in line four of the code provided in Figure 1, where the shortened notation for the definition of the *map()* is given. Such a notation is possible due to the flexible Scala syntax and combines the concepts of anonymous functions and an implied parameter (_). Both concepts are widespread in the Scala world and can save a lot of programing code—but mostly at the expense of legibility. The fourth line corresponds to the call provided in the code shown in Figure 5.

```
val mapped = rawData.map(line => line.split(','))
```

Figure 5. *Map()* method.

In the code of Figure 5, an anonymous function is transferred to *map()* (line =>line.split(',')). The underscore in the code available in Figure 1, line four, on the other hand, is implicitly interpreted by Scala as a line parameter and thus, in turn, shortens the long form of the anonymous function.

In the database analysis, data records were transformed into the RDD, separated by commas. However, this is not enough to process the data in a meaningful way, i.e., symbolic features must be converted into numeric ones. The reason for this is that the later calculation of the distances between feature values and clusters can only be performed using numerical values. For this reason, the parsing of the data records must be optimized by encoding categorical (symbolic) features into numerical ones. For example, to find out which symbolic values the *protocol_type* feature assumes, the following Spark statement available in Figure 6 can be used.

```
rawData.map(_.split(',')(1)).countByValue()
```

Figure 6. Values for *protocol_type* property.

The expression *_.split(',')(1)* splits as before using a comma and takes only the second column (index (1)), while the *countByValue()* action counts all occurrences. It can be seen here that *protocol_type* takes on the values Transmission Control Protocol (TCP), User Datagram Protocol (UDP) and Internet Control Message Protocol (ICMP). One-hot coding is used here to numerically represent these characteristic values. Due to the one-hot coding, each categorical characteristic takes on n possible numerical characteristics, with a value for each placeholder of 0 or 1. In the case of the log type or data type, the feature may have three different values, hence n = 3. For example, if a data record has a TCP log type, the one-hot representation is "1,0,0"—analogously "0,1,0" for UDP and "0,0,1" for ICMP. Features two through four are categorical; they are transformed into one-hot. It should be noted that in order to do this, all possible value options need to be known beforehand.

After the calculation performed by Spark, the *symbolData* variable references an instance of the Map type, which itself references three Map instances (see Figure 7). In Scala, Map is an iterable collection consisting of key–value pairs (see dictionaries or hashes in other programing languages). *symbolData("protocols")*, *symbolData("services")* and *symbolData("states")* contain *[String, Int]*-maps with all possible occurrences of value variants assigned to the index. The *distinct* method returns only one-time occurrences, while *collect* returns all (calculated) elements. *zipWithIndex* creates a tuple with a value and an index number for each element, and *toMap* converts the array to a map. A representation of *symbolData("services")* output shows the exemplary result of the transformation. Using this information, the numerical feature vectors can then be developed.

```
val symbolData = Map(
" protocols " -> rawData.map(_.split(',')).map(_(1)).distinct.collect.zipWithIndex.toMap,
" services " -> rawData.map(_.split (',')).map(_(2)).distinct.collect.zipWithIndex.toMap,
" states " -> rawData.map(_.split(',')).map(_(3)).distinct.collect.zipWithIndex.toMap
)
```

Figure 7. Transformation of categorical characteristics into three indexed map instances.

The *getFeatureVectors()* function in provided in Figure 8 transforms *rawData* and the data from *symbolData* into an RDD of feature vectors with exclusively numerical values. In order to increase the performance of the program, *symbolData* is not recalculated every time this function is called but is instead transferred as a parameter because *symbolData* can, for example, be transferred to all cluster nodes (via *sc.broadcast()*) to avoid time-consuming recalculations.

```
import org.apache.spark._
import org.apache.spark.rdd._
import org.apache.spark.mllib.linalg._

def getFeatureVectors(rawData: RDD[String], symbolData: Map[String,
Map[String, Int]]): RDD[Vector] = {rawData.map {line =>
```

Figure 8. *getFeatureVectors()* for creating purely numerical feature vectors.

It should be noted that *getFeatureVectors()* supplies an RDD from vector instances required for the k-means algorithm. In general, the *getFeatureVectors()* function parses a database from *rawData* together with the data from *symbolData*, which can be used for further processing in the form of an RDD from the feature vectors.

The k-means algorithm is already implemented in Apache Spark, and the first model can be created with lines of source code. The first k-means model is developed using the code provided in Figure 9.

```
import org.apache.spark.mllib.clustering._

val vectorRdd = getFeatureVectors(rawData, symbolData).cache()
val kMeansModel = (new KMeans()).run(vectorRdd)

kMeansModel.clusterCenters.length
// Output: Int = 2 (number of cluster focal points)
```

Figure 9. k-means model generation.

The RDD from feature vectors in line two (Figure 9) is cached with the *cache()* function after the first action has been carried out. This can be performance enhancing if Spark would recalculate the RDD internally in later use, for example, when calculating the k-means cluster. With *cache()* the state of the RDD is explicitly recorded after the next action. In this first model, *kMeansModel*, the number of clusters is k = 2. In order to achieve good results by means of k-means, it is essential to choose the value for k that is appropriate for the database. However, the number of possible connection types is significantly higher than 2, so the value for k is not yet optimal.

After the next cluster focus has been assigned to a feature vector of a data record, and the distance between these vectors has been calculated, possible attacks or intrusions may be detected. However, the k-means model used should be optimized to be able to achieve the most accurate anomaly detection possible.

Potentially suspicious data deviate from previously trained patterns. The distance between the data points of unnatural connection attempts exceeds the distance of ordinary connections—the distance threshold is exceeded. If the feature vector exceeds the threshold value, an anomaly is identified. However, the optimal threshold is determined by referring to the average distance to cluster centers (see Figure 10).

```
val distanceThreshold = vectorRdd.map{distanceToCentroid (_, kMeansModel)}.mean()
// Output: distanceThreshold: Double = 3423.224724378211
```

Figure 10. Calculation of the average distance threshold of a vector RDD.

It should be noted that *distanceThreshold* depends on the calculated k-means model (particularly the number of clusters, i.e., k) and can also deviate from the above-mentioned value (around 3423) with k = 2. The reason is that when the k-means model is initialized, randomly selected cluster priorities are set that are only adjusted by real values in the course of the calculation. The anomalies can now be calculated immediately.

In the code provided in Figure 11, an RDD is created with tuples of the original data records and their respective feature vectors. This RDD is then filtered using the feature vectors that exceed the distance threshold. Using Scala's pattern-matching mechanism (case *(data, featureVector)*), an RDD can be elegantly filtered from the tuples using individual tuple objects. Since we are interested in only the anomalies in original data sets, mapping ends with the keys method, which in turn discards the feature vectors of the tuple RDD. The anomaly rate is calculated as a percentage, where, on the basis of the given k-means model, around four percent (4%) of the database can be assessed as abnormal, i.e., having anomalies. Figure 12 shows the code to be used to get an overview of detected anomalies.

```
// Create an RDD from (data set, feature vector) tuples
val dataFeatureVectorTuple = rawData.zip(vectorRdd)

//Filter anomalies of the feature vectors
val anomalies = dataFeatureVectorTuple.filter{case (data, featureVector)
=> distanceToCentroid (featureVector, kMeansModel) > distanceThreshold}.keys

// Calculation of the anomaly rate:
val anomalyRate = (anomalies.count.toDouble / vectorRdd.count.toDouble) * 100
// Output: anomalyRate: Double = 3.9095171494709224
```

Figure 11. Detection of anomalies in the vector RDD.

```
anomalies.take(5).foreach(println)
//output:
// 0 ,tcp ,http ,SF ,215  ,45076 ,[...], normal
// 0 ,tcp ,http ,SF ,162  ,4528  ,[...], normal
// 0 ,tcp ,http ,SF ,181  ,5450  ,[...], normal
// 0 ,tcp ,http ,SF ,185  ,9020  ,[...], normal
// 0 ,tcp ,http ,SF ,181  ,5450  ,[...], normal
```

Figure 12. Output of the first five anomalies.

What we have identified here is that the connections with the categorical characteristic *normal* were clearly identified as anomalies. These relationships may refer to attacks that are not known during categorization or false identifications due to the unoptimized k-means model.

For the latter case, in order to optimize the number k of clusters in the model, a clustering analysis can be rated as sufficient if each data point is relatively close to the assigned cluster center of gravity. To determine an improved k-value, a number of different k-means models are created, and the resulting average distance thresholds are compared with each other.

For this purpose, the code provided in Figure 13 is used, which output is shown in Table 1.

Table 1. Distance thresholds with values for k between 10 and 100.

k	Distance Threshold
10	1612.208842642998
20	1072.0327584689705
30	916.5280722240193
40	430.3117062907457
50	464.47441715464714
60	349.7187930452894
70	299.64043177835646
80	200.98089018688424
90	234.49320063282278
100	112.83995725918483

```
var k = 0
// As long as k is between 10 and 100, increment by 10 each time
for (k <- 10 to 100 by 10) {

// Create new k-means model for "vectorRdd" with specific number of clusters from "k"
val kMeansModel = (new KMeans()).setK(k).run(vectorRdd)

// Calculate distance threshold with new k-means model
val distanceThreshold = vectorRdd.map{distanceToCentroid (_,kMeansModel)}.mean()

// Output of the current distance threshold
println (" distance threshold for k=" + k + ": " + distanceThreshold)
}
// Output in table 1
```

Figure 13. Calculation of the distance thresholds for different k-means models.

It can be seen that the increase in clusters' numbers leads to a decrease in the average distance threshold, i.e., an improvement in the result. However, the k should not be too large, as otherwise, a meaningful clustering analysis cannot be performed. For example, if k is set to the number of data records (*rawData.count*), the distance will shrink to 0 because each data record will form its own cluster. What is more, the output shows that the distance thresholds do not change proportionally to k. In addition, the result with k = 50, for example, is worse than that with k = 40. This result, which is something unexpected, is also justified by the fact that the k-means method uses random initial clusters. Therefore, the choice of the number of clusters can be further improved if the model for a specific k is created several times with different random initial clusters, and then the best model resulting therefrom is adopted. The Apache Spark Framework offers the *setRuns()* method for objects of the k-means type for this purpose. Another potential improvement to the model is the lengthening of the iteration of the algorithm. K-means uses a threshold value that controls from when the movement of the cluster centers of gravity is considered to have converged, and the cluster centers of gravity therefore change. This threshold value can be adjusted with the *setEpsilon()* method.

Table 2 provides the results of this action, i.e., the values of the distance threshold with the adjusted parameters for the number of runs and epsilon. In this case, the runs were set to 10 (default: 1) and epsilon was set to 1.0×10^{-7} (default: 1.0×10^{-4}), used as a sample. What is important here is that experiments with multiple different values should be carried out to achieve better results.

Table 2. Distance thresholds with *setRuns()* and *setEpsilon()* of the k-means model.

k	Distance Threshold
10	1673.4592273822652
20	1314.1118189071337
30	888.5934838435937
40	740.1572895242036
50	339.5696447982709
60	308.62695126344164
70	262.08132872357226
80	175.89086069939748
90	148.64064822065643
100	121.79488890567019

Here we can observe an improvement compared to standard. In addition, this time the values are constantly decreasing the higher the parameter k is selected. Further optimization can be achieved by normalizing each feature through standardization (also known as z-transformation), which is a statistical function typically used when differently distributed random variables, e.g., the values of the characteristics, should be compared [28]. The z-transformation is given by the following function:

$$Z_i = \frac{X_i - \mu_i}{\sigma_i} \quad (1)$$

Equation (1). Z-transformation.

Z-transformation shall be carried out following the formula provided in Equation (1), where Z_i is the z-transformed sample; X_i is the original value of the sample; μ_i is the mean value of the sample; and σ_i is the standard deviation of the sample. In the case of a feature, the feature is standardized by subtracting the feature value with the arithmetically average mean value of the feature and dividing the result by the standard deviation.

The generation of k-means models from Figure 14 with the *epsilon* and *runs* parameters, specified in Table 2 on the feature vectors *standardizedVectorRdd* (instead of *vectorRdd*), allows for achieving better distance threshold values. At first glance, they differ significantly from the values given in Table 2. This is due to z-transformation but does not have a negative effect on the accuracy of detection of anomaly; rather, it improves the result of their recognition, if standardized feature vectors are also used.

```
// normalization of a vector with the help of the z-transformation (standardization)
def zTransform (featureVector:Vector, featureMeans:Array[Double],
standardDeviations:Array[Double]) = {val transformedArray = (
    featureVector.toArray, featureMeans,standardDeviations).zipped.map{
    case (featureVector, featureMean, standardDeviation) =>
// Avoid undefined division with standard deviation of 0
if (standardDeviation <= 0) {
(featureVector - featureMean)
} else {
(featureVector - featureMean) / standardDeviation
}
}
// Generate feature vectors from "transformedArray"
Vectors.dense(transformedArray)
}
// Standardize feature vectors
val standardizedVectorRdd = vectorRdd.map (
    zTransform(_,featureMeans.standardDeviations)).cache()
```

Figure 14. Normalization of the feature vectors through standardization.

Table 3 shows that the standardized distance thresholds continuously improve up to k = 175. Thus, the above optimizations can now be incorporated into the anomaly detection. The following results have now been calculated.

Table 3. Distance thresholds optimized by normalization.

k	Normalized Distance Threshold
10	0.09991613251731102
25	0.08518438371789444
40	0.0682862780277426
55	0.053834029418180274
70	0.03348989872873735
85	0.02167286233830281
100	0.016910185699185214
115	0.012838884998591945
130	0.008406800382889148
145	0.008095400982023805
160	0.00617585994122557
175	0.003462104571261407
190	0.004934244635581448
205	0.003173765003862672
220	0.003787212788242129
235	0.002787838315775503

As can be seen in Table 4, even after adapting the k-means models and applying normalization, there is a certain amount of leeway for interpretation and improvement. From the data we have obtained, it is possible to predict volumes of data arose from attacks based on the anomaly rates, where the arithmetic mean of the anomaly rates is 16.4 percent, with the median of 18.9 percent. There is, therefore, sufficient evidence that 15 to 20 percent of the entire database can be attributed to the attempted attacks. This means that the parameters of IDS in production, in which we suppose the proposed solution to be in-built, should always be adjusted with the following monitoring of the results, particularly in the introductory phase.

Table 4. Anomaly rates after optimizations.

k	Anomaly Rate (in Percent)
10	4.575403838494408
20	6.116815772234007
30	7.215065395429679
40	7.412065393429679
50	8.62727692010973
60	8.821620637302026
70	9.283646130771261
80	16.541010784881934
90	17.692563190131697
100	18.892396361202188
110	22.17444728730485
120	36.08357451600319
130	23.480151093278646
140	21.05892682779445
150	22.29485318870471
160	19.1206531234185
170	20.949830670269726
180	20.31542344885536
190	17.538350545307264
200	19.67231956518322

Figure 15 illustrates an abbreviated extract of detected anomalies with a k = 100 model.

```
anomalies.sample(false,0.00001).foreach(println)
// Output:
// 0 ,tcp  ,http    ,SF   ,230  ,14954 ,[...], normal
// 0 ,icmp ,eco_i   ,SF   ,8    ,0     ,[...], ipsweep
// 3 ,tcp  ,smtp    ,SF   ,660  ,328   ,[...], normal
// 2 ,tcp  ,smtp    ,SF   ,819  ,331   ,[...], normal
// 0 ,tcp  ,http    ,REJ  ,0    ,0     ,[...], normal
// 0 ,tcp  ,auth    ,S0   ,0    ,0     ,[...], neptune
// 0 ,tcp  ,private ,RSTR ,0    ,0     ,[...], portsweep
// 0 ,tcp  ,private ,REJ  ,0    ,0     ,[...], neptune
// 0 ,tcp  ,private ,REJ  ,0    ,0     ,[...], neptune
```

Figure 15. Extract of recognized anomalies with k = 100.

The *sample()* function returns a random sample of an RDD. Since no replacement is to be made, the first parameter is set to *false*. The second parameter, however, defines the size of the sample, i.e., in this case, 0.00001 of all anomalies should be detected.

Sufficiently high result is also possible using the randomized initial clusters of the k-means algorithm, but a relatively precise approximation is certainly possible. In the end, the question always remains: which connections are to be classified as irregular and potentially dangerous and which are to be classified as normal? The algorithm cannot make a final decision, but it can support decision makers. However, an optimization of the k-means models is necessary in any case, to avoid very falsified results, as in Figure 8. In addition, it is important to monitor the detection of anomalies and to carefully investigate possible outliers.

To this end, visualized results may be an opportunity for simple monitoring. Apache Spark does not provide libraries for visualizing results; other tools should be used to create representations. We, however, have used Apache Zeppelin, and Figure 16 shows visualized anomaly rates from Table 4 using Apache Zeppelin. Although the export of the results from Spark and their visualization using something akin to R^3 is also an option, in practice, when selecting an adequate k-means model, anomalies could be represented in data sets.

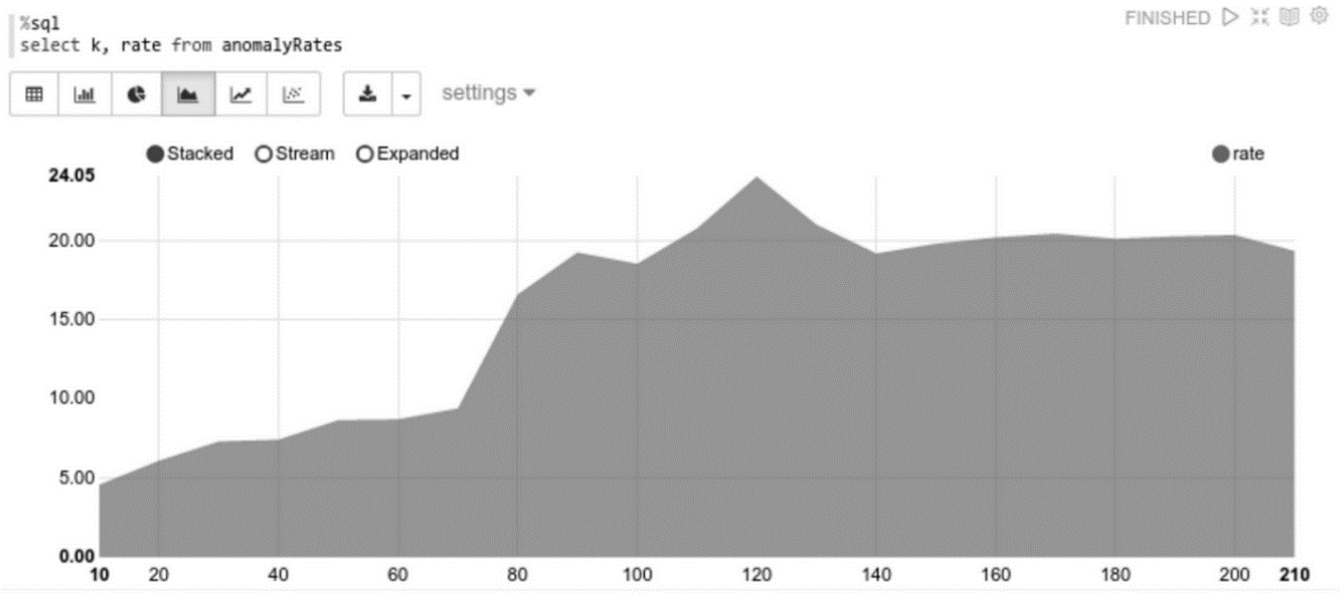

Figure 16. Anomaly rates.

The anomaly rates are included in an RDD *anomalyRates*, which consists of (k, Rate)-tuples, and are converted to a Spark Data Frame. The data frame is based on the RDDs but also has a scheme. The scheme (see Figure 16) is defined by the class rates. Unlike simple RDDs, data frames can be easily queried using Spark SQL. Zeppelin is able to visualize result sets directly from Spark SQL queries.

5. Discussion

To sum up, the prototype developed is able to detect anomalies in the database that can be used as additional mechanisms for data protection. In other words, we are not proposing a "silver bullet" but rather demonstrate how a complementary service can be added to the system in use to enhance the level of its security. What is more, our study explored capabilities provided by Apache products and services, which resulted in a conclusion that they are worthwhile to be used for protecting databases in use. There are several additional takeaways.

First, it is clear that the topic we have elaborated on is of fundamental importance in a real-world system, i.e., production environment. Therefore, not only the involvement of a data protection mechanism should take place but also the analysis of how the system reacts when an anomaly is detected. For example, an email could be sent to the IT security officer, or a ticket could be created in a corresponding system, thereby signaling about the detected anomaly. However, other options can also be used, such as destroying data identified as malicious or canceling the connection, thereby preventing data manipulation. However, it is clear that, in this particular case, the accuracy of the algorithm should be well tested. Another more advanced option could be not only the identification of anomalies or even intrusion, but also the recovery of data if changes have been made with data stored in the system.

Taking a step back to the accuracy of detection of anomalies, where optimization of the anomaly detection is crucial, the prototype achieved a detection rate of 65% in identifying anomalies in the test data set. In other words, it is important not only to achieve the detection of as many anomalies as possible, but also their relevance and compliance with actual anomalies, avoiding false positive results. These misdetections can lead to high costs in terms of both money and time spent on their inspection. This, in turn, can negatively affect the acceptance of the system and overall satisfaction level. For this reason, monitoring and optimization must be given high priority, starting with the introductory phase and beyond. There are several studies whose authors have tried to use methods such as the K-means Gaussian Interval Type 2 Fuzzy Set Prediction Method and models such as the Collaborative Intrusion Detection Model to improve network security and identify attacks and anomalies to be prevented [29,30]. As stated in [29], intrusion detection is becoming increasingly important as primary defense technology, because network security is one of the most critical issues. In addition, attacks and anomalies are often serious and can range from misuse of the systems to production downtimes and failure of vital control systems.

Given that some above-mentioned implications concern people, it should be noted that the person who interacts with the system, i.e., the user, remains one of the critical factors, and therefore their education and digital literacy, or security literacy, to be more precise, is crucial. This was also proved by the statistics provided by the recent Data Breach Investigations Report, according to which 85% of breaches in 2021 involved a human element, while social engineering was recognized as the most popular pattern [31]. The reason for this is that even in the case of highly mature mechanisms of data and system protection involved and maintained, the human factor remains to be seen as very difficult to control. Therefore, the training, instruction and education of system users, as well as defining, introducing and maintaining security policies should be complementary to the various technical advances.

6. Conclusions

The big data Apache Spark Framework offers extensive capabilities for analyzing massive amounts of data. Without further adjustments (out-of-the-box), Spark is able to distribute the calculations to (theoretically) any number of machines in a fault-tolerant manner. The outstanding flexibility and speed of execution thus provides a solid basis for a large number of data analyses. In this paper, we have provided an example of how the big data technologies and above-mentioned services can be used not only for everyday tasks but also for the protection of the data produced, collected, processed and transferred.

This paper explored and demonstrated capabilities of MLlib components and its appropriateness for pattern-based analyses, utilizing unsupervised machine learning aimed at improving IT security. However, the use of a big data system to detect security-critical anomalies must not be carried out by installing and configuring the solution on a one-time basis, but rather as a process that has to be continuously monitored and optimized.

In addition to [32], our solution offers the opportunity to gain valuable knowledge from large-scale structured, unstructured and rapidly changing data, thereby providing users with support for decision making. It is also clear that different solutions, which have different goals and requirements, were discussed in the literature. Unfortunately, we could not compare this to our solution because the Apache Spark Framework lacks extensive test support. This is particularly interesting when processing big data, as it should be possible to define certain test scenarios to check the respective processing algorithms. This remains an open question for future study.

In addition to Spark, there are other solutions that are suitable for big data applications. One particularly interesting subject for our future research is the open-source project Apache Flink. Although relatively small compared to Spark, Flink, like Spark, focuses on distributed processing of large amounts of data. In contrast to Hadoop or Spark, Flink is not primarily focused on fast batch processing, but rather on processing of data streams.

As we have mentioned, the developed service is a prototype. We are continuing our work on it and expect to advance it in the future by our own forces or by means of co-creation, where every interested person is welcome to contribute to the prototype by referring to the source code (https://github.com/OtmaneAzeroualDZHW/IDS-prototype) (accessed on 20 December 2022), thereby supporting the open science movement.

Author Contributions: O.A. and A.N. contributed to the design and implementation of the research, to the analysis of the results and to the writing of the manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data available on request from the authors. The source code and input data are available at https://github.com/OtmaneAzeroualDZHW/IDS-prototype, accessed on 20 December 2022.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Yang, F.; Gu, S. Industry 4.0, a revolution that requires technology and national strategies. *Complex Intell. Syst.* **2021**, *7*, 1311–1325. [CrossRef]
2. Jang-Jaccard, J.; Nepal, S. A survey of emerging threats in cybersecurity. *J. Comput. Syst. Sci.* **2014**, *80*, 973–993. [CrossRef]
3. Beall, J. Metadata and Data Quality Problems in the Digital Library. *J. Digit. Inf.* **2005**, *6*, 1–20.
4. Yu, S.; Guo, S. *Big Data Concepts, Theories, and Applications*; Springer: Berlin/Heidelberg, Germany, 2016.
5. Dorsche, J. *Praxishandbuch Big Data*; Springer: Berlin/Heidelberg, Germany, 2015.
6. Corsi, A.; de Souza, F.F.; Pagani, R.N.; Kovaleski, J.L. Big data analytics as a tool for fighting pandemics: A systematic review of literature. *J. Ambient. Intell. Humaniz. Comput.* **2021**, *12*, 9163–9180. [CrossRef] [PubMed]

7. Özgüven, Y.M.; Eken, S. Distributed messaging and light streaming system for combating pandemics. *J. Ambient. Intell. Humaniz. Comput.* **2021**, 1–15. [CrossRef]
8. Yang, P.; Xiong, N.; Ren, J. Data security and privacy protection for cloud storage: A survey. *IEEE Access* **2020**, *8*, 131723–131740. [CrossRef]
9. González-Granadillo, G.; González-Zarzosa, S.; Diaz, R. Security Information and Event Management (SIEM): Analysis, Trends, and Usage in Critical Infrastructures. *Sensors* **2021**, *21*, 4759. [CrossRef] [PubMed]
10. Daskevics, A.; Nikiforova, A. ShoBeVODSDT: Shodan and Binary Edge based vulnerable open data sources detection tool or what Internet of Things Search Engines know about you. In Proceedings of the International Conference on Intelligent Data Science Technologies and Applications (IDSTA2021), Tartu, Estonia, 15–16 November 2021.
11. Daskevics, A.; Nikiforova, A. IoTSE-based Open Database Vulnerability inspection in three Baltic Countries: ShoBEVODSDT sees you. In Proceedings of the 8th International Conference on Internet of Things: Systems, Management and Security (IOTSMS), Valencia, Spain, 6–9 September 2021.
12. Popescul, D. The Confidentiality–Integrity–Accessibility Triad into the Knowledge Security. A Reassessment from the Point of View of the Knowledge Contribution to Innovation. In Proceedings of the 16th International Business Information Management Association Conference (Innovation and Knowledge Management, A Global Competitive Advantage), Kuala Lumpur, Malaysia, 29–30 June 2011; pp. 1338–1345.
13. Zuech, R.; Khoshgoftaar, T.M.; Wald, R. Intrusion detection and Big Heterogeneous Data: A Survey. *J. Big Data.* **2015**, *2*, 3. [CrossRef]
14. Khraisat, A.; Gondal, I.; Vamplew, P.; Kamruzzaman, J. Survey of intrusion detection systems: Techniques, datasets and challenges. *Cybersecurity* **2019**, *2*, 20. [CrossRef]
15. Borky, J.M.; Bradley, T.H. Protecting Information with Cybersecurity. In *Effective Model-Based Systems Engineering*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 345–404.
16. Sanders, C. The Practice of Applied Network Security Monitoring. In *Applied Network Security Monitoring*; Sanders, C., Smith, J., Eds.; Syngress: Rockland, MA, USA, 2014; pp. 1–24. ISBN 9780124172081. [CrossRef]
17. Caballero, A. Information Security Essentials for IT Managers: Protecting Mission-Critical Systems. In *Computer and Information Security Handbook*; Elsevier: Amsterdam, The Netherlands, 2014; pp. 1–45.
18. Johnson, L. Chapter 11—Security component fundamentals for assessment. In *Security Controls Evaluation, Testing, and Assessment Handbook*, 2nd ed.; Academic Press: Cambridge, MA, USA, 2020; pp. 471–536. [CrossRef]
19. Kakareka, A. Chapter 1—Detecting System Intrusions. In *Network and System Security*, 2nd ed.; Syngress: Rockland, MA, USA, 2014; pp. 1–27. [CrossRef]
20. Scarfone, K.; Mell, P. *Guide to Intrusion Detection and Prevention Systems (IDPS)*; NIST Special Publication: Gaithersburg, MD, USA, 2007; 127p. [CrossRef]
21. Salloum, S.; Dautov, R.; Chen, X.; Peng, P.X.; Huang, J.Z. Big data analytics on Apache Spark. *Int. J. Data Sci. Anal.* **2016**, *1*, 145–164. [CrossRef]
22. Liu, X.; Iftikhar, N.; Xie, X. Survey of real-time processing systems for big data. In *Proceedings of the 18th International Database Engineering & Applications Symposium (IDEAS '14), Porto, Portugal, 7–9 July 2014*; Association for Computing Machinery: New York, NY, USA, 2014; pp. 356–361.
23. White, T. *Hadoop: The Definitive Guide*; O'Reilly Media Inc.: Sebastopol, CA, USA, 2012.
24. Sakr, S. General-purpose big data processing systems. In *Big Data 2.0 Processing Systems*; Springer: Cham, Switzerland, 2016; pp. 15–39.
25. Van Steen, M.; Tanenbaum, A.S. A brief introduction to distributed systems. *Computing* **2016**, *98*, 967–1009. [CrossRef]
26. Meng, X.; Bradley, J.; Yavuz, B.; Sparks, E.; Venkataraman, S.; Liu, D.; Freeman, J.; Tsai, D.B.; Amde, M.; Owen, S. MLlib: Machine Learning in Apache Spark. *J. Mach. Learn. Res.* **2016**, *17*, 1235–1241.
27. Qolomany, B.; Al-Fuqaha, A.; Gupta, A.; Benhaddou, D.; Alwajidi, S.; Qadir, J.; Fong, A.C. Leveraging Machine Learning and Big Data for Smart Buildings: A Comprehensive Survey. *IEEE Access* **2019**, *7*, 90316–90356. [CrossRef]
28. Simon, M.K. *Probability Distributions Involving Gaussian Random Variables. A Handbook for Engineers and Scientists*; Springer: Boston, MA, USA, 2002. [CrossRef]
29. Teng, S.; Wu, N.; Zhu, H.; Teng, L.; Zhang, W. SVM-DT-based adaptive and collaborative intrusion detection. *IEEE/CAA J. Autom. Sin.* **2018**, *5*, 108–118. [CrossRef]
30. Li, R.; Huang, Y.; Wang, J. Long-term traffic volume prediction based on K-means Gaussian interval type-2 fuzzy sets. *IEEE/CAA J. Autom. Sin.* **2019**, *6*, 1344–1351. [CrossRef]
31. Verizon. 2021 Data Breach Investigations Report (DBIR). 2021. 119 Pages. Available online: https://www.verizon.com/business/resources/reports/2021/2021-data-breach-investigations-report.pdf (accessed on 20 December 2021).
32. Azeroual, O.; Fabre, R. Processing Big Data with Apache Hadoop in the Current Challenging Era of COVID-19. *Big Data Cogn. Comput.* **2021**, *5*, 12. [CrossRef]