Advanced Persistent Threat Detection using Data Provenance and Metric Learning

Khandakar Ashrafi Akbar, Yigong Wang, Gbadebo Ayoade, Yang Gao,
Anoop Singhal, Latifur Khan, Bhavani Thuraisingham, and Kangkook Jee

Abstract—**Advanced persistent threats (APT) have increased in recent times as a result of the rise in interest by nation-states and sophisticated corporations to obtain high-profile information. Typically, APT attacks are more challenging to detect since they leverage zero-day attacks and common benign tools. Furthermore, these attack campaigns are often prolonged to evade detection. We leverage an approach that uses a provenance graph to obtain execution traces of host nodes in order to detect anomalous behavior. By using the provenance graph, we extract features that are then used to train an online adaptive metric learning. Online metric learning is a deep learning method that learns a function to minimize the separation between similar classes and maximizes the separation between dis-similar instances. We compare our approach with baseline models and we show our method outperforms the baseline models by increasing detection accuracy on average by 11.3% and increases True positive rate (TPR) on average by 18.3%. We also show that our method outperforms several state-of-the-art models performances in comprehensive attack datasets in both binary and multi-class settings.**

I. Introduction

Advanced Persistent Threat (APT) [16] attacks are attacks that are usually conducted by nation state actors. These attacks target the victim's network in order to gain access to confidential information for espionage or compromise the network to destroy the victim's systems.

APT attacks are stealthy and are designed to avoid detection. Therefore, it is difficult to detect the APT attacks [12]. This is a significant challenge.

In a traditional machine learning based approach, we may train a machine learning model with class A and class B attacks, but a new attack which belongs to a novel class may suddenly appear. These novel attack classes can be termed as a zero-day attacks since the new class is not part of the training data but it may appear in the test class [25]. In this case, a traditional machine learning approach may not be able to detect the newly appeared class as a malicious attack class effectively. Due to the limitation of a traditional machine learning approach, we use a deep learning method based on Online Metric Learning (OML) [8] which learns a function to minimize the separation between similar classes (attack classes) and maximizes the separation between dissimilar instances (attack classes versus benign). Therefore, our method can recognize some of the zero-day attack more clearly from the benign instances in latent space and it performs effectively better than traditional machine learning approach. However, there is no guarantee our method will always detect all the zero-day attacks.

There has been much work regarding APT detection that attempt to address this challenge. Different methods have been proposed. For example, Milajedri et al. [30] propose the HOLMES system that gathers computer audit data and ranks the severity of the APT attack in real time. In HOLMES, an APT attack is classified based on the seven stages of the APT kill chain: 1) Initial Compromise, 2) Establish Foothold, 3) Escalate Privileges, 4) Internal Reconnaissance, 5) Move Laterally, 6) Maintain Presence, and 7) Complete Mission.

While much work has been done in the field to identify certain existing APT attacks, currently, a method to detect new APT attacks as they are being carried out does not exist. New APT attacks can only be detected after the attack had already occurred. Therefore, adversaries can still inflict significant damage when they conduct a zero-day APT attack. We propose a method to address this problem that generates an alert if a novel APT attack occurs. Our method could prevent damage from the novel APT attacks. More specifically, we train our model on a subset of the attacks, for example, shell-shock attack and test on other types of attack such as database command injection attack.

In addition, most APT attackers leverage traditional non-malicious tools to complete their attacks [10]. For example, an attacker may exploit a bash vulnerability to open a back-door on the victim system without installing any malicious software and can then perform lateral movement to access high target systems like the databases to steal data. By tricking the victim into running a bash script, the attacker gains access to the victim system. Detecting such attacks is challenging if the behaviour of the attack flow is not taken into consideration. In this case, a reverse-shell connection from the victim's machine to the attacker's machine takes place. A benign network flow will just involve connection to the database server from a regular network host. Our aim is to be able to detect when a non-malicious tool is used in a malicious attack flow. Our contributions include:

- We propose and implement a system that leverages provenance graphs derived from system events for detection of APT attacks
- We propose and implement a metric learning based approach to detect novel APT attacks by learning a latent space that effectively separates benign classes from attack classes.
- We show our method OML outperforms traditional machine learning classifiers in detection of novel APT attacks by an average of 12 % in detection accuracy.

The rest of the paper is organized as follows. Section II dis-

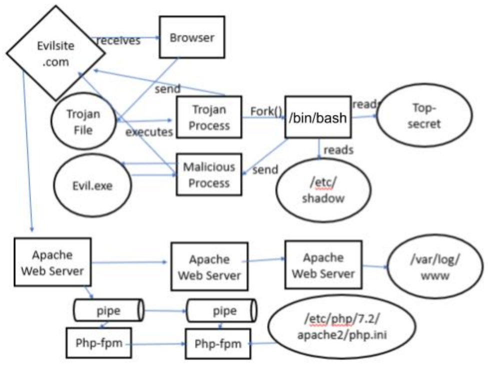

Fig. 1. Sample Camflow provenance graph data

cusses the challenges encountered in APT detection systems. Section II also discusses our approach and gives an overview of the system. Section III provides a detailed architecture of our system. Section IV shows our classification method using online metric learning. Section V presents a summary of our implementation and Section VII shows the evaluation of our approach. Finally, Section VIII provides the related work and Section X provides a conclusion and possible future work.

II. Challenges and Approach

When designing and implementing our method, we encountered some challenges that are listed below.

- **Limited training data for novel APT attacks**: Since data for novel APT attacks is limited or non-existent, it will be difficult for us to train our machine learning model for novel APT attacks due to the fact that neural network based model demands larger dataset to show better generalizing performance.
- **There is no signature for the zero-day APT attack**: When a novel APT attack occurs, it does not leave a specific signature that can be used to identify the attack. For novel APT attack identification, we need to analyze the victim machine's log files. Because the log files can be very large, it is challenging to filter out the attack activities from benign activities and detect the attacks in efficient ways in a distinguishable way.
- **Real-Time Detection**: It is also challenging to detect the APT attacks in real-time. Our system needs to detect attacks quickly and alert the system users to prevent the attack from causing damage.
- **Reduce false positives**: Events that are benign could be incorrectly identified as a novel APT attack. Unnecessary false alarm generation makes a system less dependable and effective. Our goal is to accurately detect APT attacks with a low false positive rate.

Figure 1 shows a simplified example of a provenance graph for an example APT attack. The provenance graph is usually much larger, but for readability, we have only depicted a small portion of the graph. In this APT attack, a user goes to evilsite.com and then downloads a Trojan horse. The Trojan executes malicious commands in the background via the malicious executable file evil.exe while also providing functionality that the user is aware of. For example, a Trojan could be a malicious calculator that appears to be a normal calculator but is secretly executing malicious commands in the background as the calculator application is running. In this example, the Trojan from evilsite.com is able to successfully read the /etc/shadow file and a file containing confidential information that is named 'Top-secret'. Therefore, the Trojan is able to successfully spy on the victim and gain unauthorized access to the victim's confidential information.

We generate provenance graphs that look like the graph depicted in Figure 1. However, the generated provenance graphs are much larger and show all system activities and processes instead of a small snippet. For APT attack detection, we detect vulnerabilities in the provenance graph. For this, we construct a provenance labeled graph (PLG). Here, PLG is an undirected graph that is defined as $G = (V, E)$ where V is the set of vertices which include `processes`, `tasks` and, `network socket` events, $E \subset V * V$ is the set of undirected edges which include interaction between system events such as `write` and `read` events. Given a set of training examples $T = (x_i, y_i)$ where $x_i \subset X$ is a graph, and $y_i \subset Y = +1$, 1 is a target label, the graph classification problem is to induce the mapping f: $X \Rightarrow Y$. For this, after PLG extraction, we need to convert it to a feature matrix vector by using node2vec [13], GraphSAGE [14], etc. We use node2vec at our end for each nodes of the PLG and take the average of all the nodes as the embedding for an entire PLG. Then, we apply our novel supervised learning technique to detect APT attacks in the stream of data including novel APT attacks. For novel zero day attacks, new data points which belong to different classes might not be well separable in the actual feature space, thus we use OML for defining the boundaries well for those novel attacks. More details on our experimentation are provided in sectionVII.

III. Architecture

Figure 2 illustrates our approach. Our approach leverages metric learning based detection to classify unknown APT attacks in real time. We use the provenance data to visualize the activity on the machine, and then we filter this provenance data to target attack traffic. Data *provenance* is a representation of the relationships among entities (data items), activities (changes applied to the entities), and agents (people or organizations that are associated with the activities and/or entities) [34]. We use this provenance to gather information about all of the activities occurring on the machine that could be representative of an attack tactic. In our system level context, data provenance is the story-line of kernel level objects interacting with each other. Kernel level objects can be: threads, files, sockets. The interaction among those objects are represented using provenance graph which shows the flow of information among the objects. This flow of information can be specific to certain attacks as which file is being accessed or modified, whether network level exfiltration of data is taking place can be identified from those provenance information. No matter how stealthy an APT attack is, the story-line of the provenance can depict the actual sequence of tasks being carried out in the background. The provenance graphs are extracted from the system using the CamFlow tool and the graph information (e.g., nodes and edges) is further used for generating embedding for training and testing purpose of the novel attacks

Figure 2 illustrates the steps of our proposed solution. First, we perform simulated advanced persistent attacks on the targeted victim machine. The data from these attacks is transformed into provenance data by CamFlow [34]. Second, we convert the provenance logs generated by CamFlow to a provenance graph using the CamQuery tool [35]. Third, once we have a provenance graph, we filter out sections of the graph to generate sub graphs that contain the events that are commands executed on the system. This is one of the strengths of CamFlow which allows it to configure the provenance capture in a way which can be used afterwards accordingly. We filter out extraneous noise that are common activities that occur on the machine regardless if the events are attacks or benign. Depending on certain configuration parameters, objects interacting in the system provenance can be filtered (captured or not captured). This comes handy in two different ways: first, it allows certain information to be dropped which eventually leads to keeping crucial information relating to system provenance to be kept. Secondly, it also creates visualizing, simple provenance graph which does provide significant information regarding benign or malicious instances. Compression of provenance graphs is also possible with CamFlow as it regards multiple similar events as only one. This might be another good option when it comes to suppressing provenance graph size for better visualization or keeping important information. On the other hand, attack instances like data destruction or exfiltration of data over the network might not be possible to detect with such compression. We leave the in need base provenance graph compression as future work. In this context, if more generally said, we extract the distinguishing information regarding benign and malicious traces and build the final representation. Fourth, we build a supervised model from these training graphs to detect novel APT attacks, existing APT attacks and benign events. The key point of detecting novel APT attacks comes from the fact that how well a model can identify newer forms of attacks if it has only seen few types of them so far. For feature extraction, we convert the graph into vectors using graph embedding by leveraging node2vec. An embedding vector is learned for each unique node in the graph. The vectors for the nodes in a graph are then aggregated together using the average function for each of the instances of the attack. Our feature extraction method is further discussed in Section III-B. We train our OML method to detect attacks based on the extracted features. More specifically, supervised learning is utilized to incrementally learn from the data. For supervised learning, we learn accurate models by leveraging attack and benign data, which are initially gleaned from benign data, synthetic attacks and existing APT attack traces, and later from live attack detection for detecting the novel type of APT attack. We capture the data on a machine. We capture provenance data using the tool, CamFlow [34]. The

```
"ABAAAAAAACAe9wIAAAAAAE7aeaI+200UAAAAAAAAAAAA=": {
    "cf:id": "3",
    "prov:type": "fifo",
    "cf:boot_id": 1,
    "cf:machine_id": cf:515081690,
    "cf:version": 0,
    "cf:date": "2019:08:13T15:50:53",
    "cf:jiffies": "4294902572",
    "cf:uid": 1000,
    "cf:gid": 1000,
    "cf:mode": "0x1180",
    "cf:ino": 51964,
    "prov:label": "[fifo] 0"
}
```

Listing 1. Sample node data for a provenance graph with FIFO type

```
 "cf:IAAAAAAAQIADAAAAAAAAAAAEAAA=": {
     "cf:id": "3",
     "prov:type": "write",
     "cf:boot_id": 1,
     "cf:machine_id": cf:515081690,
     "cf:date": "2019:08:13T15:50:53",
     "cf:jiffies": "4294902572",
     "prov:label": "write",
     "cf:allowed": "true",
     "prov:activity": "cf:AQAAAAAAAAECKNQEAAAAAAA",
     "prov:entity": "cf:ABAAAAAAAACDnNQEAAAAAAAA",
     "cf:offset": "0"
 }
```

Listing 2. Sample node data for a provenance graph with write type

provenance data is a record of all of the activities occurring on the machine, specifically interaction of application with the kernel and also interaction among different kernel objects. We then generate a provenance graph using CamQuery [35]. We adopt a similar strategy to HOLMES [30], but we extend this strategy further to detect novel APT attacks. We generate existing APT attack data by simulating the attacks on an Ubuntu Linux Virtual Machine. We simulate these attacks by performing various malicious activities on the machine such as downloading vulnerable software and running malicious programs on the machine. While we attack the machine, CamFlow [34] and CamQuery [35] capture the provenance data from the machine in the W3-PROV-JSON format and the provenance graph respectively.

A. Provenance Graph

The provenance graph is collected as a set of JSON files. Listing 1 shows an example of a node and write edge. The nodes contain the provenance type such as `fifo`, `file`, or `socket` [1]. It contains the machine id, boot id and unique node ids. The edges contain additional information such as provenance activity and entity nodes which are interacting together. In our case, we extract the interaction between the different provenance event types using CamQuery to form the nodes and edges in our graph. The Camquery extracts the node ids from the provenance graph and forms a pairwise list of the graph interaction nodes which we use as input to our feature extraction system.

B. Feature Extraction

Node2vec. We use Node2vec [13] to pre-process the provenance graph before classification. Node2vec is a semi-supervised algorithm for learning features from network graphs. Node2vec uses a similar approach to skip-gram by learning a vector that preserves the neighbourhood relationship of graph nodes similar to word2vec. By representing

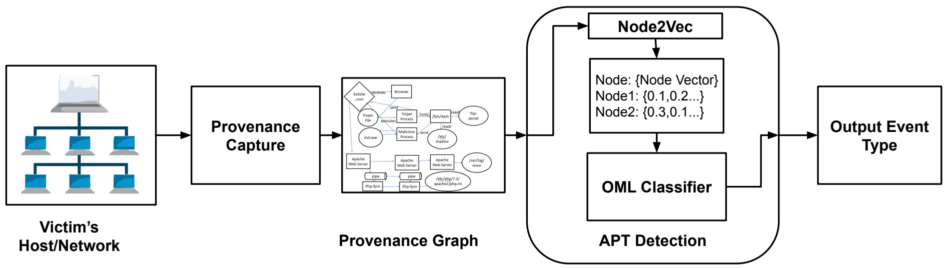

Fig. 2. Architecture for Provenance Generation and Advanced Persistent Threat Detection

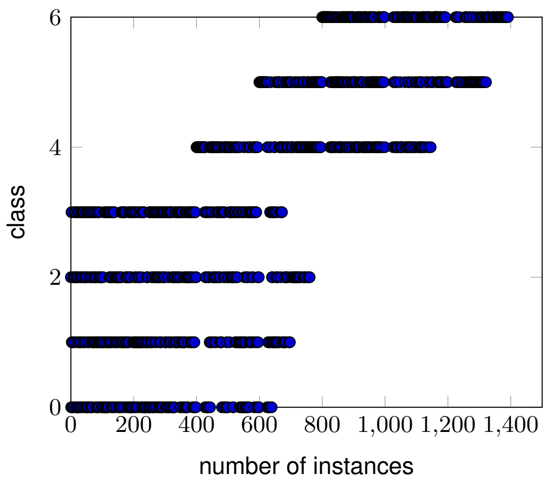

Fig. 3. Sample distribution of an evolving attack data stream

the graphs by performing a breadth-first search walk on the graph, a sequence of nodes can be generated similar to words in a document. We leverage this method to learn a vector for each node which is used for generating features for attack detection. For our approach, we extract the node id from the provenance graph. Since the graph is a representation of how each process interacts with other processes and tasks, we model the process task as nodes and the interactions as edges. The list of edges is then passed to the node2vec algorithm to generate an embedding vector for each unique node in the graph. The embedding vector for the nodes in the graph is then combined by finding the average vector representation which is used as a feature for the attack instance.

IV. Attack Detection

A. Emerging Attack Class Detection

Traditional machine learning approach is not effective at detecting novel attack classes. For example, a traditional machine learning model may be trained with instances that belong to class A and class B. However, a new attack class may emerge over time and the model may not be able to detect the new attack class effectively. In addition, in many real-world scenarios of APT attacks, instances of patterns associated with the attack type may change over time. Therefore, classifier performance is affected by the occurrences of instances from unknown or novel patterns.

Figure 3 shows a sample data set with emerging classes. For example, initial classes 1,2,3 occur in the first 600 data points while after that, new classes 4,5 and 6 emerges from points 600, 700 and 800. A traditional machine learning model trained on the first 600 data points will fail to detect attacks after this point due to change in the data point with different emerging classes.

For our novel class detection, we leverage online metric learning or distance based learning where malicious instance points can be well separated from benign class instances so that when novel attack classes emerge, we can detect it effectively. We provide more discussion in the next section.

Recently, we have proposed techniques [40] [15], [27] to detect novel class instances or identify emerging patterns along a data stream. One example is Stream-Based Novel Class Detection (SNOD) [23] which is a unique data stream classification technique that can function as a multi-class classifier and a novel class detector. Existing data stream classification techniques can only function as a multi-class classifier—i.e., they are unable to detect whether there is a new class in the stream. Conversely, techniques for novelty or anomaly detection [40] [15] can only detect novel classes in the stream but cannot function as a multi-class classifier. In other words, it designates one class as a normal class and assumes all other classes are novel. Therefore, it is essentially a one-class classifier. In contrast, SNOD is a multi-class classifier—i.e., it can recognize more than one existing class as well as recognize a novel class that does not belong to any existing classes. Furthermore, after a novel class appears and is detected by SNOD, that class is added to the list of existing classes of SNOD by updating the SNOD model so that future instances of that class can be correctly classified. For our novel class detection, we leverage online metric learning discussed in the next subsection.

B. Online Metric Learning

Online Adaptive Metric Learning (OML) [11] [4] [3] [6], [8], [17]–[19] is based on a deep learning architecture that transforms an instance feature from an original feature space to a latent feature space. By transforming to a latent feature space, the metric distance between dissimilar instances is increased and distance between similar classes is reduced. The work leverages methods which use *pairwise* and *triplet* constraints.

Our OML method learns a non-linear similarity metric unlike others which uses a pre-selected linear metric (e.g.,

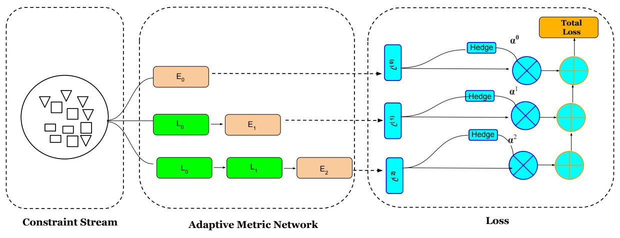

Fig. 4. OML network structure consist of L_i linear layer and Embedding layers E_i layer.

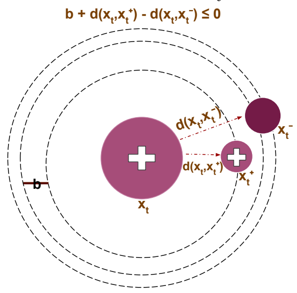

Fig. 5. Data instance before applying Online metric learning

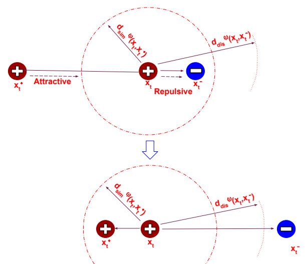

Fig. 6. Data instance after projection using OML

Mahalanobis distance [43]). Our OML method overcomes bias to a specific dataset by using an adaptive learning method. Our OML leverages neural networks where the hidden layer output is passed to an independent metric-embedding layer (MEL). The MELs then generate an n-dimensional embedding vector as output in different latent space.

1) Problem Setting: Let $S = \{(x_t, x_t^+, x_t^-)\}_{t=1}^T$ be a sequence of triplet constraints sampled from the data, where $\{x_t, x_t^+, x_t^-\} \in \mathcal{R}^d$, and x_t (anchor) is similar to x_t^+ (positive) but dissimilar to x_t^- (negative). The goal of online adaptive metric learning is to learn a model $F : \mathcal{R}^d \mapsto \mathcal{R}^{d'}$ such that $||F(x_t) - F(x_t^+)||_2 \ll ||F(x_t) - F(x_t^-)||_2$. Given these parameters, the objective is to learn a metric model with adaptive complexity while satisfying the constraints. The complexity of F must be adaptive so that its hypothesis space is automatically modified.

2) Overview: Consider a neural network with L hidden layers, where the input layer and the hidden layer are connected to an independent MEL. Each embedding layer learns a latent space where similar instances are clustered and dissimilar instances are separated.

Figure 4 illustrates our Artificial Neural Network (ANN) . Let $E_\ell \in \{E_0, E_1, E_2, \ldots, E_L\}$ denote the ℓ^{th} metric model in OML (i.e., the network branch from the input layer to the ℓ^{th} MEL). The simplest OML model E_0 represents a linear transformation from the input feature space to the metric embedding space. A weight $\alpha^{(\ell)} \in [0, 1]$ is assigned to E_ℓ, measuring its importance in OML.

For a triplet constraint (x_t, x_t^+, x_t^-) that arrives at time t, its metric embedding $f^{(\ell)}(x_t^*)$ generated by E_ℓ is

$$f^{(\ell)}(x_t^*) = h^{(\ell)}\Theta^{(\ell)} \tag{1}$$

where $h^{(\ell)} = \sigma(W^{(\ell)}h^{(\ell-1)})$, with $\ell \geq 1$, $\ell \in \mathbb{N}$, and $h^{(0)} = x_t^*$. Here x_t^* denotes any anchor (x_t), positive (x_t^+), or negative (x_t^-) instance, and $h^{(\ell)}$ represents the activation of the ℓ^{th} hidden layer. Learned metric embedding $f^{(\ell)}(x_t^*)$ is limited to a unit sphere (i.e., $||f^{(\ell)}(x_t^*)||_2 = 1$) to reduce the search space and accelerate training.

During the training phase, for every arriving triplet (x_t, x_t^+, x_t^-), we first retrieve the metric embedding $f^{(\ell)}(x_t^*)$ from the ℓ^{th} metric model using Eq. 1. A local loss $\mathcal{L}^{(\ell)}$ for E_ℓ is evaluated by calculating the similarity and dissimilarity errors based on $f^{(\ell)}(x_t^*)$. Thus, the overall loss introduced by this triplet is given by

$$\mathcal{L}_{overall}(x_t, x_t^+, x_t^-) = \sum_{\ell=0}^{L} \alpha^{(\ell)} \cdot \mathcal{L}^{(\ell)}(x_t, x_t^+, x_t^-) \tag{2}$$

Parameters $\Theta^{(\ell)}$, $\alpha^{(\ell)}$, and $W^{(\ell)}$ are learned during the online learning phase. The final optimization problem to solve in OML at time t is therefore:

$$\begin{aligned} \underset{\Theta^{(\ell)}, W^{(\ell)}, \alpha^{(\ell)}}{\text{minimize}} \quad & \mathcal{L}_{overall} \\ \text{subject to} \quad & ||f^{(\ell)}(x_t^*)||_2 = 1, \forall \ell = 0, \ldots, L. \end{aligned} \tag{3}$$

We evaluate the similarity and dissimilarity errors using an *adaptive-bound triplet loss* (ABTL) constraint [11] to estimate $\mathcal{L}^{(\ell)}$ and update parameters $\Theta^{(\ell)}$, $W^{(\ell)}$ and $\alpha^{(\ell)}$.

3) Why OML works: A typical machine learning algorithm like k-NN will misclassify the instances shown in Figure 5, since x_t^+ is closer to x_t^- and further from x_t. In our case, most APT attacks use non-malicious software to complete their attack activities making the attack events and traffic look non-malicious. To overcome this challenge, we use ABTL.

Figure 6 illustrates the main idea of ABTL. The objective is to have the distance $D_{update}^{(l)}(x_t, x_t^+)$ of two similar instances x_t and x_t^+ to be less than or equal to a similarity threshold $d_{sim}^{(l)}(x_t, x_t^+)$ so that the attractive loss $\mathcal{L}_{attr}^{(l)}(x_t, x_t^+)$ drops to zero; on the other hand, for two dissimilar instances x_t and x_t^-, we desire their distance $D_{update}^{(l)}(x_t, x_t^-)$ to be greater than or equal to a dissimilarity threshold $d_{dis}^{(l)}(x_t, x_t^-)$, thereby reducing the repulsive loss $\mathcal{L}_{rep}^{(l)}(x_t, x_t^-)$ to zero.

4) Adaptive-Bound Triplet Loss: Here $y_t \in \{+1, -1\}$ denotes whether x_t is similar $(+1)$ or dissimilar (-1) to x'_t and $b \in \mathcal{R}$ is a user-specified fixed margin. While triplet loss simultaneously learns both similarity and dissimilarity relations, the pairwise loss can only focus on one of the relations at a time, which leads to a poor metric quality. In addition, triplet loss requires a proper margin be specified. In addition, the selected margin is highly dependent on the data and it requires extensive domain knowledge. Our aim is to automatically learn the margin for our triplet-loss constraint irrespective of the available data.

With $\tau \in (0, \frac{2}{3})$, by optimizing the proposed adaptive-bound triplet loss, different classes are separated in the metric embedding space. Let $D(c_1, c_2)$ denote the minimal distance between classes c_1 and c_2, i.e., the distance between two closest instances from c_1 and c_2 respectively. Consider an arbitrary quadruple $(x_1, x_2, x_3, x_4) \in \mathcal{Q}$ where $\{x_1, x_2\} \in c_1$, $\{x_3, x_4\} \in c_2$, and $\mathcal{Q}$ is the set of all possible quadruples generated from class c_1 and c_2. Suppose (x_2, x_3) is the closest dissimilar pair among all possible dissimilar pairs that can be extracted from (x_1, x_2, x_3, x_4). We first prove that the lower bound of $D(c_1, c_2)$ is given by $\min_{(x_1,x_2,x_3,x_4)\in\mathcal{Q}} D^{(l)}(x_1, x_4) - D^{(l)}(x_1, x_2) - D^{(l)}(x_3, x_4)$.

$$\begin{aligned} D^{(l)}(x_1, x_4) &\leq D^{(l)}(x_1, x_2) + D^{(l)}(x_2, x_4) \\ &\leq D^{(l)}(x_1, x_2) + D^{(l)}(x_2, x_3) + D^{(l)}(x_3, x_4) \end{aligned} \tag{4}$$

$$\begin{aligned} D(c_1, c_2) &= \min_{(x_1,x_2,x_3,x_4)\in\mathcal{Q}} D^{(l)}(x_2, x_3) \\ &\geq \min_{(x_1,x_2,x_3,x_4)\in\mathcal{Q}} D^{(l)}(x_1, x_4) - D^{(l)}(x_1, x_2) \\ &-D^{(l)}(x_3, x_4) \end{aligned} \tag{5}$$

By optimizing the adaptive-bound triplet loss, the following constraints are satisfied.

$$\begin{cases} D^{(l)}(x_1, x_2) \leq d^{(l)}_{sim}(x_1, x_2) \leq \mathcal{T}^{(l)}_{sim} \\ D^{(l)}(x_3, x_4) \leq d^{(l)}_{sim}(x_3, x_4) \leq \mathcal{T}^{(l)}_{sim} \\ D^{(l)}(x_1, x_4) \geq d^{(l)}_{dis}(x_1, x_4) \geq \mathcal{T}^{(l)}_{dis} \end{cases} \tag{6}$$

$$\begin{aligned} D(c_1, c_2) &\geq \min_{(x_1,x_2,x_3,x_4)\in\mathcal{Q}} D^{(l)}(x_1, x_4) - D^{(l)}(x_1, x_2) \\ &-D^{(l)}(x_3, x_4) \\ &\geq \mathcal{T}^{(l)}_{dis} - 2\mathcal{T}^{(l)}_{sim} \\ &= 2 - \tau - 2\tau \\ &= 2 - 3\tau \end{aligned} \tag{7}$$

if $\tau \in (0, \frac{2}{3})$, we have $3\tau < 2$. Therefore,

$$D(c_1, c_2) \geq 2 - 3\tau > 0 \tag{8}$$

Equation 8 indicates that the minimal distance between class c_1 and c_2 is always positive so that these two classes are separated.

Note that our whole proof is solely based on the triangle inequality, which is correct as long as the triangle inequality holds. As L_2-norm is utilized to measure the distance, the metric space learned by our framework is indeed a normed vector space. The triangle inequality is a natural property in this space.

Moreover, our framework does not require the learned metric space to be convex, i.e., for any two distinct instances x and y in the metric space, there exists a third instance z in this space lying between x and y $(d(x, z)+d(z, y) = d(x, y))$. Whether or not the equality strictly holds does not affect the correctness of our proof, since it only considers the upper-bound of a distance. In our case, even the non-convex metric space is a valid solution, as the triangle inequality still holds in this space.

V. Implementation

We developed an implementation of our system for the 64-bit Ubuntu Linux operating system with 128 GB space of RAM. Our system consists of the data generation and the machine learning detection module. For the data generation, we used bash scripts which consist of 100 lines of code. We leveraged the CamFlow tool [34] installed on VirtualBox using Vagrant. We modified the CamQuery module [35] to extract the ID of the provenance graph nodes and to generate edges of process interactions.

For the machine learning detection module (e.g. kNN, Decision Tree, SVM), we leveraged scikit-learn [37] for our baseline evaluation and the OML components were implemented with approximately 500 lines of Python code using the PyTorch [36] library.

Model Parameters. For our experiments, Support Vector Machine (SVM) uses Radial Basis Function (RBF) kernel with Cost = 1.3×10^5 and gamma is set to 1.9×10^{-6}. OML uses a Rectified Linear activation Unit (ReLU) network with embedding $n = 200$, number of hidden layers $L = 5$, $k = 1$, learning rate = 0.3, learning rate decay = 1×10^{-4}, and uses ADAM (A method for stochastic optimizer) optimizer.

VI. Tactics Driven Synthetic Data Generation and Data Collection

A. Attack Description

Table I shows the contents of our dataset. We name our dataset as the 'synthetic dataset'. This dataset consists of different APT activities such as exfiltration of data, illegal login and access, opening of a reverse shell for command and control access, and illegal network scanning using **nmap** for the discovery of services on victim's network. All the classes were used for both benign and malicious scenario generation. For example, command line injection attacks were considered malicious when some third party injected some commands and executed them. Benign scenarios were mimicked with normal command execution flow which consists of usual executed commands within the system from the user. Table II shows properties of the provenance graph of our synthetic dataset based on the trace of a single execution of an attack instance. A sample provenance graph contains an average of 10 332 edges for data ex-filtration attack events labeled as class 1, while the average in-degree is 48 and the average out-degree is 153 for the same class. For our 'synthetic dataset', we collect the provenance graph which contains information such as provenance type and task type as discussed in section III-A. Benign instances include web browsing activities on

TABLE I
SUMMARY OF ATTACK WORKLOAD

Class	Description	Software
1	Data exfiltration	scp
2	Illegal network scanning	ping
3	Illegal network mapping	nmap
4	Reverse shell for Command and Control	nc
5	Data Leakage Attack	Deployed Through Mimicking Real-World Scenario
6	Remote Webservice Penetration (Shellshock) and Password Cracking	Deployed Through Mimicking Real-World Scenario
7	Command Line injection Attack	Deployed Through Mimicking Real-World Scenario

Attacks emulated from MITRE APT collection.

TABLE II
PROVENANCE GRAPH PROPERTIES OF OUR SYNTHETIC DATASET

Class	Avg # of out-deg	Avg # of in-deg	Avg # of edges
1	153	48	10 337
2	545	13	9796
3	325	5	9781
4	511	96	10 236
5	561	922	9783
6	2,084	1,437	10 638
7	300	52	10 106

benign websites, normal login activity with Secure Shell (SSH) and benign connection from a client machine to a database server. We collected 100 traces for each attack type. Our dataset consists of activities derived from previously known APT attack campaign steps. We collect multiple traces of attack sequences to gather diversified training data and test on single instance of the attack sequence. We generated 7 benign cases and 7 attack cases since each attack type has a corresponding benign activity variation. For several attacks, we collect data against both benign and malicious case and for some other ones, we only collect malicious samples. When client end is compromised by an attacker, the behavior from client end should be pretty different from the norm as this particular session will be conducted by the attacker. Based on this point, we extract benign and malicious instances depending on when the client end is compromised or not.

B. Attack Generation

Table I shows that the Data Leakage Attack & Remote Webservice Penetration (Shellshock) and Password Cracking attack are deployed through mimicking real-world scenarios. A server end user, a client end user, and a user playing the role of an attacker are required to mimic the attack. In the data leakage attack scenario, a server consists of a database and owns it and has given access to its client to access data from the database through queries. In the server end, postgresql database is used for such service. If a database is dumped from the server end into any sql file or any other file format, a client can reconstruct the database or restore the database by having that file from the server end. In the attack scenario, a third user who is essentially the attacker makes the client download some malicious program (e.g., spear phishing e-mail (from outside) to open a backdoor or sending the software to the client end by any other means) and run it in the client's computer. With the execution of that specific software, the attacker gets access to the client's system (control over client's computer) and can now communicate with the server as if the client itself was communicating. The attacker can now make queries to the server and can even restore the database in its own end but it will make the behavior seem like a client as it will be doing these tasks from the client's system. Thus bulk data can be compromised in

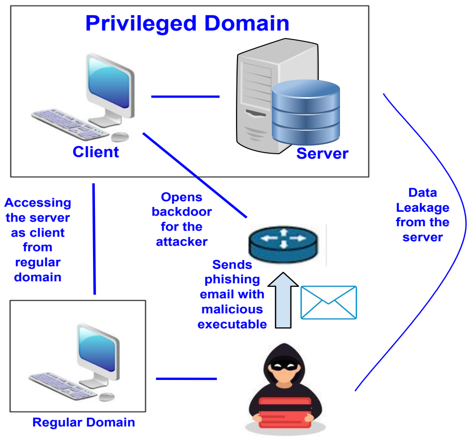
Privileged Domain
Client
Server
Accessing the server as client from regular domain
Opens backdoor for the attacker
Data Leakage from the server
Sends phishing email with malicious executable
Regular Domain

Fig. 7. Data Leakage Attack Scenario

this way and private data can get exfiltrated from the server end as the usage behavior is shadowed by victimizing client end.

The Remote Webservice Penetration (Shellshock) and Password Cracking attack is deployed when the client's system is somewhat hacked or deliberately used to attack the server end. In this scenario, an attacker exploits a remote shellshock vulnerability by sending crafted input to the CGI (Common Gateway Interface) service implemented using bash. Thus the attacker leverages the shellshock vulnerability as a backdoor to run malicious commands such as executing a password cracking tool.

C. Data Collection

1) Data Collection for Data Leakage Attack: Figure 7 illustrates the steps required to complete this attack. For the data leakage attack, data was captured in both the server and client end. For the attack scenario, the attacker sends an email with some malicious executable attached with it to the client's mail. The client downloads the attachment, executes the executable in the machine, and thus opens the backdoor for the attacker without any knowledge of it being taken place. The attacker now mimics the behavior of the client but from its own end and performs all the database queries and restoration. When the client executes the backdoor program, it goes straight to a vulnerable state and from that time Camflow provenance data is collected both at the server and client end. The benign set of data corresponds to when the server and client are communicating regularly. No other connection is established in the meantime. The server receives database requests from the client in a regular manner in benign scenario data collection. The reason for collecting benign and malicious data both at server and client end is to differentiate between usage behavior of usual client and attacker who communicates with the server from the client end only but can mimic different command execution or query submission pattern. This can substantively help to identify malicious query perform from benign ones.

2) Data Collection for Password Cracking Attack: We carried out a password cracking attack using the John the ripper tool which implements a dictionary password attack. For the password cracking attack, the normal behavior or benign scenario simply involves a client connecting with the server using a curl command with no malicious payload. Benign data is collected while these normal operations take place using a Camflow provenance graph. For the attack scenario, the attacker exploits the shellshock vulnerability and then downloads a password cracker and executes a shell script for the backdoor creation. The attacker then executes commands for password collection and cracking password from the victim's end. While these steps take place, data was collected at the victim's end using a Camflow provenance graph.

3) Data Collection for Command Line Injection Attack: For the command line injection attack, we only collected data for the attack scenario. The victim host (e.g., say embedded / IoT device that runs Linux) runs Mediaplayer (Kodi client), and it exports a remote control Application Program Interface (API) as a web service. One of its input sanitizations has an error that fails to filter invalid input from the outside, in turn allowing attackers to inject arbitrary commands blended in one of its requests. This vulnerability allows any arbitrary input to invade the remote control application as it does not know which input request is from actual user and which one is from outside world. This attack is inspired by the Jeep-Cherokee attack case where the attacker from remote gains control over the vehicle. In parallel to the above steps, the data collection was done at the victim host using a Camflow provenance graph. If the input behavior is significantly different from a regular user, the malicious attack scenario should be distinctive enough than that of the benign scenario.

D. Stage Based Comprehensive Data Collection

In the work [33], a comprehensive dataset is collected mimicking normal traffic seen on read-world cyber systems. We use this dataset directly from their github repository to analyze our method, OML's performance.

VII. Evaluation

A. Results

Table III and IV show the results of our experiment on our synthetic dataset. Table VII and VIII show the results of our experiment on the DAPT 2020 dataset which we describe in a brief later in this section. The specification about these two different dataset can be described as: our synthetic data is tactic based data of advanced persistent threat whereas the DAPT 2020 is stage based data. For this part of our experiment which involves our synthetic dataset, we trained incrementally on the attack classes and tested on all the remaining classes which includes the benign class instances. This is the key idea behind identifying or detecting novel class attacks adopted in our experimentation phase. Please note that the training data and the test data consists of the benign data instances. With this approach, we can determine if our algorithm can detect unseen novel attack classes. First, we train on all the benign classes and a single APT attack class and test on all the benign and APT attack classes. Second, we train on all benign classes and two APT attack classes and tested on all attack classes. Third, we train on all benign classes and three APT attack classes and tested on all attack classes. Lastly, we train on all benign classes and all the APT attack classes and tested on all attack classes. We measured our performance by using the following metrics: **Accuracy**, **True Positive Rate**, **False Positive Rate**, and **F2 Score**.

In the experiments, we measured the true positive rate (*tpr*), where true positive represents the number of correctly classified seen and novel APT attacks classes; false positive rate (*fpr*), where false positive represents the number of incorrectly classified seen and novel APT attacks; and F_2 score of the classifier, where the F_2 score is interpreted as the weighted average of the precision and recall. The F2 score ranges between the values 100 and 0 where 100 is the best value and 0 is the worst value. The results show our

TABLE III
RESULTS BASED ON NOVEL APT ATTACK DETECTION

Metric	Classes #	OML	KNN	SVM_RBF	Decision_Tree
Accuracy	1	**54.76**	48.02	40.03	47.95
	2	**64.99**	56.67	40.03	66.64
	3	**73.18**	72.66	50.2	60.7
	4	**75.96**	72.85	61.36	66.91
	5	**86.06**	79.52	68.03	73.51
	6	**92.07**	86.2	71.0	80.52
	7	**98.08**	92.87	77.48	87.45
	8	**98.**	96.37	91.88	93.66
	9	99	99	91.88	99
Metric	**Classes #**	**OML**	**KNN**	**SVM_RBF**	**Decision_Tree**
F2	1	**28.92**	16.12	0	15.99
	2	**47.13**	32.44	0	49.94
	3	**60.72**	59.86	22.02	39.67
	4	**65.13**	60.18	42.62	50.38
	5	**80.5**	70.69	53.91	61.25
	6	**89.14**	80.7	58.77	72.2
	7	**97.43**	90.25	68.98	82.53
	8	**97.46**	95.09	90.09	91.36
	9	99	1	90.09	99

TABLE IV
RESULTS BASED ON NOVEL APT ATTACK DETECTION

Metric	Classes #	OML	KNN	SVM_RBF	Decision_Tree
FPR	1	0	0	0	0
	2	0	0	0	0
	3	0	0	0.0231	0
	4	0	0	0.0281	0
	5	0	0	2.81	0
	6	0	0	2.81	0
	7	0	0	2.81	0
	8	0	0	2.81	0
	9	0	0	2.81	0
Metric	**Classes #**	**OML**	**KNN**	**SVM_RBF**	**Decision_Tree**
TPR	1	**24.56**	13.33	0	13.22
	2	**41.63**	27.75	0	44.38
	3	**55.29**	54.41	18.5	34.47
	4	**59.91**	54.74	37.44	44.82
	5	**76.76**	65.86	48.57	55.84
	6	**86.78**	76.98	53.52	67.51
	7	**96.81**	88.11	64.32	79.07
	8	**97.96**	93.94	88.33	89.43
	9	99	1	88.33	99

TABLE V
EXECUTION TRAINING AND TESTING TIME FOR OML

No of Training Instances	Train time(s)	Test time(s)
700	55.8	1.94
800	55.9	1.95
900	55.4	1.96
1000	55.5	1.96
1100	54.7	1.91
1200	55.9	1.97
1300	58.9	1.98
1400	59.5	2.00
1500	59.6	2.08

approach performs better than traditional machine learning based classifiers such as k-NN, SVM and Decision tree. The accuracy of detection of novel attacks with our approach is 86 % compared to 79 % for k-NN, 68 % for SVM and 73 % for Decision Tree when we train on only five attack classes. Similarly, the TPR is 76 % for OML compared to 65 % for k-NN and 55 % for Decision Tree when we train on only five attack classes. The $F2$ is 80.5 % for OML compared to 70.69 % for k-NN and 61 % for Decision Tree when we train on only five attack classes. The $accuracy$ increases to 98 % for OML and 96 % for k-NN, 91 % for SVM and 93 % for Decision Tree when we train on 7 attack classes. Our approach improves classification performance on average by **6.8 %** for accuracy, **10.19 %** for TPR and **11.4 %** for $F2$ when compared with k-NN method, while the performance improves by **17 %** for accuracy, **28 %** for TPR and **26 %** for $F2$ when compared with SVM. Likewise, our performance improves by **10 %** for accuracy, **17 %** for TPR and **16 %** for $F2$ when we compared our method with Decision Tree. Our approach detected novel APT attacks with higher accuracy than k-NN, SVM or Decision Tree even with limited training on a subset of the APT attack classes. This is possible because OML can learn to minimize the distance in feature space for similar instances and maximize the distance for dissimilar instances as shown in Theorem IV-B4.

B. Execution time for OML

Table V shows the summary of the execution time required to train our OML approach and then perform inference on the test data. The execution time for training the OML algorithm is approximately 60 seconds while the testing or inference execution time is approximately 2 seconds. As discussed in subsection IV-B, OML learns a latent embedding space vector to satisfy a constraints. As a result of this algorithm, OML training time is higher than at inference. As we can see from the execution timing information, the testing time is fast as a result of just using the learned embedding vector to classify the test data. In addition, we only show the number of instances used for the training since the number of testing instances is always constant as discussed in section VII.

C. OML Performance in Comprehensive Datasets

In the work [33], a comprehensive dataset has been introduced as part of a contribution to Advanced Persistent Threat identification and response research which is named as DAPT 2020. Compared to UNB-15 and CICDS dataset, this particular dataset named DAPT 2020 has data covering all the crucial stages of Advanced Persistent Threat campaign.

TABLE VI
DAPT DATASET STATISTICS

APT Stage	Total Instances	Malicious Instances	Benign Instances
Reconnaissance	29254	4405	24849
Foothold	17486	8632	8854
Lateral Movement	4051	4	4047
Data Exfiltration	2617	6	2611

TABLE VII
BINARY CLASSIFICATION RESULTS

Metric	OML	SVM	SAE	SVM (Modified)
Accuracy	0.984	0.7196	0.59	0.4292
Macro-Precision	0.978	0.3711	0.4033	0.4253
Macro-Recall	0.9796	0.4797	0.4183	0.4004
Macro-F1	0.9788	0.4185	0.4098	0.3886

Myneni et al. have conducted experiments on all these three datasets using SAE (Stacked Auto-Encoder), SAE-LSTM (Stacked Auto-Encoder-Long Short Term Memory) and 1-SVM (Support Vector Machine). The best performing model among them was SAE for all three datasets. We apply our method OML in this comprehensive dataset and our method outperforms the state-of-the-art in terms of binary classification. This DAPT 2020 dataset does not contain provenance graphs, thus it does not require node2vec for generating feature embedding. The data embedding of this dataset are directly used in our OML model for classification purposes. The results for the binary classification are reported in table VII. The binary classification in the work with DAPT 2020 dataset [33] is done between benign data and considering all stages of APT as attack data and so is our experiment performed. We also report multi-class classification results in table VIII considering all the stages of APT as individual classes in the DAPT 2020 dataset. The SVM result is reported using the RBF kernel function and the SVM (modified) result is reported using the polynomial kernel. We can see the effect of data-imbalance from the numerics for macro-precision and macro-recall (data-imbalance is prominent in table VI). To answer for the numerics related to precision, recall, and F1 score for multi-class classification, we also provide the data statistics in table VI which prominently indicates to the data-imbalance issue of this DAPT 2020 dataset. Due to this data-imbalance, it is imperative to have a proper separation of data points in feature space, thus, our use of OML model is justified for this dataset too.

VIII. RELATED WORK

Traditional graph-based approaches like [38] coined an automated technique to generate attack graphs using symbolic model checking algorithms [29]. Later on [41] proposed a robust, flexible graph based approach for network vulnerability analysis which allows attacks from both outside and inside the networks. This method suffers from the scalability issue when state increase occurs. [2] proposed scalable attack graphs which do not require the idea of backtracking from the attacker side relying on the idea of monotonicity.
HOLMES [30] uses a set of manually generated rules to describe different APT information flows from an attack provenance graph. Our approach uses a deep learning based approach to learn the attack patterns without the need for

TABLE VIII
MULTI-CLASS CLASSIFICATION RESULTS

Metric	OML
Accuracy	0.9528
Macro-Precision	0.5544
Macro-Recall	0.5682
Macro-F1	0.561

manual generation of rules. [9] propose a method for APT detection based on unusual or unknown domains that a malicious user could visit while conducting an APT attack. However, the work only detects APT attacks based on unusual domains visited but not APT attacks of other types. Our work detects other types of APT attacks rather than just those APT attacks that have unknown domains. Our work involves the implementation of an APT detection tool rather than proposing an APT detection method.

There are additional machine learning and deep learning based works that propose various methods for APT detection. For example, [5] proposes a novel deep learning stack for APT detection. In this method, they propose using deep learning for outlier detection to detect APT attacks and novel APT attacks. However, this method has not been implemented in practice and this method does not utilize provenance graphs generated from system logs. Cho et al. [31] proposed an idea based on decision tree to prevent APT attacks. Ghafir et al. [12] uses machine learning based correlation analysis, but the approach does not focus on detection of novel attacks. In [21] authors have proposed STREAMSPOT , a clustering based anomaly detection method in heterogenous graph. Anomaly detection is a leading problem in security, finance, medicine and so on. The authors have considered heterogenous graphs with nodes and edges, where the inputs are streams of typed edges and destinations are also typed. Through this method anomalous graphs those are prominently different from others are identified. Tian et al. [42] proposed a deep learning representation for graph clustering. Ma et al. [20] learns a latent representation of social data features by using matrix factorization. [32] uses a deep auto-encoder to learn features for novel class detection. Our work differs from these works by using provenance graphs, which take into consideration the information flow for the attack events.

[24] uses host and network data for anomaly detection. Methods such as [26] [28] [40] [15], [27] [23] detect malware in an evolving data streams. [7], [22] extract features from sequences of system call events based on co-occurrences of such events. In addition, Shu et al. uses long call sequences to detect attacks hidden within long execution paths [39].

IX. LIMITATION AND FUTURE WORK

In this work, we have not focused on multi-stage attack detection, however, our framework is able to detect a single stage in an APT attack as shown in the experiments. However, our method may not always detect all the zero-day attacks. In future work, we plan to address the challenge of multi-stage attack detection. Data collection is still challenging in cyber-attack space as the amount of standardized training data is limited. In future work, we opt to collect more data to train our machine learning models. In addition, we will perform experiments with real life attack data. Our work currently focuses on attacks based on Linux operating system, we plan to address attacks in other operating systems e.g. Windows operating systems platform. We will also scratch the surface to see if some other platforms or computational domains (e.g Internet of Things or Aerial Vehicles) are threatened by the existence of Advanced Persistent Threats.

A further extension will be about how to dynamically segment data (system interaction traces) to efficiently identify each stages of the attack. Bulk data makes it more challenging to analyze data in real time and crucial to detect zero-day attacks promptly. Moreover, a more robust system would require evident information about a single instance of some attack phase (a single technique or a whole tactic) to identify it correctly. If system traces are overlapped across different technique or tactic based instances, then efficient detection of Advanced Persistent Threat can become somewhat random and sometimes not even possible. As an ablation study, we intend to provide with the ablation study that overlapped or cropped system trace can be misleading about any specific attack trace (techniques or tactics based) and thus can provide with random instances to train the models on.

As part of correlating different stages of APT attack campaign, we intend to find out patterns in successfully completing APT attack stages so that we can predict which stage the attacker is going to target next. As same set of strategies are sometimes used as part of different tactics or stages of APT attack, it is important to analyze the flow of information so that the exact stage which has been achieved by the attacker recently can be identified. Efficiently correlating different stages of APT attack and predicting future behavior will be one of our major future tasks. We will work on correlating events in real-time manner so that we do not have to overwhelm the stored data provenance in the system to link system events taking place long back in timeline. We also intent to address the mapping of lower level system data which helps to understand or identify different stages of advanced persistent threat or even simple intrusion detection to high level description of such attacks tactics and techniques. In recent research, this mapping is almost wholly done manually which requires intense human intervention. We want to, at least, semi-automate this process. An another problem in streaming data setting is to determine the boundary within which an attack or benign instance is to be searched for. As data burst can be an immense problem when we are dealing with bulk system data to generate data provenance, a fine line to generate the graphs might be required to particularly identify different stages of APT. As our future work, we will be determining dynamically shifting window technique in such issue handling.

X. CONCLUSION

Detecting APT attacks is a challenging task based on the approach deployed by malicious actors. The different tactics and techniques are evolving at a great pace which makes it imperative to look for methods for identifying novel attacks. In this work, we focus on a machine learning based approach to detect novel APT attacks. We leverage provenance graphs for the collection of event data from host systems. We apply OML: a novel machine learning technique for detecting APT attacks. Our results show our approach has a higher detection accuracy compared to traditional machine learning techniques in detecting novel class attacks. In our future work, we will explore more novel detection machine learning methods to detect novel APT attacks. We will also address the most difficult challenges such as: detecting zero day attacks from streaming system provenance, correlating different stages of APT attack campaign in real-time while attaining such high performing methods and thus make a generalizable and robust system.

Acknowledgments

The research reported herein was supported in part by ONR award N00014-17-1-2995; NSA award H98230-15- 1-0271; AFOSR award FA9550-14-1-0173; NSF FAIN awards DGE-1931800, OAC-1828467, and DGE-1723602; NSF awards DMS-1737978 and MRI-1828467; an IBM faculty award (Research); and an HP grant. Any opinions, recommendations, or conclusions expressed are those of the authors and not necessarily of the aforementioned supporters.

References

[1] "Camflow - vertices supported by camflow," https://github.com/CamFlow/camflow-dev/blob/master/docs/VERTICES.md.

[2] P. Ammann, D. Wijesekera, and S. Kaushik, "Scalable, graph-based network vulnerability analysis," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*. ACM, 2002, pp. 217–224.

[3] F. Araujo, G. Ayoade, K. Al-Naami, Y. Gao, K. W. Hamlen, and L. Khan, "Improving intrusion detectors by crook-sourcing," in *Proceedings of the 35th Annual Computer Security Applications Conference*, San Juan, Puerto Rico, December 2019, pp. 245–256.

[4] G. Ayoade, F. Araujo, K. Al-Naami, A. M. Mustafa, Y. Gao, K. W. Hamlen, and L. Khan, "Automating cyberdeception evaluation with deep learning," in *Proceedings of the 53rd Hawaii International Conference on System Sciences (HICSS)*, Grand Wailea, Maui, January 2020.

[5] T. Bodström, T.; Hämäläinen, "A novel deep learning stack for apt detection." *Appl. Sci.*, vol. 9, no. 1055, 2019.

[6] C. Breen, L. Khan, and A. Ponnusamy, "Image classification using neural networks and ontologies," in *Proceedings. 13th International Workshop on Database and Expert Systems Applications*. IEEE, 2002, pp. 98–102.

[7] J. B. Cabrera, L. Lewis, and R. K. Mehra, "Detection and classification of intrusions and faults using sequences of system calls," *ACM SIGMOD Record*, vol. 30, no. 4, pp. 25–34, 2001.

[8] G. Chechik, V. Sharma, U. Shalit, and S. Bengio, "Large scale online learning of image similarity through ranking," *Journal of Machine Learning Research*, vol. 11, pp. 1109–1135, 2010.

[9] D. X. Cho and H. H. Namb, "A method of monitoring and detecting apt attacks based on unknown domains," *Procedia Computer Science*, vol. 150, pp. 316–323, 2019.

[10] T. Enache, "Shellshock vulnerability." [Online]. Available: https://owasp.org/www-pdf-archive/Shellshock_-_Tudor_Enache.pdf

[11] Y. Gao, Y.-F. Li, S. Chandra, L. Khan, and B. Thuraisingham, "Towards self-adaptive metric learning on the fly," in *The World Wide Web Conference*. ACM, 2019, pp. 503–513.

[12] I. Ghafir, M. Hammoudeh, V. Prenosil, L. Han, R. Hegarty, K. Rabie, and F. J. Aparicio-Navarro, "Detection of advanced persistent threat using machine-learning correlation analysis," *Future Generation Computer Systems*, vol. 89, pp. 349–359, 2018.

[13] A. Grover and J. Leskovec, "Node2vec: Scalable feature learning for networks," in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: ACM, 2016, pp. 855–864. [Online]. Available: http://doi.acm.org/10.1145/2939672.2939754

[14] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 1024–1034. [Online]. Available: http://papers.nips.cc/paper/6703-inductive-representation-learning-on-large-graphs.pdf

[15] A. Haque, L. Khan, and M. Baron, "Sand: Semi-supervised adaptive novel class detection and classification over data stream," in *AAAI*, 2016.

[16] E. M. Hutchins, M. J. Cloppert, and R. M. Amin, "Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains," *Leading Issues in Information Warfare & Security Research*, vol. 1, p. 80, 2011.

[17] P. Jain, B. Kulis, I. S. Dhillon, and K. Grauman, "Online metric learning and fast similarity search," in *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008*, 2008, pp. 761–768.

[18] R. Jin, S. Wang, and Y. Zhou, "Regularized distance metric learning: Theory and algorithm," in *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009. Proceedings of a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada.*, 2009, pp. 862–870.

[19] W. Li, Y. Gao, L. Wang, L. Zhou, J. Huo, and Y. Shi, "OPML: A one-pass closed-form solution for online metric learning," *Pattern Recognition*, vol. 75, pp. 302–314, 2018.

[20] H. Ma, H. Yang, M. R. Lyu, and I. King, "Sorec: Social recommendation using probabilistic matrix factorization," in *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, ser. CIKM '08, 2008, pp. 931–940.

[21] E. Manzoor, S. M. Milajerdi, and L. Akoglu, "Fast memory-efficient anomaly detection in streaming heterogeneous graphs," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 1035–1044.

[22] C. Marceau, "Characterizing the behavior of a program using multiple-length n-grams," in *Proceedings of the New Security Paradigms Workshop*, 2001, pp. 101–110.

[23] M. Masud, J. Gao, L. Khan, J. Han, and B. M. Thuraisingham, "Classification and novel class detection in concept-drifting data streams under time constraints," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 6, pp. 859–874, June 2011.

[24] M. Masud, L. Khan, and B. Thuraisingham, *Data Mining Tools for Malware Detection*. CRC Press, 2011.

[25] M. M. Masud, T. M. Al-Khateeb, K. W. Hamlen, J. Gao, L. Khan, J. Han, and B. Thuraisingham, "Cloud-based malware detection for evolving data streams," *ACM Trans. Manage. Inf. Syst.*, vol. 2, no. 3, Oct. 2008.

[26] ——, "Cloud-based malware detection for evolving data streams," *ACM Trans. Manage. Inf. Syst.*, vol. 2, no. 3, pp. 16:1–16:27, Oct. 2008. [Online]. Available: http://doi.acm.org/10.1145/2019618.2019622

[27] M. M. Masud, Q. Chen, L. Khan, C. C. Aggarwal, J. Gao, J. Han, A. Srivastava, and N. C. Oza, "Classification and adaptive novel class detection of feature-evolving data streams," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 7, pp. 1484–1497, 2012.

[28] M. M. Masud, J. Gao, L. Khan, J. Han, and B. Thuraisingham, "Classification and novel class detection in data streams with active mining," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2010, pp. 311–324.

[29] K. L. McMillan, "Symbolic model checking," in *Symbolic Model Checking*. Springer, 1993, pp. 25–60.

[30] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan, "Holmes: Real-time apt detection through correlation of suspicious information flows," in *2019 IEEE Symposium on Security and Privacy*, vol. 1. IEEE, 2019, pp. 447–462.

[31] D. Moon, H. Im, I. Kim, and J. H. Park, "Dtb-ids: an intrusion detection system based on decision tree using behavior analysis for preventing apt attacks," *The Journal of supercomputing*, vol. 73, no. 7, pp. 2881–2895, 2017.

[32] A. M. Mustafa, G. Ayoade, K. Al-Naami, L. Khan, K. W. Hamlen, B. Thuraisingham, and F. Araujo, "Unsupervised deep embedding for novel class detection over data stream," in *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 2017, pp. 1830–1839.

[33] S. Myneni, A. Chowdhary, A. Sabur, S. Sengupta, G. Agrawal, D. Huang, and M. Kang, "Dapt 2020 -constructing a benchmark dataset for advanced persistent threats," 07 2020.

[34] T. Pasquier, X. Han, M. Goldstein, T. Moyer, D. Eyers, M. Seltzer, and J. Bacon, "Practical whole-system provenance capture," in *Proceedings of the 2017 Symposium on Cloud Computing*, ser. SoCC '17. New York, NY, USA: ACM, 2017, pp. 405–418. [Online]. Available: http://doi.acm.org/10.1145/3127479.3129249

[35] T. Pasquier, X. Han, T. Moyer, A. Bates, O. Hermant, D. Eyers, J. Bacon, and M. Seltzer, "Runtime analysis of whole-system provenance," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: ACM, 2018, pp. 1601–1616. [Online]. Available: http://doi.acm.org/10.1145/3243734.3243776

[36] PyTorch, "Open source deep learning platform," https://pytorch.org/, 2019.

[37] scikit, "Python machine learning library," https://scikit-learn.org/stable/, 2019.

[38] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated generation and analysis of attack graphs," in *Proceedings 2002 IEEE Symposium on Security and Privacy*. IEEE, 2002, pp. 273–284.

[39] X. Shu, D. Yao, and N. Ramakrishnan, "Unearthing stealthy program attacks buried in extremely long execution paths," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2015, pp. 401–413.

[40] M. Solaimani, M. Iftekhar, L. Khan, and B. Thuraisingham, "Statistical technique for online anomaly detection using spark over heterogeneous data from multi-source vmware performance data," in *2014 IEEE International Conference on Big Data (Big Data)*, Oct 2014, pp. 1086–1094.

[41] L. P. Swiler and C. Phillips, "A graph-based system for network-vulnerability analysis," Sandia National Labs., Albuquerque, NM (United States), Tech. Rep., 1998.

[42] F. Tian, B. Gao, Q. Cui, E. Chen, and T.-Y. Liu, "Learning deep representations for graph clustering," in *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[43] S. Xiang, F. Nie, and C. Zhang, "Learning a mahalanobis distance metric for data clustering and classification," *Pattern recognition*, vol. 41, no. 12, pp. 3600–3612, 2008.

Khandakar Ashrafi Akbar is a second year PhD student at University of Texas at Dallas. She received her B.Sc in Computer Science and Engineering from Bangladesh University of Engineering and Technology. She has two years of work experience where she worked as a software engineer worked at Reve Systems and as a lecturer at Northern University Bangladesh. She is co-supervised by Dr. Latifur Khan and Dr. Bhavani Thuraisingham. Her research interest broadly falls in the area of data mining, cyber security, artificial intelligence, and human-computer interaction.

Yigong Wang Yigong Wang is currently a Ph.D student, and a member of the Big Data Analytics and Management Lab in the CS department at University of Texas at Dallas. His research interests are in the area of deep learning, computer vision, stream, and cyber-security. His recent projects include implementation of deep learning on stream data and analysis of program profiles for app behavior analysis. Yigong is supervised by Dr. Latifur Khan in his Ph.D.

Gbadebo Ayoade Gbadebo Ayoade began his career as a software engineer at one of the top electronic financial transaction companies serving the largest banks in Africa. In 2013, he decided to pursue further studies to further his knowledge in the area of reliable and secure computing. In 2014, he graduated from The University of Texas at Dallas with an MS in Computer Science. In 2015, he got a full scholarship to pursue his PhD program in Computer Science at The University of Texas at Dallas under the supervision of Dr. Latifur Khan and Dr. Kevin W. Hamlen. During his PhD studies, he was a member of the Big Data Analytics and Management Lab as a Research Assistant. His research interest ranges from big data systems, cyber security and applied machine learning. He also enjoys developing software for production systems.

Yang Gao Yang Gao received his Bachelor of Science degree in physics from the University of Science and Technology of China in 2012 and his Master of Science degree in physics from The University of Texas at Dallas in 2015. He was privileged to join The University of Texas at Dallas to pursue his Ph.D. degree in computer science. Yang was a member of the Big Data Analytics and Management Lab and worked as a research assistant under the supervision of Prof. Latifur Khan. His research interests are metric learning, representation learning, and transfer learning.

Anoop Singhal Dr. Anoop Singhal, is currently a Senior Computer Scientist and a Program Manager in the Computer Security Division at the National Institute of Standards and Technology (NIST) in Gaithersburg, MD. He has several years of research experience at NIST, George Mason University and ATT Bell Labs. His research interests are in system security, active cyber defense, network forensics, cloud computing security and machine learning systems. He is a member of ACM, senior member of the IEEE and he has co-authored over 50 technical papers in leading conferences and journals. He has taught several graduate level courses in Computer Science as an adjunct faculty and given talks at RSA, IEEE and ACM conferences. He has two patents in the area of attack graphs and he has also co-edited a book on Secure Cloud Computing and Network Security Metrics.

Latifur Khan is currently a full Professor (tenured) in the Computer Science department at the University of Texas at Dallas where he has been teaching and conducting research since September 2000. He received his Ph.D. and M.S. degrees in Computer Science from the University of Southern California in August of 2000, and December of 1996 respectively. He is an ACM Distinguished Scientist and a Senior Member of IEEE. He has received prestigious awards including the IEEE Technical Achievement Award for Intelligence and Security Informatics. Dr. Khan has published over 170 papers in 40 journals, in peer reviewed conference proceedings, and in three books. His research areas cover data mining, big data management, and analytics.

Bhavani Thuraisingham Bhavani Thuraisingham is the Founders Chair Professor of Computer Science and the Executive Director of the Cyber Security Research and Education Institute (CSI) at The University of Texas at Dallas. She is an elected Fellow of ACM, IEEE, the AAAS, the British Computer Society, and the SPDS (Society for Design and Process Science). She is the reciepient of the IEEE CS 1997 Technical Achievement Award, the 2010 Research Leadership Award presented by IEEE ITS and IEEE SMC and the 2010 ACM SIGSAC Outstanding Contributions Award.

Kangkook Jee Kangkook Jee is an Assistant Professor at Computer Science department at University of Texas at Dallas. He was previously a researcher in the Computer Security Department at NEC Labs America in Princeton, NJ. He received his Ph.D. in Computer Science from Columbia University. His advisor was Prof. Angelos D. Keromytis. Prior to his graduate study, he spent five years in industry working as a system engineer at IBM, Korea. His research interests include: Systems Security, Operating Systems, Information Flow Tracking, Program Analysis, Binary Analysis and Instrumentation, Malware Analysis.