Machine Learning Enhanced Access Control for Big Data

Hamza ES-SAMAALI , Anas Abou El Kalam, Aissam Outchakoucht, Siham Benhadou

LISER Laboratory, Hassan II University, ENSEM School, Casablanca, Morocco Cadi Ayyad University
ENSA School, Marrakech, Morocco LISER Laboratory, Hassan II University,ENSEM School, Casablanca, Morocco
LISER Laboratory, Hassan II University,ENSEM School, Casablanca, Morocco

Abstract
Access Controls (AC) are one of the main means of defense in IT systems, unfortunately, Big Data Systems are still lacking in this field, the current well-known ACs are vulnerable and can be compromised because of policy misconfiguration and lack of contextuality. In this article we propose a Machine Learning approach to optimize ABAC (Attribute Based Access Control) with the aim to reduce the attacks that are overlooked by the hardcoded policies (i.e: users abusing their privileges). We use unsupervised learning outlier detection algorithms to detect anomalous user behaviors. The Framework was implemented in Python and its performance tested using the UNSW-NB15 Data Set.

Key words:
Access Control; Big Data; Machine Learning; Outlier Detection; ABAC; Security

1. Introduction:

The quantitative explosion of digital data has forced researchers to find new ways of seeing and analyzing the world. This means discovering new orders of magnitude in the capture, retrieval, sharing, storage, analysis and presentation of data. Thus, the "Big Data" was born. It is a concept for storing an indescribable amount of information on a numerical basis.
Invented by the giants of the web, Big Data is a solution designed to allow everyone to access giant databases in real time. It aims to offer a choice to classic database and analysis solutions (Business Intelligence platform in SQL server...).
This concept brings together a family of tools that respond to a problem known as the 3V rule. These include a considerable Volume of data to be processed, a wide Variety of information (from various sources, unstructured, organized, open...) and a certain level of Velocity to be reached, i.e. the frequency with which this data is created, collected and shared.
Numerous technological advances have driven this phenomenon forward. The most important are the emergence of NoSQL datastores [1] and distributed computing paradigms such as MapReduce [2], which collectively opened the way to managing and systematically analyzing large quantities of semi-structured information (e.g. transactions, digital records and emails).
Overall, the traditional data management schemes cannot provide support for storing and analyzing large and heterogeneous datasets, thus the need for specific Big Data Platforms. These new systems not only provide exceptional flexibility and efficacy of the analytical services, but also outperform traditional systems, even in terms of performance and scalability.
However, BigData schemes do not have the same standards of information security characteristics [3]. While a range of data protection frameworks have been suggested for traditional schemes (see, for example [4, 5, 6]) these frameworks cannot work for Big Data platforms. Unconstrained access to large data volume, the sensitive and private material of certain information resources, and the sophisticated analytical and predictive capacities of Big Data analytical systems could pose a severe threat.There is also another issue rising with the public availability of large data sets: Big Data analysis tools can be used to infer more personal identity user data. As it was shown in [7] and [8] where they used publicly available data sets to de-anonymize the supposedly Anonymized Netflix Data Set [9]. As a result, while the prospective advantages of Big Data analytics are unquestionable, the absence of conventional data protection instruments is constituting a friendly environment for prospective attackers.
A very challenging study task is defining appropriate information security instruments tailored for Big Data systems. As said earlier security enforcement techniques proposed for traditional systems can not be used for Big Data. The task, therefore, is to protect privacy and confidentiality while not hindering data analytics and exchange of information. Additional elements, such as the multitude of data models and data analysis and processing tools used by Big Data systems, add to the difficulty of this objective. Indeed, Big Data applications, distinct from RDBMSs, are defined by distinct data models [1] , the most noticeable being key-value, wide columns and document-oriented applications.
Before proceeding with the presentation of our contribution, we must first define a number of criteria that should be satisfied by any access control solution for Big Data systems.

Manuscript received March 5, 2020
Manuscript revised March 20, 2020

2. Requirements for big data access controls:

In this chapter, we provide an outline of the main criteria for establishing a Big Data access control system.

A. Fine-grained access control:

Fine grained access control (FGAC) has been commonly acknowledged as one of the basic components for efficient security of private and delicate information in terms of characteristics that should be supported by the access control system [4, 10].
Since information handled by Big Data analytics systems often refer and process user private features, it is essential that access control rules should be linked to the data at the finest granularity levels. The associated enforcement processes however, needs to be developed from scratch, as those suggested for conventional systems depend on data relating to a known schema, while data is heterogeneous and does not follow one exact schema in big data.

B. Context Management:

Another important factor to consider is supporting context-based entry restrictions, as these enable for extremely tailored types of access control. They can be used, for example, to restrict entry to particular time intervals or geographical locations. In the event that contexts are used to generate access control decisions, access authorisations are given when requirements are met relating to environmental characteristics.

C. Efficiency of Access Control:

The features of the Big Data environment, such as the distributed design, the complexity of the queries and the reliance on efficiency, involve implementation policies for access control that do not change the usability of the analytical frameworks. Based on this, the number of checks to be performed during access control enforcement may match or be even greater than the number of data records, and up to hundreds of millions of such records may be included in the Big Data scenario. This needs efficient policy compliance mechanisms. Two primary methods have implemented FGAC in traditional relational DBMSs. The first is view-based, where users are only allowed to access a portion (view) of the target dataset that meets the restrictions on access control, while the second is based on the rewriting of queries: under such a method, the request is altered at runtime by injecting restrictions placed by the designated access control policies instead of pre-computing the authorized views. Therefore, it is essential to determine how appropriate these methods are for the Big Data case and how they may be tailored or expanded.

3. Related work:

There are several suggestions in the literature that tackle the problem of access control for Big Data and meet some of the criteria outlined in the last chapter [13]. We can classify these ideas into two primary classifications:

A. Platform specific approaches:

Access control frameworks in this group are intended for only one system (e.g., MongoDB, Hadoop) and may leverage the protected platform's indigenous access control attributes. The primary benefit of this approach is that it is possible to optimize the designed access control solution for the target system, but its usability and interoperability are very restricted.

B. Platform independent approaches:

The methods falling within this category suggest alternatives for access control that do not only target a particular platform. Platform independent approaches have the advantage of being more general than specific platform solutions, but they lack in terms of efficiency as they cannot compete with platform specific ones. The approaches in this category focus mainly on efforts attempting to define a unifying query language for NoSQL datastores like JSONiq [11] and SQL++ [12].

4. Contribution

In this paper, we suggest a platform independent access control framework. It’s basically an improved version of ABAC [40] to prevent the exploitation of misconfiguration. Our proposed solution makes it possible to optimize access control rules based on behavioral characteristics that are tracked at runtime. Our designed system, called DABAC (Dynamic Attribute Based Access Control), takes advantage of Machine Learning algorithms aimed at detecting unusual and anomalous user behaviours to accurately tune policies. This will both improve insider threat detection and access control [14,15].
In this chapter we will expand on details of all the components of our framework.

A. ABAC:

The Attribute-Based Access Control (ABAC), also known as policy-based access control, defines a paradigm access control in which access rights are granted to users through the use of policies that combine attributes together. Policies can use any type of attributes (user attributes, resource attributes, object, attribute environment, etc.). This model supports Boolean logic, in which rules contain " if, then " about who makes the request, the resource, and the action. For example, if the requester is a manager, then allow read/write access to sensitive data.

Unlike role-based access control (RBAC)[16], which employs predefined roles that have a set of specific privilege measures associated with them and to which subjects are assigned, the main difference with ABAC is the concept of policies that express a complex set of boolean rules that allow for the evaluation of many different attributes.

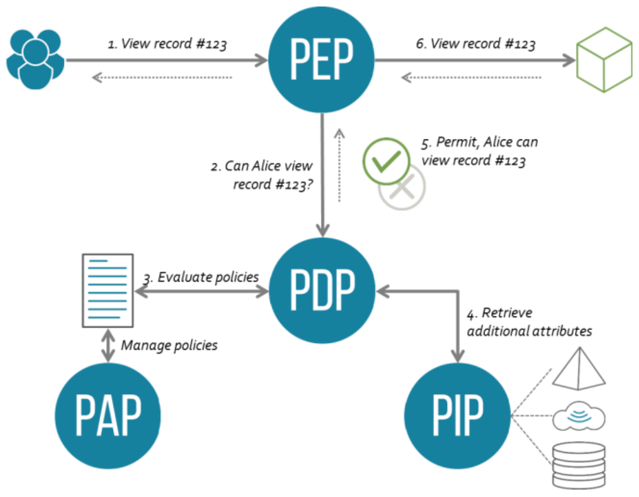
1. View record #123
PEP
6. View record #123
2. Can Alice view record #123?
5. Permit, Alice can view record #123
3. Evaluate policies
PDP
4. Retrieve additional attributes
Manage policies
PAP
PIP

Fig. 1 XACML (one of the key implementations of ABAC) architecture

Our approach aims to transform ABAC into a "next generation" authorization model, with dynamic, context-sensitive, and risk-intelligent access control to resources enabling access control policies that include specific attributes of different information systems to be defined to resolve an authorization and achieve effective regulatory compliance, allowing flexibility in companies to implement them according to their existing infrastructures.

PEP (Policy Enforcement Point) : is the component responsible for authorization. It intercepts the user's request for authorization then communicate with the PDP to obtain a decision and act upon it.

PDP (Policy Decision Point) : runs the decision evaluation of the request against decision policies

PAP (Policy Administration Point) : allows policy administration, management, and distribution

PIP (Policy Information Point) : The system entity that acts as a source of attribute values (i.e. a resource, subject, environment) [17].

B. Problematic:

To explain our approach and why it is needed, let us consider an ABAC framework that manages access to software projects within an enterprise where the permissions of users depend on their position and the projects to which they are assigned.

Let us assume a policy of access control that allows users assigned to the position of junior developer to read type ResA resources. Nevertheless, such resources can only be accessed by junior developers working on Project I and from Department I. We are using Python in our implementation, so the policy will look like this:

```
policy = dabac.Policy(
1,
actions=[Eq('read')],
resources=["ResA"],
subjects=[{'role': "Junior Developer",
'department': "Department I",
'project': "Project I"}],
effect=dabac.ALLOW_ACCESS,
description="""
Allow to junior devs from department I
working on project I to read ressources
of type ResA
"""
)
```

Intuitively, our sample policy's purpose is to specify a condition of access based on attributes that describes what action a particular subject may take on a resource.

Let's assume that Alice, a Department I junior developer, is trying to read a type ResA resource defined by the following access request:

```
inq = dabac.Inquiry(action='read',
resource="ResA",
subject={'role': "Junior Developer",
'department': "Department I",
'project': "Project I"})
```

It is easy to observe that the attributes in the request fit the rule requirement, giving rise to an ALLOW_ACCESS. Imagine now that Alice is trying to retrieve a large number of confidential project documents without a valid reason. As exemplified by the previous request, policy 1 will allow him to do so without taking into account how many documents she has retrieved. Nevertheless, this circumstance may suggest that the junior developer abuses his right of access for personal interests and benefits (e.g. selling documents to the competition).

Existing access control mechanisms are unable to avoid these insider attacks. The main problem is that access control is static in the sense that the implemented conditions for access do not change dynamically based on user behaviour. That's why our approach takes other contextual and situational factors into account. For

example: Could a user in a given time period perform multiple read queries? Can he access large quantities of data? Failure to respond to these questions that result in neglect of anomalous conduct that represents the misuse of privileges granted to insider threats.
In order to reduce the risks of users abusing their rights, we need to empower access control with proactive measures to change user behavior policies. In general, our aim is to dynamically optimize access control policies based on user activity that is controlled by narrowing privileges at runtime.
To achieve this goal, we need to equip access control systems with means of continuously monitoring user activities and flagging any suspicious and anomalous behaviour. It involves adding a component to the PIP (Policy Information Point) that provides an additional attribute to the user's query.
So for the previous scenario, the query is compatible with the policy, but the Machine Learning algorithm would flag the query as an anomaly and then the PDP would deny access to the user because of that.

C. Architecture:

Our frameworks tries to improve the classic ABAC architecture by adding a monitoring and outlier detection component into the PIP (Policy Information Point)

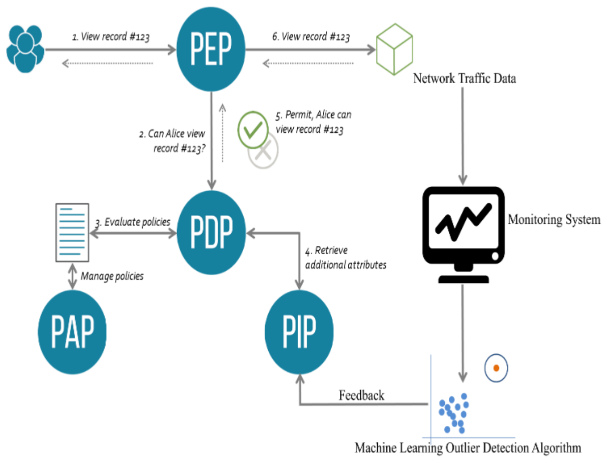
1. View record #123
PEP
6. View record #123
Network Traffic Data
2. Can Alice view record #123?
5. Permit, Alice can view record #123
3. Evaluate policies
PDP
4. Retrieve additional attributes
Monitoring System
Manage policies
PAP
PIP
Feedback
Machine Learning Outlier Detection Algorithm

Fig. 2 DABAC Architecture

The user Alice requests to read the resource #123 (step1), the PEP intercepts it and communicates it to the PDP (step2) in order for it to be processed. Then the PDP checks with both the PAP (step 3) to see if this request is compliant with any of the policies. And with the Monitoring System who is constantly feeding the Machine Learning algorithm with data. The PDP then receives feedback regarding the request (step 4). If the request is an anomaly or/and if no existing policies are matching the request then the PDP will issue a Deny for Alice and she won't be able to read the resource #123.

D. Outlier Detection:

Anomaly detection or Outlier detection is the identification of unexpected events or occurrences in a data set, that differ in some way or another from the norm. This can be an easy task for two dimensional data, as a simple visualization of the two variables is enough to detect an outlier, as shown in figure 3, when plotting X and Y we see a cluster of points with two relatively distant points, those points are outliers

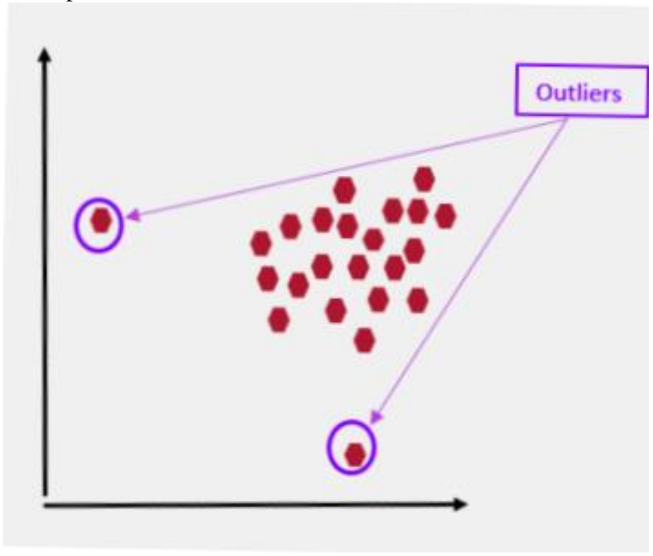

Fig. 3 Univariate Outlier Detection

Data visualization can often be a good starting point for identifying anomalies when dealing with one or two variables. However, this approach becomes increasingly difficult when it scales to high-dimensional data (which is our case here, as network traffic data have multiple variables). Luckily this is where Machine Learning algorithms come to help. We will introduce the most efficient algorithms used in anomaly detection.

1) K – Nearest Neighbours (KNN):

The KNN algorithm is one of the simplest machine learning algorithms[18, 19, 20, 21]. In a context of classification of a new observation x, the simple founding idea is to make the closest neighbours of this observation vote. The class of x is determined as a function of the majority class among the k nearest neighbors of the observation x. The KNN method is thus a neighborhood-based, non-parametric method; this means that the algorithm allows to make a classification without making a hypothesis on the function $y = f(x1, x2, \dots xp)$ which links the dependent variable to the independent variables.

K-NN needs a function to calculate the distance between two observations. The closer two points are to each other, the more similar they are and vice versa.
There are several distance calculation functions, including Euclidean distance, Manhattan distance, Minkowski distance, Jaccard distance, Hamming distance...etc. The distance function is chosen according to the types of data being manipulated. Thus for data of the same type, the Euclidean distance is a good candidate. As for the Manhattan distance, it is a good measure to use when the input variables are not of the same type as it is for our case here. The Manhattan distance: calculates the sum of the absolute values of the differences between the coordinates of two points:

$$D_m(x,y) = \sum_{i=1}^{k} | x_i - y_i |$$

2) Isolation Forest:

Isolation Forest [22] is an unsupervised machine learning algorithm calculates an anomaly score for each piece of data in the set, a measure of how atypical the data in question is. In order to calculate this score, the algorithm isolates the data in question recursively: it chooses a descriptor and a "cut-off point" at random, then evaluates whether this isolates the data in question; if so, the algorithm stops, otherwise it chooses another descriptor and another cut-off point at random, and so on until the data is isolated from the rest.
Recursive data partitioning can be represented as a decision tree [24], and the number of breaks needed to isolate a piece of data simply corresponds to the path taken in the tree from the root to the leaf, representing the isolated data. The path length defines the anomaly score: data with a very short path, i.e. data that is easy to isolate, is also likely to be anomalies, since it is very far from the other data in the set.
As with random forests [23], it is possible to do this independently by using several trees, in order to combine their results to improve performance. In this case, the anomaly score is the average of the path lengths on the different trees. This algorithm is particularly useful because it is very fast and does not require complicated parameterization.

3) Histogram-Based Outlier Score (HBOS) :

To Calculate the HBOS [25]. First, a univariate histogram is constructed for each single feature (dimension). If the feature consists of categorical data, a simple counting of the values of each category is performed and the relative frequency (height of the histogram) is calculated. Two different methods can be used for numerical features: (1) Static bin-width histograms or (2) dynamic bin-width histograms. The first method is the standard histogram building technique using k equal width bins over the range of values. The frequency (relative number) of samples falling into each bin is used to estimate the density (height of the bins). The dynamic bin-width is determined as follows: the values are sorted first and then the fixed number of successive $\frac{N}{k}$ values is grouped into a single bin where N is the total number of instances and k is the number of bins. Since the area of a bin in a histogram reflects the number of observations, it is the same for all bins. Because the width of the bin is determined by the first and last value and the area is the same for all bins, the height of each bin can be calculated.
The reason that both methods are offered in HBOS is due to the fact that the feature values have very different distributions in real world data. Particularly when value ranges have large gaps (intervals without data instances), the fixed bin width approach poorly estimates the density (a few bins may contain most data). Due to the fact that outliers are far away from normal data, anomaly detection tasks usually involve these gaps in the value ranges, we suggest using the dynamic width mode, particularly if the distributions are unknown or long tailed. Besides, you also need to set the number of bins k. A commonly used rule of thumb is to set k to the square root of the number of instances N.
Now a different histogram is created for each dimension d, where the height of each single bin represents a density approximation. Then the histograms are normalized in such a way that the maximum height is 1.0. It means that each feature has equal weight to the outlier score Ultimately, the HBOS is measured for each instance p using the corresponding height of the bins where the instance is located:

$$HBOS(p) = \sum_{i=0}^{d} \log\left(\frac{1}{hist_i(p)}\right)$$

4) Cluster Based Local Outlier Factor (CBLOF):

The anomaly detection algorithms based on the classical approaches use the full size of the dataset[26,27,28,29], thereby causing massive computational costs for large data sets like our case. We can reduce the computational cost by partitioning the large data set into meaningful clusters.
Clustering is therefore integrated into Local Outlier Factor (LOF)[27] to identify top n outliers in our data set. The approach is called the Cluster Based Local Outlier Factor (CBLOF) which will overcome the clustering and LOF drawbacks[30]. Consider a data set D containing the traffic of our Big Data System. The algorithm then partitions the dataset D into k disjoint sets $C = \{C_1, C_2, C_3, \dots, C_k\}$ with $C_i \cap C_j = \emptyset$ and $C_i \cup C_j = D$. A local outlier factor called CBLOF is allocated to each data instance It is determined based on the size of the cluster (Small Cluster (SC) or Large Cluster (LC)) and the distance between the target point and its neighboring cluster. Lets suppose $C_1 > C_2 >$

$\cdots > C_k$ then b is the boundary of a cluster (Large or Small) such as :

$$|C_1| + |C_2| + \cdots + |C_b| \geq |D| * \alpha \ \boldsymbol{or} \ \frac{|C_b|}{|C_{b+1}|} \geq \beta$$

Where α and β are the numeric values. Accordingly, $LC = \{C_i \mid i \leq b\}$ and $SC = \{C_j \mid j > b\}$.

So for every point p of the data set the CBLOF is defined by the follwing equation :

$$CBLOF(p) = \begin{cases} |C_i| * \min\left(distance(p, C_j)\right) \\ if\ p \in C_i, C_i \in SC, C_j \in LC\ for\ j = 1\ to\ b \\ |C_i| *\ distance(p, C_i) \\ if\ p\ \in C_i, C_i \in LC \end{cases}$$

The algorithm gives a distance-based anomaly score to the nearest large cluster, multiplied by the cluster size to which the entity belongs. The definition is shown in Figure 4. Point p resides in the small cluster C_2 and therefore the score would be equal to the distance to C_1 which is the closest large cluster multiplied by 5 which is the size of C_2.

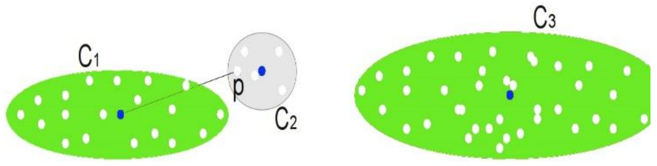

Fig. 4 for the point p, the distance to the cluster C1 is used to compute the CBLOF Score. In this case C1 and C3 are identified as large clusters and C2 is considered a small Cluster, the blue points represent the center of Clusters

5. Model evaluation and implementation:

To evaluate our model, we use Apache Spark [33] along with MLlib [32], we will be using the UNSW-NB15 [31] Dataset. This section presents details about the dataset and also a comparison between all the outlier detection methods that we talked about in the previous section. We will also present the final implementation of the framework.

A. Apache Spark:

Apache Spark is a fast data processing engine dedicated to Big Data. It allows the processing of large volumes of data in a distributed manner (cluster computing). Very popular for a few years now, this framework is about to replace Hadoop. Its main advantages are its speed, ease of use, and versatility.

It is an open source parallel data processing engine that allows for large-scale analysis through clustered machines. Coded in Scala, Spark can handle data from data repositories such as Hadoop Distributed File System, NoSQL databases, or relational data stores such as Apache Hive. This engine also supports In-memory processing, which increases the performance of Big Data analytical applications. It can also be used for conventional disk-based processing, if the datasets are too large for system memory.

Its main advantage is its speed, since it can launch programs 100 times faster than Hadoop MapReduce in-memory, and 10 times faster on disk. Its advanced DAG (Directed Acyclic Graph) execution engine supports acyclic data flow and in-memory computing. It is also easy to use, and allows you to develop applications in Java, Scala, Python and R. Its programming model is simpler than Hadoop's. Thanks to more than 80 high-level operators, the software makes it easy to develop parallel applications.

Another advantage of Apache Spark is its generality. It acts at the same time as SQL query engine, data processing software (Spark Streaming), and graph processing system (GraphX). Apache Spark also includes a large number of libraries of MLlib algorithms for Machine Learning. These libraries can be easily combined within the same application.

The engine can run on Hadoop 2 clusters based on the YARN resource manager, or on Mesos. It is also possible to run it standalone or in the cloud with Amazon's Elastic Compute Cloud service. It provides access to various data sources such as HDFS, Cassandra, Hbase and S3.

The other strong point of this engine is its massive community. Apache Spark is used by a large number of companies for processing large datasets. This community can be reached through a list of email addresses, or through events and summits. As an open-source platform, Apache Spark is developed by a large number of developers from over 200 companies. Since 2009, more than 1000 developers have contributed to the project. This makes many Spark tutorials available.

In our case Spark high speed data processing will allow our framework to process the traffic data collected and detect outliers using MLlib and PyOD [34]

B. Dataset Description:

To test our model, we opted to use the UNSW-NB15 Dataset. The raw network packets of the UNSW-NB 15 dataset were generated by the IXIA PerfectStorm tool in the Cyber Range Lab of the Australian Centre for Cyber Security (ACCS) it produces a hybrid of real modern normal network activities and synthetic contemporary attack behaviours. This dataset has nine types of attacks, namely: Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode and Worms. The total

number of records is two million and 540,044 and there are 49 features in total including the class label.
We will only use a partition of the dataset. We will configure it as two sets: a training set containing 150,000 records and a test set containing 75,000 records, making also sure that both sets have the same distribution.
UNSW-NB15 is a dataset used to train IDS, but we are using it to train our Access Control framework. We are only interested in the traffic aspect of the dataset, so before training our model we will first remove the feature: $attack\_cat$. Our framework uses Unsupervised learning algorithms and the anomalies we aim to detect are unusual behavior that may or may not be included in the 9 types of attacks listed in the dataset.

C. Results:

In this section we evaluate the performance of the four outlier detection algorithms: K-NN, Isolation Forest, HBOS and CBLOF using the UNSW-NB15 training and test sets created in the previous section. We will be using PyOD as it is a library with all the main anomaly detection algorithms that we need. The machine used for this experiment is a p3.2xlarge EC2 instance [35].

Table 1: Performance of different outlier detection algorithms on unsw-nb15 data set

Algorithm	Training AUC	Testing AUC	Training Time	Prediction Time
K - NN	94.2%	95.9%	16.82	0.23
Isolation Forest	97.3%	97.3%	5.19	0.14
HBOS	89.6%	88.4%	1.04	0.03
CBLOF	90.2%	91.3%	1.22	0.09

We used GridSearchCV [36] cross-validation to find the optimal hyper-parameters for all four algorithms. As for the metrics used to determine performance, we used AUC (Area Under the ROC Curve) [37] it provides an aggregate measure of performance for all possible classification thresholds. The AUC can be interpreted as a measure of the probability that the model will rank a random positive example above a random negative example. A model with 100% error in predictions has an AUC of 0.0. If all its predictions are correct, its AUC is 1.0. The AUC is scale invariant. It measures how well predictions are ranked, rather than their absolute values. In addition to that it is independent of classification thresholds. It measures the quality of model accuracy regardless of the classification threshold selected. We also took in consideration the training and prediction time as they are very crucial in our case, the framework must be as accurate and as fast as possible in order to be responsive towards any anomaly as soon as it occurs.

Table 1 shows the evident superiority of Isolation Forests as it achieves 97.3% AUC on both training and test sets, that means that the algorithm generalizes well, the training and prediction time are also good. K-NN have a decent AUC too, but it takes long to train the model. HBOS and CLBOF didn't do well on the dataset although they are very fast on training and predicting. The low AUC on both algorithms can be due to the high dimensionality of the dataset, both algorithms are distance based and the data gets more sparse the higher its dimensions are as shown in [38].

D. Implementation:

To implement our framework, we used the Python language, the ABAC component was based on VAKT [39], an attribute based access control toolkit coded in Python, we modified it and added our Machine Learning component then tuned it's decision process to include the feedback from the said component. We named our framework DABAC (Dynamic Attribute Based Access Control), the code can be found at : https://github.com/HamzaES/DABAC.

6. Conclusion and Future work:

In this paper, we presented an access control framework for Big Data systems: DABAC (Dynamic Attribute Based Access Control), it is based on ABAC and adds a Machine Learning
Anomaly detection component that continuously processes traffic data and flag any suspicious user behavior. We used the UNSW-NB15 data set, Apache Spark and a p3.2xlarge Amazon EC2 instance to create a Big Data environment and evaluated the performance of different outlier detection algorithms in order to determine the most suitable one that satisfies our performance metrics (AUC, training time and prediction time). Isolation Forest was clearly the best performing algorithm, thus we used it in our Python implementation. It is also worthy to note that our Framework respects the three requirements for a Big Data Access Control (Chapter II), Our Solution is fine grained since it's based on ABAC which is itself a fine grained access control. The solution is also context aware, given the fact that it constantly monitors the network traffic and flags any suspicious activity based on its context, if the user's behavior is out of the usual context it will be deemed anomalous. And finally, the framework is efficient, the outlier detection takes around 143 ms to decide whether the user's activity is anomalous or not, and sends the feedback to the ABAC component which will take it into consideration to issue a $PERMIT$ or $DENY$
In future work, we will aim to push the integration of the Machine Learning module into ABAC further, we ideally

want to be able to change the policies themselves using Machine Learning, this will give birth to a more dynamic and effective access control for Big Data.

References

[1] Cattell, Rick. "Scalable SQL and NoSQL data stores." Acm Sigmod Record 39.4 (2011): 12-27.

[2] Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." Communications of the ACM 51.1 (2008): 107-113.

[3] Colombo, Pietro, and Elena Ferrari. "Privacy aware access control for big data: A research roadmap." Big Data Research 2.4 (2015): 145-154.

[4] Agrawal, Rakesh, et al. "Hippocratic databases." VLDB'02: Proceedings of the 28th International Conference on Very Large Databases. Morgan Kaufmann, 2002.

[5] Byun, Ji-Won, and Ninghui Li. "Purpose based access control for privacy protection in relational database systems." The VLDB Journal 17.4 (2008): 603-619.

[6] Carminati, Barbara, et al. "A framework to enforce access control over data streams." ACM Transactions on Information and System Security (TISSEC) 13.3 (2010): 1-31.

[7] Narayanan, Arvind, and Vitaly Shmatikov. "Robust de-anonymization of large sparse datasets." 2008 IEEE Symposium on Security and Privacy (sp 2008). IEEE, 2008.

[8] Maryam Archie, Sophie Gershon, Abigail Katcoff, Aileen Zeng. "Who ' s Watching ? De-anonymization of Netflix Reviews using Amazon Reviews."

[9] "Netflix Dataset". Accessed on: Feb. 3, 2020. [Online]. Available: https://web.archive.org/web/20090925184737/http://archive.ics.uci.edu/ml/datasets/Netflix+Prize

[10] Rizvi, Shariq, et al. "Extending query rewriting techniques for fine-grained access control." Proceedings of the 2004 ACM SIGMOD international conference on Management of data. 2004.

[11] Florescu, Daniela, and Ghislain Fourny. "JSONiq: The history of a query language." IEEE internet computing 17.5 (2013): 86-90.

[12] Ong, Kian Win, Yannis Papakonstantinou, and Romain Vernoux. "The SQL++ unifying semi-structured query language, and an expressiveness benchmark of SQL-on-Hadoop, NoSQL and NewSQL databases." CoRR, abs/1405.3631 (2014).

[13] Es-Samaali, Hamza, Aissam Outchakoucht, and Jean Philippe Leroy. "A blockchain-based access control for big data." International Journal of Computer Networks and Communications Security 5.7 (2017): 137.

[14] Maloof, Marcus A., and Gregory D. Stephens. "Elicit: A system for detecting insiders who violate need-to-know." International Workshop on Recent Advances in Intrusion Detection. Springer, Berlin, Heidelberg, 2007.

[15] Hummer, Matthias, et al. "Adaptive identity and access management—contextual data based policies." EURASIP Journal on Information Security 2016.1 (2016): 19.

[16] Ferraiolo, D.F. & Kuhn, D.R. "Role-Based Access Control" (PDF). 15th National Computer Security Conference: 554–563.

[17] Hu, Vincent C., et al. "Guide to attribute based access control (abac) definition and considerations (draft)." NIST special publication 800.162 (2013).

[18] Fix, Evelyn. Discriminatory analysis: nonparametric discrimination, consistency properties. Report Number 4 USAF school of Aviation Medicine, 1951.

[19] Fix, Evelyn. Discriminatory analysis: nonparametric discrimination, consistency properties. Report Number 11 USAF school of Aviation Medicine, 1952.

[20] Cover, Thomas, and Peter Hart. "Nearest neighbor pattern classification." IEEE transactions on information theory 13.1 (1967): 21-27.

[21] Silverman, Bernard W., and M. Christopher Jones. "E. fix and jl hodges (1951): An important contribution to nonparametric discriminant analysis and density estimation: Commentary on fix and hodges (1951)." International Statistical Review/Revue Internationale de Statistique (1989): 233-238.

[22] Liu, Fei Tony, Kai Ming Ting, and Zhi-Hua Zhou. "Isolation forest." 2008 Eighth IEEE International Conference on Data Mining. IEEE, 2008.

[23] Liaw, Andy, and Matthew Wiener. "Classification and regression by randomForest." R news 2.3 (2002): 18-22.

[24] Breiman, Leo. Classification and regression trees. Routledge, 2017.

[25] Goldstein, Markus, and Andreas Dengel. "Histogram-based outlier score (hbos): A fast unsupervised anomaly detection algorithm." KI-2012: Poster and Demo Track (2012): 59-63.

[26] Knox, Edwin M., and Raymond T. Ng. "Algorithms for mining distancebased outliers in large datasets." Proceedings of the international conference on very large data bases. Citeseer, 1998.

[27] Breunig, Markus M., et al. "LOF: identifying density-based local outliers." Proceedings of the 2000 ACM SIGMOD international conference on Management of data. 2000.

[28] Ramaswamy, Sridhar, Rajeev Rastogi, and Kyuseok Shim. "Efficient algorithms for mining outliers from large data sets." Proceedings of the 2000 ACM SIGMOD international conference on Management of data. 2000.

[29] Aggarwal, Charu C., and Philip S. Yu. "Outlier detection for high dimensional data." Proceedings of the 2001 ACM SIGMOD international conference on Management of data. 2001.

[30] He, Zengyou, Xiaofei Xu, and Shengchun Deng. "Discovering cluster-based local outliers." Pattern Recognition Letters 24.9-10 (2003): 1641-1650.

[31] Moustafa, Nour, and Jill Slay. "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)." 2015 military communications and information systems conference (MilCIS). IEEE, 2015.

[32] Meng, Xiangrui, et al. "Mllib: Machine learning in apache spark." The Journal of Machine Learning Research 17.1 (2016): 1235-1241.

[33] Zaharia, Matei, et al. "Apache spark: a unified engine for big data processing." Communications of the ACM 59.11 (2016): 56-65.

[34] Zhao, Yue, Zain Nasrullah, and Zheng Li. "Pyod: A python toolbox for scalable outlier detection." arXiv preprint arXiv:1901.01588 (2019).

[35] "Amazon EC2 P3 Instances". Accessed on: Feb. 7, 2020. [Online]. Available: https://aws.amazon.com/ec2/instance-types/p3/?nc1=h_ls
[36] "GridSearchCV". Accessed on: Jan. 27, 2020. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
[37] Hanley, James A., and Barbara J. McNeil. "The meaning and use of the area under a receiver operating characteristic (ROC) curve." Radiology 143.1 (1982): 29-36.
[38] Kouiroukidis, Nikolaos, and Georgios Evangelidis. "The effects of dimensionality curse in high dimensional knn search." 2011 15th Panhellenic Conference on Informatics. IEEE, 2011.
[39] "Attribute-based access control (ABAC) SDK for Python". Accessed on: Jan. 20, 2020. [Online]. Available: https://github.com/kolotaev/vakt
[40] Hu, Vincent C., et al. "Guide to attribute based access control (abac) definition and considerations (draft)." NIST special publication 800.162 (2013).