Detecting abnormal DNS traffic using unsupervised machine learning

Thi Quynh Nguyen
University Paul Sabatier
Toulouse, France
MODIS
Courbevoie, France
thi-quynh.nguyen@irit.fr

Romain Laborde
University Paul Sabatier
Toulouse, France
romain.laborde@irit.fr

Abdelmalek Benzekri
University Paul Sabatier
Toulouse, France
abdelmalek.benzekri@irit.fr

Bruno Qu'hen
MODIS
Courbevoie, France
bruno.quhen@modis.com

***Abstract*— Nowadays, complex attacks like Advanced Persistent Threats (APTs) often use tunneling techniques to avoid being detected by security systems like Intrusion Detection System (IDS), Security Event Information Management (SIEMs) or firewalls. Companies try to identify these APTs by defining rules on their intrusion detection system, but it is a hard task that requires a lot of time and effort. In this study, we compare the performance of four unsupervised machine-learning algorithms: K-means, Gaussian Mixture Model (GMM), Density-Based Spatial Clustering of Applications with Noise (DBSCAN), and Local Outlier Factor (LOF) on the Boss of the SOC Dataset Version 1 (Botsv1) dataset of the Splunk project to detect malicious DNS traffics. Then we propose an approach that combines DBSCAN and K Nearest Neighbor (KNN) to achieve 100% detection rate and between 1.6% and 2.3% false-positive rate. A simple post-analysis consisting in ranking the IP addresses according to the number of requests or volume of bytes sent determines the infected machines.**

Keywords—Anomaly detection, DNS tunneling, C&C, K-means, GMM, LOF, DBSCAN

I. INTRODUCTION

Advanced Persistent Threats (APTs) are sophisticated and premeditated attacks perpetrated by a threat group of highly skilled attackers combining multiple competencies to achieve specific and predetermined fraudulent goals. Rather than opportunistically attacking a system, the intruders gain access to a network and remain undetected for an extended period. The complexity of APTs attacks makes them difficult to identify [1]. Since they target critical companies and governments, they constitute one of the biggest security challenges [2].

To tackle such a challenge, MITRE has proposed ATT&CK, an open source knowledge base of adversary Tactics, Techniques and Procedures (TTPs) based on real-world observations. Tactics are the goals an attacker aims to reach. MITRE ATT&CK lists 12 tactics: Initial access, Execution, Persistence, Privilege escalation, Defense evasion, Credential access, Discovery, Lateral movement, Collection, Command and Control (C&C), Exfiltration, Impact. To achieve these objectives, attackers use one or more techniques. For instance, APT32 [3] has targeted multiple private sector industries in Southeast Asian countries like Vietnam, Laos, and Cambodia. It employed 55 different threat techniques to attack their target.

Currently, companies are trying to improve their capability of detecting APTs by translating MITRE ATT&CK TTPs into detection rules on their Intrusion Detection Systems (IDS). However, writing such rules is a very hard task that requires building many highly complex indicators of compromise. When studying APT32, the attackers used Cobalt Strike's malleable C&C functionality to hide malicious traffic inside authorized network traffic e.g. DNS, HTTP and HTTPS. The group's backdoor could exfiltrate data by encoding it within the subdomain field of DNS packets.

In this article, we focus only on detecting the C&C tactic and more specifically on threat techniques consisting in hiding malicious traffic inside the DNS protocol. These malicious traffics are very similar to genuine DNS traffic. They use the same UDP port 53 and respect the DNS messages structure. Thus, detecting such traffic cannot be limited to analyzing port numbers or inspecting the structure of the messages. A deeper analysis is required to assess the behavior of the entities by combining multiple information such as the number of bytes exchanged, the duration, the time the message was sent, etc. Nowadays many companies are using Security Event Information Management (SIEMs) to manage event logs and protect their network from attacks. However, writing detection rules in SIEMs while considering a large number of criteria is no more possible. What is the correct threshold value for each criterion? How to combine constraints on criteria? APTs are being executed by skilled attackers, where the attack rate is controlled in order to go under the radar and remain undetected.

Machine learning techniques can help in such a complex detection task. There exist two families of machine learning techniques: supervised and unsupervised methods. Supervised methods can learn patterns from training datasets that comprise the input features and the expected results called labels. Supervised algorithms are good for specific tasks such as image detection. However, this approach has two drawbacks in our context. First, the performance of the algorithm depends on the exhaustivity of the training dataset. An algorithm trained with a specific set of threat techniques might not detect zero-knowledge attacks. An algorithm trained to detect genuine traffic will be specific to the organization where the training dataset has been captured. The second issue is related to the cost of labelling datasets. A large number of image datasets exists because labelling images does not require specific skills. In case of network security, the context is completely different. Labelling network security traffic can only be performed by security experts which implies exponential increase of the cost of this task. Unsupervised methods don't need any training which resolves the two previously mentioned drawbacks. However, it is more difficult to have the same result as the supervised methods can obtain.

978-1-6654-0458-7/20/$31.00 ©2020 IEEE

We propose in this article a behavioral analysis approach based on unsupervised machine learning. We compare the performance of four unsupervised machine-learning algorithms: K-means, GMM, DBSCAN, and LOF on the Botsv1 dataset of the Splunk project [4] to detect malicious DNS traffics. This study exhibits the benefits of DBSCAN. Then we propose a new approach that combines DBSCAN and KNN that achieves 100% for detection rate and 2.3% for false-positive rate. A simple post-processing consisting in ranking the IP addresses according to the number of requests and the volume of bytes sent helps us easily identify the infected machines.

The rest of the article is as follows. First, we present the related work. Then, we introduce K-means, GMM, DBSCAN, and LOF in section III. Section IV describes our detection approach, the dataset, and the comparison of the four unsupervised algorithms. We also show how the parameters of DBSCAN can be tuned automatically with KNN. Finally, Section V concludes the article and indicates prospects for future works.

II. RELATED WORK

We summarize in this section the different related work. First, we present unsupervised machine learning approaches for detecting generic network intrusions. Then we focus on all machine learning approaches to detect abnormal DNS traffic.

A. Unsupervised approaches for the detection of attacks

Leon et al. [5] presented a new approach to anomaly detection based on the Unsupervised Niche Clustering (UNC) and apply it to network intrusion detection. The UNC is a genetic niching technique for clustering, which can determine the number of clusters automatically. The authors characterize each predicted cluster using a fuzzy membership function. Moreover, they use the Maximal Density Estimator (MDE) refinement to improve the quality of the solution and Principal Component Analysis (PCA) to reduce the complexity of the dataset and further enhance the performance of the proposed approach. The model has been tested on the KDD Cup 1999 dataset [6] and shows a detection rate of 99.2% and a false alarm rate of 2.2%.

Casas et al. [7] presented UNIDS, an Unsupervised Network Intrusion Detection System capable of detecting unknown network attacks without using any kind of signatures, labeled traffic, or training. UNIDS contains three consecutive steps. First, it uses a standard change-detection algorithm on three simple and classic volume metrics (bytes, packets, and flows per time slot) to detect an anomalous time slot in which the clustering analysis will be performed. Secondly, it uses a robust multi-clustering algorithm, based on a combination of Sub-Space Clustering (SSC), Density-based Clustering, and Evidence Accumulation Clustering (EAC) techniques to identify outlying flows. Finally, a predefined threshold allows determining the anomalies. They evaluate UNIDS on KDD Cup 1999 dataset where the detection rate is more than 90% and the false positive rate is less than 1%.

Amoli et al. [8] proposed a real-time unsupervised NIDS for high-speed networks. This NIDS contains two separate engines which allow network intrusion detection. The first engine monitors the behavior of the network to detect intrusions in real time using DBSCAN with an automated self-adaptive threshold. The second engine is to detect the internal botnet. It clusters specific network features to detect centralized and decentralized botnet C&C communication. Two datasets were used to evaluate the proposed approach: DARPA and ISCX [9]. The proposed model achieved a 98.39% accuracy and 3.61% false positive rate.

Aliakbarisani et al. [10] propose a linear metric learning method for unsupervised Network Anomaly Detection Systems (NADSs), which uses distance-based clustering or classification methods. The unsupervised clustering based NADS contains the training and testing phases. The training phase has five steps: Filtering, Metric Learning, Transformation, Clustering and Boundary Estimation. They use one-class SVM in the Boundary Estimation. And the testing phase consists of three steps: Transformation, Dividing and Classifying. They evaluate their NADS on the Kyoto 2006+ [11] and NSL-KDD [12] datasets. With the metric learning, the algorithms can improve the performance.

Kazato et al. [13] described a method for estimating the maliciousness of indicators of compromise (IoCs). This method contains three main steps: gather relational information and extract individual IoC features; build an IoC graph using existing IoC nodes in the graphical pre-built IoC; estimate GCN-based IoC maliciousness. They use two characteristics in the graph convolutional network (GCN) to improve the precision of the estimation, which are the individual IoC functionalities and the relationships between the IoCs. They create a dataset of 20,000 domain names (resolved by DNS published on the Internet), 10,000 benign domain names (listed on Alexa Top Sites), 10,000 malicious domain names (resolvable from Abuse.ch ZeuS Tracker and DNS - BH Malware Domain Blocklist) to evaluate their approach, and the true positive rate is 88%.

Gunes Kayacik et al. [14] presented an approach of Self-Organizing Feature Maps (SOM) hierarchy with multiple layers. They build the hierarchical SOM architectures with two learning algorithms: SOM is used at each layer of the hierarchy and Potential Function Clustering is used to quantize the number of SOM neurons 'perceived' by the second layer for the 6-feature architecture alone. To evaluate this method, they apply this two-layers SOM hierarchy to the KDD CUP 1999 database. With 41 functionalities, the second and third layers are sufficient: the first layer based on six basic functions; the second layer is based on nine basic functions and 32 derived functionalities. The proposed method achieved a 90% true positive rate and a 1.38% false positive rate.

Zanero and Serazzi [15] proposed a two-tier architecture to analyze network packets for network intrusion detection. The first tier of the system classifies packet payload using an unsupervised clustering algorithm (K-means, Principal Direction Divisive Partitioning (PDDP), and SOM). The second tier detects the anomalies in each single packet and in a sequence of packets with the SDLE algorithm of SmartSifter. Moreover, they propose improvements and heuristics to increase the throughput of SOM to a rate suitable for online intrusion detection. They use the DARPA 1999 [16] database to evaluate, the true positive rate is only 66.7%.

Chen et al. [17] proposed the federated learning assisted deep autoencoding Gaussian mixture model (FDAGMM) for network anomaly detection. It is an improvement of the DAGMM (Deep Autoencoding Gaussian Mixture Model) by using Federated Learning in order to handle scenarios where training data is insufficient because normal users are unwilling or unable to share private information. FDAGMM is a

combination of three algorithms: Auto-encoding to reduce the dimension, Gaussian Mixture Model to estimate density, and Federated Learning to improve its performance with more data and also protecting privacy. They evaluate this method on the KDD CUP 1999 database, and the true positive rate is 98.03%.

Alam et al. [18] presented a network intrusion and real-time anomaly detection system that utilizes a memristor based autoencoder. Autoencoders is an unsupervised algorithm and can recognize anomalous data quickly and accurately. Memristor is a new class of nanoscale semiconductor devices and can perform in parallel many multiply-add operations in the analog domain. Thus, using memristor devices to perform autoencoders computations allows presenting extreme low power systems for real-time intrusion and anomaly detection. They use the NSL-KDD database for evaluation, and the overall detection accuracy of this system is 92.91%.

Authors employed most of unsupervised techniques and achieve good results. However, they assess their work against datasets that contain attacks that are not representative to current APTs attacks.

B. DNS based C&C communication detection

Fewer researches deal with abnormal DNS detection. And most of them proposed to use supervised machine learning techniques.

Homem and Papapetrou [19] introduced a machine learning approach, based on three features IP packet length, DNS Query Name Entropy, and DNS Query Name Length, to identify protocols (HTTP, HTTPS, FTP, POP3) transported in DNS tunneling. They benchmark the performance of four classification models (decision trees, support vector machines, k-nearest neighbors, and neural networks) on a DNS tunneled traffic dataset. Neural networks give the best result with a 95% accuracy.

Lin et al. [20] proposed an application-layer tunnel detection method by applying a rule-based domain name filtering for Domain Generation Algorithm (DGA) based on a trigram model and machine learning. They tested the effectiveness of the proposed method by conducting experiments on DNS, HTTP, HTTPS tunnels. In each tunnel type, they applied and compared six classification algorithms (Gaussian Bayes, SVM, C4.5, Random Forest, GBDT, XGboost). Except Gauss Bayes, the classification results of the five machine learning algorithms were good (Precision > 0.97, Accuracy > 0.98).

Berg et al. [21] compared the different models (Multinomial neural network and Random forest) on the Request, Response and Full feature set. Experiments showed the performance of the models increases when training the models on the query and response pairs rather than using only queries or responses. The accuracy of the models is more than 83%.

Although abnormal DNS detection works achieve good detection results, they use supervised learning approaches and thus require a training dataset. In addition, a common critic that applies to these works is the use of non-public datasets. Therefore, it is difficult to replicate the experiments. That's why we propose in this article a behavioral analysis approach based on clustering for detecting malicious DNS traffics. We evaluate our approach on a public dataset: Botsv1 dataset of the Splunk project.

III. UNSUPERVISED OUTLIERS DETECTION

As explained in the introduction, applying supervised learning algorithms for detecting malicious network traffic raises issues due to the exhaustivity of the training dataset and high labelling cost. There exist many unsupervised anomaly detection algorithms such as Density-based spatial clustering of applications with noise (DBSCAN), Isolation Forests, Local Outlier Factor (LOF), etc.[22] We present in this section four techniques: K-means, Gaussian Mixture Model (GMM), LOF and DBSCAN that will be evaluated later in this article.

A. K-means

K-means [23] is a distance-based algorithm. It tries to group the closest points to form a cluster. K-means algorithm is composed of 3 steps:

1. *Initialization*: choose randomly K data points which are K-centroids of K groups of data.
2. *Cluster Assignment*: classify the data points to the clusters where the distance between the point and the centroid is minimum (in Euclidean distance).
3. *Move the centroid*: recalculate the centroids to be the mean of all the points in the clusters. Repeat step 2 and 3 until the centroids stop moving.

K-means can run fast and efficiently on any dataset. However, the accuracy of its output depends a lot on the number of clusters (K). Moreover, K-means only uses circle forms for graphs, thus it lacks the flexibility to change the shape of the clusters.

B. Gaussian Mixture Models

Gaussian mixture models (GMM) [24] is a distribution-based algorithm. Similar to K-means, we have to choose the number of clusters. GMM assumes that there are M Gaussian component densities and it tries to group the data points belonging to the same component together.

GMM is a parametric probability density function represented as a weighted sum of M component Gaussian densities given by this equation:

$$p(x|\lambda) = \sum_{i=1}^{M} w_i \, g(x|\mu_i, \Sigma_i)$$

Where x is D-dimensional continuous-valued features, w_i are the mixture weights that satisfy the constraint $\sum_{i=1}^{M} w_i = 1$, and $g(x|\mu_i, \Sigma_i)$ are the component Gaussian densities.

Each component density is a D-variate Gaussian function of the form:

$$g(x|\mu_i, \Sigma_i) = \frac{1}{(2\pi)^{D/2}|\Sigma_i|^{1/2}} e^{-\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i)}$$

with mean vector μ_i, and covariance matrix Σ_i.

All the parameters that made up GMM, including the mean vectors, the covariance matrices, and the mixture weights from M components densities are represented by this notation:

$$\lambda = \{\mu_i, \Sigma_i, w_i\} \quad i = 1, \dots, M$$

There are several approaches for estimating the parameters of GMM and the most popular one is Expectation-Maximization (EM) algorithm [25], which iteratively optimizes the model using maximum likelihood estimates. Expectation-Maximization is composed of 4 steps:

1. *Initialization*: Values can be randomly assigned to the means, variances, and mixture weights. Moreover, another approach consists in using some form of binary Vector quantization estimation or the results obtained by a previous K-Means run to define these values.
2. *E-step*: for each point, calculate the probability that it belongs to each cluster/component. This probability will be higher when the point is assigned to the right cluster and lower otherwise.
3. *M-step*: update the Gaussian parameters (λ) for each component to fits points assigned to them.
4. *Evaluate the convergence*: check for convergence of the parameters. If some convergence threshold is reached, stop the algorithm. If not, return to *E-step* (step 2). The new model then becomes the initial model for the next iteration.

While the clusters created by K-means have a circular shape, GMM can handle oblong clusters. K-means is a hard classification whereas GMM is a soft one. GMM provides the probabilities that a given data point belongs to each of the possible clusters.

C. Density-Based Spatial Clustering of Applications with Noise

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [26] clusters points together and identifies any points not belonging to a cluster as outliers. DBSCAN has two parameters that need to be defined:

- *min_samples*: the minimum number of points required to form a dense region (a cluster).
- *eps* (ε): a distance measure that will be used to locate the points in the neighborhood of any point.

DBSCAN labels the data points as core points, border points, or outlier points. A given point p is a core point if it has at least *min_samples* points in its *eps* distance. A point q is a border point if it is not a core point, and lies within the *eps* distance of any core point. Outlier points are those that are neither core points nor border points.

DBSCAN is composed of 3 steps:

1. DBSCAN takes an arbitrary point that has not been visited (until all points have been visited).
2. The *eps* neighborhood of this point is extracted (all points within the *eps* distance). If it contains enough points (*min_samples*), then this point is a core point and a cluster is created. Otherwise, this point is labeled as noise (anomaly). However, this point might also be found in an *eps* neighborhood of another point, and therefore, in that case, it will be a border point and a part of that cluster.
3. If a point is a part of a cluster, its *eps* neighborhood is also a part of that cluster. This process continues until all points are visited.

If the value of *min_samples* is too large, DBSCAN can detect anomalies but with a high false positive rate. Otherwise, if the value is too small, DBSCAN cannot detect all anomalies. Similarly, if the value of *eps* is too large, DBSCAN cannot detect all the anomalies. While if it is too small, DBSCAN can detect anomalies but the false positive rate is high. Therefore, selecting good values for these parameters is important.

D. Local Outlier Factor

Local Outlier Factor (LOF) [27] is a density-based algorithm to identify local outliers. A point is an outlier if the density around that point has a big difference with the density around its neighbors. LOF is more concerned with a local data distribution than a global data distribution.

LOF depends on 1 single input parameter: *MinPts*, which is the number of nearest neighbors used in defining the local neighborhood. It consists of 7 steps:

1. Choose *MinPts* = K as the input parameter.
2. For each data point p in the data set D, calculate distances between p and all other points. Let's define $d(p,q)$ as the distance between 2 points p and q. We can choose any metric for the distance computation such as Euclidean or Manhattan.
3. Find the K^{th}-nearest neighbor of p. Then calculate the distance from p to that point and call it K-distance(p). K-distance(p) provides a measure of the density around point p. When K-distance(p) is small, the area around p is dense and vice versa.
4. Then find the K-distance neighborhood of p, which contains every point whose distance from p is not greater than K-distance(p), i.e. $N_K(p) = \{q \in D \mid d(p,q) \leq K - distance(p)\}$. Note that $|N_K(p)|$ or the number of points in $N_K(p)$ can be greater than K because there might multiple points with the same distance to p.
5. Calculate the reachability distance of p with respect to each of the other points using this formula:

   $$reach\_dist\ K(p,o) = \max\{K - distance(o), d(p,o)\}$$

   Simply put, if p is far away from o then the reachability distance between them is their actual distance $d(p,o)$. Otherwise, if they are sufficiently close, the actual distance is replaced by the K-distance of o.
6. Calculate the Local Reachability Density (LRD) of p, which is an estimation of the density at point p, defined as the inverse of the average reachability distance of the K nearest neighbors of p:

   $$LRD_K(p) = 1/\left(\frac{\sum_{o \in N_K(p)} reach\_dist\ K(p,o)}{|N_K(p)|}\right)$$

   A low value of $LRD_K(p)$ indicates that p is far from its closest cluster.
7. Calculate Local Outlier Factor (LOF) of p, which is a ratio that determines whether or not a point is an outlier with respect to its neighborhood. Basically, LOF(p) is the average of the ratios of the local reachability density of p and those of p's K-nearest neighbors:

   $$LOF_K(p) = \frac{\sum_{o \in N_K(p)} \frac{LRD_K(o)}{LRD_K(p)}}{|N_K(p)|}$$

A high value of $LOF_K(p)$ implies that the density around p is much different from the density around its K nearest neighbors. Thus, indicating that p is potentially an outlier.

LOF works well in outlier detection but the number of neighbors K must be manually chosen. Increasing K or reducing it too much could lead to misclassification as outliers.

IV. An Unsupervised Machine Learning-Based Approach to Detect Abnormal DNS Traffic

In this section, we present our approach for detecting abnormal DNS traffic. This process has been created considering two hypotheses. First, APTs are sophisticated attacks perpetrated by a threat group of highly skilled attackers whose objective is to remain undetected for an extended period. These attackers are trying to control their malicious traffic to go under the radar. Thus, most of the network traffic is genuine. The traffic corresponding to such attacks is very low (e.g., less than 2 or 3% of the traffic). Indeed, if the malicious traffic is higher, we consider that traditional detection systems such as SIEMs can detect it. The second hypothesis is related to detecting anomality. Without previous knowledge, detecting outliers requires comparing similar entities only. Consequently, the network traffic must be preprocessed to separate the different network services (DNS, HTTP(S), SMTP(S), etc.) before applying outlier detection. Thus, our detection process is the following (Figure 1):

1. Separate the network traffic by service.
2. For each service, apply unsupervised machine learning to detect outliers. We will evaluate four algorithms K-means, GMM, DBSCAN, and LOF. In our case, we will restrict the analysis to only DNS traffic.
3. Post-analyze the outliers to determine the infected machines

The experimentation and the evaluation of the approach have been conducted using the Botsv1 dataset [4] of the Splunk project. This dataset is public and contains DNS-based C&C traffic. After introducing the Botsv1 dataset, we describe the data preparation. Then, we present the evaluation of the four algorithms and select the best one to further improve. Finally, we analyze the results and show how a simple post-analysis can provide useful information.

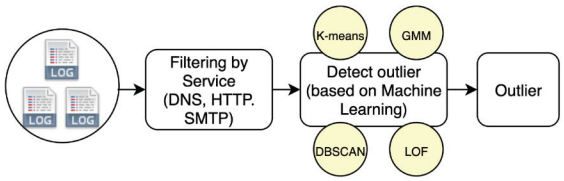

Figure 1. Our proposed approach

A. Choice of the dataset & preparation for evaluation

Our goal is to detect abnormal DNS traffic which may carry C&C other data flows. We searched for public datasets which contain both benign DNS and DNS tunneling. We also want this dataset to be representative of current DNS-based attacks.

Authors of related work ([19] [20] [21]) created their own dataset by using several free tools for performing DNS tunneling, such as IoDine, Dns2tcp (as high- throughput tunneling tools), FrameworkPOS and Backdoor.Win32.Denis (as DNS exfiltration malware). However, these datasets are not public and cannot be reused or evaluated.

Ryan Kovar, David Herrald, and James Brodsky from Splunk have recently published the Boss of the SOC Dataset Version 1 (Botsv1) [4]. This dataset contains evidence captured during actual computer security incidents, or from realistic lab recreations of security incidents. Botsv1 consists of two parts: the original dataset containing all data, and a much smaller version of the original dataset containing only attack data. The original dataset is available in several formats: 6.1GB compressed, several JSON files by source type, several CSV files by source type (such as stream:dns, stream:http, stream:snmp, etc). Our purpose is to detect abnormal DNS traffic, thus we selected "stream:dns" only (step 1 of our process). Splunk also provides an additional log file that contains only the DNS-attack connections (called in the rest of the article DNS-attack logs). This log file is used to evaluate the DNS attack detection performance of the unsupervised algorithms.

B. Data preparation

Since Botsv1-DNS and DNS-attack logs are raw Splunk logs, they must be transformed to create a proper dataset suitable for applying machine learning algorithms. First, we chose the following features: *message_type*, *transaction_id*, *src_ip*, *src_port*, *dest_ip*, *dest_port*, *bytes*, *bytes_in*, *bytes_out*, *time_taken*, *transport*, *timestamps*. From *timestamps*, we extracted three sub-features: *hour*, *minute*, *week number* (0-4: weekday, 5-6: weekend). The categorical variables have been transformed in order to calculate distances. We used one hot encoding for *message_type* (query into 1 and response into 0) and *transport* (tcp into 0 and udp into 1). We also applied a dummy encoding on the network part of the IP addresses (*src_ip* and *dest_ip*).

After selecting and preparing the features, we labelled the Botsv1-DNS dataset using the DNS-attack logs that allows us to determine which connection is abnormal. The resulting labelled dataset contains 1,362,580 genuine DNS connections and 7418 attack connections. In order to evaluate how machine learning algorithms can detect abnormal DNS traffic with different attack percentages, we generated seven sub-datasets of 100,000 records that respectively contain 0.1%, 0.5%, 1%, 2%, 3%, 4% and 5% of attacks.

C. Evaluation of algorithms

The second step of our process consists in applying outliers detection algorithms. We evaluated four algorithms, namely, K-means, GMM, LOF, and DBSCAN (see section III) based on the seven datasets (from 0.1 et 5% of attacks). We configured K-means and GMM to categorize the datasets in two clusters ($K = 2$) to separate genuine and abnormal DNS connections. DBSCAN and LOF were tuned according to the maximum attack percentage (5%). Therefore, we set to 5% the *min_samples* value of DBSCAN, and the number neighbors of LOF. Finally, we defined *eps* to 6 for DBSCAN. This value was obtained experimentally.

We use the detection rate (DR) and the false positive rate (FPR) to evaluate the performance of the algorithms. The detection rate is the number of attacks detected by the system divided by the number of attacks in the dataset. The false

positive rate is the number of normal connections that are misclassified as attacks divided by the number of normal connections in the dataset. As consequence, a good algorithm should achieve a high DR value while keeping the FPR low.

Figure 2 and Figure 3 display the abnormal DNS connections DR and the FPR of the four algorithms for the seven sub-datasets. K-means and GMM yield bad results. The DR of K-means ranges from 54% to 59.3% and the FPR from 57.7% to 58.1%. The DR of GMM is better and varies from 58.5% to 100% but the FPR fluctuates between 57.8% and 96.6%. In addition, the performance of GMM is not stable and depends on the attack rate. LOF has good DR results when the attack rate varies from 0.1% to 2%. However, the FPR ranges from 8.1% to 9.9%. DBSCAN gives the best results when the attack rate is less than or equal to 2% with the DR at 100% and the FPR from 6.6% to 8.5%. The FPR (16%) is higher than LOF when the attack rate is 3%. Nevertheless, the DR of LOF drops to 98% while it remains 100% for DBSCAN. Therefore, we choose DBSCAN to further improvement and considered the attack rate to be less than or equals to 3%. This assumption is consistent with our hypothesis that the attack is conducted by highly skilled attackers.

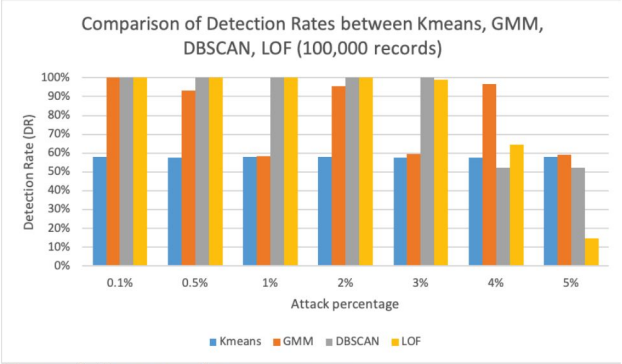

Figure 2. Detection Rate comparison

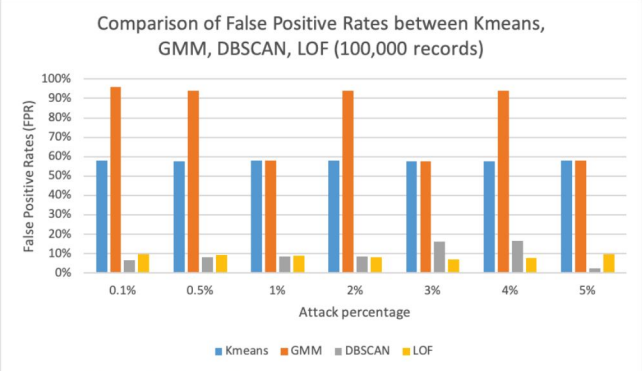

Figure 3. False Positive Rate comparison

We concentrated on DBSCAN and tried to tune the values of *min_samples* and *eps*. Firstly, we kept the value of *eps* to 6 and change the value of *min_samples* to 100, 500, 1000, 2000, and 3000 (i.e. respectively 0.1%, 0.5%, 1%, 2% and 3% of the dataset). Figure 4 and Figure 5 depict the DR and FPR of DBSCAN when changing the value of *min_samples* on the five sub-datasets with different attack percentages (attack rate 0.1%, 0.5%, 1%, 2% and 3%). The best *min_samples* value is clearly corelated with the attack rate when considering the DR. As a consequence, the value of *min_samples* should be defined according to the maximum expected attack rate value. However, the FPR increases at the same time as *min_samples*. When the value of *min_samples* is equal to 2000 (i.e. 2% of the dataset), the FPR ranges from 2.1% to 2.26% while it grows up to 6.6% to 16% when *min_samples* is equal to 3000 (e.g. 3% of the dataset).

With these observations, we decided to choose 2000 as the value of *min_samples* (i.e. 2% of the dataset) when trying to improve the performance of DBSCAN. However, the performance of DBSCAN depends also on the value of *eps*. In this case, we initially defined *eps* to 6 which was obtained experimentally. In order to generalize our approach, we need to calculate the optimal value algorithmically.

The optimal value of *eps* can be computed using the K-nearest neighbors (KNN) similar to the approach proposed by Gaonkar and Sawant [28]. The basic idea of the algorithm is to calculate the average distance between each point and its K-nearest neighbors, where $K = min\_samples$.

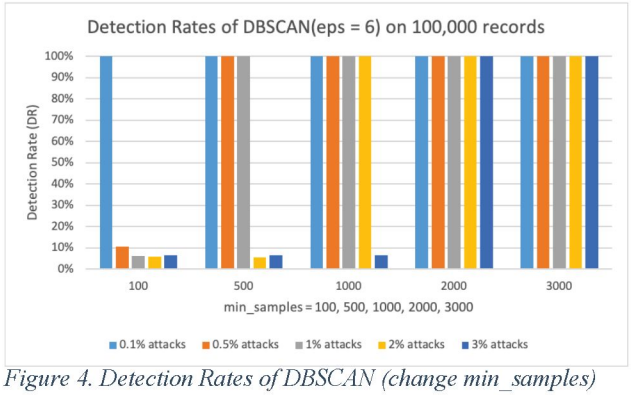

Figure 4. Detection Rates of DBSCAN (change min_samples)

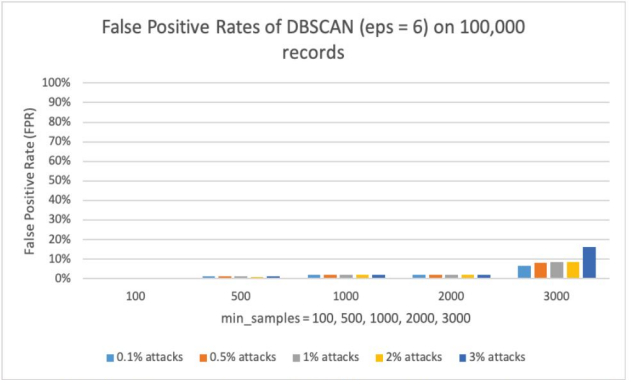

Figure 5. False Positive Rates of DBSCAN (change min_samples)

Then we sort those average distances in ascending order and choose the value that covers 96% of the data to be the value of *eps*. Thus, the value of *eps* can be automatically computed for a given dataset and a given value of K ($K = min\_samples$). By applying this technique, we obtained the value of 6.5 for *eps*. This computed value is close to the one we obtained by experimentation.

To conclude the evaluation of DBSCAN, we calculate the Area Under Curve - Receiver Operating Characteristics (AUC-ROC) [29], which is a performance metric for binary classification problem (in our case, benign and attack). ROC

is a probability curve that demonstrates the performance of models in distinguishing two classes. The whole area underneath the ROC (between the ROC line and the axis) is the AUC. The value of AUC ranges from 0 to 1. The bigger area is, the higher AUC and the better the model is at predicting benign traffic and attacks. Figure 6 shows the performance of DBSCAN. The final result is relatively good in all cases with the DR at 100%, the FPR from 1.6% to 2.3% and AUC from 0.988 to 0.992.

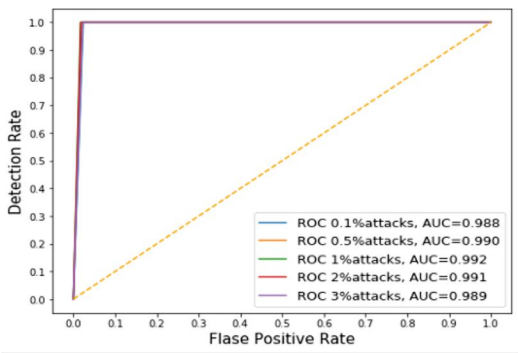

Figure 6. ROC curves for the dataset 100,000 records

D. Critical analysis of results and Post analysis

Using DBSCAN to identify outliers gives relatively good results with DR at 100% and the FPR ranges from 1.6% to 2.3%. However, 2.3% of FPR in a dataset containing 100,000 records means there are 2300 benign connections misclassified. Therefore, detecting infected hosts still requires a lot of manual work.

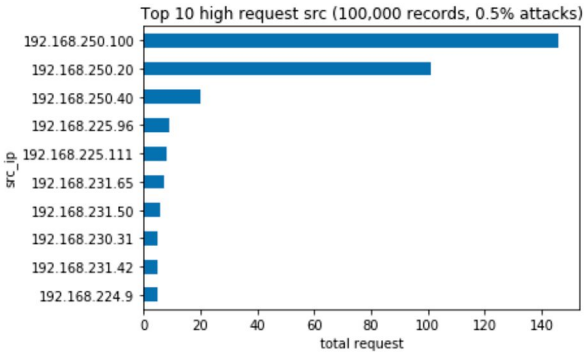

Figure 7. Top 10 number of requests of outlier dataset

We analyzed the results after applying DBSCAN to see if we can determine more precisely abnormal DNS communications. Infected hosts change their DNS behavior like increasing the volume of requests or the volume of bytes out that may identify signs of C&C communication or data movement. Thus, we ranked the IP addresses in the abnormal cluster regarding the number of requests and the number of bytes sent out. Figure 7 shows the top 10 high number of requests and Figure 8 displays the top 10 high volume of bytes sent out of the anomaly cluster after applying DBSCAN on the dataset with 0.5% attacks. Three IP addresses stand out: 192.168.250.100, 192.168.250.20 and 192.168.250.40. These IP addresses correspond exactly to the three machines that were infected in the dataset.

One can argue that apply the same ranking on the whole dataset would give the same result. Figure 9 displays the top 10 high requests and Figure 10 shows the top 10 high bytes out from the original 0.5% attacks. We can see that 192.168.250.100 and 192.168.250.20 are hidden in the middle of this top 10 ranking without any significative difference from benign DNS traffic. Worst, 192.168.250.40 is not in the top 10.

This proves the usefulness of applying DBSCAN to detect outliers and that a simple post-analysis on the abnormal cluster can help to discover infected hosts.

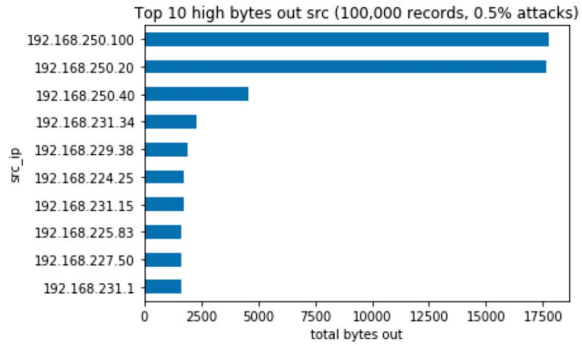

Figure 8. Top 10 volume of bytes out of outlier dataset

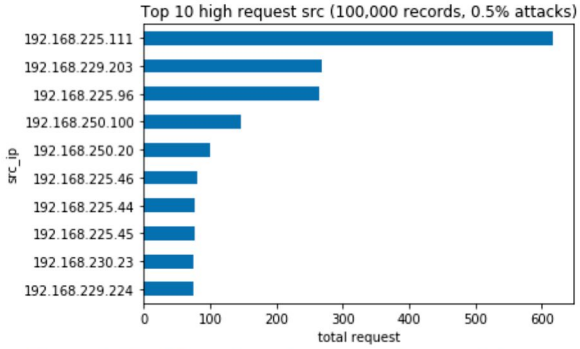

Figure 9. Top 10 number of requests of original dataset

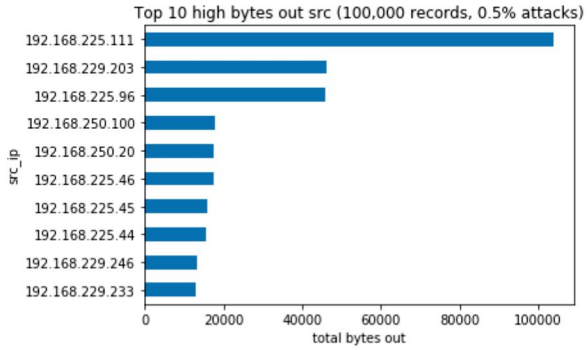

Figure 10. Top 10 volume of bytes out of original dataset

V. Conclusion and Future Work

In this paper, we presented a behavioral analysis approach based on unsupervised machine learning for detecting malicious DNS traffic with the hypothesis that the dataset contains only DNS traffic, and malicious traffic is rare (less than 3%). We used the logs Botsv1 provided by Splunk and extracted the information to build a dataset. We compared the performance of four algorithms K-means, GMM, LOF, and DBSCAN on the dataset 100,000 records with different attack percentages. DBSCAN gives the best result with the DR at

100% and the FPR ranges from 6.6% to 16% in case the attack rate varies from 0.1% to 3%.

We improved the performance of DBSCAN by tuning parameters *min_samples* and *eps*. The value of *min_samples* depends on the attack rate. Setting it at 2% of the dataset allows us to detect abnormal DNS traffic ranging from 0.1% to 3% of the whole traffic. Using KNN gives us the ability to automatically determine a good value for *eps*. Thus, we are able to achieve 100% DR, while keeping the FPR as low as 1.6% to 2.3% and the AUC as high as 0.988 to 0.992.

After removing benign entities, we analyzed the outlier cluster detected by DBSCAN. By looking at changes in DNS behavior like the high number of requests or volume of bytes out, we could determine the infected hosts, which was not possible in the original data.

In the future, we will extend our approach to other protocols like HTTP(S) and SMTP(S). We will also apply ensemble learning methods [30] that combines the decisions from multiple models to improve the overall detection performance and reliability. Finally, we aim at integrating such techniques into a global security management infrastructure (e.g., [31], [32]) to dynamically respond to detected attacks.

ACKNOWLEDGMENT

This work was supported by MODIS, France. Special thanks go to Louis Garcia, Charles Fortunet (Innovation Center, MODIS) for their fruitful comments.

REFERENCES

[1] A. Benzekri, R. Laborde, A. Oglaza, D. Rammal, and F. Barrère, "Dynamic security management driven by situations: An exploratory analysis of logs for the identification of security situations," in *2019 3rd Cyber Security in Networking Conference (CSNet)*, 2019, pp. 66–72.

[2] "Cybersecurity 2019-2020." https://www.ptsecurity.com/ww-en/analytics/cybersecurity-2019-2020/ (accessed Sep. 18, 2020).

[3] "APT32, SeaLotus, OceanLotus, APT-C-00, Group G0050 | MITRE ATT&CK®." https://attack.mitre.org/groups/G0050/ (accessed Jun. 12, 2020).

[4] *splunk/botsv1*. Splunk GitHub, 2020.

[5] E. Leon, O. Nasraoui, and J. Gomez, "Anomaly detection based on unsupervised niche clustering with application to network intrusion detection," in *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753)*, Portland, OR, USA, 2004, pp. 502–508, doi: 10.1109/CEC.2004.1330898.

[6] "KDD Cup 1999 Data." http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html (accessed Sep. 18, 2020).

[7] P. Casas, J. Mazel, and P. Owezarski, "Unsupervised Network Intrusion Detection Systems: Detecting the Unknown without Knowledge," *Comput. Commun.*, vol. 35, no. 7, pp. 772–783, Apr. 2012, doi: 10.1016/j.comcom.2012.01.016.

[8] P. V. Amoli, T. Hamalainen, G. David, and M. Zolotukhin, "Unsupervised Network Intrusion Detection Systems for Zero-Day Fast- Spreading Attacks and Botnets," p. 13.

[9] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Comput. Secur.*, vol. 31, no. 3, pp. 357–374, May 2012, doi: 10.1016/j.cose.2011.12.012.

[10] R. Aliakbarisani, A. Ghasemi, and S. Felix Wu, "A data-driven metric learning-based scheme for unsupervised network anomaly detection," *Comput. Electr. Eng.*, vol. 73, pp. 71–83, Jan. 2019, doi: 10.1016/j.compeleceng.2018.11.003.

[11] "Traffic Data from Kyoto University's Honeypots." http://www.takakura.com/Kyoto_data/ (accessed Sep. 20, 2020).

[12] "NSL-KDD | Datasets | Research | Canadian Institute for Cybersecurity | UNB." https://www.unb.ca/cic/datasets/nsl.html (accessed Sep. 20, 2020).

[13] Y. Kazato, Y. Nakagawa, and Y. Nakatani, "Improving Maliciousness Estimation of Indicator of Compromise Using Graph Convolutional Networks," in *2020 IEEE 17th Annual Consumer Communications & Networking Conference (CCNC)*, Las Vegas, NV, USA, Jan. 2020, pp. 1–7, doi: 10.1109/CCNC46108.2020.9045113.

[14] H. Gunes Kayacik, A. Nur Zincir-Heywood, and M. I. Heywood, "A hierarchical SOM-based intrusion detection system," *Eng. Appl. Artif. Intell.*, vol. 20, no. 4, pp. 439–451, Jun. 2007, doi: 10.1016/j.engappai.2006.09.005.

[15] S. Zanero and G. Serazzi, "Unsupervised learning algorithms for intrusion detection," in *NOMS 2008 - 2008 IEEE Network Operations and Management Symposium*, Salvador, Bahia, Brazil, 2008, pp. 1043–1048, doi: 10.1109/NOMS.2008.4575276.

[16] "1999 DARPA Intrusion Detection Evaluation Dataset | MIT Lincoln Laboratory." https://www.ll.mit.edu/r-d/datasets/1999-darpa-intrusion-detection-evaluation-dataset (accessed Sep. 18, 2020).

[17] Y. Chen, J. Zhang, and C. K. Yeo, "Network Anomaly Detection Using Federated Deep Autoencoding Gaussian Mixture Model," in *Machine Learning for Networking*, vol. 12081, S. Boumerdassi, É. Renault, and P. Mühlethaler, Eds. Cham: Springer International Publishing, 2020, pp. 1–14.

[18] Md. S. Alam *et al.*, "Memristor Based Autoencoder for Unsupervised Real-Time Network Intrusion and Anomaly Detection," in *Proceedings of the International Conference on Neuromorphic Systems*, Knoxville TN USA, Jul. 2019, pp. 1–8, doi: 10.1145/3354265.3354267.

[19] I. Homem and P. Papapetrou, "HARNESSING PREDICTIVE MODELS FOR ASSISTING NETWORK FORENSIC INVESTIGATIONS OF DNS TUNNELS," p. 12, 2017.

[20] H. Lin, G. Liu, and Z. Yan, "Detection of Application-Layer Tunnels with Rules and Machine Learning," in *Security, Privacy, and Anonymity in Computation, Communication, and Storage*, vol. 11611, G. Wang, J. Feng, M. Z. A. Bhuiyan, and R. Lu, Eds. Cham: Springer International Publishing, 2019, pp. 441–455.

[21] A. Berg and D. Forsberg, "Identifying DNS-tunneled traffic with predictive models," *ArXiv190611246 Cs*, Jun. 2019, Accessed: Jun. 09, 2020. [Online]. Available: http://arxiv.org/abs/1906.11246.

[22] C. Jose, "Anomaly Detection Techniques in Python," *Medium*, May 13, 2019. https://medium.com/learningdatascience/anomaly-detection-techniques-in-python-50f650c75aaf (accessed Jun. 12, 2020).

[23] J. Macqueen, "SOME METHODS FOR CLASSIFICATION AND ANALYSIS OF MULTIVARIATE OBSERVATIONS," *Multivar. Obs.*, p. 17.

[24] D. Reynolds, "Gaussian Mixture Models," in *Encyclopedia of Biometrics*, S. Z. Li and A. Jain, Eds. Boston, MA: Springer US, 2009, pp. 659–663.

[25] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum Likelihood from Incomplete Data Via the *EM* Algorithm," *J. R. Stat. Soc. Ser. B Methodol.*, vol. 39, no. 1, pp. 1–22, Sep. 1977, doi: 10.1111/j.2517-6161.1977.tb01600.x.

[26] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise.," in *Kdd*, 1996, vol. 96, no. 34, pp. 226–231.

[27] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: Identifying Density-Based Local Outliers," p. 12.

[28] M. N. Gaonkar and K. Sawant, "AutoEpsDBSCAN : DBSCAN with Eps Automatic for Large Dataset," vol. 2, no. 2, p. 6.

[29] A. P. Bradley, "The use of the area under the ROC curve in the evaluation of machine learning algorithms," *Pattern Recognit.*, vol. 30, no. 7, pp. 1145–1159, Jul. 1997, doi: 10.1016/S0031-3203(96)00142-2.

[30] O. Sagi and L. Rokach, "Ensemble learning: A survey," *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*, vol. 8, no. 4, Jul. 2018, doi: 10.1002/widm.1249.

[31] B. Kabbani, R. Laborde, F. Barrere, and A. Benzekri, "Specification and enforcement of dynamic authorization policies oriented by situations," in *New Technologies, Mobility and Security (NTMS), 2014 6th International Conference on*, 2014, pp. 1–6, Accessed: Aug. 28, 2015. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6814050.

[32] R. Laborde, A. Oglaza, A. S. Wazan, F. Barrère, and A. Benzekri, "A situation-driven framework for dynamic security management," *Ann. Telecommun.*, vol. 74, no. 3–4, pp. 185–196, 2019.