TMAS: A Traffic Monitoring Analytics System Leveraging Machine Learning

Elaheh Jalalpour*, Milad Ghaznavi*, Raouf Boutaba*, Toufik Ahmed†
* Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada
†CNRS-LaBRI (UMR5800), University of Bordeaux / Bordeaux INP, France
*{ejalalpo | eghaznav | rboutaba}@uwaterloo.ca, †tad@labri.fr

***Abstract*—Content Delivery Networks (CDNs) provide high quality of service by storing content in edge-servers close to users. Attacks against CDN edge-servers can lead to loss in revenue and reputation. Attacks are becoming more sophisticated, and new attacks are being introduced constantly. In our previous work, we developed a security orchestration system driven by high-level security policies to dynamically deploy mitigation services. In this system, security policies are triggered at the occurrence of low-level alerts that correspond to misuse of an edge-server's resources. However, a network operator must know the effects of any attack on resources to deploy an appropriate mitigation service. Moreover, pin-pointing the actual cause (e.g., malicious IPs) of resource misuse is challenging. Also, edge-server's resources may not be affected by some attacks.**

Leveraging advanced machine learning techniques, we extend our system to detect new and sophisticated attacks. The goal is to enable the network operator to specify higher-level security policies without worrying about analyzing low-level resource usage alerts. Further, policy enforcement can trigger the deployment of mitigation services only for malicious entities identified by the alerts. In this perspective, we propose a Hybrid Classification Clustering (HCC) method that not only detects known sophisticated attacks accurately (with 99.9% detection recall) but is capable of detecting new attacks (with 56.4% detection recall). Further, to improve the detection rate of new attacks and anomalies, we propose an Autoencoder-based Network Anomaly Detection (ANAD) method using a fully-connected autoencoder model. The evaluation results show that our model achieves 76.7% recall surpassing the isolation forest and the local outlier factor methods.

***Index Terms*—attacks, machine learning, hybrid, anomaly detection, autoencoder**

I. Introduction

Content Delivery Networks (CDNs) provide high quality of service by caching and delivering content in *edge-servers* located at points of presence close to users. Successful attacks on CDN edge-servers can result in loss of revenue and reputation for the CDN providers. Disrupting the operation of edge-servers can take down a CDN. With this goal in mind, attackers launch DDoS attacks against the edge-servers. Attackers also manipulate communication protocols to exhaust the resources of CDNs. They craft HTTP requests to bypass filters, poison and pollute the cache, hijack sessions, and launch other attacks [1], [10], [12]. The protection system should be able to dynamically mitigate sophisticated well-known and new attacks.

978-3-903176-15-7 © 2019 IFIP

In our recent work [13], we introduced a security orchestration system that is programmed by security policies to dynamically handle attacks. An important component of this system is the Security Monitoring Analytics System (SMAS) that monitors the resources of the edge-server and generates low-level resource misuse alerts triggering security policies. To dynamically deploy appropriate mitigation services in response to attacks, a network operator must know or investigate the effects of attacks on the resources. Acquiring this knowledge is not trivial. Specifying security policies using low-level alerts is a time-consuming and error-prone process. Moreover, identifying and mitigating the actual cause (e.g., malicious IPs) of resource misuse is complicated. Handling real world complex and new attacks requires much deeper analysis of their effects that do not necessarily leave footprints on the resource usage of an edge-server.

To address the above challenges, it is important to endow the security orchestration system with the capability to detect complex and new attacks. Traditionally, *attack signatures* are used to detect intrusions. Human experts manually craft these signatures based on their knowledge of the intrusions, which requires substantial delay to recognize new attacks and identify their signatures. This limitation motivates the application of machine learning and data mining methods that automatically devise models replacing manually crafted attack signatures [4]. In the literature, misuse and anomaly detection have been extensively used for intrusion detection. Misuse detection is commonly done using supervised machine learning, which requires training over a labeled dataset's records [19]. Although accurate in detecting known attack types, supervised learning methods are weak in detecting new attacks not seen in their training dataset. Unsupervised anomaly detection methods model the normal behavior of a system and detect deviations from it; they are capable of detecting new attacks. A number of anomaly detection algorithms have been proposed before. However, they often suffer from low accuracy rate and higher false alarms compared to supervised methods [14].

In this paper, we extend SMAS by introducing a Traffic Monitoring Analytics System (TMAS) capable of generating high-level alerts for complex and new attacks. TMAS monitors incoming traffic to the edge-server and analyzes it using two proposed machine learning methods. To tackle the aforementioned limitation of supervised machine learning, we propose a Hybrid Classification Clustering (HCC) method that not only

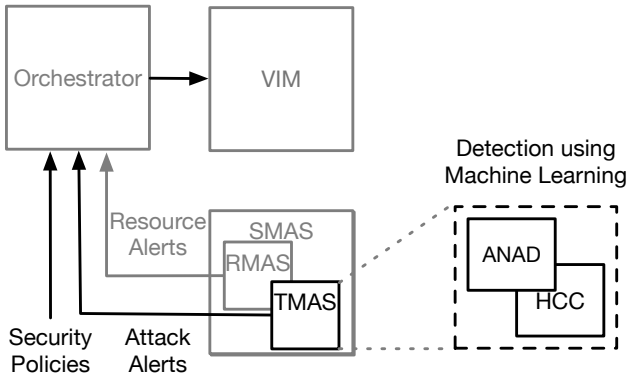


Fig. 1: Security Orchestration System

achieves high accuracy in the detection of known attacks (with 99.9% detection recall), but it can recognize new attacks (with 56.4% detection recall). To address the limitations of traditional anomaly detection methods, we propose Autoencoder-based Network Anomaly Detection method (ANAD). This method improves the accuracy of HCC in detecting new attacks. Our method achieves 76.7% recall for anomalies and surpasses the isolation forest and Local Outlier Factor (LOF), two commonly used anomaly detection methods.

The remainder of this paper is organized as follows. We provide a brief background of our security orchestration system and introduce TMAS in Section II. The proposed machine learning methods are presented in Section III. We evaluate the system in Section IV. We discuss the related work in Section V and conclude this paper in Section VI.

II. Security Orchestration System

Our security orchestration system reported in [13] is composed of three components as shown in Figure 1. A network operator programs the orchestrator using security policies dictating the system behavior. The enforcement of the policies involves the orchestrator receiving security alerts, deploying, modifying, and removing security services. Virtual Infrastructure Manager (VIM) provides an API for the orchestrator to manage security chains. Security Monitoring Analytics System (SMAS) monitors virtual edge-server's resources (e.g., memory and processing resources) and generates resource misuse alerts to the orchestrator (e.g., $high\_mem$ and $high\_cpu$).

In our previous work, SMAS is a Resource Monitoring Analyics System (RMAS) that only generates low-level resource alerts. These alerts are suitable in mitigating resource misuse attacks. However, programming security policies using such low-level alerts to mitigate complex attacks is challenging. In this work, we augment SMAS by introducing TMAS, a Traffic Monitoring Analytics System which leverages machine learning (ML) for detecting complex attacks.

Traffic Monitoring Analytics System (TMAS): As illustrated in Figure 1, SMAS is extended with a new component called TMAS. This component monitors the incoming traffic to the edge-server, analyzes it using our proposed ML methods, and fires attack alerts, such as app_ddos, $port\_scan$, and $anomaly$. Based on these alerts, appropriate security policies are triggered to mitigate attacks.

III. Traffic Monitoring Analytics System

TMAS periodically monitors the edge-server's incoming traffic and extracts features of traffic flows identified using the five tuple (source and destination IP addresses, source and destination ports, and protocol). Then, TMAS runs our ML algorithms to analyze the features for attack and anomaly detection. In the following, we describe our proposed ML methods, namely Hybrid Classification Clustering (HCC) and Autoencoder-based Network Anomaly Detection (ANAD).

A. Hybrid Classification Clustering (HCC)

Table I compares HCC with classification and clustering methods. As shown, HCC provides the benefits of both classification and clustering. Similar to a classifier, HCC detects the types of known attacks. Similar to a clustering method, HCC identifies whether a flow is a new attack (though the type of a new attack is not determined). HCC trains a classifier using labeled training dataset to detect existing attacks in the dataset, which we refer to as *known* attacks. HCC uses a clustering method to discover clusters that contain new attacks, i.e., previously unseen attacks in the classifier's training dataset.

Overview: As shown in Algorithm 1, HCC receives as incoming traffic features f and the number of clusters C (line 1). Note that C is always greater than the number of classes. For each class there is a corresponding cluster. Additional clusters correspond to new attacks. In addition to known class labels, the algorithm predicts the `new_attack` label. First, the algorithm runs a pre-trained classifier and a clustering method (lines 2 and 3). Next, HCC recognizes the clusters that most probably contain the data points of a class and returns the rest of the clusters as new attacks (line 4). Combining all results, HCC predicts new labels for the data points (line 5). Still to clarify is how to find new attack clusters and how to find a new prediction.

Algorithm 1 Hybrid Classification Clustering

1: **procedure** HCC(f, C)
2: $y_1 \leftarrow$ classification(f)
3: $y_2 \leftarrow$ clustering(f, C)
4: $N \leftarrow$ NOVELS(y_1, y_2)
5: $y_3 \leftarrow$ PREDICT(y_1, y_2, N)
6: **return** y_3
7: **end procedure**

Finding New Attack Clusters: Algorithm 2 receives the predictions and finds clusters that contain new attacks. To do so, it compares the classification and clustering predictions and relates clusters to classes. A cluster is considered *known*, if its intersection with at least one class is larger than other clusters'. The algorithm finds known clusters and considers the remaining clusters as new attacks.

The algorithm receives the results of the classifier and clustering methods in the form of two lists with indices

Methods	Known Attack Type	New Attack	New Attack Type	Labeled Training Dataset
Classification	✓	×	×	✓
Clustering	×	✓	×	×
HCC	✓	✓	×	✓

TABLE I: A Comparison Between Machine Learning Methods

showing the data points and values showing the predictions. Next, it creates two dictionaries d_1 and d_2 (line 2). The former is a mapping of the classes to their corresponding data points, and the latter is a dictionary from the clusters to their corresponding data points. The algorithm initializes a list K that is iteratively extended with known clusters (lines 3-11). For each class l_1, the algorithm finds the size of the intersection of the members of l_1 with all the clusters' members (lines 6-9). The clusters with the biggest intersection with each class are identified and added to K (line 10). Each class can be mapped to one or multiple clusters. The algorithm returns the clusters that are not in K (line 12).

Algorithm 2 Finding New Attack Clusters

```
1:  procedure NOVELS(y1, y2)
2:      d1, d2 ← map(y1), map(y2)
3:      K ← ∅
4:      for l1 in d1 do
5:          c ← array(|keys(d2)|)
6:          for l2 in d2 do
7:              s ← |values(d1, l1) ∩ values(d2, l2)|
8:              set value of c at l2 to s
9:          end for
10:         K ← K ∪ argmaxes(c)
11:     end for
12:     return keys(d2) − K
13: end procedure
```

Finding A New Prediction: Dictionary d_2 maps clusters to their corresponding data points (line 2). List L is initialized and iteratively updated by the data points in the new attack clusters (line 4-6). List y_3 that stores the final results is initialized by the classification results, then updated by changing the values of data points in L to `new_attack` (lines 7-9).

Algorithm 3 Finding a New Prediction

```
1:  procedure PREDICT(y1, y2, N)
2:      d2 ← map(y2)
3:      L ← ∅
4:      for c in N do
5:          L ← L ∪ values(d2, c)
6:      end for
7:      y3 ← y1
8:      set values of y3 at L to new_attack
9:      return y3
10: end procedure
```

B. Autoencoder-based Network Anomaly Detection (ANAD):

Anomaly detection approaches are able to discover new intrusions by modeling the normal behavior of the system and detecting any deviation from it [6], [11], [20]. Several factors, such as the difficulty of having a boundary around the normal behavior, intelligent adversaries who adapt themselves to new detection methods, and insufficient training/testing data make anomaly detection a complex task. Anomaly detection methods are categorized as *supervised*, *semi-supervised*, and *unsupervised*. The first category classifies traffic into two classes, normal and anomalous. As discussed before, the supervised methods are not adequate in detecting new attacks. Semi-supervised approaches are trained based on normal traffic only. Providing real traffic traces which only contain normal data is a complex task. Unsupervised learning approaches detect anomalies without any labeled training data with the assumption that normal traffic instances are much more frequent than anomaly instances [3], [6]. However, most unsupervised anomaly detection methods suffer from low accuracies.

We propose an unsupervised network anomaly detection method using autoencoder, a deep learning model.

Autoencoder: Autoencoders are deep neural networks trained to learn efficient data coding. Figure 2 shows a sample autoencoder. An autoencoder contains an input layer, one or more hidden layers, and an output layer. The input and output layers of an autoencoder have the same dimension, while the number of neurons may vary in hidden layers. The first part of this model, the *encoder*, transforms the high dimensional input x into a lower dimensional representation z. Equation (1) is the encoder function of a single encoder layer, where σ is the activation function, W is the weight matrix, and b is the bias vector of the hidden layer. The autoencoder decodes data in what is called the *latent space* and learns to extract the most important features in the decoding phase. The second part, *decoder*, uses this lower representation to rebuild the input in x'. Equation (2) presents the decoder function; W' is the weight matrix, and b' is the bias vector of the hidden layer. The loss function of an autoencoder is based on the *reconstruction error*, the difference between the input x and prediction x'.

$$z = \sigma(Wx + b) \tag{1}$$

$$x' = \sigma(W'z + b') \tag{2}$$

ANAD: Our method trains a fully-connected autoencoder with unlabeled training dataset. Note that the training dataset consists of mostly normal data points. The reconstruction errors of anomalies are higher than those of the normal data points, because the autoencoder tries to learn the encoding

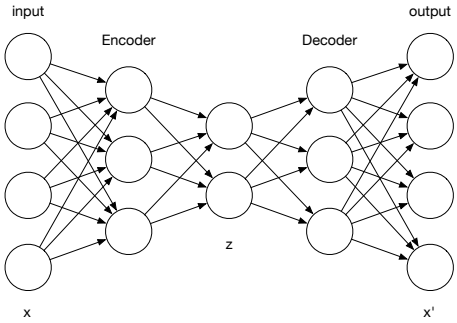


Fig. 2: Autoencoder Neural Network

and decoding of the normal data points which constitute most of the data. To compute the reconstruction error of a data point, we use its *Residual Sum of Square (RSS)* as presented in Equation (3), where x is the input, x' is the prediction, and n is the number of input features.

$$\sum_{i=1}^{n} (x_i - x'_i)^2 \tag{3}$$

ANAD analyzes incoming flows as shown in Algorithm 4. This method receives the input data points x and contamination parameter c stating the proportion of anomalies in the given data points (line 1). ANAD runs the autoencoder and stores the predictions in x' (line 2). For each data point, this method computes and stores its reconstruction error using RSS (lines 3-6). The algorithm sorts the data points descendingly based on their reconstruction errors and returns the top c percent as anomalies (lines 7-8). For a constant number of alerts, a network operator can set the contamination parameter c based on the flow incoming rate.

Algorithm 4 Autoencoder-based Network Anomaly Detection

1: **procedure** ANAD(x, c)
2: $x' \leftarrow$ autoencoder(x)
3: $e \leftarrow \emptyset$
4: **for** x_1, x_2 in x, x' **do**
5: $e \leftarrow e \cup \text{RSS}(x_1, x_2)$
6: **end for**
7: sort x based on e descendingly
8: **return** top $c\%$ of x
9: **end procedure**

IV. Evaluation

A. Experimental Platform

We use a server cluster (256 GB RAM, 32-cores 2.00 GHz Xeon CPUs), equipped with NVIDIA Tesla K10 GPU (320 GBps memory bandwidth, 3072 CUDA cores 745 MHz). The server runs Ubuntu 16.04 with Linux kernel version 4.4.0.

B. Dataset

We leverage a labeled dataset *CIC-2017* [22] to train and test our models. This dataset consists of normal and multiple types of attack traffic. In addition to packet capture (pcap) files, network flows, and their corresponding features have been extracted using CICFlowMeter [2]. To simulate an edge-server traffic, we use flows towards a victim server.

Class	Number of Flows	Label
Normal	182491	0
DoS Hulk	230124	1
Port scan	158804	2

TABLE II: Data Points for HCC Evaluation

C. Training and Testing

We use 70% and 30% of the labeled data points as the training dataset and testing dataset, respectively. These data points are labeled as either normal or attack (e.g., DDoS and port scan). There are 81 features for each flow from which we extract 76 features. IP addresses and port numbers are 32-bit and 16-bit numerical values, respectively. These numerical features are commonly used to train machine learning algorithms; however, flows from different classes might have close numerical port numbers, and the machine learning algorithm interprets the close numbers as similarity between data points. The same argument applies to IP addresses. Thus, the source and destination IPs, the source and destination ports, and the flow identification are removed from our feature set.

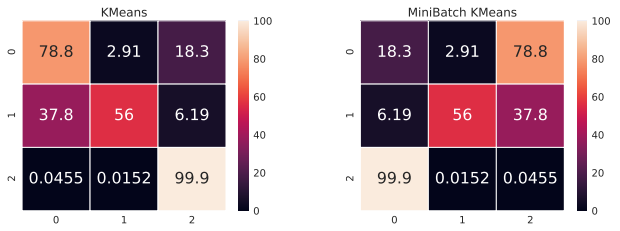
Throughout this section, the performance of a machine learning method is reported using the normalized confusion matrix, where an element ij represents what percentage of class i is classified under class j. In this way, the elements on the diagonal show the *recall* or *true positive rate* values. The sum of the non-diagonal elements of a row shows the *false negative rate* for the corresponding class.

D. Attack Detection Performance

To evaluate the performance of HCC, we need to equip this algorithm with a classifier and a clustering method. To do so, we run and compare a number of classifiers and select the one that achieves the highest performance. We do the same for clustering methods. To show the effectiveness of HCC in mitigating new attacks, we compare our hybrid method with the best selected classifier. For training and testing purposes, we use three classes of flows listed in Table II.

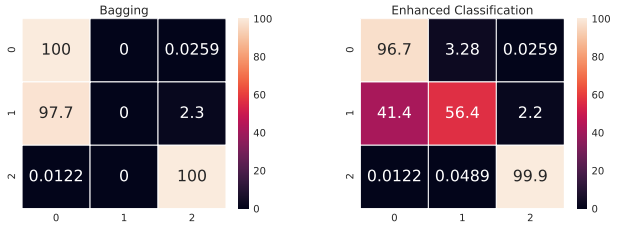
1) Classifiers and Clustering Methods Performance: We trained a decision tree, a random forest, and a bagging classifier. We also trained an *ensemble voting*, which uses a voting mechanism between the above three classifiers. Although these models have close recall values, the bagging classifier has a recall of 100% for both normal and DoS Hulk classes and 99% for the port scan class which makes this classifier the best among those tested.

We used *K-means* and *Mini-batch K-means* to divide flows into 3 clusters. Their performance is shown in Figure 3. A cluster with the most data points of a class c is mapped to c and the recall of c is computed based on the data points in



(a) K-means (b) Mini-batch K-means

Fig. 3: Performance of Clustering Methods



(a) Bagging (b) HCC

Fig. 4: Performance of Bagging and HCC

the mapped cluster. For example, in Figure 3b, the recall for the normal class is 78.8% which is the percentage of normal data points in cluster 2. The different mappings for the two algorithms are due to the random initialization of the cluster centroids. Figure 3 shows that these two algorithms have the same recall values suggesting both of them as good candidates.

2) HCC Performance: We equip HCC with the best classification method, the bagging classifier, and one of the clustering candidates, K-means. Bagging classifier and HCC are trained over two classes (normal and port scan), then they are tested for all three classes shown in Table II. The goal is to evaluate the performance of HCC in detecting the data points of the unseen class (DoS Hulk). From the new attack data points, the bagging classifier mis-classifies 97.7% and 2.3% under the normal and port scan classes, respectively. Mis-classification as the normal class can cause undesirable outcomes; no mitigation service is deployed, and DoS attack can exhaust the resources of the victim and take down its services. Mis-classification as a wrong attack results in the deployment of inappropriate mitigation services that not only does not mitigate the threat, but contributes to the attack and consumes more resources of the victim.

The performance of HCC is reported in Figure 4b. HCC correctly detects 96.7% and 99.9% of the normal and port scan data points, respectively. For the new attack data points, HCC is able to detect 56.4%, while mis-classifies 41.4% and 2.2% as normal and port scan classes, respectively. The false negative rate for the normal data points is 3.3% due to the inaccuracy of the unsupervised clustering method, K-means. We believe that a more advanced clustering algorithm can improve HCC's performance.

Class	Number of Flows	Label
Normal	438693	0
Anomaly	10293	1

TABLE III: Data Points for the ANAD Evaluation

Parameters	Isolation Forest			LOF		
	examined	default	chosen	examined	default	chosen
n_jobs	1, 1	1	-1	1, 1	1	-1
contamination	.1, .05, .022	.1	.1	.1, .05, .022	0.1	0.1
n_estimators	50, 100, 300, 500	100	300	-	-	-
max_samples	auto[a], .001, .01	auto	0.01	-	-	-
max_features	.02, .05, 1.0	1.0	.02	-	-	-
n_neighbors	-	-	-	20, 25, 30	20	30

[a]min(256, number of samples)

TABLE IV: Anomaly Detection Methods Parameters

E. Anomaly Detection Performance

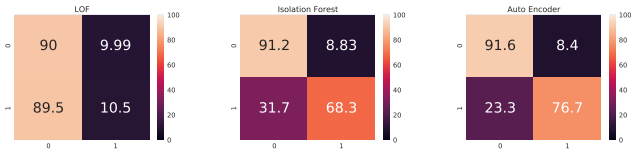
We evaluate ANAD by comparing its accuracy with that of *LOF* [5] and *isolation forest* [15], two commonly used anomaly detection methods. All these methods, including ANAD, receive several input parameters which affect their detection performance. We tune the values of the parameters to optimize their recalls for anomalies. To do so, we employ an exhaustive grid search that examines all the combinations of given values for all parameters. Finally, we use the data points of the two classes provided in Table III.

1) Anomaly Detection Methods Performance: For each data point, LOF computes the local density which is the metric depicting how isolated this data point is compared with its k neighbors. Isolation forest is an ensemble learning method that combines the results of multiple decision trees each of which produces an anomaly score. Both methods are given *contamination*, an input parameter that identifies the proportion of anomalies. Optimized using the grid search, the default, examined, and chosen values of the input parameters are shown in Table IV.

The performance of these two methods is shown in Figure 5a and Figure 5b, respectively. LOF's recall for the anomalies is 10.5%, and that of isolation forest is 68.3%. The recall of the normal class is almost 90% for both methods, and the false negative rate is almost 10%. LOF and isolation forest produce respectively 98% and 86% false discovery rates (i.e., the number of normal data points detected as anomalies divided by the total number of data points detected as anomalies). These poor results motivate the use of deep learning based approaches for anomaly detection.

2) ANAD Performance: ANAD uses an autoencoder with input and output layers of size 76 and two hidden layers. The hyper parameters, examined values, and chosen ones are reported in Table V.

Figure 5c reports ANAD performance. ANAD outperforms LOF significantly. In comparison with isolation forest, the recall values are improved by 0.4% and 8.4%. These results confirm that the autoencoder-based anomaly detection achieves higher performance in the detection recalls of normal



(a) LOF (b) Isolation Forest (c) ANAD

Fig. 5: Performance of Anomaly Detection Methods

and anomaly data points compared to the two commonly used anomaly detection methods.

Hyper-params	examined	chosen
contamination	.1, .05, .022	.1
n_neurons	5, 35	5
activation_function	relu, tanh, linear	linear
epochs	100, 150	150
batch_size	100, 1000	100
dropout_rate	0.0, 0.2	0.0
weight_constraint	1, 5	5
kernel_initialization	uniform, normal	normal

TABLE V: ANAD's Hyper Parameters

V. Related Work

Recent attention has been given to hybrid supervised and unsupervised learning. Some studies, surveyed in [17], use unsupervised methods for dimensionality reduction/feature extraction later used for supervised classification. In [24], intrusion detection is performed by combining multi-layered SVM with kernel principal component analysis to decrease the training time. Other approaches use a combined supervised unsupervised approach to enhance the mitigation of classification methods. In [16], K-means and naive bayes classification are used, and data-points are classified into two classes of attack and benign traffic. Although achieving high accuracy this work does not support unknown attack detection. In [8], random forest is used to detect known attacks, and weighted K-means is used to cluster the remaining traffic. The clusters are later labeled as either normal or attack traffic. This solution has a high false positive rate. To evaluate our proposed method, we tested multiple classifiers and clustering techniques and chose the best ones, namely bagging classifier and K-means clustering. In our solution, we detect multiple attack types, and our false positive rate is lower than that of the method proposed in [8]. Further, we use a more recent dataset, *CIC-2017* [22] to train the models. The outdated datasets used by other methods are not representative of a real network, for instance they lack traffic diversity. The CIC-2017 dataset contains a set of most up-to-date attacks collected from 2016 McAfee report.

Deep learning based anomaly detection recently received significant attention in different fields, such as fraud detection, medical diagnosis, and network intrusion detection [18], [23], [25]. In [21], a generative adversarial network simultaneously trains a generator to produce anomalous data which are close to the normal data and trains a discriminator to detect the anomalies. Unsupervised deep learning models are also employed for learning a representation of the data in a lower dimension than the input's. Later, this representation is used for supervised anomaly detection [9], [23], [25]. In [9], a one-class SVM uses features extracted by a deep belief network for classification. A stacked non-symmetric deep autoencoder has been used in [23] to extract features for random forest classifier. Unlike other methods, we use autoencoders to design an unsupervised network anomaly detection method. Our unsupervised method is not only capable of detecting new attacks but achieves higher accuracy in comparison with commonly used unsupervised anomaly detection methods.

VI. Conclusion and Future Work

In this paper, we extend our previously developed policy-based security orchestration system [13] using advanced machine learning. Our system operates based on security alerts that trigger mitigation actions. The earlier system is capable of detecting attacks and generating low-level security alerts. We extend, in this work, our orchestration system with TMAS, a Traffic Monitoring Analytics System that employs machine learning to analyze traffic. We developed two machine learning methods, HCC and ANAD. HCC not only detects known attacks accurately (recall of 96.7% to 99.9%), but is also able to detect new attacks. ANAD is a deep learning anomaly detection method that surpasses two commonly used anomaly detection methods, namely LOF and isolation forest.

As future work, we plan to enhance HCC. The performance of HCC depends on that of the employed clustering method. Variational Autoencoders (VAEs) are deep neural networks used for unsupervised clustering [7]. A VAE model forces the latent space to have a pre-defined distribution. For clustering purposes, this distribution is a mixture of Gaussians so that the latent space is divided into multiple clusters. We believe that VAE can achieve higher performance is therefore a good candidate for HCC clustering.

VII. Acknowledgement

This work benefitted from the use of the CrySP RIPPLE Facility at the University of Waterloo.

References

[1] Akamai security advisory. https://goo.gl/VmzkpG.
[2] Flowmeter. http://www.unb.ca/cic/datasets/flowmeter.html.
[3] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita. Network anomaly detection: Methods, systems and tools. *IEEE Communications Surveys Tutorials*, 16(1):303–336, First 2014.
[4] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo. A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications*, 9(1):16, Jun 2018.
[5] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: Identifying density-based local outliers. *SIGMOD Rec.*, 29(2):93–104, May 2000.
[6] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, July 2009.

[7] N. Dilokthanakul, P. A. M. Mediano, M. Garnelo, M. C. H. Lee, H. Salimbeni, K. Arulkumaran, and M. Shanahan. Deep unsupervised clustering with gaussian mixture variational autoencoders. *CoRR*, abs/1611.02648, 2016.

[8] R. M. Elbasiony, E. A. Sallam, T. E. Eltobely, and M. M. Fahmy. A hybrid network intrusion detection framework based on random forests and weighted k-means. *Ain Shams Engineering Journal*, 4(4):753 – 762, 2013.

[9] S. M. Erfani, S. Rajasegarar, S. Karunasekera, and C. Leckie. High-dimensional and large-scale anomaly detection using a linear one-class svm with deep learning. *Pattern Recognition*, 58:121 – 134, 2016.

[10] J. C. et al. Forwarding-loop attacks in content delivery networks. In *the 23st Annual Network and Distributed System Security Symposium*, 2016.

[11] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Macia-Fernandez, and E. Vazquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers and Security*, 28(1):18–28, 2009.

[12] D. Gillman, Y. Lin, B. Maggs, and R. K. Sitaraman. Protecting websites from attack with secure delivery networks. *Computer*, 48(4):26–34, Apr 2015.

[13] E. Jalalpour, M. Ghaznavi, D. Migault, S. Preda, M. Pourzandi, and R. Boutaba. A security orchestration system for cdn edge servers. In *2018 IEEE Conference on Network Softwarization (NetSoft)*, June 2018.

[14] A. Lazarevic, L. Ertöz, V. Kumar, A. Ozgur, and J. Srivastava. A Comparative Study of Anomaly Detection Schemes in Network Intrusion Detection. In *Proceedings of the Third SIAM International Conference on Data Mining*, 2003.

[15] F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation-based anomaly detection. *ACM Trans. Knowl. Discov. Data*, 6(1):3:1–3:39, Mar. 2012.

[16] Z. Muda, W. Yassin, M. N. Sulaiman, and N. I. Udzir. Intrusion detection based on k-means clustering and naÃŕve bayes classification. In *2011 7th International Conference on Information Technology in Asia*, pages 1–6, July 2011.

[17] A. Nisioti, A. Mylonas, P. D. Yoo, and V. Katos. From intrusion detection to attacker attribution: A comprehensive survey of unsupervised methods. *IEEE Communications Surveys Tutorials*, pages 1–1, 2018.

[18] E. L. Paula, M. Ladeira, R. N. Carvalho, and T. MarzagÃčo. Deep learning anomaly detection as support fraud investigation in brazilian exports and anti-money laundering. In *2016 ICMLA 15*, pages 954–960, Dec 2016.

[19] J. Raiyn et al. A survey of cyber attack detection strategies. *International Journal of Security and Its Applications*, 8(1):247–256, 2014.

[20] F. Sabahi and A. Movaghar. Intrusion detection: A survey. In *2008 Third International Conference on Systems and Networks Communications*, pages 23–26, Oct 2008.

[21] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, and G. Langs. Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. In M. Niethammer, M. Styner, S. Aylward, H. Zhu, I. Oguz, P.-T. Yap, and D. Shen, editors, *Information Processing in Medical Imaging*, pages 146–157, Cham, 2017. Springer International Publishing.

[22] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *Proceedings of the 4th International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,*, pages 108–116. INSTICC, SciTePress, 2018.

[23] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi. A deep learning approach to network intrusion detection. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2(1):41–50, Feb 2018.

[24] I. S. Thaseen and C. A. Kumar. Intrusion detection model using fusion of pca and optimized svm. In *2014 IC3I*, pages 879–884, Nov 2014.

[25] D. Xu, E. Ricci, Y. Yan, J. Song, and N. Sebe. Learning deep representations of appearance and motion for anomalous event detection. *CoRR*, abs/1510.01553, 2015.