Identifying Tactics of Advanced Persistent Threats with Limited Attack Traces

Khandakar Ashrafi Akbar[1], Yigong Wang[1], Md Shihabul Islam[1], Anoop Singhal[2], Latifur Khan[1], and Bhavani Thuraisingham[1]

[1] The University of Texas at Dallas, 800 West Campbell Road, Richardson, Texas 75080, USA

[2] National Institute of Standards and Technology, 100 Bureau Drive, Gaithersburg, MD 20899, USA

Abstract. The cyberworld being threatened by continuous imposters needs the development of intelligent methods for identifying threats while keeping in mind all the constraints that can be encountered. Advanced Persistent Threats (APT) have become an important national issue as they secretly steal information over a long period of time. Depending on the objective, adversaries use different tactics throughout the APT campaign to compromise the systems. Therefore, this kind of attack needs immediate attention as such attack tactics are hard to detect for being interleaved with benign activities. Moreover, existing solutions to detect APT attacks are computationally expensive, since keeping track of every system behavior is both costly and challenging. In addition, because of the data imbalance issue that appears due to few malicious events compared to the innumerable benign events in the system, the performance of the existing detection models is affected. In this work, we propose novel machine learning (ML) approaches to classify such attack tactics. More specifically, we convert APT traces into a graph, generate nodes, and eventually graph embeddings, and classify using ML. For ML, we use proposed advanced approaches to address class imbalance issues and compare our approaches with other baseline models and show the effectiveness of our approaches.

Keywords: Advanced Persistent Threat · Online Metric Learning · Data Imbalance

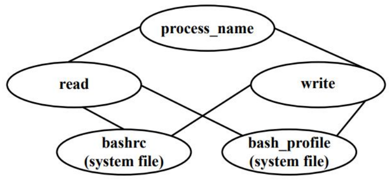
1 Introduction

Advanced Persistent Threats (APT) are specifically well-known for their masquerading characteristics and damaging power. In the recent era of cyber warfare, it has become a powerful process to systematically damage or conduct espionage against competitors. Without proper knowledge of what is going on behind the scenes, e.g. in system-level interactions and operations, these threats can be disguised for a long time and can remain undetected even after they have completed their target tasks. Due to the heavy use of computational systems, it is challenging to keep track of each and every system behavior. Moreover, identifying these types of threatening phenomena is also computationally expensive. Different stages of APTs have been defined [23] over the span of research in recent times. Different attack vectors are used at the beginning of the penetration of the system by the adversarial entities. These different stages can be achieved by using different tactics. Such tactics have corresponding techniques which execute the final task for the completion of the goals. For instance, writing malicious commands to the *bash_profile* and *.bashrc* system files is a technique that falls under the 'Persistence' tactic from the MITRE [1] framework. Figure 1a illustrates the overall system entity interaction for this technique that shows any process can use system calls 'read' and 'write' to access *bash_profile* and *.bashrc* system files and eventually write malicious commands to them.

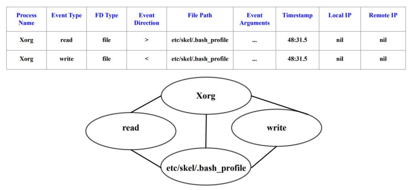
With time, adversarial behavior evolves and so are their ways of compromising systems. Newly adopted tactics and techniques are continuously chased by security professionals to make intrusion detection frameworks and other security ensuring platforms more robust. Nevertheless, APTs are stealthy in nature and can easily avoid detection [23]. To detect such types of attacks, it is imperative to obtain low-level system traces to identify suspicious activities in a system. Take for example the Sykipot attack, in which the attackers targeted U.S. and U.K. organizations [34]. The attackers used the spear-phishing technique to send emails that contained malicious contents within. If such malicious content is clicked, a system can be harmed in ways that the malicious content establishes a foothold in the system which needs to be immediately tracked down after such email content has been received or executed. Therefore, these attack events based on different techniques, such as writing malicious commands in bash_profile or bashrc (system configuration files in Ubuntu), need to be inspected to classify or identify such phenomena as an attack tactic.

Adversaries use both benign and malicious tools to complete their target tasks. In both of these use cases, it is important to capture the system behavior or interaction information in order to detect such event. Collecting logs is a well-adopted and old technique for ensuring system performance. Nowadays, it is popular to keep track of system events which are also useful for security purposes [27]. That is why in recent times, data provenance based detection of attack campaigns have become very popular [13, 17, 20, 22, 29]. Using system level raw data which are collected during system operation or interaction with the system, a meaningful depiction of the whole scenario is represented through a graph, which is known as a provenance graph [13]. But due to the fact that the bulk of provenance data is heavy to handle, it is necessary to process the data in a meaningful and efficient way. Moreover, attacks semantics are not platform invariant, thus significant domain knowledge is necessary to detect attacks on a specific platform (e.g. Linux, Windows, macOS). In our approach, we leverage domain-specific attack knowledge to simultaneously reduce noise from logs and create multiple versions of the same attack instance trace to incorporate limited attack traces in the learning process.

In this paper, we propose machine learning-based approaches to classify adversary tactics from provenance graphs. Generally, few tactics do not possess enough representative techniques which only yield a minority number of provenance graphs for those tactics. As a result, a class imbalance issue may emerge



(a) Sample Provenance Graph for a Technique under 'Persistence' Tactic



Process Name	Event Type	FD Type	Event Direction	File Path	Event Arguments	Timestamp	Local IP	Remote IP
Xorg	read	file	>	etc/skel/.bash_profile	...	48:31.5	nil	nil
Xorg	write	file	<	etc/skel/.bash_profile	...	48:31.5	nil	nil

(b) Sample Log and Generated Provenance Graph

Fig. 1: Provenance Graphs

which weakens the performance of the learning model. We propose novel machine learning algorithms to efficaciously alleviate the class imbalance problem. To classify tactic, we first convert APT traces into a graph, and then using GraphSAGE [12], we generate node and eventually graph embeddings which will be treated as instances to train the machine learning models. Finally, we perform the prediction using advanced approaches to address the class imbalance issue. We compare our approaches with other baseline models and show the effectiveness of our approach. To the best of our knowledge, this is the first known approach to address class imbalance issue in identifying tactics of APT and to limit the attack traces for computational ease well before a provenance graph is generated.

To summarize, in this paper, we propose the following contributions.

- We identify different tactics using advanced machine learning models, e.g. SetConv [10] which is defined in section 4.2 and OAML [9] which is defined in section 4.1, to address class imbalance issue, and show the superiority of these models over the baseline models.
- We address the problem of overlapping common noisy system interaction behavior in different attack traces and process those traces to build a robust model.
- We propose to incorporate domain knowledge in attack trace processing which eventually produces different versions of attack traces.

The paper is organized as follows. Section 2 presents some background on MITRE framework and GraphSAGE. Section 3 talks about the data collection in details, and how the collected data is converted to graph. Section 4 provides the details of our machine learning models. Section 5 explains the experimental setup in detail and discusses the results. Finally, section 6 presents related work and concludes our work.

2 Background

2.1 MITRE ATT&CK Framework

In recent years, several frameworks have been proposed to evaluate the defense of existing systems against adversarial cyberattacks. Some such notable frameworks for attack detection and trace collection are Red Team Automation (RTA) [8], Metta, CALDERA [4], and Atomic Red Team [2]. MITRE ATT&CK framework provides a knowledge base of threat models and practices by investigating real-world scenarios of different adversarial behavior. It provides an adversary tactic and technique taxonomy that could be utilized to assess a system's threat detection, response mechanism, and risk calculation and hence improve the effectiveness of cybersecurity solutions.

The framework contains a comprehensive matrix of tactics and techniques for multiple platforms. Some of the tactics are Defense Evasion, Discovery, Persistence, Privilege Escalation, Reconnaissance, and so on. Each of the tactics contains a list of techniques that refer to the method or type of attack. For instance, the Privilege Escalation tactic includes techniques such as Access Token Manipulation, Hijack Execution Flow, Process Injection, and the like. Each technique contains information on how to deploy the attack such as required platforms, required permissions, defenses bypassed, etc., and how to detect attacks from processes and mitigate the attacks with different strategies. The attack payloads are defined as TTP (Techniques, Tactics, & Procedures) and are numerically followed by mostly four digits to point to a specific type of technique. For example, T1046 refers to the attack or vulnerability technique named Network Service Scanning from the Discovery tactic class.

Moreover, each technique includes sub-techniques that explain different methods to implement that particular technique. For example, T1136 (i.e., Create Account) technique falls under the tactic class 'Persistence'. To maintain access to the victim system, adversaries may create accounts in the local system, within a domain, or in the cloud. Therefore, the technique T1136 contains sub-techniques Local Account, Domain Account, and Cloud Account. Thus all the sub-techniques under this TTP generate instances for the tactic 'Persistence' since they correspond to the same tactic class that the technique belongs to.

The Atomic Red Team framework supplies a collection of scripts for detection tests of certain attack techniques mapped to the MITRE ATT&CK Framework. It consists of the techniques and sub-techniques for the TTPs to be executed in Linux, Windows, or the macOS platform. In this paper, we incorporate the definition of the TTPs from the MITRE ATT&CK framework and emulate the attacks using Atomic Red Team's framework.

2.2 Graph Embedding and GraphSAGE

Graph embedding has become a well-known approach as graphs are adopted as a very popular data structure in different types of problem domains, such as protein-protein interaction systems, supply chains, knowledge graphs, social-network, and so on. A graph is a pair G = (V, E) where V is a set whose elements are called vertices, and E is a set of paired vertices, whose elements are called edges. A neural network architecture that operates on graph structures is known as the Graph Neural Network (GNN). Generally, GNNs are used for node classification purposes; that is, every node of the graph is assigned some features and a label and the GNN predicts the labels of the unseen nodes by leveraging the seen node information within a neighborhood. GNNs propose to address the challenging forecasting problem including both spatial and temporal dependencies and its recent advances greatly boost the ability of modeling data from the non-Euclidean space such as the graph structures [19]. Inspired by the mechanism of message passing in a graph, a variant of GNN known as Graph Convolutional Networks (GCN) is proposed that aggregates up to the n-hop spatial neighborhood to each location in the data.

In recent years, many algorithms for learning node representations of graphs have been proposed such as DeepWalk [30] and GraphSAGE [12]. GraphSAGE leverages graph structure to produce node embeddings in an inductive way. That is, GraphSAGE trains aggregator functions that aggregate a node's neighborhood rather than directly embedding vectors of nodes individually. This strategy makes it easy to generalize to

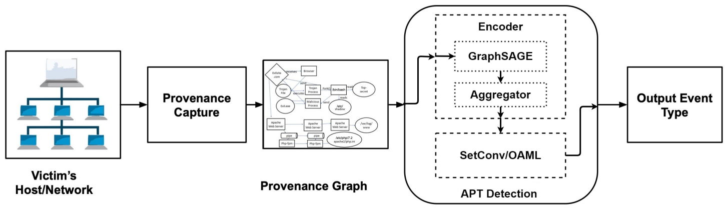


Fig. 2: System Architecture

unseen nodes given their features and neighborhood and avoids re-training the model for new nodes. The n-hop is used for controlling the coverage range of aggregation functions from selected neighbors. GraphSAGE is most appropriate for dynamic graphs, where the structure of the graph is always growing or changing. In this paper, we leverage GraphSAGE to generate node embeddings of the provenance graphs, which is then used with a global aggregator (which is usually permutation and order invariant) to generate a single graph embedding for the overall graph.

3 Architecture & Approach

The overall architecture of our system is illustrated in Figure 2 and the approach can be summarized as given in the Algorithm 1. First, we execute the attack payloads in the victim's host network (more details can be found at subsection 3.1). Second, we capture the provenance information (i.e., logs) against the deployed attacks. Third, we generate provenance graphs from the captured logs (lines 1-2 of Algorithm 1). Fourth, we encode the provenance graphs using GraphSAGE (line 3 of Algorithm 1; more details can be found at subsection 3.2) and use aggregator followed by it to generate a single vector embedding for each graph (line 4 of Algorithm 1). Finally, we classify the events using both OAML [9] and SetConv [10] for detecting the output event type (lines 5-7 of Algorithm 1; more details about these models are provided in section 4).

Algorithm 1: Steps of our Overall Approach

```
Input: Graph
Output: Label for the Graph as 'Benign' or of any of the Tactics
1 Take raw log as input
2 Generate provenance graph from the log
3 Using GraphSAGE, generate the embedding for the overall graph
4 Using Aggregator, generate a single vector space embedding for each graph
5 Split the graphs into train and test set
6 Train the Machine Learning algorithms (Baseline Models including 'SetConv'
    and 'OAML') using the training set and test the models on the test set
7 Analyze the performance of the models
```

3.1 Data Collection

3.1.1 Attack Generation As discussed in section 2.1, MITRE ATT&CK framework provides a knowledge base of tactics and techniques generated from actual observations of adversarial behavior. Moreover, the Atomic Red Team provides a library that can be leveraged to execute disparate adversarial attacks that follow the tactic/technique taxonomy of MITRE ATT&CK framework to test system robustness against cyberattacks. Therefore, to emulate attacks in the victim's host network in our work, we exploit the attacks defined in the MITRE ATT&CK framework and execute them with the help of the Atomic Red Team framework.

3.1.2 Provenance Graph Generation

After the execution of attack payloads, we collect the log data of the attack traces that resembles the data provenance of the machine activities. To collect the log data, we use *Sysdig*, which is a system monitoring service. An example of the captured log data is represented in Figure 1b. Next, we generate the provenance graphs from these captured logs by using *Apache TinkerPop* [35]. In the graph generation procedure, we follow a set of rules to achieve limited attack traces as well as ensure removal of noise from the logs (more details in section 5.1.2). We write the rules with *Groovy* and execute these Groovy scripts using the *Gremlin Console*, which is a terminal that allows users to create and traverse graphs with the Apache TinkerPop. Later, the generated graphs are streamed to the graph visualization tool *Gephi* [11], which allows us to import a graph's nodes and edges as separate CSV files. These CSVs are then used to create our final provenance graphs.

We first convert the different versions of the logs to provenance graphs. A very generic definition of such provenance graphs would be to include only the processes, files, and sockets as nodes of the graph. But we also incorporate the different system calls and how different processes make use of those different system calls into the graph. For file-level nodes, *read* and *write* type system calls are emphasized, whereas for network-level nodes special types of network-level system calls are handled such as *socket*, *connect*, and *bind*.

To generate the edges in the graph, we connect the nodes in the following ways: a process node responsible for a system call is connected by an edge with the system call node; a process node is connected by an edge with a filepath node if that process accesses that filepath; a filepath node is connected by an edge with a system call node if that system call node is either *read* or *write* type and thus is used for accessing that filepath by the process. A glimpse of the generated provenance graph is shown in Figure 1b.

It is crucial to incorporate the edge weights in the graph as well. For this purpose, we identify attack trace windows using the start and end of events (using $>$ and $<$ symbols) from the field *Event Direction* of the log data. Then, we extract how many bytes are read or written for corresponding *read* and *write* system calls from the *Event Arguments* field of the log data for that trace window. The calculated byte amounts are then used as edge weights in the provenance graphs.

3.2 Graph Embedding using GraphSAGE

From the provenance graphs, we need to create suitable representations that we can feed into the machine learning models for classification. As discussed in section 2.2, we can utilize *GraphSAGE* to generate node embeddings of graphs. Although, in our work, we need graph embeddings rather than node embeddings. Therefore, we implement a global aggregator following the original GraphSAGE to create graph embeddings. That is, after the GraphSAGE creates node embeddings of a graph, these node embeddings are feed into the global aggregator that generates a single graph vector embedding for the whole graph. This aggregator could be permutation and order invariant.

Because of the supervised learning setting, we also need the embedding encoder to be trainable using labels. Therefore, we use a very simple neural network following the graph encoder. After the encoder training phase, we discard the simple neural network and keep the encoder only for following training phases and testing. Therefore, we can produce valuable embeddings from the generated provenance graphs for the given classification task.

4 Models

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 adopt two models from our previous works to be the classifiers, which are OAML [9] and SetConv [10]. The details of these two algorithms are in the Section 4.1 and Section 4.2.

4.1 Online Metric Learning

Online Metric Learning (OML) or otherwise said, Online Adaptive Metric Learning (OAML) [9] 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 OAML method learns a non-linear similarity metric unlike others which use a pre-selected linear metric (e.g., Mahalanobis distance [36]). 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 generates an n-dimensional embedding vector as output in different latent space.

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) are similar to x_t^+ (positive) but dissimilar to x_t^- (negative) (see Figure 3). 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.

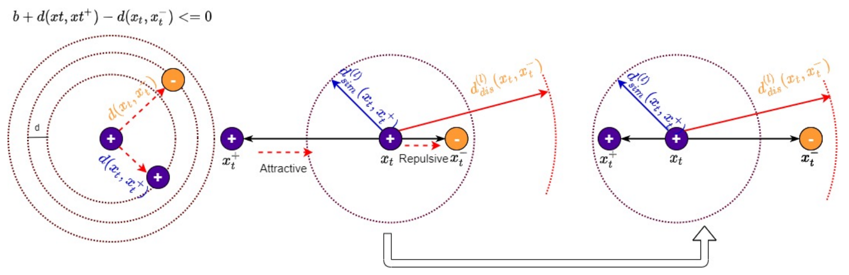

$b + d(xt, xt^+) - d(x_t, x_t^-) <= 0$
d
$d(x_t, x_t^-)$
$d(x_t, x_t^+)$
x_t^+
Attractive
$d^{(l)}_{sim}(x_t, x_t^+)$
$d^{(l)}_{dis}(x_t, x_t^-)$
x_t
Repulsive
x_t^-
$d^{(l)}_{sim}(x_t, x_t^+)$
$d^{(l)}_{dis}(x_t, x_t^-)$
x_t^+
x_t
x_t^-

Fig. 3: Data instance before applying OML (left) and data instance after projection using OML (right).

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). $E_\ell \in \{E_0, \ldots, E_L\}$ 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.

At time t, the metric embedding $f^{(\ell)}(x_t^*)$ of an arrived triplet constraint (x_t, x_t^+, 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 $h^{(0)} = x_t^*$. We use x_t^* to denote a anchor (x_t). It might be positive (x_t^+), or negative (x_t^-) instance. For the activation of the ℓ^{th} hidden layer, we use $h^{(\ell)}$ to denote. To reduce the search space and accelerate training, we limit theLearned metric embedding $f^{(\ell)}(x_t^*)$ to a unit sphere (i.e., $||f^{(\ell)}(x_t^*)||_2 = 1$)

In the training step, 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 is defined by following:

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

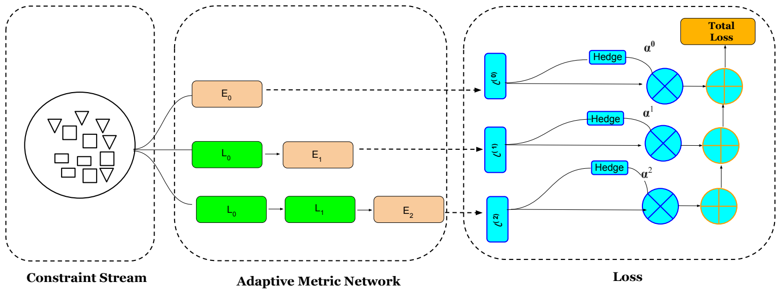


Fig. 4: OAML network structure consists of L_i linear layer and Embedding layers E_i layer.

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:

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

We evaluate the similarity and dissimilarity errors using an *adaptive-bound triplet loss* (ABTL) constraint [9] to estimate $\mathcal{L}^{(\ell)}$ and update parameters $\Theta^{(\ell)}$, $W^{(\ell)}$ and $\alpha^{(\ell)}$.

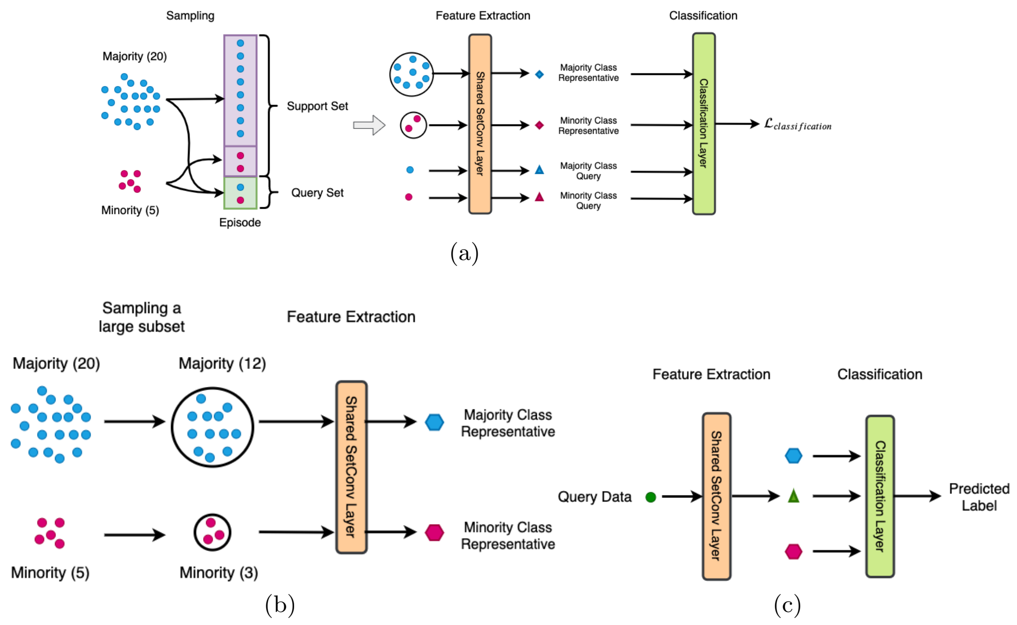


Fig. 5: Overview of the proposed approach. (a) The training procedure of SetConv. At each iteration, SetConv is fed with an episode to evaluate the classification loss for model update. Each episode consists of a support set and a query set. The support set is formed by a group of samples where the imbalance ratio is preserved. The query set contains only one sample from each class. (b) The post training step of SetConv, which is performed only once after the main training procedure. In this step, we extract a representative for each class from the training data and later use them for inference. Here we only perform inference using the trained model and do not update it. (c) The inference procedure of SetConv. Each query data is compared with every class representative to determine its label.

4.2 SetConv: A New Approach for Learning from Imbalanced Data

Machine Learning method finds its use in broad domains of applications. However, when the data Imbalance Ratio (IR) is high, most existing machine learning methods are biased towards the majority class and their performance deteriorates seriously. We use the set convolution (SetConv) [10] operation and a new training strategy named as episodic training to assist learning from imbalanced class distributions. SetConv is designed to alleviate the class imbalance by explicitly learning the weights of convolution kernels based on the intra-class and inter-class correlations, and uses the learned kernels to extract a single representative for each class. Thus, the subsequent classifier, which takes these class representatives as input, always perceives a balanced class distribution. As a naturally permutation-invariant operation, SetConv guarantees the uniqueness of the learned class representatives despite the order of input samples.

As shown in Fig. 5a, our model is composed of a SetConv layer and a classification layer. For simplicity, we first consider a binary classification problem and later extend it to the multi-class scenario. At each iteration during training, the model is fed with an *episode* sampled from the training data, which is composed of a support set and a query set. The support set preserves the imbalance ratio of training data, and the query set contains only one sample from each class. Once the SetConv layer receives an episode, it extracts features for every sample in the episode and produces a representative for each class in the support set. Then, each sample in the query set is compared with these class representatives in classification layer to determine its label and evaluate the classification loss for model update. We refer this training procedure as *episodic training.*

After training, a post training step is performed only once to extract a representative for each class from the training data, which later be used for inference (Fig. 5b). It is conducted by randomly sampling a large subset of training data (referred as S_{post}) and feeding them to the SetConv layer. *Note that we only perform inference using the trained model and do not update it in this step.* We can conduct this operation because the SetConv layer has learned to capture the class concepts, which are insensitive to the episode configuration during training.

The inference procedure of the proposed approach is straightforward (Fig. 5c). For each query sample, we extract its feature via the SetConv layer and then compare it with those class representatives obtained in post training step. The class that is most similar to the query is assigned as the predicted label.

4.2.1 SetConv Layer In many real-world applications, the minority class instances often carry important and useful knowledge that need intensive attention by the machine learning models [6, 16, 33]. Based on this prior knowledge, we choose to design the SetConv layer in a way such that the *feature extraction process focuses on the minority class.* We achieve it by estimating the weights of the SetConv layer based on the relation between the input samples and a pre-selected minority class anchor. This anchor can be freely determined by the user. We adopt a simple option, i.e., *average-pooling* of the minority class samples. As shown in Figure 6, this weight estimation method assists the SetConv layer in capturing not only the intra-class correlation of the minority class, but also the inter-class correlation between the majority and minority classes.

Suppose $\mathcal{E}_t = \{\mathcal{S}_t, \mathcal{Q}_t\}$ is the episode sent to the SetConv layer at iteration t, where $\mathcal{S}_t = \big(X_{maj} \in \mathcal{R}^{N_1 \times d}, X_{min} \in \mathcal{R}^{N_2 \times d}\big)$ is the support set and $\mathcal{Q}_t = \big(q_{maj} \in \mathcal{R}^{1 \times d}, q_{min} \in \mathcal{R}^{1 \times d}\big)$ is the query set. In general, X_{maj}, X_{min}, q_{maj} and q_{min} can be considered as a sample set of size N_1, N_2, 1 and 1 respectively. For simplicity, we abstract this sample set into $X \in \mathcal{R}^{N \times d}, N \in \{N_1, N_2, 1\}$. We define the set convolution (SetConv) operation as:

$$\begin{aligned} h[Y] &= \frac{1}{N} \sum_{i=1}^{N} X_i \cdot g(Y - X_i) \\ &= \frac{1}{N} \Big(X \circ g(Y - X) \Big) \end{aligned} \tag{4}$$

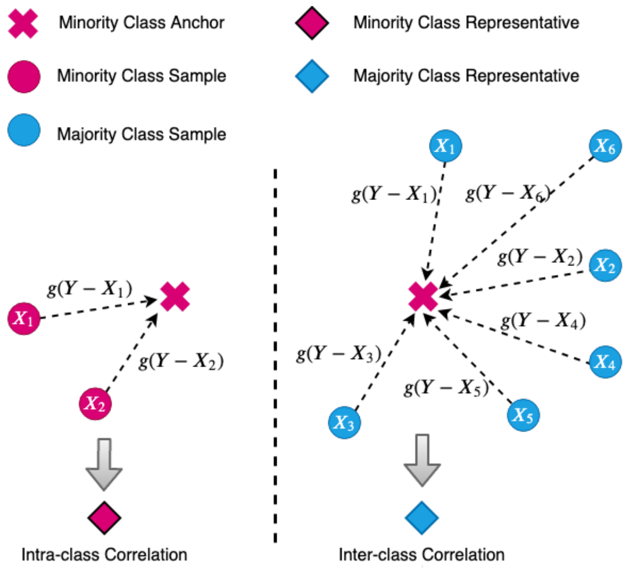


Fig. 6: Relations between the input samples and a pre-selected minority class anchor are used by SetConv to estimate both intra-class correlations and inter-class correlations.

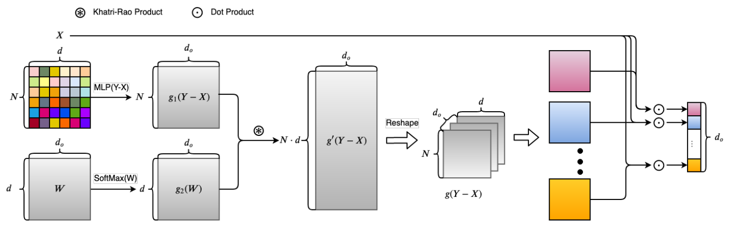


Fig. 7: The computation graph of the SetConv layer. Here Y is a minority class anchor. $W \in \mathcal{R}^{d \times d_o}$ is a weight matrix to learn that records the correlation between the input and output variables. Specifically, the i_{th} column of $g_2(W)$ gives the weight distribution over input features for the i_{th} output feature. It is indeed a feature-level attention matrix. In addition, we estimate another data-sensitive weight matrix $g_1(Y-X)$ from the input data. The final convolution weight tensor is simply the Khatri-Rao product of $g_1(Y-X)$ and $g_2(W)$.

where $Y \in \mathcal{R}^{1\times d}$, $g(Y-X) \in \mathcal{R}^{N\times d\times d_o}$ and $h[Y] \in \mathcal{R}^{1\times d_o}$ denote the minority class anchor, kernel weights and the output embedding respectively. Here, $\circ$ is the tensor dot product operator, i.e., for every $i \in \{1, 2, \dots, d_o\}$, we compute the dot product of X and $g(Y-X)[:, :, i]$.

Unfortunately, directly learning $g(Y-X)$ is memory intensive and computationally expensive, especially for large-scale high-dimensional data. To overcome this issue, we introduce an efficient method to approximate these kernel weights. Instead of taking X as a set of d-dimensional samples, we stack these samples and consider it as a giant dummy sample $X' = Concat(X) \in \mathcal{R}^{1\times Nd}$. Then, Eq. 4 is rewritten as

$$h[Y] = \frac{1}{N}\Big(X' \cdot g'(Y-X)\Big) \tag{5}$$

where $g'(Y-X) \in \mathcal{R}^{Nd\times d_o}$ is the transformed kernel weights. To efficiently compute $g'(Y-X)$, we propose to approximate it as the *Khatri-Rao* product[3] [31] of two individual components, i.e.,

$$\begin{aligned} g'(Y-X) &= g_1(Y-X) \circledast g_2(W) \\ &= \text{MLP}(Y-X;\theta) \circledast \text{SoftMax}(W, 0) \end{aligned} \tag{6}$$

where $W \in \mathcal{R}^{d\times d_o}$ is a weight matrix that represents the correlation between input and output variables. $g_2(W)$ takes softmax over the first dimension of W, and is indeed a *feature-level attention* matrix. The i_{th} column of $g_2(W)$ provides the weight distribution over input features for the i_{th} output feature. On the other hand, $g_1(Y-X)$ is a *data-sensitive* weight matrix estimated from input data via a Multilayer Perceptron (MLP) by considering their relation to the minority class anchor. Similar to data-level attention, $g_1(Y-X)$ helps the model customize the feature extraction process for input samples, which potentially improves the model performance. Figure 7 shows the detailed computation graph of the SetConv layer.

4.2.2 Classification Suppose the feature representation obtained from the layer for X_{maj}, X_{min}, q_{maj} and q_{min} in the episode are denoted by v^s_{maj}, v^s_{min}, v^q_{maj} and v^q_{min} respectively. The probability of predicting v^q_{maj} or v^q_{min} as the majority class is given by

$$P(c=0|x) = \frac{\exp(x \odot v^s_{maj})}{\exp(x \odot v^s_{maj}) + \exp(x \odot v^s_{min})} \tag{7}$$

where $\odot$ represents the dot product operation and $x \in \{v^q_{maj}, v^q_{min}\}$.

Similarly, the probability of predicting v^q_{maj} or v^q_{min} as the minority class is

$$P(c=1|x) = \frac{\exp(x \odot v^s_{min})}{\exp(x \odot v^s_{maj}) + \exp(x \odot v^s_{min})} \tag{8}$$

where $x \in \{v^q_{maj}, v^q_{min}\}$.

We adopt the well-known cross-entropy loss for error estimation and use the Adam optimizer to update model.

4.2.3 Extension to Multi-Class Scenario Extending SetConv for multi-class imbalance learning is straightforward. We translate the multi-class classification problem into multiple binary classification problems, i.e., we create a one-vs-all classifier for each of the N classes. Specifically, for a class c, we treat those instances with label $y = c$ as positive and those with $y \neq c$ as negative. The anchor is hence computed

[3] https://en.wikipedia.org/wiki/Kronecker_product

based on the smaller one of the positive and negative classes. The prediction probability $P(y = c|x)$ for a given instance x is computed in a similar way as Eq. 7,

$$P(y = c|x) = \frac{\exp(x \odot v^s_{y=c})}{\exp(x \odot v^s_{y \neq c}) + \exp(x \odot v^s_{y=c})} \tag{9}$$

Therefore, the predicted label of the instance x is $\text{argmax}_c P(y = c|x)$.

5 Experiments

5.1 Experiment Setting

5.1.1 Dataset In our experiments, we use two datasets to evaluate our approach in the paper: DAPT 2020 [24] and the Graph dataset that we collect following the procedure described in section 3.

DAPT 2020 is an APT dataset, which covers all attack stages of different aspects of the real-world APT. In this dataset, a total of 4 different APT phases are involved. Hence, it contains 5 different classes including the benign class. Although, two APT phases, which are the lateral movement and data exfiltration, only contain 6 and 4 malicious instances, respectively. Therefore, we discard these two classes, since their malicious instances only constitute 0.1 % and 0.23 % of the total instances, respectively. We run our experiments on the rest of the data with the other 3 classes. Table 1 shows the statistic information of the DAPT 2020 dataset, including the number of instances in each file and the percentages of malicious instances.

Graph Dataset includes different variations. It contains the graph embeddings generated from the collected original unmodified logs. Based on the original log file, three variations are constructed after filtering each log file leveraging the domain knowledge (filtering procedure is described in section 5.1.2). This procedure extracts 1) one-third, 2) half and 3) two-third of the whole log file. The graph embeddings of these log variations are then generated and included with the graph embeddings of the original log files to make the entire graph dataset. This is how we achieve limited attack traces for each log executed against the techniques.

Table 2 provides the Graph dataset statistics. We have one benign tactic and the other nine being malicious tactics totaling ten classes. It is evident from the number of instances for each class that this dataset is imbalanced.

5.1.2 Filtration of Logs For the filtering of logs, first, we generate *keywords* for each of the TTP that specify the TTP characteristics. For example, the attack payload T1546 accesses the *bash_profile* or *.bashrc* system files and eventually read or write them. Therefore, some keywords for T1546 would be bash_profile, bashrc, open, write, dup, etc. Some sample *keywords* for some select TTPs are provided in Table 3. Thus leveraging domain knowledge, we generate a large number of keywords for each of the TTP which are considered constituents of a whole sentence. We define this sentence as the *Base Sentence* and use it to filter the logs. We consider each of the log lines as an individual sentence by taking three specific columns in the log file namely: *process name*, *filepath*, and *system call*. We generate sentence embeddings for every log line as well as the *Base Sentence* using the Universal Sentence Encoder [5]. Next, we calculate the sentence similarity score between each of the sentences generated from the log lines and the *Base Sentence*. We then sort the similarity scores and based on the sorted results, four different copies of each log file are generated. The schematic depiction of this whole process is provided in figure 8.

This filtration not only allows to generate graph instances with limited attack traces but also removes noise from the logs as we eliminate the part of logs that are

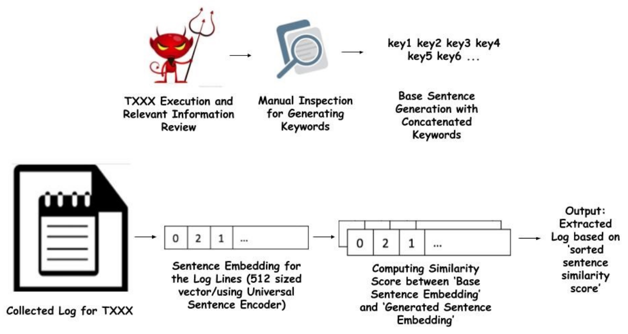

key1 key2 key3 key4 key5 key6 ...
TXXX Execution and Relevant Information Review
Manual Inspection for Generating Keywords
Base Sentence Generation with Concatenated Keywords
Collected Log for TXXX
0 2 1 ...
Sentence Embedding for the Log Lines (512 sized vector/using Universal Sentence Encoder)
Computing Similarity Score between 'Base Sentence Embedding' and 'Generated Sentence Embedding'
Output: Extracted Log based on 'sorted sentence similarity score'

Fig. 8: Generating Different Version of Logs

less related to the particulars of some attack. The log lines which are not necessarily critical to some attack are discarded as the similarity scores are calculated against the *Base Sentences* which is generated using *keywords* crucial to the attack techniques.

Table 1: DAPT 2020 Dataset Statistics

File Name	Total No.	Malicious No.	Benign No.	Malicious %	Benign %
custom_reconnaissance.csv	29254	4405	24849	15.06%	84.94%
custom_foothold.csv	17486	8632	8854	50.63%	49.37%
custom_lateralmovement.csv	4051	4	4047	0.1%	99.9%
custom_dataexf.csv	2617	6	2611	0.23%	99.77%
All Files	53408	13047	40361	24.43%	75.57%
First Two Files	46740	13037	33703	27.89%	72.11%

Table 2: Graph Dataset Statistics - Benign class is the 10^{th} *class*

Class	0	1	2	3	4	5	6	7	8	9	10	Total
Number	32	16	84	44	476	16	16	32	112	124	408	1360
Percentage	2.35%	1.18%	6.18%	3.24%	35%	1.18%	1.18%	2.35%	8.24%	9.12%	30%	100%

Table 3: Example Keyword Collection for TTPs

TTPs	Keywords
T1546	bash_profile bashrc open write dup
T1485	dd dev zero var log syslog open read write
T1529	shutdown reboot halt poweroff writev
T1049	access var run utmpx netstat who
T1148	HIST CONTROL echo export whoami

5.1.3 Baseline We compare our algorithm with several traditional and state-of-the-art ML-based models for evaluation. For traditional ML-based methods, we utilize Decision Tree and Support Vector Machine. We briefly discuss the other utilized state-of-the-art models here.

EasyEnsemble [21] is an algorithm based on ensemble and under sampling. The random under sampling method is used to produce balanced bootstrap samples. Then, it utilizes AdaBoost learners to get a final ensemble model.

Multilayer Perceptron (MLP) [15] is a feedforward Artificial Neural Network (ANN). It comprises multiple layers, and each layer contains a number of perceptrons.

Being a widely used model in the machine learning area, MLP shows competitive performance in many different practical applications.

KMeans-SMOTE [7] is a oversampling method to relieve the class imbalance problem. This method is based on kmeans clustering algorithm and SMOTE over sampling. The major advantage is generating effective instance avoided noise. We use a Support Vector Machine (SVM) to be a classifier after oversampling. We also use SVM itself to be a baseline. In both two settings, we use polynomial kernel in the SVM classifiers.

5.1.4 Evaluation Metric There are many different metrics to evaluate classification problems like accuracy. For binary classification, precision, recall, and F1 score are very popular metrics to evaluate the performance of a model. However, they are not a credible metric for multi-class classification problems, although there exist variants of these metrics to deal with such problems. For a given class c, we have True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN). The precision (P), recall (R), and F1 score (F1) are defined by following:

$$P = \frac{TP}{TP + FP} \tag{10}$$

$$R = \frac{TP}{TP + FN} \tag{11}$$

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{12}$$

Then, we choose the the Macro Averaging variants to evaluate multi-class classification problems.

$$P_{macro} = \frac{1}{m} \sum_{i=1}^{m} P_i \tag{13}$$

$$R_{macro} = \frac{1}{m} \sum_{i=1}^{m} R_i \tag{14}$$

$$F1_{Macro} = \frac{2 \times P_{macro} \times R_{macro}}{P_{macro} + R_{macro}} \tag{15}$$

where m is the number of classes. The macro averaging uses the same weight to every class even the instance numbers are imbalanced among different classes.

5.1.5 Experiment Setup For DAPT 2020 and graph datasets, we design different setups to run the experiments due to different data structures. For DAPT 2020, we utilize the embedding method from the original paper [24]. Then, we normalize the feature vectors as the ranges of different features are very different. Because most of the algorithms are designed for taking feature vectors to be the inputs, graph data is hard to feed in these models directly. To solve this problem, we have two phases to train and predict data in graph format. The first phase is the embedding encoder. We utilize GraphSAGE [12] to convert the data from graphs to their feature embeddings. Second, we train a GraphSAGE model with GCN [19] to be the aggregator, and it produces node embeddings. The feature embedding of a given graph is the average value of its node embeddings. Third, we concatenate the GraphSAGE model with a simple MLP model to train with the training set. Finally, we discard the simple MLP part and use the intermediate output (graph embedding from GraphSAGE) to encode training and testing datasets for future works.

The imbalance of datasets is a general property in the cybersecurity scenario. Although some of the methods are designed to handle this problem, the embedding encoder still suffers from this problem because they are separate phases. Therefore, we design two different experiment setups. The first one keeps the same ratios among classes to train an encoder. Next, the dataset is balanced by re-sampling. An embedding encoder is trained based on the balanced dataset. Then, the training and testing datasets are encoded by the encoder, but it remains original inter-class instance ratios. In this way, the feature embeddings are more robust, and the evaluation focuses more on the following classifier rather than the encoder.

5.2 Result

In this subsection, we briefly discuss and compare the experimental results for both dataset.

5.2.1 Graph Dataset The experiment results of the graph dataset are shown in Table 4 and Table 5. Table 4 shows the results when embedding encoder is trained without re-sampling and Table 5 shows the performances of models when embedding encoder is trained with re-sampling.

It is apparent from the Table 4 that *SetConv* achieves the best result compared to other methods in terms of the macro accuracy, precision, recall, and F1 score with 0.9228, 0.8022, 0.7163, and 0.7161 values, respectively. *KMeans-SMOTE* achieves an impressive accuracy of 0.9007, although it shows lower performance for other metrics with macro precision, recall, and F1 score of 0.7022, 0.7047, and 0.6894, respectively. Due to the data imbalance influence, *OAHU* shows substandard results. The accuracy is 0.8732 and the macro precision, recall, and F1 score are 0.674, 0.6448, and 0.6528, respectively. Overall, *SetConv* is better than other models under this experimental setup.

If we use a better embedding encoder (with re-sampling), absolute performances of every model are improved, but the relative performances do not exhibit much changes, as shown in Table 5. *SetConv* demonstrates that it exceeds other methods in terms of performance no matter which experiment setup is adopted. It achieves an accuracy of 0.9449 and obtains scores of 0.9674, 0.9196, and 0.9406 for macro precision, recall, and F1 score, respectively. *MLP* and *KMeans-SMOTE* still produce reasonable performances. On the other hand, the *EasyEnsemble* displays relatively low performance with an accuracy of just 0.864, and the macro precision, recall, and F1 score of 0.9047, 0.8479, and 0.8602, respectively.

Table 4: Results for the Graph dataset
Feature embedding encoder is trained without re-sampling method

	DecisionTree	**SVM**	**KmeansSMOTE**	**EasyEnsemble**	**MLP**	**OAHU**	**SetConv**
Accuracy	0.8971	0.9136	0.9007	0.8088	0.8915	0.8732	**0.9228**
Macro Precision	0.633	0.6162	0.7022	0.5475	0.4881	0.674	**0.8022**
Macro Recall	0.6265	0.6212	0.7047	0.5617	0.5302	0.6448	**0.7163**
Macro F1	0.6292	0.6174	0.6894	0.526	0.5058	0.6528	**0.7161**

Table 5: Results for the graph dataset
Feature embedding encoder is trained with re-sampling method

	DecisionTree	**SVM**	**KmeansSMOTE**	**EasyEnsemble**	**MLP**	**OAHU**	**SetConv**
Accuracy	0.9136	0.9063	0.9283	0.864	0.9357	0.8382	**0.9449**
Macro Precision	0.9305	0.9576	0.9149	0.9047	0.9638	0.7346	**0.9674**
Macro Recall	0.9076	0.8515	**0.9511**	0.8479	0.9182	0.7923	0.9196
Macro F1	0.9171	0.8963	0.9281	0.8602	0.9357	0.7488	**0.9406**

5.2.2 DAPT 2020 In these experiments, we discard the severe imbalance classes. The dataset has few classes as compared to the graph dataset and easier to classify. As a result, most of the algorithms exhibit good performances on this dataset, as represented in Table 6. Even *OAHU* does not have a strong design for an imbalanced dataset, yet it achieves competitive performance with an accuracy of 0.9633. *EasyEnsemble* also shows a strong performance in this experiment setup, but *SetConv* slightly performs better than most of the algorithms with an accuracy of 0.986, and the macro precision, recall, and F1 score of 0.9779, 0.98, and 0.9789, respectively.

Table 6: Results for the DAPT 2020 dataset

	DecisionTree	**SVM**	**KmeansSMOTE**	**EasyEnsemble**	**MLP**	**OAHU**	**SetConv**
Accuracy	**0.9988**	0.9795	0.9721	0.8688	0.9583	0.9633	0.986
Macro Precision	**0.9983**	0.9659	0.9507	0.9415	0.9467	0.9429	0.9779
Macro Recall	**0.999**	0.9751	0.9667	0.7632	0.896	0.9643	0.98
Macro F1	**0.9987**	0.9704	0.9585	0.7897	0.9169	0.9531	0.9789

6 Related Works

There have been several works that exploit TTPs from the MITRE ATT&CK matrix to identify the tactics and stages of an APT attack. One such work is the RapSheet [14] that makes use of the rule matching capability of an endpoint detection response (EDR) tool to discover TTPs in system logs. From there, they build provenance graph, named initial infection point (IIP) provenance graph, that highlights the threat alerts discovered by the EDR tool. A tactical provenance graph is then generated that contains only the alerts, and a threat score is assigned to the graph. In our work, we attempt to identify different tactics of an APT attack through the use of TTPs, rather than an entire APT attack. Furthermore, we propose a method for identifying TTPs rather than relying on an EDR (Endpoint Detection and Response) tool. In addition, our approach does not rely on manually set rules in an EDR tool for discovering TTPs in system logs.

Holmes [23] generates a high-level compact graph to summarize ongoing attack campaign. The key technique is to map activities from the host logs which corresponds to the kill chain. Contrary to that, our approach focuses on cropped or trimmed parts of a system level log or scenario to identify the tactics. In a real time setting, the window for capturing provenance might shift in an unwanted way, which could result in missing information that can lead to false identification or no identification of attack tactics. Our approach shows that we can detect specific attack tactics with cropped provenance capture. Our approach also differs from this work in that it does not need a rule-based scheme to generate high-level scenario (HSG) graphs to map low-level activities to high-level attack tactics; rather it utilizes simple domain knowledge to learn a robust model and identify attack tactics. As such, the extension of our model is simple and easy with new evolving tactics and techniques.

Unicorn [13] generates a fixed sized graph sketch periodically through building a runtime in-memory histogram in a sequential process. It needs to process the graph sequentially to extract crucial information about an attack campaign. Ayoade et. al. [3] propose an approach that adopts a provenance capture-based approach to detect different versions or parts of advanced persistent threats. To capture provenance, they use the Camflow [26] and CamQuery [28] tool. The limitation of their approach is that the data collection is limited in platform and also the provenance definition is not compatible with the notion of graph embedding for generating data instances. Our approach addresses this problem and also works on identifying different tactics of an advanced persistent threat attack campaign.

Sheyner et al. [32] introduce an automated way of generating attack graphs using symbolic model checking algorithms. In another work [18], a backtracking technique is used to identify processes and files which may have an impact on the detection point. This approach attempts to find out the chain of events which essentially leads to the intrusion from the entry point. Our approach adopts simple steps for generating provenance graph without any necessity of backtracking and generates data instances which are used for training the machine learning models.

Harmful Episode Reconstruction by Correlating Unsuspicious Logged Events (HERCULE) [29] uses multiple log in the system to generate a multi-dimensional weighted graph to run community detection algorithm on it. It is built on the observation of attack related log entries being heavily and densely connected. Our system only uses a single source log to identify different APT tactics which might or might not be interleaved with benign traces. Made: Security analytics for enter-prise threat detection [25] detects new malicious activities in enterprise networks and for addressing the issue of large data, they adopt a filtration process on the network communications data. But as this approach only attempts to see the start of a whole attack, e.g. detection of malicious domains which can cause the system to be compromised at the first place, it cannot detect whether a system has already been compromised and is in any intermediary tactical phase of an APT campaign.

7 Conclusion & Future Work

Our work mainly focuses on identifying different tactics of Advanced Persistent Threat based on logs that are reduced to make noise free, which also yields limited attack traces for such identification. Our approach generates single vector space embedding for each of the graphs necessary for supervised training setting of machine learning models. The data imbalance being prominently present in the Graph dataset, SetConv method performs well in handling this issue compared to other machine learning methods.

In the future, we plan to explore our approach for windows-based attacks. In addition, we plan to devise techniques to map appropriate responses to alerts of different attack tactics for the ease of system monitoring. To address the cropping or trimming problem of important provenance graphs (or otherwise said, important part of host logs), our future endeavor would be to incorporate a shifting window-based approach to accommodate crucial aspects of an attack into consideration while detecting an attack. Moreover, our future work will focus on incorporating diversified attribution of a graphical depiction into the embedding scheme so that the learning process can be aided for machine learning models.

Our approach can also detect novel attacks if the filtration of the logs is performed based on the common *'keywords'* for all the attacks. More specifically, the log lines related to common system calls for each of the Linux commands can be discarded and from those filtered logs, provenance graphs can be generated. We intend to apply our approach to identify novel techniques under the tactics in the future.

Disclaimer Commercial products are identified in order to adequately specify certain procedures. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the identified products are necessarily the best available for the purpose.

Acknowledgements The research reported herein was supported in part by NIST award 60NANB20D178, NSF awards DMS-1737978, DGE-2039542; and an IBM faculty award (Research).

References

1. https://attack.mitre.org/groups/G0049/
2. Red Canary (Sep 2020), https://github.com/redcanaryco/atomic-red-team
3. Ayoade, G., Akbar, K.A., Sahoo, P., Gao, Y., Agarwal, A., Jee, K., Khan, L., Singhal, A.: Evolving advanced persistent threat detection using provenance graph and metric learning. In: 2020 IEEE Conference on Communications and Network Security (CNS). pp. 1–9 (2020)
4. CALDERA: Caldera. https://github.com/mitre/caldera, (Accessed on June 10, 2021)
5. Cer, D., Yang, Y., Kong, S., Hua, N., Limtiaco, N., John, R.S., Constant, N., Guajardo-Cespedes, M., Yuan, S., Tar, C., Sung, Y., Strope, B., Kurzweil, R.: Universal sentence encoder. CoRR **abs/1803.11175** (2018)
6. Chen, C., Shyu, M.: Clustering-based binary-class classification for imbalanced data sets. In: Proceedings of the IEEE International Conference on Information Reuse and Integration, IRI 2011, 3-5 August 2011, Las Vegas, Nevada, USA. pp. 384–389. IEEE Systems, Man, and Cybernetics Society (2011)
7. Douzas, G., Bacao, F., Last, F.: Improving imbalanced learning through a heuristic oversampling method based on k-means and smote. Information Sciences **465**, 1–20 (Oct 2018)
8. endgameinc: Red team automation (rta). https://github.com/endgameinc/RTA, (Accessed on June 10, 2021)
9. Gao, Y., Li, Y.F., Chandra, S., Khan, L., Thuraisingham, B.: Towards self-adaptive metric learning on the fly. In: The World Wide Web Conference. pp. 503–513. ACM (2019)
10. Gao, Y., Li, Y.F., Lin, Y., Aggarwal, C., Khan, L.: Setconv: A new approach for learning from imbalanced data (2021)
11. Gephi: The open graph viz platform. https://gephi.org/, (Accessed on June 10, 2021)
12. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30, pp. 1024–1034. Curran Associates, Inc. (2017)
13. Han, X., Pasquier, T., Bates, A., Mickens, J., Seltzer, M.: Unicorn: Runtime provenance-based detector for advanced persistent threats. arXiv preprint arXiv:2001.01525 (2020)
14. Hassan, W., Bates, A., Marino, D.: Tactical provenance analysis for endpoint detection and response systems. In: 2020 IEEE Symposium on Security and Privacy (SP). pp. 1172–1189. IEEE Computer Society, Los Alamitos, CA, USA (may 2020)
15. Hastie, T., Tibshirani, R., Friedman, J.: The elements of statistical learning: data mining, inference, and prediction. Springer Science & Business Media (2009)
16. He, H., Garcia, E.A.: Learning from imbalanced data. IEEE Trans. Knowl. Data Eng. **21**(9), 1263–1284 (2009)
17. Jiang, X., Walters, A., Xu, D., Spafford, E., Buchholz, F., Wang, Y.M.: Provenance-aware tracing ofworm break-in and contaminations: A process coloring approach. In: 26th IEEE International Conference on Distributed Computing Systems (ICDCS'06). pp. 38–38 (2006). https://doi.org/10.1109/ICDCS.2006.69
18. King, S.T., Chen, P.M.: Backtracking intrusions. SIGOPS Oper. Syst. Rev. **37**(5), 223–236 (Oct 2003)

19. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)
20. Lee, K.H., Zhang, X., Xu, D.: High accuracy attack provenance via binary-based execution partition. In: NDSS (2013)
21. Liu, X.Y., Wu, J., Zhou, Z.H.: Exploratory undersampling for class-imbalance learning. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) **39**(2), 539–550 (2008)
22. Ma, S., Zhang, X., Xu, D.: Protracer: Towards practical provenance tracing by alternating between logging and tainting. In: NDSS (2016)
23. Milajerdi, S.M., Gjomemo, R., Eshete, B., Sekar, R., Venkatakrishnan, V.: Holmes: real-time apt detection through correlation of suspicious information flows. In: 2019 IEEE Symposium on Security and Privacy (SP). pp. 1137–1152. IEEE (2019)
24. Myneni, S., Chowdhary, A., Sabur, A., Sengupta, S., Agrawal, G., Huang, D., Kang, M.: Dapt 2020-constructing a benchmark dataset for advanced persistent threats. In: International Workshop on Deployable Machine Learning for Security Defense. pp. 138–163. Springer (2020)
25. Oprea, A., Li, Z., Norris, R., Bowers, K.: Made: Security analytics for enterprise threat detection. In: Proceedings of the 34th Annual Computer Security Applications Conference. p. 124–136. ACSAC '18, Association for Computing Machinery, New York, NY, USA (2018)
26. Pasquier, T., Han, X., Goldstein, M., Moyer, T., Eyers, D., Seltzer, M., Bacon, J.: Practical whole-system provenance capture. In: Proceedings of the 2017 Symposium on Cloud Computing. pp. 405–418. SoCC '17, ACM, New York, NY, USA (2017)
27. Pasquier, T., Han, X., Moyer, T., Bates, A., Hermant, O., Eyers, D., Bacon, J., Seltzer, M.: Runtime analysis of whole-system provenance (2018)
28. Pasquier, T., Han, X., Moyer, T., Bates, A., Hermant, O., Eyers, D., Bacon, J., Seltzer, M.: Runtime analysis of whole-system provenance. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 1601–1616. CCS '18, ACM, New York, NY, USA (2018)
29. Pei, K., Gu, Z., Saltaformaggio, B., Ma, S., Wang, F., Zhang, Z., Si, L., Zhang, X., Xu, D.: Hercule: Attack story reconstruction via community discovery on correlated log graph. In: Proceedings of the 32nd Annual Conference on Computer Security Applications. p. 583–595. ACSAC '16, Association for Computing Machinery, New York, NY, USA (2016)
30. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 701–710 (2014)
31. Rabanser, S., Shchur, O., Günnemann, S.: Introduction to tensor decompositions and their applications in machine learning. CoRR **abs/1711.10781** (2017)
32. Sheyner, O., Haines, J., Jha, S., Lippmann, R., Wing, J.M.: Automated generation and analysis of attack graphs. In: Proceedings 2002 IEEE Symposium on Security and Privacy. pp. 273–284 (2002)
33. Sun, Y., Kamel, M.S., Wong, A.K.C., Wang, Y.: Cost-sensitive boosting for classification of imbalanced data. Pattern Recognition **40**(12), 3358–3378 (2007)
34. Thakur, V.: The sykipot attacks (2011), `https://www.symantec.com/connect/blogs/sykipot-attacks`
35. TinkerPop, A.: Apache tinkerpop. `https://tinkerpop.apache.org/`, (Accessed on June 10, 2021)
36. Xiang, S., Nie, F., Zhang, C.: Learning a mahalanobis distance metric for data clustering and classification. Pattern recognition **41**(12), 3600–3612 (2008)