2018 IEEE Symposium on Security and Privacy Workshops

Exploring the Use of Autoencoders for Botnets Traffic Representation

Ruggiero Dargenio
CSAIL
MIT
dargenio@mit.edu

Shashank Srikant
CSAIL
MIT
shash@csail.mit.edu

Erik Hemberg
CSAIL
MIT
hembergerik@csail.mit.edu

Una-May O'Reilly
CSAIL
MIT
unamay@csail.mit.edu

Abstract—**Botnets are a significant threat to cyber security. Compromised, a.k.a. *malicious* hosts in a network have, of late, been detected by machine learning from hand-crafted features directly sourced from different types of network logs. Our interest is in automating feature engineering while examining flow data from hosts labeled to be malicious or not. To automatically express full temporal character and dependencies of flow data requires time windowing and a very high dimensional set of features, in our case 30,000. To reduce dimensionality, we generate a lower dimensional embedding (64 dimensions) via autoencoding. This improves detection. We next increase the volume in the flows originating from hosts in our dataset known to be malicious or not by injecting noise we mix in from background traffic. The resulting lower metaphorical *signal to noise ratio* makes the presence of a bot even more challenging to detect so we resort to a *filter encoder* or an off-the-shelf denoising autoencoder. Both the filter encoding and denoising autoencoder improve upon detection compared to when hand-crafted features are used and are comparable in performance to the autoencoder.**

1. Introduction

Botnets are a critical threat to cyber security [10]. A botnet setup consists of malicious software copied onto different devices, all of which are connected to the internet. Each compromised device is typically controlled from a central system to perform large scale, distributed attacks [7].

Recent works describe a number of approaches to detecting botnets [4], [8], [10]. Typically these systems reference network activity. They analyze information present in packets or flow logs and learn a malicious host detection model. Features are chosen by hand and consist mostly of information logged from standard network routers. Advanced feature engineering to better express information present in logs is uncommon and requires significant human expertise and effort. Despite the cost, it promises to improve the detection of malicious hosts particularly when the network traffic directed by the malicious software, i.e. *signal*, is deeply hidden in the background, i.e. *noise*, of its host's normal network communications. Automated feature engineering methods supporting detection in the face of low signal to noise ratios are our central interest.

The advent of deep learning systems has significantly reduced feature engineering efforts in computer vision and natural language processing (NLP) [11], [14]. Given just an objective function and input data, these systems can learn reduced dimensionality representations that efficiently support supervised learning. In a majority of common vision and NLP tasks, these learned representations have been shown to clearly outperform models built on traditional, hand-engineered features.

Encouraged by such positive results, we investigate how to likewise use deep learning to obtain reduced dimensionality representations of network data that can support the detection of botnets. This automation will relieve the burden of hand-crafting features. While we demonstrate that an autoencoder can improve detection, our primary motivation is not to outperform state of the art detection systems but to develop a methodology that can handle traffic of lower *bot* to *background*, i.e. *signal* to *noise* ratio. Therefore, in this work, we propose methods which would hold true irrespective of the specific choices of detection models and datasets one would use to build and test detection systems. We explore two related questions *a)* How can we automatically learn features which support the accurate detection of malicious hosts? *b)* How can we model "*noisy*" hosts that have background traffic similar to real world conditions and ensure such automated feature learning systems support detection well even in such noisy environments?

Specifically, our work makes the following contributions

- We use flow logs from the CTU-13 dataset [1] and show that organizing their features into time-windows improves a baseline detection model. We utilize this organization when learning features to detect botnets.
- We design autoencoders to learn feature spaces from temporally ordered flow data to better detect botnets. To the best of the authors' knowledge, this is the first work to demonstrate how traditional flow log-based features can be improved upon by using autoencoding.
- In an attempt to encourage the community to build more robust detectors, we also suggest a way to model noisier host traffic conditions by exploiting ignored background traffic within this dataset.
- We present an encoder architecture, named *filter encoders*, which extracts features from a noisy model of host traffic. Detectors trained on these extracted

DOI 10.1109/SPW.2018.00017

representations perform as well as those trained on less noisy host traffic. We propose this architecture for tasks beyond cyber security to handle signal to noise separation.

This paper is organized in the following manner - Section 2 describes related work. Section 3 describe the CTU-13 [1] dataset we work with. Section 4 describes our methodology. Section 5 details the research questions and our experiments while Section 6 discusses results from the experiments. We conclude and discuss future work in Section 7

2. Related Work

Botnet detection is widely pursued by the cyber security community. While rule-based systems were initially developed to detect bots [2], recent approaches have employed machine learning and other statistical techniques [3], [5], [6], [13], [15]. Throughout the work features are extracted from one of two sources - packets or flow logs, and used (fully or with feature selection) as inputs to either supervised or unsupervised learning models. We, like others, also reference flow logs which provide information in the context of a connection established by one host with another. However, our work departs from this general approach in two important ways *a)* We specifically choose to organize the logs temporally, per host, divided into time windows. *b)* We focus on then reducing the high dimensionality of the feature space extracted from the flow logs in order to improve our detection models.

Specifically, Pellegrino, et. al. [8] use time windows to aggregate information. However, they do not derive machine learning based models. Haddadi et. al. [6] use machine learning models and demonstrate high classification accuracies on varying datasets using "raw" features directly extracted at flow level. We go a step further in investigating automated feature representations. This likely would improve state of art detection accuracies on benchmarks but in this work we choose to focus on studying feature representations.

Another key distinction of our work is that we step off from the conventional approach of learning from flow data only from hosts that are known through curation to be either malicious or not. We insert the flows of background traffic into curated hosts with labels. To the best of our knowledge no other work learns from more challenging noisy host flows or explores similar automated representation techniques the way we do. We now proceed to describe the dataset we use and introduce our terminology.

3. Dataset

The botnet detection research community acknowledges that access to accurate, reliable datasets is an open concern [9], [10]. While some public datasets are poor representations of botnet behavior, others are outdated and do not reflect current botnet behavior. We use the CTU-13 dataset recently released by Garcia et. al. [1]. Terms specific to our work and the CTU-13 dataset [4] are:

- **Host** Any device connected to the network. Can potentially be compromised and act as a malicious bot.
- **Flow** A log created, typically by the network router, whenever an established connection by a host ends. It contains information like total packets sent over during the connection, the protocol used to establish the connection, time of established connection etc.
- **Scenarios** A scenario is a snapshot of network activity over a period of time. Within a scenario, during data capture for dataset creation, bots can be introduced into the network. Each scenario therefore has one or more types of bots associated with it. See, e.g. Table 1.

Hosts (and consequently *all* flows originating from them) can be labeled as bot, normal or background.

- **Bots** Hosts which have a copy of malicious software.
- **Normal** Hosts manually identified as NOT being a part of a botnet.
- **Background** Hosts labeled neither Bot or Normal with flows that express everyday network activities. We add the flow information of background hosts to our dataset when modeling noisy, everyday traffic in networks.

CTU-13 consists of network information gathered for 13 scenarios under controlled (i.e. dataset creation) conditions. In addition to information sent by hosts compromised by *malicious* bots, it comprises information generated by *normal* hosts on the network in the form of both packets and flows. It has a recommended train-test split between the different scenarios. Table 1 summarizes the different scenarios. CTU-13 also contains additional background information from random devices on the network that have not been labeled because there is no knowledge of whether they are malicious or not [4]. This information is more than 85% of the entire volume of information, yet is typically unused in developing detectors. More details can be found in Garcia et. al. [4].

4. Method

We now proceed to describe, in Section 4.1, the features we extract from each flow, and, in Section 4.2, how we divide the flow logs according to host, organize them into time windows and aggregate the high dimensional set of raw values to form inputs to our auto, filter and noise encoders. In Section 4.3 we describe how we use an autoencoder to reduce the high dimensional raw values. In Section 4.4 we explain our framework for learning detection models for windows and hosts, the latter using labeling from the former. Then, in Section 4.5, we describe how we inject noise from unlabeled hosts into the those of labeled hosts. In Section 4.6 we develop a *filter encoder* and describe a standard denoising autoencoder that address the noisier host traffic.

Scenario	Bots (B)	Normal (N)	Type	# bots	Tr/Te
1	57%	43%	Neris	1	Te
2	70%	30%	Neris	1	Te
3	19%	81%	Rbot	1	Tr
4	9%	91%	Rbot	1	Tr
5	16%	84%	Virut	1	Tr
6	38%	62%	Menti	1	Te
7	4%	96%	Sogou	1	Tr
8	8%	92%	Murlo	1	Te
9	86%	14%	Neris	10	Te
10	87%	13%	Rbot	10	Tr
11	75%	25%	Rbot	3	Tr
12	22%	78%	NSIS.ay	3	Tr
13	56%	44%	Virut	1	Tr
Total	56%	44%	**# flows**		
Train	55%	45%	378,104 (B) + 307,755 (N)		
Test	57%	42%	66,595 (B) + 48,678 (N)		

TABLE 1: Ratio of bot and normal flows in each scenario. Tr/Te denotes if the scenario is used for training or testing. Each scenario also includes background data which is not included in these ratios. Background data covers >85% of the overall data per scenario.

FEATURE	TYPE	# FEATURES
Protocol	Categorical*	19
Duration	Float	1
Direction	Categorical*	7
Total packets	Integer	1
Host bytes	Integer	1
Total bytes	Integer	1
Host IP	String	Not considered
Port	Integer	Not considered
Total number of features		**30**

TABLE 2: "Raw" features directly extracted from a flow. Categorical features are represented by one-hot encodings.

4.1. Flow Log Features

To prepare the input to our encoders, from each flow log we first directly extract a set of "raw" features per Table 2 that are most frequently used in the literature. We use them as our stepping off point in investigating whether we can synthesize more features through deep learning techniques. We do not use the host IP and port information because they can be easily spoofed. In the next section we describe how we next build our feature space by concatenating these features from multiple flow logs into windows, organized by labeled host.

4.2. Time-Windows of Flows from Labeled Hosts

We aim to preserve temporal information within a connection and among connections for its potential discriminatory value. We demonstrate in our experiments the case where not capturing this temporal aspect performs significantly worse. Thus, for each labeled host, we elect to create time windows that, concatenate the flow features of each of the host's distinct flows within the window. We introduce a hyperparameter *max-flows* to bound the flows per time-window. The temporally ordered, concatenated features of

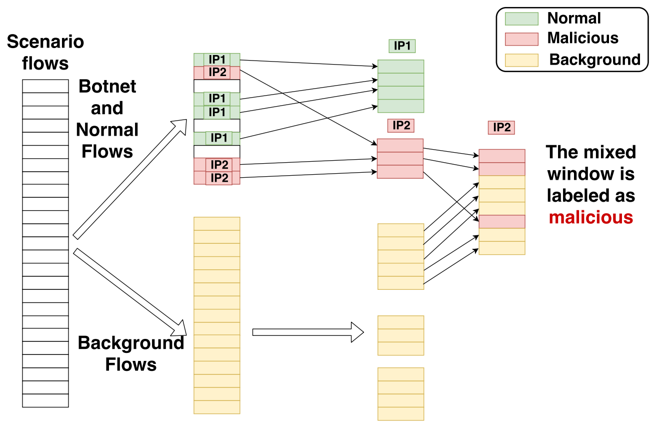

Figure 1: Construction of labeled time windows from labeled (i.e. *curated*) flows (comprising labeled dataset) and mixed labeled and *background* flows (comprising noisy dataset.)

each window then comprise a training (or test) input. Associated with each input is the label (malicious or normal) of the host. The transformed dataset for every labeled host is then passed into the encoders (see Section 4.6) of our detection framework to reduce the dimensionality and generate a more efficient representation for the subsequent window and host detection model learning tasks, see Figure 1. The time-window duration is a second hyperparameter that we tune.

To illustrate this, assume we have 60 minutes of flow logs for a host. Further, assume that the host creates 60,000 connections in this time frame. Each connection is characterized by the feature set described in Table 2. If we consider a time-window of 1 second, we will witness a varying quantity of flows in each of the 3,600 resulting windows. Assuming *max-flows* equals 10, our feature space is 300 dimensional, i.e. 30 $\times$ 10 features. Time-windows containing less than *max-flows* flows are padded with 0s.

4.3. Feature Autoencoding

Autoencoders are unsupervised methods to learn a lower-dimensional representation from a higher dimensional space by reconstructing its own inputs (instead of learning a target value) [12]. An autoencoder takes an input vector $\mathbf{x} \in \mathbb{R}^d$, and first maps it to a hidden representation $\mathbf{y} \in \mathbb{R}^{d'}$ through a deterministic mapping $\mathbf{y} = f_\theta(\mathbf{x}) = s(\mathbf{W}\mathbf{x} + \mathbf{b})$, parameterized by $\theta = \{\mathbf{W}, \mathbf{b}\}$. $\mathbf{W}$ is a $d' \times d$ weight matrix and $\mathbf{b}$ is a bias vector. The resulting latent representation $\mathbf{y}$ is then mapped back to a "reconstructed" vector $\mathbf{z} \in \mathbb{R}^d$ in input space $\mathbf{z} = g_{\theta'}(\mathbf{y}) = s(\mathbf{W}'\mathbf{y} + \mathbf{b}')$ with $\theta' = \{\mathbf{W}', \mathbf{b}'\}$. Each training $\mathbf{x}^i$ is thus mapped to a corresponding $\mathbf{y}^i$ and a reconstruction $\mathbf{z}^i$. The average reconstruction error is then minimized:

$$\theta^*, \theta'^* = \arg\min_{\theta,\theta'} \frac{1}{n} \sum_{i}^{\infty} L(\mathbf{x}^i, \mathbf{z}^i) \quad (1)$$

where L is the squared loss $||\mathbf{x} - \mathbf{z}||^2$.

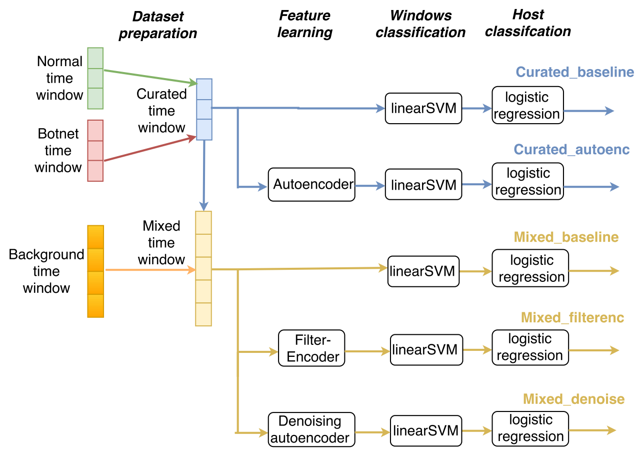

Figure 2: Experimental framework with blue path indicating labeled dataset steps and yellow path indicating "noisy" dataset steps that stem from mixing background unlabeled host flows with labeled ones to create mixed time-windows.

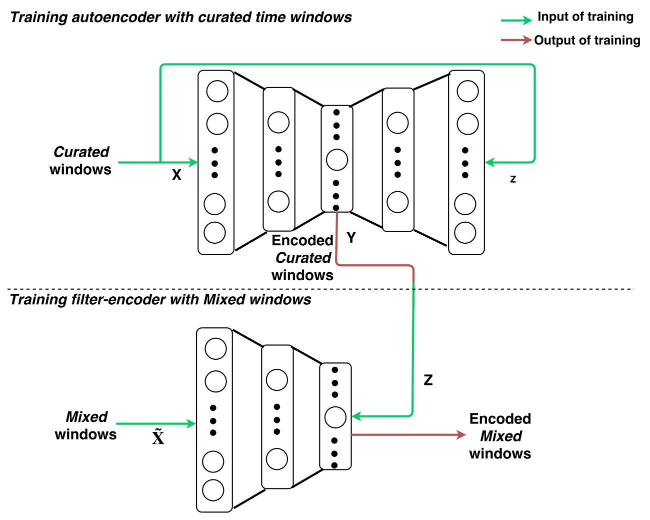

Figure 3: Autoencoder and filter-encoder architecture.

In our work, we use encoders to derive representations of information from time-windows. While the equations described above model a shallow one-layer network, we use multiple layers. The top half of Figure 3 illustrates the autoencoder we train and subsequently use for the labeled dataset's representations. The red line indicates the representation advanced forward in the framework for malicious window detection.

4.4. Malicious Window and Host Detection

Our malicious window detection task involves predicting whether a window belongs to a malicious host or not. For detection we simply train a linear SVM. It is trained on encoded features and labels that are propagated from the host labels. We note here that we do not use more complex models since the aim of our work is not to achieve state of the art detection accuracy but is instead to investigate whether neural representations are able to support the *improved* accuracy of a specific detection model vs conventional, directly extracted raw features.

Once we have predicted labels for each window, we train a detector to learn a threshold to classify a host as being malicious. For *host detection* we use the number of windows detected as malicious and the number of windows in the host as features to learn this threshold. We use logistic regression to learn the threshold.

In summary this framework (see, Figure 2) *for learning from labeled hosts* consists of *Step 1)* flow feature extraction, *Step 2)* time-window feature assembly by ordered feature concatenation and transfer of host label, *Step 3)* autoencoding of time-window features to a lower dimensional representation, *Step 4)* window detection with the lower dimensional representation, *Step 5)* host bot detection with the label quantities of a host's windows and total windows.

4.5. Modeling Additional Noisy Host Traffic

Recall our ultimate goal is to detect malicious hosts that are burying their bot-directed communication deeply within normal network traffic. We therefore need a method to increase the background traffic originating from labeled hosts in curated datasets like CTU-13. For this we will mix CTU-13's background data, itself organized into time windows, into the already prepared time windows of each labeled host, see Figure 1. For each host that is labeled, i.e *curated*, we uniformly randomly pick without replacement one window from a random host in *background* and add its flows to the *curated* host's window. We thus randomly distribute flows from the *background* dataset into the *curated* dataset and have the option of retuning the *max-flows* parameter to account for extra flows in every time-window (although in this dataset the distribution of the number of flows per time window per host for both the *background* and the *curated* dataset is similar so our value for *max-flows* remains the same). We refer to the resulting dataset as *mixed* in the remainder of this work.

We can now run experiments with *mixed* to observe the impact of "noise" on encoded feature support for malicious window and host detection accuracy. However, it would be futile to use the autoencoding step of our existing framework. If we do, we will encode the noise and bury the signal. Instead, as we next explain in Section 4.6, we resort to *filter encoders* or denoising autoencoders.

4.6. Filter and Denoising Encoders

After adding *background* flows to model "noise" in real hosts, we do not wish to utilize any properties directly from the *background* set, since in a real scenario, we would be unaware of such information. Instead we take a neural network approach, see Figure 3, similar to denoising autoencoders, to sift through the noise while learning lower dimensional feature representations. Our intuition is to move the representations learned by the noise-added dataset (*mixed* set) in the direction of the representations learned from the dataset without noise (*curated* set). Therefore we train the *mixed* data on the intermediate representation $\mathbf{y}_c$ learned from the *curated* set. Specifically, for an input $\mathbf{x}_c$ from the *curated* dataset, we proceed as discussed in the previous subsection - we learn an intermediate representation $\mathbf{y}_c = f_{\theta_c}(\mathbf{x}_c) = s(\mathbf{W}\mathbf{x_c} + \mathbf{b_c})$, by mapping it back to $\mathbf{z}_c = g_{\theta_c}(\mathbf{y}_c) = s(\mathbf{W^T}\mathbf{y_c} + \mathbf{b}')$. For an input $\mathbf{x}_m$ from the *mixed* set, we learn a representation $\mathbf{y}_m = f_{\theta_m}(\mathbf{x}_m) = s(\mathbf{W_m}\mathbf{x_m} + \mathbf{b_m})$, parametrized by $\theta_m = \{\mathbf{W_m}, \mathbf{b_m}\}$. We then use $\mathbf{y}_c$ to minimize the average reconstruction error:

$$\theta_m^*, \theta_c^* = \arg\min_{\theta_m, \theta_c} \frac{1}{n} \sum_i^{\infty} L(\mathbf{y}_m^i, \mathbf{y}_c^i) \tag{2}$$

where L is the squared loss $||\mathbf{y_m} - \mathbf{y_c}||^2$

A denoising autoencoder is a stochastic version of the autoencoder. Intuitively, a denoising autoencoder does two things: encode the input, and undo the effect of a corruption process stochastically applied to the input of the autoencoder. The latter can only be done by capturing the statistical dependencies between the inputs. Vincent et al. [12] discuss different perspectives to understanding how denoising autoencoders work.

5. Experiments

We experimentally investigate the following questions:

- **Q1** Does capturing temporal dependency in low level features by consolidating multiple flows in a window help detect presence of malicious flows in the hosts? How does it compare to detections made by individual flows. Through this question, we validate our design to consider flows features per window for each host.
- **Q2** Can autoencoders learn feature representations that perform better than just the raw features sourced from flow logs?
- **Q3** With a lower signal to noise ratio from labeled hosts, i.e. when using the *mixed* dataset, how well do the baseline features support detection? How effective are the filter and denoising autoencoders in distilling the additional noise and supporting detection?

Using the CTU-13 dataset we normalize all the features values using min-max normalization and split the dataset into its recommended training and test sets (see Table 1), see [4] for more information. We further split the training data into a validation set to tune our hyperparameters. Two bots in the validation are not in the training set. The test set has three bots that are not in the training set.

We implement the linear SVM (C=1, L2-penalty) and logistic regression (C=1, L2-penalty) with `scikit-learn`. We develop the encoders with `Keras 2.1.1`. We run all our experiments on a CUDA-enabled GTX 1080TI GPU.

Method	Recall	Precision	F1-score
curated_flow	1	0.61	0.75
curated_windows	0.81	0.89	0.85
curated_autoenc	0.87	0.9	0.88
mixed_windows	0.58	0.79	0.67
mixed_filterenc	0.78	0.81	0.79
mixed_denoise	0.81	0.79	0.8

TABLE 3: Malicious window detection performances.

Method	Recall	Precision	F1-score
curated_flow	1	0.13	0.22
curated_windows	0.79	1.0	0.88
curated_autoenc	1.0	0.88	0.93
mixed_windows	0.21	0.97	0.34
mixed_filterenc	1.0	0.78	0.88
mixed_denoise	0.7	0.87	0.71

TABLE 4: Malicious host detection performances.

For each of the 3 encoders, we train a 3 layer feed-forward deep network. The dimensions of the layers are 128, 64, 64 respectively. We train the network on a batch size of 100 for 50 epochs. We use the ReLU activation function and optimize the MSE loss. The learning rate and weight decay for Adam are both 1e-3. The hyperparameter *max-flows* is tuned to 1,000. Since there are 30 feature dimensions per flow, the encoders' input dimensions are 30,000. We select the 64 dimension layer as the feature encoding that is input to the SVM model. The duration of the time window is 1 second. The training data has 54,940 time windows labeled malicious and 56,060 labeled normal. The testing dataset has 23,146 time windows labeled malicious and 28,541 labeled normal.

We investigate the following encoders and baselines, using the prefix *curated* or *mixed* to indicate which dataset is used:

- **curated_flow** uses raw, directly sourced flow features (see Table 2) across hosts without time windows. This is trained on a $d \times 30$ dataset, where d is the total number of flows across hosts (see Table 1).
- **curated_windows** uses time-window features without encoding.
- **curated_autoenc** uses autoencoded representations of time-window features.
- **mixed_windows** uses *mixed* time-window features without encoding.
- **mixed_filterenc** uses a filter-encoded representation of time-window features.
- **mixed_denoise** uses a denoising encoded representation of time-window features

We report Precision, Recall and F1-score to measure how well detection performs. We repeat encoder training 10 times and report averages. We compute standard deviation and use a Wilcoxon RankSum test to pairwise compare the methods.

6. Results And Discussion

Tables 3 and 4 show the results for windows and hosts detection respectively. For Q1 we analyze the performance of **curated_flow** compared to **curated_windows**. We see that the detection accuracy of **curated_flow** is poor – all inputs are classified as the majority class. This suggests using just the flow level features is inadequate. In contrast the **curated_windows** method's F1-score on window detection is 0.85 and on host detection is 0.88. This is an encouraging result and suggests the advantage of using temporally organized and expressive flow information as features.

For Q2 we next analyze the relative performance of method **curated_autoenc** that uses autoencoded feature representations in addition to organizing data by time windows. This comparison only applies to **curated_flow** and **curated_windows**. We observe that, for both these comparisons, window detection is better for precision and F1-score. Host detection, in recall and F1-score is better than **curated_windows** but worse in precision. It is equivalent (recall) or better (precision and F1-score) than **curated_flow**.

To answer Q3 we consult results from *mixed* dataset methods. We notice that **mixed_windows** performs very poorly. Clearly noisier flows, even organized into time windows, drastically degrade detection performance. This seems intuitive since background flows are randomly placed in windows which potentially distorts any intra or inter-flow relationships previously present. We note at this point though that, given the observed statistics on flows, the current dataset calls for further investigation. The median number of flows per source was as low of 4 for the *curated* dataset and 32 for *background* flows which seems to be counter to the intuitive number of flows a host has on a campus network. It remains to be seen whether a higher number of flows per source would have an adverse effect on our framework.

We next compare **mixed_filterenc** and **mixed_denoise** using a Wilcoxon RankSum test. They are statistically comparable and both statistically are better than **mixed_windows** for recall and F1 metrics and both window and host detection. These are very encouraging results and suggest that complex feature engineering can be achieved through such automated feature representation techniques. Likewise, we notice that **mixed_filterenc** is able to fully filter the added noise to detect almost as well **curated_filterenc**. Although it does well on recall, it under performs in precision. This is expected, since the filtering step is lossy.

7. Conclusion And Future Work

We present in this work an exploration of automated feature engineering through deep learning encoders. It applies to flow logs in support of botnet window and host detection. The lower dimensional feature representations of the encoders can be used with challenging volumes of host background, i.e. noisy, traffic.

There are numerous directions for future work. One is to assess the value of the encoding across multiple datasets with state of the art selections for detection methods. Another is to study the interpretability of such transformed feature spaces. It will be useful to learn how the raw features non-linearly combine and embed to support detection of the final outcome. Another is to investigate variation of the detection task we set up. We currently detect a variety of bots in a single learning task. We do not investigate the performance of our detectors on individual bots in the training and test set.

Acknowledgments

The authors thank Sebastian Garcia for answering queries related to the dataset. This work was partly supported by the CyberSecurity@CSAIL initiative at CSAIL, MIT.

References

[1] Ctu-13 dataset. Available on https://mcfp.weebly.com/. Accessed: 2017-09-30.

[2] J. R. Binkley and S. Singh. An algorithm for anomaly-based botnet detection. *SRUTI*, 6:7–7, 2006.

[3] Z. B. Celik, J. Raghuram, G. Kesidis, and D. J. Miller. Salting public traces with attack traffic to test flow classifiers. In *CSET*, 2011.

[4] S. Garcia, M. Grill, J. Stiborek, and A. Zunino. An empirical comparison of botnet detection methods. *computers & security*, 45:100–123, 2014.

[5] G. Gu, R. Perdisci, J. Zhang, W. Lee, et al. Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection. In *USENIX security symposium*, volume 5, pages 139–154, 2008.

[6] F. Haddadi, D. Le Cong, L. Porter, and A. N. Zincir-Heywood. On the effectiveness of different botnet detection approaches. In *ISPEC*, pages 121–135, 2015.

[7] E. Kartaltepe, J. Morales, S. Xu, and R. Sandhu. Social network-based botnet command-and-control: emerging threats and countermeasures. In *Applied Cryptography and Network Security*, pages 511–528. Springer, 2010.

[8] G. Pellegrino, Q. Lin, C. Hammerschmidt, and S. Verwer. Learning behavioral fingerprints from netflows using timed automata.

[9] R. A. Rodríguez-Gómez, G. Maciá-Fernández, and P. García-Teodoro. Survey and taxonomy of botnet research through life-cycle. *ACM Computing Surveys (CSUR)*, 45(4):45, 2013.

[10] S. S. Silva, R. M. Silva, R. C. Pinto, and R. M. Salles. Botnets: A survey. *Computer Networks*, 57(2):378 – 403, 2013. Botnet Activity: Analysis, Detection and Shutdown.

[11] S. Srinivas, R. K. Sarvadevabhatla, K. R. Mopuri, N. Prabhu, S. S. Kruthiventi, and R. V. Babu. A taxonomy of deep convolutional neural nets for computer vision. *arXiv preprint arXiv:1601.06615*, 2016.

[12] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[13] P. Wurzinger, L. Bilge, T. Holz, J. Goebel, C. Kruegel, and E. Kirda. Automatically generating models for botnet detection. In *European symposium on research in computer security*, pages 232–249. Springer, 2009.

[14] T. Young, D. Hazarika, S. Poria, and E. Cambria. Recent trends in deep learning based natural language processing. *arXiv preprint arXiv:1708.02709*, 2017.

[15] D. Zhao, I. Traore, B. Sayed, W. Lu, S. Saad, A. Ghorbani, and D. Garant. Botnet detection based on traffic behavior analysis and flow intervals. *Computers & Security*, 39:2–16, 2013.