Evolving Advanced Persistent Threat Detection using Provenance Graph and Metric Learning

Gbadebo Ayoade*, Khandakar Ashrafi Akbar*, Pracheta Sahoo *
, Yang Gao *, Anmol Agarwal*,
Kangkook Jee *, and Latifur Khan*
*Department of Computer Science
University of Texas at Dallas, Richardson, Texas 75080
Email: (gbadebo.ayoade,KhandakarAshrafi.Akbar,pracheta.sahoo
yxg122530,anmol.agarwal,Kangkook.Jee,lkhan)@utdallas.edu

Anoop Singhal†
†National Institute of Standards and Technology
Email: anoop.singhal@nist.gov

***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 %.**

I. Introduction

Advanced Persistent Threat (APT) [13] 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.

One example of an APT attack is the Sykipot attacks. In the Sykipot attacks, attackers targeted U.S. and U.K. organizations such as defense contractors, computer hardware manufacturers, and government departments. The attackers used spear-phishing to send emails that contained malicious attachments or links. If a user were to click on a malicious link or open a malicious attachment, then this could harm the organization's system. APT attacks are stealthy and are designed to avoid detection. Therefore, it is difficult to detect the APT attacks [10]. 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 [16]. 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) [7] 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. [17] 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. For example, an attacker may exploit a bash vulnerability to open a backdoor 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

[1]To appear in *IEEE Conference on Communications and Network Security (CNS) 2020*, Avignon, France, Date: 29 June - 1 July, 2020

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

II. Challenges

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.
- **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.
- **Detection in real time**: 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. Our goal is to accurately detect APT attacks with a low false positive rate.

III. Approach

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.

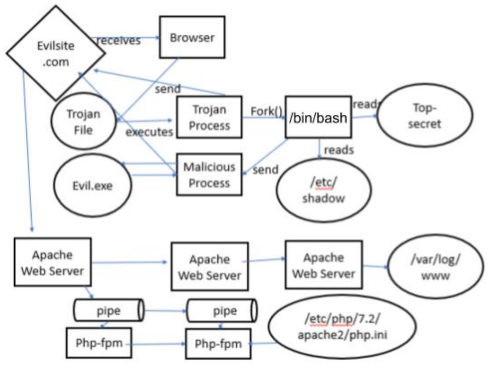

Fig. 1. Sample Camflow provenance graph data

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 [11], GraphSAGE [12], etc. Then, we apply our novel supervised learning technique to detect APT attacks in the stream of data including novel APT attacks.

IV. 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) [21]. We use this provenance to gather information about all of the activities occurring on the machine that could be representative of an attack tactic.

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 [21]. Second, we convert the provenance logs generated by CamFlow to a provenance graph using the CamQuery tool [22]. 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. We filter out extraneous noise that are common activities that occur on the machine regardless if the events are attacks or benign. Fourth, we build a supervised model from these training graphs to detect novel APT attacks, existing APT attacks and benign events.

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 IV-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 [21]. The provenance data is a record of all of the activities occurring on the machine. We then generate a provenance graph using CamQuery [22]. We adopt a similar strategy to HOLMES [17], 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 [21] and CamQuery [22] 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.

```
"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:ino": 51964,
    "prov:label": "[fifo] 0" }
```

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

B. Feature Extraction

Node2vec. We use Node2vec [11] 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 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.

V. 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.

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.

B. Online Metric Learning

Online Adaptive Metric Learning (OAML) [9] [5] [4] 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,

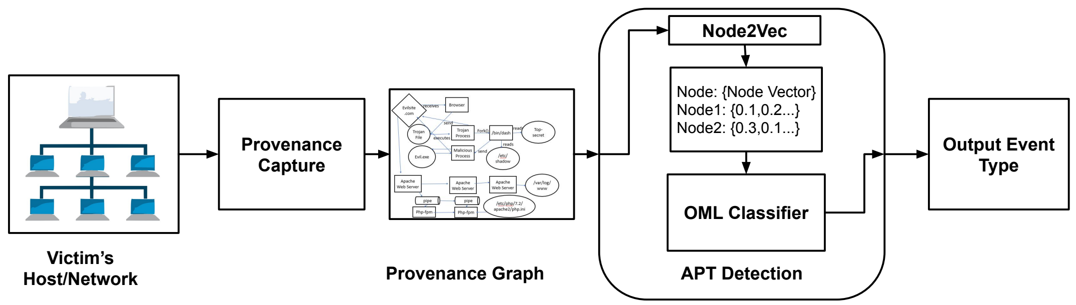

Fig. 2. Architecture

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 OAML method learns a non-linear similarity metric unlike others which uses a pre-selected linear metric (e.g., Mahalanobis distance [26]). Our OAML method overcomes bias to a specific dataset by using an adaptive learning method. Our OAML 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 3 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 OAML (i.e., the network branch from the input layer to the ℓ^{th} MEL). The simplest OAML 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 OAML.

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.

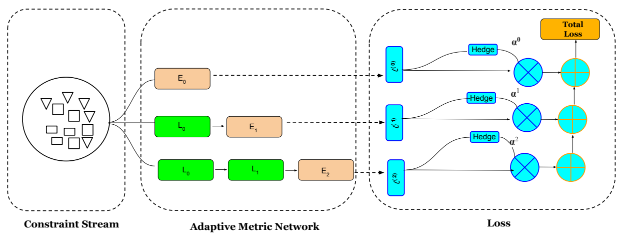

Fig. 3. OAML network structure consist of L_i linear layer and Embedding layers E_i layer.

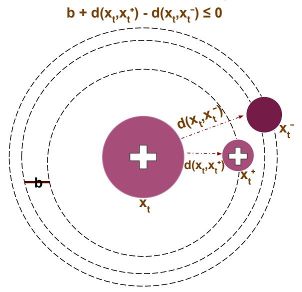

Fig. 4. Data instance before applying Online metric learning

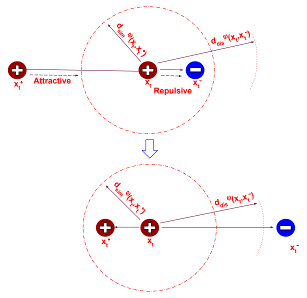

Fig. 5. Data instance after projection using OML

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

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

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

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

We evaluate the similarity and dissimilarity errors using an *adaptive-bound triplet loss* (ABTL) constraint [9] 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 4, 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 5 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 $\min_{(x_1,x_2,x_3,x_4)\in\mathcal{Q}} D^{(l)}(x_1, x_4) - D^{(l)}(x_1, x_2) - D^{(l)}(x_3, x_4)$.

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

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

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

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

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

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

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

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.

VI. 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 [21] installed on VirtualBox using Vagrant. We modified the CamQuery module [22] to extract the ID of the provenance graph nodes and to generate edges of process interactions.

For the machine learning detection module, we leveraged scikit-learn for our baseline evaluation and the OAML components were implemented with approximately 500 lines of Python code using the PyTorch 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}. OAML 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.

VII. Datasets

A. Attack Generation & Data Collection

Table I show the contents of our dataset. The 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. Table II shows properties of the provenance graph 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 dataset, we collect the provenance graph which contains information such as provenance type and task type as discussed in section IV-A. Benign instances include web browsing activities on 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.

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 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 software, the attacker gets access to the client's system (control over client's computer) and can now communicate with the server. 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. In a nutshell, the attacker makes the victim end download some executable and execute it to open the backdoor for itself to the server end so that it can leak the data to the client end while having full access to the server as a client (a client acts as a privileged domain to the attacker). The attacker can leak the data through external storage or network transfer after the attack has been deployed from the client's computer. Benign scenarios for this case were created with normal operational and communication flow between the server and the client. These operations include normal database queries made by the client to the server, access to the database file and retrieving of the database by the client itself.

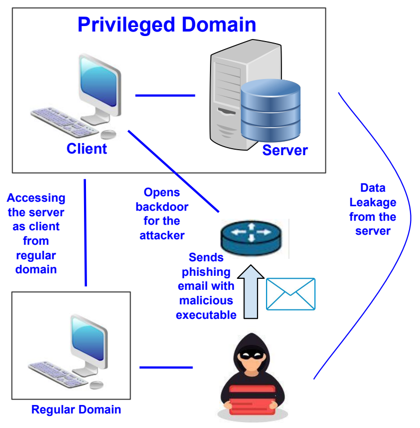
Privileged Domain
Client
Server
Accessing the server as client from regular domain
Opens backdoor for the attacker
Data Leakage from the server
Sends phishing email with malicious executable
Regular Domain

Fig. 6. Data Leakage Attack

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 6 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 it, and thus opens the backdoor for the attacker without any knowledge of it. 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

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

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

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.

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 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 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.

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

VIII. EVALUATION

A. Results

Table III and IV show the results of our experiment. For our experiment, we trained incrementally on the attack classes and tested on all the remaining classes which includes the benign class instances. 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

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

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**, **TPR**, **FPR**, and **F2**.

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 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 V-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 V-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 VIII.

IX. RELATED WORK

Our work specifically focuses on APT attack detection. An earlier approach was proposed by [23]. In their work they coined an automated technique to generate attack graphs using symbolic model checking algorithms. Later on [24] 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. [3] proposed scalable attack graphs which do not require the idea of backtracking from the attacker side relying on the idea of monotonicity. [18] proposed an idea based on Decision Tree to prevent APT attacks. Recent work on APT attack detection include Milajerdi et al. [17] called HOLMES. HOLMES 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 manual generation of rules. Ghafir et al. [10] uses machine learning based correlation analysis MLAPT, but the approach does not focus on detection of novel attacks.

There are additional works that propose various methods for APT detection. For example, [6] 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. Cho et al. [8] propose a method for APT detection based on unusual or unknown domains that a malicious user could visit while conducting an APT attack. In [14] authors have proposed STREAMSPOT, a clustering based anomaly detection method in heterogeneous graph. Through this method, anomalous graphs, that are prominently different from others are identified. Tian et al. [25] proposed a deep learning representation for graph clustering.

Various methods for detecting attacks have been proposed. [15] uses host and network data for anomaly detection. Methods such as [20], [2] detect malware in evolving data streams. [19] 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.

X. 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 a 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.

XI. Conclusion

Detecting APT attacks is a challenging task based on the approach deployed by malicious actors. In this work, we focus on a machine learning based approach to detect 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 our future work, we will explore more novel detection machine learning methods to detect novel APT attacks.

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] T. Al-Khateeb, M. M. Masud, L. Khan, C. Aggarwal, J. Han, and B. Thuraisingham, "Stream classification with recurring and novel class detection using class-based ensemble," in *Proceedings of the 2012 IEEE 12th International Conference on Data Mining*, ser. ICDM '12, USA, 2012, p. 31–40.

[3] 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.

[4] 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.

[5] 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.

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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

[12] 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

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] P. Parveen, J. Evans, B. Thuraisingham, K. W. Hamlen, and L. Khan, "Insider threat detection using stream mining and graph mining," in *2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing*, 2011, pp. 1102–1110.

[21] 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

[22] 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

[23] 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.

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

[25] 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.

[26] 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.