International Journal of Sensor Networks and Data Communications

Adams C, et al, Int J Sens Netw Data Commun 2019, 8:3

Research Open Access

Using Machine Learning to Detect APTs on a User Workstation

Adams C*, Tambay AA, Bissessar D, Brien R, Fan J, Hezaveh M and Zahed J

School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, Canada

***Corresponding author:** Dr. Carlisle Adams, Associate Director, Professor, School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, Canada, Tel: 613-562-5800/2345; E-mail: cadams@uottawa.ca

Received date: August 27, 2019; **Accepted date:** October 09, 2019; **Published date:** October 16, 2019

Abstract

Advanced Persistent Threats (APTs) are explicitly designed to be difficult to detect, but their activities necessarily include some differences from what a regular user might do. We present an analysis and comparison of four machine learning algorithms that were used to first learn a user's behavior and then to detect APT activity as an anomaly in that behavior. We also present our methodology for each step of the analysis. In particular, for each user, we collected data with Osquery on a clean machine before running the Red Team Automation (RTA) scripts to simulate an APT attack. The four algorithms we tested on each user's data (neural networks, decision trees, k-means clustering, and one-class SVM) included supervised, unsupervised and one-class algorithms. This study was undertaken as a proof-of-concept exercise to see if machine learning could be beneficial in APT detection, and our results indicate that looking at user behaviour for APT detection appears to be a promising approach. Previous work focused on APT behaviour (particularly in the context of network traffic), whereas our goal is to detect APTs on the computer where the legitimate user is present and active and to detect the APTs by discovering anomalies with respect to typical user behaviour.

Keywords Advanced Persistent Threat (APT); Machine learning; User behaviour

Introduction

This paper addresses the following problem: When a user is actively using a machine, how can we distinguish between the activity of the legitimate user and the activity of a malicious entity, such as an Advanced Persistent Threat (APT)? Previous work in this field (see, for example, [1-6]) is effective for intrusion detection generally, and may help with detecting APT behaviour on an otherwise idle machine, but those approaches may not be as successful when attempting to identify the user's normal behaviour (in order to distinguish it from deliberately stealthy malware behaviour).

Machine learning can potentially provide an interesting approach to identifying APTs. If you have a machine that you suspect may host malicious activity, by eliminating what is known to be the genuine user's activity, you can be left with what is most likely to be the malicious activity; in the case where that malicious activity is the behaviour of an APT, you would then have indirectly detected the presence of a APT. This approach differs from misuse-based and specification-based models of intrusion detection (wherein a precise model of attack behaviour, or a precise model of proper program behaviour, is specified and compared with observed system activity), and is instead in the category of anomaly-based models. However, in our study we focus on learning the behaviour of a specific targeted user on the user's home machine; this differs from many anomaly-based intrusion detection applications that aim to model "normal" system activity over an entire network or subnetwork (with many users and many machines) and then look for deviations from this expected aggregate behaviour. Furthermore, our work differs from previous research that uses machine learning to detect APTs (see, for example, [7-12]). These previous studies use ML algorithms to analyze *network traffic* for exfiltrated data or for connections to unknown domains, to detect suspicious *URLs or malicious files*, or to examine *network traffic logs* with big data analytics. Our approach is rather to learn the typical behaviour of an individual user and to look for anomalies in the system activity on that user's computer.

Therefore, in this paper, we explore the possibility of using machine learning algorithms to distinguish between user behaviour and APT behaviour. This paper begins by outlining the methodology used to accomplish this goal. We then describe how the data was prepared for training and testing, followed by an overview of the different types of machine learning algorithms that were analyzed and used. We then provide information about the algorithms' implementations that were used, as well as our experimental results. Finally, we conclude with a discussion regarding directions for future work in this research area.

Research Methodology

The research for this paper was done in two main phases: the exploration phase and the experimentation phase. Each phase was further separated into data collection, data preparation, and the machine learning step. The exploration step also included an additional data exploration step. In this section, we describe each of the two phases. The specific details are covered in sections 3 and 4.

In the exploration phase, we familiarized ourselves with Osquery [13] (see Section 3.1) and the Red Team Automation (RTA) scripts [14] (see Section 3.2). Our first step in this phase, the data collection, consisted of collecting all the data Osquery could provide on a clean virtual machine with no user activity for a period of time, then running the RTA scripts. We classified the data collected on the clean machine with an "attack_state" of 0, and the data collected during the running of the RTA scripts with an "attack_state" of 1.

The second step was data exploration. In this step, we looked at the source data to get an "initial sense" for the data's relationships: the effect of various RTA attacks on the data and on the various Osquery

tables. This process included transforming the Osquery logs into a usable set of files.

The data was then prepared to work as input for the various machine learning algorithms used in our study. Since the Osquery data contains repeating groups (i.e., multiple processes, services) and multiple datatypes (i.e., strings like filenames and process names, floating-point values like disk read rate, integers, etc.), we made decisions on how to transform or aggregate the data, and what data to merge into categories. Every decision went through various iterations; each iteration was tested on neural networks to see how it would fare.

The whole process was then repeated with user activity on the virtual machine and led to choosing general user activities for data collection, narrowing the data collected with Osquery to relevant tables. The preliminary data preparation rules were encoded in a python script.

The experimentation phase first consisted of data gathering: three users worked on virtual machines performing the user activities we selected during the exploration phase (Section 3.3). We then iterated between using and updating our data preparation rules, and testing the data on the various machine learning algorithms we chose. We then gathered, analysed and compiled the results (Section 5).

Data Collection and Preparation

Osquery

Osquery [13] was used for this paper; this is a product developed by Facebook to gather detailed information about the operating system. The collected information can then be queried as a relational database. Osquery can be used as an interactive console or as a monitoring tool. Its console version, Osqueryi, can be used to query the current state of the machine. The second version, Osqueryd, is a daemon that can be used to schedule queries and record logs of the machine state over time. The daemon logs the changes to the database since the last time the query was executed.

Red Team Automation (RTA)

The Red Team Automation (RTA) [14] is a collection of python scripts that executes or emulates various types of attacks seen from actual APTs. RTA is widely used by researchers and malware detection experts for testing and training purposes since it conveniently packages the malicious activities of many different APTs (and so testing does not need to be done on a single APT at a time). We used their scripts during data collection to simulate malicious activities. Note that the following RTA scripts do not currently run on Windows 10; therefore, they were omitted from our attack list:

at_command.py	delete_shadow.py	msxsl_network.py	net_user_add.py
egsvr32_scrobj.py	sip_provider.py	system_restore_process.py	
trust_provider.py	user_dir_escalation.py		

Data acquisition

Our goal is to determine if it is possible to use a machine learning algorithm to detect the presence or absence of APT activity on a machine. As such, we wished to label our data as "clean" (i.e., data from an uncompromised machine), and "dirty" (i.e., data from a machine that was potentially compromised due to the presence of an APT). For the purposes of our study, simulated data was generated and captured for both the "clean" and "dirty" environments (with the help of Osquery and the RTA scripts, as described above).

Using Osquery's daemon, multiple sets of data spanning three-hour periods were aggregated. To simulate a working user, we defined a series of tasks as activities likely to be performed by a regular user in an office context. These tasks were executed during those three hours on a virtual machine while all queries available were run with a 1-minute interval between queries. RTA scripts used to simulate APT activity were then run on the machine to emulate the behavior of a machine under attack. In addition, to emulate the noise generated by python while running the RTA scripts, an additional dummy python script ran alongside the tasks on the clean version of the machine.

From Osquery's available tables, we kept the following:

Interface_Details	Listening_Ports	Logical_Drives	Physical_Disk_Performance
pipes	process_memory_map		process_open_sockets
registry	scheduled_tasks	services	windows_events

The importance of good data cannot be overstated. The data used for training has a large impact on the effectiveness of the model. As such, getting data generated by real users working at their own workstations would be ideal. If this is not possible, simulated data can replace it but, in this case, one should try to use a workstation that simulates "normal usage" as closely as possible for more accurate results. Therefore, one should try to study a real working user in order to replicate his/her behavior. If possible, multiple types of users recorded should be recorded in order to resemble a typical organization more closely. In either case, one should record for long periods of time (ideally in periods spanning full work days, with multiple such periods per user).

The study described in this paper is intended to be a proof-of-concept exercise to determine the feasibility of using machine learning to detect APT activity. Because our study was constrained in both the number of users and the time we had available to conduct our experiments, we were able to construct only a very limited initial dataset. To create our data, three users collected Osquery results from a virtual machine running Windows 10 for approximately 2 to 3 hours. During the data collection, the users tried to behave like normal users. Here is the list of tasks each user had to do during data collection:

- Use Microsoft Office (Word, Excel, Power-Point). Save, copy, Delete documents. Empty data from the recycle bin.
- Use Gmail to send emails and attach some data.
- Use Dropbox to upload and download data.
- Run a python script on the command prompt.
- Search the web using Google.
- Watch YouTube videos.

The tasks were to be done in no particular order, as we tried to capture an individual's typical behavior. Note that these actions were repeated during the initial (i.e., "clean") data collection, and after running the RTA script (i.e., in the "dirty" environment) the user continued working on her/his machine as a normal user.

Data preparation

A python script was used to convert the data from the logs generated by Osquery to a usable format. The logs are in a JSON format and need to be manipulated before being saved as csv files that can be used for machine learning. The script loads the JSON file into memory and for each line identifies a few crucial details:

- The name of the table queried, to determine the corresponding query
- The name of all the fields that contain data, to be used as columns
- The timestamp at which the query was executed, for time series analysis
- The source file, to label the data
- The action taken, as an indicator of whether this was a change added to the machine state, or removed, as per Osqueryd's differential logs [15].

This extra information, as well as the data returned by Osquery, is added to a dictionary as lists split by query. Each of these lists is then converted to a pandas dataframe. For each of these dataframes, which represent the information gathered from a single query, a custom function is used to extract the information pertinent to the machine learning algorithm. The extracted information is aggregated into a new dataframe which is saved as a .csv file, which can be loaded by the machine learning algorithms used.

The data we fed the algorithms in our experimentation are an aggregation of selected variables in each of the different dataframes. The aggregate values we used included the number of unique identifiers in each table at the time of query, as well the number of "added" and "removed" actions recorded by the Osquery since the last query on each table. The amount of "added/removed" actions gives us an idea of how much change occurred since the previous query. We then combined the various timestamps into observations containing the changes observed in 60-second intervals.

After all the data preparation, we have three datasets on which we tested different machine learning algorithms. These are the datasets that were generated by different users:

- Contained 108 clean observations and 8 attack observations, for a total of 116 observations.
- Contained 166 clean observations and 7 attack observations, for a total of 173 observations.
- Contained 153 clean observations and 17 attack observations, for a total of 172 observations.

Machine Learning Algorithms

This section presents a high-level overview of the machine learning algorithms that were analyzed and used in this research. For each, we give a brief description of how the algorithm works, along with a discussion of its strengths and weaknesses. Readers who are already familiar with various machine learning algorithms and their properties may choose to skip to Section 5 (Implementation and Results).

Supervised learning

Supervised learning is a type of machine learning algorithm that uses two kinds of datasets: a training dataset and a testing dataset. In these two datasets, both input and desired output data are provided. Supervised learning uses the training dataset to build a model that can make predictions of the response values for a new dataset, while the test dataset is used to validate the model. Larger training datasets usually means models will yield higher predictive power. Supervised learning enables humans to interfere with or modify decisions that the system makes, but it has trouble addressing new information that has not been included in the training datasets.

Supervised learning algorithms can be roughly categorized into Classification (for discrete output) and Regression (for continuous output). Some of the most popular algorithms include Support Vector Machines (SVM), linear regression, logistic regression, naïve Bayes, linear discriminant analysis, decision trees, k-nearest neighbor, and Neural Networks (Multilayer perceptron).

For this paper, we used a neural network [16,17] and a decision tree [5,18,19]. Both these algorithms can be used for Classification and Regression.

We first chose neural networks because they can be considered as a mainstay in supervised learning algorithms and they have been used in a multitude of areas. As such, libraries implementing them are easily available. Decision trees were chosen because they provide a "white box" approach, where a human can look at the resulting model and gain insight on how and why the model classifies things in any given way.

Neural network

A neural network is usually structured into an input layer of neurons, one or more hidden layers, and one output layer. Typically, neurons are fully connected to the next layer. However, there are different types of neural networks that may have different topologies of layer connections and activation functions. A value of the function corresponding to connections is called a "weight". To achieve a suitable output from the input, we need to set a suitable value for weights (in our implementation we consider two hidden layers with weights of ten).

Strengths of neural networks [17]: Neural networks are flexible and can be used for both regression and classification problems. Any data that can be made numeric can be used in the model because a neural network is a mathematical model with approximation functions.

- Neural networks are good for modeling nonlinear data with a large number of inputs (for example, images). It is a reliable approach for tasks involving many features. It works by splitting the problem of classification into a layered network of simpler elements.
- Once trained, the predictions are produced very quickly.
- Neural networks can be trained with any number of inputs and layers.
- Neural networks work best with more data points.

Weaknesses of neural networks [17]: Neural Networks are black boxes, meaning we cannot know how much each independent variable is influencing the dependent variables.

- It is computationally very expensive and time consuming to train a neural network with traditional CPUs.
- Neural networks depend a lot on training data. This leads to the problem of over fitting and generalization: the model relies heavily on training data and may inadvertently be tuned to the data.

Decision tree

The goal of decision tree algorithms is to create a model that predicts the value of a target variable based on several input variables.

It constructs decision trees from labelled training datasets. Data comes in attribute vectors of this form: (x1, x2, x3,..., xk, Y). The dependent variable, Y, is the target variable that the decision trees will try to "learn" and then to predict. The features, x1, x2, x3 etc., are used for tasks. A decision tree is a flow-chart-like structure, in which each non-leaf node is labelled with an attribute (x), each branch represents the outcome of a test, and each leaf node will have a class label (Y). The topmost node in a tree is the root node [18].

Strengths of decision trees:

- Fast and accurate in learning a huge dataset [5]
- Able to handle both numerical and categorical data
- Able to handle both continuous and discrete data with the proper algorithm
- Easy to understand and interpret the created pattern
- Requires relatively little effort for data preparation: the decision tree has no need for variable scaling, it not affected by outliers, and it can deal with a reasonable number of missing values
- It uses a white box model, which means that the acquired knowledge can be explained in a readable form. By contrast, in a black box model (e.g. SVM and neural network), you cannot read the acquired knowledge in a comprehensible way
- It makes decisions more like humans do (compared to other approaches).

Weaknesses of decision trees:

- Exponential calculation growth as the problem gets bigger.
- This technique can create overly-complex trees that do not generalize well from the training data. Mechanisms such as pruning are necessarily needed [19].
- Small changes in the training data can result in large changes in the tree construction and consequently the final predictions [20].
- Too much comprehensive information and all possible solutions to an issue will require more time to process and will slow down decision-making capacity [21].

There are many notable decision-tree algorithms, such as ID3 (Iterative Dichotomiser 3), C4.5 (successor of ID3) and CART (Classification and Regression Tree). ID3 applies information theory and its class is based on entropy. Since ID3 is not likely to use continuous attributes, the C4.5 algorithm was developed to complement this [5]. C4.5 also addresses the problems of over-fitting and tree pruning in ID3 that will degrade the performance and increase the cost [4]. We utilize the C4.5 algorithm in our study; it can use continuous data and still build a relatively accurate model even with many input attributes [5].

The algorithm has some premises, such as the following [22]:

- If all cases are of the same class, a leaf node will be created, labeled with this class.
- For each attribute, calculate the potential information provided by a test on it. Also, calculate the information gain that would result from a test on the attribute.
- Find the most proper attribute to have the splitting criterion.

C4.5 works in this way [18]: at each node of the tree, the algorithm chooses the attribute of the data that most effectively splits its set of samples into subsets enriched in one class or the other. The attribute with the highest normalized information gain will be chosen to make this splitting decision. Then the algorithm works recursively on the subtrees. Pruning happens after the tree is created completely. This will reduce the classification errors that are caused by outliers in training data, and will make the tree more general.

The authors of the Weka machine learning software described the C4.5 algorithm as "a landmark decision tree program that is probably the machine learning workhorse most widely used in practice to date" [23].

Unsupervised learning

So far, in all the supervised machine learning algorithms we considered, a training example that contains a set of features categorized with labels was used. In unsupervised learning, a training set contains just a set of features alone with no labelling. The purpose of unsupervised learning is to attempt to find natural partitions in the training set where the data has no target attribute [24]. In unsupervised learning techniques, we are trying to analyze the data to find the connections between them.

Clustering is a technique for finding similarity groups in data, called clusters. In this technique, the algorithm tries to find the data points that are nearest to each other and classifies them in one group, and the data points that are furthest away from this set are classified in another group. The quality of a clustering result depends on the algorithm, the distance function, and the application [25].

K-means

The k-means algorithm divides the data into k clusters. Each cluster has a cluster center, called a centroid. The parameter k is specified by the user. In our implementation, k is equal to 2, as we have two states: "attack" and "no-attack". It may also be valuable to test this scheme by considering k=3 for three states: "attack", "no-attack-user present" and "no-attack-without user".

Given k, the k-means algorithm works as follows [26]:

- Randomly choose k data points (seeds) to be the initial centroids/ cluster centers
- Assign each data point to the closest centroid
- Re-compute the centroids using the current cluster memberships
- If a convergence criterion is not met, go back to step 2.

Strengths of k-means [26]:

- It is easy to understand and easy to implement
- It has time complexity , where is the number of data points, is the number of clusters, and is the number of iterations. Since both k and t are small, k-means is considered a linear algorithm. k-means is the most popular clustering algorithm.

Weaknesses of k-means [26]:

- The algorithm is only applicable if the mean is defined
- The user needs to specify k
- The algorithm is sensitive to outliers
- Outliers are data points that are very far away from other data points
- Outliers could be errors in the data recording or some special data points with very different values.

In conclusion, k-means is the most popular algorithm due to its simplicity and efficiency. However, it has some serious weaknesses. We should mention that the other clustering algorithms have also their own lists of weaknesses. There is no concrete evidence that shows that one clustering algorithm performs better than another one. It all depends on the data or applications, and the best way to measure

performance is by trying them with the data and comparing the results. We chose k-means for our study because we found an open-source implementation of it with Tensor Flow [27] and it is a well-known and widely used algorithm for unsupervised learning.

One-class SVM

Support Vector Machines (SVM) is a class of supervised learning algorithm. Given training data belonging to two classes, an SVM will build a hyperplane or line that separates them while maximizing the distance between each class and the hyperplane. The data points are then classified either with y =1 or y = -1 depending on which side of the hyperplane they are.

Although SVMs are built to do linear classification, they can also be used to do non-linear classification with the use of "kernel functions". These kernels are special distance functions that act on the data by implicitly mapping it into a higher dimensional space. The data is then separated by a hyperplane in that new space, which translates to non-linear classification in the original space.

The one-class SVM is a special case of SVM first presented in [28]. It is a supervised algorithm where the training data only belongs to one class. The model is then trained to assign data that is different from the training data into a second class. One-class SVMs are thus useful for anomaly detection, where a significant amount of "normal" data is available, and the abnormal data is either rare, harder to collect, or simply hard to define.

Results and Discussion

Implementation

Three users gathered their data on a Windows 10 VMware machine for 2 to 3 hours. At first, users gathered data on their clean machine by acting as a normal user who checks his/her email, searches on Google, works with Microsoft tools such as Word, Excel and PowerPoint, watches YouTube videos, uploads and downloads files from/to Dropbox, etc. Then, they ran RTA attacks while continuing their activities. During all these activities, Osquery gathers system statistics. Our implementation results are the result of this data being fed into different machine learning algorithms.

In addition to accuracy, we also chose to look at recall, precision, and Matthews Correlation Coefficient (MCC). This is because accuracy alone is not a reliable metric for the real performance of a classifier since it will yield misleading results if the data set is unbalanced.

We found and tested the open source implementations of a decision tree C4.5 from GitHub [29], a neural network classifier model from TensorFlow [30], a k-means clustering model from TensorFlow [27], and a one-class SVM machine learning algorithm from the scikit-learn python library [31]. We separated our datasets into training datasets and testing datasets. Typically, most of the data is used for training and the smaller portion is used for testing. For decision tree, neural network, and k-means, we separated our three datasets to 90%, 80%, 70%, and 60% of the clean data and under-attack data for training, with 10%, 20%, 30%, and 40%, respectively, for testing. This gave us a number of variations to try so that we could see what produced the best results.

For one-class SVM, we took about 85% of the clean data (the first 85%, to simulate a real application) as training data and kept the remaining clean data and all the attack data for testing. One-class SVM takes three main parameters as inputs to generate the model: a kernel, a value 'gamma', and a value 'nu' between 0 and 1. The available kernels in scikit-learn are "linear", "poly", "rbf", and "sigmoid". The value 'gamma' is a coefficient used with a non-linear kernel. The value 'nu' relates to a bound on the maximum amount of training error allowed as well as a bound on the minimum number of training inputs that will be in the support of the classifier.

Furthermore, for one-class SVM, each dataset was tested on the 4 available kernels with various pairs of gamma and nu values. The values of gamma and nu were iterated over their permissible range of values.

Outside of the numbers themselves, it is important to recall that neural networks and decision trees are supervised algorithms, k-means is unsupervised, and one-class SVM is trained only on clean data. Tables 1-3 show the best results of the implemented methods for three users (the complete results for user A are available in the Appendix).

User A: 108 rows clean data and 8 rows under attack		**Neural Network 80% for Training and 20% for testing**		**K-means 60% for Training and 40% for testing**		**Decision Tree 70% for Training and 30% for testing**		**One-Class SVM 'poly', d=3, nu=0.007751 gamma=0.05**	
TP	FP	1	0	0	0	3	0	13	2
FN	TN	1	22	4	44	0	33	3	5
Recall		0.5		0		1		0.8125	
Precision		1		0		1		0.8667	
MCC		0.691564		0		1		0.5089	
Accuracy		0.9583333		0.9166666		1		0.7826	

Table 1: Confusion matrix and results for user A.

From the results of the three users, we can see that *neural networks* work best with more data points. The results show the algorithm is always correct in guessing the true negatives as we have more data in our "no-attack" state. In addition, the machine cannot really guess the true positives correctly in many situations for two reasons. First, there is not enough data in our "attack" state; second, it could be that some

states in our "attack" category should really be in the "no-attack" category (because it is possible that at the specific point in time that Osquery was gathering that data, there was no attack running and therefore the data was mislabelled).

User B: 166 rows clean data and 7 rows under attack		**Neural Network 80% for Training and 20% for testing**		**K-means 60% for Training and 40% for testing**		**Decision Tree 60% for Training and 40% for testing**		**One-Class SVM 'linear', d=3, nu=0.007751 gamma=0.05**	
TP	FP	0	0	0	4	3	0	22	0
FN	TN	1	33	3	63	0	67	6	1
Recall		0		0		1		0.7857	
Precision		0		0		1		1	
MCC		0		-0.052093226		1		0.3350	
Accuracy		0.905882		0.9		1		0.7931	

Table 2: Confusion matrix and results for user B.

User C: 154 rows clean data and 17 rows under attack		**Neural Network 70% for Training and 30% for testing**		**K-means 60% for Training and 40% for testing**		**Decision Tree 80% for Training and 20% for testing**		**One-Class SVM 'sigmoid', d=3, nu=0.007751 gamma=0.05**	
TP	FP	1	0	0	0	4	0	2	20
FN	TN	4	46	7	62	0	31	11	6
Recall		0.2		0.0		1		0.1538	
Precision		1.0		0.0		1		0.0909	
MCC		0.428952211		0.0		1		-0.5850	
Accuracy		0.9019608		0.898550724		1		0.2051	

Table 3: Confusion matrix and results for user C.

In the case of *k-means*, results are not good at all for true positives, which gives us reason to think that k-means, with k=2, might not be a good technique to use for our current purpose. We plan to try other unsupervised learning algorithms as well to confirm that the unsupervised learning technique is truly not suitable for APT detection.

While we were testing the k-means algorithm, we noted that there can be an issue with mapping the results of the k-means clustering to the ground truth state [32,33]. The issue is that k-means is a clustering algorithm rather than a classification algorithm, and clustering works on unlabelled data. In general, there is an unknown correlation between the clusters derived and the ground truth state of the unlabelled data. In our study, however, we have labels. We can do a k-means on the element attributes without the labels, and then compare the clusters with the known labels to determine a) a semantic mapping between our cluster centroids and our classes, and b) a sense of "confidence" on how our model performs against the known classes of the training data. Given a) and b) above, we can evaluate a test set against the centroids. The closest centroid to a test row and our semantic mapping (from a) gives us the predicted class. Since the test set is labelled, we have our ground truth. We can then come up with the two vectors 'actual' and 'predicted' required to create our confusion matrix. The "sense of confidence" obtained in b) is a function of the error observed in a cluster centroid (given the semantic mapping from a) and the ground truth (given by the label on the training rows).

For *decision trees*, the algorithm seems very good at learning on a huge dataset that has many attributes. In our case, it shows 100% accuracy on prediction with significantly high probabilities (which, in fairness, likely indicates over-fitting to the data). To be specific, our model can accurately predict the true negatives, while sometimes it cannot accurately predict the true positives. This is because we have more data in our "no-attack" state than in our "attack" state. Furthermore, there is not enough data in our "attack" state. One piece of evidence is that the metrics (recall, precision, and MCC) rise with the increase of "attack" data used for training and testing. The higher these metrics are, the better the performance of this model.

Finally, the results of one-class SVM's predictions vary greatly depending on the choice of kernel and the choice of the 'nu' parameter. In many cases, the model is either too strict by classifying most inputs as attacks, or too lenient by classifying most inputs as non-attacks. There are nonetheless a few promising results (some shown in the table), notably using the kernels "sigmoid" or "poly" on User A's dataset. Reducing the dimensionality of the data by using different aggregate variables, subsets of the variables used, or various dimensionality reduction methods (ex: Principal Component Analysis) may improve the strength of the algorithm. An increased amount of data for training and testing would also make the algorithm perform better.

Although the decision tree algorithm seems to provide the best overall results, it suffers from the fact that it is a supervised algorithm.

This means that to find the attacks, we need to know what they look like. For the goal of this research, which is detecting when an APT attack is happening, we recommend looking more deeply at one-class SVM and other algorithms that can be considered as "one-class". This is due to their ability to discover anomalies that were not in the training data. In any case, more data gathering and testing is required to improve the results of all the algorithms.

It is important to note that the results shown in this section are largely specific to the very limited data we were able to gather and are based on certain decisions we made in our algorithm implementations. Results obtained with further work may be (very) different due to the following factors:

- Different algorithm parameters
- Different machine setting (e.g., controlled VM, test environment, live production environment, etc.)
- Different hardware and software on the target machine (e.g., different operating systems, different version/performance of hardware, different installed applications/programs/features, etc.)
- Different models for user behaviour and APT behavior (e.g., business user, IT admin user, passive attacker, active attacker, etc.)
- Different data preparation (e.g., which Osquery tables/attributes are being considered, how the data is represented in a CSV, etc.)
- Different training/testing data (e.g., amount of data used, portions of the data split between training and testing, etc.).

Furthermore, in our research, only four machine learning algorithms were analyzed and used. As discussed in Section 4, there are many other machine learning algorithms [35, 36]. It is possible that other algorithms (including recent algorithms, such as random forest and neural network refinements for deep learning applications) could perform better than the ones chosen in our study. Exploring all the above variations with more extensive testing will allow us to confirm or modify the preliminary results that we have achieved with our current study.

Conclusion and Future Work

This study was intended as a proof-of-concept exercise to see if machine learning could be beneficial in detecting the presence of malware (in particular, APTs, which are specifically designed to be as stealthy as possible) on a computing platform.

This paper presented the details of how we approached the use of machine learning algorithms for APT detection. It highlighted how we generated our datasets, how the data was prepared for the algorithms, and the different types of machine learning algorithms that were analyzed; we also presented the results achieved from the use of these algorithms on our simulated data. Given the preliminary results that we have obtained, we feel that a machine learning technique focused on user behavior appears to be a viable approach for detecting the presence of an APT on a user workstation. We do recognize that our data set is very limited, and so any deductions about the effectiveness of specific algorithms are highly tentative at this point. While extensive testing with a much larger data set will of course increase the confidence in any particular results found, our primary conclusion from this study is that this approach to APT detection holds promise and is worth pursuing. Our approach (i.e., learning typical user behaviour in order to detect the presence of an APT, and focusing on computer activity rather than network traffic) differs from previous approaches to APT detection, but appears to be a potentially useful tool in the ongoing battle against APT malware. (Note that some recently published independent research [34] uses a very similar approach with good success, which helps to confirm the viability of this approach).

In terms of future work, many options are available. Studying the effect of all the factors mentioned at the end of Section 5 is certainly one important avenue. Other possibilities include assessing other techniques to gather data, generating bigger datasets over longer periods of time, testing on more users and user behaviour types, using different aggregate values on the data, and trying different dimensionality reduction techniques, to name a few. We feel that there is much opportunity for further work in this field.

References

1. Biswas SK (2018) Intrusion detection using machine learning: A comparison study. Int J Pure Appl Math 118: 101-114.
2. Çavuşoğlu Ü (2019) A new hybrid approach for intrusion detection using machine learning methods. Appl Intelligence 49: 2735-2761.
3. Ghafir I, Hammoudeh M, Prenosil V, Han L, Hegarty R, et al. (2018) Detection of advanced persistent threat using machine-learning correlation analysis. Future Generation Computer Systems 89: 349-359.
4. Jidiga GR, Sammulal P (2014) Anomaly detection using machine learning with a case study. In: IEEE International Conference on Advanced Communications, Control and Computing Technologies.
5. Moon D, Im H, Kim I, Park JH (2017) DTB-IDS: An intrusion detection system based on decision tree using behavior analysis for preventing APT attacks. J Supercomputing 73: 2881-2895.
6. Zamani M (2013) Machine learning techniques for intrusion detection, Yale University.
7. Bodstrom T, Hamalainen T (2019) A novel deep learning stack for APT detection. Appl Sci 9: 1055.
8. Cho DX, Nam HH (2019) A method of monitoring and detecting APT attacks based on unknown domains. Elsevier. Procedia Computer Science 150: 316-323.
9. Gavrilut D (2016) The value beyond the hype: Applying machine learning in APT detection, Bitdefender Business Insights Blog.
10. Meckl S, Tecuci G, Marcu D, Boicu M, Bin Zaman A (2017) Collaborative cognitive assistants for advanced persistent threat detection, cognitive assistance in government and public sector applications. AAAI Technical Report FS-17-02.
11. Palozza F (2018) Detecting malware/APT through automatic log analysis, Radware blog.
12. https://techbeacon.com/enterprise-it/counter-security-threats-machine-learning-real-time-data-analytics.
13. https://osquery.readthedocs.io/en/stable/.
14. https://github.com/endgameinc/RTA.
15. https://osquery.readthedocs.io/en/stable/deployment/logging/.
16. Nilsson NJ (1996) Introduction to machine learning: An early draft of a proposed textbook.
17. Venkateswaran B, Ciaburro G (2017) Neural Networks with R: Smart models using CNN, RNN, deep learning, and artificial intelligence principles. Packt Publishing, Birmingham, UK.
18. Körting T (2006) C4.5 algorithm and multivariate decision trees. Image Processing Division, National Institute for Space Research–INPE São José dos Campos–SP, Brazil.
19. Strobl C, Malley J, Tutz G (2009) An introduction to recursive partitioning: Rationale, application and characteristics of classification and regression trees, bagging and random forests. Psychol Methods 14: 323-348.
20. James G, Witten D, Hastie T, Tibshirani R (2015) An introduction to statistical learning. Springer, New York, USA.
21. https://www.brighthubpm.com/project-planning/106005-disadvantages-to-using-decision-trees/.

22. Quinlan J (1992) C4.5: Programs for machine learning. Morgan Kaufmann.
23. Witten IH, Frank E, Hall MA (2011) Data mining: Practical machine learning tools and techniques. (3rd edn). Morgan Kaufmann, San Francisco.
24. https://en.wikipedia.org/wiki/Unsupervised_learning.
25. https://www.coursera.org/lecture/machine-learning/unsupervised-learning-olRZo.
26. http://www.mit.edu/~9.54/fall14/slides/Class13.pdf.
27. https://github.com/serengil/tensorflow-101/blob/master/python/KMeansClustering.py.
28. Schölkopf B, Platt JC, Shawe-Taylor J, Smola AJ, Williamson RC (2001) Estimating the support of a high-dimensional distribution. Neural Computation pp: 1443-1471.
29. https://github.com/dpkravi/DecisionTreeClassifier.
30. https://github.com/tensorflow/models/tree/8cf24f468fa72e779e8e6ae26079fadbd34c3ab9/samples/core/get_started.
31. Gramfort A, Pedregosa F, Varoquaux G (2001) Scikit-learn: Machine Learning in Python.
32. https://www.researchgate.net/post/How_can_I_test_the_performance_of_a_clustering_algorithm, last accessed 2018/09/10.
33. Olaode AA, Naghdy G, Todd CA (2014) Unsupervised image classification by probabilistic latent semantic analysis for the annotation of images.
34. Lin TC, Guo CC, Yang CS (2019) Detecting advanced persistent threat malware using machine learning-Based threat hunting. European Conference on Cyber Warfare and Security, Reading pp: 760-768.
35. https://towardsdatascience.com/a-tour-of-the-top-10-algorithms-for-machine-learning-newbies-dde4edffae11.
36. https://www.kdnuggets.com/2017/10/top-10-machine-learning-algorithms-beginners.html.